Avaliação de custo e eficácia de métodos e critérios de teste baseado em Máquinas de Estados Finitos

Flávio Dusse

Avaliação de custo e eficácia de métodos e critérios de teste baseado em Máquinas de Estados Finitos

Flávio Dusse

Orientador: ***Prof. Dr. José Carlos Maldonado***

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

USP - São Carlos
Janeiro/2010

Aos meus pais, Roberto e Izabel, e avós, Hermínia e Maria "in memorian"

AGRADECIMENTO ESPECIAL

Ao Prof. Dr. Adenilso da Silva Simão pela co-orientação, apoio e incentivo a este trabalho. Muito obrigado Ades.

Agradecimentos

A toda minha família, em especial, meus pais Roberto e Izabel, minhas avós Hermínia e Maria "*in memorian*" e meus irmãos Igor e Karina. Não existem palavras suficientes para expressar o quanto amo, admiro e tenho orgulho de vocês.

Ao meu orientador, Prof. Dr. José Carlos Maldonado, pelos conselhos, conhecimento e confiança, sem as quais este trabalho não teria sido concluído.

Ao meu co-orientador, Prof. Dr. Adenilso da Silva Simão, que sempre acompanhou de perto, mesmo a distância, a condução deste trabalho. Obrigado Ades, pela atenção, auxílio, orientação, críticas, elogios e amizade.

Aos amigos da USP, especialmente aos amigos do LABES que conviveram comigo nesses últimos anos de manhã, tarde, noite e madrugada. Não citarei nome nenhum porque vocês que estão lendo sabem quem são. Obrigado a todos!

Aos amigos que fiz em São Carlos e região. Valeu pelas horas de amizade e descontração!

Aos amigos de Salvador, Aracaju, Recife e todas as cidades desse mundo nas quais um amigo está, por, apesar da minha ausência, ainda serem meus amigos e encherem meu coração dos bons momentos da vida.

Aos professores e funcionários do ICMC, pelo constante auxílio, disposição e atenção.

À Érica, Gabriel (Ceará), Paulo (Nerso), Jorge (Piu), Rodrigo (Logan) e principalmente Waleska (Tita), pela ajuda na revisão deste trabalho.

A todas as pessoas que contribuíram de alguma forma para a realização deste trabalho.

Ao CNPQ e a FAPESP, pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

B-UIO - *Backward Unique Input-Output*
CS - *Checking Sequence*
CT - Caso de Teste
C-UIO - *Circular Unique Input-Output*
DS - *Distinguishing sequence*
FSM - *Finite State Machine*
GTSC - Geração automática de casos de Teste baseada em *StateCharts*
HSI - *Harmonized State Identification*
IEEE - *Institute of Electrical and Electronics Engineers*
IF - *Initialization Fault Coverage Criterion*
MEF - Máquina de Estados Finitos
MEFE - Máquina de Estados Finitos Estendidas
MUIO - *Multiple Unique Input-Output*
OO - Orientado a Objeto
RCP - *Rural Chinese Problem*
S - *State Coverage Criterion*
SC - *State Couting*
SOA - *Service Oriented Architecture*
SS - Sequência de Sincronização
ssh - *Secure Shell*
ST - *State Tour*
Sw - *Switch Cover*
T - *Transition Coverage Criterion*
TAG - *Test Automatic Generation*
TF - *Transition Fault Coverage Criterion*
TS - *Test Suite*
TT - *Transition Tour*
UIO - *Unique Input-Output*
UIOv - *Unique Input-Output variation*
UML - *Unified Modeling Language*
VV&T - Validação, Verificação e Teste
Wp - *Partial W*

Resumo

MÉTODOS de geração de casos de teste visam a gerar um conjunto de casos de teste com uma boa relação custo/benefício. Critérios de cobertura de teste definem requisitos de teste, os quais um conjunto de teste adequado deve cobrir. Métodos e critérios visam a selecionar casos de teste baseados em especificações, que podem ser descritas por meio de modelos, tais como Máquinas de Estados Finitos (MEF). Existem diversos métodos de geração e critérios de cobertura, diferindo entre si em função das propriedades exigidas da MEF, do custo dos testes gerados e da eficácia na revelação de defeitos. Apesar de pesquisas intensas na definição desses métodos e critérios, são poucas as ferramentas de apoio disponíveis assim como são poucos os relatos de aplicação em termos de custo e eficácia para a definição de estratégias de teste efetivas. Dessa forma, é necessário obter dados reais das vantagens e desvantagens dos métodos e critérios para subsidiar a tomada de decisão no processo de desenvolvimento de software no que tange às atividades de teste e validação. Este trabalho apresenta resultados de experimentos para avaliar o custo e a eficácia de aplicação dos métodos e critérios mais relevantes para subsidiar a definição de estratégias de teste em diversos contextos, como por exemplo, no desenvolvimento de protocolos e de sistemas reativos. Utiliza-se um protótipo desenvolvido a partir de uma reengenharia da ferramenta Plavis/FSM para apoiar os experimentos.

Abstract

TEST case generation methods aim to generate a test suite that offers an acceptable trade-off between cost and avail. Test coverage criteria define testing requirements, which an adequate test suite must fulfill. Methods and criteria help to select test case from specifications, which can be describe as models, for example Finite State Machines (FSM). There are several generation methods and coverage criteria that differ depending on the required properties of the FSM, the cost of generated tests and the effectiveness in revealing faults. In spite of intense researches in the definition of those methods and criteria, there are few available tools to apply them as well as application reports about cost and effectiveness issues to define effective test strategies. Thus, it is necessary to obtain real data of the advantages and disadvantages of the methods and criteria to provide decision-making in the software development process as far in the validation and test activities. This work aimed to lead experiments to evaluate the cost and the effetiveness in applying the most relevant methods and criteria to subsidize test strategies definition in several contexts as the communication protocol development and the reactive systems development. A prototype was developed based on reengineering of the Plavis/FSM tool to support the experiments.

CAPÍTULO

1

Introdução

1.1 Contextualização

O avanço da Tecnologia de Informação é um fato percebível em praticamente todas as atividades da sociedade; o uso de softwares mais complexos e mais confiáveis tornou-se indispensável. Como resultado desse avanço, a demanda por quantidade e, principalmente, por qualidade de produtos de software, encontra-se em crescimento. Somando a competitividade de mercado existente nos dias atuais, é essencial que produtos de softwares sejam de boa qualidade e que o tempo e o custo de desenvolvimento desses produtos sejam minimizados. É função da Engenharia de Software prover recursos que visem a tornar o desenvolvimento do software mais consistente, reduzindo o custo sem interferir na qualidade do produto.

As atividades de Validação, Verificação e Teste (VV&T) são executadas para garantir a qualidade em um desenvolvimento de software e devem ser conduzidas, assim como as outras atividades de Engenharia de Software, de forma sistemática, clara e rigorosa. Equipes de VV&T estão sempre buscando métodos e ferramentas que possibilitem facilitar a compreensão, aumentar a produtividade, diminuir custos e atender aos padrões de qualidade, assim como viabilizar a obtenção de resultados reproduzíveis e auditáveis.

Teste de software auxilia detectar erros introduzidos involuntariamente no produto de software. Nesse contexto, tem-se o teste de conformidade (Petrenko et al., 1993), que consiste em confrontar

o comportamento do software com o comportamento previamente esperado. A identificação de inconformidades, de preferência o mais cedo possível, visa a reduzir os gastos de manutenção e evitar ao máximo o risco do cliente descobrir tais falhas e, em caso de sistemas críticos, que uma catástrofe ocorra. Ademais, em razão de o teste constituir uma atividade onerosa no desenvolvimento de software, estudos teóricos e experimentais têm sido conduzidos por grupos de pesquisas, a fim de identificar métodos eficazes para aplicar os recursos disponíveis.

Segundo Pressman (2006), existem diversas categorias de produto de software, tais como software básico, sistemas de informação, sistemas científicos, sistemas embutidos e sistemas reativos. Esta última é de particular interesse para o presente trabalho. Um sistema reativo caracteriza-se por interagir permanentemente com o ambiente em que está inserido por meio de eventos sequenciais de entrada e saída. O comportamento de um sistema reativo é fortemente baseado em estados, ou seja, uma mesma entrada pode gerar saídas diferentes em relação ao estado em que o sistema se encontra. Usualmente, sistemas reativos são empregados em atividades que envolvem questões críticas, nas quais falhas podem implicar em perda de vidas humanas e prejuízos financeiros. Por essa razão, atividades de teste devem ser conduzidas com o máximo rigor, para que o sistema atinja um grau aceitável de qualidade e confiabilidade. Como exemplos desses sistemas, pode-se citar sistemas embarcados críticos, controle de tráfego e o monitoramento de ambientes industriais e hospitalares. O contexto mais geral deste trabalho é a investigação de estratégias de teste e validação para o desenvolvimento de sistemas reativos.

Para execução de testes de conformidade, testes baseados em modelos são úteis, pois os custos de geração de casos de teste (CTs) e de correção de defeitos tendem a ser menores se os testes forem executados antes da fase de codificação. Em razão de sua simplicidade conceitual e expressividade na descrição do comportamento de um sistema, uma das representações mais utilizadas e pesquisadas no contexto de teste baseado em modelos são as Máquinas de Estados Finitos (MEFs). Por meio de MEFs e com apoio de ferramentas apropriadas, a geração de CTs para verificar o comportamento esperado de um sistema é automatizada, reduzindo custos de geração e de manutenção. A utilização de ferramentas de teste reduz também as falhas humanas que eventualmente possam ocorrer.

Métodos de geração de casos de teste a partir de MEFs, ou simplesmente métodos de geração, têm sido propostos há décadas (Moore, 1956; Chow, 1978; Fujiwara et al, 1991; Dorofeeva et al, 2005b). Tais métodos buscam gerar sequências de entradas adequadas para o teste de implementações a partir de especificações modeladas por MEFs. Da mesma forma, critérios de cobertura de teste baseado em MEFs também têm sido propostos (Simão et al., 2009). Um critério define requisitos de teste, os quais um conjunto de teste adequado deve cobrir. A seleção de qual método ou critério utilizar depende das propriedades da MEF, do custo e da eficácia dos métodos ou critérios. Deseja-se nessa seleção o menor custo e sem que a eficácia seja prejudicada. Os métodos e critérios foram desenvolvidos inicialmente no contexto de teste de hardware e, posteriormente, no contexto de teste e validação de protocolos de comunicação. Atualmente, os métodos e critérios

têm sido aplicados em diversos outros contextos, como no desenvolvimento de sistemas orientado a objetos (OO), de sistemas distribuídos e principalmente de sistemas reativos (Binder, 1999).

Apesar de intensas pesquisas na definição e melhorias desses métodos e critérios, são poucas as ferramentas de apoio disponíveis, assim como são poucos os relatos de aplicação em termos de custo e eficácia para a definição de estratégias de teste efetivas. Dessa forma, é necessário obter dados reais das vantagens e desvantagens de cada método e critério para subsidiar a tomada de decisão no processo de desenvolvimento de software no que tange às atividades de teste e validação. Assim como em outras áreas (Química, Física, Medicina, etc.), na Engenharia de Software tais dados podem ser obtidos por meio de experimentos.

Este trabalho é realizado no contexto do grupo de pesquisas do Laboratório de Engenharia de Software (Labes) do Instituto de Ciências Matemáticas e de Computação (ICMC/USP). O grupo realiza pesquisas na área de qualidade de software visando à viabilização de experimentos científicos e educacionais dentro da área. O grupo participou no Projeto PLAVIS, com apoio da CAPES, COFECUB e CNPq, com envolvimento de instituições nacionais e francesas (PLAVIS, 2005). Entre outras atividades, foi implementada a ferramenta Plavis/FSM para apoiar a aplicação de métodos de geração.

Atualmente alguns membros do grupo participam do projeto *Qualipso* que é financiado pela União Europeia, contendo universidades e empresas da Europa, Brasil e China (Qualipso, 2004). O objetivo do projeto é prover confiabilidade e qualidade em sistemas *open source*. Um dos resultados esperados é desenvolver um novo conceito de *forge* baseado em SOA (Arquitetura Orientada em Serviços). Todas as funcionalidades serão fornecidas por *web services* e *gadgets* que acessarão os dados. Este trabalho de mestrado também está inserido no contexto do projeto *Qualipso*.

1.2 Motivação

Para o cliente, softwares devem possuir um alto nível de qualidade e serem entregues até o tempo previsto. Softwares de alta qualidade consistem na garantia do nível de confiança, bem como de segurança, esperado. Para atingir esse nível, o software deve passar por uma bateria de testes, que verifica se os requisitos estão atendidos. No contexto de sistemas reativos, no qual uma falha pode ocasionar prejuízos irreparáveis, é conveniente a concentração de esforços nos testes.

Pesquisas de modelos formais para planejamento, geração, execução e avaliação de testes podem ser utilizadas para aprimorar as atividades de teste. Um desses modelos formais são as MEFs que, em conjunto com os métodos de geração e critérios de cobertura, permitem gerar CTs para softwares especificados por esse modelo. Um problema identificado na utilização de MEFs em aplicações reais é a grande quantidade de CTs gerados que, em muitos casos, inviabiliza um teste completo em razão das limitações de tempo e recursos disponíveis. Os testadores devem planejar uma estratégia a partir de objetivos de teste bem definidos, gerando um conjunto menor ou

selecionando um subconjunto dos CTs gerados. Estratégias de teste podem ser definidas com o conhecimento de experimentos previamente conduzidos.

O propósito de conduzir experimentos é entender os pontos fracos e fortes do que é avaliado de uma forma que seja possível adaptar para os objetivos e características de projetos particulares, empacotando a experiência adquirida e aumentando seu potencial de reutilização em outros projetos (Wohlin et al., 2000).

Foram encontrados poucos relatos que avaliam em termos de custo e eficácia os métodos de geração e os critérios de cobertura e não foi encontrada uma ferramenta disponível para condução de experimentos para compará-los. Assumiu-se a importância de desenvolver uma ferramenta de apoio a experimentos para os métodos de geração e critérios de cobertura, visto que estes métodos estão sendo cada vez mais aplicados na indústria. Foram encontrados dois exemplos atuais de aplicação dos métodos na validação de softwares embarcados. O primeiro pelo Instituto Nacional de Pesquisa Espacial (INPE) (Santiago et al., 2008) e o segundo exemplo encontrado é de uma parceira da Hewlett-Packard Brasil (HP) com a Pontifícia Universidade do Rio Grande do Sul (PUCRS) (Peralta, 2009). MEFs e alguns métodos de geração são utilizados no INPE e na HP para validação de softwares espaciais embarcados em sistemas de satélites ou sistemas de controle de solo e softwares embarcados em impressoras, respectivamente.

1.3 Objetivo

O objetivo deste trabalho é elaborar experimentos que avaliem o custo e a eficácia dos métodos de geração e critérios de cobertura utilizados no teste baseado em MEFs. Além disso, é feita uma reengenharia da ferramenta Plavis/FSM. Ela é estudada a fim de extrair informações para o desenvolvimento de um protótipo para dar apoio aos experimentos.

A arquitetura do protótipo é baseada em serviços e incorpora recursos para a preparação do experimento e para a coleta dos dados. Os *web services* implementados são disponibilizados no contexto do projeto *Qualipso*. Alguns passos dos experimentos são realizados no protótipo, porém a maior parte de sua execução é realizada em um *cluster*. As MEFs utilizadas nos experimentos são provenientes de geração aleatória e os tamanhos médios dos conjuntos de teste gerados por cada método e por cada critério são comparados a partir de sete casos. Cada caso é diferenciado por quais parâmetros de MEF sofrem variações. Para os critérios e para dois métodos, é avaliada a eficácia dos conjuntos de teste gerados por meio de análise de mutantes e satisfação de condições de suficiência.

A comparação dos resultados experimentais permite avaliar as vantagens e desvantagens em termos de custo e eficácia de cada método e de cada critério e os problemas práticos envolvidos na escolha dos métodos e dos critérios em diversos contextos e tipos de MEFs. A avaliação dos resultados também permite que, a partir dos recursos disponíveis, estratégias de teste efetivas e de baixo custo sejam planejadas.

1.4 Organização

O restante deste trabalho está organizado como segue.

No Capítulo 2, são apresentados os conceitos, as visões, as fases e as técnicas a respeito da atividade de teste de software, com ênfase nas técnicas baseadas em modelos. No Capítulo 3, discutem-se as motivações para o teste baseado em MEFs. São apresentados os principais conceitos referentes às MEFs. Os métodos de geração de CTs e os critérios de cobertura são apresentados no Capítulo 4. Três ferramentas relacionadas a teste baseado em MEFs são apresentadas nesse capítulo, assim como dois trabalhos relacionados à avaliação experimental desse tipo de teste. O desenvolvimento de um protótipo de apoio aos experimentos e suas limitações são os principais tópicos abordados no Capítulo 5. Todos os detalhes da execução, da coleta de dados e da análise dos resultados obtidos são descritos no Capítulo 6. Por fim, no Capítulo 7, conclui-se esta dissertação resumindo as principais contribuições desta pesquisa. São também apresentadas algumas observações finais e direções para trabalhos futuros.

CAPÍTULO

2

Teste de Software

2.1 Considerações Iniciais

A grande área da Engenharia de Software, na qual este trabalho está inserido, é o teste de software. Grupos de desenvolvimento de software, tanto na academia como na indústria, consideram que é imprescindível a atividade de teste e que esta esteja sujeita ao rigor e controle necessários a depender de quão crítico é o software a ser desenvolvido. Testar um produto de software é determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado. Teste de Software é de fundamental importância para o controle da qualidade dos produtos desenvolvidos.

Neste capítulo é apresentada uma fundamentação teórica sobre teste de software para auxiliar o leitor a entender os conceitos envolvidos. Na Seção 2.2, é introduzida a importância da disciplina de teste e seus principais objetivos. Uma discussão geral sobre os conceitos, visões, fases e técnicas de teste está divida nas subseções. Em seguida, na Seção 2.3, a técnica de teste baseada em erros é apresentada. Na Seção 2.4, é discutida com mais detalhes a técnica de teste baseada em modelos com a apresentação de alguns modelos.

2.2 Teste de Software

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, ferramentas e métodos empregados, erros no produto ainda podem ocorrer (Maldonado et al., 2004). Equipes de desenvolvimento de software têm destinado recursos em atividades que diminuem a introdução de erros e, caso introduzidos, que facilitem a descoberta deles. É função do teste de software mostrar a presença de erros (e não a ausência) no software para eventuais manutenções, proporcionando-o qualidade e confiabilidade.

Teste é o processo de executar um sistema ou componente sob condições específicas, observando os resultados e avaliando algum aspecto do sistema ou componente (IEEE, 1990). Em um ciclo de desenvolvimento de software é comumente aceito que o teste de software seja uma atividade indispensável. Segundo Myers (2004), enquanto outras atividades, como a programação, são ditas construtivas, diz-se que teste é uma atividade destrutiva. A função do testador é projetar CTs capazes de "destruir" o software em busca de defeitos, para que outras atividades sejam refeitas.

Teste existe para fornecer confiança na corretude de um programa e garantir sua qualidade. Embora, em geral, seja impossível comprovar a ausência de erros no programa, testes auxiliam na localização de erros durante o desenvolvimento de uma aplicação e determinam se um sistema computadorizado se comporta conforme especificado nos requisitos (Beizer, 1990). Nas próximas seções, serão apresentados alguns conceitos e definições de teste, as visões, as fases do processo teste e algumas técnicas comumente utilizadas. A divisão das atividades de teste em visões, fases e técnicas é uma maneira teórica e prática de estudar e desempenhar essas atividades.

2.2.1 Conceitos e Definições

A IEEE estabeleceu o padrão de número 610.12-1990 (IEEE, 1999), um glossário da terminologia da Engenharia de Software a fim de evitar enganos de interpretação com termos de outras áreas da computação e de outras ciências. Os seguintes conceitos são diferenciados:

- Falha (*fail*): refere-se à incapacidade do software de realizar a função requisitada, como por exemplo terminação anormal ou restrição temporal violada.
- Falta (*fault*): diz respeito à causa de uma falha. Pode ser um código incorreto ou faltando.
- Erro (*error*): é o estado intermediário entre uma falta e uma falha. Pode resultar em falha, se propagada até a saída.

Por esses conceitos observa-se que a causa de um erro é uma falta e uma falha é o efeito de um erro. Teste de software busca encontrar o maior número possível de falhas, ocasionadas por erros, em razão de uma ou mais faltas. Neste trabalho, será usado somente o termo erro para fazer referência às inconsistências de software.

2.2.2 Visões do Processo de Teste de Software

Na Figura 2.1 são ilustradas as visões de um processo de Teste, segundo Binder (1999).

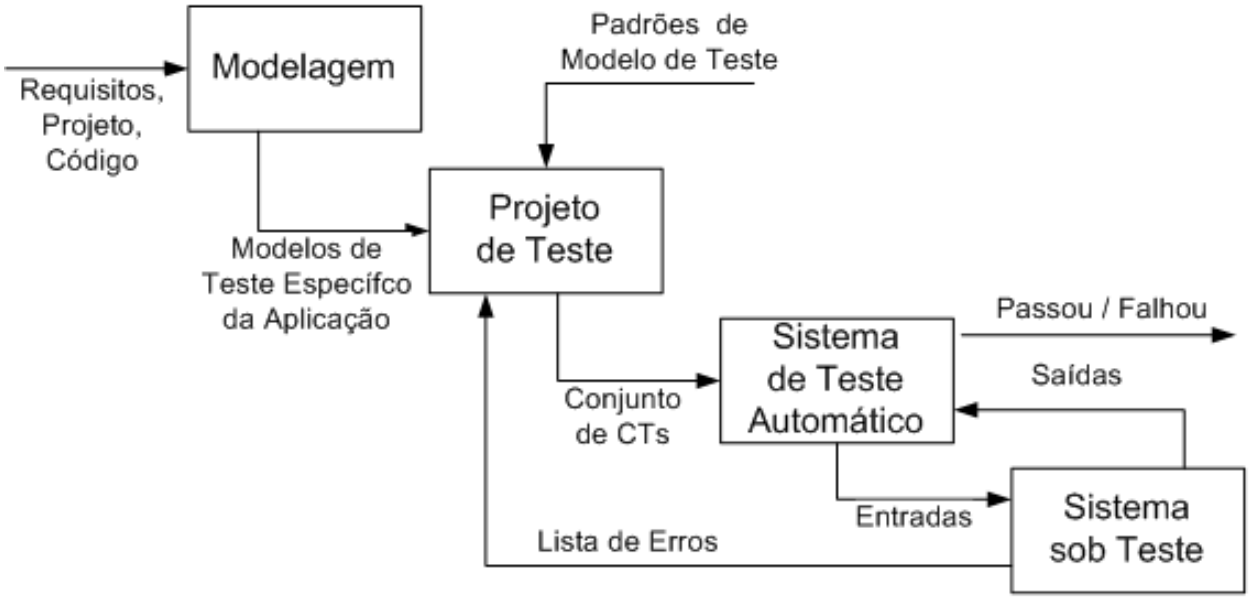

Figura 2.1: Visões de um processo de Teste. Adaptado de Binder (1999)

O início de um processo de teste ocorre simultaneamente com o início do processo do desenvolvimento de software, apesar de algumas vezes isso não transparecer de forma tão óbvia. A princípio, elabora-se um planejamento dos testes, que define a priorização e quais requisitos do software serão testados. Também são definidas estratégias de teste, assim como um cronograma, esforço e recursos exigidos. Em geral, o planejamento dos testes é realizado no início do projeto de desenvolvimento do software junto com o planejamento das outras atividades do desenvolvimento.

Em uma fase mais adiante, os artefatos já desenvolvidos (**requisitos, projeto, código**) passam por uma fase de **modelagem**, que produz artefatos de teste que representam o comportamento requerido do sistema sob teste. Esses **modelos de teste específico da aplicação** servem de entrada para o **projeto de teste**. O projeto de teste também se baseia em **padrões de modelos de teste**, como por exemplo MEFs, que oferecem uma maneira sistemática e reusável de gerar CTs.

A quantidade de CTs necessária para um teste completo tende a ser extremamente elevada, incorrendo em altos custos de criação e manutenção do **conjunto de CTs**. Segundo Binder (1999), o projetista de teste é responsável por:

1. Identificar, modelar e analisar o comportamento do sistema.

2. Decidir a forma (método ou algoritmo) que os CTs serão gerados baseados nessa perspectiva externa e nos recursos disponíveis.

3. Adicionar (ou eliminar) CTs baseados em análise de código, suspeitas e heurísticas.

4. Definir um resultado esperado para cada caso de teste e escolher uma forma de avaliar se o teste passou ou falhou. Pode-se utilizar um oráculo de teste (programa que decide se os valores das saídas dos CTs estão corretos).

Depois de realizado o projeto de teste, um conjunto viável de CTs é aplicado no **sistema sob teste** e um **sistema de teste automático** deve executá-los, a menos que testes manuais sejam indicados. É proveitoso automatizar o processo, visto que o **sistema de teste automático** aplica as **entradas** e avalia as saídas do **sistema sob teste**, que reporta uma **lista de erros** para o projetista de teste. Outra motivação para a automação na execução de testes se baseia na re-execução de CTs, principalmente quando alguma modificação é feita e se deseja garantir que outras partes do sistema, que estavam corretas, continuem funcionando da forma esperada. Essa re-execução é chamada de teste de regressão.

2.2.3 Fases de Teste

Além das visões do processo, testes podem ser realizados em três fases que, segundo Pressman (2006), se inicia em unidades de programa individuais, movendo para projetos de teste para facilitar a integração dessas unidades, culminando em testes que exercitam o sistema desenvolvido por inteiro. As três fases são detalhadas a seguir:

- **Teste de Unidade**: os esforços de verificação são focados na menor unidade executável do projeto de software: componente ou módulo. Nessa fase, unidades de software são testadas separadamente com o objetivo de identificar erros na lógica de implementação e estruturas de dados internas de uma unidade. Esse tipo de teste pode ser conduzido paralelamente entre múltiplos módulos. Em alguns casos, em que há dependências de outros módulos, é necessária a criação de *stubs* e *drivers*. *Stub* simula os módulos subordinados ao módulo sob teste e *driver* é um "programa principal" que fornece dados de teste ao módulo sob teste e revela algum resultado.

- **Teste de Integração**: mesmo depois de todas as unidades testadas, erros ainda podem existir se as integrarem. Nessa fase, testes são conduzidos para identificar erros associados às interfaces entre as unidades, ou seja, se estas estão operando coletivamente sem erros. Para facilitar esse acoplamento das unidades, utilizam-se estratégias de integração incremental, sendo as principais conhecidas como integração *top-down* e *bottom-up*. Na *top-down* os módulos são integrados movendo em sentido decrescente da hierarquia de controle, ou seja, se inicia com o módulo principal e os módulos subordinados são integrados incrementalmente. Na estratégia *bottom-up* ocorre o contrário, se inicia com os módulos atômicos, que são os do nível mais baixo.

- **Teste de Sistema**: o objetivo dessa fase é testar a aplicação completamente integrada. Nessa fase são testadas as capacidades e características presentes somente com o sistema inteiro. O

objetivo é garantir a funcionalidade completa e correta da especificação como: confirmar o comportamento esperado (funções, performance, segurança, etc.), e verificar se a interação com o ambiente do sistema (hardware, banco de dados, usuários, etc.) não propicia erros.

2.2.4 Técnicas de Teste

Existem diversas técnicas de teste, que devem ser aplicadas de acordo com o aspecto do sistema (interface, carga, desempenho, por exemplo) a ser testado, bem como seu tipo (sistema orientado a objetos, reativo, distribuído, por exemplo). De acordo com Maldonado et al. (2004), técnicas de teste possuem diferentes perspectivas e impõe-se a necessidade de se estabelecer uma estratégia de teste que contemple as vantagens e os aspectos complementares dessas técnicas, uma vez que elas detectam categorias de erros distintas. A principal diferença entre as técnicas está na origem da informação utilizada na avaliação e construção dos conjuntos de CTs. As duas técnicas de teste mais conhecidas são o teste estrutural e o teste funcional.

- **Teste estrutural**: também conhecido como teste de caixa branca, baseia-se em implementações (código). Seu principal objetivo é testar detalhes procedimentais de partes elementares do programa (Myers, 2004), de onde os requisitos de teste devem ser extraídos. Os critérios de cobertura dessa técnica, assim como a geração de CTs, baseiam-se na utilização de grafos como ferramenta para a descrição de caminhos, como os definidos pelos fluxos de controle (comandos ou desvios dos programa) e dados (definição ou uso das variáveis) da implementação. Esses grafos são chamados de grafo de programa e grafo def-uso, respectivamente. Dentre os critérios baseados em fluxo de controle, os mais conhecidos são: critério todos-nós, critério todos-arcos, critério todos-caminhos (Myers, 2004). Enquanto os critérios baseados em fluxo de dados se destacam: critério todas-definições, critério todos-usos (Rapps e Weyuker, 1982) e critério potenciais-usos (Maldonado, 1991).

- **Teste funcional**: também conhecido como teste de caixa preta, baseia-se na especificação do sistema para sua definição e aplicação tendo como objetivo validar somente as funcionalidades do sistema. Ao contrário do teste estrutural, esta técnica é, em geral, independente de implementação, o que permite sua definição durante a fase de projeto do sistema. O nome caixa preta vem dessa característica, pois se considera um programa como sendo uma caixa em que apenas os dados de entrada e saída podem ser visualizados. Exemplos de critérios da técnica de teste funcional são: particionamento em classes de equivalência, análise do valor limite e grafo de causa-efeito (Pressman, 2006).

Na literatura, encontram-se vastos relatos de técnicas, métodos e ferramentas utilizadas nas atividades de teste. Para este trabalho duas técnicas de teste são amplamente utilizadas: teste baseado em erros e teste baseado em modelos, sendo descritos em detalhes a seguir.

2.3 Teste Baseado em Erros

Segundo Maldonado et al. (2004) a técnica de teste baseado em erros utiliza informações sobre os tipos de erros mais frequentes no processo de desenvolvimento de software para derivar os requisitos de teste. Teste algébrico de defeitos (Morell, 1990), semeadura de erros (Pressman, 2006) e análise de mutantes (Demillo, 1980) são exemplos de critérios que se baseiam em erros, com destaque para esse último.

A análise de mutante visa a avaliar a qualidade de um conjunto de casos de teste TS em relação a um programa P. Deriva-se um conjunto de programas I denominados mutantes de P, nos quais um ou mais erros são introduzidos em cada programa por meio de operadores de mutação. Tais operadores representam regras a serem aplicadas em P baseadas nos erros que são comumente cometidos pelos desenvolvedores.

Essa análise busca verificar se TS é capaz de revelar as diferenças entre o programa P e os programas de I. Se uma diferença é revelada em um mutante de P é dito que TS "mata" esse programa mutante. Caso ainda existam mutantes vivos, faz-se necessário avaliar se os mutantes vivos são equivalentes a P. Caso negativo, novos casos de teste devem ser inseridos no conjunto TS. Ao final da execução de I, o escore de mutação pode ser calculado. Esse escore avalia a qualidade do conjunto TS e a fórmula para calculá-lo é:

$$ms(P, TS) = \frac{DM(P,TS)}{M(P) - EM(P)}$$

Onde, $ms(P, TS)$ é o escore de mutação; $DM(P, TS)$ é o número de mutantes mortos por TS; $M(P)$ é o número total de mutantes gerados; e $EM(P)$ é o número de mutantes equivalente identificados.

2.4 Teste Baseado em Modelos

Teste Baseado em Modelos (TBM) refere-se à geração de casos de teste a partir de um modelo que representa um comportamento do software. Modelos comportamentais são desenvolvidos em fases iniciais em um ciclo de desenvolvimento, permitindo o início de atividades de teste antes da fase de codificação. Nessa técnica, testes são baseados em o que o software deve fazer, e não em o que o software faz (Apfelbaum e Doyle, 1997).

Modelos podem ser diagramas UML, especificações formais, como Máquinas de Estados Finitos (MEFs), ou ainda modelos arquiteturais. O que há em comum nessas representações é que, em testes baseados em modelos, a geração de CTs e a correção de defeitos tendem a ser menos complexas e custosas, pois os testes, em geral, são executados anteriormente à fase de codificação. Isso decorre em razão de um modelo possuir maior simplicidade conceitual e expressividade na descrição do comportamento de um sistema do que um código (Binder, 1999). Em outras palavras, tende-se ser mais simples, rápido e barato corrigir um modelo do que um código.

Além das MEFs, outros modelos formais são amplamente utilizados como as MEFEs, *Statecharts* e Redes de Petri. Esses modelos permitem um maior poder de representação, evitando a explosão de estados que ocorre com mais frequência nas MEFs. No entanto, em geral a validação de sistemas modelados por MEFEs, *Statecharts* e Redes de Petri é mais complexa em relação às MEFs.

2.4.1 Máquinas de Estados Finitos (MEFs)

MEFs são modelos hipotéticos compostos por um número finito de estados e transições (Gill, 1962). A cada instante, somente um estado da MEF está ativo. Geralmente, MEFs são representadas na forma de diagrama de transição de estado, que se trata de um grafo direcionado, no qual os estados são representados por vértices e as transições são representadas por arestas direcionadas. Em cada transição são rotulados os eventos (entrada), responsáveis pela mudança de estado.

Há dois tipos, comumente, utilizados de MEFs: máquina de Mealy (Mealy, 1955) e máquina de Moore (Moore, 1956). A principal diferença entre elas é onde se encontra a definição da saída. Na máquina de Mealy a saída é definida na transição, enquanto na máquina de Moore é no estado. Neste trabalho será utilizado apenas a máquina de Mealy.

Por se tratar do modelo utilizado neste trabalho, o capítulo 3 é dedicado para MEFs no qual os principais conceitos são detalhados.

2.4.2 Máquinas de Estados Finitos Estendidas (MEFEs)

MEFEs são MEFs incrementadas com o uso de condicionais ou predicados, utilizados para tornar as transições dependentes de variáveis (parâmetros) ou do contexto do atual do sistema. As variáveis podem assumir a função de controle (condição), de contador, ou de priorizar transições, dando pesos a elas. Outros recursos relacionados às MEFEs são: condição de guarda, parametrização e *clock*. Na Figura 2.2 ilustra-se um exemplo de MEFE. Na transição do estado S_2 para o estado S_3, há uma condição de guarda representada pela expressão $[u = 0]$ e uma parametrização representada pela expressão $z(u)$. Ou seja, se o evento b é acionado no estado S_2 e $u = 0$ a saída desse evento será z, o valor de z é igual ao valor de u e o estado destino é S_3; caso $u \neq 0$ a saída é x, o valor 0 é atribuído a u e o estado destino é S_0. A variável T representa o *clock* que é uma variável contínua ou discreta que representa o tempo e que está no sistema implicitamente.

A validação de MEFEs pode ser dividida em fluxo de controle e fluxo de dados. A validação do fluxo de controle é idêntica a uma MEF normal e pode-se utilizar os métodos de geração descritos nesta dissertação. Para validação do fluxo de dados, técnicas de simulação e análise de alcançabilidade, normalmente são empregadas por meio de ferramentas de apoio. A simulação pode ser realizada interativamente, em lote ou de forma exaustiva. As entradas são inseridas e o comportamento da MEFE pode ser observado posteriormente. Por meio da análise de alcançabilidade, pode-se verificar se alguma configuração desejada, ou não, é possível de ser atingida. Em (Mar-

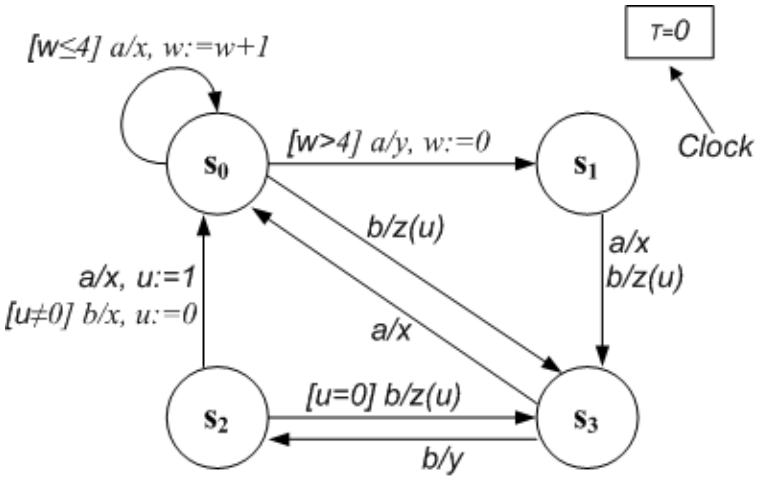

Figura 2.2: Exemplo de Máquina de Estados Finitos Estendidas. Adaptado de Petrenko et al. (2004)

tins et al., 1999), utiliza-se a técnica funcional de particionamento em classes de equivalência para geração dos CTs para o teste do fluxo de dados das MEFEs. Outros trabalhos procuram utilizar o conhecimento adquirido no teste estrutural, mapeando critérios de teste empregados no nível de programa para o nível de especificação (Bourhfir et al., 1999; Maloku e Frey-Pucko, 2001; Souza et al., 2001; Wong et al., 2008).

2.4.3 *Statecharts*

Statecharts foram introduzidos por Harel (1987) e também podem ser considerados extensões de MEFs, com adição de alguns recursos como: ortogonalidade, decomposição, *broadcasting* e história. A característica de decomposição permite criar uma hierarquia e uma modularização de estados. É possível criar diversos estados sendo tratados como um único superestado ou que um superestado seja decomposto em vários subestados chamados de estados ortogonais. As características de ortogonalidade e *broadcasting* permitem representar o paralelismo, ou seja, uma entrada pode gerar diversas saídas para estados diferentes. A característica de história guarda o último subestado atingido de um superestado. Na Figura 2.3 apresenta-se um exemplo de diagrama de *Statechart*, na qual é possível observar a transição x, válida para os estados ortogonais $C1$ e $C2$, partindo do estado A para o superestado C.

Assim como MEFEs, *Statecharts* também podem ser validados por meio de ferramentas de simulação e pela árvore de alcançabilidade (Masiero et al., 1994). A partir da Figura 2.3 é possível construir a árvore de alcançabilidade ilustrada na Figura 2.4. A árvore de alcançabilidade é construída da seguinte forma: o nó inicial representa a configuração inicial do *Statechart*. Para o exemplo da Figura 2.3, tem-se o estado A ativo. Para cada nó atingido, todas as possibilidades de entrada devem ser expandidas. Para a entrada x no estado A, o superestado C é ativado, assim como os subestados ortogonais C_1 e C_2 do superestado C. Também é ativado o subestado C_{11} do superestado C_1. Nesse caso, só é necessário ilustrar os subestados C_{11} e C_2 pois subentende-se que

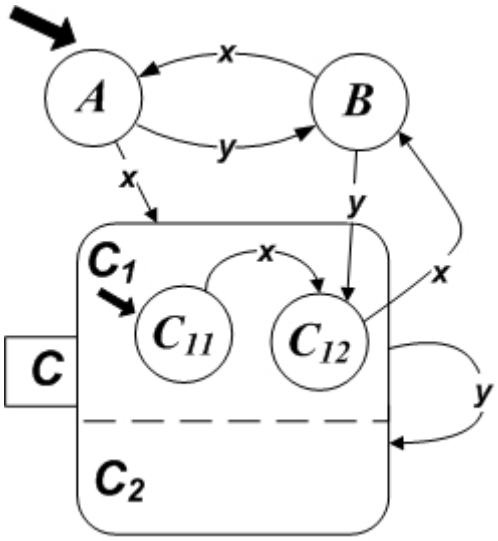

Figura 2.3: Exemplo de diagrama *Statechart*

um subestado ativo indica que seus superestados também estão ativos. A expansão acaba quando um nó, chamado de nó velho, já existente na árvore é alcançado. Na Figura 2.4, os nós velhos são marcados com um ‘ * ’.

É possível também produzir o grafo de alcançabilidade a partir da árvore de alcançabilidade. Se o grafo for finito, tem-se uma MEF, a qual os métodos e as técnicas relacionadas às MEFs podem ser aplicados. Como já dito, tais métodos serão apresentados no presente texto nos próximos capítulos. No entanto, em razão da explosão de estados, o grafo pode ser muito grande, ou infinito, o que dificulta a obtenção de uma MEF e, caso obtida, a aplicação dos métodos podem não ser viáveis. O grafo de alcançabilidade do *Statechart* representado na Figura 2.3 é ilustrado na Figura 2.5. Assim como para MEFEs, trabalhos como (Fabbri et al., 1999a; Souza, 2000) abordam técnicas de teste tais quais análise de mutantes e teste estrutural para validação de *Statecharts*.

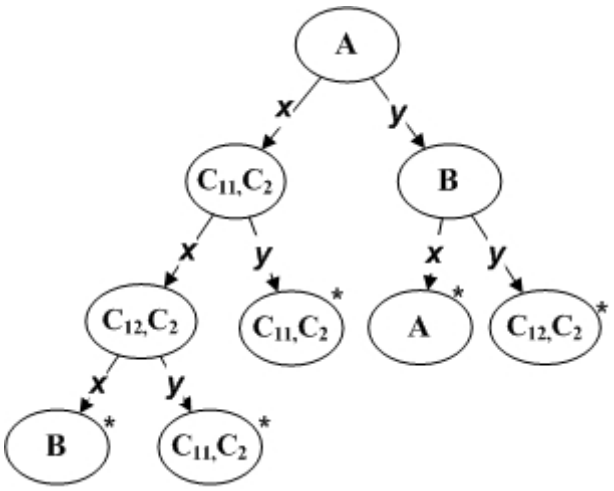

Figura 2.4: Árvore de alcançabilidade do *Statechart* representado na Figura 2.3

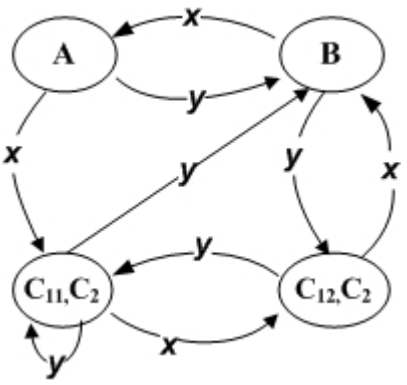

Figura 2.5: Grafo de alcançabilidade do *Statechart* representado na Figura 2.3

2.4.4 Redes de Petri

Redes de Petri foram introduzidas por Carl Adam Petri e são modelos em grafo que permitem representar sistemas de processamento de informação que não são bem descritos por outras técnicas de modelagem como: processos concorrentes, processos assíncronos, processos distribuídos, processos não-determinísticos e processos estocástico (Peterson, 1977). O grafo é composto por dois tipos de vértice: os lugares (representados por círculos) e as transições (representadas por barras). Os lugares e as transições são ligados por arcos direcionados. Um marcador, chamado *token*, que é representado por um ponto preto, simula a execução da rede. Os arcos podem possuir pesos que representam a quantidade de marcadores necessários para que a transição seja ativada, em caso do arco ir em direção a uma transição. Se o arco é direcionado a um lugar, o peso representa a quantidade de marcadores a serem inseridos nesse lugar caso a transição for ativada. Um diagrama de Redes de Petri é apresentado na Figura 2.6.

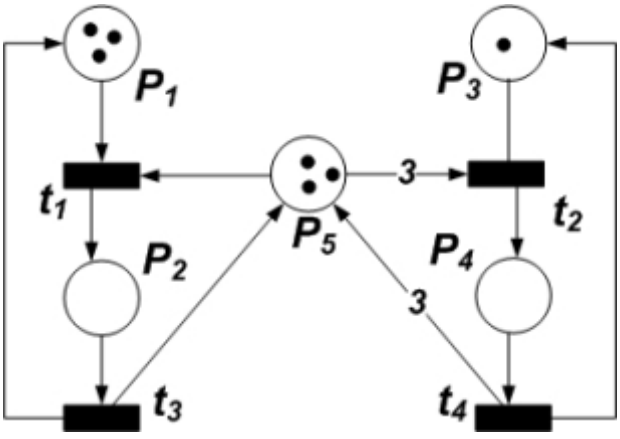

Figura 2.6: Exemplo de Rede de Petri

Assim como MEFEs e *Statecharts*, Redes de Petri também podem ser testadas por ferramentas de simulação e pela árvore de alcançabilidade (Peterson, 1977). A partir da Figura 2.6 é possível construir a árvore de alcançabilidade ilustrada na Figura 2.7. A árvore de alcançabilidade é construída da seguinte forma: o nó inicial é a marcação inicial da rede. Para o exemplo da Figura

2.6, tem-se a marcação inicial igual a <3 0 1 0 3>. Os próximos nós da árvore representam as marcações atingidas após o disparo das transições. Para a transição t_1, a marcação atingida será <2 1 1 0 2>. A expansão acaba quando um nó velho é alcançado ou quando não há transições a serem disparadas. Na Figura 2.7, os nós velhos são marcados com um ‘ * ’.

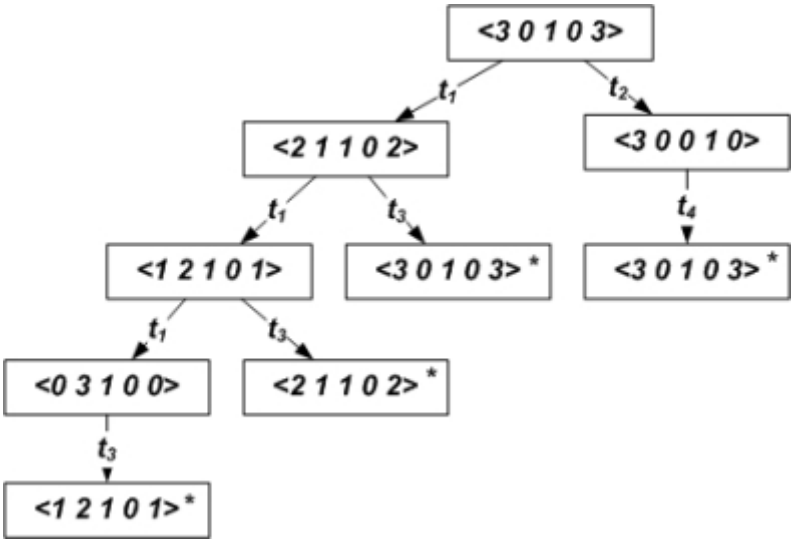

Figura 2.7: Árvore de alcançabilidade da Rede de Petri representada na Figura 2.6

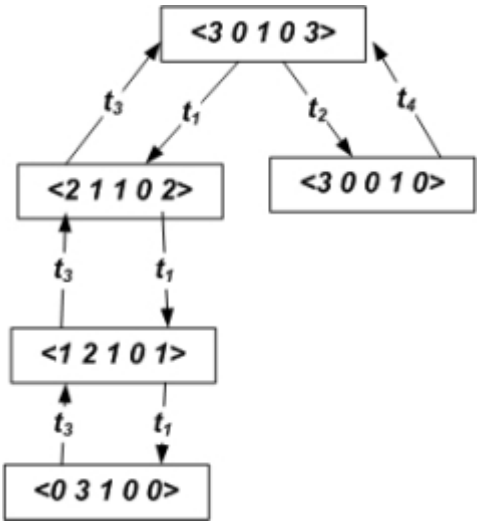

Figura 2.8: Grafo de alcançabilidade da Rede de Petri representada na Figura 2.6

É possível também produzir o grafo de alcançabilidade a partir da árvore de alcançabilidade. Assim como o grafo de alcançabilidade para *Statecharts*, se o grafo de alcançabilidade para Redes de Petri for finito, tem-se uma MEF, na qual os métodos e as técnicas relacionadas às MEFs podem ser aplicados. O grafo de alcançabilidade da Rede de Petri representada na Figura 2.6 é ilustrado na Figura 2.8. Além da árvore e grafo de alcançabilidade, também são utilizadas a árvore de cobertura e a matriz de incidência como técnicas de análise de Redes de Petri (Peterson, 1977). Assim como para MEFEs e *Statecharts*, é possível encontrar trabalhos na literatura que se baseiam em teste de mutantes ou teste estrutural para validação de Redes de Petri (Simão, 2004; Fabbri et al., 1995).

Na literatura, pode-se encontrar outros modelos, ou extensões dos citados. A modelagem de sistemas por meio de MEFs é muito utilizada tanto na academia como na indústria, em razão da sua facilidade de manuseio e à quantidade de pesquisas feitas na área. Quanto maior o poder de representação, menor é a facilidade de validar o sistema. Em razão de sua simplicidade, MEFs possuem uma gama de aplicação bastante grande e genérica, permitindo a modelagem de vários tipos de sistemas. Como dito anteriormente, há muitos trabalhos envolvendo MEFs e, em especial, teste baseado em MEFs.

Apesar da simplicidade das MEFs, a descrição do comportamento de um sistema por meio de MEFs nem sempre é trivial, tampouco barata. Estudos que visam à redução do custo em aplicar MEFs na Engenharia de Software são necessários. Apesar dos custos envolvidos, testes baseados em modelos formais, mais especificamente MEFs, são adequados em sistemas reativos de nível crítico, pois é essencial à garantia da qualidade e da segurança desses sistemas (Pressman, 2006). Nesses tipos de sistema o teste precisa ser sistemático (cobertura completa), focalizado (maior probabilidade de encontrar erros) e automatizado (consistente e reusável). Teste baseado em modelos alcança esses três objetivos (Binder, 1999).

2.5 Considerações Finais

Neste capítulo, foi apresentada uma fundamentação teórica sobre teste de software, atividade indispensável para garantia de qualidade de um software. Foi comentado que apesar das técnicas, ferramentas e métodos empregados no processo de desenvolvimento de software, erros ainda podem ocorrer no produto final. Por essa razão, testes são vitais para detectar erros durante o desenvolvimento e garantir que o software está de acordo com os requisitos especificados. Ainda assim, testes não garantem a ausência completa de erros, contudo um processo de teste sistemático, claro, rigoroso e formal propicia atividades de teste mais eficientes com um custo reduzido. Uma visão geral do processo de teste, suas fases e algumas técnicas utilizadas foram apresentadas no início deste capítulo. Enfatizou-se a técnica de teste baseada em modelos (TBM), na qual este trabalho está inserido. Exemplificaram-se alguns exemplos de modelos tais como: MEFEs, *Statecharts* e Redes de Petri e foram citadas algumas formas de validação desses modelos.

No próximo capítulo, discutem-se as características referente ao teste baseado em MEFs, contexto no qual este trabalho está inserido.

CAPÍTULO

3

Máquinas de Estados Finitos

3.1 Considerações Iniciais

Segundo Pressman (2006), existem diversas categorias de produto de software, tais como os sistemas reativos. A forma como sistemas reativos interagem com o ambiente nos quais estão inseridos favorece a utilização de de modelos formais em sua especificação. Não só sistemas reativos podem ter seu comportamento especificado por tais modelos, outros projetos inseridos na Engenharia de Software que utilizam o paradigma de programação orientada a objetos e sistemas distribuídos também utilizam modelos formais nas fases de análise de requisitos, modelagem e teste.

Por estar diretamente relacionada a este trabalho, o modelo formal representado por Máquina de Estados Finitos é apresentado neste capítulo. Na Seção 3.2, é apresentada a definição formal de MEF, assim como algumas de suas propriedades, conceitos utilizados neste trabalho e algumas sequências básicas. Na Seção 3.3, são descritos os tipos de erros que podem ser introduzidos na implementação de uma MEF.

3.2 Máquinas de Estados Finitos

MEFs são modelos utilizados para representar o comportamento de um sistema por um número finito de estados e transições (Gill, 1962). Modelos comportamentais são desenvolvidos em fases iniciais em um ciclo de desenvolvimento, permitindo o início de atividades de teste antes da fase de codificação (Apfelbaum e Doyle, 1997).

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

S: é um conjunto de estados, incluindo o estado s_0 chamado de estado inicial,

X: é um conjunto de símbolos de entradas,

Y: é um conjunto de símbolos de saídas,

$D_M \subseteq S \times X$: é o domínio da especificação,

λ: $D_M \rightarrow Y$ é uma função de saída e

δ: $D_M \rightarrow S$ é uma função de transição.

Uma MEF tem um estado ativado e em resposta a um evento de entrada pode ocorrer a mudança de estado e a produção de uma ação de saída. O novo estado e a saída produzida são definidos unicamente em função do estado ativado e da entrada. Portanto uma transição de estados é representada pelo par estado origem e entrada e pelo par saída e estado destino.

O número de estados (n), de entradas (k), de saídas (l) e de transições (t) de uma MEF são chamados de **parâmetros de MEF**. Na Figura 3.1 é apresentado um exemplo de MEF.

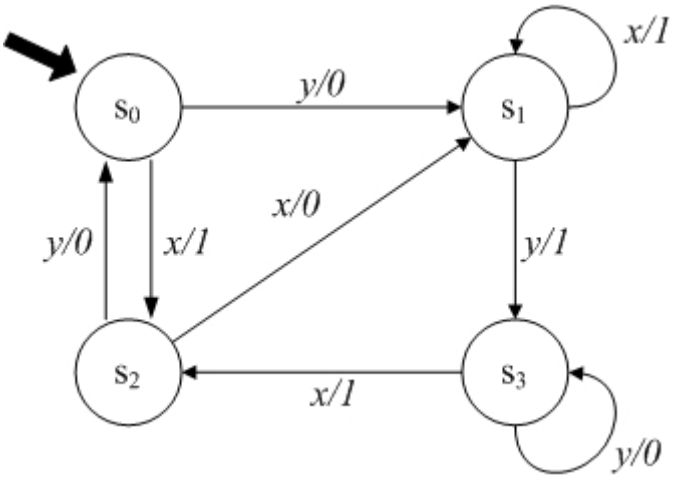

Figura 3.1: Exemplo de MEF. Adaptado de Dorofeeva et al. (2005a)

A máquina acima também pode ser representada pela notação da tupla: $M = (\{s_0, s_1, s_2, s_3\}$, s_0, $\{x, y\}$, $\{0, 1\}$, D_M, λ, $\delta)$, sendo $n = 4$, $k = 2$, $l = 2$, $t = 8$, $D_M = S \times X$ (ou $D_M = \{(s_0, x), (s_0, y), (s_1, x), ..., (s_3, y)\}$) e as funções de saída e transição representada na Tabela 3.1. A primeira coluna representa os estados origens e a primeira linha as entradas. Nos outros campos dessa tabela, o item antes da seta é a saída e o item depois da seta é o estado destino da transição.

λ -> δ	x /	y /
s_0 --	1 -> s_2	0 -> s_1
s_1 --	1 -> s_1	1 -> s_3
s_2 --	0 -> s_1	0 -> s_0
s_2 --	1 -> s_2	0 -> s_3

Tabela 3.1: Função de Saída e Transição

Para os conceitos descritos abaixo, supõe-se uma MEF $M = (S, s_0, X, Y, D_M, \delta, \lambda)$ com n estados, k entradas, l saídas e t transições e X^m é o conjunto das sequências de tamanho m e X^* é o conjunto das sequências de qualquer tamanho sobre X.

Uma sequência finita de entrada $\alpha = x_1...x_m \in X^*$ é chamada de sequência de entrada definida para o estado $s \in S$ se existem $s_i, ..., s_m, s_{m+1} \in S$, tal que $s_1 = s$, $(s_i, x_i) \in D_M$ e $\delta(s_i, x_i) = s_{i+1}$ para todos $i = 1, ..., m$. A notação $\Omega_M(s)$ representa o conjunto de todas as sequências de entrada definidas no estado s e a notação Ω_M representa o conjunto de todas as sequências de entrada definidas no estado inicial e, portanto, da especificação M. Uma sequência de entrada definida α de M é chamada de um caso de teste (CT) ou, simplesmente, teste.

Se uma sequência de entrada $\alpha.x \in \Omega_M(s)$ então $\delta(s, \alpha.x) = \delta(\delta(s, \alpha), x)$ e $\lambda(s, \alpha.x) = \lambda(s, \alpha).\lambda(\delta(s, \alpha), x)$.

A notação s_i -- x / y -> s_j indica que M no estado origem s_i responde com uma saída y e executa a transição para o estado destino s_j quando a entrada x é aplicada. Utiliza-se a notação s_i -- α -> s_j para indicar que a máquina M, encontrando-se originalmente no estado s_i, termina no estado s_j quando toda a sequência α é aplicada. Por exemplo, s_1 -- y / 0 -> s_2 e s_1 -- yyx -> s_2 são exemplos de notações válidas da MEF representada na Figura 3.1. Se existe $\alpha, \beta\gamma \in X^*$ e $\alpha = \beta\gamma$ é dito que a sequência β é um prefixo da sequência α.

Considerando dois estados distintos, s_i e s_j de M, diz-se que:

- São distinguíveis por uma sequência $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$ se $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Essa sequência α é chamada de **sequência de separação** entre os estados s_i e s_j de M. Na MEF apresentada na Figura 3.1 a sequência y é uma sequência de separação entre os estados s_1 e s_3, pois as saídas resultantes (1 e 0) são distintas.

- São equivalentes se para todo $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$ se $\lambda(s_i, \alpha) = \lambda(s_j, \alpha)$. Também existe equivalência entre máquinas. M é equivalente a outra MEF I caso seus estados iniciais sejam equivalentes.

- São k-distinguíveis se existe $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$, tal que $|\alpha| = k$ e $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Se são k-distinguíveis, também são l-distinguíveis, para $l \geq k$.

- São k-equivalentes se para todo $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$, tal que $|\alpha| = k$ e $\lambda(s_i, \alpha) = \lambda(s_j, \alpha)$. Se são k-equivalentes, também são l-equivalentes, sendo $l \leq k$.

Ainda supondo a MEF M, com relação as propriedades das MEFs, diz-se que:

- A máquina é não **determinística** se tiver ao menos um estado com duas ou mais transições definidas para o mesmo símbolo de entrada de X.
- M é **completamente especificada**, ou simplesmente **completa**, se $D_M = S \times X$, ou seja, para cada estado de M existe uma transição para cada símbolo de entrada em X. Do contrário, é dito que M é **parcialmente especificada** ou **parcial**.
- M é **inicialmente conexa** se para cada estado s_i existe uma sequência de entrada a qual leva M de s_0 para s_i. M é **fortemente conexa** se para cada par de estados (s_i, s_j) existe uma sequência de entrada que leva M de s_i para s_j.
- M é **minimal** quando M não possui estados equivalentes.

A MEF apresentada na Figura 3.1 é uma máquina determinística, completamente especificada, fortemente conexa e minimal. Ao contrário, na Figura 3.2 ilustra-se uma MEF não determinística, parcial, inicialmente desconexa e não minimal. É não determinística, pois s_4 possui duas transições para o símbolo de entrada y; é parcial, pois o estado s_2 não possui transição para o símbolo de entrada x; é inicialmente desconexa, pois é impossível atingir o estado s_3; e é não minimal, em razão dos estados s_0 e s_1 serem equivalentes.

As propriedades das máquinas são importantes para definição de métodos de geração e critérios de cobertura. Neste trabalho, as MEFs estudadas são máquinas de Mealy, com as propriedades determinística, completa, conexa e minimal em virtude da exigência dessas propriedades por parte de alguns métodos e critérios.

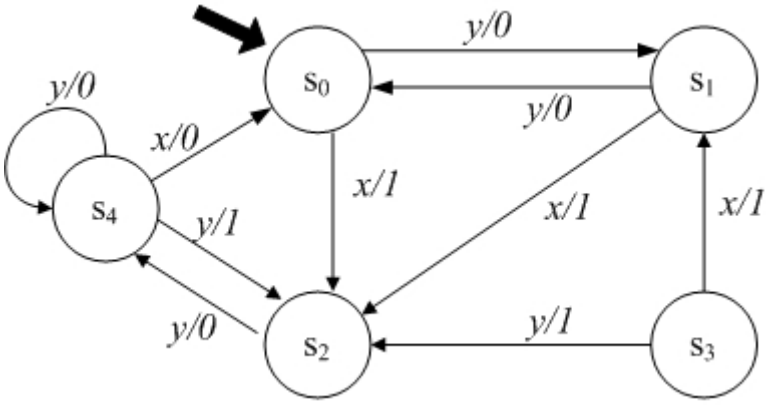

Figura 3.2: MEF não determinística, parcial, inicialmente desconexa e não minimal

Ao longo das últimas décadas, algumas sequências básicas de MEFs foram definidas. Tais sequências são muitas vezes utilizadas para obtenção de um resultado parcial importante na geração das sequências finais de teste (Simão e Petrenko, 2007). Em geral, mais de uma sequência básica pode ser encontrada em M e, comumente, é aconselhável selecionar a menor sequência

básica. Em alguns casos, o problema de encontrar a menor sequência básica é indecidível. Algumas heurísticas são utilizadas para escolha de qual sequência utilizar a depender do contexto. Neste trabalho, utilizou-se heurísticas nos algoritmos que tentam encontrar a menor sequência básica, mas nem sempre a sequência encontrada é a que gera melhores resultados. Tais heurísticas serão explicadas nos capítulos seguintes desta dissertação. As sequências básicas mais citadas na literatura são:

- **Conjunto *state cover*** (Q) de M é um conjunto de n sequências de entradas, incluindo a sequência vazia ϵ, que ao serem aplicadas, a partir do estado inicial, termina em cada estado de M.

- **Conjunto *transition cover*** (P) de M é um conjunto de entradas que exercitam cada uma das transições de M, partindo do seu estado inicial. O conjunto *transition cover* inclui o conjunto *state cover*, pois para atingir todas as transições é trivialmente necessário passar por todos os estados. Para exemplificar, considerando a MEF apresentada na Figura 3.1, são gerados os seguintes conjuntos *state cover* Q e *transition cover* P:

  - **Q** $= \{\epsilon, y, x, yy\}$
  - **P** $= \{\epsilon, y, x, yy, yx, xx, xy, yyx, yyy\}$

- **Sequência de sincronização** (SS) é uma sequência de entrada $\alpha \in X^*$ que faz M, partindo de qualquer estado, atingir um estado desejado. Geralmente, essa sequência, quando existe, não é única. Para a MEF representada na Figura 3.1, as seguintes sequências *xyxy, xx, xyx* e *xxy* são sequências SS para os estados s_0, s_1, s_2 e s_3 respectivamente.

- **Sequência de separação**, como visto, uma sequência de separação entre os estados s_i e s_j é uma sequência de entrada $\alpha \in \Omega_M(s_i) \cap \Omega_M(s_j)$ e $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$ para $s_i \neq s_j$. A seguir, serão explicadas as principais sequências de separação (ou conjunto de sequências) que distinguem um estado s_i de todos os outros estados da MEF.

- **Sequência de distinção** (DS) é uma sequência de separação $\alpha \in X^*$ em que a saída produzida por M em resposta a α é diferente para cada estado de M. Assim como a sequência SS, a sequência DS pode não existir e não é necessariamente única. A sequência yyy é uma sequência DS para a MEF representada na Figura 3.1. A sequência DS pode ser minimizada para cada estado de M, ou seja, os últimos símbolos de entradas da sequência DS podem ser eliminados, desde que após a minimização para um estado s_i, a sequência minimizada ainda consiga distingui-lo dos demais estados. Dessa maneira, a sequência reduzida DS_i do estado s_i sempre será um prefixo da DS. Nesse contexto, um conjunto das sequências DS_i da MEF representada na Figura 3.1 seria DS_0 = *{yy}*, DS_1 = *{y}*, DS_2 =*{yyy}*, e DS_3 = *{yyy}*.

- **Sequência única de entrada e saída** (UIO) é uma sequência de separação definida para cada estado s_i, cuja saída é diferente dos outros estados da MEF. Uma sequência DS é uma UIO para todos os estados. Contudo, em geral pode-se obter sequências UIO mais curtas para cada estado. Para a MEF ilustrada na Figura 3.1 as sequências UIO para os estados s_0, s_1, s_2, e s_3 seriam $UIO_0 = yy, UIO_1 = y, UIO_2 = x$ e $UIO_3 = yyy$, respectivamente. As sequências UIO também podem não existir para uma dada MEF.

- **Conjunto de caracterização** (W) é um conjunto de sequências de entrada que distingue o comportamento de cada par de estados da MEF especificada. Em outras palavras, são sequências que geram saídas distintas quando aplicadas a partir de estados distintos. O conjunto formado por todas as sequências UIO forma um conjunto W (o conjunto unitário formado pela sequência DS é um conjunto W com apenas uma sequência), mas em geral obtêm-se um conjunto W menor que a união das sequências UIO. Para a MEF da representada na Figura 3.1, $W = \{x, yy\}$.

- **Conjunto de identificação** (W_i) é um subconjunto do conjunto W definido para cada estado s_i de M que distingue o estado s_i dos demais. Um conjunto de identificação da MEF ilustrada na Figura 3.1 seria $W_0 = \{yy\}$, $W_1 = \{yy\}$, $W_2 = \{x\}$, e $W_3 = \{x, yy\}$. Assim como para a sequência DS, o conjunto W_i também pode ser minimizado por estado. Nesse caso, $W_1 = \{y\}$, pois basta a sequência y para diferenciar o estado s_1 dos demais e os outros W_i continuam os mesmos ($W_0 = \{yy\}$, $W_2 = \{x\}$, e $W_3 = \{x, yy\}$).

- **Conjunto de identificadores harmonizados** (H_i) é o conjunto união das sequências de separação de s_i para cada s_j de M, sendo $i \neq j$. Para a MEF representada na Figura 3.1 os H_i poderiam ser $H_0 = \{x, yy\}$, $H_1 = \{x, y\}$, $H_2 = \{x\}$, e $H_3 = \{x, yy\}$. O conjunto dos conjuntos H_i é chamado de conjunto HSI.

A função *reset* (representada por um 'r' no início de uma sequência de entrada) reinicia corretamente a implementação sob teste I, levando I ao seu estado inicial. Supõe-se que essa função é confiável para todas as implementações.

Um **domínio de erro** $\Im(X)$ é o conjunto de todas as implementações possíveis de M definidas sobre o alfabeto de entrada X. Similarmente, $\Im_n(X)$ denota o conjunto de todas MEFs completas definidas sobre o alfabeto de entrada X com n estados. Um conjunto de teste TS é um conjunto finito de casos de teste de M, no qual nenhum caso de teste é um prefixo do outro. TS é dito **n-completo** se para cada implementação $I \in \Im_n(X)$ que é distinguível de M, existe pelo menos uma sequência em TS que distingue M e I. Se TS possui uma única sequência, essa sequência é chamada de **sequência de verificação** (*checking sequence*).

3.3 Teste Baseado em Máquinas de Estados Finitos

MEFs são modelos baseados em estados que são utilizados intensamente, há anos, em testes de conformidade em circuitos digitais (Mealy, 1955; Moore, 1956), em testes de conformidade de protocolos de comunicação (Chow, 1978; Fujiwara et al., 1991), testes em sistemas reativos (Binder, 1999). Segundo Alberto e Simão (2009), a Engenharia de Software é favorecida por esse modelo formal na especificação do comportamento de sistemas reativos, que se torna mais precisa e menos sujeita a ambiguidades. Em todos os sistemas nos quais se pode modelar especificações por meio de MEFs, testes visam a garantir que uma MEF implementada atenda completamente à MEF especificada. De acordo com Chow (1978) uma MEF minimal M, que representa a versão correta da MEF (especificada), se comparada com uma MEF minimal I (implementada) podem ocorrer os seguintes erros:

- **Erros de saída**: quando I não é equivalente a M, e I pode ser modificada para se tornar equivalente a M, somente mudando a função de saída de I. Um exemplo de erro de operação da MEF ilustrada na Figura 3.1 é mostrado na Figura 3.3(a).
- **Erros de transferência**: quando I não é equivalente a M, e I pode ser modificada para se tornar equivalente a M, somente mudando a função de transição de I. Um exemplo de erro de transferência da MEF representada na Figura 3.1 é mostrado na Figura 3.3(b).
- **Erro de transição**: quando há um erro de saída, ou de transferências ou ambos.
- **Estados extras**: quando I não é equivalente a M, e I pode ser modificada para se tornar equivalente a M, diminuindo o número de estados de I.
- **Estados ausentes**: quando I não é equivalente a M, e I pode ser modificada para se tornar equivalente a M, acrescentando o número de estados de I.
- **Erro de estado inicial**: quando I não é equivalente a M, e I pode ser modificada para se tornar equivalente a M, modificando o estado inicial.

Uma forma de identificar os erros descritos acima seria aplicar o conjunto X^*, ou seja, todas as sequências de entradas possíveis e verificar se as saídas de I equivalem as saídas de M. Todavia, isso não é viável, pois o número de sequências possíveis é geralmente infinito. Diversos métodos de geração de CTs têm sido propostos para encontrar um conjunto de sequências suficiente para garantir a equivalência das MEFs. Tais métodos automatizam o processo de geração de CTs a partir de MEFs. O método ideal é aquele que gera o menor conjunto de teste e que seja suficiente para revelar todos os possíveis erros de uma implementação.

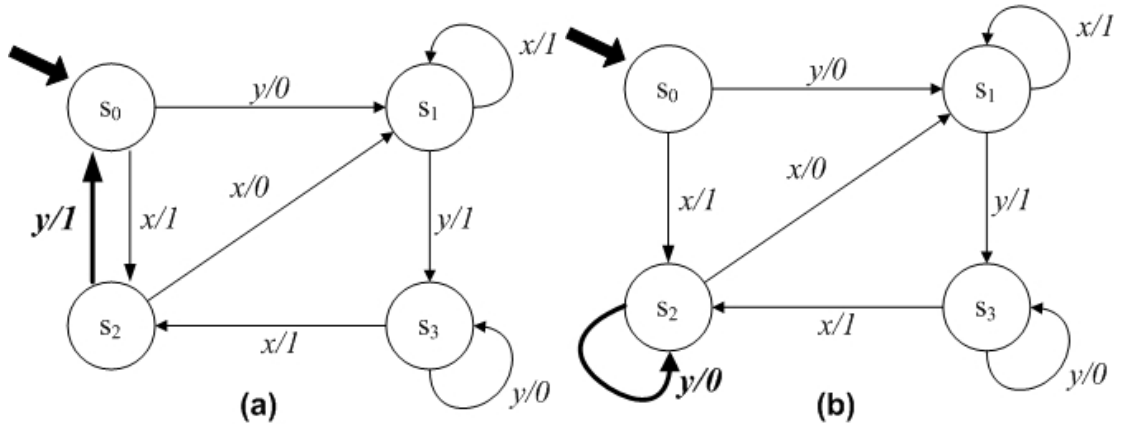

Figura 3.3: Erros de implementação

3.4 Considerações Finais

Neste capítulo, foi apresentada a definição de um modelo formal amplamente utilizado na Computação e na Engenharia de Software, as Máquinas de Estados Finitos. Modelos facilitam o entendimento do sistema, em uma fase do desenvolvimento na qual erros são localizados e corrigidos mais facilmente em relação a outras fases. Modelos e testes formais são imprescindíveis para sistemas de segurança crítica. A vantagem na utilização de MEFs está na facilidade de seu manuseio e na quantidade de pesquisas feitas na área. Outra vantagem das MEFs está na automatização em gerar CTs para sua validação.

A definição formal e os principais conceitos relativos à MEF foram apresentados. Dentre esses conceitos citaram-se: os parâmetros de MEF, as propriedades de MEF e algumas sequências básicas. Os tipos específicos de erros possíveis de ocorrerem em implementações baseadas em MEFs também foram explicados. Todos esses conceitos apresentados influenciam o teste baseado em MEFs e os métodos de geração e os critérios de coberturas.

No próximo capítulo, são apresentados os métodos de geração e critérios de cobertura de teste baseados em MEFs estudados neste trabalho, assim como algumas ferramentas de teste de MEFs, que utilizam um ou mais métodos. Também serão citados dois trabalhos relacionados que apresentam resultados de avaliações experimentais relativos a teste baseado em MEFs.

CAPÍTULO

4

Métodos de Geração e Critérios de Cobertura de Teste Baseado em MEFs

4.1 Considerações Iniciais

Especificar o comportamento de um sistema por meio de MEFs viabiliza a geração automática de casos de teste para o processo de teste de conformidade. Há décadas métodos de geração e critérios de cobertura são propostos para selecionar sequências de entradas (casos de teste) adequadas para o teste de implementações. Essas entradas são aplicadas nas implementações e as saídas são confrontadas com as que estão definidas a partir da MEF especificada. Caso as saídas sejam diferentes, um erro é detectado. Os métodos e critérios buscam selecionar um subconjunto de sequências de entradas que possua uma boa relação de custo/benefício.

Neste capítulo são apresentados os métodos de geração e critérios de cobertura de teste baseado em MEFs estudados neste trabalho. Basicamente, métodos de geração são algoritmos cuja a entrada é uma MEF e a saída é um conjunto de teste capaz de revelar erros na implementação dessa MEF. Critérios de cobertura definem requisitos de teste que devem ser cobertos por um conjunto de teste. Na Seção 4.2, são descritos os principais métodos de geração de CTs, que são os objetos de estudo deste trabalho. São apresentados os trabalhos que propuseram os métodos, as características, as melhorias estudadas, as propriedades e sequências exigidas de cada método, assim como a aplicação do método em um exemplo. Na Seção 4.3, são apresentados alguns critérios de cobertura

de MEFs, também utilizados nos experimentos deste trabalho. Uma tabela comparativa baseada nas propriedades de MEF e sequências básicas exigidas pelos métodos e critérios é ilustrada na Seção 4.4. Por fim, apresentam-se algumas ferramentas e alguns trabalhos sobre teste baseado em MEFs nas Seções 4.5 e 4.6 respectivamente. Nenhuma das ferramentas discutidas permite o apoio a experimentos de teste baseado em MEFs. Foram encontrados somente dois trabalhos que apresentam resultados de experimentos para o teste baseado em MEFs.

4.2 Métodos de Geração de Teste Baseado em MEFs

Os métodos de geração de teste baseado em MEFs têm o mesmo objetivo de verificar se a máquina implementada está de acordo com a especificação, mas diferem em relação ao custo, à eficácia dos casos de teste (CTs) e à aplicabilidade do método.

Na literatura, encontram-se muitos trabalhos que definem métodos de geração ou que propõe melhorias a métodos já existentes. Neste trabalho, fez-se uma classificação dividindo os métodos estudados em dois grupos. Os métodos que geram conjuntos n-completos e os que não geram conjuntos com essa característica. Serão chamados de métodos n-completos e métodos não n-completos, respectivamente. Os métodos estudados neste trabalho estão ilustrados na Tabela 4.1.

Métodos n-completos	**Métodos não n-completos**
W, Wp, UIOv, DS, CS, HSI, H, SC	TT, UIO, *Switch Cover*[1]

Tabela 4.1: Métodos de geração de teste baseado em MEFs

Diversos métodos de geração têm sido propostos como o método W que é considerado um dos métodos mais clássicos no contexto de geração de sequência de teste baseado em MEFs. Se a implementação da MEF gerar saídas corretas a partir das sequências de entrada geradas pelo método W, esta máquina está correta, pois o método é confiável para testar o comportamento modelado por uma MEF e detectar todos os possíveis erros descritos na Seção 3.3 (Chow, 1978).

Basicamente, outros métodos como o DS, UIO, UIOv, Wp, HSI e H são melhorias do método W. Segundo Dorofeeva et al. (2005a), todos esses métodos, exceto o UIO, garantem uma cobertura de erros completa e possuem duas fases. Na Fase 1, chamada de fase de identificação de estados, a geração de teste verifica se cada estado presente na especificação também existe na implementação, enquanto os testes da Fase 2, chamada de fase de identificação de transições, verifica se a saída e o estado final de cada transição da implementação estão em conformidade com a especificação. Para identificar os estados durante a Fase 1 e verificar os estados finais da Fase 2, os métodos utilizam sequências de separação de estado. Ainda segundo Dorofeeva et al. (2005a), a principal diferença dos métodos citados é como essas sequências são selecionadas. Os métodos TT, SC e *Switch Cover* não utilizam sequências de separação de estado.

[1] O método *Switch Cover* não é utilizado nos experimentos deste trabalho. Sua citação se deve a sua utilização em algumas ferramentas que são explicadas na Seção 4.5

Todos os métodos acima, com exceção dos métodos TT e CS, utilizam a função *reset*. Isso implica que o conjunto de teste pode ser composto por vários CTs individuais iniciados pela função *reset* que reinicia a máquina, garantindo que o estado inicial esteja ativo antes que cada CT seja aplicado.

Em geral, métodos que não utilizam a função *reset* exigem que a MEF seja fortemente conexa. Outros métodos também exigem que as MEF possuam determinadas propriedades, ou determinadas sequências básicas. Por exemplo, os métodos W e Wp são aplicáveis somente em MEFs completas, enquanto o método HSI é aplicável em MEFs parciais; já os métodos CS e DS exigem que a máquina possua uma sequência DS, e os métodos UIO e UIOv exigem que todos estados da máquina possuam sequências UIO.

Os métodos também se diferenciam em relação ao custo de execução do conjunto de teste gerado. Custo está associado ao número de caracteres de um conjunto de sequências.[2] Por exemplo, enquanto o método W gera um tamanho elevado de CTs que garantem a cobertura completa de falhas para MEFs, o método UIO resulta em um tamanho menor, mas não garante uma cobertura completa. Nas próximas seções, são apresentados com mais detalhes os métodos mencionados acima.

4.2.1 Método TT

Também chamado de método T, o método TT (*Transition Tour*) (Naito e Tsunoyama, 1981) é relativamente simples, comparado aos demais métodos apresentados. Segundo Sidhu e Leung (1989) uma ***transition tour*** é uma sequência de teste que pode ser gerada simplesmente aplicando entradas aleatórias em uma MEF, saindo do estado inicial, até passar por todas as transições pelo menos uma vez e retornando ao estado inicial. Entretanto, muitas sequências redundantes podem ser geradas e devem ser eliminadas por um procedimento de redução.

Esse método assume MEFs determinísticas, fortemente conexas e completamente especificadas e não necessita da função *reset*. Algumas implementações do método TT não exige que a MEF seja completamente especificada, porém é necessário que seja fortemente conexa.

Uma *transition tour* para a MEF apresentada na Figura 3.1 é TS_{TT} ={*xxxyxyyyyxy*} de tamanho 12.

Um dos problemas de difícil solução é encontrar a menor *transition tour*. Esse problema corresponde ao *problema do carteiro chinês misto* que pertence à classe dos problemas NP-Completo (Aho et al., 1988). A solução direta consiste em uma abordagem de força bruta e tentar todas as permutações possíveis, de modo a verificar a menor sequência que exercite todas as transições da MEF. Dado que a função para o cálculo das permutações é exponencial com relação ao número de transições, tal solução logo torna-se impraticável. Para contornar esse problema, heurísticas são utilizadas para encontrar uma solução próxima da melhor solução.

[2] O termo utilizado em inglês "*test suite length*" é traduzido para tamanho do conjunto de teste

A sequência gerada pelo método TT somente verifica a existência das transições e não testa se o estado final de uma sequência está correto. Dessa forma, este método só detecta os erros de operação e não garante a descoberta de todos os erros de transferência.

4.2.2 Método W e Método Wp

Como dito anteriormente, um dos métodos mais conhecidos para geração de CTs é o método W. Foi proposto por Chow (1978), chamado por ele de *Automata Theoretic*. Pode-se dizer que este método foi um dos percussores da área, visto que a maioria dos trabalhos seguintes se baseiam no método W, propondo algumas melhorias, pois o método W tende a gerar conjuntos de teste grandes.

O método W possui algumas restrições quanto às propriedades das MEFs. Somente é aplicável para MEFs determinísticas, completas, inicialmente conexas e minimais. Segundo Chow (1978), o conjunto de teste gerado por esse método garante a cobertura de todos os erros descritos na Seção 3.3.

O método W gera o conjunto de teste pela expressão $r.P.W$, onde o símbolo . significa a concatenação de todas as sequências do primeiro conjunto com todas as sequências do segundo conjunto. Portanto o conjunto de teste gerado pelo método W é produzido pela concatenação de elemento a elemento do conjunto *transition cover* P com o conjunto W, incluindo a função *reset* no início de cada CT gerado. Em (Chow, 1978), o conjunto P é denominado árvore de teste.

A aplicação do método W na MEF ilustrada na Figura 3.1 é descrita a seguir. Foi visto na seção 3.2 que $P = \{\epsilon$, *y, x, yy, yx, xx, xy, yyx, yyy}* e W =*{x, yy}*. Pela concatenação de P com W obtém-se as sequências *{x, yy, yx, yyy, xx, xyy, yyx, yyyy, yxx, yxyy, xxx, xxyy, xyx, xyyy, yyxx, yyxyy, yyyx, yyyyy}*.

O conjunto de sequências *{x, yy, yx, yyy, xx, xyy, yyx, yyyy}* podem ser eliminadas, pois são prefixos de outras sequências presentes no conjunto. Isso significa que, por exemplo, ao executar uma sequência como xxx, os resultados que seriam produzidos pelas sequências xx e x já são verificados.

Portanto, acrescentando a função r no início de cada sequência, o conjunto final com os casos de teste gerados é TS_W =*{ryxx, ryxyy, rxxx, rxxyy, rxyx, rxyyy, ryyxx, ryyxyy, ryyyx, ryyyyy}*de tamanho 49.

Fujiwara et al. (1991) apresentou um aprimoramento ao método W, que foi chamado de método Wp (*partial* W). A principal vantagem está na redução do tamanho do conjunto de teste gerado, pois ele não utiliza o conjunto W para verificar cada estado s_i da MEF, e sim um subconjunto deste. Como dito anteriormente, esse subconjunto W_i depende do estado s_i alcançado e foi chamado pelos autores de **conjunto de identificação**. O método Wp possui duas fases:

- **Fase 1**: inclui as sequências geradas por $r.Q.W$.
- **Fase 2**: inclui as sequências geradas por $r.R \otimes W$, onde $R = P - Q$ e $R \otimes W = \bigcup_{p \in R} \{p\} .W_i$, tal que i é definido pelo estado final da execução da sequência de p.

A heurística de minimizar o conjunto W_i por estado, explicados na Seção 3.2, geralmente é utilizada na implementação do método Wp, inclusive neste trabalho.

Segundo Fujiwara et al. (1991), caso o conjunto W_i contenha somente uma sequência simples (UIO) para cada estado s_i da MEF, o método Wp se torna equivalente ao método UIOv. Se essa sequência simples for igual para todos os estados o método Wp se torna equivalente ao método DS. Por esses termos, é dito que os métodos DS e UIOv são casos especiais do método Wp.

Os autores demonstraram que o método Wp gera um conjunto de teste com o mesmo poder de detecção de erros que o conjunto gerado pelo método W, porém esse conjunto é menor, pois a concatenação na segunda fase ocorre com um subconjunto W_i, ao invés de ocorrer com o conjunto W. Assim como o método W, para a sua aplicação o método Wp exige que a MEF seja determinística, completa, inicialmente conexa e minimal.

Para a MEF representada na Figura 3.1, o método Wp é dessa forma aplicado. Foi visto na seção 3.2 que $Q = \{\epsilon$, *y, x, yy}*, $P = \{\epsilon$, *y, x, yy, yx, xx, xy, yyx, yyy}*, $W =$*{x, yy}* e $W_0 = \{yy\}$, $W_1 = \{y\}$, $W_2 = \{x\}$, e $W_3 = \{x, yy\}$.

Na fase 1 do método Wp, obtém-se as sequências *{x, yy, yx, yyy, xx, xyy, yyx, yyyy}*.

Na fase 2, tem-se que: $R =$*{yx, xx, xy, yyx, yyy}* e realizando a operação $R \otimes W_i$ obtém-se as sequências na forma $\{yxW_1, xxW_1, xyW_0, yyxW_2, yyyW_3\}$. Realizando as substituições necessárias, as sequências obtidas são: *{yxy, xxy, xyyy, yyxx, yyyx, yyyyy}*.

Eliminando as sequências que são prefixos de outras e acrescentando a função r no início de cada sequência, o conjunto final com os casos de teste gerados é $TS_{Wp} =$*{ryxy, rxxy, rxyyy, ryyxx, ryyyx, ryyyyy}* de tamanho 29.

4.2.3 Método UIO e Método UIOv

Sabnani e Dahbura (1988) propuseram um método que eles chamaram método UIO, por utilizar as sequências UIO em substituição ao conjunto W. O método UIO é também referenciado na literatura como método U.

O conjunto de teste gerados pelo método UIO são menores, pois utilizam uma única sequência de separação para cada estado. Essa é a principal vantagem em relação ao método W. Os autores presumiam que o método UIO garantia uma cobertura completa de erros. Porém, em Vuong et al. (1989) foi apresentado um contra-exemplo que demonstrava um erro de transferência não detectado pelo conjunto de teste gerado pelo método UIO. Além disso, no trabalho de Dorofeeva et al. (2005a) foi demonstrado com resultados experimentais que 49% dos conjuntos de teste gerados

eram incompletos e que para algumas MEFs o método UIO não é aplicável. Este presente trabalho também apresenta resultados sobre a eficácia e a aplicabilidade desse método.

Assim como o método W, o método UIO só é aplicável para MEFs determinísticas, inicialmente conexas, completas e minimais. O método UIO gera o conjunto de teste incluindo as sequências geradas por:

- $r.P \otimes UIO$, onde $P \otimes UIO = \bigcup_{p \in P} \{p\}.UIO_i$, tal que i é definido pelo estado final da execução da sequência de p.

Nesse caso, $i = 0, 1, ..n-1$ e portanto cada uma das sequências UIO dos estado da MEF será utilizada ao menos uma vez, pois pelos menos uma transição atinge um estado da MEF, caso contrário a MEF seria desconexa. Caso um estado não possua sequência UIO, não é possível a aplicação desse método.

Para a MEF apresentada na Figura 3.1, a aplicação do método UIO é ilustrada a seguir. Foi visto na seção 3.2 que $P = \{\epsilon$, *y, x, yy, yx, xx, xy, yyx, yyy}* e $UIO_0 = \{yy\}$, $UIO_1 = \{y\}$, $UIO_2 = \{x\}$, e $UIO_3 = \{yyy\}$.

Concatenando P com as sequências UIO correspondente, obtém-se as sequências da forma: $\{\epsilon UIO_0$, *y*UIO_1, *x*UIO_2, *yy*UIO_3, *yx*UIO_1, *xx*UIO_1, *xy*UIO_0, *yyx*UIO_2, *yyy*UIO_3*}*, que resulta nas sequências *{yy, yy, xx, yyyyy, yxy, xxy, xyyy, yyxx, yyyyyy}*.

Eliminando as sequências que são prefixos de outras e acrescentando a função r no início de cada sequência, o conjunto final com os casos de teste gerados é TS_{UIO} =*{ryxy, rxxy, rxyyy, ryyxx, ryyyyyy}* de tamanho 25.

Algumas melhorias ao método UIO foram propostas como o método RCP (Aho et al., 1988), método MUIO (Shen et al., 1989), método MUIO com sobreposição (*overlapping*) (Yang e Ural, 1990), método B-UIO (Shen et al., 1991) e método C-UIO (Shen e Li, 1992). A semelhança desses métodos com o método UIO está na utilização da sequência UIO para gerar os CTs, além disso esses métodos não possuem uma cobertura de erros completa. Em Vuong et al. (1989), uma melhoria ao método UIO é proposta, que garante uma cobertura completa de erros chamada de método UIOv (UIO *variation*) ou simplesmente método Uv.

A solução encontrada pelos autores para tornar o método UIOv completo foi aplicar todas as sequências UIO dos outros estados ao visitar um estado s_i da MEF. Esse procedimento é referenciado pelos autores como **processo ∼Uv**. Dessa forma, é garantido que não existe uma mesma sequência UIO definida para outro estado da MEF. Portanto, a diferença do método UIO em relação ao método UIOv consiste na fase 1 do método UIOv, que é a concatenação das sequências de Q com as sequências UIO de cada estado da MEF:

- **Fase 1**: inclui as sequências geradas por $r.Q.UIO$
- **Fase 2**: inclui as sequências geradas por $r.P \otimes UIO$

Na fase 1, as sequências geradas pelo método UIOv para a MEF apresentada na Figura 3.1 são da forma: $\{\epsilon UIO_0$, ϵUIO_1, ϵUIO_2, ϵUIO_3, $yUIO_0$, $yUIO_1$, $yUIO_2$, $yUIO_3$, $xUIO_0$, $xUIO_1$, $xUIO_2$, $xUIO_3$, $yyUIO_0$, $yyUIO_1$, $yyUIO_2$, $yyUIO_3\}$, que resulta nas sequências *{yy, y, x, yyy, yyy, yy, yx, yyyy, xyy, xy, xx, xyyy, yyyy, yyy, yyx, yyyyy}*.

A fase 2 obtêm-se as mesmas sequências do método UIO.

Com a retirada das sequências que são prefixos de outras e acrescentando a função *reset* no início de cada sequência, o conjunto final com os casos de teste gerados é TS_{UIOv} *={rxxy, ryxy, rxyyy, ryyxx, ryyyyyy}* de tamanho 25. Coincidentemente, para esse exemplo os métodos UIO e UIOv apresentaram o mesmo TS. Por essa informação pode-se concluir que, para a MEF representada na Figura 3.1, o método UIO gera um conjunto n-completo.

Nesse exemplo, como visto, o conjunto de teste gerado pelo método UIOv não foi acrescido. Contudo essa nova abordagem, em geral, resultou em um aumento do tamanho dos conjuntos de teste.

4.2.4 Método DS e Método CS

Também encontrado na literatura escrito simplesmente como método D, como o nome já informa, os métodos DS e CS são baseados na sequência de distinção (DS) e, em geral, gera uma sequência de verificação (do inglês *checking sequence* por isso a abreviação de método CS). O primeiro trabalho a propor um método de geração de uma sequência de verificação foi Hennie (1965), seguido por Gönnenc (1970). Além da sequência DS, Hennie (1965) utiliza sequências SS para gerar uma sequência de verificação e a geração proposta por Gönnenc (1970) é baseada em grafos.

Segundo Gönnenc (1970), além da existência da sequência DS, o método DS só é aplicável em MEFs determinísticas, completas, fortemente conexas e minimais.

O método DS foi melhorado em trabalhos subsequentes ao de Gönnenc (1970). Trabalhos de Ural et al. (1997), Hierons e Ural (2002) Hierons e Ural (2006) foram desenvolvidos com o intuito de diminuir o tamanho da sequência de verificação. Em geral, esses trabalhos utilizam as condições de suficiência proposta em Ural et al. (1997) e modelagem teórica de grafos para minimizar o tamanho da sequência de verificação. Os resultados obtidos por Hierons e Ural (2006) demonstram uma redução de 25 a 40% do tamanho do tamanho da sequência de verificação em relação ao trabalho de Gönnenc (1970).

Neste trabalho, o método DS utilizado está descrito em Dorofeeva et al. (2005a). MEFs inicialmente conexas podem ser utilizadas por esse método e usa-se a função *reset*. O conjunto de teste é gerado em duas fases:

- **Fase 1**: inclui as sequências geradas por $r.Q.DS$.
- **Fase 2**: inclui as sequências geradas por $r.P \otimes DS$, onde $P \otimes DS = \bigcup_{p \in P} \{p\} .DS_i$, tal que i é definido pelo estado final da execução da sequência de p.

Considerando a MEF ilustrada na Figura 3.1 o método DS gera o conjunto TS_{DS} ={*rxxy, ryxy, rxyyy, ryyxyyy, ryyyyyy*} de tamanho 27.

No trabalho de Simão e Petrenko (2008), é proposto um método para a geração de sequências de verificação aplicável em MEFs completas ou parciais e que podem possuir, ou não, a função *reset*. Esse trabalho se baseia no conjunto de condições de suficiência proposto por Simão e Petrenko (2007) e realiza a melhor escolha local em cada passo. Os resultados obtidos apontam que, em média, houve uma redução de 45% do tamanho das sequências obtidas em Hierons e Ural (2006).

O método CS utilizado neste trabalho está descrito em Simão e Petrenko (2008) e exige que as MEFs sejam fortemente conexas. O método CS gera o conjunto TS_{CS} ={*yyyyyyxyyyxxyxxxyxyxyyy*} de tamanho 23

A grande desvantagem desses métodos consiste que somente um número limitado de MEFs possui sequência DS. Experimentos conduzidos por Dorofeeva et al. (2005a) mostraram que de um grupo de 1100 MEFs geradas aleatoriamente ($30 \leq n \leq 100$; $4 \leq k \leq 10$; $4 \leq l \leq 10$) somente 19% possuíam sequência DS. Este presente trabalho também apresenta um estudo da porcentagem de MEFs geradas aleatoriamente em que os métodos DS e CS são aplicáveis.

4.2.5 Método HSI e Método H

Igualmente aos outros métodos, é uma variação do método W cuja finalidade é reduzir o tamanho do conjunto de teste gerado. Proposto por Luo et al. (1994), a principal melhoria desse método é que ele é aplicável em MEFs parciais, enquanto que os outros métodos, em geral, necessitam que a MEF seja completa.

Um identificador de estado no método HSI é semelhante ao conjunto W_i do método Wp, ou seja, para cada estado um conjunto H_i é construído para distinguir o estado s_i dos demais. A diferença do conjunto W_i e H_i é a forma que são construídos. Enquanto o primeiro é um subconjunto do conjunto W, a construção do conjunto H_i é feita pela união das sequências de separação de cada par de estado da MEF.

O método HSI, como a maioria dos métodos abordados, consiste de duas fases.

- Fase 1: inclui as sequências geradas por $r.Q \otimes HSI$, onde $Q \otimes HSI = \bigcup_{q \in Q} \{q\}.H_i$, tal que i é definido pelo estado final da execução da sequência de q.
- Fase 2: inclui as sequências geradas por $r.P \otimes HSI$, onde $P \otimes HSI = \bigcup_{p \in P} \{p\}.H_i$, tal que i é definido pelo estado final da execução da sequência de p.

Como visto, para a MEF da Figura 3.1 Q = {ϵ, *y, x, yy*}, P = {ϵ, *y, x, yy, yx, xx, xy, yyx, yyy*} e os conjuntos H_i são: $H_0 = \{x, yy\}$, $H_1 = \{x, y\}$, $H_2 = \{x\}$, e $H_3 = \{x, yy\}$

Na fase 1, as sequências geradas são da forma: $\{\epsilon H_0, yH_1, xH_2, yyH_3\}$, que resulta nas sequências *{x, yy, yx, yy, xx, yyx, yyyy}*.

Na fase 2, as sequências geradas são da forma: $\{\epsilon H_0$, yH_1, xH_2, yyH_3, yxH_1, xxH_1, xyH_0, $yyxH_2$, $yyyH_3\}$. Fazendo as substituições obtém-se as sequências *{x, yy, yx, yy, xx, yyx, yyyy, yxx, yxy, xxx, xxy, xyx, xyyy, yyxx, yyyx, yyyyy}*

Retirando as sequências prefixos de outras e acrescentando a função *r* o método HSI gera o conjunto TS_{HIS} *={ryxx, ryxy, rxxx, rxxy, rxyx, rxyyy, ryyxx, ryyyx, ryyyyy}* de tamanho 41.

Baseado no método HSI e proposto por Dorofeeva et al. (2005b), o método H também utiliza os conjuntos H_i para identificação dos estados e pode ser aplicável em MEFs parciais e não determinísticas.

O método H possui o mesmo poder de detecção de erros que o método HSI, portanto esse método garante também uma cobertura total dos erros. A diferença entre esse método e o método HSI está na seleção dos identificadores de estados, que são construídos dinamicamente baseados nos CTs já gerados (*on-the-fly*). Partindo de um conjunto Q, o método analisa quais sentenças são necessárias para identificar cada estado.Os autores mostraram com experimentos que os conjuntos de teste gerados pelo método H são, em média, 66% do tamanho dos conjuntos gerados pelo método HSI. Nesse mesmo trabalho, os autores afirmaram que o tamanho do conjunto de teste depende da ordem de escolha das transições da MEF, portanto novas melhorias ainda podem ser incorporadas ao método, diminuindo ainda mais o tamanho médio dos conjuntos de teste gerados.

Considerando a MEF apresentada na Figura 3.1, o método H produz o conjunto TS_H *={rxxy, ryxy, rxyyy, ryyxx, ryyyyyy}* de tamanho 25.

4.2.6 Método SC

O método *State Couting* (SC), foi proposto por Petrenko e Yevtushenko (2005) atinge a mesma eficácia em detectar erros que o método W, mas é aplicável em MEFs parciais e não minimais.

Segundo os autores, a maioria dos métodos exige que todos os estados da MEF sejam distinguíveis. Ao contrário desses métodos baseados em identificação de estados, o método SC é aplicável a MEFs determinísticas mesmo que seus estados não sejam distinguíveis. O método baseia se em um algoritmo que expande as sequências de teste a partir de um estado da MEF até que seja atingida uma condição de parada, pois todos os erros já foram verificados. Por exemplo, se um estado é visitado mais do que n vezes, pode-se parar de expandir a sequência, uma vez que o comportamento começará a se repetir a partir desse momento.

Considerando a MEF representada na Figura 3.1, o conjunto TS_{SC} *={rxxx, rxxy, rxyx, ryxx, ryxy, ryyxx, ryyyx, ryyyyy}* de tamanho 41 é um conjunto de teste para o método SC.

4.2.7 Método *Switch Cover*

Descrito em Pimont e Rault (1976), o método *Switch Cover* (Sw) é baseado na sequência de Brujin (de Bruijn, 1946) utilizada na Teoria dos Grafos. Esse método usa uma abordagem de busca em profundidade e percorre todas as combinações de transições pelo menos uma vez. O método *Switch Cover* utiliza o conceito de árvore de transição (*transition tree*), na qual a raiz é o estado inicial, para cada transição é desenhado um galho e para cada próximo estado é adicionado um nó. Em caso de *loops*, é considerado apenas uma iteração. O nó folha é o estado inicial ou um estado já adicionado na árvore. Esse procedimento é repetido até todas as combinações de transições serem exercitadas.

Esse método é exaustivo, pois ele cobre todos os caminhos atingíveis pelo estado inicial, enquanto os demais métodos se baseiam somente nas transições (conjunto P). O conjunto de teste gerado por esse método cresce exponencialmente com o tamanho da MEF, contudo não requer que a MEF seja completamente especificada. O método *Switch Cover* não exige uma fase anterior para identificação dos estados (sequência de separação) como os demais métodos, como por exemplo o método W que necessita calcular o conjunto de caracterização W para posteriormente gerar os CTs.

Considerando a MEF representada na Figura 3.1, o conjunto TS_{Sw} =*{ryyxy, ryxyyxxyyyxy, rxy, rxxxxyyxxxyyxxxyyxy}* de tamanho 39 é um conjunto de teste para o método *Switch Cover*.

Apesar desse método ser utilizado em alguns trabalhos relacionados, ele não foi utilizado nos experimentos deste trabalho de mestrado. As sequências geradas por esse método, em geral, são muito longas e não garantem uma cobertura completa.

4.3 Critérios de Cobertura de Teste baseado em MEFs

Um critério de cobertura define um conjunto de requisitos que devem ser cobertos por um conjunto de teste adequado. Usualmente, esses requisitos são derivados de elementos do modelo que o testador considera importante para serem testados. Critérios de cobertura de teste são definidos baseados na especificação do software ou cobertura de erros. Em especificações de software modeladas por meio de MEFs, os critérios de teste podem focar em diversos elementos da MEF, como estados, entradas, saídas e transições. Em cobertura de erros, os critérios focam em possíveis erros da implementação da MEF, como erros ocorridos na escolha errada do estado inicial ou erros ocorridos em uma transição. Em Simão et al. (2009), são investigados quatro critérios de cobertura: cobertura de estados, cobertura de transições, cobertura de erro de estado inicial e cobertura de erro de transição.

No trabalho de Simão et al. (2009), utiliza-se o método HSI para gerar os conjuntos adequados a esses critérios. O conjunto gerado pelo método HSI é adequado para os quatro critérios, pois ele é n-completo, e os conjuntos adequados aos critérios não necessitam ser n-completo. Algoritmos de minimização são aplicados sobre o conjunto gerado pelo método HSI para selecionar um

conjunto de sequências adequadas para cobrir cada critério. Essa abordagem força a uniformidade da geração, evitando o impacto da utilização de diversos algoritmos de busca que podem gerar conjuntos de diferentes tamanhos.

Assim como o método HSI, todos os quatro critérios são aplicáveis em MEFs inicialmente conexas e parciais, porém as MEFs devem ser determinísticas e minimais.

4.3.1 Critério de Cobertura de Estados

Para o critério de cobertura de estados (S), assume-se que todos os estados da MEF devem ser alcançados como requisito de teste. Esse é o critério mais fraco dentre os estudados neste presente trabalho, pois não cobre todas as transições.

Para a MEF ilustrada na Figura 3.1, TS_S ={*rxxy*}, de tamanho 4, é adequado ao critério S, pois todos os estados são visitados por essa sequência.

4.3.2 Critério de Cobertura de Erro de Estado Inicial

Para o critério de cobertura de erro de estado inicial (IF), os requisitos de teste são definidos em relação aos estados erroneamente modelados como estado inicial. Para satisfazer esse critério, um conjunto de teste deve incluir sequências de separação entre o estado inicial e os demais estados da MEF. Portanto, para MEFs minimais um conjunto de teste adequado ao critério IF é adequado ao critério S.

Para a MEF apresentada na Figura 3.1, TS_{IF} ={*rxx, ryyyy*}, de tamanho 8, é adequado ao critério IF. De fato, se o estado s_2 fosse equivocadamente selecionado como estado inicial ter-se-ia que: a sequência de entrada *x* é uma sequência de transferência para o estado s_2 e *x* é uma sequência de separação entre o estado inicial s_0 e s_2. A sequência *rxx* de TS_{IF} cobre esse caso. Similarmente, a sequência *yyyy* satisfaz os requisitos relacionados aos estados s_1 e s_3.

4.3.3 Critério de Cobertura de Transições

Para o critério de cobertura de transições (T), assume-se que todas as transições da MEF devem ser exercitadas como requisito de teste. É fácil verificar que um conjunto adequado a T, também é adequado a S. Também verifica-se que esse conjunto detecta todos os erros de operação, mas não todos os erros de transferência. Esse critério se assemelha ao método TT, porém não há a obrigação das sequências de teste geradas retornarem ao estado inicial. E como foi visto, o método TT só funciona em MEFs fortemente conexas, o que não ocorre com o critério T, pois utiliza-se a função *reset* nesse critério.

Para a MEF representada na Figura 3.1, TS_T ={*rxy, rxxx, ryyyx*}, de tamanho 12, é adequado ao critério T, pois as 8 transições da MEF são exercitadas por essas 3 sequências.

4.3.4 Critério de Cobertura de Erro de Transição

Para o critério de cobertura de erro de transição (TF), os requisitos de teste são definidos a partir dos possíveis erros de uma transição, que podem ser uma saída inesperada ou/e um estado destino errado. O requisito de teste para esse critério é um par: uma transição (a qual é verificada a saída) e um estado (o qual é verificado o estado destino da transição). Portanto, para satisfazer esse critério, um conjunto de teste não basta somente cobrir as transições. Além disso, é necessário incluir uma sequência de separação entre o estado destino esperado com os demais estados. Para MEFs minimais esse critério cobre todos os outros acima mencionados.

Para a MEF ilustrada na Figura 3.1, TS_{TF} ={*rxxy, rxyyy, ryxy, ryyxx, ryyyx, ryyyyy*}, de tamanho 29, é adequado ao critério TF.

Exemplificando com a transição "s_1 -- y / 1 -> s_3", são necessárias sequências de separação entre o estado destino s_3 e os demais estados. A sequência *yy* cobre essa transição. Uma sequência de separação entre os estados s_3 e s_0 é *yy* que é seguida pela sequência que cobre essa transição (*yy*) na sequência *ryyyyy* de TS_{TF}. Uma sequência de separação entre os estados s_3 e s_1 é *y* que é seguida por *yy* na sequência *ryyyx*. E os estados s_3 e s_2 pode ser distinguidos pela sequência *x* que é seguida por *yy* na sequência *ryyxx*. Pode-se verificar que os requisitos relacionados às outras transições também são satisfeitos.

Pode-se dizer que os requisitos dos critérios S e IF são baseados nos estados da MEF, e os requisitos dos critérios T e TF são baseados nas transições da MEF. Como dito, o critério S exige que todos os estados sejam alcançados pelo conjunto de teste gerado e o critério IF exige que as possíveis trocas do estado inicial sejam detectadas. Da mesma forma, o critério T exige que todas as transições sejam alcançadas, e o critério TF exige que as possíveis trocas de saída ou de estado destino das transições sejam detectadas.

4.4 Comparação dos Métodos e Critérios

Para que o testador escolha um método de geração ou um critério de cobertura, com o objetivo de aplicá-lo em alguma especificação baseada em MEF, é necessário que algumas características sejam observadas. Essas características referem-se ao tamanho do conjunto de teste gerado, à eficácia de cada um e à aplicabilidade do método ou critério.

Na Tabela 4.2 é fornecida uma comparação entre os métodos e os critérios apresentados neste capítulo. Todos são aplicáveis em MEFs determinísticas, fortemente conexas, completas e minimais. Também são apresentadas outras propriedades de MEFs e sequências básicas exigidas pelos métodos e critérios.

Analisando a Tabela 4.2 as seguintes informações podem ser observadas:

- Dentre os métodos n-completos apresentados, o método SC é o único aplicável em MEFs não-minimais e que obtém um conjunto de casos de teste completo. De acordo com Petrenko

	Métodos											**Critérios**			
	TT	**W**	**Wp**	**UIO**	**UIOv**	**DS**	**CS**	**HSI**	**H**	**SC**	**Sw**	**S**	**IF**	**T**	**TF**
Não-minimal	X									X	X				
Parcial	X						X	X	X	X	X	X	X	X	X
Não-determinística									X		X				
Inicialmente Conexa		X	X	X	X	X		X	X	X	X	X	X	X	X
função r opcional	X						X								
sequência DS						X	X								
sequência UIO				X	X										
conjunto W		X	X												
conjunto HSI								X	X			X	X	X	X
n-completo		X	X		X	X	X	X	X	X					
tamanho TS	11	49	29	25	25	27	23	41	25	41	39	4	8	12	29

Tabela 4.2: Comparação entre os métodos de geração e critérios de cobertura

e Yevtushenko (2005), tem-se trabalhado para que esse método seja aplicável em MEFs não-determinísticas. Dos métodos n-completos estudados somente o método H é aplicável em MEFs não determinísticas.

- Os métodos TT, CS, HSI, H, Sw e SC e os critérios de cobertura são aplicáveis em MEFs parciais, sendo que o método TT, assim como os critérios, não são n-completo. O método UIO também não é n-completo.

- Apesar do método Sw (*Switch cover*) ser aplicável em diversos tipos de MEFs, ele, em geral, é o mais exaustivo dentre todos os métodos estudados e não gera conjuntos de teste n-completos.

- Para a aplicação dos métodos W e Wp, o conjunto de caracterização (conjunto W) deve existir, sendo que ele sempre existe em MEFs minimais. Os métodos CS e DS, e UIO e UIOv, para serem aplicáveis, ficam restritos à existência da sequência DS e de sequências UIO, respectivamente.

- Os métodos H e HSI utilizam o conjunto HSI na geração. Os critérios de cobertura utilizam o mesmo conjunto, pois neste trabalho o conjunto de teste adequado aos critérios é minimizado do conjunto gerado pelo método HSI.

- Os métodos TT e CS não exigem a utilização da função *reset* e somente são aplicáveis em MEFs fortemente conexas. Ainda sobre esses métodos, foram gerados os menores conjuntos de teste para a MEF apresentada na Figura 3.1 (última linha da Tabela 4.2), sendo que o método TT não é um método n-completo. O método W foi o pior em relação ao custo dos CTs gerados.

4.5 Ferramentas de Apoio a Teste Baseado em MEFs

A utilização de ferramentas de apoio torna-se essencial para que a atividade de teste seja automatizada. Boas ferramentas permitem ao testador um aumento da produtividade durante a atividade de teste e diminui a probabilidade de erros humanos. No contexto deste trabalho, ferramentas de apoio ao teste baseado em MEFs, além das vantagens mencionadas, possibilita a utilização de MEFs complexas, facilita a aplicação de algoritmos complexos, bem como trabalhar com amostras significativas de MEFs (na ordem de 10^7), o que é inviável de ser tratado manualmente por questões de complexidade e tempo.

A seguir, descrevem-se algumas ferramentas que se relacionam ao teste baseado em MEFs, dentre elas a já citada Plavis/FSM, que será utilizada como ferramenta de apoio neste trabalho na fase de experimentação.

4.5.1 Ferramenta Plavis/FSM

O projeto PLAVIS (*PLAtform for software Validation & Integration on Space Systems*) (PLAVIS, 2005) foi uma parceria composta pelas instituições brasileiras: Instituto Nacional de Pesquisas Espaciais (INPE), Universidade de São Paulo (USP), Universidade de Campinas (Unicamp), Universidade Federal de São Carlos (UFSCar), Universidade Federal de Minas Gerais (UFMG), Centro Universitário de Eurípides de Marília, Universidade de Ponta Grossa (UEPG) e pelas instituições francesas: *Institut National dés Télécommunications* (INT) e *Université de Bordeaux* com apoio da CAPES, COFECUB e CNPq.

O projeto objetivou a criação de uma plataforma para apoio às atividades de VV&T, fornecendo um ambiente que possibilitasse o processo de validação, verificação e testes de softwares espaciais. Em princípio, algumas ferramentas foram incorporadas no projeto PLAVIS, as quais utilizam especificações baseadas em MEFs e testes de mutantes (Simão et al., 2005). Dentre essas ferramentas estão a Protem/FSM (Fabbri et al., 1999b), a ConData (Martins et al., 1999) e a MGA-Set (Candolo et al., 2001). Dentre os resultados do projeto PLAVIS, foi desenvolvida a ferramenta Plavis/FSM, que promove a integração das ferramentas citadas.

A Plavis/FSM provê um conjunto mínimo de operações de testes baseados em MEFs, focada no critério de teste análise de mutantes. Dentre as principais funcionalidades encontram-se a manipulação de casos de teste (geração, execução, inclusão/exclusão, habilitar/desabilitar, importar) e a manipulação de mutantes (criação, seleção, execução e análise). É possível criar um projeto e, dentro desse projeto, várias sessões. Inicialmente, uma MEF é projetada, podendo ser criada manualmente, utilizando o editor de MEFs da ferramenta ou importando-a. Em seguida, são gerados os mutantes dessa MEF, por meio dos operadores de mutação selecionados pelo usuário. Para cada sessão criada, o usuário gera (manualmente ou utilizando um dos métodos implementados) um conjunto de teste. Por fim, o usuário pode executar os conjuntos de teste em alguns mutantes selecionados ou em todos, e analisar os resultados. Alguns métodos de geração já estão integrados

na Plavis/FSM, como os métodos W, *Switch Cover*, HSI, UIO e SC. A Plavis/FSM foi desenvolvida em plataforma *web* e iniciativas de melhorias estão sendo implementadas, inclusive por este trabalho.

As próximas seções descrevem sucintamente as ferramentas que integram a Plavis/FSM, que são a Proteum/FSM, MGASet e Condado.

- Uma das ferramentas que integram a Plavis, a Proteum/FSM (*Program Testing Using Mutants/Finite Sate Machines*) (Fabbri et al., 1999b) é um componente de uma família de ferramentas (Maldonado et al., 2000) que dá apoio à aplicação de testes de mutantes em vários contextos como teste de código (Delamaro et al., 2001) e teste de especificação (Sugeta et al., 2001).

  Assim como a Plavis/FSM, a Proteum/FSM permite editar MEFs, manipular um conjunto de mutantes e de casos de teste. A execução da ferramenta é iniciada após a inclusão e confirmação dos CTs e geração dos mutantes, utilizando operadores de mutação (Fabbri et al., 1994). Cada CT é executado e aplicado para cada mutante e, por fim, o testador verifica se os mutantes ainda vivos são equivalentes; caso contrário, novos CTs precisam ser incluídos. A Proteum/FSM também é uma ferramenta em plataforma *web*.

- Outra ferramenta, a MGASet (Módulo de Geração de Sequencias de Teste) (Candolo et al., 2001) utiliza o método W para a geração dos CTs. Contudo, foi projetada para trabalhar com vários métodos de geração. A MGASet permite a edição e simulação de MEFs gerando um conjunto de teste para validação da MEF implementada. Incorpora a funcionalidade de verificação de propriedades de MEFs, tais como se a máquina é fortemente conexa, completa, determinística e minimal. Outra saída da ferramenta é a geração de sequências básicas, como a sequência DS, a sequência UIO e o conjunto W. Foi desenvolvida em Java, e pode ser executada em uma interface gráfica (GUI) ou por invocações em linha de comando.

- Por fim, a terceira ferramenta integrada no projeto Plavis, a ConData (Martins et al., 1999) foi desenvolvida para geração de CTs baseados na especificação. Além de MEFs, também trabalha com MEFEs, e os testes gerados seguem uma classificação: dados ou controle.

  A parte de dados trabalha com os valores dos parâmetros das interações e são realizados teste de sintaxe e particionamento de classe de equivalência para a geração de CTs. A parte de controle é direcionada para validar sequências de interações que são geradas baseadas em caminhos que começam e terminam no estado inicial. O método utilizado para geração de CTs é o *Switch Cover*. Como foi visto, esse método é mais exaustivo que os métodos tradicionais, pois o preço para garantir uma cobertura completa exige muitos CTs e cresce exponencialmente com o tamanho da MEF. Para lidar com esse problema, estratégias devem ser elaboradas para definir objetivos de teste que não consideram o comportamento completo do sistema, chamadas de teste seletivo. Um exemplo de teste seletivo é restringir quais

e quantas vezes determinadas transições serão exercitadas. ConData foi desenvolvida em Prolog.

Na Figura 4.1 apresenta-se a arquitetura da ferramenta Plavis/FSM. A integração promovida pela Plavis/FSM é baseada nos dados. Ela converte a representação interna das MEFs (estrutura de dado) para cada formato de entrada das ferramentas integradas e vice versa. A Plavis/FSM prepara os dados e invoca a respectiva ferramenta, em um ambiente controlado. Os resultados de cada ferramenta são armazenados em um arquivo particular e convertido para a representação interna da Plavis/FSM. O núcleo dela é o módulo **gerenciador de casos de teste**, responsável pela comunicação dos módulos e processar as entradas do usuário e retornar as saídas a ele. Para interação com as ferramentas integradas o gerenciador delega a tarefa a dois **adaptadores**. Um adaptador interage com as ferramentas na geração de casos de teste e o outro interage na análise de adequação dos casos de teste. Para a reusabilidade dos adaptadores, eles somente provem uma interface pública para as ferramentas. Por se tratar de ferramentas de código legados, escritas em diversas linguagens de programação, não é uma tarefa simples modificar seus códigos para integração direta com os adaptadores. Assim, *wrappers* podem ser desenvolvidos com a finalidade de satisfazer os requisitos de entrada e conversão de dados das ferramentas integradas. Desse modo, as interfaces de usuário de cada ferramenta podem ser descartadas o que aumenta a extensibilidade da plataforma, ou seja, a integração de novas ferramentas. Essa obrigatoriedade de desenvolver *wrappers* para aumentar a extensibilidade justifica a utilização de *web services* na reengenharia dessa ferramenta que será explicada na Seção 5.2.

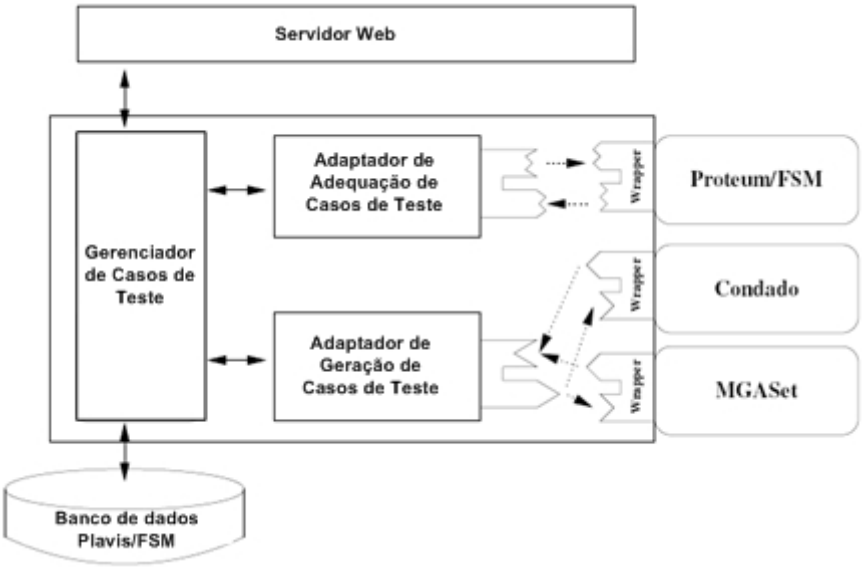

Figura 4.1: Arquitetura da Plavis/FSM. Adaptado de Simão et al. (2005)

4.5.2 Ferramenta TAG

A ferramenta TAG (*Test Automatic Generation*) (Tan et al., 1996), como o nome já diz, é uma ferramenta para geração automática de CTs para especificações na forma de máquinas de estados. A ferramenta utiliza o método HSI, portanto pode ser aplicada a MEFs parciais e completas. Para MEFs que possuem estados não distinguíveis (MEF não-minimal) é possível a geração, porém o conjunto de teste gerado não garante a cobertura completa de defeitos. Assim como a ferramenta ConData, é possível gerar testes seletivos na TAG. O usuário tem a opção de gerar um conjunto de CTs que testa todas as transições da MEF ou de apenas uma única transição. A ferramenta implementa uma heurística para obter, na maioria das vezes, o menor conjunto HSI, o que reduz o tamanho do conjunto de teste gerado. Segundo Tan et al. (1996), foram conduzidos alguns experimentos que indicaram que em 87% dos casos a ferramenta foi capaz de gerar o menor conjunto HSI existente.

4.5.3 Ambiente GTSC

O ambiente GTSC (Geração automática de casos de Teste baseada em *StateCharts*) (Santiago et al., 2008) é uma ferramenta desenvolvida no INPE, que permite ao testador modelar o comportamento de um software por meio de MEFs ou *Statecharts* para geração automática de testes baseados em alguns métodos de geração de MEFs e critérios de teste para *Statecharts*. Os métodos de geração utilizados são *Switch Cover*, DS, UIO e TT e os critérios de teste para *Statecharts* são baseados na família SCCF (Souza, 2000).

Por meio de uma linguagem de especificação chamada PcML, em geral, é possível converter um *Statechart* em uma MEF com a utilização de *scripts* escritos na linguagem Perl e da ferramenta PerformChats (Santiago et al., 2006). Uma vez obtida a MEF, o projetista de teste pode escolher em gerar os CTs entre os métodos de geração ou os critérios de teste baseado em *Statecharts*. Para a geração de CTs pelo método *Switch Cover*, o ambiente utilizava a ferramenta ConData, contudo esse método foi re-implementado e integrado no ambiente (Ferreira et al., 2008). Para os demais métodos e critérios citados, os autores também desenvolveram novas implementações e embutiram no ambiente.

Em Santiago et al. (2008) são apresentados dois estudos de caso para mostrar a utilidade do ambiente desenvolvido, que envolve softwares embarcados para sistemas espaciais desenvolvidos no INPE.

4.6 Avaliação Experimental de Teste Baseado em MEFs

Como já fora dito, são poucos os trabalhos que relatam avaliações experimentais em termos de custo e eficácia para teste baseado em MEFs. Nesta seção, são abordados dois trabalhos encontrados na literatura que apresentam resultados de experimentos para teste baseado em MEFs que são Dorofeeva et al. (2005a) e Simão et al. (2009).

Segundo Wong et al. (1995), três características principais são utilizadas para avaliação experimental de critérios de teste: custo, eficácia e *strength*. O custo corresponde ao esforço da aplicação e, em geral, é medido pelo tamanho do conjunto de CTs gerado. A eficácia, ou efetividade, equivale à capacidade de detecção de erros. *Strength*, traduzido em dificuldade de satisfação, diz respeito à probabilidade de um conjunto de CTs adequado a um critério satisfazer outro.[3] Neste trabalho, não será avaliado o *strength* dos métodos de geração e dos critérios de cobertura, somente o custo e a eficácia dos métodos e critérios não n-completos.

No trabalho Dorofeeva et al. (2005a) é apresentada uma avaliação experimental, comparando os métodos W, Wp, UIO, UIOv, DS, HIS e H. Para condução dos experimentos foram geradas aleatoriamente 1.100 MEFs completas e mínimas divididas em 22 grupos. Para cada um desses grupos os parâmetros de MEF n, k, l, t variavam. Para cada método foi calculado o tamanho do conjunto gerado por cada método e o tempo de geração. Também foi calculado quantas vezes os métodos UIO e DS foram aplicáveis.

Os resultados mostram que o método W gera, em média, conjuntos de teste maiores que os outros métodos. O tamanho médio dos conjuntos de teste dos métodos Wp e HIS quase coincidem, com o segundo tendo um tempo de geração um pouco mais eficiente. Os conjuntos menores são gerados pelos métodos UIO, DS e H, nessa ordem, sabendo que os métodos UIO e DS são aplicáveis em 99% e 19% dos casos, respectivamente. O método UIOv não é melhor em relação aos métodos Wp e HIS. Isso era esperado, pois o UIOv é um caso particular do Wp. Os conjuntos de teste gerados pelos métodos Wp, HIS, UIOv, H, UIO, DS são em média 61%, 61%, 65%, 40%, 27%, 36%, respectivamente, do tamanho dos conjuntos gerados pelo método W.

Outro experimento foi conduzido para mensurar a capacidade de detecção de erros do conjunto de teste gerado pelo método UIO. Para cada MEF gerada aleatoriamente foi gerado um conjunto de teste pelo método UIO (quando aplicável) e todas as possibilidades de implementação erradas foram explicitamente derivadas[4]. Depois, determinam-se quantas dessas implementações são "mortas", ou seja, se algum erro é detectado pelo conjunto de teste gerado. Esse experimento mostra que 41% dos conjuntos de teste gerados pelo método UIO não são n-completos.

[3] Wong et al. (1995) avalia critérios de teste estrutural e o critério análise de mutantes. Neste presente trabalho, instancia-se alguns conceitos de avaliação de critérios de teste presentes em Wong et al. (1995) para os critérios de cobertura e para os métodos de geração estudados neste trabalho

[4] No artigo não é descrito de forma explícita como essas implementações foram derivadas

No trabalho de Simão et al. (2009) são apresentados resultados de uma avaliação experimental de vários critérios de cobertura utilizados comumente em testes de MEFs. Como visto, esse trabalho define quatro critérios de cobertura: S, T, IF e TF.

Os resultados mostram que o tamanho médio do conjunto de teste é bem menor que o pior caso e não cresce tão rapidamente como o limite superior sugere. Evidentemente, é demonstrado que para critérios de cobertura mais poderosos (TF) são necessários conjuntos de teste maiores que critérios de cobertura mais fracos (S).

Outro experimento conduzido mediu a característica de *strength* entre os critérios. A partir de um conjunto de teste adequado a um critério é calculado a cobertura dos requisitos para os outros critérios. Como esperado, o conjunto de teste adequado para o critério T é 100% adequado para o critério S e o conjunto de teste adequado para o critério TF é quase sempre adequado a qualquer outro dos critérios de cobertura apresentados.

Conduziram-se outros experimentos para entender como os parâmetros de MEF interferem no tamanho do conjunto de teste. Com esses experimentos observa-se que os critérios S e IF praticamente não são afetados pela variação desses parâmetros. Para os critérios T e TF, o tamanho médio dos conjuntos de teste aumenta quase que linearmente com o aumento de k e t. Para o critério TF, o tamanho dos testes diminui com o aumento de l, enquanto os demais critérios não são afetados.

Ambos os trabalhos (Dorofeeva et al., 2005a; Simão et al., 2009) estão inseridos no contexto de teste baseado em MEFs e buscam identificar alguns resultados nessa área, mas possuem metodologias diferentes. Dorofeeva et al. (2005b) avalia experimentalmente métodos de geração, enquanto Simão et al. (2009) avalia critérios de cobertura. Os dois trabalhos analisam o limite superior do crescimento da curva do tamanho médio dos conjuntos de teste n-completos com relação ao aumento do número de estados em notação O. Teoricamente esta curva é da ordem $O(kn^3)$, mas na prática com os dados obtidos por Dorofeeva et al. (2005b) esta curva cresce na ordem de $O(cn^2)$, com c = 4 (c = 5 para o método W). Em contrapartida os dados obtidos em Simão et al. (2009) sugerem que esta curva cresce mais lentamente que $O(kn^2)$.

4.7 Considerações Finais

Neste capítulo foram apresentados os métodos de geração e os critérios de cobertura. Diversos métodos de geração de CTs foram abordados, inclusive trabalhos recentes foram comentados, o que mostra o interesse de grupos de pesquisa nessa área. Neste capítulo também foram discutidas as condições de suficiência, ferramentas de teste baseado em MEFs e dois trabalhos que apresentam resultados de experimentos referentes ao teste baseado em MEFs.

As atividades de teste necessitam de ferramentas para automatizar o processo e diminuir a probabilidade de erros humanos, quando executadas de forma manual. Pelos mesmos motivos, atividades de teste baseado em modelos, tais quais as MEFs, necessitam também de ferramentas.

Foram apresentadas algumas ferramentas para teste baseado em MEFs, com destaque à ferramenta Plavis/FSM, alvo da reengenharia de um protótipo que permita apoio a experimentos. Em geral, todas as ferramentas trabalham com MEFs (e outros modelos) e com um ou mais métodos de geração. Nenhuma das ferramentas citadas dão apoio a experimentos utilizando os métodos de geração. Permitir que o usuário planeje e conduza experimentos com o auxílio de uma ferramenta é um *gap* encontrado nessa área, ou seja, não foram identificadas na pesquisa uma ferramenta de apoio a experimentos de teste baseado em MEFs. Um dos objetivos deste trabalho é desenvolver um protótipo por meio de uma reengenharia da Plavis/FSM, incluindo essa funcionalidade não encontrada nas ferramentas descritas.

Como foi dito, modelagem de sistemas por meio de MEFs são amplamente utilizados na indústria (Santiago et al., 2008; Peralta, 2009) e para validar esses modelos, métodos de geração de CTs podem ser usados. Todavia, ainda há poucos relatos sobre as reais vantagens e desvantagens dos métodos e critérios em aplicações reais, em termos de eficácia e de custos envolvidos. A validação desses modelos depende dos objetivos de teste dos desenvolvedores e do contexto em que o software está inserido. O objetivo deste trabalho é avaliar essa relação de custo/benefício para que estratégias de teste sejam definidas e executadas da forma planejada.

Foi identificada uma contribuição para essa linha de pesquisa, que é o desenvolvimento de uma ferramenta de apoio a experimentos. Decidiu-se estudar o ambiente Plavis/FSM para realizar uma reengenharia e construir uma ferramenta com essa função. No próximo capítulo, discute-se essa reengenharia.

CAPÍTULO

5

Reengenharia da Ferramenta Plavis/FSM

5.1 Considerações Iniciais

Apesar de existirem ferramentas de apoio a teste baseado em MEFs, nenhuma das ferramentas encontradas oferece apoio à condução de experimentos para comparar os métodos e critérios. A ferramenta Plavis/FSM promove a integração de algumas ferramentas de teste baseado em MEFs, contudo não dá suporte à condução de experimentos. Dessa forma, realizou-se uma reengenharia na Plavis/FSM, resultando em um protótipo implementado por *web services* que dentre suas novas funcionalidades oferece apoio a experimentos.

Neste capítulo são apresentadas as atividades relacionadas à reengenharia da ferramenta Plavis/FSM. Os resultados da reengenharia da ferramenta (dados levantados, códigos reusados e serviços implementados) são apresentados na Seção 5.2. A análise das limitações e restrições do protótipo desenvolvido, assim como os principais problemas de implementação dos métodos e critérios utilizados são apresentadas na Seção 5.3.

5.2 Reengenharia da Plavis/FSM

O processo de reengenharia de software é constituído de duas etapas distintas. A primeira analisa o software, objeto de reconstrução, para a extração de informação e abstração. A segunda o constrói novamente, em uma nova forma desejada, a partir dos produtos da primeira etapa juntamente com os ajustes que se fizerem necessários (Bianchi et al., 2003).

Essa atividade visava à análise e à reengenharia da ferramenta Plavis/FSM quanto à sua aplicabilidade para experimentos de teste baseado em MEFs. Essa reengenharia refere-se à implementação de novas funcionalidades que permita o apoio a experimentos. Foi desenvolvido um protótipo de ferramenta para execução dos experimentos. O protótipo se baseou em dois projetos inseridos no contexto do grupo de pesquisas do Laboratório de Engenharia de Software (Labes) do Instituto de Ciências Matemáticas e de Computação (ICMC/USP).

5.2.1 Projetos Relacionados

Como dito na Seção 4.5, o grupo do autor desta dissertação participou do projeto PLAVIS e um dos resultados desse projeto foi o desenvolvimento da ferramenta Plavis/FSM para apoiar a aplicação de métodos de geração.

Atualmente o grupo participa do projeto *Qualipso* (*Quality Plataform for Open Source Software*) que é financiado pela União Europeia contendo universidades e empresas da Europa, Brasil e China (Qualipso, 2004). O objetivo do projeto é prover confiabilidade e qualidade em sistemas *open source*. Um dos resultados esperados é desenvolver um novo conceito de *forge* baseado em SOA (do inglês *Service Oriented Architecture*). Todas as funcionalidades serão fornecidas por *web services* e *gadgets* que acessarão os dados por meio de uma arquitetura orientada a serviços.

Os projetos PLAVIS e *Qualipso* estão no contexto deste trabalho, motivando a reengenharia da ferramenta. Basicamente, o primeiro projeto forneceu "o que" será desenvolvido (domínio de aplicação e a ferramenta Plavis/FSM) e o segundo projeto deu subsídios para "como" essa reengenharia pode ser desenvolvida. Dessa forma, optou-se em utilizar a tecnologia SOA na reengenharia da Plavis/FSM em concordância ao projeto *Qualipso*. Essa etapa da reengenharia do software focou em dois aspectos: 1) uso de *web services* para adicionar as funcionalidades requisitadas e permitir a disponibilização dos serviços oferecidos pela ferramenta na *web*; 2) técnicas de reúso, no intuito de aproveitar códigos da ferramenta já implementados e testados, visando a diminuir o tempo de desenvolvimento e a aumentar a qualidade do produto final. Na primeira etapa da reengenharia, algumas fontes de informação foram utilizadas para executar essa atividade e estão listadas a seguir:

- **Revisão Bibliográfica:** a revisão bibliográfica permitiu a obtenção de conhecimentos gerais sobre o problema e o domínio de aplicação da ferramenta.

- **Experiências pessoais:** por meio de entrevistas com os participantes do projeto PLAVIS, foi possível extrair informações sobre as funcionalidades da ferramenta, como e onde está sendo utilizada e possíveis *bugs* e melhorias. Constatou-se que a ferramenta não estava sendo utilizada na USP - São Carlos, o que dificultou o levantamento de dados. Dentre os participantes do projeto PLAVIS, verificou-se que somente o INPE (Instituto Nacional de Pesquisas Espaciais) utilizava a ferramenta. As dificuldades de utilização relatadas nas entrevistas são explicadas no próximo item.

- **Utilização:** percebeu-se que a ferramenta apresenta deficiência de amigabilidade e flexibilidade. Essa constatação também foi reportada nas entrevistas do item anterior. A ferramenta não disponibiliza *feedbacks* ao usuário, os quais são de suma importância para um melhor entendimento do que ocorre na execução. Além disso, as funções da ferramenta não são flexíveis e devem seguir uma ordem determinada e praticamente não há a possibilidade de desfazer uma ação. Percebeu-se que o foco da ferramenta está na análise de mutantes e não nos métodos de geração. Não é possível gerar uma sequência de teste de uma MEF por meio de um método, sem gerar primeiramente as MEFs mutantes.

- **Instalação:** a ferramenta foi reinstalada localmente em uma máquina do laboratório de pesquisa Labes, ICMC/USP. Desejou-se com essa instalação utilizar e executar testes em casos de alterações, além da visualização do código. Assim como sua utilização, a instalação da Plavis/FSM não é trivial. Exige-se a alteração de vários arquivos de configuração e a instalação de outros aplicativos para sua perfeita execução.

- **Documentação existente:** dentre a documentação da ferramenta, foi disponibilizado o site do projeto, o manual da ferramenta e um artigo publicado na seção de ferramentas de um simpósio (Simão et al., 2005), além do próprio código fonte. Não foi encontrada nenhuma documentação relativa à especificação de requisitos, nem a análise e projeto da ferramenta. Conseguiu-se identificar as funcionalidades da ferramenta e separá-las em módulos.

  - módulo responsável pela segurança de usuários (*login*);
  - módulo responsável pela comunicação com o banco de dados;
  - módulo responsável pela geração de mutantes;
  - módulo responsável pela edição de MEF;
  - módulo responsável pela integração com outras ferramentas;
  - módulo responsável pela geração dos casos de teste;
  - módulo responsável pela geração dos resultados.

Fez-se uma lista de requisitos e um diagrama de arquitetura para o desenvolvimento de um protótipo de ferramenta de apoio a teste baseado em MEFs. Foi possível modularizar algumas

dessas funcionalidades, aproveitando na construção do protótipo, que possibilitou a condução dos experimentos. Alguns desses módulos foram reusados no protótipo e outros foram reimplementados.

5.2.2 Reuso de Código

Um dos resultados da reengenharia da Plavis/FSM foi a divisão em componentes de seu código. Foi possível identificar os componentes responsáveis pela geração de conjunto de teste por alguns métodos. Como visto na Seção 4.5, a ferramenta Plavis/FSM é composta por várias ferramentas implementadas por diferentes tecnologias.

Dentre os métodos integrados na ferramenta encontram-se: método W (implementado em Java), método UIO e HSI (implementados em C), método *Switch Cover* (implementado em Prolog) e método *State Couting* (implementado em C++). Como comentado na Seção 4.2, o método *Switch Cover* não foi incluído nos experimentos, pois foi considerado um método pouco efetivo. Dos métodos integrados na Plavis/FSM, somente as implementações dos métodos *State Couting* e HSI (Cutigi e Simão, 2006) foram reusadas no desenvolvimento do protótipo.

As implementações dos critérios de cobertura S, T, IF e TF utilizadas nos experimentos descritos em Simão et al. (2009) também foram reusadas. Outros códigos que foram reutilizados no desenvolvimento do protótipo são: o método H (implementado em C) descrito em Stuchi e Simão (2008) e o método CS (implementado em Ruby) proposto em Simão e Petrenko (2008). Houve um grande esforço para entendê-los e adaptá-los a atender às necessidades dos experimentos e formatação dos arquivos de entrada.

5.2.3 Reimplementação de Código

Optou-se pela reimplementação dos métodos W e UIO (implementados em C++), apesar da Plavis/FSM incluir implementações para esses dois métodos, como visto na seção anterior. Essa decisão foi motivada por questões de adaptação do formato de entrada à estrutura da ferramenta e por um melhor entendimento na leitura do código pelo autor para manutenções. Além disso, os métodos Wp e UIOv são adaptações dos métodos W e UIO respectivamente, portanto as implementações desses métodos possuem código reaproveitável. Os demais métodos estudados neste trabalho e que não foram mencionados, assim como conversores de formato de entradas e *scripts* para rodar automaticamente alguns procedimentos dos experimentos também foram projetados e implementados.

Todo o módulo responsável pela geração de mutantes foi reimplementado. Neste trabalho, são utilizadas somente MEFs completas e minimais e assume-se que as possíveis implementações seguem essas mesmas propriedades e que o domínio da especificação (D_M) das implementações seja igual ao das MEFs geradas. Segundo Fabbri et al. (1994), para aplicação da análise de mutantes no contexto de MEFs, assume-se válida a hipótese de competência do desenvolvedor, que nesse

caso significa que o modelo produzido por um projetista é correto ou muito próximo da versão correta. Cada mutante representa um possível erro na implementação das MEFs geradas. Fabbri et al. (1994) definem um conjunto de operadores de mutação para MEFs. Considera-se que operadores passíveis de gerar mutantes parciais e não-minimais simplifica a identificação dos erros inseridos nos mutantes pelos conjuntos de teste utilizados neste trabalho, tornando tais resultados pouco atrativos. Por esse motivo, restringiu-se na reimplementação de dois operadores de mutação. O primeiro operador *output-exchanged* gera mutantes com erros de saída. Para cada transição a saída é alterada, sendo assim gerados t mutantes para cada MEF do experimento. O segundo operador (*destination-exchanged*) gera mutantes com erros de transferência. Para cada transição, todos os possíveis estados destinos foram alterados, totalizando $t \times (n-1)$ mutantes. Observa-se também que esses dois operadores não geram mutantes equivalentes a MEF especificada.

Implementou-se um algoritmo no protótipo que executa cada CT do conjunto de teste gerado, verificando se a saída do CT aplicado na MEF especificada é diferente da saída retornada por cada mutante gerado. Caso positivo, soma-se os mutantes mortos para o cálculo do escore de mutação. Caso negativo, diz-se que o conjunto não foi capaz de matar o mutante.

5.2.4 Arquitetura do Protótipo

Na Figura 5.1 apresenta-se a arquitetura do protótipo desenvolvido. Com exceção do **Cliente**, todos os objetos da arquitetura fazem parte do protótipo **(Serviços, Conversores, Programas)**. A aplicação **Cliente**, portanto, é responsável por invocar os **Serviços** do protótipo. Os serviços, por sua vez, executam os **Programas** representados pelas caixas quadradas na Figura 5.1. Foram definidos três serviços que são:

- **Serviço *"Define MEF"***: Por meio desse serviço, o usuário é capaz de definir uma MEF. A entrada desse serviço é um conjunto de transições no formato s_i -- x / y -> s_j, representado pelo parâmetro de entrada **trans** na Figura 5.1. Esse serviço executa o **programa *"Define MEF"*** que recebe esse conjunto de transições e retorna um arquivo *txt* (***fsm1.txt***) ao serviço. Esse arquivo é convertido para outro arquivo *txt* (***fsm.txt***) gerando a saída desse serviço ao cliente. Esses arquivos retornados contêm a MEF definida no formato utilizado pelo protótipo. O protótipo somente aceita MEFs completas, determinísticas, conexas e minimais. Caso a MEF definida pelo usuário não siga tais propriedades um erro é retornado.

- **Serviço *"Gera MEFs"***: Esse serviço é responsável por executar o **programa *"Gera MEFs"*** que gera e agrupa MEFs completas, determinísticas, conexas e minimais de forma aleatória. O agrupamento das MEFs é baseado nos parâmetros de entrada fornecidos pelo usuário que são: os parâmetros de MEF iniciais, os parâmetros de MEF finais, o valor de incremento para cada parâmetro de MEF e quantas MEFs serão geradas para cada grupo. Os parâmetros de MEF são representados pelas letras n (número de estados), k (número de entradas) e l

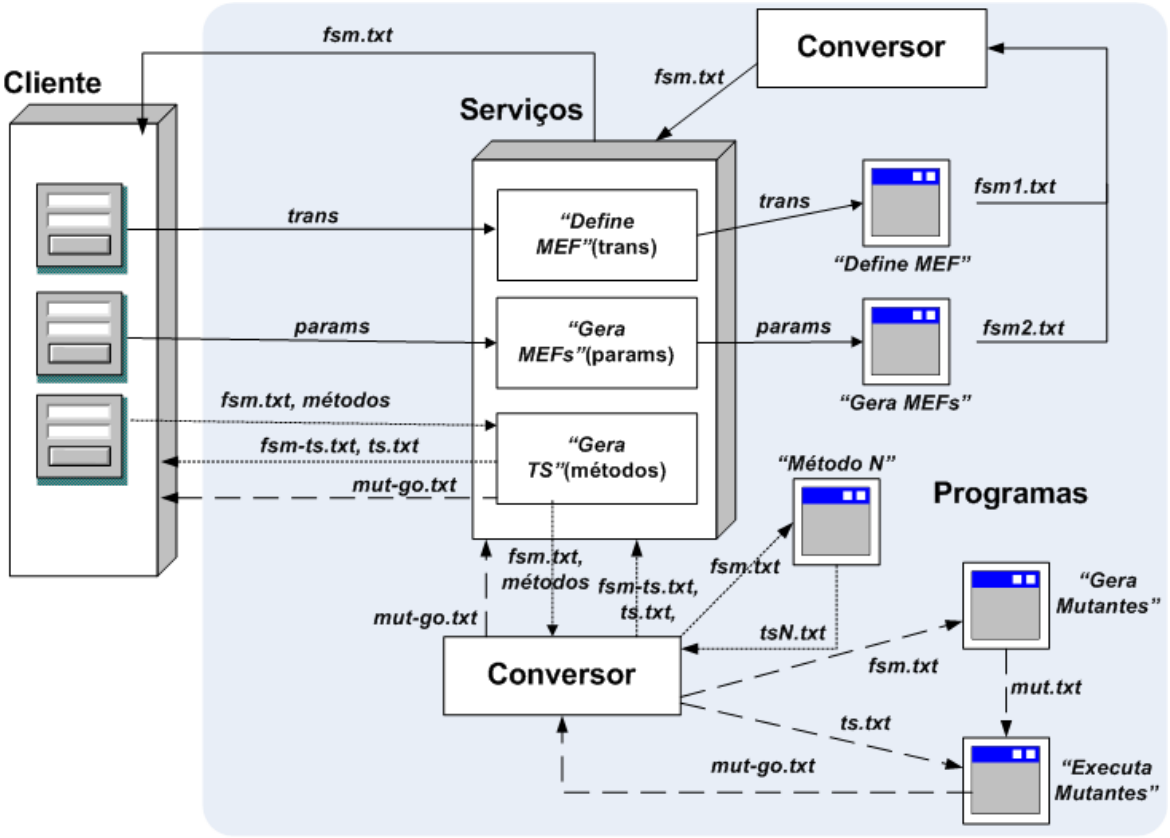

Figura 5.1: Arquitetura do protótipo desenvolvido baseado em serviços

(número de saídas). Na Figura 5.1 o termo **params** engloba todos os parâmetros de entradas desse serviço.

Por exemplo, o usuário entra com os valores (4,2,2) para (n, k, l), respectivamente, como parâmetros iniciais de MEF; (6,2,2) como parâmetros finais de MEF; (1,1,1) como o incremento dos parâmetros de MEF; e 2 para a quantidade (x) de MEFs para cada grupo. Nesse caso, serão gerados três grupos cada um contendo duas MEFs. O primeiro grupo as MEFs terão $n = 4$, $k = 2$, $l = 2$; o segundo $n = 5$, $k = 2$, $l = 2$; e $n = 6$, $k = 2$, $l = 2$ para o terceiro grupo. Assim como o primeiro serviço, um arquivo *txt* (***fsm.txt***) com o formato das MEFs é retornado ao usuário e também é possível visualizar as MEFs geradas. Tanto esse serviço como o serviço explicado acima, as setas contínuas na Figura 5.1 ilustram essa comunicação entre os módulos do protótipo.

- **Serviço *"Gera TS"***: Esse serviço é responsável por gerar os conjuntos de teste, gerar os mutantes e executar os CTs nos mutantes. O usuário seleciona quais métodos serão utilizados na geração dos conjuntos. Caso algum método não n-completo seja selecionado, o usuário ainda pode executar a análise de mutantes. Esse serviço executa os seguintes programas:

  1. **Programas *"Método N"***: Geram conjuntos de teste, seguindo os métodos selecionados pelo usuário. Como visto, vários métodos foram reusados de outros trabalhos e,

portanto, implementados em diversas linguagens de programação. Os métodos são embutidos nesses programas (*N* é nome do método)[1].

Esses programas possuem como entrada um arquivo contendo uma ou mais MEFs. Para cada programa, a MEF é convertida pelo módulo **Conversor** do protótipo para o formato de entrada lida pelo programa correspondente ao método selecionado.

Esses programas geram arquivos *txt* ***(tsN.txt)*** com seus respectivos formatos. Esses arquivos representam o conjunto de teste gerado por cada método. Novamente, o **Conversor** transforma esses formatos para um formato adequado para o **Serviço** ***(fsm-ts.txt, ts.txt)*** que retorna o arquivo convertido ao **Cliente**. O arquivo ***fsm-ts.txt*** contém os tamanhos dos conjuntos de teste e o arquivo ***ts.txt*** contém todos os conjuntos de teste gerados para as MEFs contidas no arquivo ***fsm.txt***. Essa comunicação entre os módulos do protótipo é representada pelas setas pontilhadas na Figura 5.1. Caso algum método não n-completo seja selecionado, os programas dos itens 2 e 3 também são chamados pelo **Serviço**.

2. **Programa *"Gera Mutantes"***: Gera mutantes para cada MEF gerada anteriormente. Esse programa possui como entrada um arquivo contendo uma ou mais MEFs. A saída desse programa é um arquivo *txt* ***(mut.txt)*** contendo os mutantes gerados.

3. **Programa *"Executa Mutantes"***: Executa os conjuntos de teste para cada mutante gerado no item anterior e contabiliza os mutantes mortos. Esse programa recebe como entradas o arquivo *txt* ***(mut.txt)*** contendo os mutantes gerados no item anterior e um arquivo *txt* ***(ts.txt)*** contendo os CTs gerados por algum método não n-completo. A saída desse programa é um arquivo *txt* ***(mut-go.txt)*** contendo a quantidade de mutantes mortos pelo conjunto de CTs executado. Assim como os demais, esse arquivo é convertido pelo **Conversor**, que envia ao **Serviço**, que por sua vez retorna o arquivo convertido ao **Cliente**. Para os dois últimos itens desse serviço, a comunicação entre os módulos do protótipo pode ser vista na Figura 5.1 pelas setas tracejadas.

Em termos técnicos da linguagem JAVA (JAVA, 1994), a assinatura desses três serviços, respectivamente, são:

- "*public DataHandler defFSM(DataHandler fsmFile)*"
- "*public DataHandler genFSM(String[] s)*".
- "*public DataHandler[] spanTS(DataHandler fsmFile, String[] s2)*"

DataHandler, que é o tipo de retorno dos serviços, é uma classe que representa um arquivo de dados. Os serviços ***"Define MEF"*** e ***"Gera MEFs"*** fornecem como saída um arquivo, no caso um arquivo *txt* ***fsm.txt*** que contêm a MEF definida ou a(s) MEF(s) geradas, respectivamente. O serviço ***"Gera TS"*** que fornece um *array* de arquivos. Como explicado os

[1] A relação 1 método para 1 programa não é verdadeira, ou seja, 1 programa pode gerar CTs para mais de 1 método

arquivos são: os conjuntos de teste gerados (***ts.txt***), tamanho dos conjuntos de teste gerados (***fsm-ts.txt***), os mutantes gerados (***mut.txt***) e o resultado da análise de mutantes (***mut-go.txt***).

O serviço ***"Define MEF"*** também requer um arquivo como entrada, o arquivo que contêm as transições da MEF. O serviço ***"Gera MEFs"*** requer um *array* de *strings s* como entrada. Como dito, esse *array* é representado pelo termo **params** na Figura 5.1. O serviço ***"Gera TS"*** requer um arquivo como entrada (***fsm.txt***), que contêm as MEFs geradas, e um *array* de *strings*, que contêm os métodos selecionados pelo usuário.

5.2.5 Aplicação Cliente

Como dito, foi desenvolvida uma aplicação cliente responsável pela chamada dos serviços implementados no protótipo e pelos testes de aceitação desses serviços. Tal aplicação foi batizada de **Plavis/FSM - Web Services** por ser um produto da reengenharia da ferramenta Plavis/FSM e por ser baseada em *web services*.

As entradas são definidas pelo usuário e passadas aos serviços por páginas JSP. As chamadas aos serviços são realizadas por classes JAVA, que estão inseridas em *Servlets*. O retorno dos *Servlets* é processado pelas páginas JSP e mostrado ao usuário em forma de página *html* que representa a saída dos serviços.

Na Figura 5.2, ilustra-se uma tela de entrada, na qual o usuário informa os parâmetros do serviço *"Gera MEFs´´*. O primeiro campo do formulário, após a descrição "n =", corresponde ao valor do número de estados n inicial. O segundo campo, após "n <=", corresponde ao valor de n final. O terceiro campo dessa linha, após o "n +=", corresponde ao valor do incremento de n. Da mesma forma, deve-se informar os valores do número de entradas k e saídas l nas próximas linhas. Por fim, na última linha só há um campo no qual deve se inserir a quantidade x de MEFs para cada grupo.

Na Figura 5.3, ilustra-se uma tela de saída, na qual o serviço foi executado retornando ao usuário os resultados da análise de mutantes. A visualização dos resultados é separada pelos operadores mutantes. É possível analisar o total de mutantes, a quantidade e a porcentagem de mutantes mortos por cada método ou critério.

5.3 Limitações do Protótipo Desenvolvido

O desenvolvimento do protótipo possui algumas limitações inerentes a questões de complexidade, implementação dos métodos e redução de escopo.

Com relação às propriedades de MEFs, somente são geradas MEFs determinísticas, completas, conexas e mínimas nos experimentos. Essa limitação se deve excepcionalmente pela necessidade dessas propriedades de MEFs que alguns métodos exigem.

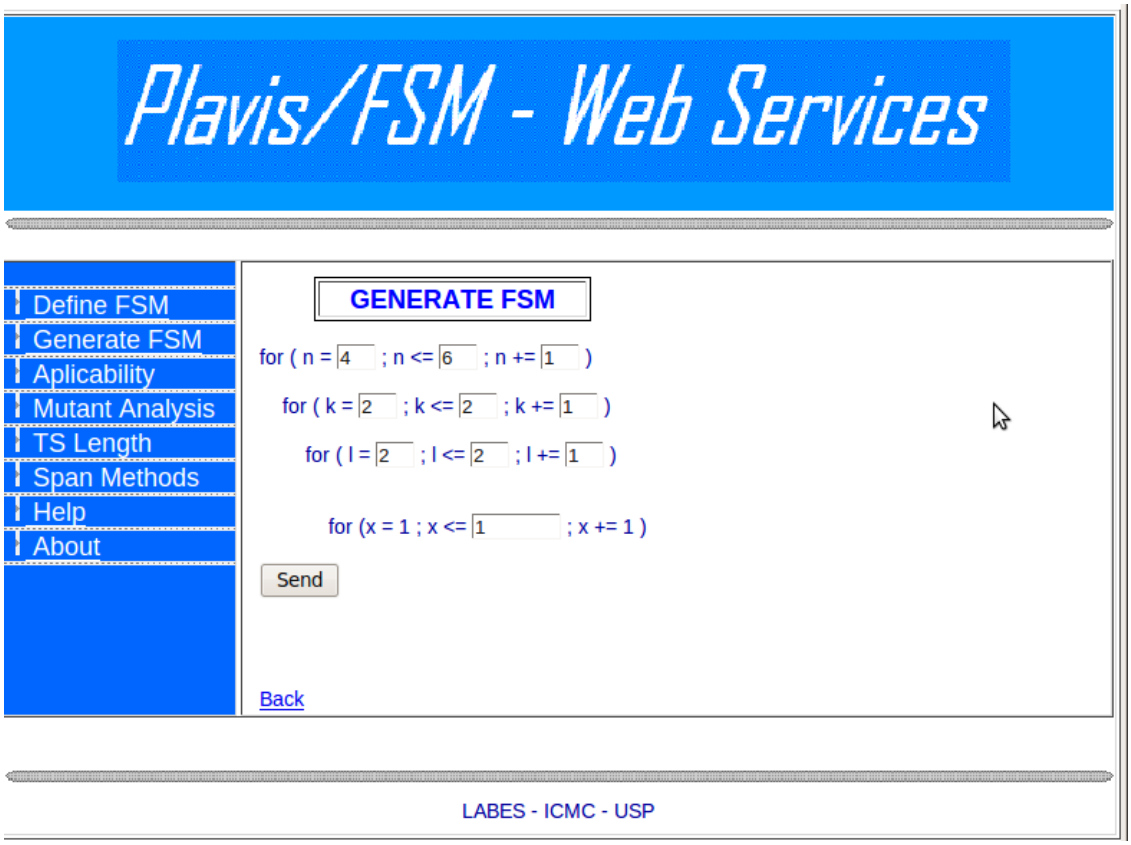

Figura 5.2: Tela de Entrada (geração de MEFs)

As MEFs utilizadas possuem um número máximo de 100 estados, 10 entradas, 10 saídas e consequentemente um número máximo de 1000 transições, por se tratarem de MEFs determinísticas.

Algumas alternativas e aproximações foram implementadas a fim de contornar problemas intratáveis que podem ocorrer quando se trabalha com MEFs, principalmente relacionados à explosão de estados.

Um dos problemas de difícil solução é encontrar a menor sequência de entrada que passe por todos os estados ou todas as transições. Esse problema corresponde ao *problema do caixeiro viajante misto* que pertence à classe dos problemas NP-Completo (Aho et al., 1988). Foi visto no Capítulo 4, que esse problema ocorre na geração do método TT. Foi utilizada uma heurística "gulosa" para contornar esse problema. Uma heurística similar foi utilizada para a minimização dos conjuntos de teste dos critérios de cobertura. Sabe-se que, por se tratar de heurísticas, ganha-se em recursos computacionais, mas não garante que a solução seja a melhor existente.

Outro problema NP-completo é determinar o menor conjunto W para uma MEF. As MEFs geradas no experimento possuem infinitos conjuntos W. Da mesma forma que ocorre no *problema do caixeiro viajante misto*, é intratável utilizar um algoritmo que retorne o menor conjunto W, principalmente para MEFs grandes. O algoritmo implementado neste trabalho tenta encontrar um conjunto W pequeno, mas não necessariamente o menor. O algoritmo utiliza as tabelas P_k e para cada par de estados é calculada a k-distinção (Nakazato et al., 1994). Cada combinação de entradas

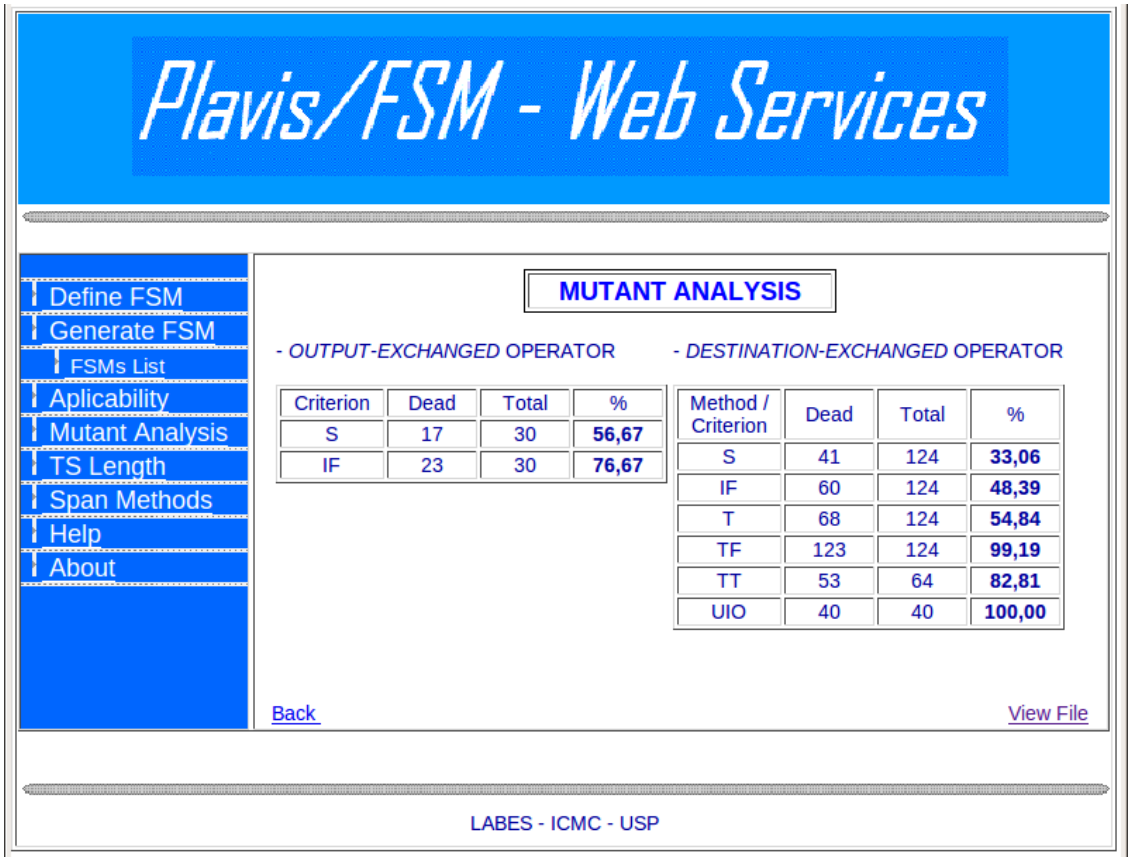

Figura 5.3: Tela de Saída (análise de mutantes)

possíveis de tamanho k é examinada até encontrar uma sequência que os distingue. A união dessas sequências forma um conjunto W.

Além do tamanho do conjunto W, a quantidade de elementos do conjunto também influencia no tamanho final do conjunto de teste. Foram observados casos em que um conjunto W maior gerou um conjunto de teste menor, pois possuía menos elementos que outro conjunto W (menor em tamanho). Problemas similares ao da escolha do conjunto W que gere o menor conjunto de teste, também ocorre para a sequência DS, sequências UIO e conjunto HSI.

Outra diferença que pode ocorrer nos resultados finais está nos algoritmos e nas heurísticas implementadas nos métodos. Foi visto, no Capítulo 4, que a área de pesquisa sobre os métodos de geração está ativa há muitas décadas e, portanto, várias melhorias foram pesquisadas e desenvolvidas nos diversos métodos apresentados. Faz-se necessário enfatizar os detalhes e as otimizações utilizadas na implementação dos métodos para o conhecimento de quais melhorias (heurísticas) foram utilizadas na implementação dos métodos. Em razão desse fato, o estudo comparativo deste trabalho com trabalhos que não disponibilizam o código-fonte, nem a descrição detalhada dos algoritmos e heurísticas utilizadas na implementação dos métodos, pode ser pouco efetivo.

5.4 Considerações Finais

Ao longo deste capítulo foram apresentados os elementos referentes a reengenharia da ferramenta Plavis/FSM. Para essa reengenharia optou-se na utilização da tecnologia baseada em *web services* consonante ao projeto Qualipso.

O entendimento da Plavis/FSM, o reúso de códigos, a implementações dos métodos, arquitetura do protótipo foram discutidos neste capítulo. Algumas dificuldades foram encontradas e soluções foram identificadas. No entanto, com as limitações conhecidas e o protótipo final desenvolvido, pode-se descrever os detalhes dos experimentos conduzidos.

No próximo capítulo, serão abordadas a condução dos experimentos e a discussão dos resultados obtidos.

CAPÍTULO

6

Condução dos Experimentos e Discussão dos Resultados

6.1 Considerações Iniciais

A condução dos experimentos descreve detalhes da forma como eles foram definidos em termos de objetivos, questões e métricas, assim como o emprego dos instrumentos envolvidos e os procedimentos de coleta e validação dos dados. Após validados, os dados foram submetidos à análise, da qual foram derivadas informações relevantes ao experimento e que serviram de embasamento para as conclusões acerca do tema investigado e para o direcionamento de trabalhos futuros.

Tais experimentos são baseados na medição de custo e eficácia dos métodos e critério de cobertura de teste baseado em MEFs. A comparação do custo é visualizada por meio de gráficos representados por sete casos. Cada caso é diferenciado por quais parâmetros de MEF sofrem variações. Para comparação da eficácia utilizam-se somente os métodos e critérios não n-completos. A técnica de análise de mutantes e a porcentagem de conjuntos que satisfazem as condições de suficiência foram utilizadas para avaliação da eficácia.

Este capítulo está dividido na seguinte forma. Na Seção 6.2, definem-se os experimentos explicitando os objetivos, as questões a serem respondidas e as métricas que são utilizadas. Na Seção

6.3, discute-se a geração de MEFs e dos conjuntos de teste utilizados nos experimentos. Na Seção 6.4, são apresentados os procedimentos e resultados da avaliação experimental do custo dos métodos e critérios. A aplicabilidade de alguns métodos é descrita na Seção 6.5. Por fim, na Seção 6.6, são apresentados os detalhes e resultados dos experimentos que avaliam a eficácia dos métodos e critérios estudados neste trabalho.

6.2 Definição dos Experimentos

Para a definição dos experimentos, utiliza-se a abordagem GQM (Goal Question Metric) (Basili, 1996). Essa abordagem identifica os objetivos do experimento, quais as questões que o experimento visa a responder e as métricas que serão utilizadas para tentar responder a tais questões.

- **Objetivos:** avaliar em termos de custo e eficácia os métodos de geração e os critérios de cobertura de teste baseado em MEFs, com a finalidade de obter dados experimentais para elaborar estratégias de teste efetivas no planejamento de futuros desenvolvimentos de software. Os métodos estudados são: TT, W, Wp, UIO, UIOv, DS, CS, HSI, H e SC e os critérios estudados são: S, IF, T e TF.

- **Questões:** baseados nos objetivos deste trabalho, os experimentos estão direcionados a responder às seguintes questões:

  i) Como os conjuntos de teste gerados pelos métodos e critérios se relacionam em termos de custo?

  ii) Quais parâmetros de MEF contribuem mais com a explosão de teste?

  iii) Quão aplicáveis são os métodos TT, CS, DS, UIO e UIOv?

  iv) Quão eficazes são os conjuntos de teste gerados pelos métodos e critérios não n-completos (TT, UIO, S, T, IF e TF);

- **Métricas:** para responder as questões (*i*) e (*ii*), a métrica utilizada é o tamanho médio dos conjuntos de teste gerados por cada método ou critério. Após a geração dos conjuntos de teste, é possível calcular a porcentagem e MEFs aplicáveis ao métodos citados na questão (*iii*). Para avaliação da eficácia é utilizada a técnica de análise de mutantes, contabilizando a quantidade de mutantes mortos ao aplicar os conjuntos de teste. Essa técnica, como visto, analisa a qualidade do conjunto de teste medindo sua capacidade de matar os mutantes. Também é investigado como os conjuntos de teste gerados pelo método UIO e pelo critério TF se relacionam com a noção de completude. Portanto utiliza-se o escore de mutação e a porcentagem de conjuntos que satisfazem as condições de suficiência como métrica para responder a questão (*iv*).

6.3 Condução dos Experimentos

Ao término da definição dos experimentos, inicia-se a condução desses. Essa atividade engloba a preparação e execução dos experimentos com o apoio do protótipo implementado, descrito no Capítulo 5, e a validação dos dados coletados durante a execução. A condução foi a etapa que demandou mais tempo neste estudo. Durante essa atividade, falhas foram detectadas e corrigidas, o que possibilitou a validação do protótipo. Em consequência, o protótipo facilita a condução dos experimentos tornando-a menos propensa a erros. O primeiro passo da condução dos experimentos consiste na geração de MEFs.

6.3.1 Geração e Agrupamento de MEFs

A geração e o agrupamento das MEFs são baseados nos parâmetros de entrada requeridos ao protótipo. Uma ferramenta geradora de MEF é utilizada para geração das MEFs dos experimentos, que segue cinco fases:

- A ferramenta inicialmente gera os conjuntos de estados (S), entradas (X) e saídas (Y), que são conjuntos formados pelo elemento 0 ao número requerido do parâmetro de MEF menos um. Exemplo: se $n = 4$, $S = \{0, 1, 2, 3\}$.

- A segunda fase produz uma MEF em árvore, ou seja, uma MEF inicialmente conexa. O estado 0 é definido como estado inicial e marcado como "alcançado", os demais estados são marcados como "não alcançados". Para cada estado s marcado como "não alcançado", cria-se uma transição, cujo estado destino é s e são selecionados aleatoriamente o estado origem s' ("alcançado"), uma entrada x (ainda não definida para s') e uma saída y da transição. Após a criação da transição, o estado s é marcado como "alcançado".

- Na terceira fase, transições são adicionadas, selecionando aleatoriamente um par de estados, uma entrada e uma saída, até que a MEF torne-se completa.

- Na quarta fase é verificado se todos os valores y do conjunto de saídas Y são utilizados. Caso contrário, a saída de um transição, escolhida aleatoriamente, é modificada para um símbolo de saída ainda não usado. Repete-se a quarta fase, até todos valores de Y serem utilizados.

- Por fim, é verificado se a MEF gerada é minimal, caso contrário, a mesma é descartada e o procedimento de geração é reiniciado.

As MEFs geradas de forma aleatória são agrupadas de acordo com quais parâmetros de MEF sofrem variações. O conjunto total de MEFs geradas pelo protótipo é dividido em sete casos representados por um arquivo ***fsm.txt*** (ver Figura 5.1). Portanto, são utilizados sete arquivos contendo as MEFs geradas nos experimentos que são divididas nos seguintes casos:

- **Caso 1** - MEFs com variação do número de estados,
- **Caso 2** - MEFs com variação do número de entradas,
- **Caso 3** - MEFs com variação do número de saídas,
- **Caso 4** - MEFs com variação do número de estados e entradas,
- **Caso 5** - MEFs com variação do número de estados e saídas,
- **Caso 6** - MEFs com variação do número de entradas e saídas,
- **Caso 7** - MEFs com variação do número de estados, entradas e saídas;

Para cada um desses sete casos uma nova divisão é feita baseada nos valores dos parâmetros de MEF. Foram subdivididos em 9 grupos, exceto o caso 1, que é subdividido em 15, totalizando 69 grupos. Cada grupo possui seu respectivo identificador (id) de 1 a 69. Todos os grupos de MEFs utilizados nos experimentos estão descritos na Tabela 6.1. Cada grupo contém 1.000 MEFs, totalizando, então, 69.000 MEFs geradas neste experimento. O agrupamento de MEFs visa a diminuir a complexidade do problema em problemas menores e a prover resultados comparativos entre os grupos.

Observa-se que como as MEFs são geradas aleatoriamente não há grupos iguais, ou seja, cada grupo possui MEFs diferentes, mesmo tendo valores dos parâmetros de MEFs idênticos, como por exemplo os grupos 3 e 63.

6.3.2 Geração dos Conjuntos de Teste

Após a geração das MEFs, a condução segue com a geração dos conjuntos de teste dos métodos e critérios estudados. Foi visto na Seção 5.2.1 que alguns métodos e critérios foram reutilizados de outros trabalhos e outros foram implementados. Com os conjuntos de teste gerados torna-se possível a avaliação dos métodos e critérios, comparando-os em termos de custo e eficácia.

Neste estudo, o custo está relacionado na execução do conjunto de teste e é calculado pelo **tamanho final** do conjunto de teste gerado pelos métodos e critérios. Entende-se por tamanho final, o conjunto de teste que não possua CTs prefixos de outros e, se necessário, a função *reset* no início de cada CT. Calcula-se o **tamanho médio** dos conjuntos de teste para cada método e critério, que é a razão da soma dos tamanhos dos conjuntos gerados pela quantidade de MEFs aplicáveis por cada método e critério. Como é visto, alguns métodos exigem propriedades ou sequências básicas para serem aplicáveis. Dessa forma é possível calcular também a aplicabilidade desses métodos.

Eficácia diz respeito à capacidade do método em detectar erros. Nessa avaliação, os experimentos envolvem a geração de mutantes e a execução dos conjuntos de teste gerados nesses mutantes, assim como a verificação se esses conjuntos satisfazem as condições de suficiência descritas em Simão e Petrenko (2007).

Caso	Grupo Id	n	k	l	t
Caso 1	Grupo 1	4	4	4	16
	Grupo 2	6	4	4	24
(Variação	Grupo 3	8	4	4	32
do número	Grupo 4	10	4	4	40
de estados)	Grupo 5	12	4	4	48
	Grupo 6	14	4	4	56
	Grupo 7	16	4	4	64
	Grupo 8	18	4	4	72
	Grupo 9	20	4	4	80
	Grupo 10	25	4	4	100
	Grupo 11	30	4	4	120
	Grupo 12	35	4	4	140
	Grupo 13	40	4	4	160
	Grupo 14	45	4	4	180
	Grupo 15	50	4	4	200
Caso 2	Grupo 16	10	2	2	20
	Grupo 17	10	3	2	30
(Variação	Grupo 18	10	4	2	40
do número	Grupo 19	10	5	2	50
de entradas)	Grupo 20	10	6	2	60
	Grupo 21	10	7	2	70
	Grupo 22	10	8	2	80
	Grupo 23	10	9	2	90
	Grupo 24	10	10	2	100
Caso 3	Grupo 25	10	2	2	20
	Grupo 26	10	2	3	20
(Variação	Grupo 27	10	2	4	20
do número	Grupo 28	10	2	5	20
de saídas)	Grupo 29	10	2	6	20
	Grupo 30	10	2	7	20
	Grupo 31	10	2	8	20
	Grupo 32	10	2	9	20
	Grupo 33	10	2	10	20

Caso	Grupo	n	k	l	t
Caso 4	Grupo 34	4	2	2	8
	Grupo 35	6	3	2	18
(Variação	Grupo 36	8	4	2	32
do número	Grupo 37	10	5	2	50
de estados	Grupo 38	12	6	2	72
e entradas)	Grupo 39	14	7	2	98
	Grupo 40	16	8	2	134
	Grupo 41	18	9	2	162
	Grupo 42	20	10	2	200
Caso 5	Grupo 43	4	2	2	8
	Grupo 44	6	2	3	12
(Variação	Grupo 45	8	2	4	16
do número	Grupo 46	10	2	5	20
de estados	Grupo 47	12	2	6	24
e saídas)	Grupo 48	14	2	7	28
	Grupo 49	16	2	8	32
	Grupo 50	18	2	9	36
	Grupo 51	20	2	10	40
Caso 6	Grupo 52	10	2	2	20
	Grupo 53	10	3	3	30
(Variação	Grupo 54	10	4	4	40
do número	Grupo 55	10	5	5	50
de entradas	Grupo 56	10	6	6	60
e saídas)	Grupo 57	10	7	7	70
	Grupo 58	10	8	8	80
	Grupo 59	10	9	9	90
	Grupo 60	10	10	10	100
Caso 7	Grupo 61	4	2	2	8
	Grupo 62	6	3	3	18
(Variação	Grupo 63	8	4	4	32
do número	Grupo 64	10	5	5	50
de estados,	Grupo 65	12	6	6	72
entradas	Grupo 66	14	7	7	98
e saídas)	Grupo 67	16	8	8	134
	Grupo 68	18	9	9	162
	Grupo 69	20	10	10	200

Tabela 6.1: Agrupamento das MEFs geradas

6.4 Avaliação do Custo

Nesses experimentos, visa-se a responder às questões (*i*) 'Como os conjuntos de teste gerados pelos métodos e critérios se relacionam em termos de custo?' e (*ii*) 'Quais parâmetros de MEF contribuem mais com a explosão de teste' descritas na Seção 6.2.

Para visualização dos resultados e efeito de comparação entre os métodos e critétios, utilizam-se gráficos. O eixo vertical do gráfico representa o tamanho do conjunto de teste e o eixo horizontal representa o valor do parâmetro de MEFs que é variado. Pode-se observar pelos gráficos o crescimento das curvas dos tamanhos dos conjuntos de teste de cada método. Em razão da quantidade de métodos, separa-se os métodos n-completos dos métodos e critérios não n-completos. A seguir, são apresentados os experimentos conduzidos divididos em 7 casos. Os resultados dos casos são discutidos juntamente após a apresentação deles.

Caso 1. Variação do número de estados (n)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 1 que é subdividido em 15 grupos de 1.000 MEFs, totalizando 15.000 MEFs. Do grupo 1 ao grupo 9, o número de estados

varia de 4 a 20 com incremento igual a 2 e do grupo 9 ao grupo 15, o número de estados varia de 20 a 50 com incremento igual a 5, como ilustrado na Tabela 6.1. O número de entradas e o de saídas são fixados em 4. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.1, 6.2, 6.3, 6.4. Com o intuito de melhorar a visualização dos gráficos, divide-se o caso 1 em dois gráficos; do grupo 1 ao 9 e do grupo 9 ao 15.

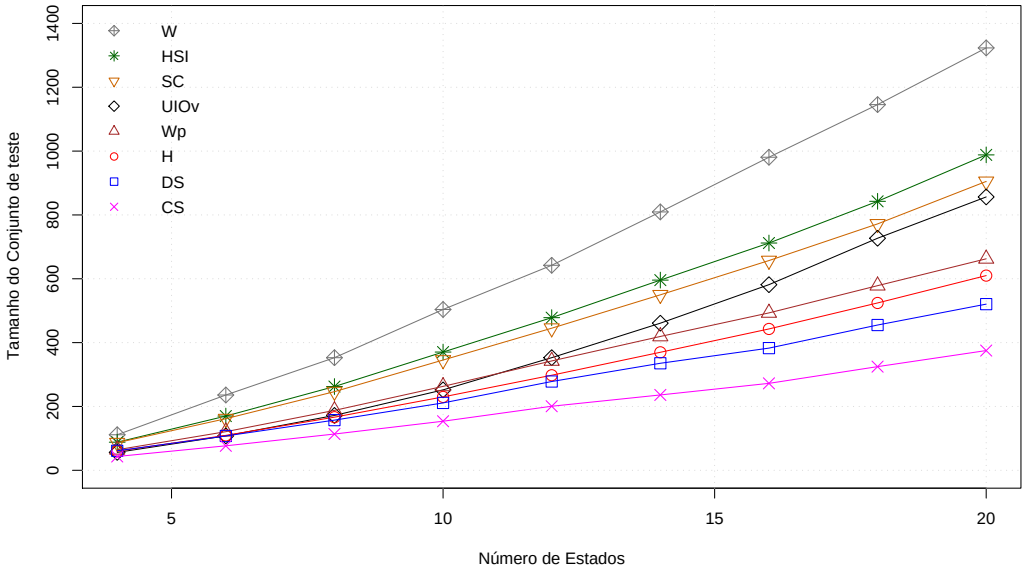

Figura 6.1: Variação do n para métodos n-completos (grupos 1 ao 9)

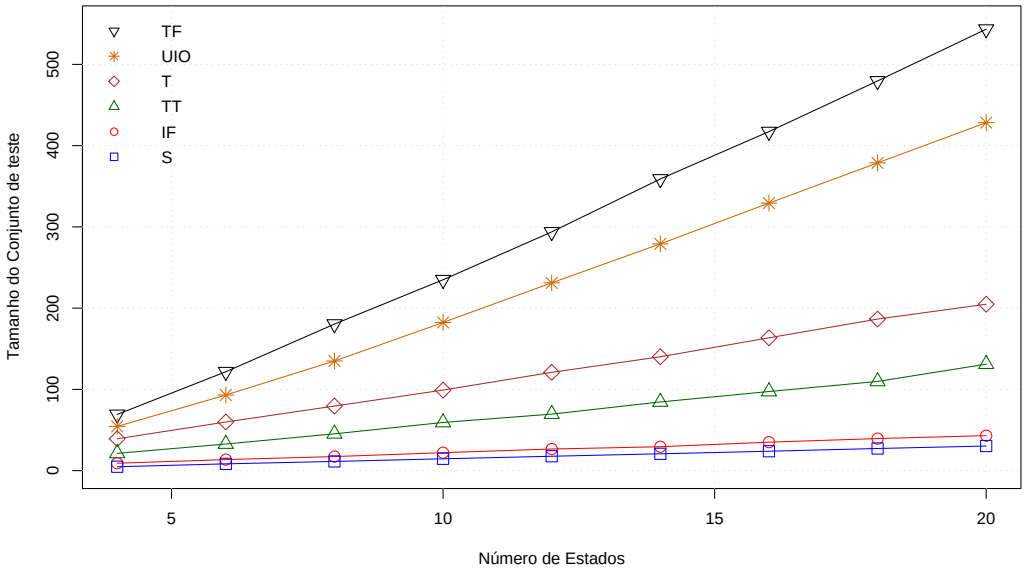

Figura 6.2: Variação do n para métodos e critérios não n-completos (grupos 1 ao 9)

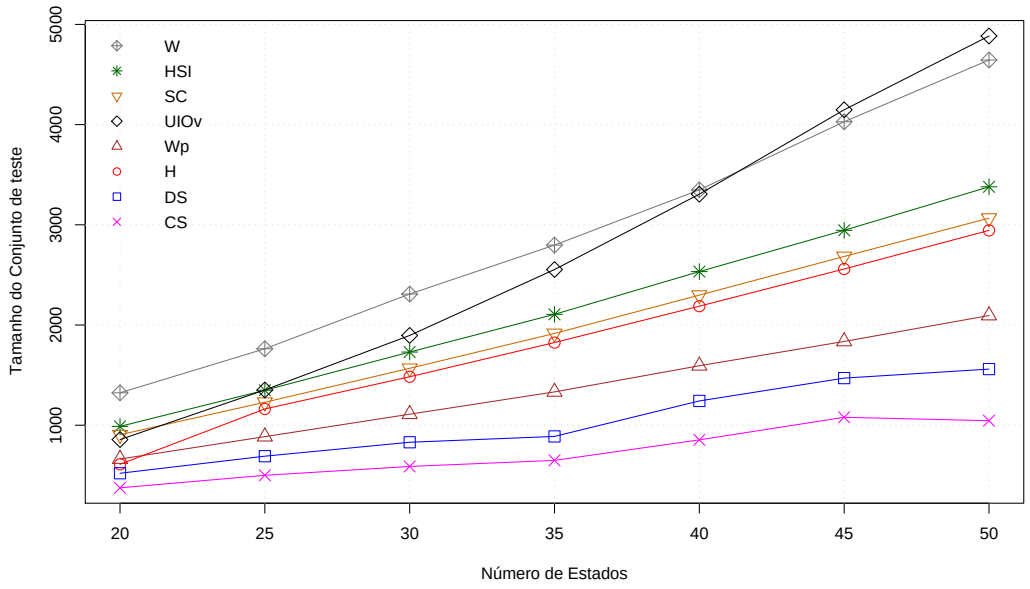

Figura 6.3: Variação do n para métodos n-completos (grupos 9 ao 15)

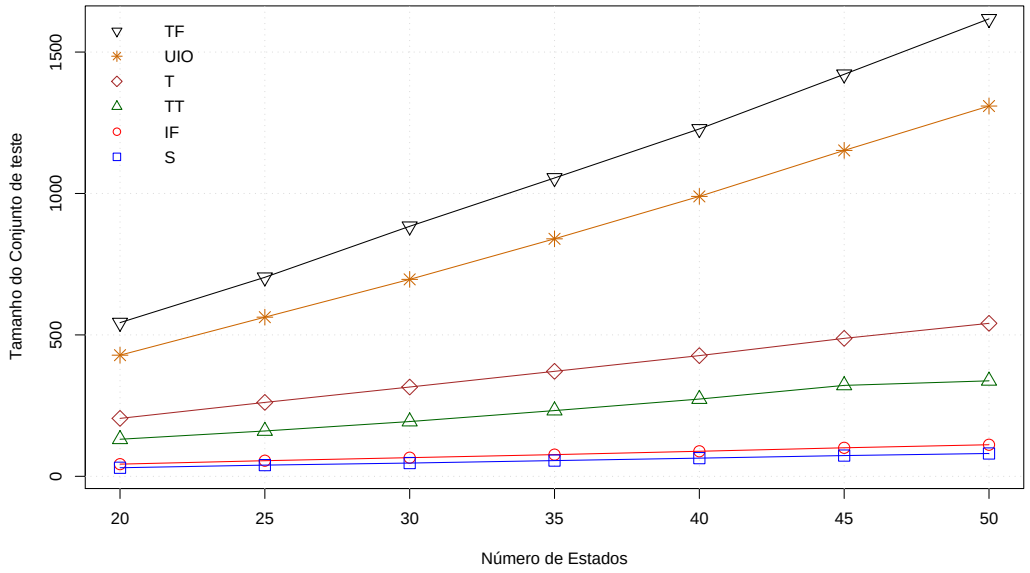

Figura 6.4: Variação do n para métodos e critérios não n-completos (grupos 9 ao 15)

Caso 2. Variação do número de entradas (k)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 2 que é subdividido em 9 grupos (16 a 24) de 1.000 MEFs, totalizando 9.000 MEFs. O número de entradas varia de 2 a 10 com incremento igual a 1. O número de estados é fixado em 10 e o de saídas é fixado em 2. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.5 e 6.6.

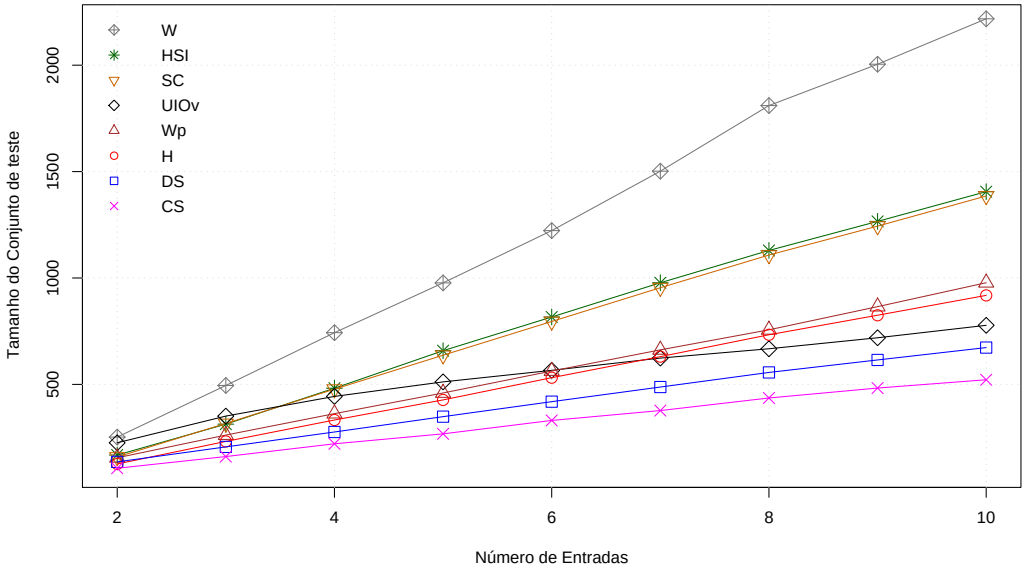

Figura 6.5: Variação do k para métodos n-completos

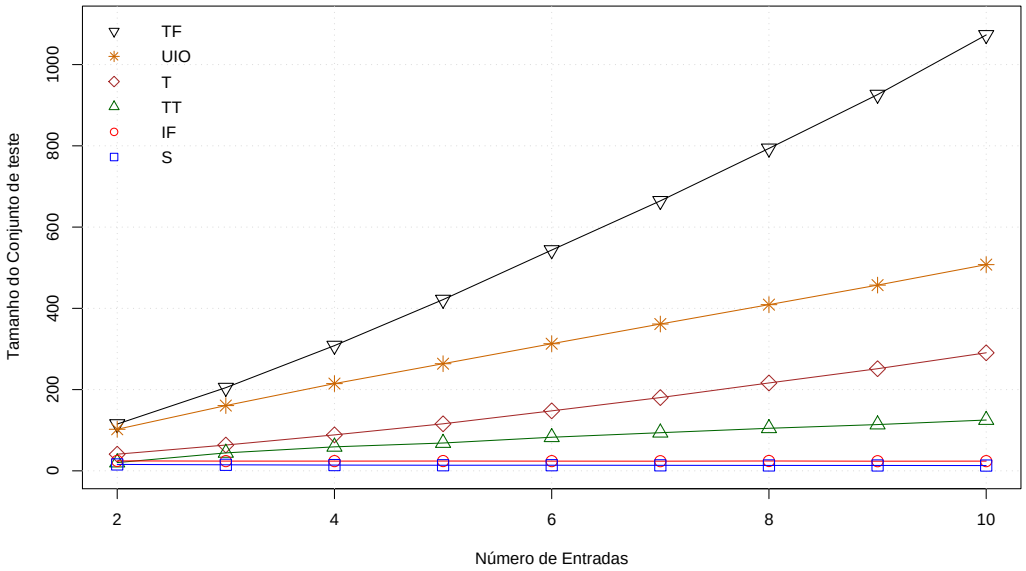

Figura 6.6: Variação do k para métodos e critérios não n-completos

Caso 3. Variação do número de saídas (l)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 3 que é subdividido em 9 grupos (25 a 33) de 1.000 MEFs, totalizando 9.000 MEFs. O número de saídas varia de 2 a 10 com incremento igual a 1. O número de estados é fixado em 10 e o de entradas é fixado em 2. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.7 e 6.8.

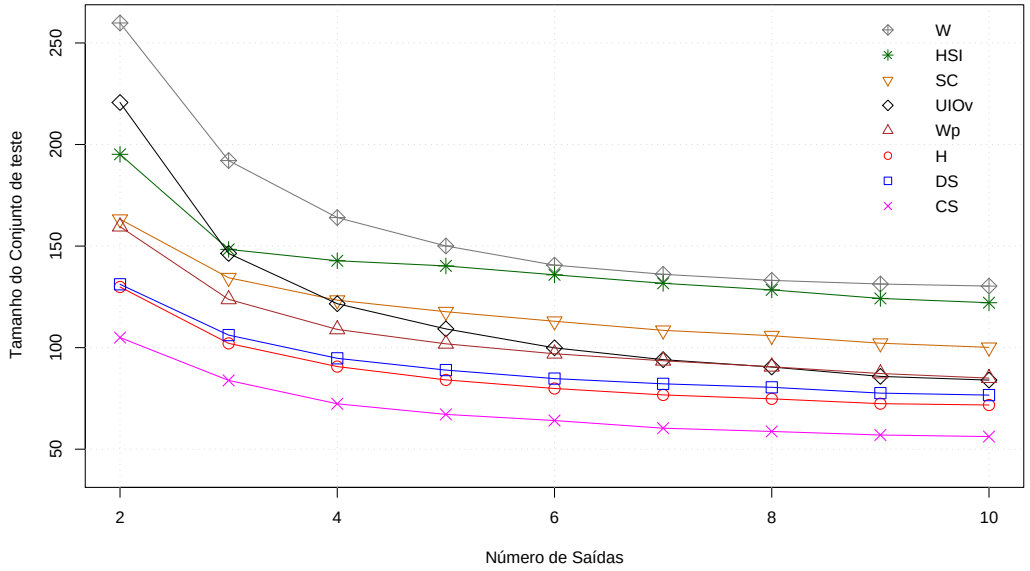

Figura 6.7: Variação do l para métodos n-completos

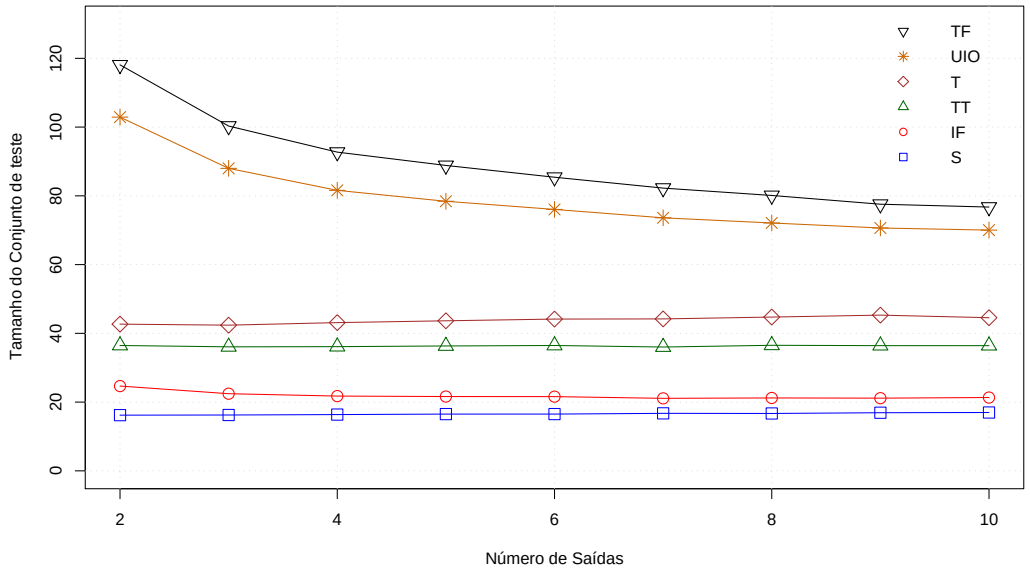

Figura 6.8: Variação do l para métodos e critérios não n-completos

Caso 4. Variação do número de estados (n) e entradas (k)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 4 que é subdividido em 9 grupos (34 a 42) de 1.000 MEFs, totalizando 9.000 MEFs. O número de estados varia de 4 a 20 com incremento igual a 2. O número de entradas varia de 2 a 10 com incremento igual a 1. O número de saídas é fixado em 2. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.9 e 6.10. A partir desse caso, mais de um parâmetro de MEF é variado, portanto o rótulo do eixo horizontal será representado pelo identificador (id) do grupo.

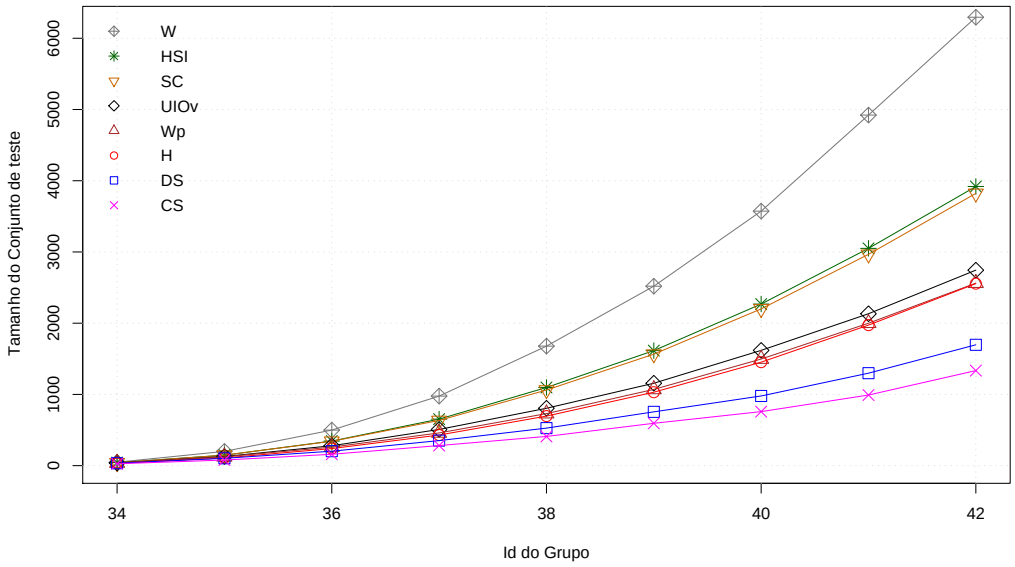

Figura 6.9: Variação do n e k para métodos n-completos

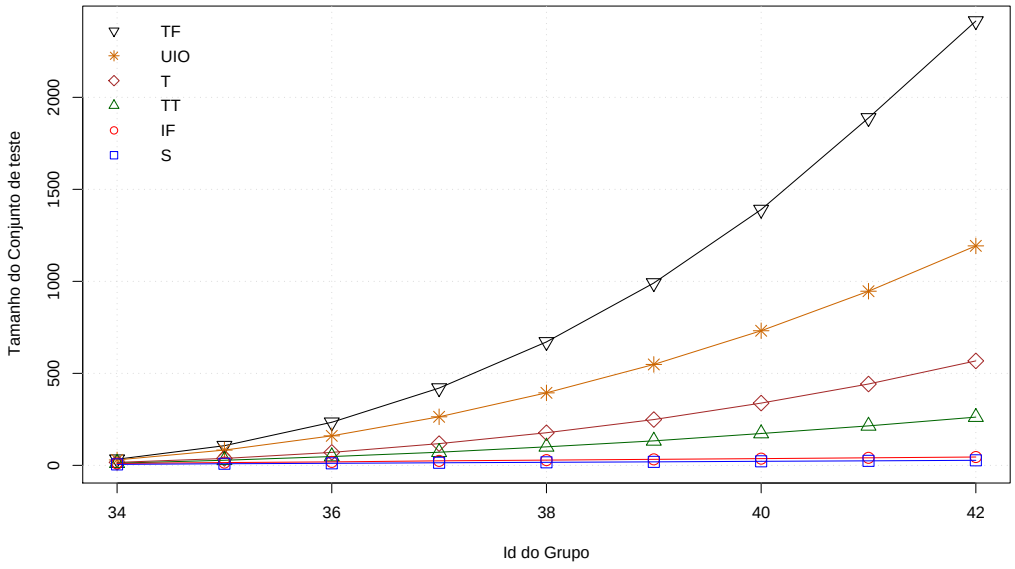

Figura 6.10: Variação do n e k para métodos e critérios não n-completos

Caso 5. Variação do número de estados (n) e saídas (l)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 5 que é subdividido em 9 grupos (43 a 51) de 1.000 MEFs, totalizando 9.000 MEFs. O número de estados varia de 4 a 20 com incremento igual a 2. O número de saídas varia de 2 a 10 com incremento igual a 1. O número de entradas é fixado em 2. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.11 e 6.12.

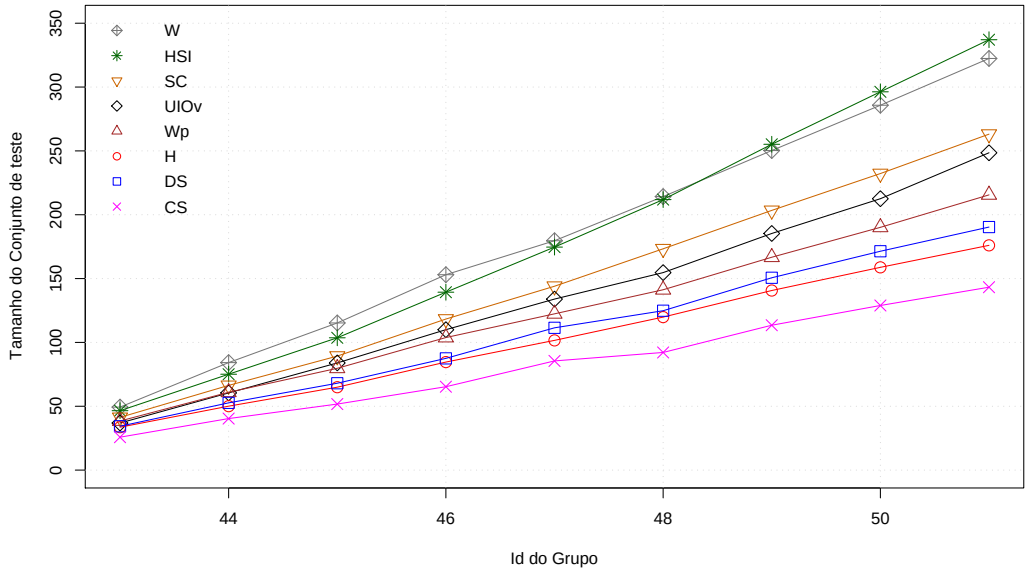

Figura 6.11: Variação do n e l para métodos n-completos

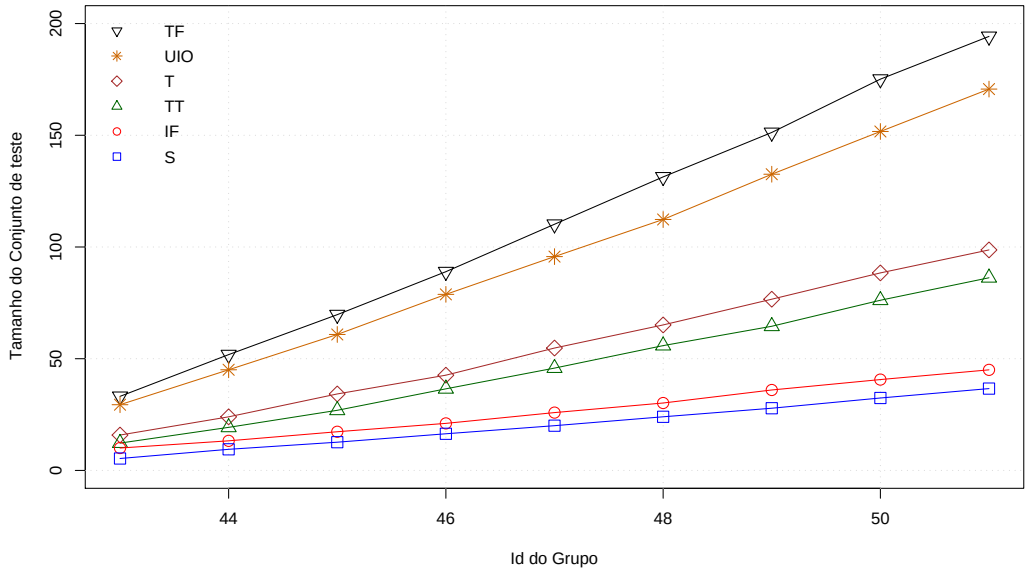

Figura 6.12: Variação do n e l para métodos e critérios não n-completos

Caso 6. Variação do número de entradas (k) e saídas (l)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 6 que é subdividido em 9 Grupos (52 a 60) de 1.000 MEFs, totalizando 9.000 MEFs. O número de entradas e de saídas varia de 2 a 10 com incremento igual a 1 para ambos. O número de estados é fixado em 10. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.13 e 6.14.

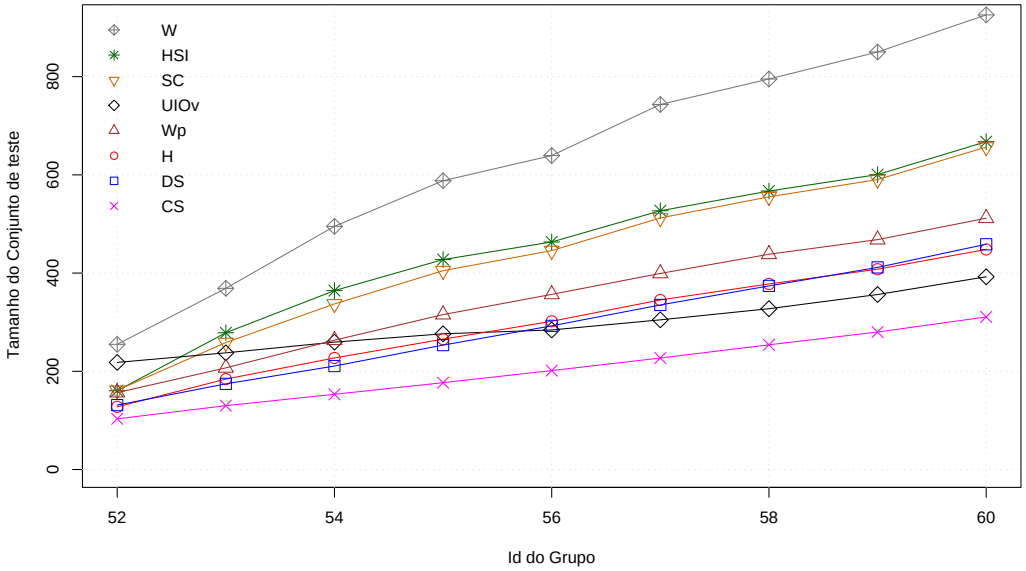

Figura 6.13: Variação do k e l para métodos n-completos

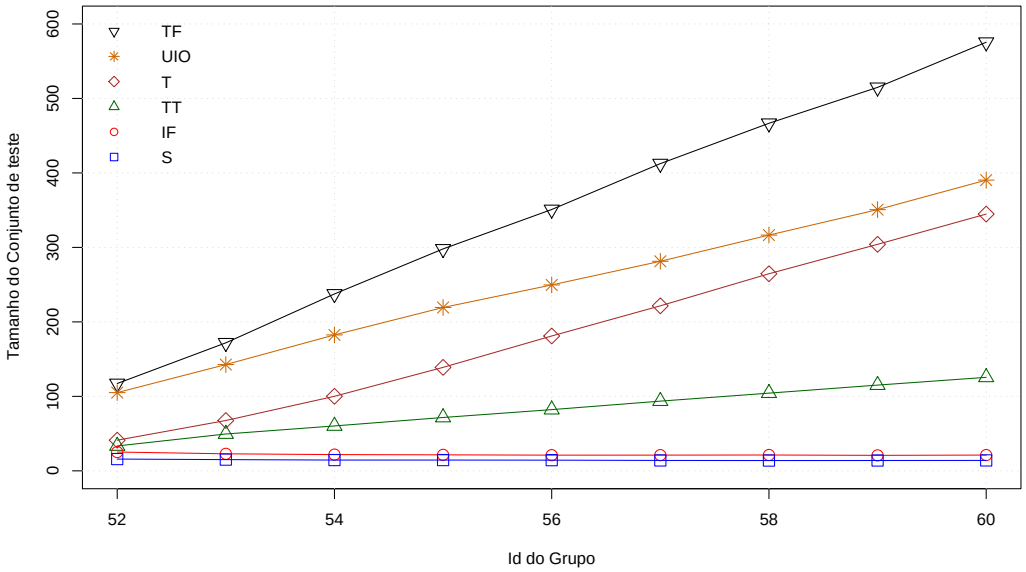

Figura 6.14: Variação do k e l para métodos e critérios não n-completos

Caso 7. Variação do número de estados (n), entradas (k) e saídas (l)

As MEFs utilizadas neste experimento são as MEFs agrupadas no caso 7 que é subdividido em 9 Grupos (61 a 69) de 1.000 MEFs, totalizando 9.000 MEFs. O número de estados varia de 4 a 20 com incremento igual a 2. O número de entradas e de saídas varia de 2 a 10 com incremento igual a 1 para ambos. Os gráficos que contém o tamanho médio dos conjuntos de teste gerados estão exibidos nas Figuras 6.15 e 6.16.

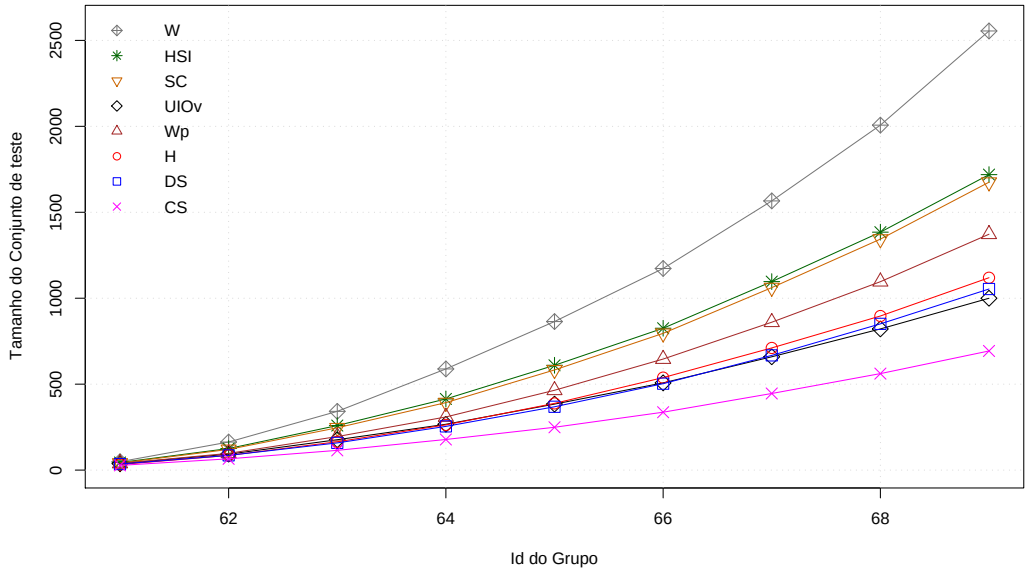

Figura 6.15: Variação do n, k e l para métodos n-completos

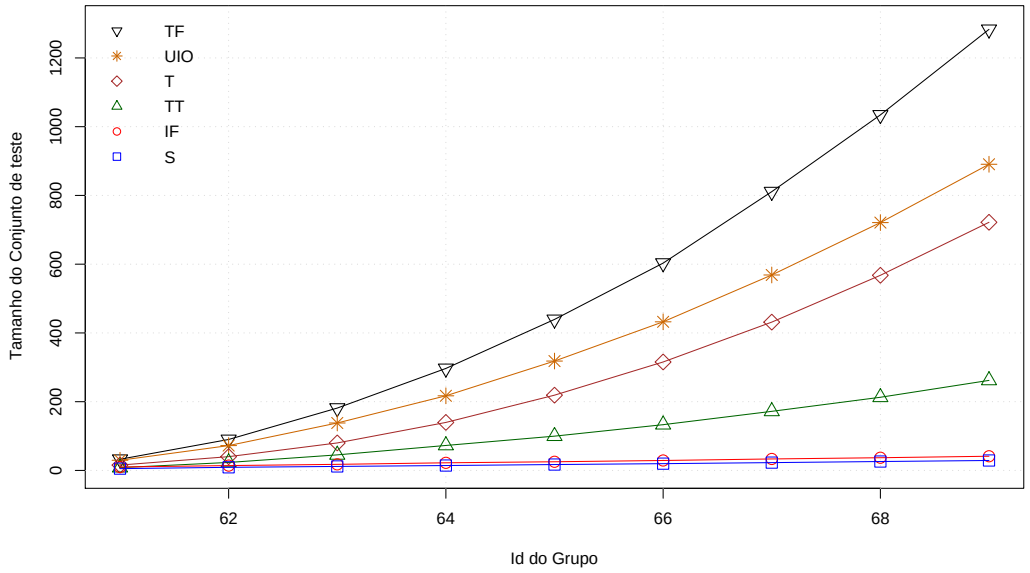

Figura 6.16: Variação do n, k e l para métodos e critérios não n-completos

Discussão dos Resultados

Com relação aos parâmetros de MEFs e aos conjuntos de teste os seguintes resultados são observados:

- Todos os conjuntos de teste crescem quase linearmente com o incremento do número de estados. A explicação para isso é trivial, pois nesse caso mais estados e mais transições necessitam ser testados, o que, em geral, aumenta o número e o tamanho das sequências de teste.

- Quase todos os conjuntos de teste crescem quase linearmente com o incremento do número de entradas. A explicação para isso também é trivial, pois nesses casos o acréscimo do número de entradas produz mais transições a serem testadas. Somente para os critérios de cobertura S e IF a curva se mantêm praticamente constante, pois os requisitos desses critérios são baseados nos estados e não nas transições.

- Em contrapartida, os tamanhos dos conjuntos de teste tendem a decrescer com o incremento do número de saídas. Observa-se que não ocorre aumento do número de transições quando se aumenta o número de saídas de uma MEF. A redução do tamanho dos conjuntos de teste também envolve o tamanho das sequências de separação encontradas. Elas tendem a ser menores, pois torna-se mais fácil distinguir um estado do outro com o acréscimo do número de saídas. Os critérios S e T e o método TT mantêm a curva praticamente constante, pois eles não necessitam de sequências de separação de estados.

- Como visto, com o incremento do número de estados ou entradas separadamente os conjuntos de teste aumentam. É trivial que o incremento dos dois parâmetros aumentasse ainda mais o tamanho dos conjuntos de teste. O incremento do número de estados ou incremento do número de entradas, junto com o incremento do número de saídas, causa uma diminuição do crescimento da curva do tamanho dos conjuntos de teste.

Com relação aos parâmetros de MEFs e aos métodos n-completos os seguintes resultados são observados:

- Como esperado, o método W gera os maiores conjuntos de teste exceto em dois casos. O primeiro caso é para MEFs de 45 e 50 estados (grupos 14 e 15) que o método UIOv gera os maiores conjuntos de teste. Com relação aos outros métodos, a curva do método UIOv cresce mais rápido com o aumento do número de estados. Como visto na Seção 4.2.3 a fase 1 do método UIOv consistem em concatenar todas sequências de Q com as sequências UIO de todos os estados da MEF. Em MEFs com muitos estados, ocorre um aumento considerável do tamanho médio dos conjuntos de teste e, em alguns casos, esse conjunto é maior que o conjunto gerado pelo método W. O segundo caso é para MEFs acima de 16 estados e 8 saídas (grupos 57, 58 e 59). Nesse caso, o método HSI é maior que o método W. Observa-se

que, pela implementação utilizada do método HSI (Cutigi e Simão, 2006) o conjunto HSI nesses casos é maior, em média, que o conjunto W.

- Em geral, os tamanhos dos conjuntos de teste do método SC e do método HSI são parecidos, sendo menos custosos apenas que o método W, exceto no caso já explicado que o método HSI supera o método W.

- Em todos os 7 casos de MEFs o método CS é o menos custoso, principalmente por seu conjunto possuir apenas uma sequência e não utilizar a função *reset*. Todavia, esse método, como será visto adiante, é o menos aplicável de todos.

- Em geral, o método H é o método sempre aplicável com melhores resultados exceto em dois casos. O primeiro caso é para MEFs com mais de 25 estados (grupos 10 a 15), o método H gera conjuntos de teste maiores que o método Wp. Isso se deve à implementação do método H (Stuchi e Simão, 2008) utilizada neste trabalho. Nesse método, durante a geração dos testes, heurísticas são utilizadas na escolha *on-the-fly* das sequências de separação de estados. A depender da heurística utilizada, o conjunto produzido sofre alterações. Provavelmente, para MEFs acima de 25 estados, a implementação deste trabalho nem sempre faz uma boa escolha *on-the-fly* das sequências de separação de estados em comparação à implementação utilizada em Dorofeeva et al. (2005a). O segundo caso é para MEFs com relação ao incremento do número de entradas. Com relação aos outros métodos, percebe-se que o custo do método UIOv diminui consideravelmente com o incremento do número de entradas e saídas. Para os grupos 21 a 24, 56 a 60 e 65 a 69 os conjuntos de teste gerados pelo método UIOv é menos custoso, em média, que o método H. A razão plausível para esse acontecimento é o encurtamento de algumas sequências UIO para alguns estados da MEF, quando o número de entradas ou saídas é incrementado.

- O método DS somente é mais custoso que o método H nos casos 3 e 5 e os tamanhos médios dos conjuntos de teste gerados por ambos os métodos ficam próximos nos casos 6 e 7. Nota-se que o incremento do número de saídas reduz o tamanho dos conjuntos de teste do método H mais rapidamente do que o método DS. Assim como o método CS, o método DS nem sempre é aplicável.

- Em geral, os tamanhos dos conjuntos de teste dos métodos Wp e UIOv ficam em uma posição intermediária, exceto para os casos explicados nos itens anteriores. Conclui-se que o método que mais sofre alterações relacionados aos parâmetros de MEF é o método UIOv. Prova-se essa afirmação observando a inclinação da curva desse método em vários casos. Observa-se que no caso 1 o início da curva (Figura 6.1) o método UIOv ganha dos métodos W, HSI, SC e Wp e, no final, a curva (Figura 6.2) do método UIOv ultrapassa a curva dos outros métodos se tornando o mais custoso. Nos casos 2, 3, 5 e 7, nota-se que esse método só é menos custoso que o método W no início da curva, mas no final diminui o custo em relação a

outros métodos. No caso 7, por exemplo, do segundo método mais custoso para o grupo 61, o método UIOv se torna o segundo método menos custoso a partir do grupo 66, perdendo só para o método CS (ver 6.15).

Com relação aos parâmetros de MEFs e aos métodos não n-completos os resultados são mais homogêneos. Observa-se que:

- Era esperado que métodos ou critérios menos exigentes gerassem conjuntos de teste menores. Critérios cujos requisitos de cobertura são baseados em estados (critérios S e IF) geram conjuntos de teste menores que métodos de cobertura de transições (critérios T e TF e o método TT) que, por sua vez, geram conjuntos de teste menores que o método UIO. A exceção dessa última afirmação é o critério TF. Ele gera conjuntos de teste maiores que o método UIO, pois a geração desse conjunto é baseada na minimização do conjunto gerado pelo método HSI. As sequências de separação do método HSI, em geral, são maiores que as sequências UIO. Para os métodos não n-completos a ordem crescente dos tamanhos dos conjuntos de teste ($S < IF < TT < T < UIO < TF$) é idêntica aos 7 casos.

6.5 Aplicabilidade dos Métodos

Alguns métodos exigem que as MEFs possuam determinadas propriedades ou determinadas sequências para serem aplicáveis. Por esse motivo e a fim de abranger todos os métodos estudados, somente MEFs determinísticas, completas e minimais são utilizadas nos experimentos. Todavia, os métodos TT, DS, CS, UIO e UIOv não são aplicáveis para todas as MEFs geradas neste trabalho, pois:

- O método TT exige a propriedade fortemente conexa.
- O método DS exige que a MEF possua uma sequência DS.
- O método CS exige, além da propriedade fortemente conexa, que a MEF também possua uma sequência DS.
- Os métodos UIO e UIOv exigem que cada estado da MEF possua uma sequência UIO.

A fim de responder à questão (*iii*) 'Quão aplicáveis são os métodos TT, CS, DS, UIO e UIOv?', para todos os casos de MEFs, são calculadas a quantidade de MEFs que não podem ser aplicadas para esses cinco métodos mencionados.

Os casos são separados em grupos para analisar como a aplicabilidade desses métodos é afetada com a variação dos parâmetros de MEF n, k e l. Nas Tabelas 6.2, 6.3 e 6.4 ilustra-se a aplicabilidade desses métodos por grupos. Por exemplo, 968 MEFs são aplicáveis ao método TT para o grupo 1 e, obviamente, 32 não são aplicáveis, pois lembrando que cada grupo possui 1000 MEFs.

Caso 1. Variação do número de estados (n)

Grupo	n	k	l	t	TT	UIO e UIOv	DS	CS
1	4	4	4	16	**968**	**1000**	**992**	**961**
2	6	4	4	24	**936**	**996**	**942**	**897**
3	8	4	4	32	**934**	**995**	**868**	**846**
4	10	4	4	40	**958**	**992**	**717**	**691**
5	12	4	4	48	**952**	**978**	**705**	**660**
6	14	4	4	56	**951**	**992**	**644**	**614**
7	16	4	4	64	**952**	**963**	**438**	**411**
8	18	4	4	72	**931**	**952**	**423**	**377**
9	20	4	4	80	**956**	**969**	**241**	**216**
10	25	4	4	100	**952**	**971**	**169**	**160**
11	30	4	4	120	**960**	**947**	**181**	**180**
12	35	4	4	140	**956**	**929**	**18**	**16**
13	40	4	4	160	**947**	**951**	**29**	**29**
14	45	4	4	180	**940**	**939**	**4**	**4**
15	50	4	4	200	**961**	**857**	**1**	**1**

Tabela 6.2: Variação do n para avaliar aplicabilidade dos métodos TT, UIO e UIOv, DS e CS

Nesse experimento utilizam-se as MEFs do caso 1. Pela Tabela 6.2, observa-se que o incremento do número de estados não interfere na geração aleatória de MEFs fortemente conexas, que é uma exigência para o método TT ser aplicável. Contudo, esse incremento interfere na aplicabilidade dos métodos UIO e UIOv, DS e CS, pois torna-se mais difícil que a MEF possua uma sequência DS e que todos seus estados possuam uma sequência UIO. Suponha-se uma MEF M com n estados sendo $S = \{s_1, s_2, .., s_i, .., s_j, .., s_m, .., s_n\}$. A possibilidade das sequências que distinguem um estado s_i de um estado s_j não distinguirem esse estado s_i de um estado s_m cresce com o incremento do número de estados. Nota-se que para MEFs grandes com $n = 50$, $k = 4$ e $l = 4$ (grupo 15), somente 0,1% das MEFs são aplicáveis pelos métodos DS e CS.

Caso 2. Variação do número de entradas (k)

Grupo	n	k	l	t	TT	UIO e UIOv	DS	CS
16	10	2	2	20	**572**	**488**	**253**	**198**
17	10	3	2	30	**785**	**666**	**228**	**191**
18	10	4	2	40	**959**	**865**	**270**	**264**
19	10	5	2	50	**981**	**932**	**368**	**358**
20	10	6	2	60	**997**	**971**	**368**	**365**
21	10	7	2	70	**1000**	**991**	**391**	**391**
22	10	8	2	80	**1000**	**989**	**506**	**506**
23	10	9	2	90	**1000**	**1000**	**513**	**513**
24	10	10	2	100	**1000**	**1000**	**519**	**519**

Tabela 6.3: Variação do k para avaliar aplicabilidade dos métodos TT, UIO e UIOv, DS e CS

Nesse experimento utilizam-se as MEFs do caso 2. Pela Tabela 6.3, observa-se que o incremento do número de entradas interfere na aplicabilidade dos cinco métodos. Como todas as MEFs são completas, o incremento do número de entradas implica no aumento de transições, o que aumenta a possibilidade da MEF gerada aleatoriamente ser fortemente conexa. A possibilidade da MEF possuir uma sequência DS e de todos seus estados possuírem uma sequência UIO também é aumentada. Suponha-se a mesma MEF M do **caso** anterior. A possibilidade das sequências que distinguem um estado s_i de um estado s_j também distinguirem esse estado s_i de um estado s_m cresce com o incremento do número de entradas. Nota-se que 100% das MEFs dos grupos 23 e 24 os métodos TT, UIO e UIOv são aplicáveis. Percebe-se um aumento de MEFs aplicáveis para todos o métodos.

Caso 3. Variação do número de saídas (l)

Grupo	n	k	l	t	**TT**	**UIO e UIOv**	**DS**	**CS**
25	10	2	2	20	**635**	**406**	**191**	**137**
26	10	2	3	20	**604**	**686**	**454**	**280**
27	10	2	4	20	**598**	**763**	**619**	**396**
28	10	2	5	20	**625**	**846**	**714**	**458**
29	10	2	6	20	**612**	**884**	**776**	**484**
30	10	2	7	20	**616**	**920**	**822**	**500**
31	10	2	8	20	**596**	**925**	**856**	**511**
32	10	2	9	20	**588**	**960**	**902**	**538**
33	10	2	10	20	**607**	**946**	**902**	**555**

Tabela 6.4: Variação do l para avaliar aplicabilidade dos métodos TT, UIO e UIOv, DS e CS

Nesse experimento utilizam-se as MEFs do caso 3. Pela Tabela 6.4, observa-se que o incremento do número de saídas não interfere na geração aleatória de MEFs fortemente conexas, que é uma exigência para o método TT ser aplicável. Contudo, esse incremento interfere na aplicabilidade dos métodos UIO e UIOv, DS e CS, pois a possibilidade da MEF possuir uma sequência DS e de todos seus estados possuírem uma sequência UIO também é aumentada. Suponha-se a mesma MEF M dos casos anteriores. A possibilidade das sequências que distinguem um estado s_i de um estado s_j também distinguirem esse estado s_i de um estado s_m cresce com o incremento do número de saídas. Por exemplo, uma MEF com $n = 4$ e $l = 4$ é possível possuir uma sequência DS de tamanho 1, desde que para mesma entrada x as saídas sejam diferentes para cada um dos quatro estados. Esse fato é impossível se $l < 4$, pois no mínimo para dois estados a entrada x terá a mesma saída y. Nota-se que a quantidade de MEFs que não são fortemente conexas é de **39,10**%, pois para $k = 2$, é alta a possibilidade de um estado s_i não alcançar todos os outros n estados de M.

Discussão

- Como esperado, é comprovado por esses experimentos que as MEFs aplicáveis pelo método CS também são aplicáveis pelos métodos TT, DS, UIO e UIOv, assim como as MEFs aplicáveis pelo método DS também são para os métodos UIO e UIOv. A explicação é trivial, pois toda sequência DS é uma sequência UIO. Se uma MEF possui uma sequência DS obrigatoriamente todos seus estados possuem uma sequência UIO.

- No total, o método TT, os métodos UIO e UIOv, o método DS e o método CS não são aplicáveis para **14,86**%, **9,52**%, **42,33**% e **50,84**% das MEFs geradas neste trabalho, respectivamente. Conclui-se que apesar do método CS produzir os menores conjuntos de teste para os métodos n-completos ele é aplicável para menos da metade das MEFs geradas neste trabalho.

- Foram feitos experimentos para os demais casos, porém os resultados pouco contribuíram para mais conclusões do que as já descritas nos três primeiros casos e, portanto, não foram apresentados na presente dissertação.

6.6 Avaliação da Eficácia

Como dito anteriormente, o termo eficácia diz respeito à capacidade do método de detectar erros. Para isso os métodos n-completos não necessitam fazer essa avaliação, pois todos são completa e igualmente eficazes. Tais métodos possuem o poder de encontrar qualquer erro em MEFs descritos na Seção 3.3, apesar, de como visto neste capítulo, exigirem custos diferentes. São avaliados os métodos não n-completos (UIO e TT) e os quatro critérios de cobertura (S, IF, T e TF) estudados neste trabalho.

Para esses métodos e critérios utiliza-se a técnica de análise de mutantes, juntamente com alguns operadores de mutação definidos em Fabbri et al. (1994) para avaliação experimental da eficácia. A qualidade dos conjuntos de teste gerados por cada critério e método é avaliada pela capacidade de matar os mutantes. Além dessa avaliação, para o critério TF e para o método UIO é verificado se os conjuntos de teste gerados satisfazem as condições de suficiência descritas em (Simão e Petrenko, 2007).

No trabalho de Simão et al. (2009), utilizou-se uma abordagem semelhante a de verificar a n-completude dos conjuntos gerados pelos critérios de cobertura para avaliar a eficácia. Simão et al. (2009) concluíram que para os critérios S, IF e T são mínimos os conjuntos gerados para MEFs completas serem n-completos, assim como para o critério TF, que para MEFs pequenas e com grau de completude baixo a porcentagem de conjuntos n-completos gerados é relativamente alta, chegando acima dos 80%, porém com o incremento do número de estados ou do grau de completude essa porcentagem cai rapidamente. Dorofeeva et al. (2005a) descrevem uma avaliação de eficácia somente para o método UIO. Utilizou-se especificações pequenas ($n = 2$ a 6, $k = 2$ e

$l = 2$) e de um total de 193 MEFs, 41% dos conjuntos de teste gerados pelo método UIO, a partir dessas MEFs, são incompletos. No artigo, não há detalhes de como foi conduzida essa avaliação.

Somente para os casos 1, 2 e 3 é feita a avaliação de eficácia, sendo que para o caso 1 os grupos 10 a 15 também não são avaliados. Não são conduzidos experimentos para os demais casos em razão da quantidade de MEFs estudadas que necessitaria de tempo dispendioso no processamento dos dados da análise de mutantes e verificação das condições de suficiência.

6.6.1 Análise de Mutantes

Uma forma de avaliar a eficácia dos métodos e critérios não n-completos é baseada na técnica de análise de mutantes. Por meio de operadores de mutação, erros são propositalmente inseridos nas MEFs geradas, criando um conjunto de mutantes. Cada MEF gerada representa uma especificação e cada mutante representa um possível erro de implementação da MEF. Executa-se os CTs gerados pelos métodos e critérios na MEF gerada e em cada mutante. As saídas dos CTs de cada mutante são confrontadas com a saída da MEF gerada com a finalidade de encontrar os erros inseridos, matando, ou não, tais mutantes.

A Seção 5.2.1, que trata da reengenharia da ferramenta Plavis/FSM, aborda como os mutantes são gerados e executados. Como visto, utilizam-se nessa avaliação dois operadores: o operador *output-exchanged* que gera t mutantes com erros de saída; e o operador *destination-exchanged* que gera $t \times (n-1)$ mutantes com erros de transferência. Portanto para cada MEF gerada, são gerados $t + t(n-1) = tn$ mutantes. Contabiliza-se a quantidade de mutantes mortos e o escore de mutação. Em razão da impossibilidade da geração de mutantes equivalentes, o escore de mutação é calculado somente pela razão dos mutantes mortos pela quantidade de mutantes gerados.

Caso 1. Variação do número de estados (n)

Para este experimento, utilizam-se as MEFs dos grupos 1 a 9 do caso 1. Nas Figuras 6.17 e 6.18 apresentam-se gráficos, nos quais pode-se observar que há um ligeiro aumento do escore de mutação em ambos operadores para o critério S; não há impacto algum para o critério IF e UIO; e para os critérios T e TF ocorre uma redução do escore para o operador *destination-exchanged*, ao contrário do critério TT que ocorre um aumento.

Como visto na Figura 6.2, os tamanhos dos conjuntos de teste aumentam com o incremento do número de estados. Verifica-se que somente para o critério S e método TT esse aumento do conjunto de teste é acompanhado do aumento da capacidade em revelar erros. MEFs que possuem mais estados implica em uma dificuldade maior em revelar erros de transferência, pois a possibilidade de haver um erro de estado destino de uma transição aumenta. Devido a esse fato, a capacidade de identificar tais erros é reduzida para os critérios T e TF.

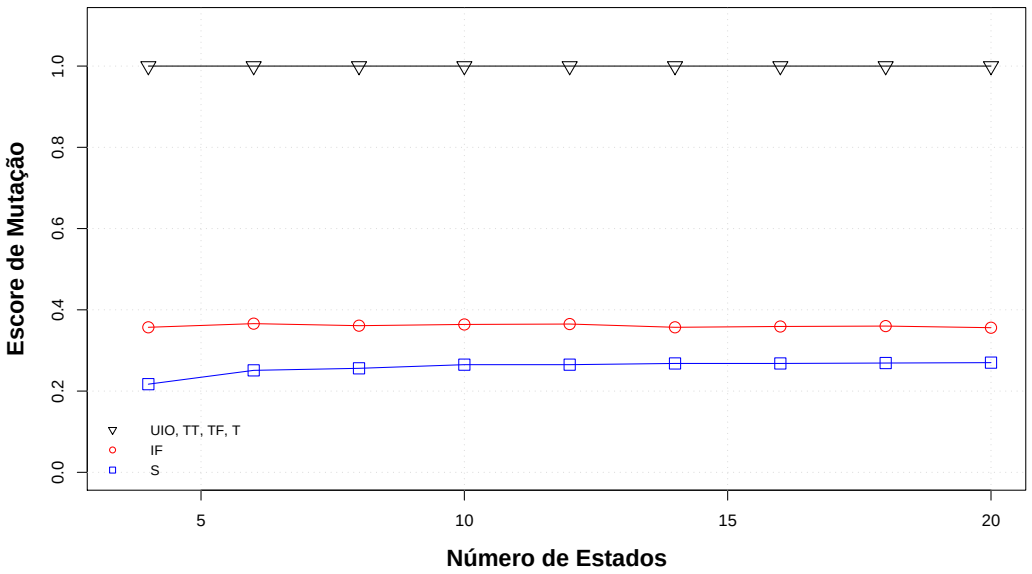

Figura 6.17: Variação do n × Escore de Mutação para o operador *output-exchanged*

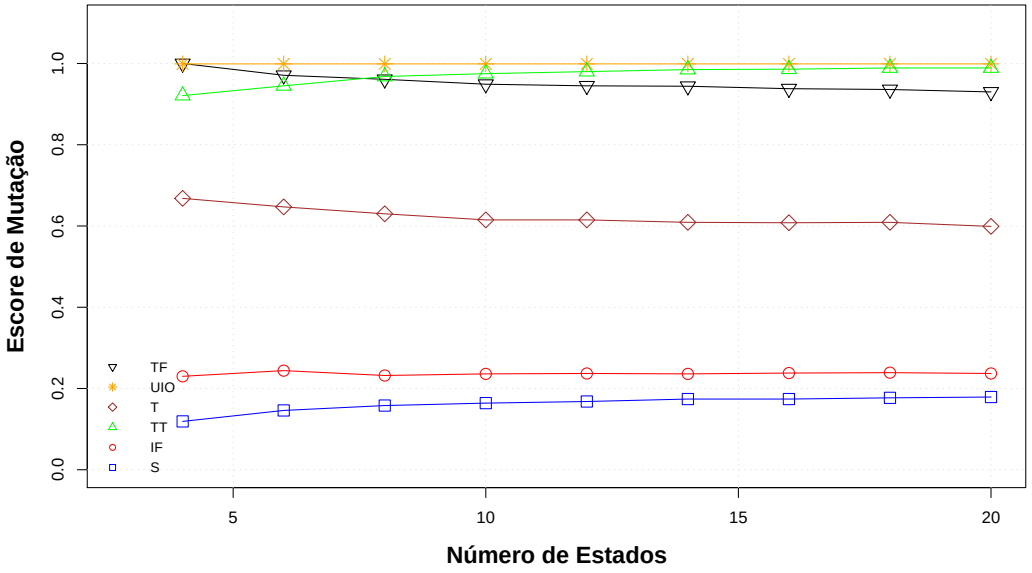

Figura 6.18: Variação do n × Escore de Mutação para o operador *destination-exchanged*

Neste experimento, o método TT é mais eficaz que os critérios T e TF, mesmo possuindo tamanho do conjunto de teste menor. O método TT só possui uma sequência no seu conjunto de teste. Conclui-se que, em geral, sequências maiores são mais eficazes que várias sequências menores em revelar erros de transferência com a variação do número de estados.

Para o método UIO, a diferença do escore de mutação é mínima e desprezível (quarta casa decimal), para todas as variações do número de estados o escore fica acima de 0,9990.

Caso 2. Variação do número de entradas (k)

Para este experimento, utilizam-se as MEFs do caso 2. Nas Figuras 6.19 e 6.20 apresentam-se gráficos, nos quais pode-se observar que para o método TT e critérios S, IF e T há uma diminuição do escore de mutação com o aumento do número de entradas, ao contrário do critério TF. Novamente, para todas as variações do número de entradas os escores do método UIO ficaram perto do 1, sempre acima de 0,9950.

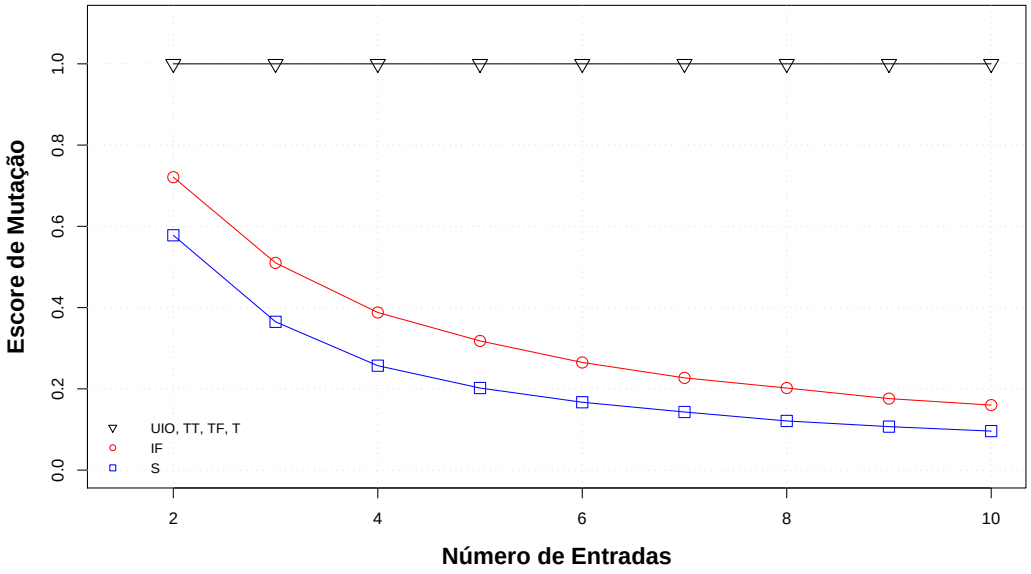

Figura 6.19: Variação do k × Escore de Mutação para o operador *output-exchanged*

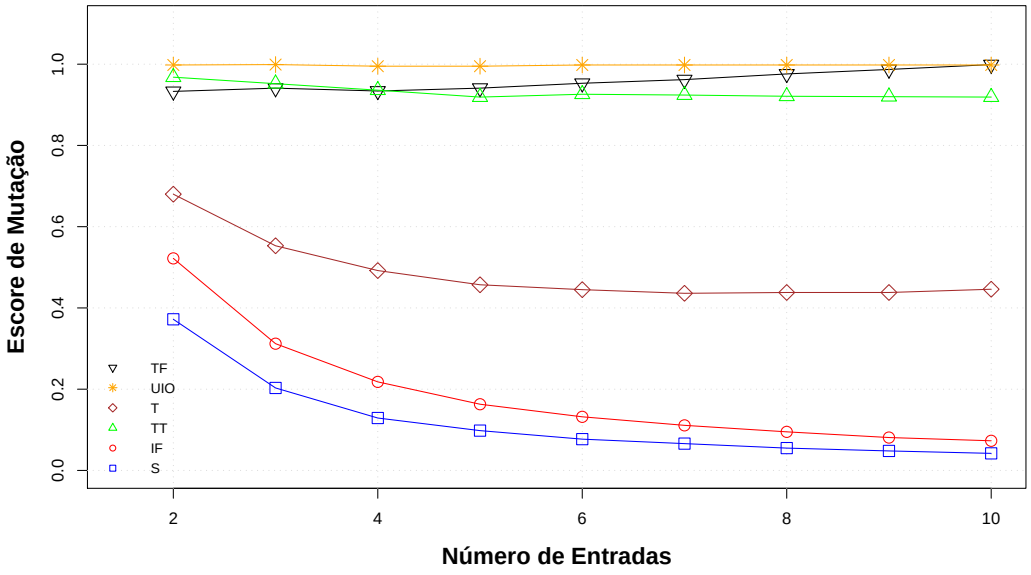

Figura 6.20: Variação do k × Escore de Mutação para o operador *destination-exchanged*

O aumento do k implica no aumento do número de transições, consequentemente aumento do número de mutantes. Na Figura 6.6 ilustra-se que os conjuntos de teste dos critérios S e IF se mantêm constantes, portanto diminui a capacidade em revelar erros.

Os tamanhos dos conjuntos de teste dos critérios T e TF e do método TT crescem com o incremento do número de entradas, contudo só o crescimento do critério TF é acompanhado de um crescimento da capacidade em revelar erros. Como visto, além de verificar as saídas das transições, é requisito de teste do critério TF cobrir os possíveis erros de transferência das transições.

Caso 3. Variação do número de saídas (l)

Para este experimento, utilizam-se as MEFs do caso 3. Nas Figuras 6.21 e 6.22 apresentam-se gráficos, nos quais pode-se concluir que para o operador *output-exchanged* o escore permanece constante para o critério S e diminui para o critério IF. Para o operador *destination-exchanged*, há um ligeiro aumento do escore de mutação com o aumento do l para os métodos e critérios.

O incremento do l não implica no aumento do número de transições, consequentemente o número de mutantes permanece o mesmo para todos os grupos.

Como ilustra na Figura 6.8, não há impactos no tamanho do conjunto de teste do critério S, por essa razão a capacidade de matar mutantes para o operador *output-exchanged* se mantém a mesma.

As sequências de separação tendem a ser menores em MEFs que possuem mais saídas. O critério IF utiliza tais sequências para verificar se o estado inicial está correto. Na Figura 6.8 também ilustra-se que o tamanho do conjunto de teste gerado pelo critério IF diminui ligeiramente com o aumento do l. Conjunto de teste menor implica em menos transições verificadas e por essa razão há uma perda na capacidade do critério IF em revelar erros para o operador *output-exchanged*.

Todavia, erros inseridos pelo operador *destination-exchanged*, em geral, necessitam das sequências de separação para serem revelados. Como visto na Seção 4.2, os critérios IF e TF e o método UIO utilizam explicitamente sequências de separação, ocorrendo um aumento da capacidade em revelar erros de transferência.

Os conjuntos de teste dos critérios S e T e o método TT se mantêm constantes com o aumento do número de saídas, como visto na Figura 6.8. Trivialmente, a probabilidade desses conjuntos de teste possuírem implicitamente sequências de separação aumenta à medida que essas sequências diminuem. Dessa forma, a capacidade em revelar erros para o operador *destination-exchanged* também é aumentada.

O aumento da capacidade do método UIO é imperceptível no gráfico por se tratar de um aumento apenas da terceira casa decimal do escore de mutação. Para $l = 2$ o escore é de 0,9976 enquanto para $l = 10$ esse escore é de 0,9999.

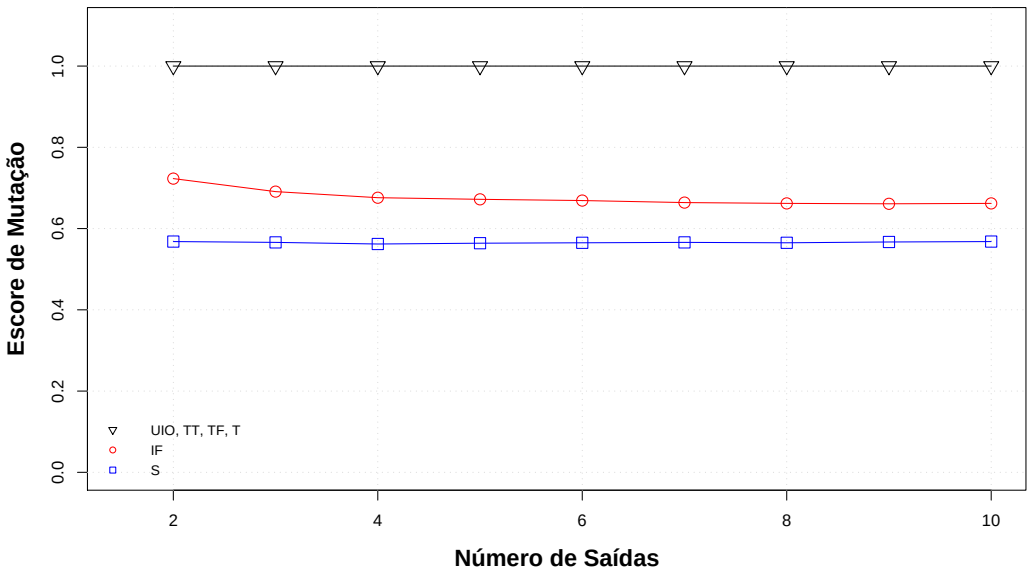

Figura 6.21: Variação do l × Escore de Mutação para o operador *output-exchanged*

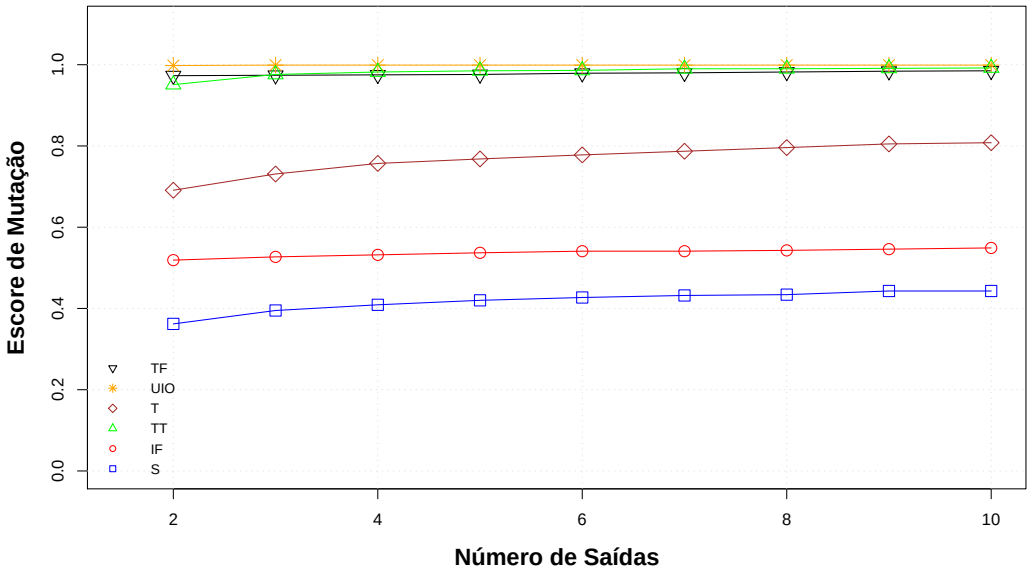

Figura 6.22: Variação do l × Escore de Mutação para o operador *destination-exchanged*

Discussão

- Em erro de transferência necessita-se verificar, além da saída da transição, se o estado destino está correto. Uma sequência de separação do estado destino para os demais necessita ser aplicada para tal verificação. É dito que erros de saída são mais fáceis de serem identificados (Chow, 1978). Por essa razão, o escore de mutação para o operador *output-exchanged*, em média, é maior que o operador *destination-exchanged*.

- Como esperado, o critério S possui a menor eficácia para matar os mutantes e o método UIO é o mais eficaz. Apesar do critério TF gerar, em média, conjuntos de teste maiores que o método UIO, sua eficácia em revelar erros não é maior. Como visto na Seção 4.2, o método UIO verifica cada transição da MEF com a sequência UIO do estado destino. O conjunto de teste adequado ao critério TF deve cobrir os requisitos de erros de transição. Mesmo utilizando sequências de separação, conjuntos de teste gerados pelo método UIO e pelo critério TF não garantem que todos os erros de transferência sejam encontrados.

- Outra relação custo/eficácia contraditória observada nesse experimento se refere ao método TT. Neste capítulo, é visto que o método TT possui conjuntos de teste menores que os critérios T e TF, porém nos 3 casos experimentados o método TT é mais eficaz que o critério T e algumas vezes mais eficaz que o critério TF.

- Como o operador de mutação *output-exchanged* só gera mutantes com erros de saída, basta o conjunto de teste exercitar todas as transições, ao menos uma vez, para tais erros serem identificados. Por essa razão, os métodos e critérios que satisfazem tais requisitos conseguem matar todos os mutantes gerados por esse operador. Os métodos TT e UIO e os critérios T e TF, como esperado, identifica 100% dos erros incluídos nesses mutantes.

- Considerando-se todos os mutantes gerados, os critérios S, IF, T e TF e os métodos TT e UIO são capazes de matar **17,89%, 24,86%, 57,20%** e **96,00%** e **96,25% 99,91%** dos mutantes respectivamente.

6.6.2 Satisfação das Condições de Suficiência

Uma das linhas de pesquisa utilizada na tentativa de diminuir o custo dos conjuntos de teste gerados pelos métodos são as condições de suficiência. Tratam-se de condições que, uma vez satisfeitas, garantem que o conjunto de teste possua a propriedade de ser n-completo. Essas condições, como o nome informa, são de suficiências e não necessárias. Ou seja, não se pode afirmar que um conjunto não é completo somente por não satisfazer essas condições. Tais condições podem ser utilizadas para avaliar a efetividade de um conjunto.

Muitos trabalhos definem um conjunto de condições de suficiência e meios de provar que um conjunto de teste é n-completo (Petrenko et al., 1996; Ural et al., 1997; Dorofeeva et al., 2005a; Simão e Petrenko, 2007). Como explicado na Seção 4.2, um método n-completo é aquele que garante que o conjunto de teste gerado sempre é n-completo; métodos ou critérios não n-completos não possuem essa garantia, o que não quer dizer que não geram conjuntos n-completos.

Para os experimentos descritos a seguir, verifica-se se os conjuntos gerados pelo critério TF e pelo método UIO satisfazem as condições de suficiência descritas em (Simão e Petrenko, 2007). Essa verificação não é feita para o método TT e para os critérios S, IF e T, pois a probabilidade deles gerarem conjuntos de teste n-completos para MEFs completas é mínima, em virtude do tamanho pequeno dos conjuntos de teste.

Caso 1. Variação do número de estados (n)

Para este experimento, utilizam-se as MEFs dos grupos 1 a 9 do caso 1. Na Tabela 6.5 apresentam-se os resultados dessa avaliação.

Id	n	k	l	t	**UIO**	**TF**
1	4	4	4	16	854 de 1000 **(85,40%)**	1000 de 1000 **(100,00%)**
2	6	4	4	24	470 de 996 **(47,18%)**	82 de 1000 **(8,20%)**
3	8	4	4	32	113 de 995 **(11,35%)**	20 de 1000 **(2,00%)**
4	10	4	4	40	41 de 992 **(4,13%)**	3 de 1000 **(0,30%)**
5	12	4	4	48	8 de 978 **(0,81%)**	0 de 1000 **(0,00%)**
6	14	4	4	56	0 de 992 **(0,00%)**	0 de 1000 **(0,00%)**
7	16	4	4	64	0 de 963 **(0,00%)**	0 de 1000 **(0,00%)**
8	18	4	4	72	0 de 952 **(0,00%)**	0 de 1000 **(0,00%)**
9	20	4	4	80	0 de 969 **(0,00%)**	0 de 1000 **(0,00%)**

Tabela 6.5: Variação do n para avaliar se satisfaz as condições de suficiência

Caso 2. Variação do número de entradas (k)

Para este experimento, utilizam-se as MEFs do caso 2. Na Tabela 6.6 apresentam-se os resultados dessa avaliação.

Id	n	k	l	t	**UIO**	**TF**
16	10	2	2	20	0 de 488 **(0,00%)**	1 de 1000 **(0,01%)**
17	10	3	2	30	0 de 666 **(0,00%)**	0 de 1000 **(0,00%)**
18	10	4	2	40	0 de 865 **(0,00%)**	0 de 1000 **(0,00%)**
19	10	5	2	50	0 de 932 **(0,00%)**	0 de 1000 **(0,00%)**
20	10	6	2	60	0 de 971 **(0,00%)**	0 de 1000 **(0,00%)**
21	10	7	2	70	0 de 991 **(0,00%)**	2 de 1000 **(0,20%)**
22	10	8	2	80	0 de 989 **(0,00%)**	18 de 1000 **(1,80%)**
23	10	9	2	90	0 de 1000 **(0,00%)**	181 de 1000 **(18,10%)**
24	10	10	2	100	0 de 1000 **(0,00%)**	726 de 1000 **(72,60%)**

Tabela 6.6: Variação do k para avaliar se satisfaz as condições de suficiência

Caso 3. Variação do número de saídas (l)

Para este experimento, utilizam-se as MEFs do caso 3. Na Tabela 6.7 apresentam-se os resultados dessa avaliação.

Id	n	k	l	t	UIO	TF
25	10	2	2	20	7 de 406 **(1,72%)**	14 de 1000 **(1,40%)**
26	10	2	3	20	63 de 686 **(9,18%)**	18 de 1000 **(1,80%)**
27	10	2	4	20	190 de 763 **(24,90%)**	22 de 1000 **(2,20%)**
28	10	2	5	20	330 de 846 **(39,00%)**	33 de 1000 **(3,30%)**
29	10	2	6	20	458 de 884 **(51,80%)**	36 de 1000 **(3,60%)**
30	10	2	7	20	534 de 920 **(58,04%)**	56 de 1000 **(5,60%)**
31	10	2	8	20	579 de 925 **(62,59%)**	75 de 1000 **(7,50%)**
32	10	2	9	20	678 de 960 **(70,62%)**	59 de 1000 **(5,90%)**
33	10	2	10	20	690 de 946 **(72,93%)**	79 de 1000 **(7,90%)**

Tabela 6.7: Variação do l para avaliar se satisfaz as condições de suficiência

Discussão

Analisando as três tabelas, pode-se observar que:

- Para MEFs com $n = 4$ o método UIO e o critério TF a porcentagem de conjuntos de teste que satisfazem as condições de suficiência é alta, chegando a 100% para o critério TF, contudo essa porcentagem cai bruscamente com o incremento do número de estados, principalmente para o critério TF.

- Nenhum conjunto de teste gerado pelo método UIO do caso 2 satisfaz as condições de suficiência. Em contrapartida, observa-se que o incremento do número de entradas aumenta a probabilidade do conjunto gerado pelo critério TF ser garantidamente n-completo, concordando com o que é concluído na seção anterior, que mostra que com o incremento do número de entradas ocorre um aumento da capacidade de matar mutantes para o critério TF.

- Com o incremento do número de saídas, ocorre um aumento na geração de conjuntos n-completos tanto para o método UIO como para o critério TF. Esse aumento é mais acentuado para o método UIO.

- No total, **20,83**% e **9,46**% dos conjuntos gerados pelo método UIO e critério TF satisfazem as condições de suficiência, respectivamente. Conclui-se que mesmo gerando conjuntos menores, em geral, para essa análise o método UIO é mais eficaz que o critério TF, porém

o critério TF sempre é aplicável. Repetindo, para **9,52**% das MEFs geradas neste trabalho, não é possível aplicar o método UIO.

6.7 Considerações Finais

Neste capítulo discutiram-se todos os detalhes da condução e análise dos resultados do estudo proposto neste trabalho, cujo objetivo é avaliar em termos de custo e eficácia os métodos de geração e os critérios de teste baseados em MEFs.

Com o apoio do protótipo desenvolvido e de um *cluster* é possível realizar uma série de experimentos divididos em três passos. O primeiro passo dos experimentos é a geração e o agrupamento de uma amostra de MEFs. O segundo passo objetiva a avaliação do custo (tamanho do conjunto de teste) e da aplicabilidade dos métodos e critérios. O terceiro passo objetiva a avaliação da eficácia sendo utilizada duas métricas: o escore de mutação e as condições de suficiência.

Constata-se que a utilização de uma ferramenta de apoio é imprescindível para a condução de experimentos desse porte. Não foi encontrada nenhuma ferramenta que permita tal funcionalidade o que motivou o desenvolvimento de uma ferramenta a partir da reengenharia da ferramenta Plavis/FSM.

Outra constatação é que vários fatores influenciam nos resultados do custo e da eficácia dos métodos e critérios. Dentre esses fatores, os principais estudados foram os parâmetros de MEFs, mas pode-se citar também propriedades das MEFs, heurísticas de implementação e operadores de mutação.

No capítulo seguinte, são apresentadas as conclusões deste trabalho. As contribuições resultantes deste projeto são declaradas e algumas direções para trabalhos futuros são propostas.

CAPÍTULO

7

Conclusões

MEFs são modelos formais amplamente utilizadas em diversas áreas como na Engenharia Elétrica, Linguística, Lógica Matemática, entre outras. Na computação MEFs podem representar modelagem de hardware, protocolos de comunicação e algoritmos, entre outras aplicações. Ultimamente, na Engenharia de Software, MEFs são cada vez mais utilizadas na modelagem de sistemas distribuídos, sistemas embarcados e, principalmente, para descrever o comportamento de sistemas reativos. A utilização de MEFs nem sempre é feita de forma trivial e muito menos barata. Para sistemas reativos de nível crítico, o uso desse modelo é relevante, pois o teste pode ser automatizado e o conjunto de casos de teste gerado pode ser completo para revelar os erros de uma implementação. Há diversos métodos de geração e critérios de cobertura de teste baseado em MEFs propostos na literatura ou trabalhos que definem melhorias aos métodos e critérios, porém poucos trabalhos avaliaram as vantagens e desvantagens de tais métodos e critérios.

Duas características que devem ser investigadas são o custo e a eficácia dos métodos de geração e critérios de cobertura de teste baseado em MEFs. O custo mede o esforço necessário para aplicação do conjunto de teste gerado pelos métodos e critérios e a eficácia mede a capacidade desses em detectar erros. Este trabalho visa a contribuir com a avaliação do custo e da eficácia dos métodos de geração TT, UIO, UIOv, W, Wp, DS, CS, HSI e H; e dos critérios de cobertura S, IF, T e TF.

Para tal avaliação foi desenvolvido um protótipo que dê apoio à condução de experimentos, uma vez que nenhuma ferramenta com tal finalidade havia sido identificada. A produção e a coleta de dados é automatizada pelo protótipo desenvolvido.

7.1 Contribuições

A principal contribuição deste trabalho é a avaliação de métodos e critérios de teste baseado em MEFs em termos de custo e eficácia. Essa avaliação baseia-se em resultados de experimentos obtidos com o apoio de um protótipo de ferramenta. A análise desses resultados viabiliza a determinação de estratégias de teste baseado em MEFs a depender do contexto envolvido. Outra contribuição, não menos importante, é justamente o desenvolvimento desse protótipo a partir de uma reengenharia da ferramenta Plavis/FSM. Até a presente data desta escrita, não havia sido identificada uma ferramenta de teste baseado em MEFs que permitia a condução de experimentos. Em cooperação ao projeto *Qualipso*, utilizou-se no desenvolvimento a tecnologia de *web services* que permite a disponibilização dos serviços pela *web*.

Com relação ao custo dos métodos e critérios são descritos sete casos que diferem em relação aos parâmetros de MEF que são variados. Para os métodos n-completos, o método W é o que obtém o maior custo dos conjuntos de teste, seguido dos métodos HSI e SC. Os métodos CS, DS e H geram os melhores resultados, porém somente o método H é sempre aplicável. O método UIOv foi um caso a parte, em alguns casos gera conjuntos de teste maiores que o método W e em outros gera conjuntos menores que o método H. Para os métodos não n-completos provou-se que os testes são mais custosos para os requisitos de teste mais difíceis de serem cobertos. O critério S obtém conjuntos de teste pequenos enquanto o critério TF e o método UIO obtém conjuntos maiores.

Com relação à eficácia foi conduzido um experimento utilizando a técnica de análise de mutantes somente para os métodos e critérios não n-completos. Conclui-se que vários fatores influenciam no aumento ou redução da capacidade em revelar erros dos métodos e critérios como os parâmetros de MEFs e os operadores de mutação. Nesse experimento, três parâmetros de MEF são variados separadamente e dois operadores são utilizados. Destaca-se como resultado final desse experimento o método TT, que mesmo possuindo conjuntos de tamanho médio menores que os do critério TF, é ligeiramente mais eficaz por essa análise e o método UIO obtém escore de mutação próximo ao 1,0.

Outro experimento foi conduzido para avaliar a eficácia utilizando as condições de suficiência descritas em Simão e Petrenko (2008). Dessa vez somente são avaliados o método UIO e o critério TF. Em geral, o método UIO, mesmo possuindo um conjunto de teste menor que o critério TF, se saiu melhor; porém esse método nem sempre é aplicável.

7.2 Dificuldades e Limitações

O presente trabalho apresentou algumas dificuldades e limitações, além das descritas na Seção 5.3 que refere-se aos problemas encontrados no desenvolvimento do protótipo.

Certamente, a principal dificuldade está nos detalhes dos algoritmos dos métodos de geração definidos em seus artigos originais. Na Seção 4.6, é visto que dois trabalhos relacionados a este

projeto de mestrado foram identificados. As implementações dos critérios de cobertura investigados em Simão et al. (2009) foram reusadas neste presente trabalho. Em Dorofeeva et al. (2005a), alguns métodos de geração são avaliados por meio de experimentos. Contudo, a indisponibilidade de implementações executáveis desses métodos inviabilizou uma comparação efetiva com o trabalho de Dorofeeva et al. (2005a). Algumas implementações dos métodos foram reusadas de trabalhos anteriores a este projeto de mestrado (PLAVIS, 2005; Cutigi e Simão, 2006; Stuchi e Simão, 2008) e para alguns métodos seus algoritmos foram implementados pelo autor dessa dissertação. Utilizou-se como referência as informações contidas nos trabalhos originais, porém tais informações sobre o algoritmo desses trabalhos nem sempre são completas. Como visto, diversas heurísticas podem ser utilizadas nas implementações dos métodos, principalmente na geração de sequências exigidas para a aplicação de tais métodos. Essas sequências, geralmente, são infinitas e a escolha de qual utilizar interfere nos resultados finais da geração.

Com relação ao desenvolvimento do protótipo, ressalta-se a dificuldade de adequar os métodos reusados, implementados em diversas linguagens de programação, ao protótipo. Cada implementação possui formatos de entradas, processamento e formatos de saídas distintos. Outros fatos que dificultaram na reengenharia estão descritos na Seção 5.2.1.

Há também outras limitações inerentes ao protótipo discutidas na Seção 5.3. Com relação às propriedades de MEFs o protótipo só aceita MEFs determinísticas, minimais, completas e conexas. Em razão do formato do arquivo utilizado, os parâmetros de MEFs são limitados: $n \leq 100$ e k e $l \leq 10$.

7.3 Trabalhos Futuros

As funcionalidades e a interface com o usuário podem ser aprimoradas em versões futuras do protótipo. A aplicação cliente desenvolvida pode ser continuada para o desenvolvimento de uma ferramenta de teste baseado em MEFs completa por meio de *web services* para o teste baseado em MEFs. Outros métodos e critérios podem ser integrados à ferramenta e espera-se que a condução de novos experimentos produza novos resultados acrescentando, assim, contribuições a este trabalho. Contornar as limitações sobre as propriedades de MEFs aceitas pelo protótipo, assim como as limitações sobre os parâmetros de MEFs impostas pelo formato do arquivo de entrada, também, é uma possibilidade de melhoria.

Para o experimento envolvendo análise de mutantes sugere-se a replicação dos experimentos com outros operadores de mutação, os quais podem gerar MEFs mutantes parciais, não-minimais e não determinísticas. Utilizar tais propriedades nas especificações de MEFs também é sugerido, visto que essas MEFs se aproximam mais da realidade trabalhada por testadores do que MEFs completas e minimais. Neste experimento, só é avaliado a variação de um parâmetro de MEF sendo os demais fixados em um valor. Seria interessante avaliar resultados com mais de um parâmetro sendo variado assim como o experimento de avaliação do custo, ou seja, conduzir experimentos

para os casos 4, 5, 6 e 7. A condução desses demais casos também seria interessante para o experimento que analisa as condições de suficiência do método UIO e do critério TF.

No âmbito do grupo de pesquisa em que este trabalho é desenvolvido, dois outros trabalhos de mestrado vem sendo realizados considerando o contexto de teste baseado em MEFs. Ambos os trabalhos buscam propor novos mecanismos para a geração de casos de teste a partir de MEFs utilizando as condições de suficiência descritas em Simão e Petrenko (2008). O primeiro trabalho, que já possui uma publicação (Ribeiro et al., 2009), propõe um algoritmo que busca gerar sequências de verificação menores do que as dos métodos identificados na literatura, inclusive os resultados dessa publicação mostram que o tamanho médio das sequências geradas por esse novo algoritmo é menor que o método CS utilizado neste trabalho de mestrado. O segundo trabalho investiga definir um método de geração que produza sequências de teste independentes de serem sequências de verificação, que como visto exige que a MEF possua uma sequência DS. Métodos já definidos na literatura não investigados neste trabalho e novos métodos a serem definidos no futuro poderão ser integrados ao protótipo desenvolvido.

Outro trabalho (Gondim, 2009) visa ao desenvolvimento de um registro de serviços de ferramenta de teste. O protótipo desenvolvido é uma das ferramentas que será integrada nesse registro, além da integração ao projeto *Qualipso*.

Referências Bibliográficas

AHO, A. V.; DAHBURA, A. T.; LEE, D.; UYAR, M. U. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. In: *Proceedings of the 8th Symposium on Protocol Specification, Testing, and Verification, IFIP*, 1988, p. 75–86.

ALBERTO, A. D. B.; SIMÃO, A. S. Minimization of incompletely specified finite state machines based on distinction graphs. In: *Latin-American TestWorkshop*, 2009.

APFELBAUM, L.; DOYLE, J. Model based testing. In: *Software Quality Week Conference*, 1997.

BASILI, W. R. The role of experimentation in software engineering: Past, present and future. In: *18th International Conference on Software Engineering (ICSE)*, Berlin, Germany, 1996, p. 442–449.

BEIZER, B. *Software testing techniques*. 2 ed. New York: Van Nostrand Eeinhold, 1990.

BIANCHI, A.; CAIVANO, D.; MARENGO, V.; VISAGGIO, G. Iterative reengineering of legacy systems. *IEEE Transactions on Software Engineering*, v. 29, n. 3, p. 225–241, 2003.

BINDER, R. V. *Testing object-oriented systems: Models, patterns, and tools*, v. 1. Addison Wesley Longman, Inc., 1999.

BOURHFIR, C.; DSSOULI, R.; ABOULHAMID, E. M.; RICO, N. A test case generation tool for conformance testing of sdl specifications. In: *Proceedings of the 9th SDL Forum*, 1999, p. 405–419.

BRUIJN, N. G. A combinatorial problem. In: *Proc. Koninklijke Nederlandse Akademie van Wetenschappen (A) 9̃, Part 2*, 1946, p. 758–764.

CANDOLO, M. A.; SIMÃO, A. S.; MALDONADO, J. C. Mgaset - uma ferramenta para apoiar o teste e validação de especificações baseadas em máquinas de estado finito. In: *Anais do XV Simpósio Brasileiro de Engenharia de Software*, 2001, p. 386–391.

CHOW, T. S. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, v. 4, n. 3, p. 178–187, 1978.

CUTIGI, J. F.; SIMÃO, A. S. *Módulo para geração de casos de teste a partir de máquinas de estados finitos parciais*. Relatório Técnico 286, ICMC/USP, São Carlos, SP, 2006.

DELAMARO, M. E.; MALDONADO, J. C.; MATHUR, A. P. Interface mutation. an approach for integration testing. *IEEE Transactions on Software Engineering*, v. 27, n. 3, p. 228–247, 2001.

DEMILLO, R. A. Mutation analysis as a tool for software quality assurance. In: *COMPSAC80*, Chicago, IL, 1980.

DOROFEEVA, R.; EL-FAKIH, K.; MAAG, S.; CAVALLI, A. R.; YEVTUSHENKO, N. Experimental evaluation of fsm-based testing methods. In: *Third IEEE International Conference on Software Engineering and Formal Methods*, 2005a, p. 23–32.

DOROFEEVA, R.; EL-FAKIH, K.; YEVTUSHENKO, N. An improved conformance testing method. In: *FORTE*, 2005b, p. 204–218.

FABBRI, S.; MALDONADO, J.; MASIERO, P.; DELAMARO, M. Mutation analysis applied to validate specifications based on petri nets. In: *8th IFIP Conference o Formal Descriptions Techniques for Distributed Systems and Communication Protocol (FORTE'95)*, Montreal, Canadá, 1995, p. 329–337.

FABBRI, S.; MALDONADO, J.; SUGETA, T.; MASIERO, P. Mutation testing applied to validate specifications based on statecharts. In: *International Symposium on Software Reliability Engineering (ISSRE'99)*, 1999a.

FABBRI, S. C. P. F.; MALDONADO, J. C.; MASIERO, P. C.; DELAMARO, M. E. Mutation analysis testing for finite state machines. In: *Fifth International Symposium on Software Reliability Engineering*, Monterey, California, USA, 1994, p. 220–229.

FABBRI, S. C. P. F.; MALDONADO, J. C.; MASIERO, P. C.; DELAMARO, M. E. Proteum/fsm: A tool to support finite state machine validation based on mutation testing. In: *SCCC'99: Proceedings of the 19th International Conference of the Chilean Computer Science Society*, Washington, DC, USA: IEEE Computer Society, 1999b, p. 96.

FERREIRA, E.; S., D. G. V.; VIJAYKUMAR, N. L. Evaluation of test criteria for space application software modeling in statecharts. In: *International Conference on Innovation in Software Engineering (ISE 2008)*, Viena, Austria, 2008, p. 157–162.

FUJIWARA, S.; BOCHMAN, G. V.; KHENDEK, F.; AMALOU, M.; GHEDAMSI, A. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, v. 17, n. 6, p. 591–603, 1991.

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

GÖNNENC, G. A method for the design of fault detection experiments. *ieeec*, v. 19, p. 551–558, 1970.

GONDIM, R. P. Desenvolvimento e avaliação de um registro de serviços de ferramentas de teste. Monografia de Qualificação, 2009.

HAREL, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. 3, p. 231–274, 1987.

HENNIE, F. C. Fault-detecting experiments for sequential circuits. In: *Proceedings of Fifth Annual Symposium on Circuit Theory and Logical Design*, 1965, p. 95–110.

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

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

IEEE *IEEE standard glossary of Software Engineering terminology*. Padrão 620.12, IEEE, 1990.

IEEE *IEEE standard glossary of Software Engineering terminology*. Padrão 610.12-1990, IEEE, 1999.

JAVA Developer resources for java technology. Online em: `http://www.java.sun.com`, último acesso: 25/11/2009, 1994.

LUO, G.; PETRENKO, A.; BOCHMANN, G. *Selecting test sequences for partially-specified nondeterministic finite state machines*. Relatório Técnico, Department d'IRO, Université de Montreál, 1994.

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

MALDONADO, J. C.; BARBOSA, E. F.; VINCENZI, A. M. R.; DELAMARO, M. E.; SOUZA, S. R. S.; JINO, M. *Introdução ao teste de software*. Relatório Técnico 65, ICMC/USP, São Carlos, SP, notas Didáticas do ICMC, Série Computação, 2004.

MALDONADO, J. C.; DELAMARO, M. E.; FABBRI, S. C. P. F.; SIMÃO, A. S.; SUGETA, T.; VINCENZI, A. M. R.; MASIERO, P. C. Proteum: A family of tools to support specification and program testing based on mutation. In: *Mutation 2000*, San Jose, California, 2000, p. 146–149.

MALOKU, N.; FREY-PUCKO, M. Sdl-based feasible test generation for communication protocols. In: *Proceedings of International Conference on Trends in Communication*, Bratislava, Slovak Republic, 2001.

MARTINS, E.; SABIÃO, S. B.; AMBROSIO, A. M. Condata: a tool for automating specification-based test case generation for communication systems. *Software Quality Journal*, v. 8, n. 4, p. 303–319, 1999.

MASIERO, P.; MALDONADO, J.; BOAVENTURA, I. A reachability tree for statecharts and analysis of some propertie. *Information and Software Technology*, v. 36 (10), p. 615–624, 1994.

MEALY, G. H. A method for synthesizing circuits. *Ell System Technical Journal*, v. 34, p. 1045–1079, 1955.

MOORE, E. F. Gedanken experiments on sequential machines. In: *Automata Studies, Annals of Mathematial Studies*, Princeton University Press, Princeton, N.J., 1956, p. 129–153.

MORELL, L. J. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, v. 16, n. 8, p. 844–857, 1990.

MYERS, G. J. *The art of software testing*. 2 ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004.

NAITO, S.; TSUNOYAMA, M. Fault detection for sequential machines by transition tours. In: *Proceedings of the 11th IEEE Fault Tolerant Computing Conference (FTCS 1981)*, IEEE Computer Society Press, 1981, p. 238–243.

NAKAZATO, K. K.; MALDONADO, J. C.; FABBRI, S. C. P. F.; MASIERO, P. C. *Propriedades de máquinas de estado finito relevantes para critérios de geração de sequências de teste*. Relatório Técnico, ICMC/USP, São Carlos, SP, relatorios Tecnicos do Icmsc-Usp, 27, 1994.

PERALTA, K. P. *Estratégia para especificação e geração de casos de teste a partir de modelos uml*. Dissertação de Mestrado, PUCRS, Porto Alegre, RS, 2009.

PETERSON, J. L. Petri nets. *ACM Computing Surveys*, v. 9, n. 3, p. 223–252, 1977.

PETRENKO, A.; BOCHMANN, G.; YAO, M. Y. On fault coverage of tests for finite state specifications. In: *Computer Networks and ISDN Systems*, 1996, p. 81–106.

PETRENKO, A.; PETRENKO, R.; GROZ, R.; BORODAY, S. Confirming configurations in efsm testing. *IEEE Transactions on Software Engineering*, v. 30, p. 2004, 2004.

PETRENKO, A.; YEVTUSHENKO, N. Testing from partial deterministic fsm specifications. *IEEE Transactions on Computers*, v. 54, n. 9, 2005.

PETRENKO, A.; YEVTUSHENKO, N.; LEBEDEV, A.; A.DAS Nondeterministic state machines in protocol conformance testing. In: *Protocol Test Systems*, 1993, p. 363–378.

PIMONT, S.; RAULT, J. C. A software reliability assessment based on a structural and behavioral analysis of programs. In: *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, p. 486–491.

PLAVIS Plavis - platform for software validation & integration on space systems. Online em: `http://www.labes.icmc.usp.br/plavis/index.html`, último acesso: 25/11/2009, 2005.

PRESSMAN, R. S. *Software engineering — a practitioner's approach*. 6 ed. McGraw-Hill, 2006.

QUALIPSO Qualipso - quality platform for open source software. Online em: `http://www.qualipso.org`, último acesso: 25/11/2009, 2004.

RAPPS, S.; WEYUKER, E. J. Data flow analysis techniques for program test data selection. In: *6th International Conference on Software Engineering*, Tokio, Japan, 1982, p. 272–278.

RIBEIRO, P. H.; CUTIGI, J. F.; SIMÃO, A. S. Geração de sequências de verificação baseado em algoritmos genéticos. In: *SAST - Workshop Brasileiro de Teste de Software Sistemático e Automatizado*, Gramado, RS, 2009.

SABNANI, K. K.; DAHBURA, A. A protocol test generation procedure. *Computer NetWorks and ISDN Systems*, v. 15, n. 4, p. 285–297, 1988.

SANTIAGO, V.; AMARAL, A.; VIJAYKUMAR, N.; MATTIELLO-FRANCISCO, M.F. AMD MARTINS, E.; LOPES, O. A practical approach for automated test case generation using statecharts. In: *2nd International Workshop on Testing and Quality Assurance for Component-Based Systems, IEEE COMPSAC Conference*, Chicago, EUA, 2006, p. 183–188.

SANTIAGO, V.; VIJAYKUMAR, N. L.; GUIMARAES, D.; AMARAL, A. S.; FERREIRA, E. An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. *Software Testing Verification and Validation Workshop, IEEE International Conference on*, v. 0, p. 63–72, 2008.

SHEN, X.; LI, G. A new protocol conformance test generation method and experimental results. In: *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, New York, NY, USA: ACM, 1992, p. 75–84.

SHEN, X.; SCOGGINS, S.; TANG, A. An improved rcp-method for protocol test generation using backward uio sequences. In: *Proceedings of Symposium on Applied Computing*, ACM, 1991, p. 284–293.

SHEN, Y. N.; LOMBARDI, F.; DAHBURA, A. T. Protocol conformance testing using multiple uio sequences. In: *Proceedings of the IFIP WG6.1 Ninth International Symposium on Protocol Specification, Testing and Verification IX*, Amsterdam, The Netherlands: North-Holland Publishing Co., 1989, p. 131–143.

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

SIMÃO, A.; PETRENKO, A. Generating checking sequences for partial reduced finite state machines. In: *TestCom '08 / FATES '08: Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems*, Berlin, Heidelberg: Springer-Verlag, 2008, p. 153–168.

SIMÃO, 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.

SIMÃO, A. S.; AMBRÓSIO, A. M.; FABBRI, S. C. P. F.; AMARAL, A. S. M. S.; MARTINS, E.; MALDONADO, J. C. Plavis/fsm: An integrated platform for validating fsm based system. In: *Latin-American Symposium on Dependable Computing*, 2005.

SIMÃO, A. S.; PETRENKO, A. *Checking completeness of tests for finite state machines*. Relatório Técnico, CRIM - Centre de Recherche Informatique de Montreal, 2007.

SIMÃO, A. S.; PETRENKO, A.; MALDONADO, J. C. Comparing finite state machine test coverage criteria. *IET Software*, p. 91–105, 2009.

SOUZA, S. *Validação de especificações de sistemas reativos: Definição e análise de critérios de teste*. Tese de Doutoramento, ICMC/USP, São Carlos, SP, 2000.

SOUZA, S.; MALDONADO, J. C.; FABBRI, S. C. P. F. Fcce: uma família de critérios de teste para validação de sistemas especificados em estelle. In: *Anais do XV Simpósio Brasileiro de Engenharia de Software*, Rio de Janeiro, RJ, 2001.

STUCHI, J. A.; SIMÃO, A. S. *Investigação de mecanismos para redução do conjunto de teste de máquinas de estados finitos*. Relatório Técnico, ICMC/USP, São Carlos, SP, 2008.

SUGETA, T.; MALDONADO, J. C.; MASIERO, P. C.; FABBRI, S. C. P. F. Proteum-rs/st - a tool for support statecharts validation based on mutation testing. In: *4th Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software - IDEAS'2001*, Santo Domingo, Costa Rica, 2001, p. 370–384.

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

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

VUONG, S. T.; CHAN, W. W. L.; ITO, M. R. The uiov-method for protocol test sequence generation. In: *Proc. of the IFIP TC6 2nd IWPTS*, North-Holland, 1989, p. 161–175.

WOHLIN, C.; HUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000.

WONG, W. E.; MATHUR, A. P.; MALDONADO, J. C. Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness. In: *Software Quality and Productivity: Theory, practice and training*, London, UK, UK: Chapman & Hall, Ltd., 1995, p. 258–265.

WONG, W. E.; RESTREPO, A.; QI, Y.; CHOI, B. An efsm-based test generation for validation of sdl specifications. In: *AST '08: Proceedings of the 3rd international workshop on Automation of software test*, New York, NY, USA: ACM, 2008, p. 25–32.

YANG, B.; URAL, H. Protocol conformance test generation using multiple uio sequences with overlapping. *SIGCOMM Comput. Commun. Rev.*, v. 20, n. 4, p. 118–125, 1990.