Estratégia para geração de sequências de verificação para máquinas de estados finitos

Faimison Rodrigues Porto

Estratégia para geração de sequências de verificação para máquinas de estados finitos

Faimison Rodrigues Porto

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

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Junho de 2013

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Porto, Faimison Rodrigues
P839e Estratégia para geração de sequências de verificação para máquinas de estados finitos / Faimison Rodrigues Porto; orientador Adenilso da Silva Simão. -- São Carlos, 2013.
70 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2013.

1. Sequência de verificação. 2. Teste baseado em MEFs. 3. Geração de casos de teste. 4. Conjunto de distinção. I. Simão, Adenilso da Silva, orient. II. Título.

Aos meus pais Francisco e Zuleide.

Agradecimentos

Agradeço primeiramente a Deus por me conceder a força e a determinação necessária para a superação de cada novo desafio. Agradeço, de forma honrosa, aos meus pais Francisco e Maria Zuleide pelo imenso incentivo e apoio aos meus estudos. Sem o carinho e o apoio de vocês dificilmente teria conquistado mais esta realização em minha vida.

Agradeço ao meu irmão Franklin pelo grande aprendizado de vida que me proporcionou e à minha irmã Leidiane pelo apoio e constante incentivo. À minha sobrinha afilhada Alessandra por trazer alegria por onde passa, e à toda a minha família. É de grande importância o apoio de vocês nesta conquista.

Ao meu orientador Adenilso fica o agradecimento e a admiração pela pessoa atenciosa, carismática e competente que se mostrou durante esses anos. Agradeço os ensinamentos, os conselhos e, em especial, à confiança depositada em meu trabalho no início de tudo.

Agradeço aos meus companheiros de república Rayner, Fernando e Edvard pelas trocas de conhecimento e pelo companheirismo nos últimos anos. Aos meus irmãos Ursão, Pezão, Negão, Bené e Erik, por fazerem parte da nossa família Coqueiro. Aos grandes parceiros Adam e Lucas por estarem sempre presentes.

Aos amigos do LabES fica o agradecimento pelas inúmeras experiências adquiridas e as alegrias que me proporcionaram nestes quase dois anos de convivência. Em especial, agradeço aos *brothers* Frota, Cabeça, Endo, Maurício e Rafa pela amizade. Do Biocom, restaram boas amizades de minha breve passagem por esse laboratório.

Agradeço à todos e à cada um dos meus amigos não citados, mas que são essenciais em minha vida.

Aos professores e funcionários do ICMC, obrigado pela atenção, disposição e prestatividade.

À CAPES, pelo apoio financeiro.

Resumo

O teste de software engloba diferentes técnicas, métodos e conceitos capazes de garantir a qualidade dos mais variados tipos de sistemas. Dentre tais técnicas, encontra-se o teste baseado em Máquinas de Estados Finitos (MEFs), que visa a garantir a conformidade entre a implementação e a especificação de um software. Com esse propósito, diversos métodos foram propostos para a geração de sequências de verificação que garantam cobertura total das possíveis falhas existentes em uma implementação. A maioria dos métodos conhecidos são baseados na utilização de sequências de distinção. Esse recurso, porém, não existe para toda MEF. Alguns métodos buscam a geração de sequências de verificação baseados em recursos alternativos às sequências de distinção, contudo, as sequências geradas são exponencialmente longas. Este trabalho apresenta um método para geração de sequências de verificação que visa a reduzir o tamanho das sequências geradas para o domínio de MEFs que não dispõem de sequência de distinção. Para isso, o método proposto baseia-se na utilização de conjuntos de distinção. Uma avaliação experimental foi realizada a fim de mensurar a redução proporcionada pelo método proposto em relação aos principais métodos existentes na literatura. Com esse intuito, foram geradas MEFs aleatórias sob a perspectiva de diferentes fatores. Em relação à variação do número de estados, os resultados indicaram reduções acima de $99,5\%$ em comparação com os métodos existentes, quando analisadas 75% das MEFs geradas.

Abstract

Software testing involves several techniques, methods, and concepts employed to guarantee a high level of quality in different application domains. Among such techniques, Finite State Machine (FSM) based testing aims to guarantee the conformance between the implementation and the specification of a system under test. In this context, several methods were proposed to generate checking sequences that cover all the possible faults existing in an implementation. Most of these methods are based on a special sequence, named distinguishing sequence, which does not exist for every minimal machine. Some methods were proposed to generate checking sequences based on alternative solutions in order to be applied on FSMs that do not have distinguishing sequences. However, these methods generate checking sequences exponentially long. This work proposes a method to generate checking sequences using identification sets. These sets exist for every minimal FSM and also lead to shorter checking sequences. We conducted an experimental study to compare the proposed method with the main existing methods. In the experiments, we used random FSMs that have different configurations of states, inputs, and outputs. Concerning the variation of number of states, the results show reductions higher than 99.5% in comparison with the existing methods for 75% of the experimented machines.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo

1

Introdução

1.1 Contextualização

No processo de desenvolvimento de software são empregadas diversas metodologias, ferramentas e técnicas na tentativa de se construir softwares de qualidade e livres de defeitos. No entanto, mesmo com a aplicação desses recursos, os sistemas desenvolvidos ainda estão suscetíveis a falhas. Dessa forma, é necessário que, paralelamente ao processo de desenvolvimento, sejam realizadas atividades para garantir a qualidade do sistema desenvolvido, dentre as quais se destacam as atividades de teste, verificação e validação.

O teste de software consiste na atividade de execução da implementação de um sistema no intuito de encontrar defeitos (Myers, 2004). O teste é um elemento crítico na garantia da qualidade de um sistema e representa a revisão final da sua especificação, projeto e codificação (Pressman, 2009). É desejável que os defeitos de um software sejam identificados e corrigidos o mais cedo possível, reduzindo os impactos e o custo do processo de desenvolvimento. As atividades de teste, por sua vez, contribuem para a detecção de defeitos causados por diversos motivos, como, por exemplo, defeitos no código fonte do sistema ou a implementação incorreta dos requisitos.

O *teste de conformidade* tem como intuito analisar a concordância entre a implementação do sistema e sua especificação (Bochmann e Petrenko, 1994). Em outras palavras, esse tipo de teste busca garantir que a implementação de um determinado software tenha comportamento condizente com o que foi proposto na especificação de suas funções.

O teste de conformidade pode ser auxiliado pela criação de modelos formais capazes de representar e descrever o comportamento de um determinado sistema. Nesse contexto, o teste baseado em modelos utiliza informações sobre o comportamento do modelo para derivar os casos de teste (ou sequências de teste) (Tretmans, 1996). 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. A representação por meio de modelos formais proporciona um maior conhecimento sobre o software a ser testado e também possibilita a geração do conjunto de teste de maneira automática, o que pode acarretar na redução do custo de geração de casos de teste (Simao, 2007).

Algumas técnicas utilizadas na criação de modelos são baseadas em *Máquinas de Transição de Estados*, tais como Máquinas de Estados Finitos (MEFs) (Gill, 1962), *Statecharts* (Harel, 1987) e Redes de Petri (Peterson, 1977). A representação por meio de MEFs vem sendo frequentemente utilizada devido à sua simplicidade e expressividade. Outro benefício relacionado ao uso de MEFs refere-se às diferentes técnicas de geração de casos de teste já desenvolvidas, o que fornece um melhor apoio e direcionamento nos testes gerados e executados.

Devido à grande dificuldade encontrada na geração de conjuntos de casos de teste, diversos pesquisadores da área de computação tem focado na pesquisa de métodos mais eficientes. O principal objetivo almejado está na geração de conjuntos de teste capazes de identificar o maior número possível de falhas que a implementação possa conter (Fujiwara et al., 1991). Além disso, busca-se também a redução do custo de aplicação do teste.

O custo de aplicação desse tipo de método está relacionado tanto ao custo despendido na construção das sequências de teste, quanto ao custo de sua execução (Fujiwara et al., 1991). No entanto, em relação à eficiência de um método, o custo de execução das sequências de teste é mais crítico. Esse tipo de custo é fortemente afetado por dois fatores: o tamanho das sequências de teste e o número de sequências que compõem o conjunto de teste. Assim, é desejável que tanto a quantidade de sequências quanto o tamanho das sequências seja o menor possível, agilizando sua execução (Fujiwara et al., 1991). Em geral, alguns métodos assumem a existência de uma operação *reset*, que inicializa a MEF e a implementação, conduzindo-as ao seu estado inicial. Essa operação *reset* é inserida no início de cada sequência, o que, consequentemente, também determina o número de sequências existentes em um determinado conjunto de teste. Contudo, essa operação pode não existir ou ter custo elevado em determinados tipos de sistemas, tornando o teste impraticável.

Alguns métodos propostos na literatura buscam a geração de conjuntos de teste que independem da existência de uma operação *reset*. Nesses casos, o conjunto de teste gerado é composto por uma única sequência de símbolos de entrada, denominada de sequência de

verificação. Além disso, uma sequência de verificação deve ser capaz de garantir a identificação de qualquer falha existente na implementação, considerando algumas suposições. Dentre os principais métodos para geração de sequências de verificação, destacam-se os métodos de Gonenc (1970), Ural et al. (1997), Hierons e Ural (2002), Hierons e Ural (2006), Chen et al. (2005) e Simao e Petrenko (2008).

Um fator crucial para o funcionamento de um método de geração de sequências de verificação está em garantir que, após a aplicação de alguma sequência de entrada, a implementação permaneça em um estado conhecido (Simao e Petrenko, 2008). Existem duas metodologias comumente utilizadas com esse intuito: a *sequência de distinção* (*distinguishing sequence* ou DS) e o *conjunto de caracterização* (ou conjunto W) (Rezaki e Ural, 1995). Uma sequência de distinção refere-se a uma sequência única de entrada que permite que, para cada estado diferente da MEF, seja produzida uma saída diferente. Apesar de fornecer uma maneira eficiente na identificação dos estados de uma MEF, esse recurso não existe para toda MEF (Lee e Yannakakis, 1996). Um conjunto de caracterização é um conjunto de sequências de entrada que permitem identificar todos os estados de uma MEF, quando aplicadas em conjunto. Ao contrário da sequência de distinção, esse conjunto existe para toda MEF que não possui estados equivalentes (Gill, 1962).

1.2 Motivação

Alguns métodos tradicionais para geração de casos de teste, baseados em MEFs, geram conjuntos de teste compostos por várias sequências que devem ser aplicadas ao estado inicial do sistema, tais como os métodos W (Chow, 1978), UIO (Sabnani e Dahbura, 1988), Wp (Fujiwara et al., 1991), HSI (Petrenko et al., 1993), H (Dorofeeva et al., 2005) e SPY (Simao et al., 2009). O custo de aplicação do teste, no entanto, é fortemente influenciado pelo número de sequências e pelo tamanho das sequências. Consequentemente, para maior efetividade do método, é desejável que o conjunto de teste gerado seja composto pelo mínimo possível de operações *reset*, bem como pelo menor tamanho possível das sequências. Além disso, é possível que esse tipo de operação não exista ou tenha um custo elevado para utilização.

Considerando esses fatores, alguns métodos buscam maior efetividade por meio da geração de sequências de verificação, as quais são constituídas por uma única sequência de teste desprovida de operações *reset*. Contudo, a geração desse tipo de sequência é geralmente realizada com a utilização de sequências de distinção (Chen et al., 2005; Gonenc, 1970; Hierons e Ural, 2002, 2006; Ural et al., 1997). A utilização de sequências de distinção limita o domínio de aplicação do método, uma vez que nem toda MEF dispõe desse tipo de sequência.

Alguns métodos buscam a utilização de recursos alternativos à sequência de distinção. Os métodos propostos por Hennie (1964) e Rezaki e Ural (1995) geram sequências de verificação a partir do conjunto W. Uma vez que toda MEF minimal dispõe de um conjunto W, ao utilizar esse tipo de recurso o método torna-se mais abrangente. No entanto, os métodos de Hennie (1964) e Rezaki e Ural (1995) geram sequências de verificação exponencialmente longas em relação ao conjunto W. Esse fator compromete a efetividade do método pelo alto custo computacional exigido em sua execução. Muitas vezes, a execução do teste torna-se impraticável. Consequentemente, a investigação de estratégias capazes de reduzir o tamanho das sequências geradas nesse contexto torna-se um desafio. A geração eficiente de sequências de verificação nesse contexto não é uma atividade trivial e, por isso, é o foco de pesquisa neste trabalho.

1.3 Objetivos

Este projeto tem como objetivo a investigação de possíveis soluções que possam reduzir o tamanho de sequências de verificação geradas no contexto de MEFs que não dispõem de sequências de distinção. Novas soluções nesse contexto também visam a contemplar um maior número de MEFs, uma vez que nem toda MEF dispõe de sequências de distinção.

Este trabalho propõe um método para geração de sequências de verificação baseado em conjuntos de identificação, os quais são derivados a partir do conjunto W. Conjuntos de identificação, assim como conjuntos W, existem para toda MEF. Esse fator acarreta em um domínio de aplicação tão abrangente quanto os métodos de Hennie (1964) e Rezaki e Ural (1995). Além disso, a utilização de conjuntos de identificação proporciona maior redução no tamanho das sequências de verificação geradas. Estudos empíricos foram conduzidos a fim de mensurar a redução proporcionada pela estratégia proposta em comparação com os métodos existentes.

Os métodos comparados foram analisados em relação à variação de três fatores distintos: número de estados, número de entradas e número de saídas. Os resultados indicaram reduções de até $99,5\%$ do método proposto em relação aos métodos comparados, para 75% das MEFs geradas. Conclui-se também, a partir dos dados apresentados, que as reduções proporcionadas pelo método proposto são acentuadas para MEFs com mais estados, menos entradas e menos saídas.

1.4 Organização

Além desta seção introdutória, o restante desta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta os principais conceitos inseridos no contexto de teste de

software, com enfoque na técnica de teste baseado em modelos. O Capítulo 3 apresenta as definições e conceitos inseridos no contexto de teste baseado em MEFs, além dos principais métodos conhecidos na literatura para geração de sequências de verificação. O Capítulo 4 apresenta o método para geração de sequências de verificação proposto neste trabalho. Nesse capítulo são explanadas as características e o funcionamento do método proposto. Além disso, ainda no Capítulo 4, realiza-se uma avaliação experimental da redução proporcionada pelo método proposto em relação aos métodos existentes e consolidados na literatura. Por fim, o Capítulo 5 apresenta as conclusões deste trabalho.

Capítulo

2

Teste de Software

2.1 Considerações Iniciais

Durante o processo de desenvolvimento de softwares é possível a utilização de diversas metodologias, técnicas e ferramentas capazes de facilitar, aprimorar e automatizar essa tarefa. Contudo, mesmo com a utilização de todos esses recursos, o desenvolvimento de sistemas ainda está suscetível a enganos, o que, possivelmente, acarreta em falhas no produto gerado. Dessa forma, nesse processo, é fundamental a aplicação de conceitos e técnicas que garantam um nível aceitável de qualidade ao software gerado.

As atividades de teste estão inseridas nesse contexto e visam a identificação de erros ainda desconhecidos em um software. O teste de software é responsável por verificar o funcionamento correto de um sistema no ambiente para o qual foi projetado, além de verificar se o sistema está de acordo com suas especificações. A qualidade obtida no teste de um software, geralmente, está associada à geração de casos de teste eficientes na cobertura e detecção de falhas. No teste baseado em modelos, o processo de geração de casos de teste é baseada nas informações inerentes ao modelo formal do sistema.

Um modelo formal representa e descreve o comportamento do sistema em teste. Esse modelo deve ser obtido a partir de sua especificação, por meio de alguma técnica de modelagem formal. Além de permitir a automatização de testes, a modelagem formal de um sistema permite que o conhecimento sobre esse sistema seja capturado de forma não ambígua e reutilizado durante diversas fases do desenvolvimento.

Neste capítulo são apresentados conceitos gerais no âmbito de teste de software, bem como conceitos estritamente relacionados ao teste baseado em modelos. Os conceitos introduzidos neste capítulo perfazem o embasamento e a contextualização geral deste trabalho.

2.2 Fundamentos do Teste de Software

Dentre as principais atividades que objetivam aprimorar a qualidade do processo de desenvolvimento de softwares, estão as atividades de Verificação, Validação e Teste (VV&T) (Andriole, 1986; Pressman, 2009). As atividades de *verificação* visam a assegurar que o processo de desenvolvimento está sendo realizado de maneira correta. As atividades de *validação*, por sua vez, possibilitam avaliar se o software produzido está em conformidade com os requisitos especificados. Por fim, as atividades de *teste* buscam encontrar erros na implementação de um sistema por meio de sua execução.

As atividades inseridas nesse contexto podem ser classificadas de duas maneiras: estáticas ou dinâmicas (Delamaro et al., 2007). A primeira categoria refere-se às atividades que não necessitam de artefatos executáveis para serem aplicadas, tais como inspeção de software e revisões técnicas. As atividades dinâmicas, por sua vez, se baseiam na verificação dos resultados obtidos pela execução de artefatos. Nesse contexto, as atividades de teste são classificadas como dinâmicas, visto que dependem da execução do sistema implementado.

A aplicação desses tipos de atividades durante o processo de desenvolvimento possibilitam a identificação precoce de erros inseridos no software, evitando, assim, possíveis riscos associados. As atividades de teste, em específico, constituem um elemento crítico na garantia da qualidade de um sistema, visto que essas atividades representam a revisão final da especificação, projeto e codificação do sistema (Myers, 2004). De acordo com Pressman (2009), não é incomum que as atividades de teste despendam grande parte de todo o esforço empregado no projeto e desenvolvimento de um software.

O objetivo da aplicação de atividades de teste é a detecção de erros ainda desconhecidos em um sistema, embora, geralmente, não seja possível garantir a ausência completa de erros (Myers, 2004). Dessa forma, com o intuito de garantir um nível aceitável de confiabilidade e qualidade em um software, essas atividades devem ser planejadas e aplicadas com o rigor e controle necessários, a depender de quão crítico é o software a ser desenvolvido (Harrold, 2000).

Na área de teste de software existem algumas terminologias importantes utilizadas em seu embasamento teórico. Dessa forma, a fim de padronizar tais termos, são estabelecidas as seguintes definições (Delamaro et al., 2007; IEEE, 1990):

- **Sistema em teste** ou **SUT** (*System Under Test*): é o sistema, subsistema ou componente de software a ser testado;

- **Domínio de entrada**: é o conjunto de todos os possíveis valores permitidos como entrada na execução do SUT;

- **Domínio de saída**: é o conjunto de todos os possíveis resultados produzidos pela execução do SUT;

- **Dado de teste**: corresponde a um elemento pertencente ao domínio de entrada do programa;

- **Caso de teste** (*test case*): é um par entrada/saída esperada do SUT. Corresponde a um dado de teste, como entrada, e o resultado esperado de sua execução, como saída;

- **Conjunto de teste** (*test case suite* ou *test suite*): também chamado de conjunto de casos de teste, corresponde ao conjunto de todos os casos de teste utilizados durante uma determinada atividade de teste;

- **Oráculo**: mecanismo de decisão utilizado para reconhecer se a saída produzida pelo SUT corresponde à saída esperada. Esse mecanismo pode ser uma tabela de valores, um algoritmo ou simplesmente o conhecimento do testador;

- **Defeito** (*fault*): refere-se a um passo, processo ou definição de dados incorretos (instrução ou comando incorreto);

- **Engano** (*mistake*): ação humana que resulta em um defeito;

- **Erro** (*error*): é a consequência da execução de um defeito, o qual acarreta em um estado inconsistente ou inesperado no sistema;

- **Falha** (*failure*): corresponde à propagação de um erro até a saída do sistema, ou seja, é uma saída incorreta em relação à especificação.

Ao executar os casos de teste no SUT, é possível identificar falhas no sistema por meio da análise dos resultados obtidos com sua execução, considerando o oráculo utilizado. Dessa forma, a atividade de teste tem como objetivo revelar as falhas / defeitos existentes em um sistema, enquanto a atividade de depuração é responsável pela localização de tais defeitos.

2.3 Fases de Teste

A execução do processo de teste de software deve ser aplicada de maneira controlada e rigorosa, conforme as necessidades críticas inerentes ao sistema em teste. Segundo Pressman (2009), uma atividade de teste deve seguir uma estratégia que possibilite um acompanhamento gerencial à medida que o projeto avança, além de proporcionar um roteiro que descreva os passos a serem seguidos de maneira controlada. Para isso, a execução de uma atividade de teste é dividida em quatro etapas (Beizer, 1990; Pressman, 2009):

- **Planejamento**: nessa etapa é elaborado um plano de teste essencial para condução e gerenciamento do processo de teste. Nesse plano, segundo Myers (2004), deve conter a definição de todos os recursos necessários à condução do processo de teste, como: cronograma, objetivos, critérios de finalização, recursos humanos, ferramentas a serem utilizadas, procedimentos aplicados na depuração, dentre outros elementos que são arbitrados pelas necessidades do projeto;

- **Projeto dos casos de teste**: nessa etapa são elaborados os casos de teste. Os casos de teste podem ser construídos a partir de técnicas distintas, conforme explanado na Seção 2.4. Segundo Myers (2004), a construção dos casos de teste é uma etapa importante para a efetividade e qualidade do teste, uma vez que não é possível garantir a ausência total de erros e a cobertura de erros aplicada ao SUT é determinada pelos casos de teste;

- **Execução do teste**: nessa etapa são executados os procedimentos previstos e planejados na primeira etapa; e

- **Coleta e avaliação dos resultados**: nessa etapa, após a execução do teste, os resultados produzidos são coletados, registrados, organizados e apresentados na forma de relatórios. Tais resultados são avaliados em relação aos resultados esperados, definidos conforme o oráculo empregado.

As etapas descritas acima referem-se ao processo estratégico que compõe uma atividade de teste. Entretanto, atividades de teste podem ser aplicadas de acordo com o ciclo de vida do software. Segundo Mathur (2008), durante o desenvolvimento de um software, existe uma fase de teste compatível às necessidades e aos recursos disponíveis para cada artefato produzido. Em sistemas complexos, por exemplo, é desejável que o teste de software seja dividido em fases que permitam testar, além do sistema por completo, as pequenas partes que compõem o sistema. Dessa forma, o teste de software pode ser classificado em cinco fases complementares que visam testar desde as pequenas unidades do sistema até

as alterações provenientes de manutenção do software. Tais fases são descritas a seguir (Pressman, 2009):

- **Teste de unidade**: nessa fase são testadas todas as unidades do sistema individualmente, visando a identificação de erros de especificação e/ou implementação. Nesse contexto, uma unidade é um componente de software que não pode ser subdivido (IEEE, 1990). Assim, esse tipo de teste se concentra na verificação de falhas nos parâmetros de entrada/saída, nas estruturas de dados que influenciam na integridade dos dados armazenados e nas condições que determinam os limites de operação da unidade;

- **Teste de integração**: nessa fase as unidades já testadas individualmente são integradas, formando subsistemas. Esses subsistemas propiciam a identificação de diversos tipos de falhas, tais como: interface incorreta, falta ou conflito de funcionalidades, violação de integridade de arquivos e estruturas de dados globais, sequência incorreta de unidades, tratamento de exceções incorreto, problema de configuração e falta de recursos para atender a demanda das unidades;

- **Teste de sistema**: nessa fase o sistema é testado como um todo, levando em consideração os requisitos funcionais e não funcionais especificados. Além disso, nesse tipo de teste é avaliado o funcionamento do sistema em relação ao ambiente para o qual foi projetado, incluindo a plataforma de execução e outros sistemas externos que estão direta ou indiretamente relacionados ao sistema em teste;

- **Teste de aceitação**: nessa fase é verificada a aceitação do produto gerado perante os usuários finais. O teste, normalmente, é realizado por um grupo de usuários finais que verificam se o sistema está de acordo com as funcionalidades estabelecidas para o projeto; e

- **Teste de regressão**: nessa fase, os testes realizados anteriormente são reaplicados a fim de garantir o funcionamento correto do software a cada nova versão. Uma vez que alterações realizadas em determinadas partes de um software podem comprometer o funcionamento do software como um todo.

As fases de teste descritas acima permitem a cobertura de diferentes tipos de erros, visto que cada fase abrange aspectos distintos do processo de desenvolvimento de software. Um dos fatores determinantes para a detecção de erros existentes em uma implementação está na cobertura de erros alcançada por um conjunto de casos de teste (Myers, 2004). Dessa forma, um conjunto de casos de teste ideal seria aquele composto por todos os elementos do domínio de entrada do programa em teste. Contudo, isso geralmente é inviável, uma vez

que o domínio de entrada pode ser extremamente grande, ou até mesmo infinito (Rapps e Weyuker, 1985). Sendo assim, é necessária a seleção de um subconjunto de casos de teste finito capaz de garantir a qualidade desejada. Para isso, são definidos os *critérios de teste*.

Um critério de teste estabelece requisitos e propriedades que devem ser cumpridos por um conjunto de teste. Tais requisitos garantem um determinado nível de confiança ao conjunto de teste na identificação de erros e são utilizados tanto para avaliação quanto para a construção de conjuntos de teste (Frankl e Weyuker, 2000). A construção do conjunto de teste, contudo, pode ser realizada por diferentes técnicas. Cada técnica pode adotar diferentes modelos de informação para derivar os requisitos de teste e, por isso, são agrupadas em quatro diferentes categorias: funcional, estrutural, baseada em defeitos e baseada em modelos.

Embora existam técnicas que não se enquadram de maneira adequada a nenhuma das quatro categorias citadas, como o teste exaustivo e o teste aleatório, essas categorias abrangem as principais técnicas conhecidas na literatura (Simao, 2004). Na próxima seção, essas técnicas são brevemente descritas.

2.4 Técnicas de Teste

Para cada fase de teste apresentada na seção anterior é necessária a construção de um conjunto de casos de teste que garanta um nível de qualidade adequado na identificação de erros no sistema. Contudo, para que a aplicação do conjunto de teste seja viável, é necessário que o conjunto de teste seja composto por um conjunto finito de casos de teste. Para isso, são definidos critérios de teste compostos por requisitos que, caso sejam satisfeitos, garantem um nível conhecido de confiança na avaliação ou construção de um conjunto de teste, uma vez que não é possível garantir a ausência completa de erros em um software (Myers, 2004).

Os critérios de teste podem ser definidos por meio de técnicas distintas, as quais são agrupadas em quatro categorias: funcional, estrutural, baseada em defeitos e baseada em modelos. A principal característica que distingue essas técnicas refere-se à forma de concepção da informação que é utilizada na avaliação e construção do conjunto de teste (Maldonado, 1991). Dessa forma, por abranger aspectos distintos de um sistema, essas técnicas se complementam e devem ser utilizadas em conjunto, uma vez que, provavelmente, revelam classes diferentes de defeitos (Pressman, 2009).

No **teste funcional** (ou caixa-preta), os casos de teste são derivados a partir dos dados de entrada e das saídas produzidas pelo software já implementado, desconsiderando aspectos de implementação e do comportamento interno do software. Dessa forma, o

único mecanismo de decisão que determina a corretude do programa em teste é a sua especificação, que define os requisitos e as funcionalidades esperadas. Segundo Pressman (2009), as principais classes de erros passíveis de serem detectados por esse tipo de teste são: funcionalidades incorretas ou omitidas, erros de interface, erros de estrutura de dados ou de acesso a dados externos, erros de comportamento ou desempenho e, por fim, erros de iniciação e término. Uma das dificuldades encontradas no teste funcional está no entendimento correto dos requisitos e resultados esperados pela execução do software, uma vez que, em alguns casos, a especificação é construída de maneira inconsistente e/ou incompleta. No teste funcional, os principais critérios de teste empregados são: particionamento em classes de equivalência, análise do valor limite e grafo de causa-efeito (Myers, 2004; Pressman, 2009).

No **teste estrutural** (ou caixa-branca), o processo de geração de casos de teste é baseado nas estruturas lógicas e funcionais da implementação do software (Myers, 2004). Ao contrário do teste funcional, é necessário o conhecimento do código fonte do programa em teste. Os critérios de teste estrutural, geralmente, utilizam uma representação em grafo do fluxo de controle lógico do programa para definição dos requisitos de teste (Pressman, 2009). Segundo Pressman (2009), nesse tipo de teste é possível identificar, principalmente, as seguintes classes de defeitos: defeitos lógicos, pressuposições incorretas e defeitos de projeto. Devido ao uso de informações de fluxo de dados para derivar os requisitos de teste, os critérios de teste estrutural são aplicados, principalmente, nas fases de teste de unidade e teste de integração (Barbosa et al., 2007).

No **teste baseado em defeitos**, a formulação dos requisitos de teste consiste na exploração de determinadas classes de defeitos típicos do processo de implementação de software (Demillo, 1980). Dessa forma, as informações utilizadas para derivação dos requisitos de teste são concebidas a partir de possíveis defeitos já conhecidos. Essas informações, no entanto, são influenciadas pelas características do software em teste, como a linguagem de programação, método ou técnica utilizada em seu desenvolvimento. Dentre os principais critérios inseridos nessa categoria estão os critérios Análise de Mutantes (*Mutation Analysis*) e Semeadura de Erros (*Error Seeding*).

O **teste baseado em modelos** está diretamente relacionado ao foco deste trabalho e, devido a isso, será abordado em detalhes a seguir.

2.4.1 Teste Baseado em Modelos

O Teste Baseado em Modelos (TBM) é uma técnica de teste que utiliza as informações sobre o comportamento funcional do sistema para realização do teste, desconsiderando a estrutura de implementação e o comportamento interno do software. Esse tipo de teste está inserido no teste funcional, contudo, o oráculo que define e separa o comportamento

adequado do comportamento errôneo é constituído pelo modelo formal da especificação do sistema. Embora um modelo não ambíguo também possa ser gerado por uma especificação informal, a modelagem formal permite uma análise sistemática e automatizada do sistema, além de possibilitar uma melhor compreensão do sistema (Simao, 2007).

O modelo formal de um sistema deve representar tanto as suas funcionalidades quanto as sequências de ações possíveis a partir de cada ponto específico em seu comportamento. No teste baseado em modelos, essas informações permitem a aplicação de métodos sistemáticos capazes de gerar conjuntos de casos de teste de maneira automática (Tretmans, 1996). Uma vez que a atividade de geração de casos de teste constitui uma tarefa complexa no teste de software, a aplicação desses métodos pode tornar a atividade de teste menos complexa. Outro fator benéfico, advindo da automatização dessa atividade, está na redução dos custos para reaplicação do processo de teste, visto que o software e o modelo, geralmente, são alterados com frequência. Dessa forma, os profissionais de teste podem reaplicar rapidamente o processo de geração de casos de teste a cada mudança realizada no modelo, evitando o trabalho manual nessa tarefa (Dalal et al., 1999).

O processo de modelagem de um sistema, geralmente, é realizado no início do ciclo de desenvolvimento do software, antes do início da fase de codificação. Esse fator possibilita a iniciação das atividades de teste antes mesmo da implementação do software, permitindo revelar erros na especificação de requisitos ou na fase de projeto do software (Sinha e Smidts, 2006). No trabalho de Boberg (2008), os resultados obtidos indicam que o teste baseado em modelos, devidamente aplicado a partir do início do processo de desenvolvimento de software, aumenta significativamente o número de falhas detectadas durante o teste.

O processo de teste baseado em modelos é constituído por um conjunto de atividades e diretrizes que, ao serem conduzidas com rigor, proporcionam um teste eficaz e preciso na detecção de falhas de um sistema. No trabalho de Utting e Legeard (2006), o processo para aplicação desse tipo de teste é definido em 5 etapas:

1. **Modelagem do comportamento do sistema**: Essa etapa tem como intuito obter um modelo formal suficientemente correto e condizente com a especificação do software a ser testado. O processo de modelagem deve ser realizado de maneira criteriosa, visto que todo o comportamento do sistema deve ser especificado. Uma boa prática a ser aplicada, após a construção do modelo, é a utilização de técnicas e ferramentas para verificação do modelo construído. Para isso, técnicas de *model checking* e simulação são comumente utilizadas (Clarke et al., 1999).

2. **Geração dos casos de teste abstratos**: Nessa etapa, geralmente, são utilizadas ferramentas que implementam técnicas para geração de casos de teste. Os casos de

teste são gerados em conformidade com o modelo criado na etapa anterior e com os critérios de teste estabelecidos. Como resultado, nessa etapa é gerado um conjunto de sequências de operações executáveis no modelo.

3. **Concretização dos casos de teste abstratos**: Os casos de teste abstratos gerados na etapa anterior são, então, concretizados de forma que possam ser executados no sistema. Dessa maneira, o resultado obtido nessa etapa é um conjunto de *scripts* de teste.

4. **Execução dos casos de teste**: Nessa etapa, os *scripts* gerados na etapa anterior são executados no sistema em teste. As saídas geradas são então confrontadas com as saídas esperadas, provenientes da execução dos casos de teste no modelo de especificação. Ao final dessa etapa, é gerado um relatório que descreve a execução e os resultados dos testes.

5. **Avaliação do resultado do teste**: Por fim, os resultados descritos no relatório de execução dos testes são então analisados. Mediante as falhas identificadas, são definidas ações a serem tomadas para correção do sistema ou do modelo inicial.

A geração de casos de teste está diretamente relacionada tanto ao modelo quanto à técnica de modelagem utilizada. Cada técnica de modelagem possui características particulares que podem se adequar melhor na modelagem de determinados tipos de sistemas. Algumas dessas peculiaridades estão, por exemplo, no grau de formalidade e nos mecanismos de modelagem oferecidos por cada técnica. Dentre as principais técnicas de modelagem, pode-se citar: Máquinas de Estados Finitos (MEFs) (Gill, 1962), Máquinas de Estados Finitos Estendidas (MEFEs) (Petrenko et al., 2004), Statecharts (Harel, 1987), Redes de Petri (Peterson, 1977) e SDL (Belina e Hogrefe, 1989).

A modelagem de softwares por meio de MEFs vem sendo frequentemente utilizada devido à sua simplicidade e expressividade. Um outro benefício advindo do uso de MEFs está relacionado às diferentes técnicas de geração de casos de teste já desenvolvidas, o que fornece um melhor suporte e direcionamento nos testes gerados e executados. Essa técnica é melhor descrita no próximo capítulo.

2.5 Considerações Finais

Neste capítulo, foram introduzidos conceitos que perfazem o embasamento do teste de software. A atividade de teste, no seu contexto geral, constitui uma das mais importantes atividades voltadas para a garantia de qualidade no desenvolvimento de um software. Devido aos diversos domínios específicos e as diferentes complexidades encontradas no

desenvolvimento de software, a atividade de teste dispõe de diversas técnicas distintas direcionadas ao mesmo objetivo: indicar a presença de defeitos.

Contudo, a geração de casos de teste eficientes, que garantam um nível adequado de qualidade e de maneira viável, ainda possui diversas limitações. O teste baseado em modelos, visa a auxiliar essa tarefa por meio da especificação formal do software em teste. O modelo formal de um sistema deve representar tanto as suas funcionalidades quanto as sequências de ações possíveis a partir de cada ponto específico em seu comportamento. A partir dessas informações, é possível a aplicação de métodos sistemáticos capazes de gerar conjuntos de casos de teste de maneira automática.

Dentre as principais técnicas de modelagem, pode-se destacar as Máquinas de Estados Finitos. Esse tipo de técnica vem sendo largamente utilizado na modelagem de sistemas devido, principalmente, à simplicidade dos modelos gerados. Os conceitos específicos ao teste baseado em MEFs são apresentados no próximo capítulo. Além disso, também são introduzidos alguns dos principais métodos para geração de casos de teste, inseridos no contexto deste trabalho.

Capítulo

3

Teste Baseado em Máquinas de Estados Finitos

3.1 Considerações Iniciais

O teste baseado em modelo tem como intuito a geração de casos de teste a partir de modelos obtidos com a aplicação de alguma técnica de modelagem. Dentre as principais técnicas de modelagem, encontram-se as Máquinas de Estados Finitos (MEFs) (Gill, 1962). Essa técnica pode ser aplicada em uma gama genérica de sistemas, principalmente na modelagem de sistemas que possuem como característica um comportamento reativo (Broy et al., 2005). Ou seja, sistemas cujo comportamento é arbitrado por estímulos de entrada.

Alguns exemplos, passíveis de modelagem por meio de MEFs, são aplicativos de áreas como Análise Léxica, Processamento de Dados, Controle de Processos em Tempo de Execução, Protocolos de Comunicação e Telecomunicações. A modelagem desses sistemas por meio de MEFs possibilita a automatização de atividades do processo de teste, o que pode acarretar na redução do custo de aplicação do teste (Simao, 2007).

Neste capítulo são apresentados os principais conceitos e definições sobre MEFs, bem como sobre o teste baseado em MEFs. Além disso, são introduzidos os métodos mais relevantes no contexto de geração de sequências de verificação. Dentre os quais encontram-se os métodos de Gonenc (1970), Ural et al. (1997), Simao e Petrenko (2008), Hennie (1964)

e Rezaki e Ural (1995). Os métodos de Hennie (1964) e Rezaki e Ural (1995), especificamente, perfazem o embasamento para a formulação do método proposto neste trabalho, apresentado no Capítulo 4.

3.2 Fundamentos do Teste Baseado em MEFs

Dada uma MEF M que representa a especificação de um sistema e uma MEF I que representa uma implementação construída a partir M, considera-se que I deva ser capaz de produzir o mesmo comportamento descrito por M. Sendo assim, o testador deve verificar a conformidade entre as MEFs M (especificação) e I (implementação). O teste baseado em MEFs tem como intuito gerar um conjunto de casos de teste eficiente a partir da MEF que representa o modelo do software a ser testado. Esse conjunto, ao ser aplicado a uma implementação, deve ser capaz de identificar qualquer divergência entre a implementação e a especificação, o que evidencia uma falha. Assim, caso o conjunto de casos de teste tenha cobertura total sobre as possíveis falhas do sistema, e nenhuma falha for encontrada na implementação, pode-se concluir que a implementação está em conformidade com a sua especificação.

O conjunto de casos de teste aplicado deve ser capaz de identificar qualquer falha, caso essa exista. Assume-se que I é composta por no máximo n estados, onde n é o número de estados de M. Um conjunto de teste é n-completo quando é capaz de distinguir M de todas as outras MEFs possíveis com no máximo n estados. Assim, ao aplicar um conjunto n-completo em ambas as MEFs I e M, as saídas produzidas devem ser idênticas. Caso contrário, a implementação não está em conformidade com a especificação.

Durante o processo de teste baseado em MEF é possível encontrar alguns tipos de falhas (Chow, 1978):

- **Falha de transferência**: quando uma transição atinge um estado incorreto;
- **Falha de saída**: quando a saída produzida é incorreta;
- **Estados ausentes**: quando estados devem ser inseridos na implementação para torná-la equivalente à especificação; e
- **Estados extras**: quando os estados da implementação devem ser reduzidos para torná-la equivalente à especificação.

Uma maneira possível para identificação das falhas descritas acima está na aplicação de todas as combinações do conjunto de símbolos de entrada. Ou seja, aplicar todas as sequências de entradas possíveis e verificar a conformidade entre as saídas geradas pela

implementação I e a especificação M. Entretanto, esse método torna-se inviável, visto que o número de sequências possíveis é, geralmente, intratável.

Dessa forma, diversos métodos têm sido propostos com o intuito de gerar sequências de teste, no domínio de MEFs, que garantam a cobertura total de falhas de uma maneira eficiente. Além disso, tais métodos são constituídos de procedimentos sistemáticos que possibilitam sua automatização. O método ideal deve gerar o menor conjunto de sequências de teste possível, de forma que seja suficiente para revelar todas as falhas presentes na implementação, caso existam.

3.3 Máquinas de Estados Finitos: Definição

Segundo Gill (1962), uma MEF corresponde a uma máquina hipotética composta por estados e transições. Em um determinado instante, a máquina deve estar em apenas um estado. A interação da máquina entre os estados ocorre por meio das transições. Cada transição define a ligação entre dois estados s_i e s_j, podendo, inclusive, ser uma ligação reflexiva onde s_i e s_j correspondem ao mesmo estado. Ao receber uma entrada, a máquina percorre uma transição e gera uma saída. O comportamento geral de uma MEF, tanto a saída gerada quanto o estado atingido, é definido em função do estado atual e do evento de entrada (Davis, 1988).

De acordo com Petrenko e Yevtushenko (2005), uma MEF M é definida como uma tupla (S, s_0, X, Y, D_M, δ, λ), onde:

- S é um conjunto finito de estados pertencentes a M;
- $s_0 \in S$ é o estado inicial;
- X é o alfabeto finito de símbolos de entrada;
- Y é o alfabeto finito de símbolos de saída;
- $D_M \subseteq S \times X$ é o domínio da especificação;
- δ é a função que determina o próximo estado de uma transição, $\delta : D_M \rightarrow S$;
- λ é a função que determina a saída, $\lambda : D_M \rightarrow Y$.

Dado um estado $s \in S$ e um símbolo de entrada $x \in X$, uma tupla $(s, x) \in D_M$ representa uma *transição definida* de M. Uma sequência $\alpha = x_1x_2...x_k$ é denominada uma *sequência de entrada definida* no estado $s \in S$ se existe $s_1, s_2, ..., s_{k+1}$, onde $s_1 = s$, tal que $(s_i, x_i) \in D_M$ e $\delta(s_i, x_i) = s_{i+1}$ para todo $1 \leq i \leq k$. O conjunto de todas as sequências de entrada definidas para o estado s é denotado por $\Omega(s)$.

A função de transição δ e a função de saída λ podem ser estendidas para sequências de entrada definidas compostas por um ou mais símbolos de entrada, inclusive o símbolo de sequência vazia ϵ. Dessa maneira, seja α uma sequência de entrada definida para $s \in S$ e $\delta(s, \alpha) = s'$, então, para todo $x \in X$ define-se $\delta(s, \alpha x) = \delta(s', x)$ e $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(s', x)$, sendo que $\delta(s, \epsilon) = s$ e $\lambda(s, \epsilon) = \epsilon$.

Dada uma MEF M, um estado $s \in S$, e um conjunto de sequências $C \subseteq \Omega(s)$, $\delta(s, C)$ é definido como o conjunto de estados alcançados pelas sequências em C, ou seja, $\delta(s, C) = \{\delta(s, \alpha) | \alpha \in C\}$.

Uma MEF pode ser representada de duas maneiras: por meio de uma tabela de transição ou graficamente por meio de um grafo dirigido. No primeiro método, por meio da tabela de transição, cada estado é representado por uma linha e cada símbolo de entrada é representado por uma coluna na tabela. Dessa forma, cada intersecção entre uma linha e uma coluna na tabela define o estado a ser atingido e a saída a ser gerada, de acordo com o estado de origem e a respectiva entrada (Davis, 1988). Nas Tabelas 3.1 e 3.2, são mostrados exemplos de MEFs com 4 e 3 estados, respectivamente, representadas por tabelas de transição. O estado "1" representa o estado inicial de cada MEF.

	λ		δ	
S \ X	a	b	a	b
1	0	0	4	2
2	1	0	2	4
3	0	1	2	4
4	1	1	3	1

Tabela 3.1: MEF de 4 estados representada por uma tabela de transição.

	λ		δ	
S \ X	a	b	a	b
1	0	1	2	3
2	1	0	3	1
3	0	0	2	1

Tabela 3.2: MEF de 3 estados representada por uma tabela de transição.

Alternativamente, uma MEF pode ser representada de maneira que os estados e as transições correspondem, respectivamente, aos vértices e as arestas de um grafo. A cada aresta são associadas uma entrada e uma saída produzida em resposta a essa entrada. Nas Figuras 3.1 e 3.2 são exibidas as representações dos grafos referentes às MEFs definidas nas Tabelas 3.1 e 3.2, respectivamente.

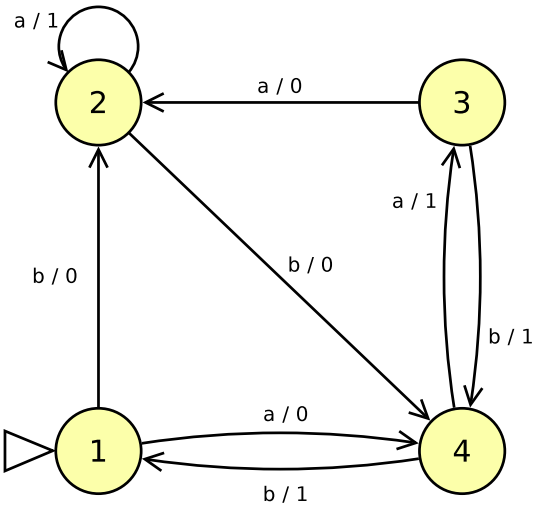

Figura 3.1: Representação gráfica da MEF definida na Tabela 3.1.

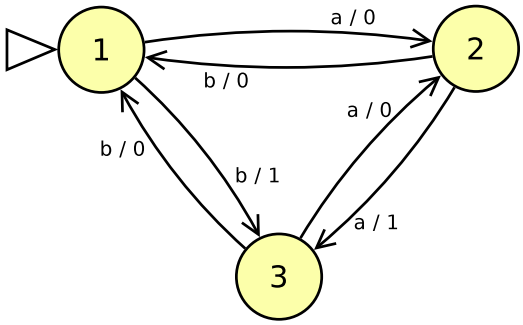

Figura 3.2: Representação gráfica da MEF definida na Tabela 3.2.

Os conceitos e métodos introduzidos no restante deste trabalho fazem uso das notações apresentadas nesta seção. Para a exemplificação de tais conceitos e métodos serão utilizadas as MEFs das Figuras 3.1 e 3.2.

3.4 Características e Propriedades de uma MEF

MEFs possuem algumas propriedades importantes para o contexto deste trabalho. Algumas das principais nomenclaturas utilizadas na classificação de MEFs são apresentadas a seguir (Gill, 1962; Lee e Yannakakis, 1996; Simao e Petrenko, 2010):

- Uma MEF é *determinística* se, para cada símbolo de entrada $x \in X$, existe no máximo uma transição definida em cada estado.

- Uma MEF é denominada como *Mealy* se, para toda transição definida, existe uma saída associada.

- Uma MEF é *completamente especificada* ou *completa* se, para todo estado $s \in S$, existe uma transição definida para cada símbolo $x \in X$. Formalmente, uma MEF é

completa se $D_M = S \times X$. Caso contrário, uma MEF não-completa é dita *parcial* ou *parcialmente especificada*.

- *Fortemente conexa* é a denominação dada a uma MEF tal que, para todo par de estados $(s_i, s_j) \in S$, existe uma sequência de símbolos de entrada que leve de s_i até s_j. Ou seja, existe $\alpha \in \Omega(s_i)$, tal que $s_j = \delta(s_i, \alpha)$. Analogamente, uma MEF é dita *inicialmente conexa* se, para todo estado $s \in S$, existe uma sequência de símbolos de entrada que leve do estado inicial s_0 até s.

- Dois estados s_i e s_j, são *equivalentes* se para as mesmas entradas produzem as mesmas saídas, ou seja, $\lambda(s_i, \alpha) = \lambda(s_j, \alpha)$ para toda entrada $\alpha \in \Omega(s_i) \cap \Omega(s_j)$. Analogamente, dois estados (s_i, s_j) são *distinguíveis* se existe uma entrada $\gamma \in \Omega(s_i) \cap \Omega(s_j)$ tal que $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$. Essa sequência γ é denominada como uma *sequência de separação* entre s_i e s_j, pois distingue s_i de s_j ao ser aplicada em ambos os estados. Dado um conjunto $C \subseteq \Omega(s_i) \cap \Omega(s_j)$, os estados s_i e s_j são *C-equivalentes* se, para toda sequência $\alpha \in C$, $\lambda(s_i, \alpha) = \lambda(s_j, \alpha)$.

- MEFs são *equivalentes* (ou *C-equivalentes*) se seus estados iniciais são equivalentes (ou C-equivalentes, respectivamente). Similarmente, MEFs são *distinguíveis* se seus estados iniciais são distinguíveis.

- Uma MEF é dita *minimal* (ou *reduzida*) se, para todo par de estados $(s_i, s_j) \in S$, $s_i \neq s_j$, s_i e s_j são distinguíveis. Em outras palavras, nenhum par de estados da máquina é equivalente.

- Dada uma MEF minimal M com n estados, $\Im(M)$ denota o conjunto de todas as MEFs determinísticas, completas e minimais com o mesmo alfabeto de símbolos de entrada e de saída e, no máximo, n estados.

- Um conjunto de teste T de uma MEF M é *n-completo*, se para cada MEF $N \in \Im(M)$, tal que N e M são distinguíveis, existe um caso de teste em T que distingue M de N.

É importante notar que a definição de MEF utilizada neste trabalho permite apenas MEFs que sejam determinísticas e de Mealy.

3.5 Sequências Básicas

Métodos de geração de casos de teste são geralmente baseados em um conjunto de sequências básicas de entradas. Essas sequências, normalmente, são utilizadas na geração de re-

sultados parciais importantes no processo de geração das sequências finais de teste (Simao, 2007). A seguir, são abordadas algumas dessas sequências básicas.

Sequência de sincronização SS é uma sequência de entrada capaz de levar uma MEF de qualquer estado $s \in S$ a um estado s' específico. Ou seja, dada uma sequência de sincronização α_{ss} para o estado $s' \in S$, para qualquer estado $s \in S$, $\delta(s, \alpha_{ss}) = s'$. Dessa forma, ao aplicar uma sequência de sincronização, é possível, por exemplo, garantir a presença atual da MEF no estado inicial. Contudo, é importante considerar que uma sequência de sincronização para uma MEF pode não existir em sua implementação, pois pode haver falhas na implementação. Considerando a MEF da Figura 3.1, a sequência $SS = aabb$ sempre leva a implementação ao estado inicial 1.

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

Uma *sequência de distinção DS* trata-se de uma sequência que permite a identificação de cada estado de uma MEF de forma única. Ao aplicar essa sequência a partir de um determinado estado, a saída produzida é única e possibilita definir em qual estado a MEF se encontrava originalmente. Em uma determinada MEF pode ou não existir uma sequência de distinção. Assim, para todo par de estados $(s_i, s_j) \in S$, $s_i \neq s_j$, caso DS exista, $\lambda(s_i, DS) \neq \lambda(s_j, DS)$. A MEF da Figura 3.1 possui a sequência de distinção $DS = ab$. A MEF da Figura 3.2, contudo, não possui uma sequência de distinção.

Um *conjunto de caracterização*, também denominado conjunto W, é um conjunto composto por sequências de entrada que, ao serem aplicadas em conjunto, fazem o papel de identificação única para cada estado. Ou seja, para quaisquer dois diferentes estados $(s_i, s_j) \in S$, existe uma sequência $\alpha \in W$, tal que $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Diferentemente de uma DS, um conjunto W pode possuir mais de uma sequência. Assim, quando se aplica uma sequência contida em W, é necessário utilizar algum meio de restaurar a MEF para o estado original, de forma que possam ser aplicadas as demais sequências. Um conjunto W unitário é equivalente a uma DS. Na MEF da Figura 3.1 pode ser adotado como conjunto de caracterização o conjunto $W = \{ab\}$ ou $W = \{aa, b\}$, por exemplo. Para a MEF da Figura 3.2, a distinção dos estados pode ser obtida com o conjunto $W = \{a, b\}$.

Seja as sequências α,φ, β sequências definidas em M, se $\beta = \alpha\varphi$, então α é um *prefixo* de β, denotado como $\alpha \leq \beta$. Para uma sequência β definida em M, $pref(\beta)$ denota o conjunto de todos os prefixos de β, ou seja, $pref(\beta) = \{\alpha | \alpha \leq \beta\}$. Dada uma sequência $\beta = \alpha\omega$, a notação $\beta \backslash \omega$ representa a sequência α, a qual é obtida da sequência β após a exclusão do sufixo ω.

Um *conjunto de distinção* $E = \{E_1, E_2, ..., E_n\}$ é um conjunto contendo $|S|$ sequências de identificação, uma para cada estado $s \in S$, tal que, E_s distingue s de qualquer outro estado de S e E_s contém um prefixo comum com as demais sequências de identificação de

E. Ou seja, dado o estado $s_i \in S$ e sua respectiva sequência de identificação $E_i = \{\beta_i\}$, se $E_i \in E$, então, para cada estado $s_j \in S$ e sua respectiva sequência de identificação $E_j = \{\beta_j\} \in E$, existe $\alpha \in \Omega(s_i) \cap \Omega(s_j)$, tal que $\alpha \leq \beta_i$, $\alpha \leq \beta_j$, e $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Um conjunto de distinção pode ser obtido por meio de uma sequência DS, de forma que para cada estado $s \in S$ é associada uma sequência $E_s = \{\alpha\}$, tal que α seja prefixo da sequência DS e distingua s dos demais estados de S. Dessa forma, para a MEF da Figura 3.1, considera-se o conjunto de distinção $E = \{E_1, E_2, E_3, E_4\}$, onde $E_1 = E_2 = E_3 = E_4 = \{ab\}$.

3.6 Métodos de Geração de Sequências de Verificação

O processo de geração de casos de teste é considerado uma tarefa complexa na atividade de teste de software (Hamon et al., 2004). Segundo Fujiwara et al. (1991), esse processo tem como intuito a geração de um conjunto de sequências de teste capaz de identificar falhas existentes em um sistema. Essas falhas são caracterizadas pela não conformidade entre a especificação e a implementação do sistema em teste.

No teste baseado em MEFs, diversos métodos foram desenvolvidos com o intuito de otimizar e automatizar esse processo. Esses métodos diferem por vários fatores, como, por exemplo, os tipos de sequências básicas utilizadas, as restrições exigidas para sua aplicação, o comprimento das sequências de teste geradas, a efetividade quanto à cobertura de falhas, à quantidade de operações *reset* contidas no conjunto de casos de teste, dentre outros. Uma operação *reset* é a representação de algum procedimento confiável capaz de inicializar a MEF e a implementação, conduzindo-as ao seu estado inicial. Contudo, essa operação pode não existir em determinados tipos de sistemas ou demandar recursos caros (tempo, mão de obra, capital, etc.) que tornam o teste impraticável.

De acordo com Fujiwara et al. (1991), um conjunto de sequências de teste deve prover duas características importantes em relação ao desempenho de sua execução. Primeiramente, é desejável que tanto a quantidade de sequências quanto o comprimento das sequências seja o menor possível, agilizando sua execução. Em segundo lugar, a aplicação dessas sequências deve ser capaz de identificar todas as falhas que a implementação possa conter.

Uma sequência de verificação (*checking sequence*) corresponde a uma sequência única de símbolos de entrada capaz de garantir a conformidade entre a implementação de um sistema e sua especificação (Simao e Petrenko, 2008). Em outras palavras, uma sequência de verificação é um conjunto n-completo, gerado a partir de M, composto por uma única sequência de teste. Dessa forma, esse tipo de conjunto de teste não requer a aplicação de operações *reset*. Esse fator possibilita a aplicação de sequências de verificação tanto

em sistemas que dispõem de operações *reset* quanto em sistemas que não dispõem de tal recurso (Simao e Petrenko, 2008).

Na literatura, é possível encontrar diversos métodos propostos para geração de sequências de verificação. Apesar de compartilharem o mesmo objetivo, cada método possui características particulares que influenciam em fatores como o domínio de aplicação do método e o tamanho da sequência gerada. Por exemplo, um dos fatores cruciais na geração de sequências de verificação está em garantir que, após a aplicação de alguma sequência de entrada, a implementação permanece em um estado conhecido. Com esse intuito, a maior parte dos métodos conhecidos na literatura são baseados na utilização de sequências de distinção (Chen et al., 2005; Gonenc, 1970; Hennie, 1964; Ural et al., 1997).

Como definido na Seção 3.5, uma sequência de distinção (ou sequência DS) é uma sequência única de entradas que permite a identificação individual de cada estado de uma MEF. A utilização desse tipo de sequência no processo de geração, apesar de, geralmente, produzir sequências de verificação menores, limita a aplicação do método apenas ao domínio de modelos que dispõem de tal recurso. Considerando essa limitação, alguns métodos adotam recursos alternativos às sequências de distinção, como, por exemplo, *conjuntos de distinção* e *conjuntos de caracterização*, também definidos na Seção 3.5. A utilização de conjuntos de distinção aumenta a abrangência do método, uma vez que, apesar desse conjunto também poder ser derivado a partir de uma sequência de distinção, existem MEFs providas de conjunto de distinção que não possuem uma sequência de distinção (Boute, 1974). No entanto, assim como as sequências de distinção, os conjuntos de distinção não existem para toda MEF (Simao e Petrenko, 2008). Conjuntos de caracterização, por outro lado, existem para toda MEF minimal (Gill, 1962). Devido a isso, métodos baseados nesse recurso se tornam ainda mais abrangentes (Hennie, 1964; Rezaki e Ural, 1995). Em contrapartida, os métodos baseados em conjuntos de caracterização existentes na literatura geram sequências de verificação exponencialmente longas. Esse fator habilita a investigação de novos métodos capazes de reduzir o tamanho das sequências geradas a partir da utilização de conjuntos de caracterização. Essa investigação é o propósito desta dissertação e será discutido no Capítulo 4.

Dentre os métodos apresentados nesta seção, encontram-se métodos baseados na utilização de sequências de distinção (Chen et al., 2005; Gonenc, 1970; Hierons e Ural, 2002, 2006; Ural et al., 1997), conjuntos de distinção (Simao e Petrenko, 2008) e conjuntos de caracterização (Hennie, 1964; Rezaki e Ural, 1995). É importante enfatizar que os métodos de Hennie (1964) e Rezaki e Ural (1995) estão inseridos no contexto mais específico deste trabalho.

3.6.1 Método de Gonenc (1970)

O método desenvolvido no trabalho de Gonenc (1970), conhecido por método DS, tem como principal característica a utilização de sequências de distinção no processo de geração de sequências de verificação. Esse método visa a percorrer, de forma explícita, todos os estados e transições da MEF de especificação. Para isso, além de possuir sequência de distinção, é necessário que a MEF em teste seja completa, minimal e fortemente conexa, conforme definido na Seção 3.4.

O processo de geração é composto pela concatenação de duas sequências: Sequência Inicial (SI) e Sequência de Verificação (SV). A sequência SI tem como objetivo inicializar a MEF, de maneira que o estado atual da MEF, após sua execução, seja o estado inicial. Para isso, uma vez que a MEF não possui a operação *reset*, é utilizada uma *sequência de sincronização*, conforme definido na Seção 3.5. Como consequência disso, a aplicação desse método fica condicionada à existência de uma sequência de sincronização, visto que nem toda MEF minimal possui esse recurso (Gill, 1962).

A sequência SV deve garantir que cada estado e transição da MEF em teste seja verificado pelo menos uma vez. Entende-se por verificação o reconhecimento de um estado ou transição por meio da aplicação da sequência DS. Para isso, a sequência SV é dividida em duas subsequências: sequência α e sequência β. A sequência α busca atingir dois propósitos: 1) verificar todos os estados e 2) determinar todos os *estados consequentes*. Um estado consequente corresponde ao estado final q_j obtido após a aplicação da sequência de distinção X_d no estado s_j, ou seja, $q_j = \delta(s_j, X_d)$, onde $s_j \in S$. A sequência β, por sua vez, visa a verificar cada transição da MEF em teste.

Com o intuito de cumprir os dois propósitos da sequência α, a sequência de distinção X_d é aplicada duas vezes para cada estado. Ou seja, a sequência X_dX_d é aplicada para cada estado, de forma que a primeira parte X_d é responsável por verificar o estado de origem e a segunda parte por verificar o estado consequente. Considerando que a sequência α deve ser única, caso o estado final obtido após a aplicação da sequência X_dX_d já tenha sido verificado, é necessário a concatenação de uma *sequência de transferência* que transfira a execução para algum estado ainda não verificado. Dessa forma, a sequência X_dX_d é aplicada até que todos os estados tenham sido verificados.

Como apoio na geração dessa sequência, após a seleção da menor sequência de distinção X_d possível, é gerado o *grafo* X_d (Gonenc, 1970). Nesse grafo, os vértices representam os estados da MEF e as arestas representam ligações entre o estado origem e o estado destino correspondentes à aplicação da sequência X_d. Na Figura 3.3 é apresentado um exemplo de grafo X_d gerado a partir da MEF representada na Figura 3.1, considerando a sequência $X_d = ab$. O estado 2, por exemplo, possui ligação com o estado 4, uma vez que

o estado 4 corresponde ao estado destino da sequência X_d, quando aplicada no estado de origem 2.

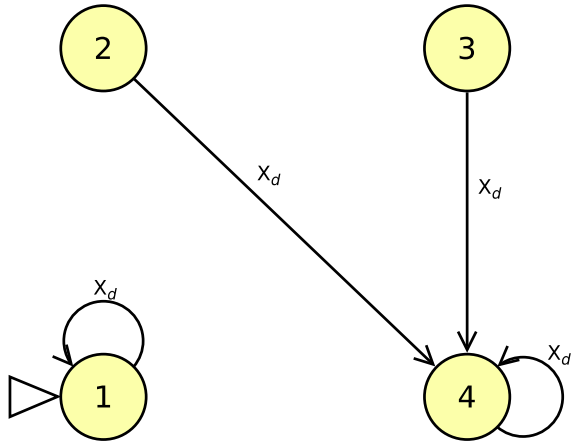

Figura 3.3: Grafo X_d referente à MEF da Figura 3.1.

No processo de geração da sequência α, quando a sequência X_d é aplicada a algum estado de origem, esse estado é marcado como *verificado*. Caso o estado destino ainda não tenha sido verificado, aplica-se X_d novamente, como no passo anterior. Caso contrário, quando o estado destino já está verificado, a sequência X_d é aplicada mais uma vez e, então, é concatenada a uma sequência de transferência que leve a execução para um estado de origem ainda não verificado. Esse processo se repete até que todos os estados sejam verificados.

Considerando o grafo da Figura 3.3, primeiramente, é escolhido o estado 1 como o estado de origem. Nesse estado, é aplicada a sequência $X_d = ab$ e, em seguida, o estado é marcado como verificado. No próximo passo, aplica-se novamente X_d no estado destino 1. Uma vez que tal estado já foi verificado no passo anterior, é concatenada a sequência de transferência b que leva a execução para o estado 2, ainda não verificado. Esse procedimento é repetido até a formação da sequência $\alpha = ababbababababaabab$, quando todos os estados, bem como os estados consequentes, já foram verificados.

Para geração da sequência β, o estado final de cada transição deve ser verificado, aplicando-se a sequência X_d. Uma transição é representada por um símbolo de entrada x aplicado a um determinado estado s_j, onde $x \in X$ e $s_j \in S$. Dessa forma, para verificar uma determinada transição, a execução da MEF deve estar no estado s_j correspondente e, então, executar a sequência xX_d. Eventualmente, é necessário utilizar sequências de transferência para transferir a execução da MEF para algum estado que contém transições ainda não verificadas.

De forma análoga ao processo de geração da sequência α, para geração da sequência β, é construído o *grafo* β. Nesse grafo, os vértices representam os estados da MEF e as arestas dirigidas representam a ligação entre o estado de origem e o estado de destino

após aplicação de cada sequência xX_d. Para exemplificar esse processo, a MEF da Figura 3.1 é representada pelo grafo β da Figura 3.4.

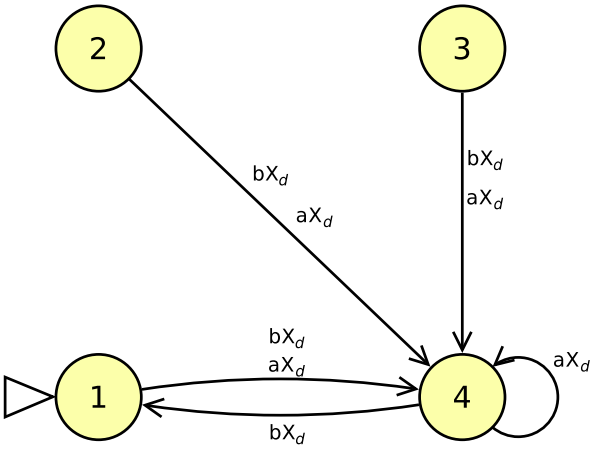

Figura 3.4: Grafo β referente à MEF da Figura 3.1.

Ao percorrer todas as arestas do grafo β, todas as transições serão verificadas. Dessa forma, a sequência β é composta pelo conjunto de todos os caminhos disjuntos existentes no grafo β, conectados por meio de sequências de transferência. Esse processo pode gerar sequências distintas que cumprem o mesmo objetivo, no entanto, com tamanhos diferentes. Com o intuito de otimizar esse processo e selecionar a menor sequência possível, no trabalho de Gonenc (1970) é apresentado um conjunto de diretrizes que permitem reduzir o número de arestas que compõem o grafo β.

A primeira redução está no fato de que, ao aplicar a sequência X_dX_d em cada estado, na geração da sequência α, a transição correspondente ao último símbolo de X_d já é verificada por meio da segunda parte X_d dessa sequência. Assim, para cada estado, não é necessário verificar a aresta referente ao último símbolo de entrada da sequência X_d. Para exemplificar, considerando a sequência $X_d = ab$, todas as arestas compostas por bab devem ser excluídas do grafo β.

Analogamente, na sequência α, uma vez que os estados de origem e destino das sequências de transferência são verificados, as transições no grafo β correspondentes ao último símbolo dessas sequências também podem ser excluídas do grafo. Assim, considerando que na sequência α foram utilizadas as sequências de transferência b (do estado 1 para o estado 2) e a (do estado 4 para o estado 3), as arestas bab, partindo do estado 1, e aab partindo do estado 4, podem ser excluídas do grafo β.

A Figura 3.5 representa o grafo β da Figura 3.4 de forma reduzida. A sequência β corresponde à execução de todas as arestas do grafo reduzido. Por fim, partindo do estado 1 e aplicando as respectivas sequências de transferência que permitam percorrer as arestas dos estados 2 e 3, é gerada a sequência $\beta = aabbbaabaaab$.

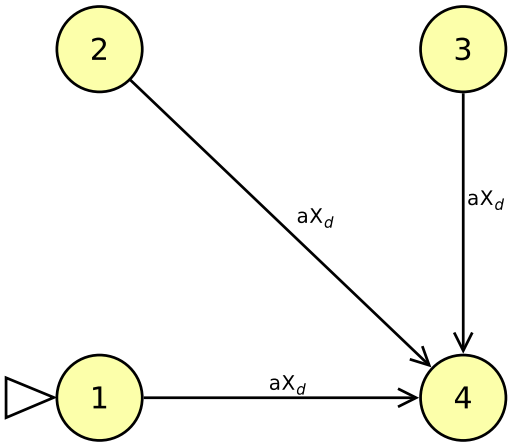

Figura 3.5: Grafo β reduzido.

Por fim, as sequências α e β são concatenadas para formação da sequência $SV = ababbababababaabbabbaabbbbaabaaab$ de tamanho 29. É importante observar que a concatenação dessas sequências só é possível quando o estado final referente à sequência α coincide com o estado inicial da sequência β. Dessa forma, para permitir a concatenação dessas sequências, entre as sequências α e β foi inserida a sequência de transferência b, referente à transferência do estado 4 para o estado 1. Considerando a sequência $SI = aabb$, após a concatenação das sequências SI e SV, o método DS gera a sequência $ST_{DS} = aabbababbababababaabbabbaabbbbaabaaab$ de tamanho 33.

3.6.2 Método de Ural et al. (1997)

Ural et al. (1997) propõem um novo método para geração de sequências de verificação baseado na utilização de sequências de distinção, assim como Gonenc (1970). Contudo, os autores estipulam condições de suficiência que, além de garantir a completude das sequências geradas, permitem otimizar o tamanho das sequências de verificação geradas, em relação ao método de Gonenc (1970). Tais condições de suficiência fundamentaram o desenvolvimento de diversos métodos de geração de sequências de verificação, tais como os métodos propostos por Hierons e Ural (2002), Hierons e Ural (2006) e Chen et al. (2005), os quais são sucintamente discutidos nesta seção.

No método de Ural et al. (1997), uma MEF M é representada por um grafo $G = (V, E)$, onde $V = \{v_1, v_2, ..., v_n\}$ representa o conjunto S de estados de M, e o conjunto $E = \{(v_j, v_k; x/y) : v_j, v_k \in V\}$ representa as transições definidas em M. Cada aresta $e = (v_j, v_k; x/y) \in E$ representa uma transição de M saindo do estado s_j para o estado s_k, $\{s_j, s_k\} \in S$, que consome a entrada $x \in X$ e produz a saída $y \in Y$, conforme Seção 3.3. Um caminho em G é representado por $P = (n_1, n_r; Q) = (n_1, n_2; L_1)(n_2, n_3; L_2)...(n_{r-1}, n_r; L_{r-1})$, de forma que $Q = L_1...L_{r-1}$ representa uma sequência de entrada/saída que conduz o caminho P do nó n_1 ao nó n_r de G, onde $L_j = x_j/y_j$, $1 \leq j \leq r-1$ e $r > 1$.

Por definição, um nó n_i de P é dito *reconhecido* em Q como algum estado a de M se há uma subsequência X_d em Q que, ao ser aplicada em n_i, gera uma saída equivalente a $\lambda(a, X_d)$. Da mesma forma, uma aresta $e = (a, b; x/y)$ de G, é dita *verificada* em Q caso satisfaça a duas condições: 1) o caminho P, conduzido por Q, contém um subcaminho $(n_i, n_j; (xX_d)/\lambda(a, xX_d))$ para $e = (a, b; x/y)$; e 2) n_i é reconhecido em Q como o estado a de M, sendo n_i o estado inicial de $(n_i, n_j; (xX_d)/\lambda(a, xX_d))$.

De acordo com o teorema proposto por Ural et al. (1997), se todas as arestas de G são verificadas em Q, Q é uma sequência de verificação de M. Assim, dada uma sequência de distinção X_d pertencente a M, o método de Ural et al. (1997) tem como objetivo encontrar uma sequência Q que conduz o menor caminho em G, de maneira que toda aresta de G seja verificada em Q. Para isso, assim como em Gonenc (1970), é exigido que a MEF M seja completa, minimal e fortemente conexa.

O método de Ural et al. (1997) é baseado na resolução do problema *Rural Chinese Postman* (RCP) (Aho et al., 1991), o qual se encontra na categoria de problemas NP-Completo e é derivado do problema *Rural Postman* (RP) (Kuan, 1962). Dado o subconjunto E' de arestas e os vértices v_i e v_j, ambos pertencentes a um grafo G, o problema RP consiste na busca de um caminho em G que inicie no vértice v_i e termine em v_j, de maneira que, nesse caminho, estejam inclusas todas as arestas de E'. O problema RCP consiste no problema RP, porém, com custo mínimo.

Para que seja possível a geração de uma sequência de verificação baseada no problema RCP, é construído um novo grafo $G' = (V', E')$ baseado no grafo $G = (V, E)$, de forma que o conjunto de arestas E', além de conter as arestas E, é acrescido de mais dois principais subconjuntos de arestas, denominados de E_α e E_c. O conjunto E_α é formado por um conjunto de arestas que, juntas, contém subsequências que tornam todos os estados de M reconhecidos. O procedimento de construção de E_α é similar ao procedimento de construção da sequência α do método de Gonenc (1970) (Seção 3.6.1). O conjunto de arestas E_c, por sua vez, é formado por subcaminhos de G que permitem a uma determinada aresta $e \in E$ ser verificada, ou seja, cada aresta contida em E_c constitui um subcaminho $(n_i, n_j; (xX_d)/\lambda(a, xX_d))$ para $e = (a, b; x/y)$.

Após a construção do grafo G', é aplicado um algoritmo para resolução do problema RCP (Aho et al., 1991), de forma que a solução encontrada em G' conduza um caminho que, partindo do estado inicial de M, contenha, obrigatoriamente, o conjunto de arestas composto por $E_\alpha \cup E_c$. Dessa forma, de acordo com o teorema proposto por Ural et al. (1997), a solução desse problema constitui uma sequência de verificação de G, uma vez que contém todos os estados reconhecidos e todas as arestas verificadas.

O método proposto por Ural et al. (1997) fundamentou o desenvolvimento de outros trabalhos que, em sua maioria, otimizam procedimentos internos ao método de Ural et

al. (1997) visando a reduzir o tamanho das sequências de verificação geradas. Dentre tais métodos encontram-se os métodos de Hierons e Ural (2002), Hierons e Ural (2006) e Chen et al. (2005).

No trabalho de Hierons e Ural (2002), o método de Ural et al. (1997) é modificado em relação aos procedimentos de geração dos conjuntos E_α e E_c. Para formação do conjunto E_α, o procedimento adotado em Ural et al. (1997) utiliza o conceito de sequências α, onde cada sequência α reconhece um subconjunto de estados da MEF M. Em seu trabalho, Hierons e Ural (2002) aplicam duas modificações: a primeira na definição das sequências α, estendendo o conceito de sequências α para o de sequências α', gerando o conjunto $E_{\alpha'}$; e a outra no uso de tais sequências para composição do conjunto E_c. A utilização desses novos conceitos proporciona algumas vantagens em relação ao método de Ural et al. (1997), que resultam diretamente na redução do tamanho global das sequências de verificação geradas.

Hierons e Ural (2006) aprofundam em seu trabalho os conceitos apresentados em Hierons e Ural (2002). Em seu trabalho anterior, as modificações e conceitos propostos em relação às sequências α' acarretam na geração de novos conjuntos que são combinados na composição do grafo G'. Contudo, nesse trabalho, não é abordada a maneira pela qual os elementos de tais conjuntos são escolhidos. Assim, Hierons e Ural (2006) propõem novas estratégias eficientes para composição desses conjuntos, que também acarretam na redução global do tamanho das sequências geradas.

Chen et al. (2005) exploram a eliminação de subsequências redundantes, passíveis de serem geradas no método de Hierons e Ural (2002). Os conjuntos $E_{\alpha'}$ e E_c, gerados no método de Hierons e Ural (2002), são combinados na composição do grafo G' e, apesar de possuírem objetivos distintos, podem conter elementos redundantes entre si. Dessa forma, no trabalho de Chen et al. (2005), é investigado um método que omite as subsequências de E_c que se referem a transições já verificadas, indiretamente, no conjunto $E_{\alpha'}$. Dessa forma, as sequências de verificação geradas omitem subsequências redundantes, acarretando na redução de seu tamanho.

Para exemplificar o método de Ural et al. (1997), considere a MEF da Figura 3.1 e a sequência de distinção $X_d = ab$. Em resumo, para geração do grafo $G' = (V', E')$, o conjunto de vértices V' é composto pelos subconjuntos V e U', onde $U' = \{v_i' : \text{para cada } v_i \in V\}$ e o conjunto de arestas E' é composto pelos conjuntos E_α, E_c, E_ϵ e E'', definidos a seguir:

- $E_\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$, onde:

  $\alpha_1 = (v_1, (\delta(v_1, X_dX_d))'; X_dX_d/\lambda(v_1, X_dX_d))$,

  $\alpha_2 = (v_2, (\delta(v_2, X_dX_dX_d))'; X_dX_dX_d/\lambda(v_2, X_dX_dX_d))$,

$$\alpha_3 = (v_3, (\delta(v_3, X_d X_d))'; X_d X_d / \lambda(v_3, X_d X_d)),$$

- $E_c = \{\beta_{1a}, \beta_{1b}, \beta_{2a}, \beta_{2b}, \beta_{3a}, \beta_{3b}, \beta_{4a}, \beta_{4b}\}$, onde:

$$\beta_{1a} = (v'_1, (\delta(v'_1, aX_d))'; (aX_d)/\lambda(v_1, aX_d)),$$
$$\beta_{1b} = (v'_1, (\delta(v'_1, bX_d))'; (bX_d)/\lambda(v_1, bX_d)),$$
$$\beta_{2a} = (v'_2, (\delta(v'_2, aX_d))'; (aX_d)/\lambda(v_2, aX_d)),$$
$$\beta_{2b} = (v'_2, (\delta(v'_2, bX_d))'; (bX_d)/\lambda(v_2, bX_d)),$$
$$\beta_{3a} = (v'_3, (\delta(v'_3, aX_d))'; (aX_d)/\lambda(v_3, aX_d)),$$
$$\beta_{3b} = (v'_3, (\delta(v'_3, bX_d))'; (bX_d)/\lambda(v_3, bX_d)),$$
$$\beta_{4a} = (v'_4, (\delta(v'_4, aX_d))'; (aX_d)/\lambda(v_4, aX_d)),$$
$$\beta_{4b} = (v'_4, (\delta(v'_4, bX_d))'; (bX_d)/\lambda(v_4, bX_d));$$

- $E_\epsilon = \{(v'_i, v_i; \epsilon) : \text{para cada } v_i \in V\}$; e
- $E'' = \{(v'_i, v'_j; x/y) : \text{para cada } (v_i, v_j; x/y) \in E\}$.

Após construir o grafo G' (Figura 3.6), contendo todos os vértices e arestas definidos acima, aplica-se o algoritmo de resolução do problema RCP, proposto em Aho et al. (1991). Assim, deve ser encontrado o menor caminho em G' que, obrigatoriamente, contenha todas as arestas de $E_\alpha \cup E_c$. Os conjuntos E_ϵ e E'', por sua vez, constituem arestas auxiliares para obtenção do menor caminho.

Partindo do estado 1, o menor caminho para o grafo G' gerado consiste na sequência $\{\alpha_1\beta_{1a}\beta_{4b}\beta_{1b}\beta_{4a}bb\alpha_2bb\beta_{2a}bb\beta_{2b}a\alpha_3a\beta_{3a}a\beta_{3b}\}$. Por fim, ao substituir os rótulos pelos respectivos símbolos de entrada, é gerada a sequência de verificação $ST_{Ural\,et\,al.} = abaabbabbabaabbbabbbaabbbbabaabaaababab$ de tamanho 39.

3.6.3 Método de Simao e Petrenko (2008)

O método de Simao e Petrenko (2008) apresenta duas importantes características que o diferencia dos demais métodos já apresentados. Primeiro, ao invés de utilizar a modelagem baseada na teoria de grafos para minimização do tamanho das sequências geradas, esse método realiza a otimização das sequências por meio de busca local. Outra característica, está no fato de permitir que esse método seja aplicado a MEFs parciais, ao contrário dos métodos anteriores que exigem MEFs completas.

A utilização de estratégia baseada em busca local proporciona algumas importantes vantagens sobre os métodos baseados em modelagem teórica de grafos. Uma das vantagens consiste no fato de não haver dependência de parâmetros externos para sua execução, visto que as escolhas realizadas no método dependem apenas das informações disponíveis

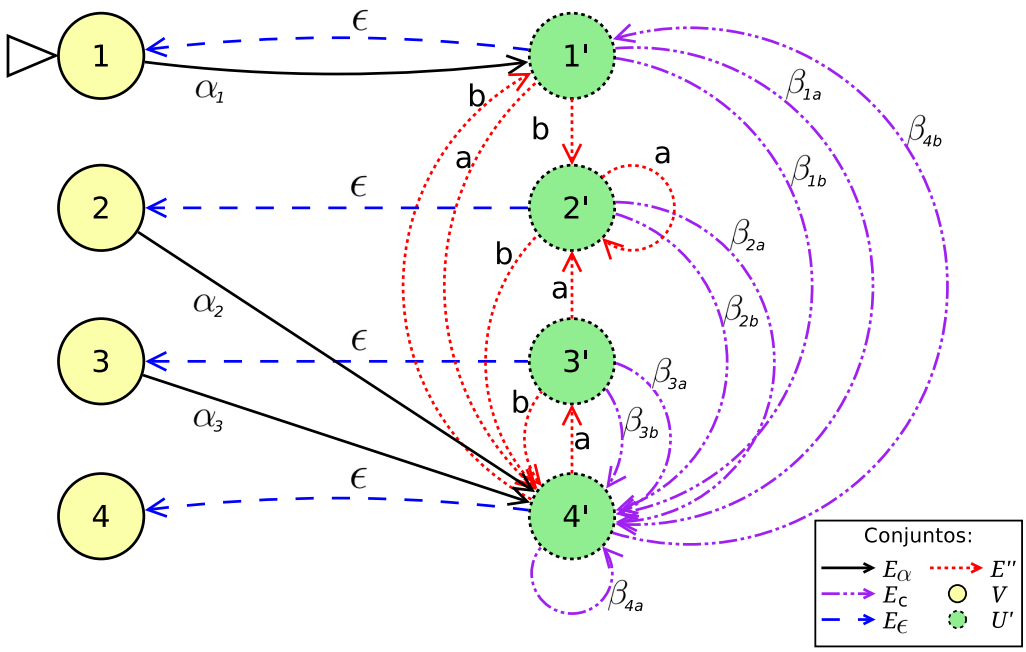

Figura 3.6: Grafo G' referente à MEF da Figura 3.1.

durante a execução. Já os métodos baseados no método de Ural et al. (1997), por exemplo, usufruem de alguns parâmetros, como as sequências α, que são gerados por meio de algoritmos externos. A maneira como esses parâmetros são gerados influencia no tamanho das sequências de verificação final geradas, podendo torná-las, de fato, sub-otimizadas.

Nesse método, a identificação dos estados é realizado por meio de conjuntos de distinção, definidos na Seção 3.5. A utilização desse tipo de identificação amplia a abrangência do método, uma vez que, apesar desse conjunto também poder ser derivado a partir de uma sequência de distinção, existem MEFs providas de conjunto de distinção que não possuem uma sequência de distinção (Boute, 1974). Outra característica encontrada no método de Simao e Petrenko (2008) está na possibilidade de utilização de operações *reset*, para MEFs que possuam tal recurso, na tentativa de redução do tamanho das sequências de verificação.

O método de Simao e Petrenko (2008) segue um procedimento classificado como uma estratégia "gulosa", uma vez que adota apenas a perspectiva local em sua execução. A cada passo, duas situações podem ocorrer: *i*) caso o estado final da sequência gerada até o momento seja conhecido, escolhe-se uma nova transição para ser verificada, de forma que o estado de origem de tal transição seja alcançado pela menor sequência de transferência possível; e *ii*) caso contrário, aplica-se a sequência de identificação adequada para verificação do estado. Esse procedimento é repetido até que todas as transições sejam verificadas.

Para aplicação desse método, assume-se que tanto a especificação quanto a implementação já estão inicializadas e, por isso, diferente do método de Gonenc (1970), não

faz uso de sequências de sincronização. Para exemplificar o método, considere a MEF M da Figura 3.1 e o conjunto de distinção $E = \{E_1, E_2, E_3, E_4\}$, tal que as sequências $E_1 = E_2 = E_3 = E_4 = \{ab\}$ referem-se, respectivamente, aos estados $1, 2, 3$ e 4 de M.

Na Tabela 3.3 é apresentada a sequência de passos na geração da sequência de verificação ω, para a MEF da Figura 3.1. Na primeira coluna é representada cada iteração do procedimento. Na segunda coluna, é apresentada a formação da sequência ω gerada até a interação i. Na terceira coluna, por sua vez, é representado o conjunto de sequências cujo estado final está reconhecido (ou confirmado) (Simao e Petrenko, 2008). Por fim, a quarta coluna exibe as transições já verificadas, ou seja, transições cujo o estado inicial e o estado final estão reconhecidos por meio das sequências em $R(\omega_i)$.

Ao final do procedimento, é gerada a sequência $ST_{Simao\,e\,Petrenko} = ababaababaabab\ bbababaaabaaaab$ de tamanho 29. Ao comparar com os métodos anteriores, esse método obteve um desempenho satisfatório, visto que gerou uma sequência de verificação de tamanho equivalente à sequência gerada pelo método de Gonenc (1970) (desconsiderando a sequência de sincronização) e menor que a sequência gerada pelo método de Ural et al. (1997), com diferença de 10 símbolos.

Iteração (i)	Sequência ω_i	Sequências confirmadas $R(\omega_i)$	Transições verificadas $U(\omega_i)$
0	ϵ	$\varnothing$	$\varnothing$
1	ab	$\{\epsilon\}$	$\varnothing$
2	$abab$	$\{\epsilon, ab\}$	$\varnothing$
3	$ababaab$	$\{\epsilon, a, ab\}$	$\{(1,a),(4,b)\}$
4	$ababaabab$	$\{\epsilon, a, aab, ab\}$	$\{(1,a),(4,b)\}$
5	$ababaababaab$	$\{\epsilon, a, aa, aab, ab\}$	$\{(1,a),(3,b),(4,a),(4,b)\}$
6	$ababaababaabab$	$\{\epsilon, a, aa, aaab, aab, ab\}$	$\{(1,a),(3,b),(4,a),(4,b)\}$
7	$ababaababaabab\ bbab$	$\{\epsilon, a, aa, aaab, aab, ab, b\}$	$\{(1,a),(1,b),(3,b),(4,a),(4,b)\}$
8	$ababaababaabab\ bbabab$	$\{\epsilon, a, aa, aaab, aab, ab, b, bb\}$	$\{(1,a),(1,b),(2,b),(3,b),(4,a),(4,b)\}$
9	$ababaababaabab\ bbababaaab$	$\{\epsilon, a, aa, aaa, aaab, aab, ab, b, bb\}$	$\{(1,a),(1,b),(2,b),(3,a),(3,b),(4,a),(4,b)\}$
10	$ababaababaabab\ bbababaaabaaaab$	$\{\epsilon, a, aa, aaa, aaaa, aaab, aab, ab, b, bb\}$	$\{(1,a),(1,b),(2,a),(2,b),(3,a),(3,b),(4,a),(4,b)\}$

Tabela 3.3: Iteração do método de Simao e Petrenko (2008).

3.6.4 Método de Hennie (1964)

No trabalho de Hennie (1964) é proposto um método pioneiro para geração de sequências de verificação. Esse método torna-se importante na literatura pois abrange o domínio de

MEFs que não dispõem de sequências de distinção. A identificação de um estado durante a execução do teste é baseada no conjunto de caracterização (conjunto W) (Seção 3.5). Uma vez que esse conjunto sempre existe para uma MEF minimal, a estratégia utilizada por esse método permite sua aplicação em um domínio mais abrangente de MEFs. No entanto, apesar de abrangente, o método produz sequências de verificação exponencialmente longas em relação ao tamanho de W, podendo, inclusive, inviabilizar a execução do teste. As sequências de teste produzidas por esse método possuem essa característica em relação ao seu tamanho devido à necessidade de construção de sequências parciais que garantem a identificação de um determinado estado. Essas sequências foram definidas no trabalho de Hennie (1964) e são denominadas de *sequências de localização*.

A ideia básica para identificação de um estado utilizando o conjunto W é a aplicação de cada sequência do conjunto W sobre o mesmo estado. Contudo, considerando que uma sequência de verificação é composta por uma sequência única e não conta com operações *reset*, garantir que todas as sequências são aplicadas sobre o mesmo estado constitui um problema não trivial. Considerando uma MEF com no máximo n estados, é possível perceber que no máximo n estados diferentes podem produzir uma mesma saída para uma sequência de entrada específica. Dessa forma, se uma sequência de entrada é aplicada $n+1$ vezes e sempre produz a mesma saída, pode-se afirmar que o último estado alcançado é um estado já percorrido e produz uma saída conhecida para tal sequência de entrada.

Uma sequência de localização L_i para um estado $s_i \in S$ é construída a partir dessa constatação e permite a localização de um estado s_i' pertencente a uma implementação de M cujo comportamento é equivalente a s_i, em relação às sequências de caracterização derivadas a partir de M (Hennie, 1964). Dessa forma, seja $W = \{w_1, w_2, ..., w_r\}$ um conjunto de caracterização para M, onde w_τ constitui uma sequência de caracterização pertencente a W, $1 \leq \tau \leq r$. Seja também U_k^i, $1 \leq k \leq r-1$, uma sequência de entrada contendo o prefixo w_k e cuja aplicação leve a execução de M do estado s_i de volta para o estado s_i, ou seja, $U_k^i = w_k I_i^k$ onde I_i^k é uma sequência de transferência de $\delta(s_i, w_k)$ para s_i. Além disso, considere U_r^i uma sequência de entrada com o prefixo w_r que não necessariamente traga a execução de M do estado s_i de volta ao estado s_i, ou seja, $U_r^i = w_r I_i^r$, onde, se I_i^r não é uma sequência vazia, então I_i^r é uma sequência de transferência de $\delta(s_i, w_r)$ para s_i.

Definição 3.1. Uma *sequência de localização* para um estado s_i é $L_i = F(r, \{U_1^i, U_2^i, ..., U_r^i\})$ onde $F : \mathbb{N} \times 2^{\Omega(s_i)} \rightarrow \Omega(s_i)$ é uma função recursiva tal que $F(2, \{a_1, a_2\}) = (a_1)^{n+1}.a_2$, $F(k, \{a_1, a_2, ..., a_k\}) = (F(k-1, \{a_1, a_2, ..., a_{k-1}\}))^{n+1} \,.\, F(k-1, \{a_1, a_2, ..., a_{k-2}, a_k\})$, $3 \leq k \leq r$, e a_i, $1 \leq i \leq k$, é uma sequência de entrada para M.

Uma sequência de localização L_i para um estado s_i tem a propriedade de que todo o conjunto de sequências de entrada $w_1, w_2, ..., w_r$ é aplicado sobre o estado $\delta(s_i, L_i \backslash U_r^i)$. O estado $\delta(s_i, L_i \backslash U_r^i)$ é definido como o *estado relacionado a* L_i. $L = \{L_1, L_2, ..., L_n\}$ representa o conjunto de sequências de localização para M onde L_k denota a sequência de localização para o estado $s_k \in S$, $1 \leq k \leq n$.

Para exemplificar, considere o conjunto $W = \{w_1, w_2, w_3\}$ derivado a partir de uma MEF M. Uma sequência de localização para s_i é $L_i = F(3, \{U_1^i, U_2^i, U_3^i\}) = F(3, \{w_1 I_i^1, w_2 I_i^2, w_3 I_i^3\}) = ((w_1 I_i^1)^{n+1} w_2 I_i^2)^{n+1}.(w_1 I_i^1)^{n+1} w_3 I_i^3$. Na Figura 3.7 é ilustrada uma sequência de localização para um determinado estado s construída a partir de um conjunto W composto por duas sequências de entrada. Em termos práticos, considerando a MEF apresentada na Figura 3.2 e o conjunto $W = \{a, b\}$, na Tabela 3.4 são apresentadas as sequências de localização para cada estado, bem como as saídas esperadas e os respectivos *estados relacionados*.

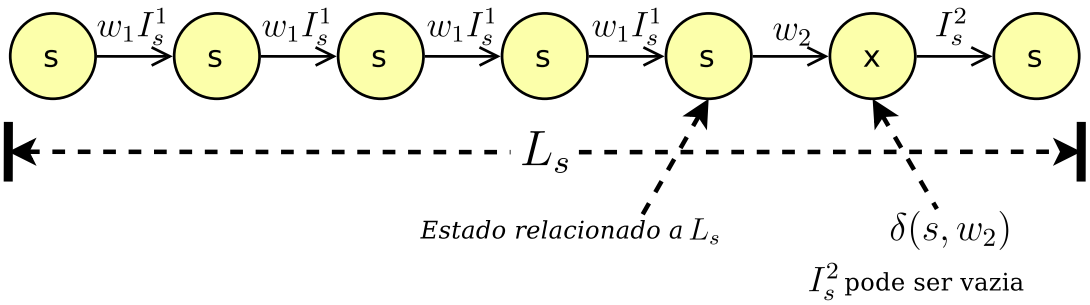

Figura 3.7: L_s e *Estado relacionado a* L_s – adaptado de Rezaki e Ural (1995).

Estado	L_i	$\lambda(s_i, L_i)$	*Estado relacionado a* L_i
1	$L_1 = (ab)^4 b$	$(00)^4 1$	$\delta(s_1, L_1 \backslash b) = \delta(s_1, abababab)$
2	$L_2 = (aa)^4 b$	$(10)^4 0$	$\delta(s_2, L_2 \backslash b) = \delta(s_2, aaaaaaaa)$
3	$L_3 = (aa)^4 b$	$(01)^4 0$	$\delta(s_3, L_3 \backslash b) = \delta(s_3, aaaaaaaa)$

Tabela 3.4: Sequências de localização para a MEF apresentada na Figura 3.2 (Método de Hennie (1964)).

A partir de uma sequência de localização é possível garantir a presença de um determinado estado em uma implementação (*estado relacionado a* L_i). Porém, somente com a sequência de localização não é possível determinar qual o estado alcançado ao final de sua aplicação. Uma maneira de identificar o estado alcançado está na aplicação de todas as sequências de caracterização que compõem o conjunto W sobre o mesmo estado final. Isso é possível por meio da utilização de sequências de transferência aliadas às sequências de localização. Por exemplo, para a MEF da Figura 3.2, considere o conjunto $W = \{w_1, w_2\} = \{a, b\}$ e a sequência $L_1 a b L_1 b$ iniciando no estado "1". Ao aplicar a sequência L_1, se a saída gerada pela implementação estiver em conformidade com a MEF

de especificação, então, garante-se que a execução passou pelo estado "1" e se encontra em um estado desconhecido s. Aplica-se então $w_1 = a$ sobre s seguido da sequência de transferência b com o intuito de retornar a execução ao estado "1". Em seguida, aplica-se novamente L_1. Uma vez que trata-se de uma MEF determinística, os estados alcançados após ambas aplicações de L_1 correspondem ao mesmo estado. Por fim, a sequência $w_2 = b$ é aplicada sobre o estado alcançado. Se a implementação estiver em conformidade com a especificação em relação a W, pode-se garantir que o estado alcançado após a aplicação de L_1 é o estado "3".

Seguindo o mesmo raciocínio, cada transição de M pode ser verificada por meio da aplicação de W sobre seu estado final. De maneira similar ao procedimento anterior, considere agora a sequência $L_1abL_1b.abL_1bb$ também iniciando no estado "1". A sequência é dividida em duas partes separadas por um ponto de concatenação para um melhor entendimento. A primeira parte da sequência foi explanada acima e resulta na identificação do estado alcançado após aplicação de L_1. Uma vez que a aplicação de L_1 resulta no estado "3", o último estado alcançado após aplicação da primeira parte da sequência corresponde à transição b do estado "3" para o estado "1". Contudo, o estado final dessa transição ainda não foi verificado. Dessa forma, aplica-se a sequência $w_1 = a$ seguida da sequência de transferência b que retorna a execução ao estado "1". Novamente, a sequência L_1b é aplicada com o intuito de retornar ao estado final a ser verificado. Por fim, a sequência $w_2 = b$ é aplicada. Como ambas as sequências de caracterização foram aplicadas sobre o estado final da transição $(3, b)$, se a saída produzida estiver em conformidade com a especificação, pode-se garantir que o estado final dessa transição é o estado "1".

Assim, seguindo o procedimento exemplificado acima, o método proposto por Hennie (1964) tem como objetivo gerar uma sequência de verificação ω na qual todos os estados de M são reconhecidos e todas as transições de M são verificadas. Além disso, em seu trabalho, Hennie aborda algumas otimizações que acarretam na redução da sequência de verificação gerada. Por exemplo, a ordem na qual os estados e transições são verificados pode poupar a aplicação de sequências de transferência. Uma outra otimização utilizada está na redução do número de repetições realizadas por cada sequência de caracterização na construção das sequências de localização. Essa otimização provém do conhecimento prévio do número máximo de estados com uma saída específica. Por exemplo, considere a construção de L_i para o estado $s_i \in S$ para uma MEF de 5 estados. Há o conhecimento prévio, de acordo com a especificação, de que no máximo 2 estados produzem uma saída específica para a sequência $w_\tau \in W$. Então, em L_i, ao invés de repetir a sequência $w_\tau I_i^\tau$ por $n+1 = 6$ vezes, repete-se por apenas $2+1$ vezes. Essa otimização tem impacto direto na redução de ω, uma vez que a construção de ω é baseada em sequências de localização.

Para a MEF da Figura 3.2, considerando as otimizações discutidas, o método de Hennie (1964) gerou uma sequência de verificação de tamanho 171.

3.6.5 Método de Rezaki e Ural (1995)

O método de Rezaki e Ural (1995), assim como o método de Hennie (1964), é baseado em sequências de localização que, por sua vez, são baseadas nos conjuntos de caracterização. Essa característica torna o método abrangente em relação ao domínio de aplicação, contudo, também o torna ineficiente no que se refere ao tamanho das sequências de verificação geradas. O método proposto por Rezaki e Ural (1995) aborda uma estratégia diferente ao utilizar grafos na construção das sequências de verificação. Essa estratégia permite reduzir o tamanho das sequências geradas, uma vez que realiza otimizações globais quanto a ordem em que os estados e transições são reconhecidos. O método apresentado constitui um modelo geral e, portanto, ignora demais otimizações.

A notação utilizada por Rezaki e Ural é similar à utilizada no trabalho de Ural et al. (1997). Dessa forma, nesta seção, serão adotadas as mesmas notações introduzidas na Seção 3.6.2. Dada uma MEF M, representada pelo grafo $G = (V, E)$, e um caminho P em G, representado pela sequência Q, o método tem como objetivo encontrar um caminho P no qual cada aresta (transição) de G é verificada. Rezaki e Ural (1995) formalizam os conceitos de reconhecimento de estados e verificação de transições. Em resumo, um estado $s_i \in S$ é reconhecido em Q quando:

1. A sequência de localização L_i é aplicada sobre s_i em Q (*estado relacionado a* L_i);
2. Todas as sequências de caracterização de W são aplicadas sobre s_i em Q de forma que s_i seja alcançado por uma sequência de transferência que tem como estado inicial um estado s_j já reconhecido por (1); ou
3. Dado um estado s_j já reconhecido por (1) ou (2), se em algum ponto de Q uma sequência de transferência T aplicada sobre s_j leva ao estado s_i (reconhecido por (2)), então, toda aplicação de T sobre s_j em Q leva a s_i.

Uma transição $(s_i, x) \in D_M$ é verificada em Q se:

a. Os estados s_i e $s_j = \delta(s_i, x)$ são reconhecidos por (1) em Q; e

b. Todas as sequências de caracterização de W são aplicadas sobre s_j em Q de forma que s_j seja alcançado pela transição (s_i, x).

Em seu trabalho, Rezaki e Ural (1995) definem condições de suficiência que comprovam a completude das sequências de verificação geradas pelo método. Tais condições estão

inseridas em um teorema proposto, que determina que, se todas as arestas (transições) de G são verificadas em Q, então Q é uma sequência de verificação de M. Dessa forma, o método proposto por Rezaki e Ural (1995) visa a encontrar uma sequência Q que conduz o menor caminho em G, de maneira que toda aresta de G seja verificada em Q. Embora as condições de suficiência propostas são semelhantes às condições propostas em Ural et al. (1997), o reconhecimento dos estados e verificação das transições no método de Rezaki e Ural (1995) é baseado nas sequências de localização.

O procedimento empregado na geração das sequências de verificação segue a mesma estratégia empregada no método de Ural et al. (1997). A partir do grafo $G = (V, E)$, que representa a MEF M sob teste, é construído um grafo auxiliar $G' = (V', E')$ contendo, além dos vértices e arestas de G, um conjunto de arestas e vértices que permitem solucionar o problema de geração de sequências de verificação por meio da resolução do problema RCP (Aho et al., 1991). Como discutido na Seção 3.6.2, o problema RCP consiste na busca do menor caminho possível em um grafo, de maneira que um conjunto de arestas, previamente estabelecido, seja, obrigatoriamente, coberto pelo caminho encontrado. Assim, por meio do grafo G' gerado, busca-se encontrar o menor caminho, partindo do estado inicial de G, que cubra o subconjunto de arestas composto pelas sequências de entrada que tornam os estados e transições de G, respectivamente, reconhecidos e verificadas. Em resumo, para geração do grafo $G' = (V', E')$, o conjunto de vértices V' é composto pelos subconjuntos V e U', onde $U' = \{v'_i : \text{para cada } v_i \in V\}$ e o conjunto de arestas E' é composto pelos conjuntos E_α, E_c, E_ϵ e E''. Na seção anterior, por definição de sequência de localização (Definição 3.1), a sequência de transferência I_i^r associada à última sequência de caracterização de W pode ou não ser vazia. No método de Hennie (1964), as sequências de localização geradas possuem I_i^r vazia, conforme apresentado na Tabela 3.4. No método de Rezaki e Ural (1995), contudo, I_i^r é não vazia e, portanto, tem como intuito transferir a execução de volta para o estado s_i. A Tabela 3.5 apresenta as sequências de localização para a MEF da Figura 3.2, adaptadas ao método de Rezaki e Ural (1995).

Estado	L_i	$\lambda(s_i, L_i)$	*Estado relacionado a* L_i
1	$L_1 = (ab)^4bb$	$(00)^410$	$\delta(s_1, L_1 \backslash bb) = \delta(s_1, abababab)$
2	$L_2 = (aa)^4ba$	$(10)^400$	$\delta(s_2, L_2 \backslash ba) = \delta(s_2, aaaaaaaa)$
3	$L_3 = (aa)^4bb$	$(01)^401$	$\delta(s_3, L_3 \backslash bb) = \delta(s_3, aaaaaaaa)$

Tabela 3.5: Sequências de localização para a MEF apresentada na Figura 3.2 (Método de Rezaki e Ural (1995)).

As arestas contidas em E_α constituem as sequências que reconhecem cada estado de G. Para cada vértice v_k, $1 \leq k \leq n$, é gerada uma sequência α_k que tem como intuito reconhecer o estado $\delta(v_k, L_k)$. Assim, dado um conjunto $W = \{w_1, w_2, ..., w_r\}$,

$\alpha_k = L_k w_1 I_k^1 L_k w_2 I_k^2 ... I_k^{r-1} L_k w_r I_k^r L_k$, onde I_k^l, $1 \leq l \leq r$, correspondem à sequências de transferência do estado $\delta(v_k, L_k w_l)$ de volta ao vértice v_k. É possível observar que todas as sequências de caracterização são aplicadas ao estado $\delta(v_k, L_k)$ por meio de α_k, o que permite o reconhecimento do estado $\delta(v_k, \alpha_k)$. Para a MEF da Figura 3.2, considerando o conjunto $W = \{a, b\}$ e as sequências de localização da Tabela 3.5, é gerado o conjunto de sequências $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$ onde $\alpha_1 = L_1 w_1 b L_1 w_2 b L_1$, $\alpha_2 = L_2 w_1 a L_2 w_2 a L_2$, e $\alpha_3 = L_3 w_1 a L_3 w_2 b L_3$. A partir do conjunto α é então gerado o conjunto de arestas $E_\alpha = \{(v_k, (\delta(v_k, \alpha_k))'; \alpha_k / \lambda(v_1, \alpha_k)) : \text{para cada } \alpha_k,\ 1 \leq k \leq 3\}$.

O conjunto de arestas E_c representam as sequências de entrada que verificam cada transição de G. Para isso, cada sequência de caracterização deve ser aplicada sobre o estado final de cada transição. Além disso, para manter a sequência de verificação em um estado conhecido, aplica-se a sequência de localização referente ao estado alcançado após a aplicação da sequência de caracterização. Formalmente, $E_c = \{(v'_k, (\delta(v_k, x w_\tau L_{\delta(v_k, x w_\tau)}))'; (x w_\tau L_{\delta(v_k, x w_\tau)}) / \lambda(v_k, x w_\tau L_{\delta(v_k, x w_\tau)})), 1 \leq \tau \leq r : \text{para cada } (v_k, v_l; x/y) \in E, 1 \leq k \leq n, 1 \leq l \leq n\}$. A Tabela 3.6 apresenta as sequências que compõem o conjunto E_c, onde β_{i,x,w_τ} representa uma sequência de E_c na qual a transição (s_i, x) é reconhecida pela sequência de transferência $w_\tau \in W$.

Transição	β_{i,x,w_1}	β_{i,x,w_2}
(s_1, a)	$\beta_{1,a,a} = aaL_3$	$\beta_{1,a,b} = abL_1$
(s_1, b)	$\beta_{1,b,a} = baL_2$	$\beta_{1,b,b} = bbL_1$
(s_2, a)	$\beta_{2,a,a} = aaL_2$	$\beta_{2,a,b} = abL_1$
(s_2, b)	$\beta_{2,b,a} = baL_2$	$\beta_{2,b,b} = bbL_3$
(s_3, a)	$\beta_{3,a,a} = aaL_3$	$\beta_{3,a,b} = abL_1$
(s_3, b)	$\beta_{3,b,a} = baL_2$	$\beta_{3,b,b} = bbL_3$

Tabela 3.6: Sequências do conjunto E_c para a MEF apresentada na Figura 3.2.

Os conjuntos E_ϵ e E'' constituem arestas auxiliares em G', onde $E_\epsilon = \{(v'_i, v_i; \epsilon) : \text{para cada } v_i \in V\}$ e $E'' = \{(v'_i, v'_j; x/y) : \text{para cada } (v_i, v_j; x/y) \in E\}$. Após construir o grafo G', contendo todos os vértices e arestas definidos acima, aplica-se o algoritmo de resolução do problema RCP. Assim, deve ser encontrado o menor caminho em G' que, obrigatoriamente, contenha todas as arestas de $E_\alpha \cup E_c$.

Para a MEF da Figura 3.2, o menor caminho encontrado produz uma sequência de verificação de tamanho 248. É importante observar que o método de Rezaki e Ural (1995) constitui um modelo geral e tem como objetivo introduzir a estratégia de resolução por meio de grafos, ignorando otimizações adicionais. Essa estratégia permite a redução das sequências geradas, uma vez que possibilita uma otimização global em relação à ordem em que os estados e transições são verificados. O método de Hennie (1964) aborda uma estratégia de construção local, o que pode acarretar na utilização de um maior número de

sequências de transferência. Em contrapartida, adota otimizações adicionais que visam a reduzir o tamanho das sequências geradas. Para esse exemplo, o método de Hennie (1964) apresentou uma redução de 77 símbolos de entrada em relação ao método de Rezaki e Ural (1995).

3.7 Considerações Finais

Neste capítulo foram apresentados os principais conceitos que fundamentam o teste baseado em MEFs. Foram introduzidos e exemplificados os métodos de Gonenc (1970), Ural et al. (1997) e Simao e Petrenko (2008), além de uma breve discussão a respeito dos métodos de Hierons e Ural (2002), Hierons e Ural (2006) e Chen et al. (2005), derivados do método de Ural et al. (1997). Além desses métodos, foram apresentados os métodos de Hennie (1964) e de Rezaki e Ural (1995). Esses dois últimos métodos são baseados na utilização de conjuntos de caracterização e, portanto, estão inseridos no contexto específico deste trabalho.

O método de Gonenc (1970) apresenta uma abordagem baseada na geração e concatenação de duas sequências, denominadas de sequências α e β, que buscam verificar, respectivamente, os estados e transições da MEF. Em Ural et al. (1997), são definidas condições de suficiência que garantem que as sequências geradas são sequências de verificação. Baseado nessas condições, no método é construído um grafo auxiliar contendo um subconjunto de arestas que contém sequências equivalentes às sequências α e β. Dessa forma, tais arestas devem ser cobertas pelo menor caminho gerado com base na resolução do problema RCP (Aho et al., 1991). Os métodos citados acima são baseados em sequências de distinção. Esse tipo de sequência provê uma maneira eficiente na identificação dos estados de uma MEF, porém, não existe para toda MEF minimal. Esse fator limita o domínio de aplicação de tais métodos.

O método de Simao e Petrenko (2008), diferente dos demais métodos, utiliza uma estratégia de busca local baseada em conjuntos de distinção. Esse tipo de conjunto permite um domínio de aplicação mais abrangente em relação aos métodos baseados em sequências de distinção, apesar de também não existir para toda MEF minimal. Com base no exemplo gerado, esse método apresentou uma sequência de verificação reduzida em relação aos demais métodos.

O método de Hennie (1964) introduz a estratégia de geração de sequências de verificação baseada em sequências de localização. Esse tipo de sequência é construída a partir de conjuntos de caracterização, o que torna o método mais abrangente em relação aos métodos anteriores. Essa característica se deve ao fato de que toda MEF minimal possui conjunto de caracterização. Esse método utiliza uma estratégia de busca local na geração

de sequências de verificação. Além disso, otimizações são aplicadas a fim de reduzir a sequência gerada. O método de Rezaki e Ural (1995) utiliza uma estratégia de geração baseada em grafos e também é baseado em sequências de localização. Essa estratégia tem como intuito reduzir as sequências geradas por meio da resolução do problema RCP (Aho et al., 1991).

Apesar de mais abrangentes, ambos os métodos baseados em conjuntos de caracterização produzem sequências de verificação com tamanho exponencialmente longo em relação ao número de sequências no conjunto utilizado. Esse fator, muitas vezes, torna impraticável a execução da sequência de teste gerada. Nesse contexto, o próximo capítulo apresenta um novo método para geração de sequências de verificação que visa a reduzir o tamanho das sequências geradas no contexto de MEFs que não dispõem de sequências de distinção. Além disso, é apresentada uma avaliação experimental com o intuito de mensurar a redução proporcionada pelo método proposto em relação aos métodos já existentes.

Capítulo

4

Estratégia para Geração de Sequências de Verificação

4.1 Considerações Iniciais

Métodos para a geração de sequências de verificação são importantes no contexto de teste baseado em MEFs, uma vez que abrangem o domínio de MEFs que não dispõem de operações *reset*. Essa característica incentivou a elaboração de diferentes métodos existentes na literatura, conforme apresentado no capítulo anterior. A maioria desses métodos, no entanto, são baseados em uma sequência especial denominada de sequência de distinção (DS). Esse tipo de sequência não existe para toda MEF minimal, o que incentiva a pesquisa de métodos baseados em recursos alternativos às sequências de distinção.

Os métodos de Hennie (1964) e Rezaki e Ural (1995) (Seções 3.6.4 e 3.6.5) são baseados nos denominados conjuntos de caracterização, os quais existem para toda MEF minimal. Tais métodos, no entanto, geram sequências de verificação exponencialmente longas, acarretando em um alto custo de execução do teste. Dessa forma, neste capítulo é proposto um novo método para geração de sequências de verificação que tem como intuito reduzir o tamanho das sequências geradas mantendo a abrangência proporcionada pelos métodos baseados em conjuntos de caracterização. Além disso, neste capítulo, é apresentada uma avaliação experimental a fim de investigar as vantagens e desvantagens do método proposto em relação aos métodos de Hennie (1964) e Rezaki e Ural (1995).

Nos experimentos conduzidos analisou-se a redução proporcionada pelo método proposto no tamanho das sequências geradas. As análises foram conduzidas por meio da variação de três fatores principais: número de estados, número de entradas e número de saídas das MEFs geradas. Além disso, foram analisadas a proporção de MEFs que não dispõem de DS e o número médio de sequências que constituem o conjunto W, em relação a cada um dos três fatores citados.

4.2 Geração de Sequências de Verificação Utilizando Conjuntos de Identificação

Nesta seção é proposto um novo método de geração que tem como intuito reduzir o tamanho das sequências de verificação geradas a partir de conjuntos W. Para a aplicação do método proposto, assume-se que M é uma MEF minimal, completa e fortemente conexa. Além disso, assume-se que uma implementação a ser testada pode ser modelada por uma MEF $I \in \Im(M)$. A redução proporcionada pelo método proposto é baseada no fato de que, em certos casos, a identificação de um estado pode ser realizada somente por um subconjunto de W, ao invés de utilizar todo o conjunto W para verificar cada estado de M.

Definição 4.1. Dado um conjunto de sequências de entrada $W_i = \{w_1^i, w_2^i, ..., w_p^i\} \subseteq W$, W_i é um *conjunto de identificação* para $s_i \in S$ se e somente se para cada estado $s_j \in S$, $i \neq j$, existe uma sequência de entrada $\alpha \in W_i$ tal que $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$ e nenhum outro subconjunto de W_i possui essa mesma propriedade (Fujiwara et al., 1991).

O uso de conjuntos de identificação, ao invés do conjunto W, pode prover algumas otimizações que influenciam na redução do tamanho das sequências de verificação geradas. A principal otimização está na redução das sequências de localização. O tamanho de uma sequência de localização é exponencialmente proporcional ao número de sequências de caracterização em W (Hennie, 1964). Como consequência direta, a redução do número de sequências de caracterização utilizadas na construção das sequências de localização pode acarretar em reduções significativas no tamanho das sequências de verificação geradas. Por consequência, a redução das sequências de localização acarretam em uma redução global proporcional ao domínio de M. Isto se deve ao fato de que o reconhecimento dos estados, bem como a verificação das transições de M, dependem da aplicação de sequências de localização.

Enquanto os métodos de Rezaki e Ural (1995) e Hennie (1964) são baseados em sequências de localização, o método proposto é baseado em um novo tipo de sequência, denomi-

nada de *sequência de reconhecimento* (*recognizing sequence*). Seja $W_i = \{w_1^i, w_2^i, ..., w_p^i\}$ um conjunto de identificação para o estado $s_i \in S$ onde w_τ^i corresponde a uma sequência de caracterização pertencente a W_i, $1 \leq \tau \leq p$. Seja também S_k^i, $1 \leq k \leq p-1$, uma sequência de entrada contendo o prefixo w_k^i e cuja aplicação leve a execução de M do estado s_i de volta para o estado s_i, ou seja, $S_k^i = w_k^i I_i^k$ onde I_i^k é uma sequência de transferência de $\delta(s_i, w_k^i)$ para s_i. Além disso, considere S_p^i uma sequência de entrada com o prefixo w_p^i que não necessariamente traga a execução de M do estado s_i de volta ao estado s_i, ou seja, $S_p^i = w_p^i I_i^p$, onde, se I_i^p não é uma sequência vazia, então I_i^p é uma sequência de transferência de $\delta(s_i, w_p^i)$ para s_i. Para o método proposto nesta seção, I_i^p constitui uma sequência de transferência não vazia, com exceção do caso em que $\delta(s_i, w_p^i) = s_i$.

Definição 4.2. Uma *sequência de reconhecimento* para o estado s_i, denotada por P_i, é a sequência $P_i = F'(p, \{S_1^i, S_2^i, ..., S_p^i\})$ onde $F' : \mathbb{N} \times 2^{\Omega(s_i)} \rightarrow \Omega(s_i)$ é uma função recursiva tal que $F'(1, \{a_1\}) = a_1$, $F'(2, \{a_1, a_2\}) = (a_1)^{n+1}.a_2$, $F'(k, \{a_1, a_2, ..., a_k\}) = (F'(k-1, \{a_1, a_2, ..., a_{k-1}\}))^{n+1}$. $F'(k-1, \{a_1, a_2, ..., a_{k-2}, a_k\})$, $3 \leq k \leq r$, e a_i, $1 \leq i \leq k$ é uma sequência de entrada para M.

O procedimento definido acima é similar ao procedimento definido para uma sequência de localização (Definição 3.1, Seção 3.6.4), exceto por dois fatores. Primeiro, a sequência é derivada de um conjunto de identificação, ao invés de um conjunto W. Segundo, a função F' trata o caso em que o conjunto de identificação é formado apenas por uma sequência.

Seja $N = (Q, q_0, I, O, \Delta, \Lambda)$ uma possível implementação de M pertencente a $\Im(M)$ e $W = \{w_1, w_2, ..., w_r\}$ um conjunto de caracterização de M. Os estados de M são *identificáveis* em N se para cada estado $s \in S$ existe um estado $q \in Q$ tal que $\lambda(s, w_\tau) = \Lambda(q, w_\tau)$, $1 \leq \tau \leq r$. O conceito definido acima, de estados *identificáveis*, é importante para o entendimento da propriedade que cerca uma sequência de reconhecimento, explanada a seguir.

Uma sequência de reconhecimento P_i para o estado s_i tem a propriedade de que todo o conjunto de sequências de entrada $W_i = \{w_1^i, w_2^i, ..., w_p^i\}$ é aplicado sobre o estado $\delta(s_i, P_i \backslash S_p^i)$. Essa propriedade permite a uma sequência de reconhecimento P_i a localização de um estado $q_i \in Q$ em N cujo comportamento é equivalente a s_i, em relação à W_i. Contudo, essa propriedade não garante que exista apenas um estado em N cujo comportamento seja equivalente a s_i, em relação à W_i. A localização de q_i em N com a garantia de que q_i é único torna-se possível a partir da suposição de que todos os estados de M são identificáveis em N. O método proposto satisfaz a essa suposição na primeira etapa de seu procedimento por meio da aplicação de todo o conjunto W sobre cada estado de M. Como consequência, no método proposto, as sequências de reconhecimento são utilizadas

no lugar das sequências de localização no processo de construção da sequência de verificação. Além disso, ao considerar essa mesma suposição, o estado final de uma transição pode ser verificado por meio da aplicação de seu conjunto de identificação correspondente, ao invés de todo o conjunto W.

4.2.1 Algoritmo

A ideia básica do algoritmo proposto é a construção de uma sequência de verificação por meio da concatenação de dois grupos distintos de sequências. O primeiro grupo tem como intuito reconhecer cada estado de M. O segundo, por sua vez, tem como intuito verificar todas as transições de M. Dessa forma, o método proposto produz uma sequência de verificação na qual cada transição de M é verificada de acordo com o teorema proposto no trabalho de Rezaki e Ural (1995). Assume-se que tanto a especificação quanto a implementação estão inicializadas e, portanto, nenhum recurso (por exemplo, sequência de sincronização) é utilizado para a inicialização do teste.

O método é composto por três etapas principais. Na primeira etapa, para cada estado de s_i é gerada uma sequência α_i na qual todas as sequências de caracterização de W são aplicadas sobre o estado final alcançado por P_i, onde $\delta(s_i, P_i)$ corresponde ao próprio s_i. Dada uma implementação $N \in \Im(M)$, o conjunto de sequências α_i garante que todos os estados de M são identificáveis em N. Além disso, ambos os estados $\delta(s_i, P_i)$ e $\delta(s_i, P_i \backslash S_p^i)$ são reconhecidos em N como o estado s_i de M. Em seguida, a sequência P_i é concatenada ao final de α_i no intuito de manter a sequência gerada em um estado conhecido.

Na segunda etapa, cada transição $(s_i, x) \in D_M$ é verificada por meio da aplicação do conjunto de identificação W_k sobre seu estado final $s_k = \delta(s_i, x)$. A verificação do estado final de uma transição pode ser realizada por meio da aplicação do conjunto de identificação correspondente (ao invés de todo o conjunto W), se todos os estados de M são identificáveis em N; essa condição é satisfeita pelo procedimento realizado na primeira etapa do método. Novamente, com o intuito de manter a sequência gerada em um estado conhecido, a aplicação de cada sequência $w_\tau^k \in W_k$, $1 \leq \tau \leq p$, é seguida pela aplicação da sequência P_q correspondente, onde $s_q = \delta(s_k, w_\tau^k)$. O reconhecimento do estado alcançado por P_q é garantido na primeira etapa do método, na qual cada sequência de W é aplicada sobre $s_q = \delta(s_q, P_q)$.

Por fim, na terceira etapa do método, as sequências de entrada geradas nas etapas anteriores são concatenadas por meio de sequências de transferência. O método proposto é formalmente definido no Algoritmo 1.

Algoritmo 1 Método para geração de sequências de verificação.

Entrada: MEF minimal $M = (S, s_0, I, O, D, \delta, \lambda)$; Conjunto de caracterização W de M;
Conjunto de identificação W_i para cada $s_i \in S$; P_i para cada $s_i \in S$;
Saída: Sequência de verificação ω para M;
//Etapa 1
$A \leftarrow \emptyset$
foreach $s_i \in S$ **do**
    $\alpha_i \leftarrow P_i w_1 I_i^1 P_i w_2 I_i^2 ... I_i^{r-1} P_i w_r I_i^r P_i$, onde I_i^l, $1 \leq l \leq r$, são sequências de transferência entre o estado $\delta(s_i, P_i w_l)$ e o estado s_i.
    $A \leftarrow A \cup \{\langle s_i, \alpha_i \rangle\}$.
endforeach
//Etapa 2
$B \leftarrow \emptyset$
foreach $(s_i, x) \in D$ **do**
    $s_k \leftarrow \delta(s_i, x)$
    foreach $w_\tau^k \in W_k$ **do**
        $\beta_{i,x,w_\tau^k} \leftarrow x w_\tau^k P_{\delta(s_k, w_\tau^k)}$
        $B \leftarrow B \cup \{\langle s_i, \beta_{i,x,w_\tau^k} \rangle\}$
    endforeach
endforeach
//Etapa 3
$\omega \leftarrow \epsilon$
$C \leftarrow A \cup B$
while $C \neq \emptyset$ **do**
    if existe $\langle s, \gamma \rangle \in C$ tal que $s = \delta(s_0, \omega)$ **do**
        $\omega \leftarrow \omega\gamma$
        $C \leftarrow C \backslash \{\langle s, \gamma \rangle\}$
    else
        Determine a menor sequência de transferência θ entre o estado $\delta(s_0, \omega)$ e algum outro estado $s \in S$, tal que existe $\langle s, \gamma \rangle \in C$.
        $\omega \leftarrow \omega\theta$
    endif
endwhile
return ω

4.2.2 Exemplo

Nesta seção, o método proposto é ilustrado para a MEF M da Figura 3.2. É importante observar que essa MEF não dispõe de sequência de distinção. Para exemplificação do método, é considerado o conjunto de caracterização $W = \{w_1, w_2\} = \{a, b\}$. Na Tabela 4.1 são apresentados os conjuntos de distinção (W_i), as sequências de reconhecimento (P_i), e as saídas produzidas por P_i para cada estado $s_i \in S$.

Estado	W_i	P_i	$P_i/\lambda(s_i, P_i)$
s_1	$W_1 = \{b\}$	$P_1 = bb$	$P_1/10$
s_2	$W_2 = \{a\}$	$P_2 = aa$	$P_2/10$
s_3	$W_3 = \{a, b\}$	$P_3 = (aa)^4bb$	$P_3/(01)^401$

Tabela 4.1: Conjuntos de identificação e sequências de reconhecimento para os estados de M.

Na primeira etapa do método proposto é gerado o conjunto de sequências $A = \{\alpha_1, \alpha_2, \alpha_3\}$ onde $\alpha_1 = P_1w_1bP_1w_2bP_1$, $\alpha_2 = P_2w_1aP_2w_2aP_2$, e $\alpha_3 = P_3w_1aP_3w_2bP_3$. Esse conjunto de sequências garante o reconhecimento de todos os estados de M, uma vez que o conjunto W é aplicado sobre cada estado de M. Além disso, o procedimento aplicado na geração dessas sequências também torna reconhecido o estado final alcançado por P_i.

As sequências que compõem o conjunto B representam as sequências de entrada que verificam cada transição $(s, x) \in D_M$. Essas sequências são geradas na segunda etapa do Algoritmo 1 e são apresentadas na Tabela 4.2. Como já mencionado na descrição da segunda etapa, a verificação do estado final de cada transição é realizada por meio de conjuntos de distinção, ao invés de todo o conjunto W. Dessa forma, algumas sequências de caracterização presentes em W não são aplicadas para todo estado. Essas sequências poupadas são representadas na Tabela 4.2 por meio do caractere "-" . Por exemplo, para a transição (s_1, a), o estado final alcançado $s_2 = \delta(s_1, a)$ é reconhecido por meio da aplicação do conjunto de identificação $W_2 = \{a\}$ ao invés de todo o conjunto $W = \{a, b\}$, tornando desnecessária a aplicação da sequência β para $w_2 = b$. Em seguida, é aplicada a sequência de reconhecimento $P_{\delta(s_1,aa)} = P_3$ a fim de trazer a execução do teste a um estado conhecido. É importante notar que as células preenchidas com "-" acarretam em otimizações na sequência de verificação gerada.

Por fim, as sequências que compõem os conjuntos A e B são concatenados. Quando necessário, é selecionada a menor sequência de transferência possível que leve a execução da MEF ao estado apropriado. Assim, ao final do procedimento proposto, é gerada a sequência de verificação $\omega = \alpha_1\beta_{1,a,a}\alpha_3\beta_{3,a,a}\beta_{3,b,b}a\alpha_2\beta_{2,a,a}\beta_{2,a,b}\beta_{1,b,a}\beta_{2,b,b}b\beta_{1,b,b}$ com tamanho $|\omega| = 120$. Para a mesma MEF, conforme apresentado no capítulo anterior, os

Transição	β_{i,x,w_1}	β_{i,x,w_2}
(s_1, a)	$\beta_{1,a,a} = aaP_3$	$-$
(s_1, b)	$\beta_{1,b,a} = baP_2$	$\beta_{1,b,b} = bbP_1$
(s_2, a)	$\beta_{2,a,a} = aaP_2$	$\beta_{2,a,b} = abP_1$
(s_2, b)	$-$	$\beta_{2,b,b} = bbP_3$
(s_3, a)	$\beta_{3,a,a} = aaP_3$	$-$
(s_3, b)	$-$	$\beta_{3,b,b} = bbP_3$

Tabela 4.2: Conjunto B gerado na Etapa 2.

métodos propostos por Hennie (1964) e Rezaki e Ural (1995) geraram sequências de verificação com tamanho $|\omega_h| = 171$ e $|\omega_r| = 248$, respectivamente.

4.3 Resultados Experimentais

Os experimentos conduzidos são baseados em MEFs geradas de maneira aleatória. Os métodos comparados nos experimentos foram elaborados para o contexto de MEFs que não dispõem de sequências de distinção. Dessa forma, apenas MEFs com essa característica foram consideradas. Foram geradas MEFs desprovidas de DS, completas, fortemente conexas e reduzidas, conforme o procedimento a seguir (Simao e Petrenko, 2008). Primeiramente, foram criados os conjuntos de estados, entradas e saídas com tamanhos previamente estabelecidos. A geração de MEFs é então realizada em 3 etapas. Na primeira etapa, um estado é selecionado como estado inicial e então marcado como "estado alcançado". Em seguida, para cada estado s não marcado como "estado alcançado", aplica-se o seguinte procedimento: 1) seleciona-se aleatoriamente um estado s' já marcado como "estado alcançado", uma entrada x, e uma saída y; 2) adiciona-se uma transição de s' para s com a entrada x e a saída y previamente selecionadas; e 3) s é marcado como "estado alcançado". Após essa etapa, obtém-se uma MEF inicialmente conexa. Na segunda etapa, são adicionadas transições (dois estados, uma entrada e uma saída são selecionados aleatoriamente) até que seja obtida uma MEF completa. Caso a MEF gerada não seja fortemente conexa, ela é descartada e uma nova MEF é gerada. Caso contrário, na terceira etapa, busca-se uma sequência de distinção para a MEF gerada. Se a MEF possui sequência de distinção, ela é descartada e uma nova MEF é gerada.

As próximas seções apresentam os resultados experimentais obtidos sob diferentes perspectivas em relação ao tamanho das sequências geradas por cada método. A avaliação é realizada considerando-se a variação de três fatores: número de estados, número de entradas e número de saídas. Para cada fator analisado, o número de elementos varia em um intervalo de valores pré-determinados enquanto os valores referentes aos demais fatores permanecem fixos. Essas avaliações têm como intuito trazer uma visão geral do desempenho do método proposto em relação aos métodos existentes. Além disso,

analisa-se também o tamanho dos conjuntos W e a concentração de MEFs que não dispõem de DS no domínio de MEFs geradas.

Para cada MEF considerada nos experimentos, são executados o método proposto (Algoritmo 1), o método de Hennie (1964) e o método de Rezaki e Ural (1995), produzindo, respectivamente, as sequências ω, ω_h e ω_r. A notação $\mu(\omega)$ representa o tamanho médio das sequências de verificação geradas pelo método proposto. Essa notação é estendida para as sequências ω_h e ω_r, representadas, respectivamente, por $\mu(\omega_h)$ e $\mu(\omega_r)$. As relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$ indicam a taxa de redução média obtida pelo método proposto em relação aos demais métodos. Esses dados são discorridos por meio de gráficos em função dos fatores considerados: número de estados, número de entradas ou número de saídas. De maneira complementar, para cada fator considerado, também são analisadas as relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$, que representam a taxa de redução proporcionada pelo método proposto, em relação aos métodos existentes, para cada MEF gerada. Esses dados são discorridos por meio de gráficos boxplot.

4.3.1 Número de Estados

Os experimentos apresentados nesta seção têm como intuito elucidar o desempenho do método proposto considerando a variação do número de estados. Para isso, foram geradas MEFs com duas entradas e duas saídas, enquanto o número de estados n varia no intervalo entre três e 12 estados; para cada valor de n, foram geradas 100 máquinas (1000 MEFs, no total).

Como já explanado, apenas MEFs que não dispõem de DS foram consideradas no experimento. No entanto, ao relacionar a quantidade de MEFs descartadas (que possuem DS) com as 100 MEFs consideradas (que não possuem DS) obtém-se a proporção existente para cada tipo de MEF no domínio de MEFs geradas aleatoriamente. Essa informação é importante no contexto deste trabalho, pois apresenta uma visão geral a cerca da abrangência do método proposto em relação aos métodos baseados em DS. Na Figura 4.1 é apresentada a proporção de MEFs sem DS no domínio total das MEFs geradas. É possível observar que a proporção de MEFs sem DS aumenta conforme o número de estados. Acima de 10 estados, MEFs sem DS correspondem a mais de 50% das MEFs geradas.

Na Figura 4.2 é apresentada a quantidade média de sequências que compõem o conjunto W para cada valor de n. É possível observar que a quantidade média de sequências em W aumenta proporcionalmente ao número de estados. Uma vez que os métodos comparados são baseados no conjunto W e o método proposto é baseado em conjuntos de distinção (que também são derivados a partir de W), o aumento no tamanho desse conjunto influencia diretamente no tamanho das sequências de verificação geradas. A

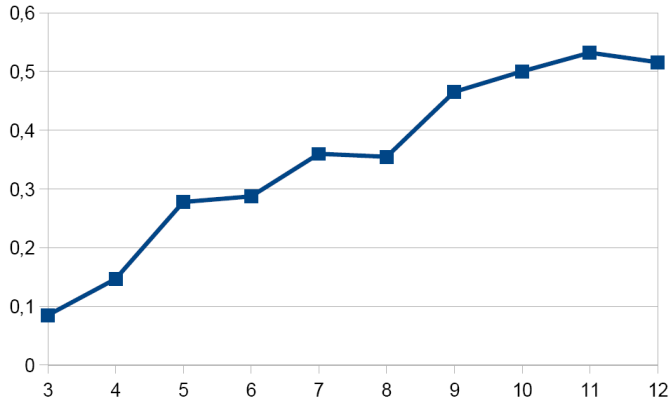

Figura 4.1: Proporção de MEFs sem DS no domínio total de MEFs geradas, em relação ao número de estados.

influência do conjunto W sobre o tamanho das sequências geradas pode ser observada no gráfico da Figura 4.3. Nessa figura é apresentado o tamanho médio das sequências de verificação (em escala logarítmica) geradas para cada método analisado. É possível observar que o aumento no número de estados aumenta significativamente a diferença entre a escala em que se encontra o método proposto e a escala em que se encontram os métodos de Hennie (1964) e de Rezaki e Ural (1995). Para 12 estados, o tamanho médio das sequências geradas pelos métodos de Hennie (1964) e de Rezaki e Ural (1995) é de 33.422.266 e 233.658.897, respectivamente, enquanto o tamanho médio obtido pelo método proposto é de 11.184.

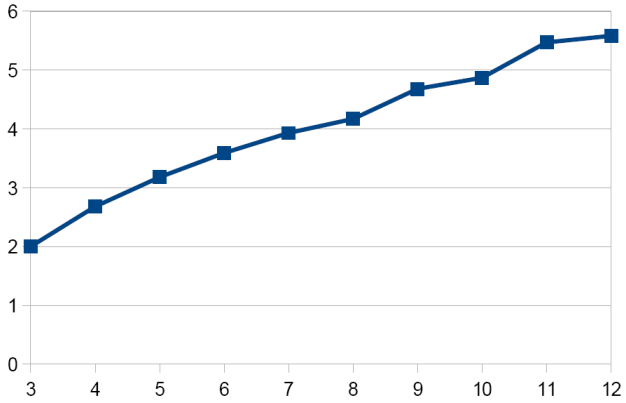

Figura 4.2: Número médio de sequências que compõem o conjunto W, em relação ao número de estados.

A redução obtida pelo método proposto é melhor visualizada na Figura 4.4. Nessa figura são apresentadas as relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$, que representam a taxa de redução do tamanho médio proporcionada pelo método proposto em relação aos métodos de Hennie (1964) e Rezaki e Ural (1995), respectivamente. Pode-se notar que o tamanho médio das sequências de verificação geradas pelo método proposto representa, em geral, apenas uma pequena fração do tamanho médio obtido pelos demais métodos. Com mais

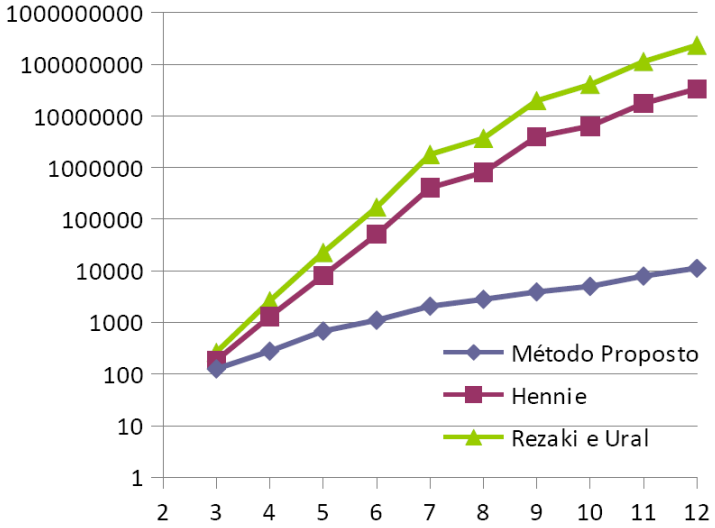

Figura 4.3: Tamanho médio das sequências de verificação geradas, em relação ao número de estados.

de sete estados, $\mu(\omega)$ representa menos que 1% de $\mu(\omega_h)$ e $\mu(\omega_r)$. Nessa figura, os dados são apresentados em escala logarítmica a fim de ilustrar a real magnitude dos resultados obtidos. Além disso, nessa figura, também é apresentada a relação $\mu(\omega_h/\omega_r)$ que representa a redução média obtida pelo método de Hennie (1964) em relação ao método de Rezaki e Ural (1995). Pode-se observar uma taxa de redução de até 85% em $\mu(\omega_h/\omega_r)$ para $n = 12$. Essa informação torna-se importante ao considerar que as otimizações adotadas no método de Hennie (1964) não são adotadas no método proposto. Consequentemente, se agregadas ao método proposto, tais otimizações podem acarretar em uma redução ainda maior.

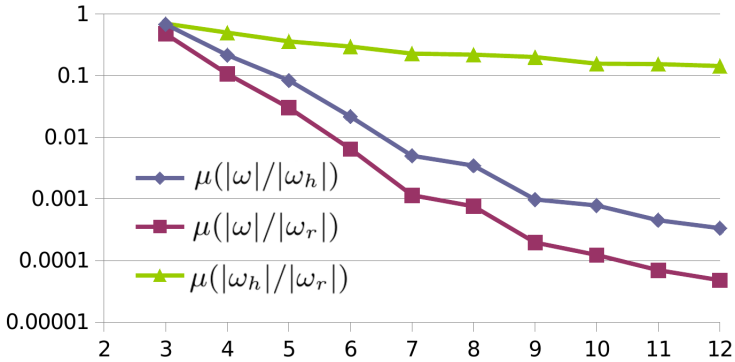

Figura 4.4: Relações $\mu(\omega/\omega_h)$, $\mu(\omega/\omega_r)$ e $\mu(\omega_h/\omega_r)$ apresentadas em escala logarítimica, em relação ao número de estados.

As informações discutidas acima, extraídas a partir do tamanho médio das sequências, trazem uma visão geral a cerca das otimizações obtidas. No entanto, o tamanho médio pode ser influenciado por valores excepcionais (*outliers*) existentes nos dados. Considerando essa limitação, também foram analisados os dados referentes à taxa de redução

obtida para cada MEF gerada, apresentados na Figura 4.5. Nessa figura, são apresentados os boxplots referentes às MEFs geradas com três, oito e 12 estados. Apesar de dispostos na mesma figura, os boxplots estão inseridos em escalas distintas. Ao observar os boxplots sob uma perspectiva geral, é possível notar maiores reduções no tamanho das sequências quanto maior o número de estados. No boxplot referente a três estados, a mediana indica uma redução de até 32% do método proposto em relação ao método de Hennie (1964). Em relação ao método de Rezaki e Ural (1995), o método proposto provê uma redução de até 53%, de acordo com o valor da mediana. Para oito estados, o boxplot apresentado indica uma redução ainda maior do método proposto em relação aos demais. Ao analisar o terceiro quartil, no qual estão representados 75% dos dados, o método proposto provê uma redução de até 88% em relação ao método de Hennie (1964) e de até 98% em relação ao método de Rezaki e Ural (1995). Ao analisar o boxplot referente a 12 estados, considerando o terceiro quartil, nota-se uma redução proporcionada pelo método proposto com valores acima de $99,5\%$ em relação a ambos os métodos comparados. No pior caso, quando os conjuntos de distinção são compostos por todas as sequências do conjunto W, nenhuma redução é atribuída ao método proposto. Contudo, esses casos constituem apenas uma pequena fração do conjunto de dados e, consequentemente, foram interpretados como *outliers* (representados por pequenos círculos acima do valor máximo de cada boxplot) na construção dos gráficos boxplot.

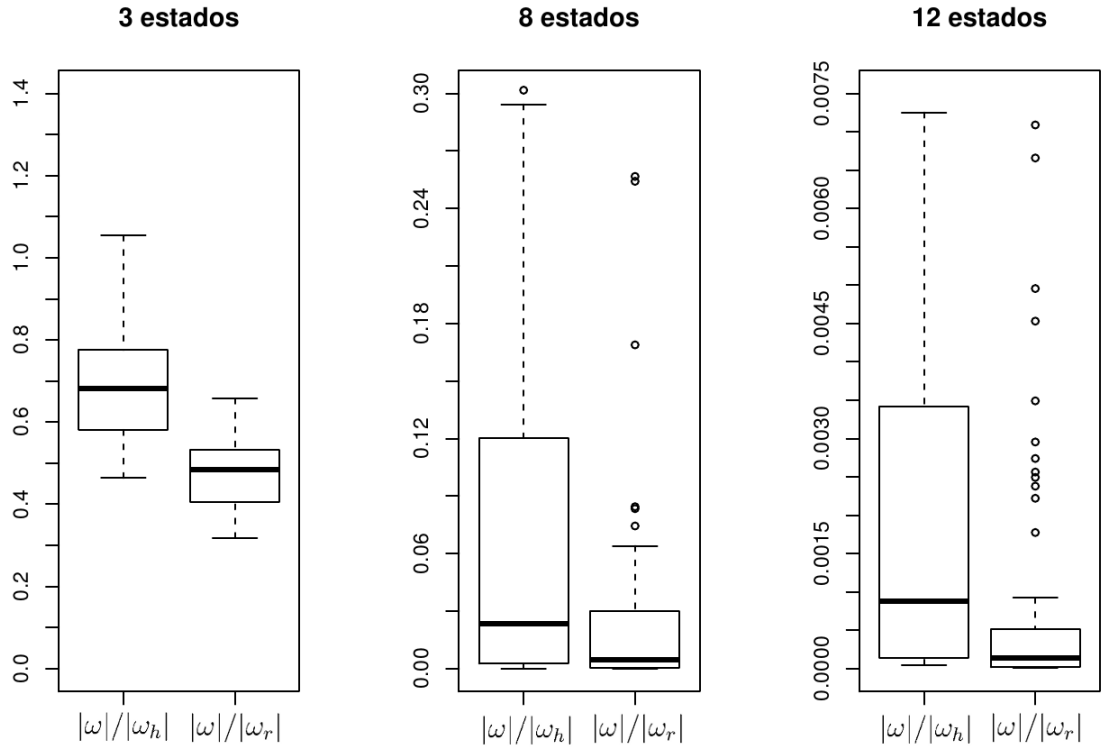

Figura 4.5: Boxplots para as relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$, para três, oito e 12 estados.

4.3.2 Número de Entradas

Nesta sessão, são discutidos os experimentos realizados sob a perspectiva da variação do número de entradas. Assim, a geração das MEFs consideradas no experimento possuem um número fixo de oito estados e quatro saídas. O intervalo de valores para o número de entradas varia entre duas e dez entradas. Para cada valor nesse intervalo são geradas 100 MEFs, totalizando uma amostragem de 900 MEFs.

Dentre o domínio total de MEFs geradas, a proporção de MEFs que não dispõem de sequência de distinção pode ser observada na Figura 4.6. É possível notar que a concentração de MEFs sem DS é inversamente proporcional ao número de entradas. Em outras palavras, quanto maior o número de entradas (considerando um valor fixo para o número de saídas) menor é a proporção de MEFs sem DS. A partir de cinco entradas, a proporção de MEFs desse tipo corresponde a menos de 10%, chegando a menos de 1% pra 10 entradas. Em termos práticos, para dez entradas, foram geradas 13702 MEFs para obtenção de 100 MEFs sem DS.

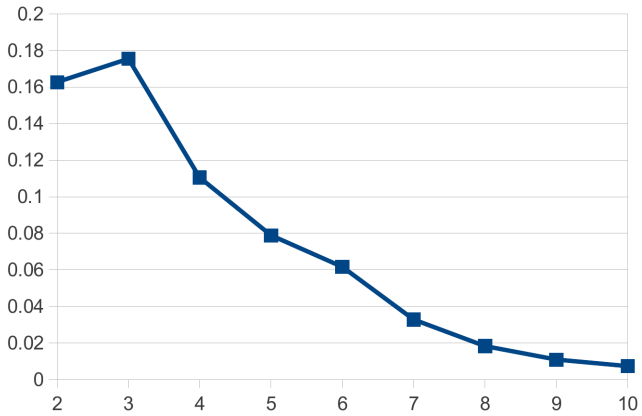

Figura 4.6: Proporção de MEFs sem DS no domínio total de MEFs geradas, em relação ao número de entradas.

Na Figura 4.7, é apresentada a quantidade média de sequências que compõem o conjunto W em relação ao número de entradas. Observa-se uma distribuição irregular no gráfico. Com a amostragem de dados analisada, não é possível observar uma forte influência do número de entradas sobre o número de sequências em W. Diante disso, é possível deduzir que o tamanho das sequências geradas pelos métodos comparados também não é fortemente influenciado pela variação do número de entradas. Essa dedução provém do fato de que, nos métodos comparados, o tamanho das sequências geradas é diretamente proporcional ao número de sequências em W. Na Figura 4.8, é possível observar que o aumento do número de entradas tem leve influência sobre o aumento do tamanho médio das sequências geradas. Ainda assim, há uma diferença significativa entre a escala que comporta o método proposto e a escala que comporta os demais métodos.

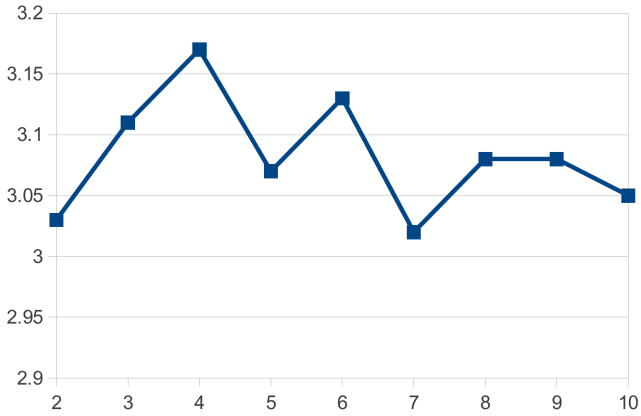

Figura 4.7: Número médio de sequências que compõem o conjunto W, em relação ao número de entradas.

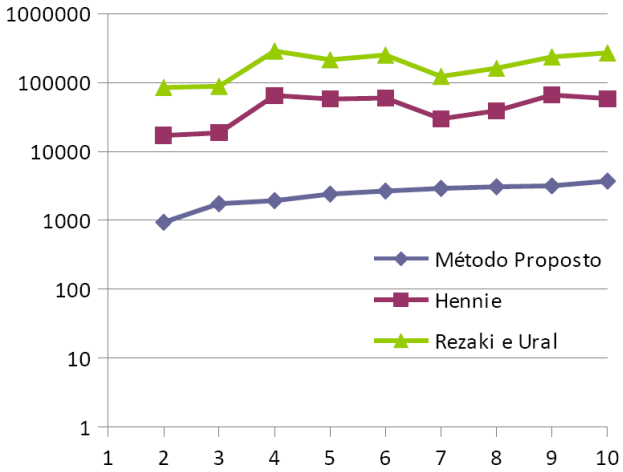

Figura 4.8: Tamanho médio das sequências de verificação geradas, em relação ao número de entradas.

As relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$ são apresentadas no gráfico da Figura 4.9. Essas relações representam a taxa de redução média provida pelo método proposto em relação aos métodos de Hennie (1964) e Rezaki e Ural (1995), respectivamente. É possivel notar uma redução média de, no mínimo, 90% do método proposto em relação ao método de Hennie (1964). Em relação ao método de Rezaki e Ural (1995), a redução obtida é ainda maior e corresponde a mais de 97% para todos os valores referentes ao número de entradas. Contudo, diferentemente da distribuição apresentada em relação ao número de estados (Figura 4.4), no gráfico da Figura 4.9 é apresentada uma distribuição pouco influenciada pela variação do número de entradas.

A taxa de redução média discutida acima mostra que, de uma maneira geral, o método proposto consegue reduzir o tamanho das sequências geradas de maneira significativa em relação aos métodos existentes. A Figura 4.10 apresenta os boxplots das relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$ referentes a duas, seis e dez entradas. Essas relações representam a redução provida pelo método proposto para cada MEF gerada. Dessa maneira, é possível fazer uma

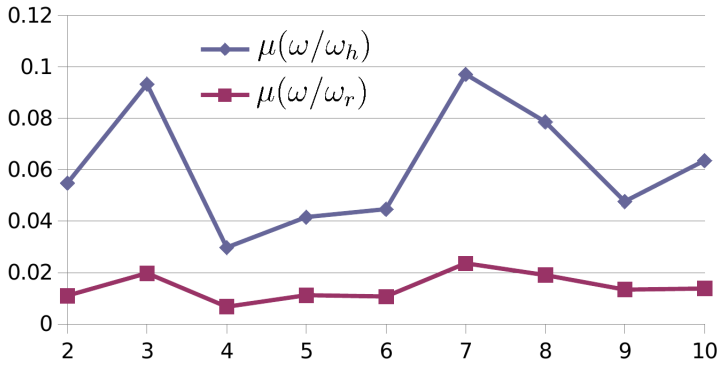

Figura 4.9: Relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$, que representam a taxa de redução média obtida em relação ao número de entradas.

análise mais consistente dos resultados, uma vez que possíveis *outliers* não interferem na análise dos dados. De maneira geral, ao analisar o terceiro quartil dos boxplots é possível observar uma menor redução para valores maiores do número de entradas. Em relação ao método de Hennie (1964), para duas, seis e dez entradas, o terceiro quartil apresenta reduções de 69%, 52% e 46%, respectivamente. Em relação ao método de Rezaki e Ural (1995), ao analisar o terceiro quartil, são apresentadas reduções de 94%, 88% e 86%, para duas, seis e dez entradas, respectivamente.

4.3.3 Número de Saídas

Nesta seção, os experimentos realizados visam a análise das reduções providas pelo método proposto considerando a variação do número de saídas. Dessa maneira, os valores referentes ao número de estados e número de entradas são fixados em oito e quatro, respectivamente. O intervalo de valores do número de saídas varia de duas a dez saídas. Novamente, para cada valor desse intervalo, são geradas 100 MEFs (900 MEFs, no total).

Na Figura 4.11 é apresentada a proporção de MEFs que não dispõem de DS em relação ao domínio total de MEFs geradas. Importante notar que apenas MEFs sem DS foram consideradas nos experimentos e que as demais MEFs, as quais possuem DS, foram descartadas. Dessa forma, a análise desse tipo de informação elucida o quão abrangente é o domínio de MEFs para aplicação do método proposto. No gráfico da Figura 4.11 observa-se um decaimento na concentração de MEFs sem DS conforme aumenta o número de saídas. Com quatro saídas (o mesmo número de entradas) a proporção de MEFs sem DS é de 11% do total de MEFs geradas. Para dez saídas, contudo, a proporção diminui para menos de 1%. Para obtenção das 100 MEFs almejadas, com dez saídas, foram geradas 15169 MEFs.

Na Figura 4.12 é apresentada a relação entre a quantidade média de sequências pertencentes ao conjunto W e o número de saídas. Observa-se um decaimento do número de sequências em W conforme aumenta o número de saídas. Essa relação interfere di-

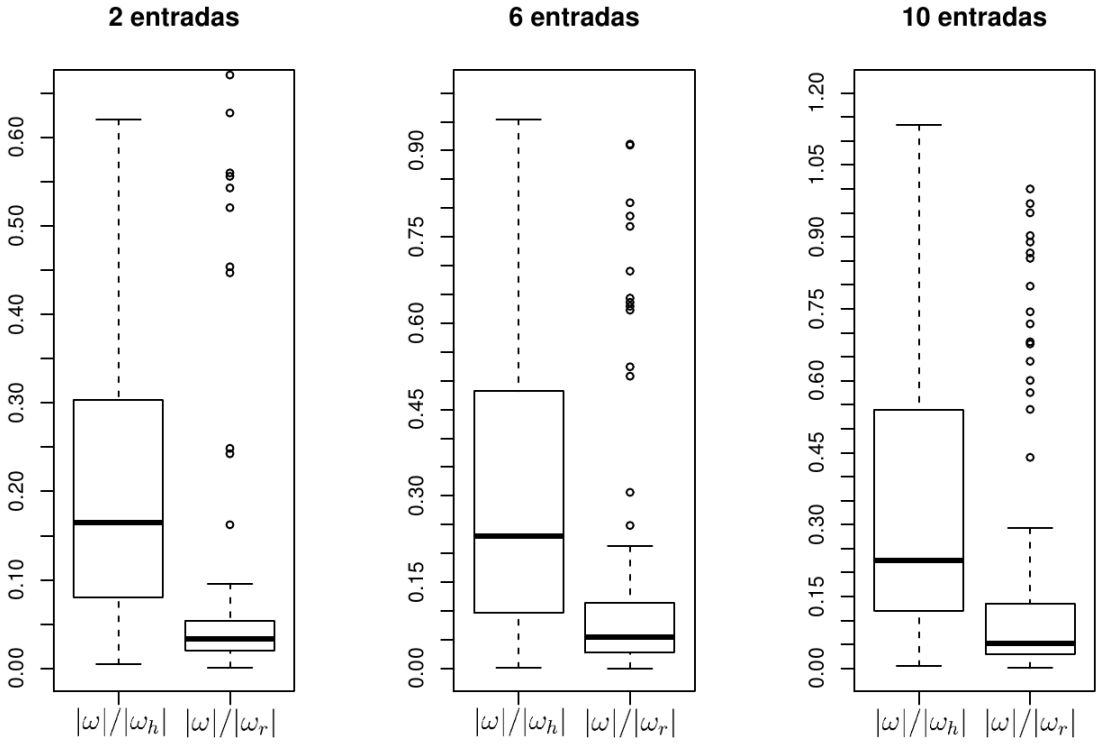

Figura 4.10: Boxplots para as relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$, para duas, seis e dez entradas.

retamente no tamanho das sequências de verificação geradas, uma vez que os métodos abordados nos experimentos são baseados em sequências de caracterização. Ou seja, o aumento no número de sequências em W acarreta no aumento do tamanho das sequências de verificação geradas, como pode ser observado na 4.13.

Menores valores para o número de saídas acarretam em uma maior redução no tamanho médio das sequências geradas pelo método proposto em relação aos demais métodos. Essa observação pode ser visualizada na Figura 4.14, a qual apresenta as relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$. Em relação ao método de Hennie (1964), a redução média obtida pelo método proposto é de $99,7\%$, 97% e $64,5\%$, para duas, quatro (mesmo número de entradas) e dez saídas, respectivamente. A redução é ainda maior em relação ao método de Rezaki e Ural (1995), apresentando os valores de $99,94\%$, $99,3\%$ e $91,5\%$, para duas, quatro e dez saídas, respectivamente.

Os boxplots apresentados na Figura 4.15 ilustram os resultados obtidos para as relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$, referentes a duas, seis e dez saídas. Ao analisar os boxplots, de maneira geral, é possível observar que o boxplot referente a duas saídas apresenta as maiores reduções, seguido dos boxplots referente a seis e dez saídas, respectivamente. Ao considerar a mediana, todos os boxplots apresentam reduções significativas proporcionadas pelo método proposto. Contudo, o que é importante observar nesses gráficos é que, em relação ao método de Hennie (1964), considerando o terceiro quartil, o método proposto proporciona um aumento de até 20% no tamanho das sequências geradas. Esse aumento

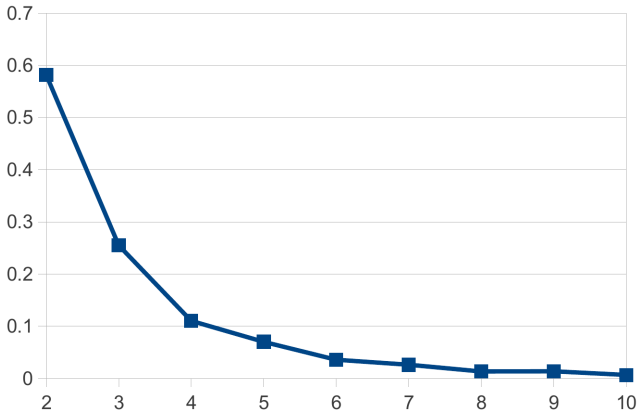

Figura 4.11: Proporção de MEFs sem DS no domínio total de MEFs geradas, em relação ao número de saídas.

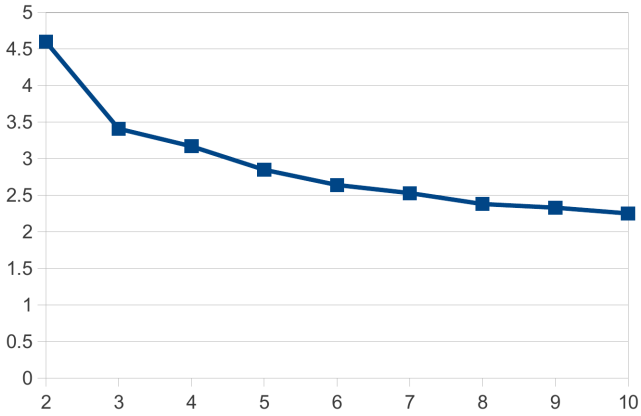

Figura 4.12: Número médio de sequências que compõem o conjunto W, em relação ao número de saídas.

decorre do fato de que o método de Hennie (1964) adota otimizações que são ignoradas no método proposto. No método proposto, as otimizações são proporcionadas somente pela utilização do conjunto de distinção ao invés do conjunto W. Dessa forma, quando o conjunto de distinção é equivalente ao conjunto W nenhuma otimização é provida, o que torna o método de Hennie (1964) mais eficiente nesses casos. Contudo, as mesmas otimizações empregadas no método de Hennie (1964) podem ser aliadas às otimizações provenientes do método proposto. As otimizações empregadas pelo método de Rezaki e Ural (1995) também podem ser agregadas ao método proposto, otimizando o número de sequências de transferência utilizadas.

4.4 Considerações Finais

Neste capítulo foi apresentado um novo método para geração de sequências de verificação baseado em conjuntos de distinção. Esse tipo de conjunto é derivado a partir de conjuntos de caracterização e, portanto, também existe para toda MEF minimal. Essa caracterís-

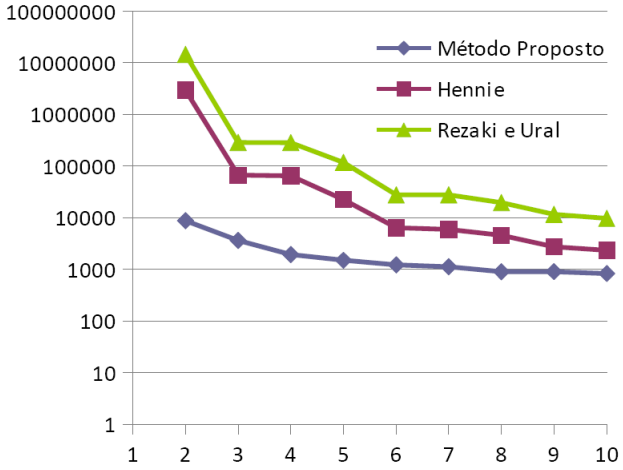

Figura 4.13: Tamanho médio das sequências de verificação geradas, em relação ao número de saídas.

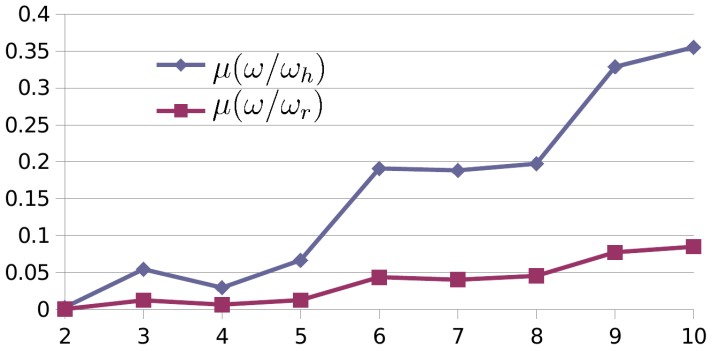

Figura 4.14: Relações $\mu(\omega/\omega_h)$ e $\mu(\omega/\omega_r)$, que representam a taxa de redução média obtida em relação ao número de saídas.

tica permite ao método um domínio de aplicação mais abrangente em relação aos métodos baseados em sequências de distinção. Além disso, o intuito principal do método proposto está na redução do tamanho das sequências de verificação geradas. A utilização de conjuntos de identificação, ao invés de conjuntos de caracterização, proporciona diferentes otimizações. Primeiro, as sequências de localização são reduzidas, dando lugar às denominadas sequências de reconhecimento. Essa redução impacta em uma redução global das sequências geradas, uma vez que esse tipo de sequência é utilizada em todo o processo de construção das sequências de verificação. Outra otimização decorrente da utilização de conjuntos de distinção está na redução do número de sequências necessárias para verificar cada transição de uma MEF.

O método proposto é composto por três etapas. A primeira etapa tem como intuito gerar um conjunto de sequências que perfazem o reconhecimento de todos os estados da MEF. Na segunda etapa, são geradas sequências de entrada que garantem a verificação de todas as transições da MEF. Por fim, na terceira etapa, as sequências geradas nas etapas anteriores são concatenadas com o auxílio de sequências de transferência. O método

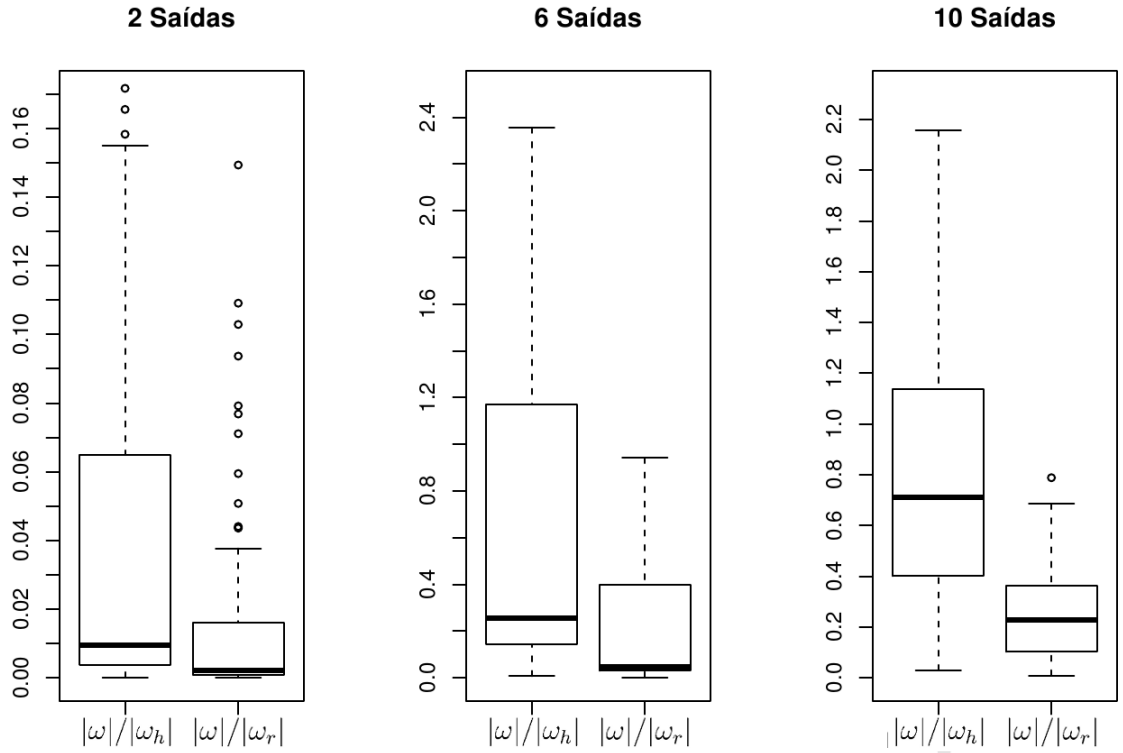

Figura 4.15: Boxplots para as relações $|\omega|/|\omega_h|$ e $|\omega|/|\omega_r|$, para duas, seis e dez saídas.

proposto foi ilustrado para uma MEF de exemplo. Ao final do processo foi gerada uma sequência de verificação com tamanho significativamente menor que as sequências geradas pelos métodos de Hennie (1964) e Rezaki e Ural (1995), para a mesma MEF. Esse resultado indica que o método proposto é capaz de reduzir o tamanho das sequências geradas em relação aos métodos existentes. Contudo, é necessário uma avaliação experimental mais aprofundada com o intuito de revelar as vantagens e desvantagens do método proposto em relação aos métodos existentes.

Dessa forma, neste capítulo, também foi apresentada uma análise comparativa entre o método proposto e os métodos de Hennie (1964) e de Rezaki e Ural (1995). Os experimentos foram conduzidos com o intuito de quantificar a redução proporcionada pelo método proposto no que se refere ao tamanho das sequências de verificação geradas. Para isso, os métodos comparados foram analisados em relação a variação de três fatores distintos: número de estados, número de entradas e número de saídas. Além disso, as MEFs geradas foram analisadas em relação à quantidade de sequências que compõem o conjunto W, uma vez que esse fator influencia diretamente no tamanho das sequências de verificação. Outro quesito analisado foi a proporção de MEFs que não possuem sequência de distinção em relação ao domínio total de MEFs geradas (incluindo MEFs que possuem tal recurso). Essa informação é importante se considerado que os métodos comparados foram elaborados para aplicação no domínio de MEFs sem sequência de distinção.

Considerando a variação do número de estados, o método proposto apresentou reduções significativas em relação aos demais métodos. Observou-se que quanto maior o número de estados, maior o número de sequências que compõem o conjunto W e, por consequência, maior o tamanho médio das sequências geradas. Ao analisar o tamanho médio obtido por cada método, considerando valores acima de sete estados, o tamanho das sequências geradas pelo método proposto representa menos que 1% do tamanho das sequências geradas pelos demais métodos. Ao analisar a redução obtida para cada MEF, considerando 75% das MEFs geradas com doze estados, o método proposto apresenta reduções acima de $99,5\%$ em relação à ambos os métodos comparados. Quanto à proporção de MEFs sem sequência de distinção no domínio de MEFs geradas, observou-se que o aumento do número de estados também aumenta a proporção de MEFs desse tipo. Nesse contexto, acima de dez estados, a proporção de MEFs sem sequência de distinção é superior a 50%.

Considerando a variação do número de entradas, o método proposto também apresentou reduções significativas em relação aos demais métodos comparados. Contudo, não foi constatada uma forte influência nos dados proveniente da variação do número de entradas. Em relação ao método de Hennie (1964), o método proposto provê redução no tamanho médio das sequências geradas de até 90%. Considerando o método de Rezaki e Ural (1995), essa redução é de até 97%. Por meio da análise dos boxplots, observou-se uma tendência a maiores reduções proporcionadas pelo método proposto para valores menores do número de entradas. A concentração de MEFs que não dispõem de sequência de distinção é maior para valores menores do número de entradas.

Considerando a variação do número de saídas, de maneira geral, o método proposto apresenta maiores reduções para valores menores do número de saídas. A quantidade de sequências em W é inversamente proporcional ao número de saídas. Dessa forma, o tamanho médio das sequências geradas também é menor conforme o aumento do número de saídas. A redução média proporcionada pelo método proposto é de até 97% em relação ao método de Hennie (1964), para duas saídas. Em relação ao método de Rezaki e Ural (1995), esse valor chega a $99,94\%$. Contudo, ao analisar a redução obtida para cada MEF gerada, observou-se que, para maiores valores do número de saídas, ao invés de reduzir, o método proposto apresenta um aumento no tamanho das sequências geradas em relação ao método de Hennie (1964). Esse aumento é decorrente do fato de que, em alguns casos, o conjunto de distinção não apresenta nenhuma redução em relação ao conjunto de caracterização. Dessa forma, nesses casos, as otimizações adotadas no método de Hennie (1964) tornam esse método mais eficiente. Contudo, as mesmas otimizações empregadas nesse método também podem ser agregadas ao método proposto.

Conclui-se que o método proposto apresenta grandes reduções em relação aos métodos comparados. Essas reduções são acentuadas para MEFs com mais estados, menos entradas e menos saídas. Existem casos em que não há otimização no tamanho das sequências geradas, uma vez que os conjuntos de distinção podem ser equivalentes ao conjunto W. O número de sequências que compõem o conjunto W influencia diretamente no tamanho das sequências geradas e tende a ser maior para mais estados e menos saídas. A concentração de MEFs sem sequência de distinção, no contexto de geração aleatória, tende a ser maior para mais estados, menos entradas e menos saídas.

Capítulo

5

Conclusões

Atividades de teste de software visam a agregar maior qualidade no processo de desenvolvimento de software e, além disso, abranger os mais diversos domínios de aplicação. No teste baseado em MEFs, foram propostos diferentes métodos com o intuito de otimizar a geração de casos de teste. No contexto de sequências de verificação, os casos de teste são compostos por uma sequência única capaz de identificar todos os possíveis defeitos em uma implementação, considerando algumas suposições. Contudo, a maioria dos métodos existentes para geração desse tipo de sequência são baseados em um recurso especial denominado de sequência de distinção.

Uma vez que nem toda MEF minimal possui sequência de distinção, os métodos baseados nesse recurso possuem um domínio de aplicação limitado. Assim, pesquisadores da área buscaram a elaboração de métodos baseados em recursos alternativos a esse tipo de sequência. Os métodos com domínio de aplicação mais abrangente, que são foco de estudo neste trabalho, são métodos baseados em conjuntos de caracterização. Esse tipo de conjunto existe para toda MEF minimal, possibilitando maior abrangência aos métodos. No entanto, a geração de sequências de verificação a partir desse recurso é complexa e produz sequências exponencialmente longas capazes de inviabilizar a aplicação do teste. Assim, a investigação de novos métodos capazes de gerar sequências de verificação menores nesse contexto torna-se essencial.

Neste trabalho investigou-se a redução das sequências de verificação geradas no contexto de MEFs que não dispõem de sequências de distinção. Dessa maneira, uma nova

estratégia foi proposta para geração de sequências de verificação baseada na utilização de conjuntos de distinção. A utilização desse tipo de conjunto proporciona reduções significativas em relação aos métodos existentes. Essas reduções foram constatadas a partir de uma avaliação experimental, também discutida neste trabalho. A seguir são apresentadas as principais contribuições deste trabalho de mestrado. Além disso, também são discutidas as limitações, dificuldades e perspectivas de trabalhos futuros.

5.1 Contribuições

A principal contribuição deste trabalho de mestrado é a proposta de uma estratégia para geração de sequências de verificação capaz de reduzir o tamanho das sequências geradas sem comprometer sua abrangência em relação ao domínio de aplicação. Para isso, a estratégia utilizada é baseada em conjuntos de identificação. Esse tipo de conjunto é derivado a partir do conjunto de caracterização de uma MEF e também pode ser utilizado no reconhecimento de seus estados. O reconhecimento dos estados de uma MEF, nos métodos existentes, é proporcionado pelas sequências de localização. Neste trabalho, um novo tipo de sequência foi proposto com esse intuito, as denominadas sequências de reconhecimento. A utilização das sequências de reconhecimento ao invés de sequências de localização acarretaram em reduções significativas nas sequências de verificação produzidas. Além disso, outras reduções foram constatadas a partir da utilização de conjuntos de distinção ao invés da utilização de conjuntos de caracterização no processo de construção das sequências de verificação.

As reduções proporcionadas pelo método proposto foram mensuradas por meio de uma avaliação experimental. Nessa avaliação foram geradas MEFs aleatórias sob a perspectiva de três diferentes fatores: variação do número de estados, número de entradas e número de saídas das MEFs. Os métodos avaliados foram executados e então comparados em relação ao método proposto. Os resultados apurados mostraram que, nas condições avaliadas, o tamanho das sequências geradas pelo método proposto correspondem a apenas uma pequena fração do tamanho das sequências geradas pelos métodos comparados.

5.2 Dificuldades e Limitações

Diante do objetivo estipulado para investigação neste trabalho, uma das dificuldades encontradas está relacionada ao entendimento dos detalhes que regem o funcionamento dos métodos existentes para geração de sequências de verificação. Além disso, há um consenso na literatura de que o procedimento para geração de sequências de verificação

no contexto de MEFs que não dispõem de sequências de distinção é complexo e não-trivial (Lee e Yannakakis, 1996).

Outra dificuldade encontrada está relacionada à investigação de possíveis otimizações que poderiam acarretar na redução das sequências geradas. Essa investigação foi realizada de maneira conjunta à implementação dos métodos e à experimentação de possíveis soluções. A implementação possibilitou um melhor entendimento dos métodos e propiciou uma visão mais ampla dos detalhes que regem o processo de construção das sequências de verificação. Além disso, os experimentos prévios acerca das soluções investigadas foram capazes de validar ou invalidar possíveis soluções, direcionando as investigações.

Após concluídas as investigações a cerca da estratégia proposta, a avaliação experimental resultou em novas dificuldades e limitações. Primeiramente, houve a dificuldade na escolha dos parâmetros e fatores a serem considerados nos experimentos. Segundo, dados os parâmetros avaliados e as circunstâncias envolvidas nos experimentos, há a limitação no que se refere à generalização dos resultados apurados. Apesar da avaliação apontar grandes reduções proporcionadas pelo método proposto em relação aos métodos existentes, há a possibilidade de novos experimentos em outras circunstâncias apontarem resultados diferentes.

A estratégia proposta neste trabalho apresenta algumas limitações. Em relação ao tamanho das sequências geradas, é possível que os conjuntos de distinção derivados para cada estado de uma MEF correspondam exatamente ao conjunto de caracterização. Nesses casos, nenhuma otimização é provida pelo método proposto. Além disso, apesar de prover reduções significativas na maioria dos experimentos apresentados, a estratégia proposta também gera sequências de verificação exponencialmente longas em relação ao conjunto de caracterização. Dessa forma, apesar das reduções, é possível que as sequências geradas pela estratégia proposta também sejam inviáveis para execução do teste.

5.3 Trabalhos Futuros

No contexto de sequências de verificação, as sequências geradas devem ser capazes de identificar qualquer defeito no domínio de MEFs com no máximo o mesmo número de estados da MEF de especificação. Dessa maneira, análises teóricas podem ser conduzidas com o intuito de revelar a cobertura de defeitos proporcionada pelas sequências geradas a partir do método proposto. Além disso, como discutido, as otimizações propostas nos trabalhos de Hennie (1964) e Rezaki e Ural (1995) podem ser aliadas às otimizações propostas neste trabalho, acarretando em reduções ainda maiores.

As reduções proporcionadas pelo método proposto podem ser avaliadas sob circunstâncias diferentes das abordadas neste trabalho. Além disso, uma amostragem maior de

MEFs pode ser utilizada. Outra possibilidade de investigação está na condução de experimentos com o intuito de comparar o método proposto com métodos baseados na utilização de operações *reset* e métodos baseados em sequências de distinção. Esse tipo de avaliação pode revelar informações importantes a cerca do quão aplicáveis são as sequências geradas no contexto abordado neste trabalho.

Referências Bibliográficas

Aho, A.; Dahbura, A.; Lee, D.; Uyar, M. An optimization technique for protocol conformance test generation based on UIO sequences and rural chinese postman tours. *IEEE Transactions on Communications*, v. 39, p. 1604–1615, 1991.

Andriole, S. J. *Software validation, verification, testing and documentation: A source book*. Princeton, NJ, USA: Petrocelli Books, Inc., 1986.

Barbosa, E. F.; Chaim, M. L.; Vincenzi, A. M. R.; Delamaro, M. E.; Jino, M.; Maldonado, J. C. *Introdução ao teste de software*, cap. Teste Estrutural Rio de Janeiro: Elsevier, p. 47–74, 2007.

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

Belina, F.; Hogrefe, D. The CCITT-specification and description language SDL. *Comput. Netw. ISDN Syst.*, v. 16, p. 311–341, 1989.

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

Bochmann, G. V.; Petrenko, R. Protocol testing: Review of methods and relevance for software testing. In: *Proceedings of the 1994 International Symposium on Software Testing and Analysis*, New York, NY, USA: ACM, 1994, p. 109–124.

Boute, R. T. Distinguishing sets for optimal state identification in checking experiments. *IEEE Trans. Comput.*, v. 23, p. 874–877, 1974.

Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A. *Model-based testing of reactive systems: advanced lectures*. 1st ed. Springer, 2005.

Chen, J.; Hierons, R. M.; Ural, H.; Yenigun, H. Eliminating redundant tests in a checking sequence. In: *TestCom*, 2005, p. 146–158.

Chow, T. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, v. 4, p. 178–187, 1978.

Clarke, E. M.; Grumberg, O.; Peled, D. A. *Model checking*. The MIT Press, 1999.

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: *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. *Commun. ACM*, v. 31, p. 1098–1115, 1988.

Delamaro, M. E.; Jino, M.; Maldonado, J. C. *Introdução ao teste de software*, cap. Conceitos básicos Elsevier, p. 1–7, 2007.

Demillo, R. A. Mutation analysis as a tool for software quality assurance. In: *COMPSAC80*, Chicago, IL, 1980, p. 390–393.

Dorofeeva, R.; El-Fakih, K.; Yevtushenko, N. An improved conformance testing method. In: *Formal Techniques for Networked and Distributed Systems - FORTE 2005*, v. 3731, Springer Berlin - Heidelberg, p. 204–218, 2005.

Frankl, P. G.; Weyuker, E. J. Testing software to detect and reduce risk. *J. Syst. Softw.*, v. 53, p. 275–286, 2000.

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

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

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

Hamon, G.; de Moura, L.; Rushby, J. Generating efficient test sets with a model checker. In: *Proceedings of the Second International Conference on Software Engineering and Formal Methods*, 2004, p. 261–270.

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

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

Hennie, F. C. Fault detecting experiments for sequential circuits. *Annual IEEE Symposium on Foundations of Computer Science*, v. 0, p. 95–110, 1964.

Hierons, R. M.; Ural, H. Reduced length checking sequences. *IEEE Transactions on Computers*, v. 51, p. 93–99, 2002.

Hierons, R. M.; Ural, H. Optimizing the length of checking sequences. *IEEE Trans. Comput.*, v. 55, p. 618–629, 2006.

IEEE IEEE standard glossary of software engineering terminology. Padrão 610.12-1990, IEEE, 1990.

Kuan, M. Graphic programming using odd or even points. *Chinese Math.*, v. 1, p. 273–277, 1962.

Lee, D.; Yannakakis, M. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, v. 84, p. 1090–1123, 1996.

Maldonado, J. C. Critérios potenciais usos: Uma contribuição ao teste estrutural de software. Tese de doutoramento, DCA/FEE/UNICAMP, Campinas, SP, 1991.

Mathur, A. P. *Foundations of software testing.* Pearson Education, 2008.

Myers, G. J. *The art of software testing.* 2 ed. Wiley, 2004.

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

Petrenko, A.; Boroday, S.; Groz, R. Confirming configurations in EFSM testing. *IEEE Trans. Softw. Eng.*, v. 30, p. 29–42, 2004.

Petrenko, A.; Yevtushenko, N.; Lebedev, A.; Das, A. Nondeterministic state machines in protocol conformance testing. In: *Protocol Test Systems*, Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993, p. 363–378.

Petrenko, R.; Yevtushenko, N. Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers*, p. 1154–1165, 2005.

Pressman, R. S. *Software engineering: A practitioner's approach.* 7 ed. McGraw-Hill Science, 2009.

Rapps, S.; Weyuker, E. J. Selecting software test data using data flow information. *IEEE Trans. Softw. Eng.*, v. 11, p. 367–375, 1985.

Rezaki, A.; Ural, H. Construction of checking sequences based on characterization sets. *Computer Communications*, v. 18, p. 911–920, 1995.

Sabnani, K.; Dahbura, A. A protocol test generation procedure. *Comput. Netw. ISDN Syst.*, v. 15, p. 285–297, 1988.

Simao, A. *Introdução ao teste de software*, cap. Teste Baseado em Modelos Rio de Janeiro: Elsevier, p. 27–46, 2007.

Simao, A.; Petrenko, A. Generating checking sequences for partial reduced finite state machines. In: *Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems: 8th International Workshop*, Berlin, Heidelberg: Springer-Verlag, 2008, p. 153–168.

Simao, A.; Petrenko, A. Checking completeness of tests for finite state machines. *IEEE Trans. Comput.*, v. 59, p. 1023–1032, 2010.

Simao, A.; Petrenko, A.; Yevtushenko, N. Generating reduced tests for FSMs with extra states. In: *Testing of Software and Communication Systems*, v. 5826 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 129–145, 2009.

Simao, A. S. Aplicação da análise de mutantes no contexto do teste e validação de redes de petri coloridas. Tese de doutoramento, ICMC/USP, São Carlos, SP, 2004.

Sinha, A.; Smidts, C. HOTTest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Trans. Softw. Eng. Methodol.*, v. 15, p. 242–278, 2006.

Tretmans, J. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, p. 103–120, 1996.

Ural, H.; Wu, X.; Zhang, F. On minimizing the lengths of checking sequences. *IEEE Trans. Comput.*, v. 46, p. 93–99, 1997.

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