Uma estratégia para a minimização de máquinas
de estados finitos parciais

Alex Donizeti Betez Alberto

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Uma estratégia para a minimização de máquinas de estados finitos parciais

Alex Donizeti Betez Alberto

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

USP - São Carlos
Junho/2009

MÁQUINAS de Estados Finitos, além de suas inúmeras aplicações, são amplamente utilizadas na Engenharia de Software para modelar especificações de sistemas. Nesses modelos, projetistas podem inserir, inadvertidamente, estados redundantes, ou seja, que exibem o mesmo comportamento. A eliminação desses estados traz diversos benefícios para as atividades que utilizam o modelo, como menor complexidade e menos recursos físicos para implementação. O processo de eliminação desses estados é denominado minimização, e pode ser realizado em tempo polinomial para máquinas completamente especificadas. Por outro lado, a minimização de máquinas parciais, cuja especificação não cobre todo o domínio de entrada, somente pode ser obtida em tempo polinomial com o uso de abordagens não determinísticas, ou seja, trata-se de um problema NP-Completo. Este trabalho apresenta uma estratégia para a minimização de máquinas de estados finitos parciais que faz o uso de heurísticas e otimizações para tornar o processo mais eficiente. Visando mensurar tal ganho de eficiência, foram realizados experimentos, nos quais os tempos de execução de uma implementação do método proposto foram medidos, juntamente com os tempos de implementações de dois outros métodos conhecidos. Os resultados mostraram vantagens significativas de performance para o novo método em relação aos métodos anteriores.

Abstract

FINITE State Machines are largely used on Software Engineering to model systems specifications. In these models, designers may inadvertently include redundant states, i.e., states which exhibit the same input/output behavior. The absence of such states brings benefits to the modeling activities, reducing the complexity and taking less physical resources on implementations. The process of eliminating redundant states is known as minimization, and can be accomplished in polynomial time for completely specified machines. On the other hand, the minimization of partially specified machines, i.e., machines which have undefined behavior for some inputs, can only be done in polynomial time when non-deterministic approaches are applied. It is a known NP-Complete problem. This work presents a deterministic approach to minimize incompletely specified Finite State Machines, using heuristics and optimizations to accomplish the task more efficiently. In order to measure the performance improvements, experiments were done, observing the running time of an implementation of the proposed method, along with running times of implementations of two other known methods. The results revealed a significant performance advantage when using the proposed approach.

Sumário

Resumo	i
Abstract	iii
1 Introdução	1
1.1 Contextualização	1
1.2 Motivação	2
1.3 Objetivos e Contribuições do Trabalho	3
1.4 Organização	4
2 Teste de Software	7
2.1 Considerações Iniciais	7
2.2 Teste e Qualidade de Software	7
2.3 Conceitos e Definições	8
2.4 Critérios de Teste	10
2.4.1 Teste Baseado em Especificação	11
2.5 Modelos Formais em Engenharia de Software	11
2.5.1 Máquinas de Estados Finitos	14
2.6 Métodos de Geração de Casos de Teste	17
2.6.1 Conceitos Envolvidos	17
2.6.1.1 <i>State Cover</i>	18
2.6.1.2 <i>Transition Cover</i>	19

2.6.2	Método W	19
2.7	Análise da Cobertura de Casos de Teste	21
2.7.1	Verificação da Completude Através da Minimização de MEF	22
2.8	Considerações Finais	24
3	Minimização de Máquinas de Estados Finitos	27
3.1	Considerações Iniciais	27
3.2	Minimização de MEFs Parciais	29
3.2.1	Método <i>bica</i>	30
3.2.2	Método <i>chesmin</i>	32
3.3	Considerações Finais	34
4	Método Cosme	35
4.1	Considerações Iniciais	35
4.2	Construção do Grafo de Distinção	36
4.3	Seleção de Estados Compatíveis	37
4.4	Fusão de Estados Compatíveis	41
4.5	Procedimento para Minimização	45
4.6	Aplicação do Método	50
4.6.1	Modo Exato	50
4.6.2	Modo Aproximado	55
4.7	Considerações Finais	60
5	Experimentos	61
5.1	Considerações Iniciais	61
5.2	<i>Comparação Empírica</i>	62
5.3	Avaliação de Casos de Teste	64
5.4	Minimização de MEFs Aleatórias	66
5.4.1	Número de Estados	67
5.4.2	Número de Transições	68
5.4.3	Número de Entradas	69

5.4.4	Número de Saídas	69
5.5	Comparação do Poder de Minimização	70
5.6	Considerações Finais	71
6	Conclusões	75
6.1	Contribuições	76
6.2	Limitações	76
6.3	Trabalhos Futuros	76
	Referências	83

Lista de Figuras

2.1	Rede de Petri (Peterson, 1977)	13
2.2	Exemplo de Diagrama <i>Statechart</i> (Harel, 1987)	14
2.3	Representação por diagrama de transição de uma MEF (Fujiwara et al., 1991)	15
2.4	MEF exibindo erro de transferência para o conjunto P	19
2.5	Uma MEF simples para utilização em exemplos	20
2.6	MEF em árvore (TFSM)	23
2.7	MEF reconstruída reduzida e MEF da especificação	24
3.1	Redução de uma MEF	28
3.2	Visão geral do método <i>chesmin</i> (Gören e Ferguson, 2002)	34
4.1	Uma MEF e seu respectivo grafo de distinção	36
4.2	A MEF M' , após a primeira iteração, e seu respectivo grafo de distinção	49
4.3	MEF resultante da segunda iteração e seu respectivo grafo de distinção	50
4.4	MEF mínima para geração de seqüências de teste	51
4.5	MEF em árvore construída sobre as seqüências do método HSI	52
4.6	Grafo de distinção construído sobre a MEF em árvore	52
4.7	MEF M' , resultante da fusão de q_1 e q_6	53
4.8	MEF M'' , resultante da fusão de q'_2 e q'_3	54
4.9	MEF M''' , resultante da fusão de q''_2 e q''_9	54
4.10	Grafo de distinção da MEF resultante M'''	54

4.11	MEF M para redução	55
4.12	Grafo de distinção G para M na primeira iteração	55
4.13	A MEF M' resultante da primeira iteração no modo aproximado	57
4.14	Grafo de distinção G' para M' na segunda iteração	57
4.15	A MEF M'' resultante da segunda iteração	58
4.16	Grafo de distinção G'' para M'' na terceira iteração	58
4.17	A MEF M''' resultante da terceira iteração	58
4.18	Grafo de distinção G''' para M''' na terceira iteração	59
4.19	A MEF M'''' resultante da quarta iteração	59
5.1	Comparação de desempenho entre os métodos na redução de MEFs em árvore	65
5.2	Porcentagem de falhas exibidas pelo stamina para cada valor de n	66
5.3	Desempenho dos métodos na redução de MEFs aleatórias em função de n .	67
5.4	Falhas exibidas pelo bica na redução de MEFs aleatórias em função de n .	68
5.5	Falhas exibidas pelo stamina na redução de MEFs aleatórias em função de n	69
5.6	Desempenho dos métodos na redução de MEFs aleatórias em função de t .	70
5.7	Desempenho do cosme na redução de MEFs aleatórias em função de t . . .	71
5.8	Desempenho dos métodos na redução de MEFs aleatórias em função de i .	72
5.9	Desempenho do cosme na redução de MEFs aleatórias em função de i . . .	73
5.10	Desempenho dos métodos na redução de MEFs aleatórias em função de o .	73
5.11	Comparação do poder de minimização entre os métodos cosme e bica . . .	74

Lista de Tabelas

2.1	Representação da MEF em tabela de transição	15
2.2	Seqüências de Entrada/Saída geradas pelo método W	22
4.1	Iterações do Algoritmo 3 para o grafo de distinção da Figura 4.1(b)	40
4.2	Iterações do Algoritmo 4 para a MEF da Figura 4.1(a)	43
4.3	Seqüências de Entrada/Saída geradas pelo método HSI para a MEF da Figura 4.4	51
4.4	Seqüências de entradas para distinção dos pares de estados da MEF da Figura 4.5	51
4.5	Seqüências de entradas para distinção dos pares de estados da MEF da Figura 4.11	56
5.1	Desempenho observado	63

Introdução

1.1 Contextualização

A evolução dos recursos computacionais, tal como ocorreu nas décadas recentes, aumentou o potencial de utilização dos computadores como ferramentas para um número maior de atividades. Os sistemas tornaram-se progressivamente mais complexos, acompanhando as necessidades das novas responsabilidades que passaram a lhes serem atribuídas. Logo os computadores haviam deixado de ser apenas ferramentas, que simplesmente aumentavam a eficiência das tarefas, para realizar atividades as quais sem os mesmos seriam inviáveis. Dessa forma, sistemas computacionais tornam-se indispensáveis para a sociedade.

Como parte desse processo, a computação é hoje empregada em tarefas que envolvem questões críticas, onde falhas podem implicar em circunstâncias indesejáveis e, em alguns casos, irreversíveis, como a perda de vidas humanas ou prejuízos materiais. Organizações como hospitais e bolsas de valores, por exemplo, apóiam grande parte de sua estrutura operacional sobre sistemas computacionais complexos e de alta demanda. Esses sistemas devem ser confiáveis; portanto é importante buscar garantias de que se comportarão adequadamente nas possíveis situações às quais estarão expostos.

Para que essa confiança seja estabelecida, é necessário ponderar a respeito da qualidade dos sistemas construídos. Nesse sentido, a Engenharia de Software provê recursos, como métodos e ferramentas, que visam tornar mais consistente o complexo processo de elaboração que é exigido por esses produtos. Dentre as fases que compõe um processo usual de desenvolvimento, uma possui elevada importância em viabilizar o julgamento de

quão correto é um sistema: o teste de software. As atividades de teste em um projeto de software consomem grande parte do esforço total de desenvolvimento e, em se tratando de sistemas nos quais o mau funcionamento pode ocasionar conseqüências severas, a máxima atenção para sua verificação é justificada (Adrion et al., 1982).

Uma das formas de se testar software consiste em confrontar o comportamento exibido pelo produto final contra aquele esperado, previamente especificado. Essa abordagem é conhecida como teste de conformidade, ou *conformance test* (Bochmann e Petrenko, 1994). Apesar da simplicidade do conceito, sua aplicação em situações práticas esbarra em complicações. Sistemas suficientemente simples podem ser verificados de maneira manual; já sistemas com maior complexidade inviabilizam esse tipo de solução. Há então a necessidade de se automatizar o processo e, para tal, é preciso que o comportamento desejado seja especificado por meio de modelos passíveis de processamento.

Descrever o comportamento de sistemas em geral utilizando modelos formais não é trivial, no entanto, sistemas que possuem determinadas características e recebem a classificação de sistemas reativos (Pressman, 2002), são adequados para explorar os benefícios do uso de tais modelos em sua especificação. Sistemas reativos têm como característica interagir constantemente com seus contextos por intermédio de eventos sequenciais de entrada e saída. Sistemas desse tipo são usualmente encontrados no controle de operações críticas, como gerenciamento de tráfego e o monitoramento de ambientes industriais.

A forma como os sistemas reativos interagem com os contextos nos quais estão inseridos favorece a utilização de um tipo particular de modelo formal em sua especificação: são as Máquinas de Estados Finitos (MEFs). Modelar o comportamento de um sistema utilizando uma MEF viabiliza a automação do processo de teste de conformidade. Há décadas (Chan et al., 1989; Chow, 1978; Fujiwara et al., 1991; Moore, 1956; Petrenko e v. Bochmann, 1995a) métodos são propostos com o objetivo de selecionar seqüências de entradas, a partir de especificações modeladas por MEFs, adequadas para o teste de implementações. Essas seqüências são utilizadas para confrontar as saídas obtidas do produto final com aquelas que estão definidas no modelo. As entradas são aplicadas na implementação em teste e os erros são detectados pela produção de seqüências de saídas incompatíveis. Tais métodos, conhecidos como métodos de geração, buscam selecionar o subconjunto de seqüências de entradas de maior relevância, ou seja, maximizando o poder de detecção de falhas e minimizando o número e o tamanho das seqüências que devem ser aplicadas.

1.2 Motivação

O uso de MEFs para modelar especificações e derivar testes de sistemas pode ser beneficiado por um pré-processamento, cujo objetivo é eliminar estados redundantes desses

modelos. O processo é conhecido como minimização. Uma MEF em sua forma mínima não possui estados redundantes, ou seja, não há como se construir uma nova máquina, que represente o mesmo comportamento, utilizando um número menor de estados. Nessas circunstâncias, tem-se o modelo de menor complexidade, que exige a menor quantidade de recursos físicos para implementação.

Além dos benefícios relacionados à diminuição da complexidade do modelo, a maioria dos métodos de geração de casos de teste baseados em MEFs, dentre os quais se pode citar o W (Chow, 1978), o Wp (Fujiwara et al., 1991) e o HSI (Petrenko et al., 1993), **exige** que as máquinas de especificação submetidas para geração sejam apresentadas em sua forma mínima.

A redução de estados em MEFs completamente especificadas, ou seja, aquelas que possuem um comportamento definido para todas as possíveis seqüências de entrada, pode ser realizada em tempo polinomial (Pena e Oliveira, 1998). Por outro lado, minimizar MEFs que não contam com tal característica (ditas MEFs parciais) pode requerer custo computacional de ordem exponencial em relação ao número de estados, sendo classificado como um problema NP-Completo. Em situações práticas é comum encontrar modelos de MEFs parciais que possuem um número elevado de estados.

Em outro contexto, Yao et al. (1994) mostrou que, utilizando um procedimento de minimização para MEFs parciais, é possível decidir quando um conjunto de seqüências de testes para MEFs provê cobertura completa de falhas. Conjuntos que contam com tal poder de detecção são de especial interesse, pois fica estabelecida a garantia de que uma implementação testada nessas condições comportar-se-á sempre em conformidade com o que foi especificado.

Considerando aplicações como as citadas, é importante que se disponha de maneiras para realizar a redução de estados em MEFs parciais com eficiência. Tratando-se de um problema NP-Completo, propostas para solucioná-lo fazem uso de heurísticas e otimizações com o intuito de reduzir a complexidade do processo. Apesar de ser um problema investigado há muito tempo (Grasselli e Luccio, 1965), a existência de publicações recentes (Gören e Ferguson, 2007; Pena e Oliveira, 1998) evidenciam que ainda há interesse da comunidade científica por métodos mais eficientes.

1.3 Objetivos e Contribuições do Trabalho

Este trabalho objetivou a elaboração de uma abordagem capaz de gerar soluções para o problema NP-Completo da minimização de MEFs parciais. O método que é apresentado foi denominado *cosme*, em referência aos termos, em inglês, *Compatible State Merging*.

A técnica desenvolvida explora informações inferidas a partir de cliques maximais em grafos que relacionam estados distinguíveis, ou seja, estados que podem ser diferencia-

dos por gerar seqüências de saídas distintas a partir da aplicação de, pelo menos, uma seqüência de entrada válida. Tais informações são a base da heurística apresentada para a seleção de estados compatíveis, os quais podem ser combinados em um único estado de forma a não implicar em alterações no comportamento externado pela MEF.

Como é esperado em métodos heurísticos, as soluções geradas não são garantidamente exatas, isto é, a MEF resultante de um processo de minimização por meio do método *cosme* pode não ser o modelo com menor número de estados capaz de representar o comportamento original. No entanto, em experimentos empíricos realizados, soluções exatas foram obtidas para a maioria das amostras e, nas demais instâncias, a diferença no número final de estados não é proporcionalmente grande.

Os experimentos empíricos também foram utilizados na medição do desempenho exibido pela implementação da abordagem apresentada em relação às implementações de dois outros métodos que compartilham o mesmo propósito. As amostras utilizadas são provenientes de procedimentos de geração aleatória e também de um conjunto de teste selecionado por alguns autores de trabalhos anteriores neste tópico. A comparação dos resultados permite concluir que o método *cosme* produz resultados com menor tempo de execução que os métodos com os quais foi comparado, mostrando-se como uma abordagem viável para a minimização de MEFs parciais.

1.4 Organização

No Capítulo 2, são apresentados conceitos e técnicas a respeito da atividade de teste de software. Com ênfase nas abordagens de teste baseadas em modelos, a utilização de modelos formais na descrição de especificações de sistemas é discutida. O capítulo levanta a relação entre a utilização de MEFs na Engenharia de Software e os benefícios que esta obtém a partir do aprimoramento das técnicas de minimização.

O Capítulo 3 é direcionado ao problema da minimização de MEFs parciais. Os conceitos envolvidos são introduzidos e as características que implicam em dificuldades para solucioná-lo são discutidas. Além da solução clássica para o problema, também são apresentadas algumas das técnicas de minimização que exploram heurísticas e demais otimizações, encontradas na literatura.

A contribuição deste trabalho, uma abordagem para a minimização das MEFs parciais, é apresentada no Capítulo 4. Nele estão descritos todos aspectos considerados na elaboração do método, além de algoritmos e exemplos de aplicações.

No Capítulo 5 estão relatados os experimentos que validam a abordagem proposta. Foram realizadas comparações entre uma implementação do método proposto com implementações de outros métodos de mesmo propósito. A metodologia e os resultados, favoráveis na maioria dos cenários, são detalhadamente apresentados.

Por fim, conclusões, contribuições e trabalhos futuros são discutidos no Capítulo 6.

Teste de Software

2.1 Considerações Iniciais

Conceitos e informações relevantes acerca da teoria e de práticas em teste de software foram reunidas no presente capítulo. Considerações e discussões sobre o papel da atividade de teste em processos de desenvolvimento são apresentadas na Seção 2.2. Em seguida, na Seção 2.3, as definições de termos e conceitos necessários para o restante da discussão sobre o tema são apresentados. Tipos de critérios e técnicas de teste são discutidos na Seção 2.4. A Seção 2.5 conta com uma análise acerca dos benefícios do uso de modelos formais na Engenharia de Software. Por fim, a Seção 2.6, seguida pela Seção 2.7, contém, respectivamente, informações sobre técnicas de geração e avaliação de casos teste baseadas em MEFs.

2.2 Teste e Qualidade de Software

Dijkstra (1972) declarou que a prática de testar software pode apenas apontar a presença de defeitos, porém jamais garantir que eles não existam. Essa observação é incontestável, tendo em vista as várias limitações às quais está sujeita a atividade de teste de software. Algumas questões que formam a base para essa atividade, quando analisadas da perspectiva da teoria da computação, são classificadas como problemas indecidíveis, incluindo: determinar se dois programas são equivalentes; determinar se um determinado trecho de um programa é executável; determinar se duas sequências de comandos computam uma

mesma função (Maldonado et al., 2004). Uma maneira de se garantir a corretude de um programa é, em vez de se aplicar testes, apresentar-se uma prova formal de tal comportamento (Naur e Randell, 1969). No entanto, essa abordagem não é viável para situações reais por uma série de razões práticas. Dentre elas, pode-se citar a exigência de que todo o ambiente no qual o software estaria envolvido esteja disponível em forma axiomática (Goodenough e Gerhart, 1975).

Sendo então impossível garantir que um software não possua defeitos, buscaram-se, nas últimas décadas, maneiras de reduzir ao mínimo sua incidência em produtos dessa natureza. Atividades para Garantia de Qualidade de Software (GQS) conduzem os processos de desenvolvimento para práticas mais sistemáticas e consistentes, gerando produtos com melhor qualidade. Entre as práticas que pretendem aprimorar os produtos de software estão as atividades de Verificação, Validação e Teste (VV&T). Verificação e validação são atividades que visam, respectivamente, garantir que as funções de um software estejam implementadas de maneira correta e que estas funções desempenham seu papel de acordo com os requisitos (Pressman, 2005). É a atividade de teste que objetiva a busca por defeitos não previamente detectados, o que normalmente é realizado por meio da execução de, ao menos, parte do software (Myers, 2004), ou seja, é necessário que algum tipo de implementação executável esteja disponível.

2.3 Conceitos e Definições

Com a intenção de evitar enganos de interpretação e aprimorar a troca de informações e o compartilhamento do conhecimento entre os envolvidos no contexto de teste de software, a IEEE estabeleceu o padrão de número 610.12-1990 (IEEE, 1999), um glossário da terminologia da Engenharia de Software, no qual os seguintes conceitos são diferenciados:

fault: defeito, em português. Faz referência a elementos incorretos que possam estar presentes em um software, como passos, processos ou definições de dados.

mistake: engano, em português. Refere-se a uma ação ou decisão de um indivíduo envolvido no processo de desenvolvimento que tenha levado à inserção de um defeito.

error: erro, em português. Refere-se à diferença entre um valor produzido por uma implementação e aquele que seria esperado, segundo a especificação.

failure: falha, em português. Diz respeito à exibição de um erro, normalmente provocada pela manifestação de um defeito.

Com a aplicação de testes, busca-se revelar o maior número possível de falhas, ocasionadas por erros, oriundos de defeitos, potencialmente inseridos por enganos. Ainda nessa

linha de raciocínio: falhas e defeitos são causados, respectivamente, por erros e enganos, portanto, pode-se separar os conceitos em causas e efeitos. Para o presente texto, foram selecionados os termos **erro** e **falha** para fazer referência, respectivamente, às razões e as consequências de problemas em softwares.

É desejável que atividades de teste sejam realizadas nos vários níveis do processo de elaboração de um software. A partir do momento em que partes do produto estão disponíveis na forma executável, técnicas já podem ser utilizadas para descobrir erros. Segundo Pressman (2002), as atividades de teste em diferentes níveis do processo de desenvolvimento são classificadas da seguinte maneira:

Teste de Unidade: por unidade, entende-se o menor fragmento executável de um software. A definição de unidade é dependente do contexto. Por exemplo, no desenvolvimento orientado a objetos, uma unidade pode ser um método, uma classe, um componente ou um outro tipo qualquer de módulo. O teste individual de unidades auxilia na detecção de falhas na lógica implementada, possibilitando realizar as devidas correções ainda em fases iniciais do processo de desenvolvimento.

Teste de Integração: avaliação das interfaces das unidades. Assumindo que o funcionamento individual das unidades já foi verificado pelas atividades prévias, são realizados testes para verificar como se dá o seu comportamento em grupos. As funcionalidades obtidas pela colaboração entre duas ou mais unidades são, nesse nível, avaliadas.

Teste de Sistema: nas últimas etapas do processo de desenvolvimento, o teste de sistema visa avaliar de uma maneira mais abrangente o produto construído. Todos os elementos que compõe o ambiente no qual estará inserido o sistema devem ser considerados, incluindo a plataforma de execução e outros sistemas externos que, de alguma forma, estão relacionados ao sistema em teste.

Uma especificação S é uma descrição, formal ou não, de como um determinado programa P deve produzir saídas quando elementos de um domínio D são a ele apresentados como entradas. O programa P funciona corretamente se a saída produzida para todo elemento de D está de acordo com o que fora descrito em S , ou seja, $\forall x \in D, S(x) = P(x)$. A manifestação de uma falha se dá pela produção de uma saída não prevista em S , indicando que P se encontra em estado de erro. Um **caso de teste** é um par ordenado $(x, S(x)) \mid x \in D$ que consiste em um elemento x , pertencente ao domínio D , juntamente com a saída que, segundo S , deve ser produzida por P ao receber x como entrada.

Para detectar todas as falhas que podem ser apresentadas por um software, pode-se testá-lo utilizando como entrada todo o domínio D , comparando as saídas produzidas com as definidas em S ; mais especificamente, deve-se montar e executar casos de teste para

todos os elementos de D . Na prática, no entanto, o domínio D , quando não é infinito, contém um número muito grande de elementos (Frankl e Weyuker, 2000). Diante da impossibilidade de utilizar todo o domínio, deve-se selecionar um subconjunto $T \subseteq D$, cujo tamanho viabilize a execução dos testes.

O subconjunto T deve ser tão representativo quanto o possível em relação a D , culminando na condição $\forall t \in T, P(t) = S(t) \Rightarrow \forall x \in D, P(x) = S(x)$, ou seja, seria suficiente testar P com os elementos de T para garantir sua corretude. Embora T com tal propriedade sempre exista, não há uma maneira sistemática para encontrá-lo (Budd e Angluin, 1982), pois trata-se de uma questão semelhante a decidir quando dois programas são equivalentes, um dos problemas indecidíveis dentre os citados no início Seção 2.2 como suporte para a observação de Dijkstra sobre a atividade de teste de software.

2.4 Critérios de Teste

Na impossibilidade de se selecionar um número viável de elementos para $T \subseteq D$ de forma a compor casos de teste que revelem tantos erros quanto o próprio conjunto D , foram elaboradas estratégias para selecionar, sistematicamente, um número relativamente pequeno de elementos de D afim de maximizar o potencial de revelação de erros sob algum aspecto. Essas estratégias são conhecidas como **critérios de teste**. Os critérios devem auxiliar na atividade de teste respondendo a duas questões: quais elementos de D devem ser selecionados para compor os casos de teste e o quão bom é um subconjunto T para testar P (confiabilidade) (Maldonado et al., 2004). Apesar de relacionadas, as questões são distintas, e para atendê-las são definidos dois procedimentos (Budd e Angluin, 1982):

Gerador: procedimento $G_C(P, S)$, que partindo de um programa P e sua especificação S , produz um conjunto T adequado para testar P , de acordo com o critério C .

Avaliador: dados P , S e um conjunto T , o procedimento $A_C(P, S, T)$ deve decidir quando T é adequado para testar P , segundo o critério C .

Budd e Angluin (1982) ainda demonstraram que, definido um dos procedimentos para um critério C , é também possível definir o outro. Tal como pode ser observado na definição do procedimento gerador, os subsídios para julgar quais elementos de D têm maior potencial para revelar erros são a especificação S e a implementação P .

Ao explorar cada uma dessas fontes, esses critérios ficam sujeitos às classificações: **teste baseado em especificação** e **teste baseado em programa** (Howden, 1986). Também é comum encontrar os critérios classificados sob esses mesmos aspectos, porém fazendo o uso das nomenclaturas de **teste estrutural** e **teste funcional**, respectivamente relacionadas à teste baseado em especificação e programa. Os métodos de teste

considerados neste trabalho são baseados em especificação e, por essa razão, tal tópico é abordado a seguir com maiores detalhes.

2.4.1 Teste Baseado em Especificação

Nos modelos e documentos que compõem a especificação de um software estão definidas as funções que devem ser desempenhadas. Critérios de teste baseados em especificação fazem uso das informações nela disponíveis para julgar quais são os casos de teste mais relevantes, de forma a verificar se o comportamento que foi definido é de fato exibido pelas implementações. Não existe a preocupação com detalhes sobre a implementação que está sendo testada, sendo considerada como uma caixa de conteúdo desconhecido. Por essa razão, é comum encontrar o termo **teste caixa-preta** (*black box testing*) relacionado à prática de técnicas de teste baseado em especificação (Myers, 1979).

Por desconsiderar detalhes de elaboração das implementações, critérios de teste funcional podem ser aplicados sem preocupações com o contexto do desenvolvimento dos produtos de software em teste. Dessa maneira, um mesmo conjunto de casos de teste selecionados por um critério baseado em especificação é, em teoria, adequado para testar tanto uma implementação construída sob o paradigma procedural quanto uma baseada na orientação a objetos.

Uma limitação desse tipo de teste se dá pela ampla utilização de especificações informais, o que, além de adicionar complicações à automatização do processo de seleção de casos de teste, implica em coberturas de falhas igualmente informais (Perry e Kaiser, 1990) e, de certa forma imprecisas. Esses critérios são normalmente aplicados de maneira manual, e bons resultados são obtidos somente quando as especificações disponíveis são concisas e devidamente consistentes com os requisitos. Por outro lado, o uso de modelos formais para especificar o comportamento de sistemas possibilita a aplicação de métodos automáticos de seleção de casos de teste que apresentam grande eficácia no teste de sistemas reativos. Dada a importância desses métodos para este trabalho, eles serão discutidos com maiores detalhes na seção seguinte.

2.5 Modelos Formais em Engenharia de Software

A utilização de métodos precisos de desenvolvimento assegura um menor número de defeitos presente no resultado final, pois resulta na redução de inconsistências e ambigüidades (Barroca e McDermid, 1992). Por essa razão, o uso de métodos formais nos processos de desenvolvimento de software contribui para a elaboração de produtos com maior qualidade. Especificar um sistema fazendo uso desses modelos já é suficiente para eliminar um grande número de potenciais erros, inerentes ao uso de linguagem natural, devidamente apontada como ambígua (Wing, 1990). Porém, deve-se considerar que o

uso desses métodos aumenta muito a complexidade do processo de desenvolvimento. Em situações práticas, sistemas de informação são elaborados sobre especificações demasiadamente complexas, o que inviabiliza o emprego de modelos formais em suas descrições. Portanto é conveniente considerar este recurso apenas em situações onde o mesmo é de fato necessário.

Segundo Pressman (2005) é possível enumerar as categorias de produtos de software da seguinte maneira: software básico, sistemas de informação, sistemas científicos, sistemas embutidos, sistemas pessoais, sistemas de inteligência artificial e sistemas reativos. Esta última é de particular interesse para o presente estudo: um sistema reativo caracteriza-se por manter uma interação permanente com seu ambiente externo, seja esse um usuário, um dispositivo de entrada ou outro sistema. São exemplos de sistemas reativos: sistemas de telefonia, módulos de sistemas operacionais, redes de comunicação, tecnologias embarcadas em automóveis, mísseis e aviões (Harel e Pnueli, 1985). Dessa forma, sistemas reativos podem ser utilizados em cenários críticos, onde é possível que falhas levem a situações irreversíveis, como a perda de vidas humanas ou o prejuízo material. Segundo Harel (1987), a literatura em Engenharia de Software enfatiza a existência de grandes dificuldades na especificação e projeto de sistemas reativos complexos. A principal questão parte da dificuldade de se descrever o comportamento reativo de forma clara e realista, porém ao mesmo tempo formal e rigorosa, a tal ponto que seja possível submeter essa descrição à simulação por computador.

O comportamento de um sistema reativo consiste, basicamente, em um conjunto de seqüências de entradas permitidas e seus respectivos eventos de saída. A interação deve ocorrer ao longo de intervalos discretos de tempo. Devido a essa natureza, um tipo particular de modelo formal é bastante adequado para a elaboração de especificações para esses sistemas, são as **Máquinas de Transição de Estados**. Uma Máquina de Transição de Estados é uma estrutura que possibilita definir possíveis próximas ações (ou saídas) com base no estado atual e estímulos (ou entradas) externos. Alguns tipos particulares de Máquinas de Transição de Estados são introduzidos a seguir.

Redes de Petri

Trata-se de um modelo abstrato para representar o fluxo de informações (Peterson, 1977). O modelo foi baseado em conceitos de operações assíncronas e concorrentes entre partes de sistemas. Sua representação em grafo conta com dois tipos de nós: círculos e barras, respectivamente chamados de **lugares** e **transições**. Os nós podem ser ligados por arcos direcionados, partindo de lugares e alcançando transições, ou o inverso. Quando existe um arco que parte de um nó i e incide em um nó j , diz-se que i é entrada para j e, conseqüentemente, j é saída para i . Além da representação de características estáticas, o modelo possibilita a simulação de características dinâmicas, representadas pela

movimentação de uma **marca** (do termo em inglês *token*) entre os lugares, através das transições. As marcas são representadas por pontos negros, posicionados no interior dos nós em círculo. O movimento de uma marca acontece quando uma transição é disparada. As transições, por sua vez, podem ser disparadas somente quando estão ativas, isto é, todas as suas entradas possuem uma marca. Quando isto ocorre, marcas são transferidas para a saída da transição que disparou. Um diagrama de uma Rede de Petri é ilustrado na Figura 2.1.

Figura 2.1: Rede de Petri (Peterson, 1977)

Statecharts

Este modelo parte de uma abordagem orientada a estado/evento, com estruturas modulares e hierárquicas. A modelagem de um comportamento complexo pode exigir um número grande de estados e transições, resultando em diagramas mal estruturados, pouco realistas e caóticos (Harel, 1987). Pelo agrupamento de estados em **superestados**, um diagrama *statechart* introduz uma série de melhorias, possibilitando modularidade, ortogonalidade e refinamento. O projeto de diagramas com superestados permite ao desenvolvedor explorar diferentes níveis de abstração, além possibilitar uma modelagem mais adequada para comportamentos comuns a vários estados. Com relação à ortogonalidade, os diagramas são capazes de modelar ações simultâneas e independentes utilizando a decomposição de superestados em dois ou mais componentes, cujos comportamentos podem ser relacionados por intermédio de condições e operadores lógicos.

Sobre a representação em diagrama, os nós são estados ou superestados, dependendo do nível de abstração da visualização corrente, e os arcos são as transições. Os nós se apresentam na forma de retângulos e, para o caso dos superestados, pode-se detalhar os estados agrupados em seu interior. A modelagem concorrente é representada por superestados divididos, onde os diagramas inscritos em cada divisão comportam-se simultaneamente. Um exemplo de um diagrama *statechart* é apresentado na Figura 2.2, onde é possível observar a transição β partindo do superestado D , ou seja, a transição é válida para ambos os estados, A e C , que estão agrupados em D .

Apesar da relevância desses dois modelos em tarefas envolvidas na construção de sistemas, um terceiro modelo será apresentado com mais detalhes na próxima subseção: a

Figura 2.2: Exemplo de Diagrama *Statechart* (Harel, 1987)

Máquina de Estados Finitos. O foco deste trabalho está nesse terceiro modelo, o que justifica a preocupação em realizar uma melhor exposição de suas características. O modelo é também a base para os métodos de geração de casos de teste que serão apresentados adiante.

2.5.1 Máquinas de Estados Finitos

Uma **Máquina de Estados Finitos** é um modelo de comportamento composto por um número finito de estados e transições (Gill, 1962). Cada transição transfere a máquina de um estado atual s a um próximo estado s' , sendo permitido que o próximo estado s' seja igual ao estado atual s . O estado atual é único, ou seja, a máquina pode estar somente em um estado.

Pode-se encontrar definições de características distintas para MEFs, apresentando pequenas variações em conceitos e funcionalidades. Neste texto será discutida a representação conhecida como **Máquina de Mealy**, onde as ações de entrada e saída são modeladas nas transições. Uma ação de entrada dispara uma transição que pode gerar uma saída. Tanto a saída gerada quanto o novo estado alcançado são definidos unicamente em função do estado atual e do evento de entrada.

As representações adotadas para MEFs são os diagramas e as tabelas de transição. Nos diagramas, os estados são nós, representados por círculos rotulados, conectados por arcos que dizem respeito às transições. Nesses arcos são especificadas a entrada i , a qual dispara a transição representada, e a saída o gerada. Para tal, utiliza-se um rótulo no formato i/o . A Figura 2.3, extraída de (Fujiwara et al., 1991), mostra uma MEF representada em um diagrama de transição de estados.

As tabelas de transição são uma forma de especificar o comportamento de uma MEF em uma estrutura de dados pronta para ser utilizada em aplicações de software. A primeira coluna e a primeira linha contém, respectivamente, os estados de origem e todas as possíveis entradas. As transições são definidas nas intersecções entre seu estado de origem e a entrada que a dispara. Neste ponto, são especificados o próximo estado e a

Figura 2.3: Representação por diagrama de transição de uma MEF (Fujiwara et al., 1991)

saída produzida. A Tabela 2.1 contém uma tabela de transição construída para a MEF da Figura 2.3.

Tabela 2.1: Representação da MEF em tabela de transição

Estado Atual	Entradas		
	<i>a</i>	<i>b</i>	<i>c</i>
<i>S</i> ₀	<i>S</i> ₁ / <i>e</i>	<i>S</i> ₁ / <i>f</i>	<i>S</i> ₂ / <i>e</i>
<i>S</i> ₁	<i>S</i> ₀ / <i>f</i>	<i>S</i> ₂ / <i>f</i>	<i>S</i> ₁ / <i>f</i>
<i>S</i> ₂	<i>S</i> ₂ / <i>f</i>	<i>S</i> ₀ / <i>e</i>	<i>S</i> ₁ / <i>e</i>

Por fim, segue a definição formal de uma máquina de estados finitos, como encontrada em (Gören e Ferguson, 2002). Essa definição será útil para a compreensão dos métodos apresentados na Seção 2.6. Segundo os autores, uma MEF M pode ser definida por uma quintupla $(\Sigma, \Delta, Q, q_0, \delta, \lambda)$, onde:

$\Sigma \neq \emptyset$: um conjunto finito de símbolos de entrada,

$\Delta \neq \emptyset$: um conjunto finito de símbolos de saída,

$Q \neq \emptyset$: um conjunto finito de estados,

$q_0 \in Q$: o estado inicial,

$\delta : Q \times \Sigma \rightarrow Q \cup \{\phi\}$: a função de transição, onde ϕ denota um estado não especificado,

$\lambda : Q \times \Sigma \rightarrow \Delta \cup \{\epsilon\}$: a função de saída, onde ϵ denota uma saída não especificada.

A notação $\Omega(q_i)$ é utilizada para representar o conjunto de todas as seqüências de entrada definidas a partir do estado q_i , sendo Ω_M uma forma de representar $\Omega(q_0)$, referindo-se a todas as seqüências de entrada válidas para a MEF M . As definições de δ e λ podem ser estendidas para os domínios $\delta(q, \alpha) : Q \times \Sigma^* \rightarrow Q \cup \{\emptyset\}$ e $\lambda(q, \alpha) : Q \times \Sigma^* \rightarrow \Delta^* \cup \{\epsilon\}$, onde para todo $\alpha \in \Omega(q_i)$, $\delta(q_i, \alpha)$ é a representação do estado atingido e $\lambda(q_i, \alpha)$ a representação da seqüência de saídas gerada por M , inicialmente em q_i , após a aplicação da seqüência α de entradas.

Uma MEF é **completa** quando estão definidas transições para cada elemento do conjunto de entrada, partindo de cada elemento do conjunto de estados. Caso contrário, a MEF é considerada **parcial**. Quando não mais que uma transição é definida para cada elemento do conjunto de entrada, a MEF é dita **determinística**. Considerando dois estados, q_i e q_j , diz-se que:

- são **equivalentes** quando $\Omega(q_i) = \Omega(q_j)$ e $\forall \alpha \in \Sigma, \lambda(q_i, \alpha) = \lambda(q_j, \alpha)$, ou seja, em ambos os estados estão definidas exatamente as mesmas seqüências de entradas e, para quaisquer seqüências de entradas, as respectivas seqüências de saídas são iguais;
- q_i é **quasi-equivalente** a q_j se $\Omega(q_i) \supseteq \Omega(q_j)$ e $\forall \alpha \in \Omega(q_j), \lambda(q_i, \alpha) = \lambda(q_j, \alpha)$, ou seja, todas as seqüências de entrada definidas em q_j também o estão em q_i e, na aplicação de cada uma dessas seqüências, as saídas produzidas à partir de ambos os estados são iguais;
- ambos são **compatíveis** quando $\nexists \alpha \in \Omega(q_i) \cup \Omega(q_j), \lambda(q_i, \alpha) \neq \lambda(q_j, \alpha)$, ou seja, não há uma seqüência de entradas definida em ambos os estados que leve à produção de seqüências de saídas distintas a partir de cada um deles;
- são **distinguíveis** por uma seqüência $\alpha \in \Omega(q_i) \cup \Omega(q_j)$ se $\lambda(q_i, \alpha) \neq \lambda(q_j, \alpha)$, ou seja, há ao menos uma seqüência de entradas definida a partir de ambos os estados cuja aplicação em cada um gere seqüências de saída distintas.

Estados equivalentes são também quasi-equivalentes e compatíveis. Estados quasi-equivalentes são também compatíveis, mas nem sempre equivalentes. Estados compatíveis não são necessariamente quasi-equivalentes ou equivalentes. Estados distinguíveis não são compatíveis, quasi-equivalentes ou equivalentes.

Quando para cada par de estados (q_i, q_j) , existe pelo menos uma seqüência de entradas α de forma que $\delta(q_i, \alpha) = q_j$, a máquina é **fortemente conexa**. Duas MEFs são consideradas equivalentes, ou quasi-equivalentes, quando seus estados iniciais são equivalentes, ou, respectivamente, quasi-equivalentes. Diz-se que uma MEF é **reduzida** quando todos os estados nela presente são distinguíveis entre si. Uma MEF M é dita **mínima** quando não existem outras máquinas com menor número de estados ou transições capazes de representar as mesmas funcionalidades que M .

2.6 Métodos de Geração de Casos de Teste

Nesta seção são apresentados os métodos para a geração de seqüências de teste baseada em especificação por máquinas de estados finitos. De acordo com (Fujiwara et al., 1991), estes métodos devem gerar conjuntos de seqüências capazes de revelar os defeitos que estejam presentes nas implementações, sem perder de vista que a quantidade de seqüências nos conjuntos não deve inviabilizar sua aplicação.

Para uma melhor compreensão dos métodos, na próxima subseção são apresentados alguns conceitos importantes. Primeiramente, discutir-se-á a abordagem comum para a utilização desses métodos. Em seguida serão enumerados os tipos de erros que podem estar presentes em uma implementação que tenha sido baseada em uma especificação formal modelada por uma MEF.

2.6.1 Conceitos Envolvidos

Seja S uma MEF construída para modelar uma especificação. Uma implementação dessa especificação externará um comportamento tal como uma segunda MEF I . Para validar essa implementação, deve-se mostrar que I é equivalente a S . Não é possível compará-las explicitamente, pois se parte do princípio que a construção da MEF I não é conhecida. No entanto, é possível entrar com seqüências em I e obter suas respectivas saídas, ou seja, tem-se uma implementação que é executável. Como foi discutido na Seção 2.4, a prática de testes nessas circunstâncias é classificada como teste caixa-preta, ou seja, os métodos aos quais se referem a presente seção são procedimentos geradores relacionados a critérios baseados em especificação.

Basicamente, a estratégia consiste em gerar seqüências de entradas, a partir de uma MEF conhecida, cujas saídas só poderiam ser reproduzidas por uma outra MEF equivalente à primeira. Dessa forma, aplicando-se as seqüências geradas a partir da especificação S (conhecida) na implementação I (desconhecida) e comparando as saídas obtidas com aquelas que seriam exibidas por S , é possível garantir que a MEF ali implementada representa exatamente o mesmo comportamento especificado em S , pois seriam equivalentes. Nesse contexto, pode-se dizer que revelar a existência de diferenças entre a MEF que modela a especificação e aquela utilizada na elaboração da implementação tem o mesmo significado que revelar defeitos presentes na implementação.

A questão que os métodos buscam então solucionar consiste em, dada uma MEF S , selecionar seqüências de entrada que gerem saídas idênticas somente quando aplicadas em uma outra MEF I equivalente a S . É trivial que a aplicação de todas as seqüências possíveis para os símbolos de entrada válidos entre duas MEFs não equivalentes seja suficiente para revelar suas diferenças. No entanto, como foi discutido na definição de critérios de teste, essa solução não é viável para situações práticas, uma vez que o número

grande, ou mesmo infinito, de seqüências possíveis possa implicar em um pesado ônus para a atividade de teste.

A busca por um conjunto de seqüências aplicável, ou seja, não demasiadamente grande, porém com tal capacidade de revelar diferenças, explora características particulares do modelo formal em questão. Na comparação de duas MEFs, considerando-as como especificação/implementação, através da aplicação de seqüências de teste, pode-se revelar defeitos de construção das seguintes naturezas (Chow, 1978):

Transfer Faults: quando o estado atingido por uma transição não é o correto. Neste texto, utilizar-se-á a expressão **erro de transferência** para apontá-los.

Output Faults: quando a saída gerada por uma transição não é a correta. A expressão adotada neste texto para este tipo de defeito é **erro de saída**.

Missing States: (ou **estados a menos**) quando estados faltam na implementação.

Extra States: (ou **estados a mais**) quando estados sobram na implementação.

Quando um conjunto de seqüências de teste contempla todas as possibilidades de defeitos supracitadas, o resultado de sua aplicação em uma implementação sob teste garante a equivalência entre as MEFs envolvidas. Conjuntos com essas características, e que utilizam o menor número possível de seqüências, são o objetivo dos métodos de geração de casos de teste baseados em especificação por MEF.

A seguir, são discutidas algumas estratégias triviais para a seleção de seqüências de teste. Essas estratégias são utilizadas como parte de métodos mais avançados, como os apresentados mais adiante.

2.6.1.1 State Cover

Trata-se de um conjunto de seqüências de entradas que faz com que todos os estados de uma MEF sejam atingidos. Para uma MEF com n estados, um conjunto *state cover* contém n seqüências, incluindo a seqüência vazia ϵ . A seqüência ϵ faz com que a MEF fique em seu estado inicial, enquanto cada uma das n seqüências levam a máquina para cada um de seus n estados. Normalmente, um conjunto dessa natureza é referenciado por Q .

As seqüências de um conjunto Q por si não garantem a detecção de todos os defeitos de nenhum dos tipos apresentados, embora alguns defeitos possam ser eventualmente revelados. Considerando a MEF exibida na Figura 2.3, apresenta-se o conjunto *state cover*: $Q = \{\epsilon, a, c\}$.

2.6.1.2 Transition Cover

Um conjunto de seqüências de entrada é considerado um **transition cover** se, quando aplicado a uma MEF M , exercita (ao menos) uma vez cada uma das transições definidas em M (Naito e Tsunoyama, 1981). Autores que trabalham nessa linha de pesquisa costumam referir-se a esses conjuntos como P (Chow, 1978; Fujiwara et al., 1991).

Somente a utilização das seqüências de um conjunto P para testar uma implementação I revela todos os erros de saída que possam estar inseridos em I , porém não apresenta garantias de que todos os erros de transferência sejam encontrados. Isso pode ser observado no contra-exemplo da Figura 2.4, onde está representada uma MEF que gera saídas idênticas para o conjunto de entradas $P = \{a, b, c, aa, ab, ac, ca, cb, cc\}$, construído como um possível conjunto *transition cover* a partir da MEF exibida na Figura 2.3.

Figura 2.4: MEF exibindo erro de transferência para o conjunto P

Para entender onde uma falha de transferência não é revelada, basta observar a seqüência ca , que em ambas as MEFs gera a saída ef , porém, a MEF da Figura 2.4 é transferida ao estado S_1 , diferentemente da MEF da Figura 2.3, que permanece em S_2 .

2.6.2 Método W

Chow (1978) propôs um método que foi por ele chamado de *Automata Theoretic*. Apesar do nome introduzido pelo autor, o método acabou ficando conhecido como *The W Method*, pois W é a denominação lá utilizada para o conjunto de caracterização, a base para o funcionamento do método.

Um **conjunto de caracterização** (também conhecido como **conjunto de identificação**) é um conjunto de seqüências de entradas que, através das saídas correspondentes, pode distinguir cada um dos estados de uma MEF mínima. Em outras palavras, são

(uma ou mais) seqüências que sempre resultam em saídas distintas quando aplicadas em estados distintos. Sua função é possibilitar que seja verificada a existência de estados implementados em I que sejam correspondentes aos estados especificados em S .

Existem diversas abordagens para a geração sistemática desses conjuntos (Gill, 1962), no entanto, essas abordagens não serão aqui discutidas, uma vez que não são relevantes ao objetivo do presente estudo. Por outro lado, será apresentado um conjunto de identificação construído sobre uma MEF simples, ilustrada na Figura 2.5, a qual também será mais adiante utilizada em exemplos.

Figura 2.5: Uma MEF simples para utilização em exemplos

Dada a pouca complexidade da MEF, não é difícil concluir que o conjunto $W = \{bb\}$ é suficiente para identificar cada estado nela presente, através das seguintes saídas:

$$S_0 \mid bb = 10;$$

$$S_1 \mid bb = 01;$$

$$S_2 \mid bb = 00.$$

Normalmente, mais de um conjunto de caracterização pode ser encontrado em uma mesma MEF. No exemplo apresentado, $W' = \{aa\}$ também seria de mesma eficácia para o reconhecimento dos estados. Ambos os conjuntos apresentados contêm apenas uma seqüência, de tamanho idêntico. Quando mais de um conjunto de identificação está disponível, é sempre interessante que se utilize aquele que implique em seqüências de teste com menor custo de aplicação, uma vez que a qualidade do resultado é idêntica.

O método proposto por Chow consiste em aplicar as seqüências de um conjunto *transition cover*, concatenadas com seqüências de um conjunto de identificação. Isso garante que todas as transições da máquina testada sejam exercitadas e relaciona cada um dos estados alcançados com um único estado da especificação. A aplicação do método W para a especificação representada pela MEF da Figura 2.5 seria:

$$\text{Conjunto de caracterização } W = \{bb\};$$

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

Casos de teste $P \bullet W = \{bb, abb, bbb, aabb, abbb, babb, bbbb\}$.

Ainda é possível eliminar do conjunto as seqüências $\{bb, abb, bbb\}$, pois elas são prefixos de outras seqüências presentes no conjunto. Isso significa que, por exemplo, ao executar uma seqüência como $bbbb$, o resultado que seria produzido pelas seqüências bb e bbb já está sendo verificado. Portanto, o conjunto final com os casos de teste gerados é $\{aabb, abbb, babb, bbbb\}$.

Implementações que produzam as saídas $\{1000, 1010, 1010, 1001\}$ para as entradas do conjunto de teste gerado, tal como definidas na especificação da MEF da Figura 2.5, também produzirão saídas compatíveis com a especificação na aplicação de quaisquer outras seqüências de entradas válidas. Chow (1978) demonstrou que o método provê tal garantia, desde que as seguintes restrições sejam observadas: o método deve ser aplicado somente em MEFs mínimas, completamente especificadas e determinísticas. Exige-se que seja estimado o número de estados utilizados na implementação: para ser aplicado da maneira como foi descrita e obter-se a cobertura de todas as falhas, é necessário que a implementação possua exatamente o mesmo número de estados da especificação.

Caso estime-se que a implementação faz uso de um número superior de estados, é necessário construir um conjunto estendido de identificação Z , que consiste nas seqüências do conjunto W concatenadas a prefixos formados a partir de todas as possíveis combinações de k símbolos do conjunto de entrada, onde k é a diferença entre o número de estados estimados e o número de estados da especificação.

2.7 Análise da Cobertura de Casos de Teste

Uma questão importante no estudo das técnicas de teste baseado em especificação por MEF é a avaliação da **qualidade**, ou **cobertura de falhas**, de um conjunto gerado (Yao et al., 1994). Ao longo da seção anterior, o método W foi introduzido. O autor deste método apresentou provas formais que garantem a cobertura de determinados tipos de defeitos pelos conjuntos gerados. Dessa forma, sabendo-se qual o método utilizado para gerar determinado conjunto de casos de teste, sabe-se também os tipos de falhas que este conjunto garante descobrir.

Por outro lado, quando se tem disponível um conjunto de casos de teste gerado a partir de um método que não é conhecido, é importante definir maneiras de se avaliar esse conjunto. O objetivo de tal avaliação é revelar, sob algum aspecto, sua capacidade de encontrar falhas. Como discutido na Subseção 2.6.1, o conjunto de casos de teste ideal é aquele com menor custo de aplicação, ou seja, o menor conjunto em relação à quantidade e o tamanho das seqüências de entradas incluídas, que, por meio das seqüências de saídas

produzidas em sua aplicação, revele todas as possíveis diferenças entre a especificação a partir do qual foi gerado, e qualquer implementação interativa desconhecida.

Quando um conjunto possui tal capacidade de detecção de falhas, o mesmo é classificado como **m-completo** e, quando utilizado em uma atividade de teste, aprova somente implementações totalmente compatíveis com o comportamento especificado. O nome m-completo alude à capacidade de encontrar erros em implementações que contenham um número m , desconhecido, de estados. Em uma segunda classificação, os conjuntos de teste que provêem cobertura total de falhas somente quando se tem conhecimento de que número de estados utilizados na construção da implementação sob teste é idêntico ao número n de estados da MEF de especificação. Um conjunto com estas características é dito ser **n-completo**.

2.7.1 Verificação da Completude Através da Minimização de MEF

Yao et al. (1994) mostraram que o problema da verificação da completude de um conjunto de casos de teste pode ser convertido no problema de se reduzir MEFs. O método consiste em reconstruir a MEF a partir das seqüências que compõem o conjunto. Após reconstruída, a MEF deve ser minimizada, e então comparada com a MEF de especificação. Verificar a equivalência das MEFs é suficiente para decidir quando o conjunto é m-completo. Os resultados apresentados pelos autores demonstram a validade dessa abordagem, que é descrita com mais detalhes a seguir.

A partir de um conjunto de seqüências de entradas e saídas (E/S), sejam essas seqüências parte de uma especificação ou casos de teste gerados a partir de um dos métodos apresentados ao longo deste capítulo, é possível construir uma MEF capaz de exibir um comportamento consistente, ou seja, produzir saídas semelhantes àquelas que estão definidas nas seqüências do conjunto. Para tal, basta organizar os valores de E/S em uma estrutura de árvore, explorando seqüências que possuem valores em comum (prefixos iguais). Um exemplo desse processo é apresentado a seguir, considerando as seqüências listadas na Tabela 2.2, geradas a partir do método W para a MEF da Figura 2.5.

Tabela 2.2: Seqüências de Entrada/Saída geradas pelo método W

Entradas/Saídas			
$a/1$	$a/0$	$b/0$	$b/0$
$a/1$	$b/0$	$b/1$	$b/0$
$b/1$	$a/0$	$b/1$	$b/0$
$b/1$	$b/0$	$b/0$	$b/1$

Pode-se observar que as seqüências definidas na Tabela 2.2 são reproduzidas pela MEF da Figura 2.6. Tal MEF é construída com a seguinte estratégia: estabelece-se um estado inicial, no exemplo q_0 , e a partir desse estado, são criadas ramificações para cada seqüência

de E/S no conjunto. O número de transições em cada ramificação é equivalente ao número de valores de E/S da sequência correspondente. Essas transições são ordenadamente definidas de acordo com cada valor de E/S da sequência a qual estão relacionadas. Quando duas seqüências de E/S principiam por valores de entrada comuns, ou seja, contam com um prefixo em comum, suas ramificações correspondentes são combinadas neste prefixo, de forma a se obter uma MEF determinística.

Figura 2.6: MEF em árvore (TFSM)

As MEFs que contam com essa característica, a organização em forma de árvore, recebem a denominação *Tree Finite State Machines (TFSM)*, ou ainda *Tree Machines*. Neste texto, o termo utilizado para fazer referência a esse tipo de MEF será **MEF em árvore**. Uma MEF em árvore é uma MEF cuja representação em diagrama resulta em um grafo com estrutura de árvore, tendo como nó raiz o estado inicial (Yao et al., 1994). Com base nas propriedades desse tipo peculiar de grafo, sabe-se que sempre há um, e apenas um, caminho entre o estado inicial e cada um dos outros estados de uma MEF em árvore. Ainda, para cada estado (exceto o estado inicial) S_i dessas MEFs, existe um, e somente um, estado S_j que o precede, ou seja, a partir do qual se pode originar uma transição para S_i .

Segundo Yao et al. (1994), para afirmar que o conjunto de seqüências utilizadas na construção da MEF em árvore é m-completo, é suficiente verificar a equivalência entre a MEF construída e a especificação, a partir da qual as seqüências foram geradas. Os autores propõem que tal tarefa seja realizada comparando ambas as máquinas em sua forma mínima. Dessa maneira, o problema de se decidir se um dado conjunto de seqüências de E/S é m-completo se resume ao problema da redução de estados, ou seja, a minimização de MEFs. Na Figura 2.7, (a) exhibe a MEF em árvore da Figura 2.6 com os estados redundantes combinados (forma mínima), enquanto (b) reproduz novamente a MEF de

especificação, da Figura 2.5. Foi a partir desta segunda MEF que método W gerou os casos de teste sobre os quais a MEF em árvore foi construída. É visível a equivalência entre as MEFs, comprovando que as seqüências geradas pelo método W são, de fato, um conjunto m-completo.

Figura 2.7: MEF reconstruída reduzida e MEF da especificação

No que diz respeito à solução sob essa perspectiva, como discutida na Seção 3.1, o tempo necessário para a redução é exponencial em relação ao número inicial de estados, particularmente quando a MEF em questão não é completamente especificada. No processo de construção de MEFs em árvore, o número de estados é afetado pelo tamanho e quantidade de seqüências de E/S consideradas, além do fato de que a máquina resultante deverá ser parcialmente especificada na maioria dos casos práticos. Tais características tornam a aplicação da abordagem de Yao et al. (1994) dependente de boas soluções para o problema da minimização de MEFs. Características do problema e algumas das soluções disponíveis são discutidas no próximo capítulo.

2.8 Considerações Finais

Ao longo deste capítulo foram apresentados conceitos e técnicas gerais sobre teste de software. A importância da atividade de teste nos processos de desenvolvimento foi considerada na Seção 2.2. O conceito de critério de teste foi introduzido na Seção 2.4 e suas classificações, funcional e estrutural, foram diferenciadas. Maior atenção foi direcionada ao teste baseado em especificação, com ênfase nas técnicas de geração de casos de teste a partir de MEFs.

A Seção 2.5 dedicou-se ao uso dos modelos formais na Engenharia de Software. Nela se discutiu sobre como a definição de especificações utilizando esses modelos aumenta a precisão e remove possíveis ambigüidades as quais são suscetíveis as descrições informais, como as que utilizam textos em linguagem natural. Alguns modelos formais foram apresentados, dentre eles, as MEFs, que receberam maior atenção.

O uso de MEFs para modelar o comportamento de sistemas reativos foi introduzido na Seção 2.5, assim como, na Seção 2.6, as técnicas para a geração de casos de teste a partir desses modelos. Foi dito que implementações baseadas em MEFs estão sujeitas a tipos específicos de erros, e que determinadas técnicas de teste, sendo uma delas descrita em detalhes, são capazes de garantir a total conformidade com a especificação através de testes funcionais, desde que algumas condições sejam satisfeitas. Para a maioria destas técnicas, tais condições dependem da disponibilidade de MEFs mínimas.

A Seção 2.7 trouxe informações sobre a análise da cobertura de falhas para casos de teste gerados a partir de especificações modeladas em MEFs. Nela explicou-se como é possível utilizar técnicas de minimização de MEFs parciais para se avaliar quando um conjunto de casos de teste é capaz de garantir a conformidade entre especificação e implementação, evidenciando a relação entre o uso de MEFs na elaboração/teste de sistemas e a motivação para o aprimoramento de técnicas de minimização de MEFs parciais.

Minimização de Máquinas de Estados Finitos

3.1 Considerações Iniciais

É possível encontrar estados redundantes, ou seja, de comportamento semelhante, em modelos de máquinas de estados finitos. Esses estados são inseridos inadvertidamente pelo projetista da especificação ou por aplicações que geram modelos formais a partir de descrições de alto nível (Kannan e Sarma, 1991). Como apresentado na Seção 2.5.1, uma MEF M é dita como mínima quando não existe uma outra MEF N , com menor número de estados em relação a M , capaz de representar o mesmo comportamento que M . Uma MEF não é mínima quando possui estados redundantes. A eliminação desses estados, ou seja, a minimização de uma MEF, resulta em vários benefícios para projetos que fazem uso desses modelos.

Máquinas reduzidas resultam em maior simplicidade na utilização, com modelos mais fáceis de serem compreendidos pelos desenvolvedores. Implementações que utilizam essas máquinas têm melhor desempenho se comparadas com aquelas que consideram estados redundantes, pois estes levam à execução de um maior número de operações desnecessárias, obtendo-se resultados idênticos. Destaca-se que grande parte dos métodos de geração de casos de teste baseados em MEFs, como o método W, apresentado no capítulo anterior, são aplicáveis somente em MEFs que não possuem estados redundantes.

Para ilustrar o que foi discutido, a Figura 3.1 contém o seguinte exemplo: em (a) é exibida uma MEF que possui três estados, no entanto os estados S_1 e S_2 são redundantes. Em (b) é possível verificar como uma outra MEF, com apenas dois estados, é capaz de responder às entradas de maneira semelhante à primeira. Os estados S_1 e S_2 são redundantes, e o mesmo comportamento é obtido quando estes estão combinados em um único estado $S_{1,2}$. O resultado é um modelo mais simples, porém representando a mesma funcionalidade em relação ao original.

Figura 3.1: Redução de uma MEF

Embora em modelos simples e relativamente pequenos, como o apresentado no exemplo, a identificação de estados redundantes possa ser realizada até mesmo visualmente, modelos de maiores proporções tornam extremamente difícil a localização e eliminação desses estados. Para auxiliar na tarefa, existem algoritmos capazes de efetuar todo o processo de redução de maneira automatizada. Todavia, a natureza do problema faz com que, em alguns casos, o custo de aplicação desses algoritmos seja exponencial em relação ao número de estados da máquina a qual se deseja minimizar.

Para máquinas completas, é possível efetuar o processo de análise e redução em tempo polinomial. Para tal, utiliza-se o algoritmo de redução de autômatos finitos de Hopcroft (1972). No entanto, o problema de se minimizar MEFs parciais implica em grande complexidade, provavelmente exponencial. Foi a conclusão publicada por Pfleeger (1973), onde o autor mostrou como o problema da coloração em grafos, um problema já devidamente reconhecido como pertencente ao conjunto dos problemas NP-completo, é redutível ao problema da minimização de autômatos de estados finitos parcialmente especificados.

A publicação de tal conclusão motiva a busca por heurísticas e algoritmos otimizados para a solução do problema de maneira exata, ou mesmo aproximada, em tempo satisfatório. Esse fato é constatado por trabalhos das últimas décadas onde propostas com esse objetivo foram apresentadas (Gören e Ferguson, 2007; Hachtel et al., 1991; Higuchi e Matsunaga, 1996; Kam et al., 1994; Pena e Oliveira, 1998). Dentre outros benefícios, boas soluções para o problema viabilizam o uso da estratégia de Yao et al. (1994) para a análise da cobertura de falhas em conjuntos de casos de teste para MEFs.

A seguir, será introduzida a abordagem clássica para a minimização de máquinas parciais, seguida de discussões acerca de algumas propostas de soluções aprimoradas para o problema.

3.2 Minimização de MEFs Parciais

A abordagem descrita nesta seção é oriunda de um dos primeiros trabalhos neste tópico, publicado em 1959 por Paull e Unger. Os autores propuseram um método baseado na seleção de conjuntos de classes de compatibilidade, definidas mais adiante, de forma que os estados que compõem uma determinada classe possam ser combinados sem implicar em alterações no comportamento externo da MEF. Tal estratégia é conhecida como a abordagem clássica para a minimização de MEFs parciais.

Dois estados pertencentes a uma mesma MEF são ditos **compatíveis** se ambos produzem a mesma saída quando submetidos a qualquer seqüência válida de entradas. São esses estados, anteriormente referidos por estados redundantes, que devem ser eliminados para que se obtenha a MEF mínima. Uma **classe de compatibilidade** $C = \{Q_0, Q_1, \dots, Q_n\}$ é um conjunto de estados compatíveis entre si, quando comparados dois a dois.

A estratégia clássica para a minimização de MEFs não completamente especificadas é baseada em enumerar todas as possíveis classes de compatibilidade, combinando-as exaustivamente até que se obtenha um conjunto de classes com mínima cardinalidade, sujeito a algumas restrições. Mais especificamente, pode-se encarar o processo de minimização como um problema de cobertura: a MEF mínima pode ser encontrada através da identificação de uma cobertura fechada, de cardinalidade mínima, sobre o conjunto de classes de compatibilidade (Pena e Oliveira, 1998).

Um conjunto de classes de compatibilidade $S = \{C_1, C_2, \dots, C_n\}$ cobre os estados da MEF $M = (\Sigma, \Delta, Q, Q_0, \delta, \lambda)$ se cada um dos estados de M pertencer a uma única classe C do conjunto S . Obtida a cobertura, a garantia de que a mesma é fechada para S se dá pela restrição: $\forall \alpha \in \Sigma, C_i, C_j \in S, \forall Q \in C_i, \delta(Q, \alpha) = Q' \Leftrightarrow Q' \in C_j$, ou seja, os próximos estados atingidos por transições que partem de estados que estão em uma determinada classes C_i , quando definidas para um símbolo de entrada válido α , devem estar contidos em uma única classe C_j , permitindo-se que $i = j$. O conjunto de próximos estados atingidos a partir daqueles pertencentes a uma classe C para um símbolo de entrada α é chamado de **conjunto implícito**, representado como $D_\alpha(C)$. Portanto, uma cobertura é fechada quando $\forall C_i \in S$ e $\forall \alpha \in \Sigma, D_\alpha(C_i) \subset C_k$ tal que $C_k \in S$.

Dadas essas condições, encontrar a MEF mínima compatível com M passa ser uma questão de selecionar a cobertura fechada para S com menor cardinalidade. Tal problema pode ser formulado como uma instância do problema NP-hard *binate covering*, solucionável através de abordagens específicas (Olivier Coudert, 1995).

Ao se considerar particularidades da questão da minimização de MEFs parciais, otimizações podem ser inseridas, reduzindo drasticamente o espaço de busca (Pena e Oliveira, 1998). Pode-se notar que os conjuntos de classes de compatibilidade com cardinalidade 1 podem ser ignorados no processo, uma vez que estes não geram qualquer redução em relação ao número de estados da MEF original, assim como todos os conjuntos implícitos dominados por aquelas classes de compatibilidade que os geraram.

Uma classe de compatibilidade C' domina outra classe C quando $C' \supset C$ e $\forall \alpha, D_\alpha(C') \subseteq D_\alpha(C)$. Grasselli e Luccio (1965) provaram que somente aquelas classes que não são dominadas por nenhuma outra classe de compatibilidade precisam ser consideradas no procedimento, chamadas de classes de compatibilidade primárias (do termo *prime compatibles*). Essa observação resulta em grande otimização na busca pelo conjunto de classes com cardinalidade mínima.

Hachtel et al. (1991) publicaram um trabalho acerca da viabilidade de se resolver instâncias reais, ou seja, não geradas por métodos artificiais para efeito de avaliação dos métodos, do problema da minimização de MEFs parciais. Na ocasião, os autores apresentaram uma versão do método de Grasselli e Luccio (1965) baseada na enumeração explícita das classes de compatibilidade. Tal versão, conhecida como **stamina**, mostrou grande performance no processamento de MEFs projetadas a mão para fins reais, sendo até hoje utilizada como referência.

Kam et al. (1994) propuseram uma estratégia baseada na enumeração implícita. A principal vantagem dessa estratégia, chamada de **ism**, surge quando é aplicada em MEFs geradas por processos de síntese de lógica para aplicações reais, as quais usam grande número de estados. Tal vantagem existe uma vez que a estratégia de enumeração explícita muitas vezes excede as limitações do espaço de memória disponível.

Higuchi e Matsunaga (1996) apresentaram um novo algoritmo heurístico baseado em incrementos iterativos para a redução de MEFs parciais. Técnicas foram mescladas: para ganhar performance, quando o número de classes de compatibilidade é menor que um determinado limiar, a enumeração explícita é utilizada. Quando tal limiar é excedido, diagramas de decisão binários são utilizados, evitando uma explosão combinatorial.

A seguir, dois dos mais recentes trabalhos que abordam o problema da minimização de MEFs parciais são apresentados com maiores detalhes: os métodos **bica** e **chesmin**.

3.2.1 Método bica

Pena e Oliveira (1998) propuseram uma forma de minimizar máquinas parciais que não é baseada na enumeração dos conjuntos das classes de compatibilidade, mas sim em técnicas de identificação de MEFs, previamente utilizadas em atividades de teste (Biermann e Feldman, 1972). O método foi apresentado com o nome de **bica**.

A proposta explora a reconstrução de MEFs parciais arbitrárias como MEFs em árvore. Tais MEFs possuem aspectos que tornam o processo de minimização mais simples. Esses aspectos são considerados pelo algoritmo de Biermann e Feldman (1972), construído especificamente para a minimização desses modelos. O algoritmo, que é discutido a seguir, está inserido no método **bica** como uma das etapas.

Biermann e Feldman fazem uso do conceito **função de mapeamento**, que refere-se a uma função que associa estados de comportamento compatível entre duas MEFs distintas. A função é considerada válida quando cada um dos estados da primeira é associado a pelo menos um estado da segunda MEF. Sempre deve existir uma função de mapeamento válida entre uma MEF e sua versão reduzida, pois todos os estados da MEF original devem existir na MEF mínima, com atenção especial para os estados redundantes, que lá devem estar presentes uma única vez. O estabelecimento de uma função de mapeamento entre duas MEFs garante que ambas são compatíveis, porém, dadas duas MEFs compatíveis, nem sempre é possível estabelecer uma função (Pena e Oliveira, 1998).

O problema de minimização pode ser então resolvido encontrando-se a função de mapeamento válida entre uma MEF e sua versão reduzida. Ao abordar o problema desta perspectiva, as formas de solucioná-lo para MEFs em árvore contemplam algoritmos totalmente diferentes daqueles exigidos para MEFs parciais em geral (Pena e Oliveira, 1998). Segundo os autores, quando se trata de MEFs em árvore, o fato de existir somente um único predecessor e um único caminho que atinja cada estado garante que sempre será possível estabelecer uma função de mapeamento válida com uma segunda MEF de comportamento compatível, se tal MEF existir.

A busca pela função F de mapeamento tem por base duas restrições, considerando S_i um estado para o qual será mapeado um estado Q_i , ou seja, $S_i = F(Q_i)$: (i) se dois estados Q_i e Q_j , na MEF em árvore original, são incompatíveis, então $S_i \neq S_j$; (ii) se dois estados Q_i e Q_j têm como sucessores, respectivamente, os estados Q_k e Q_l , para alguma entrada α , então $S_i = S_j \Rightarrow S_k = S_l$.

Na busca pela função de mapeamento que possibilite a construção de uma MEF reduzida, exibindo comportamento compatível com uma MEF em árvore, não é necessário considerar todas as possíveis relações para funções de mapeamento, mas somente as de natureza muitos-para-um. Se uma máquina M' possui m estados e é compatível com uma máquina M , com n estados, há no máximo n^m possíveis funções de mapeamento, enquanto existem 2^{mn} possíveis relações entre estados. Embora as quantidades sejam consideravelmente grandes, o algoritmo de Biermann e Feldman realiza essa busca de uma forma muito mais eficiente que a abordagem clássica para a minimização de MEFs parciais (Pena e Oliveira, 1998).

Para extrair alguma vantagem deste algoritmo na minimização de MEFs parciais de uma maneira geral, o método **bica** as reestrutura em árvores. Para tal, parte-se de

seqüências de E/S geradas a partir da MEF original. Estas são então organizadas utilizando uma estratégia semelhante à descrita na Subseção 2.7.1. A seguir é apresentada uma visão geral dos passos utilizados pelo método para a redução de uma MEF parcial M' .

1. Definir um conjunto R de seqüências de E/S, a partir de seqüências de entrada arbitrárias válidas para M' .
2. Construir uma MEF M'_R , em árvore, a partir das seqüências de R , tal como descrito na Subseção 2.7.1.
3. Construir M , uma MEF reduzida, compatível com M'_R , usando o algoritmo exato de minimização para MEFs em árvore.
4. Testar se M é compatível com M' . Caso positivo, o objetivo foi atingido. Caso negativo, selecionar um contra-exemplo, ou seja, uma seqüência s tal que $\lambda(q_0, s) \neq \lambda'(q'_0, s)$.
5. Adicionar o contra-exemplo a R , $R = R \cup \{(s, \lambda'(q'_0, s))\}$. Voltar ao passo 2.

Para tornar o processo mais eficiente, os autores definiram um procedimento para a geração das seqüências iniciais de E/S com base em uma versão adaptada do algoritmo L^* , de Angluin (1987). O algoritmo original foi proposto para, a partir de um conjunto de sentenças de uma linguagem, determinar o autômato finito determinístico canônico (com número mínimo de estados) que reconhece tal linguagem. Pena e Oliveira utilizam as primeiras etapas do algoritmo como um critério para selecionar as seqüências de E/S que serão utilizadas no início do processo de minimização. Combinando essas estratégias, o método consegue bons resultados como os publicados pelos autores.

3.2.2 Método chesmin

Gören e Ferguson (2002) propuseram um método de minimização baseado em seqüências de E/S, denominado *checking sequence based state minimization*, ou *chesmin*. Os autores mostraram experimentalmente que a nova estratégia apresenta eficácia semelhante e, em muitos casos, maior eficiência em relação aos métodos que eram anteriormente utilizados em problemas de redução de estados. A seguir, uma descrição de seu funcionamento será apresentada.

Primeiramente, o método exige que se adicione um novo estado à MEF original, para o qual são direcionadas transições a partir de todos os demais estados. O novo estado também deve possuir uma transição para retornar ao estado inicial da MEF. Essa alteração implementa uma função de *reset*, ou seja, a partir de qualquer estado, uma seqüência

específica de entradas leva a MEF ao novo estado e, em seguida ao estado inicial. A alteração também garante que todos os estados passam a ser alcançáveis, evitando possíveis *deadlocks*.

Serão consideradas duas MEFs: M' , uma MEF parcial que se deseja minimizar e M , uma MEF compatível com M' , porém que possui menos estados. Antes de iniciar o procedimento de minimização, é necessário que se forneça uma estimativa para o limite superior, o número máximo de estados esperados em M . Partir de valores baixos para o limite superior permite se chegar mais rápido ao melhor resultado, porém um valor menor do que o número mínimo necessário de estados faz com que o método falhe. Como valor inicial, os autores sugerem adotar o número de estados em M' decrescido de um. O método também utiliza uma seqüência de E/S *seq* válida para a MEF original, sendo inicialmente selecionada aquela que ativa a função de *reset*, implementada em M' .

O algoritmo de minimização constrói uma MEF M , consistente com a MEF M' original, utilizando um número de estados não maior que o limite fornecido. Para tal, percorre-se M' e, com base na seqüência de E/S selecionada, são criadas as transições em M . Os autores classificam uma MEF M **consistente** para uma seqüência *seq* quando M produz uma seqüência de saída que é compatível com aquela gerada por M' para a mesma seqüência de entrada *seq*.

Se foi possível construir uma MEF M , consistente com M' , verifica-se então a equivalência entre ambas. Utiliza-se um procedimento para encontrar seqüências de entrada que resultam em saídas distintas quando aplicadas em M e M' . Se uma seqüência de E/S com tal característica for identificada, essa seqüência é concatenada à seqüência anteriormente utilizada e o processo retorna ao ponto em que se tenta encontrar uma nova MEF consistente com M , dessa vez para a nova combinação de E/S resultante da concatenação. Caso nenhuma seqüência que indique a não equivalência entre as MEFs seja encontrada, então uma versão menor da MEF M' foi gerada. Nesse ponto, o método apresenta o resultado (a MEF M), corrige o limite superior para o número de estados como sendo o número de estados de M decrescido de um, e retorna ao ponto em que tenta encontrar uma nova MEF consistente com M' , agora em busca de melhores resultados, com menos estados.

Se não foi possível construir uma MEF M , consistente com M' , então o método de minimização termina. Nesse caso, o melhor resultado é aquele obtido na última iteração em que uma M equivalente foi produzida. Se nenhum resultado foi antes gerado, o método simplesmente falhou para M' . Uma visão geral do processo está representada no fluxograma da Figura 3.2, onde $|Q'|$ e $|Q|$ são, respectivamente, o número de estados de M' e M .

De acordo com os resultados experimentais apresentados pelos autores (Gören e Ferguson, 2002), o método é praticamente tão eficaz quanto aqueles que até então eram utilizados. Além da eficácia semelhante, o novo método mostrou-se também mais eficiente para maioria dos exemplos utilizados nas comparações. Um aprimoramento foi

Figura 3.2: Visão geral do método *chesmin* (Gören e Ferguson, 2002)

recentemente introduzido (Gören e Ferguson, 2007), fazendo o uso do conceito de limite inferior de estados para a MEF reduzida.

3.3 Considerações Finais

A questão da minimização de MEFs parciais foi exposta na Seção 3.1, onde foram discutidas as características do problema que tornam sua solução computacionalmente cara. Também foram citadas abordagens que visam reduzir esse custo com o uso de heurísticas e outras otimizações. Duas das mais recentes propostas foram discutidas em detalhes, completando uma visão geral da evolução desse tópico de pesquisa.

No capítulo seguinte será apresentada uma abordagem para a minimização de MEFs parciais, que consiste na contribuição principal do presente trabalho.

Método Cosme

4.1 Considerações Iniciais

Neste capítulo é descrita e discutida a proposta para a qual o presente trabalho é direcionado: uma nova abordagem para a solução do problema da minimização de MEFs parciais. Tal abordagem foi batizada de **cosme**, com base nos termos em inglês: *Compatible State Merging*. O **cosme** explora características de um modelo denominado grafo de distinção, um grafo não-direcionado onde os vértices estão mapeados aos estados de uma determinada MEF, e arestas existem entre dois vértices quando estes estão relacionados a estados compatíveis. A definição formal e o procedimento para a construção de um grafo de distinção estão descritos na Seção 4.2. A partir de um grafo de distinção, uma estratégia, descrita em detalhes na Seção 4.3, é utilizada para a seleção de estados compatíveis. Para este objetivo, explora-se as informações contidas em cliques maximais, ou seja, conjuntos de vértices conectados entre si com maior cardinalidade, os quais, neste contexto, representam agrupamentos de estados incompatíveis entre si. Conhecendo-se os estados compatíveis, pode-se separar os compatíveis em classes de compatibilidade, que, quando não violam as duas condições necessárias para a obtenção da cobertura fechada na minimização de MEFs, como descritas no capítulo anterior, podem ser combinados de forma a gerar uma MEF de mesmo comportamento, com menos estados. Este procedimento para a combinação de estados é discutido na Seção 4.4. Por fim, alguns exemplos que ilustram a aplicação do método **cosme** na solução de algumas instâncias do problema são apresentados na Seção 4.6.

4.2 Construção do Grafo de Distinção

Seja a MEF $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$, e seu conjunto de estados $Q = \{q_0, q_1, \dots, q_n\}$. O grafo $G = (V, E)$ é um **grafo distinção de estados** para M se $V = Q$ e $E = \{(q_i, q_j) \mid q_i, q_j \in Q \wedge \exists \alpha \in \Omega(q_i) \cap \Omega(q_j), \lambda(q_i, \alpha) \neq \lambda(q_j, \alpha)\}$, ou seja, os vértices do grafo são os estados da MEF e as arestas conectam todos os pares de estados **distinguíveis**. Dois estados q_i e q_j são distinguíveis quando para eles existe ao menos uma **seqüência de distinção**, que refere-se a uma seqüência $\alpha \in \Omega(q_i) \cap \Omega(q_j)$ de símbolos de entrada definida para ambos os estados que, quando aplicada, produza saídas incompatíveis em cada um deles. As notações $q_i \neq q_j$ e $q_i \sim q_j$ serão respectivamente utilizadas para indicar pares de estados distinguíveis e não distinguíveis. A Figura 4.1 contém uma MEF M e seu respectivo grafo de distinção G .

Figura 4.1: Uma MEF e seu respectivo grafo de distinção

O Algoritmo 1 contém o pseudo-código das operações implementadas no método para decidir quando existe uma seqüência de distinção entre um par de estados. Trata-se de um procedimento recursivo, onde os parâmetros q_i e q_j são, inicialmente, os estados a serem comparados. Para todos os símbolos de entrada $x \in \Sigma$, a partir dos quais existam transições definidas em ambos os estados ($\delta(q_i, x) \neq \emptyset \wedge \delta(q_j, x) \neq \emptyset$), compara-se as saídas $\lambda(q_i, x)$ e $\lambda(q_j, x)$ produzidas. Caso as saídas sejam diferentes, já se pode concluir que $q_i \neq q_j$, caso contrário, o procedimento é recursivamente solicitado para então comparar os estados atingidos pelas transições disparadas por x , $\delta(q_i, x)$ e $\delta(q_j, x)$.

Uma lista L com todos os pares de estados já comparados durante a chamada é mantida. Isto é necessário para casos em que a recursão leva à comparação repetida, devido à existência de laços na MEF. A ocorrência desses casos define que $q_i \sim q_j$, pois impossibilita a identificação de uma seqüência de distinção, uma vez que a comparação das saídas estaria atrelada a um laço no qual o critério de parada jamais seria atendido. A linha 3 garante que um par distinguível já analisado não o será novamente, verificando se $(q_i, q_j) \in V$.

Considerando a MEF da Figura 4.1 (a), a aplicação do procedimento *DistinguirEstados* a partir dos estados q_2 e q_8 dar-se-á da seguinte forma: primeiramente, os testes das linhas

2 e 3 do Algoritmo 1 devem resultar em valores falsos, pois as listas de arestas E e pares analisados L ainda estão vazias, uma vez que esta é a chamada inicial, não proveniente de recursão. Após a inclusão do par (q_2, q_8) na lista L , inicia-se a análise das transições a partir da única entrada comum, definida em ambos os estados, que é $x = b$.

As transições definidas com tal entrada produzem a mesma saída 0, partindo de q_2 ou q_8 , então $\lambda(q_2, x) = \lambda(q_8, x) = 0$, o que faz com que a comparação da linha 12 falhe. Até este ponto, não se pode distinguir os estados, portanto, os próximos estados atingidos pelas transições $\delta(q_2, x)$ e $\delta(q_8, x)$, respectivamente, q_1 e q_2 , serão analisados na próxima recursão, que inicia-se com a chamada da linha 16.

Após os testes das linhas 2 e 3, que devem falhar, dado que o par não foi previamente analisado, verifica-se que, também nesta segunda recursão, há somente uma entrada comum, $x' = a$. A entrada gera saídas distintas a partir de cada estado: $\lambda(q_1, x') = 1$ e $\lambda(q_2, x') = 0$. Esta diferença revela que os estados q_1 e q_2 são distinguíveis, portanto, a aresta correspondente é adicionada ao grafo G , de acordo com a linha 13. O valor retornado pela chamada recursiva, definido na linha 18, é verdadeiro, o que faz com que a chamada anterior também defina os estados q_2 e q_8 como distinguíveis e, da mesma maneira, adicione a aresta correspondente ao grafo G utilizando a operação da linha 17.

A montagem do grafo de distinção é realizada pelo Algoritmo 2, que, a partir de laços sequenciais, aplica a rotina *DistinguirEstados* em cada par de estados $(q_i, q_j) \in Q$ da MEF M em questão. Uma vez que a operação da linha 17 do Algoritmo 1 marca como distinguíveis todos os pares de estados analisados nas recursões que precedem a identificação de um par distinguível, os laços das linhas 3 e 4 ficam sujeitos a aplicar a rotina de distinção de estados em pares que já estão conectados em G . Embora tal condição seja verificada no início do procedimento *DistinguirEstados*, adicionou-se ainda o teste da linha 5 do Algoritmo 2, que verifica a existência das arestas, evitando que o trabalho redundante seja iniciado.

A aplicação do procedimento *MontarGrafoDeDistincao*(M, G) resulta no grafo de distinção G para a MEF M , respectivamente representados na Figura 4.1(b) e Figura 4.1(a).

4.3 Seleção de Estados Compatíveis

Um grafo G é **completo** se todos os seus vértices são adjacentes entre si, ou seja, para todo $q_i, q_j \in V$, $q_i \neq q_j$ implica que $(q_i, q_j) \in E$. Um grafo de distinção completo sempre estará relacionado a uma MEF mínima, uma vez que todos os estados de tal MEF devem ser distinguíveis entre si. Dado um subconjunto $S \subseteq V$, $G(S) = (S, E \cap S \times S)$ é o **subgrafo** induzido por S . O subconjunto S é considerado um **clique** se o grafo induzido $G(S)$ é completo. O número de vértices no subconjunto S é a **cardinalidade** do clique, para a qual se utiliza a notação $|S|$.

Algoritmo 1 Análise da Distinção de Estados

Requer: a MEF $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$, um par de estados $q_i, q_j \in Q$, o grafo $G = (V, E)$ onde $V = Q$, e uma lista L para armazenar pares já analisados

Provê: retorna *true* quando é possível distinguir q_i e q_j , *false* caso contrário

```

1: function DISTINGUIREESTADOS( $M, G, L, q_i, q_j$ )
2:   if  $(q_i, q_j) \in L$  then
3:     if  $(q_i, q_j) \in E$  then
4:       return true
5:     else
6:       return false
7:     end if
8:   else
9:      $L \leftarrow L \cup \{(q_i, q_j)\}$ 
10:  end if
11:  for all  $x \in \Sigma, \delta(q_i, x) \neq \phi \wedge \delta(q_j, x) \neq \phi$  do
12:    if  $\lambda(q_i, x) \neq \lambda(q_j, x)$  then
13:       $E \leftarrow E \cup \{(\delta(q_i, x), \delta(q_j, x))\}$ 
14:      return true
15:    else
16:      if DistinguirEstados( $M, G, L, \delta(q_i, x), \delta(q_j, x)$ ) then
17:         $E \leftarrow E \cup \{(\delta(q_i, x), \delta(q_j, x))\}$ 
18:        return true
19:      else
20:        return false
21:      end if
22:    end if
23:  end for
24: end function

```

Algoritmo 2 Montagem do Grafo de Distinção

Requer: a MEF $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$ e o grafo $G = (V, E)$, com $V = Q$ e $E = \emptyset$

Provê: $E = \{(q_i, q_j) \mid q_i, q_j \in Q \wedge \exists \alpha \in \Sigma^*, \lambda(q_i, \alpha) \neq \lambda(q_j, \alpha)\}$

```

1: procedure MONTARGRAFODEDISTINCAO( $M, G$ )
2:    $L \leftarrow \emptyset$ 
3:   for  $i \leftarrow 0, n$  do
4:     for  $j \leftarrow i, n$  do
5:       if  $(q_i, q_j) \notin E$  then
6:         DistinguirEstados( $M, G, L, q_i, q_j$ )
7:       end if
8:     end for
9:   end for
10: end procedure

```

Cliques em um grafo de distinção apontam subconjuntos de estados da MEF que são distinguíveis entre si. Seja $n = |S|$ a cardinalidade de um clique S em um grafo de distinção G de uma MEF M . A existência de tal clique permite afirmar que a forma mínima da MEF M utilizará, ao menos, n estados. Dessa maneira, estimar o número mínimo de estados necessários para reproduzir o comportamento de M , sem estados redundantes, é questão de se conhecer a cardinalidade do maior clique presente no grafo G .

O uso de cliques maximais para estimar o número mínimo de estados já foi explorado por Biermann e Feldman (1972) em seu algoritmo para a redução de MEFs em árvore, o mesmo que está contido nos procedimentos do método **bica**, que foi apresentado na Subseção 3.2.1.

A abordagem que é aqui proposta não utiliza o grafo de distinção e seus cliques maximais para estimar o número de estados. As informações contidas nessas estruturas são utilizadas na seleção de estados para compor classes de compatibilidade. É importante notar que a busca por cliques maximais é um problema NP-Hard. No entanto, o uso de heurísticas e algoritmos híbridos têm sido aplicado com sucesso na solução de problemas reais (Bomze et al., 1999). Há na literatura uma grande quantidade de trabalhos com propostas de métodos para a busca desses cliques (Battiti e Protasi, 2001; Gendreau et al., 1993; Östergård, 2002; Pardalos e Rodgers, 1992) e, por essa razão, a questão não será abordada com maiores detalhes no presente trabalho. Qualquer abordagem válida pode ser utilizada como parte da estratégia para minimização de MEFs parciais que é aqui descrita. Para integrar a implementação computacional que resultou da elaboração do método **cosme**, o trabalho de Östergård (2002) foi utilizado, pois está disponível uma implementação funcional da solução proposta.

A abordagem clássica para minimização de MEFs parciais faz uso de busca exaustiva sobre os possíveis conjuntos de classes de compatibilidade, procurando por aqueles de cardinalidade mínima que obedeçam às restrições discutidas na Seção 3.2. No método que é aqui proposto, a seleção dos estados para as classes de compatibilidade é realizada por meio de uma heurística que explora as informações contidas em grafos de distinção e seus respectivos cliques maximais.

Primeiramente, são introduzidas algumas definições. Define-se o conjunto de vizinhança do vértice q_a como $N_G(q_a) = \{q \in V \mid (q, q_a) \in E\}$, ou seja, todos os vértices adjacentes ao vértice q_a em G . Seja $K_m(S)$ o conjunto de todos os subconjuntos de S com cardinalidade m , onde $1 \leq m \leq n$, $m \in \mathbb{Z}$ e $n = |S|$. Para todo $k \in K_m(S)$, define-se o conjunto de estados potencialmente compatíveis $C_k = \{q \in V \mid N_G(q) \cap S = S \setminus k\}$, ou seja, vértices de V os quais são adjacentes aos vértices em S , com exceção dos que pertencem ao subconjunto $k \subset S$.

A seleção de estados tem por parâmetro o valor de m , referido como a **profundidade** do procedimento. Partindo de $m = 1$, quando cada $k_i \in K_1(S)$ corresponde à um subconjunto que contém exclusivamente o vértice $q_i \in S$, todas as vezes que a seleção resultar

em conjuntos C_k vazios, m é incrementado em passos de 1, aumentando a abrangência da busca, até o valor máximo $m = n$. Os subconjuntos em $K_m(S)$ são m-combinações dos elementos de S , portanto, no valor máximo, $K_n(S) = S$.

O Algoritmo 3 contém a representação do procedimento utilizado na implementação para a construção dos conjuntos C_k . Sua aplicação sobre o grafo da Figura 4.1(b), considerando o clique $S = \{q_0, q_1, q_2\}$, com profundidade inicial $m = 1$, cujo conjunto $K_1(S) = \{\{q_0\}, \{q_1\}, \{q_2\}\}$, dar-se-ia como segue. O primeiro conjunto a ser utilizado no laço iniciado na linha 2 seria, pela ordem, o conjunto $k = \{q_0\}$. O laço da linha 4 deverá iterar seqüencialmente pelos vértices q_3, q_4, q_5, q_6, q_7 e q_8 , que são os vértices restantes no grafo quando se desconsidera aqueles que pertencem à S . Em cada iteração, será definido um conjunto com a intersecção da vizinhança de q com S . Quando este conjunto contiver os mesmos elementos do clique, com exceção do vértice que pertence ao conjunto k , q é adicionado ao conjunto C_k . O resumo das iterações é apresentado na Tabela 4.1. Nela, pode-se observar que as condições são atendidas para $k = \{q_1\}$, nos vértices q_6 e q_8 , resultando no conjunto $C_{\{q_1\}} = \{q_6, q_8\}$, e para $k = \{q_2\}$, no vértice q_7 , resultando no conjunto $C_{\{q_2\}} = \{q_7\}$.

Tabela 4.1: Iterações do Algoritmo 3 para o grafo de distinção da Figura 4.1(b)

Vértice q	Conjunto $N_G(q) \cap S$	$k = \{q_0\}$		$k = \{q_1\}$		$k = \{q_2\}$	
		$S \setminus k$	C_k	$S \setminus k$	C_k	$S \setminus k$	C_k
q_3	$\{q_1\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{\emptyset\}$	$\{q_0, q_1\}$	$\{\emptyset\}$
q_4	$\{\emptyset\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{\emptyset\}$	$\{q_0, q_1\}$	$\{\emptyset\}$
q_5	$\{q_1\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{\emptyset\}$	$\{q_0, q_1\}$	$\{\emptyset\}$
q_6	$\{q_0, q_2\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{q_6\}$	$\{q_0, q_1\}$	$\{\emptyset\}$
q_7	$\{q_0, q_1\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{q_6\}$	$\{q_0, q_1\}$	$\{q_7\}$
q_8	$\{q_0, q_2\}$	$\{q_1, q_2\}$	$\{\emptyset\}$	$\{q_0, q_2\}$	$\{q_6, q_8\}$	$\{q_0, q_1\}$	$\{q_7\}$

Algoritmo 3 Busca por Estados Compatíveis

Requer: o grafo de distinção $G = (V, E)$, um clique maximal S em G e a profundidade da busca m

Provê: os conjuntos $C_k = \{q \in V \mid N_G(q) \cap S = S \setminus k\}$

```

1: procedure BUSCARESTADOSCOMPATÍVEIS( $G, S, m$ )
2:   for all  $k \in K_m(S)$  do
3:      $C_k \leftarrow \emptyset$ 
4:     for all  $q \in V \setminus S$  do
5:       if  $N_G(q) \cap S = S \setminus k$  then
6:          $C_k \leftarrow C_k \cup \{q\}$ 
7:       end if
8:     end for
9:   end for
10:  return  $C$ ;
11: end procedure

```

É a partir dos conjuntos de estados potencialmente compatíveis que serão construídas as classes de compatibilidade, sobre as quais se realiza a minimização. Tais classes são definidas com base em classes de equivalência, que, por sua vez, são resultantes da operação de fusão de estados, apresentada na próxima seção.

4.4 Fusão de Estados Compatíveis

Dada uma MEF $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$ e um par de estados $q_i, q_j \in Q, q_i \sim q_j$, seja $J = \{(q_a, q_b) \in Q \times Q \mid \exists \alpha \in \Omega(q_i) \cap \Omega(q_j), \delta(q_i, \alpha) = q_a \wedge \delta(q_j, \alpha) = q_b \wedge q_a \neq q_b\}$, ou seja, o conjunto de todos os pares de estados alcançados a partir das seqüências de entrada definidas para ambos q_i e q_j . Seja R uma relação de equivalência sobre Q de forma que $[q_a] = \{q_b \in Q \mid (q_a, q_b) \in J\}$.

Define-se: **fundir** o par (q_i, q_j) significa produzir uma nova MEF $M' = (\Sigma, \Delta, Q', q'_0, \delta', \lambda')$, quasi-equivalente à M , onde o comportamento de entrada/saída de todos os estados em cada classe de equivalência é definida por um único estado. Uma classe de equivalência $[q_a] \in Q / \sim$ tornar-se-á um estado $q'_a \in Q'$. As transições de q'_a são definidas como segue: $\forall x \in \Sigma, \exists q_m \in [q_a], \exists q_n \in [q_b], \delta(q_m, x) = q_n \Rightarrow \delta'(q'_a, x) = q'_b$. A operação como um todo é denotada como $M' = M/(q_i, q_j)$.

A quasi-equivalência de M' em relação à M é garantida se as classes da relação de equivalência constituem uma cobertura fechada sobre o conjunto Q , o qual contém os estados da MEF original M . Como já discutido na estratégia clássica para a minimização de MEFs, descrita na Seção 3.2, pode-se validar tal condição quando duas restrições são observadas:

- (i) cada estado do conjunto Q está em uma, e apenas uma, classe de equivalência;
- (ii) dada uma classe de equivalência $[q_a]$ e um símbolo de entrada $x \in \Sigma$, para todo $q_b \in [q_a], \delta(q_b, x) = q, q \in [q_c]$, ou seja, quaisquer transições definidas para um determinado símbolo de entrada, a partir de quaisquer estados em uma mesma classe de equivalência $[q_a]$, devem atingir estados pertencentes a uma única classe de equivalência $[q_c]$.

As restrições são as mesmas definidas no trabalho de Paull e Unger (1959) para as classes de compatibilidade. Quando aplicadas em relações de equivalência, apenas a restrição (ii) requer verificação, uma vez que a restrição (i) já é parte da própria definição das relações. Uma operação de fusão é dita ser **bem-sucedida** quando a restrição (ii) é observada nas classes de equivalência nela geradas.

No Algoritmo 4 é descrita a maneira utilizada para se construir J e determinar as classes da relação de equivalência sobre Q . Considerando uma MEF M e um par de estados

Algoritmo 4 Obtenção das Classes de Equivalência

Requer: a MEF $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$, um par de estados $q_i, q_j \in Q, q_i \sim q_j$, selecionados para fusão

Provê: o conjunto J e os conjuntos associados J_{q_a} , com todos os pares de estados que devem ser fundidos em função de $M/(q_i, q_j)$

```

1: function OBTTERCLASSESEQUIVALENCIA( $M, q_i, q_j$ )
2:    $P \leftarrow PUSH(\{(q_i, q_j)\})$ 
3:   while  $P \neq \{\emptyset\}$  do
4:      $(q_a, q_b) \leftarrow POP(P)$ 
5:      $J \leftarrow J \cup \{(q_a, q_b)\}$ 
6:     for all  $x \in \Omega(q_a) \cap \Omega(q_b)$  do
7:        $q'_a \leftarrow \delta(q_a, x)$ 
8:        $q'_b \leftarrow \delta(q_b, x)$ 
9:       if  $(q'_a, q'_b) \notin J$  then
10:         $P \leftarrow PUSH(\{(q'_a, q'_b)\})$ 
11:       end if
12:     end for
13:   end while
14:   for all  $(q_a) \in Q$  do
15:      $J_{q_a} = \{q_a\}$ 
16:   end for
17:   for all  $(q_a, q_b) \in J$  do
18:      $UNION(J_{q_a}, J_{q_b})$ 
19:   end for
20:   return  $J$ 
21: end function

```

(q_i, q_j) , a partir dos quais se deseja efetuar a operação de fusão $M/(q_i, q_j)$, a função *ObterClassesEquivalencia* utiliza a pilha auxiliar P para iterar sobre os pares de estados que também devem ser fundidos em consequência do par inicial (q_i, q_j) . O primeiro elemento inserido na pilha é o par inicial, que é em seguida transferido para o conjunto J . Os pares alcançáveis por um mesmo símbolo de entrada a partir de q_i e de q_j são então colocados na pilha. Cada par da pilha será submetido às mesmas operações: é adicionado em J e seus estados subseqüentes são empilhados. O conjunto J estará completo quando P estiver novamente vazia, o que indicará que todos os pares a serem fundidos em consequência da operação $M/(q_i, q_j)$ foram considerados.

Construído o conjunto J , para todo $q_a \in Q$, constrói-se o conjunto associado J_{q_a} , que conterà os elementos da classe de equivalência $[q_a]$. A princípio, cada conjunto J_{q_a} é inicializado com o próprio elemento q_a . Em seguida, para cada par (q_a, q_b) em J , realiza-se a operação de união dos conjuntos J_{q_a} e J_{q_b} . Tal operação de união, utilizada na linha 18 do Algoritmo 4, é definida segundo os moldes da estratégia *Union-Find* (La Poutré, 1990), que contempla as propriedades exigidas na definição de relações de equivalência. Os resultados, o conjunto J e os conjuntos J_q associados, são retornados.

Para exemplificar o procedimento, será considerada a sua aplicação sobre a MEF da Figura 4.1(a), utilizando o par (q_0, q_3) . Primeiramente, o par é empilhado na pilha P , que estava inicialmente vazia. O laço da linha 3 é iniciado, e deve repetir-se até que P esteja novamente vazia. Após desempilhar o par (q_0, q_3) e adicioná-lo ao conjunto J , o segundo laço, na linha 6, deve iterar por todos os símbolos de entrada para os quais existam transações definidas a partir de ambos os estados do par. Os pares de estados atingidos por essas transações, caso ainda não estejam em P , são empilhados, como define a linha 10. Para (q_0, q_3) , os próximos estados atingidos com o único símbolo de entrada comum, $x = a$, são, respectivamente, q_1 e q_4 , portanto, o próximo par empilhado é (q_1, q_4) . Este par será desempilhado para a próxima iteração e será processado de maneira semelhante. O conteúdo das variáveis nas iterações restantes é listado na Tabela 4.2.

Tabela 4.2: Iterações do Algoritmo 4 para a MEF da Figura 4.1(a)

q_a	q_b	J	P
q_0	q_3	$J = \{(q_0, q_3)\}$	$P = \{(q_1, q_4)\}$
q_1	q_4	$J = \{(q_0, q_3), (q_1, q_4)\}$	$P = \{(q_2, q_5)\}$
q_2	q_5	$J = \{(q_0, q_3), (q_1, q_4), (q_2, q_5)\}$	$P = \{(q_1, q_6)\}$
q_1	q_6	$J = \{(q_0, q_3), (q_1, q_4), (q_2, q_5), (q_1, q_6)\}$	$P = \{\emptyset\}$

Após a construção do conjunto J , os dois laços finais do Algoritmo 4 organizarão os conjuntos J_q . O primeiro laço, da linha 14, inicializa cada conjunto J_{q_a} com o próprio estado q_a , resultando em: $J_{q_0} = \{q_0\}$, $J_{q_1} = \{q_1\}$, $J_{q_2} = \{q_2\}$, $J_{q_3} = \{q_3\}$, $J_{q_4} = \{q_4\}$, $J_{q_5} = \{q_5\}$, $J_{q_6} = \{q_6\}$, $J_{q_7} = \{q_7\}$, $J_{q_8} = \{q_8\}$ e $J_{q_9} = \{q_9\}$. A seguir, os conjuntos são unidos pelo laço da linha 17. Segundo a estratégia do algoritmo Union-Find (La Poutré, 1990), a união

de $J_{q_1} = \{q_1\}$ com $J_{q_4} = \{q_4\}$, motivada por $(q_1, q_4) \in J$, resulta em $J_{q_1} = \{q_1, q_4\}$, sendo q_1 o menor índice. De maneira semelhante, uma segunda fusão, agora com $J_{q_1} = \{q_1, q_4\}$ e $J_{q_6} = \{q_6\}$, motivada por $(q_1, q_6) \in J$, resulta em $J_{q_1} = \{q_1, q_4, q_6\}$. Os conjuntos gerados são: $J_{q_0} = \{q_0, q_3\}$, $J_{q_1} = \{q_1, q_4, q_6\}$, $J_{q_2} = \{q_2, q_5\}$, $J_{q_7} = \{q_7\}$ e $J_{q_8} = \{q_8\}$.

Uma vez que estão disponíveis os conjuntos J_q , deve-se então utilizar a rotina descrita no Algoritmo 5 para checar a restrição (ii) sobre os mesmos. O algoritmo realiza uma iteração sobre todos os estados de cada conjunto J_{q_a} verificando se, para cada um dos símbolos de entrada definidos, todas as transições atingem estados que pertencem a um único conjunto J_{q_b} , tal como é definida a restrição.

Algoritmo 5 Verificação da Restrição (ii) Sobre as Classes de Equivalência

Requer: os conjuntos associados J_q e a MEF M a partir da qual foram construídos

Provê: verdadeiro, quando a restrição é respeitada, ou falso, caso contrário.

```

1: function CHECARRESTRICAO( $J, M$ )
2:   for all  $q_a \in Q$  do
3:     if  $|J_{q_a}| > 1$  then
4:       for all  $x \in \Sigma$  do
5:          $q_b \leftarrow \phi$ 
6:         for all  $q_i \in J_{q_a}$  do
7:            $q \leftarrow \delta(q_i, x)$ 
8:           if  $q \neq \phi$  then
9:             if  $q_b = \phi$  then
10:              for all  $q_b \in Q$  do
11:                if  $q \in J_{q_b}$  then
12:                  break
13:                end if
14:              end for
15:            else
16:              if  $q \notin J_{q_b}$  then
17:                return false
18:              end if
19:            end if
20:          end if
21:        end for
22:      end for
23:    end if
24:  end for
25:  return true
26: end function

```

Considerando os conjuntos $J_{q_0} = \{q_0, q_3\}$, $J_{q_1} = \{q_1, q_4, q_6\}$, $J_{q_2} = \{q_2, q_5\}$, $J_{q_7} = \{q_7\}$ e $J_{q_8} = \{q_8\}$, construídos com o procedimento do Algoritmo 4 para o par de estados (q_0, q_3) da MEF da Figura 4.1(a), a aplicação da rotina *ChecarRestricao* aconteceria da seguinte forma: o laço da linha 2 iterará sobre todos os conjuntos J_{q_a} , porém, a restrição da linha 3 garante que somente aqueles que contém mais do que um elemento são analisados, isto é, J_{q_0} , J_{q_1} e J_{q_2} . Pela ordem, a análise é iniciada com J_{q_0} . O laço da linha 4 vai iterar por

todos os símbolos x do conjunto de entrada Σ . Para cada x , o laço da linha 6 iterará por todos os estados q_i do conjunto J_{q_0} .

Com o primeiro valor de $x = a$ e o primeiro estado $q_i = q_0$, a operação da linha 7 associa o próximo estado atingido pela transição $\delta(q_0, x)$ à variável temporária q . Tal transição está definida, logo $q = q_1$ de forma que o teste da linha 8 permita que a iteração prossiga. Caso a transição não estivesse definida, o teste falharia e a iteração seria desprezada, pois tais casos são irrelevantes para a finalidade do algoritmo. A continuação dar-se-á no laço da linha 10, pois a variável q_b , ainda não foi inicializada, validando o teste da linha 9. Este laço iterará por todos os estados $q_b \in Q$ até que um conjunto J_{q_b} , que contenha o estado alvo q_1 , seja alcançado. Na iteração em discussão, o conjunto em questão é J_{q_1} . No momento em que este for analisado, o teste da linha 11 será validado e a iteração do laço da linha 10 será interrompida. A variável q_b passa então a conter o estado q_1 que, coincidentemente, é o índice do conjunto que contém o estado $q_1 = \delta(q_0, a)$.

A próxima iteração do laço da linha 6 é iniciada, associando o próximo estado $q_3 \in J_{q_a}$ à q_i . Trata-se ainda da mesma iteração sobre o símbolo $x = a$, portanto $q_b = q_1$. O próximo estado atingido pela transição $\delta(q_3, a)$ é q_4 , logo, na linha 7 tem-se $q = q_4$. Estando q_b definida, o teste da linha 9 é invalidado. Para terminar o procedimento com a conclusão de que a restrição foi violada, é necessário verificar na operação da linha 16 que $q_4 \notin J_{q_1}$. Como tal condição é falsa, a próxima iteração é iniciada. Não havendo mais estados em J_{q_0} o laço da linha 6 termina. O próximo símbolo de entrada $x = b$ é associado pelo laço da linha 4 e é reiniciado o laço da linha 6. Como não há transições definidas com o símbolo de entrada b em nenhum dos estados de J_{q_0} , ambas iterações serão desprezadas no teste da linha 8. Não havendo mais símbolos de entrada, o laço da linha 4 termina, e a próxima iteração do laço da linha 2 é iniciada. O conjunto J_{q_1} será validado nesta nova iteração, seguindo os mesmos procedimentos descritos na iteração anterior.

Após a análise da restrição (ii) sobre os conjuntos J_q , pode-se concluir se as classes de equivalência, resultantes da fusão de um par de estados, são adequadas para a organização de uma nova MEF, onde o comportamento de todos os estados de um conjunto J_q é representado por um único estado q . Tal conclusão garante que a nova MEF será quasi-equivalente à MEF original. O procedimento que faz uso dos algoritmos apresentados para minimizar MEFs parciais é apresentado na seção seguinte.

4.5 Procedimento para Minimização

Seja o conjunto $P = \{(q_i, q_j) \in k \times C_k \mid k \in K_m(S)\}$, ou seja, o conjunto que contém todos os pares selecionados com o uso da estratégia apresentada na Seção 4.3. O potencial desses pares em gerar classes de equivalência de forma a não violar a restrição (ii) reduz à medida em que a profundidade m é incrementada.

Particularmente, quando $m = 1$, dado $k = \{q_a\}$, para todos os estados $q \in C_k$, a operação $M' = M/(q_a, q)$ é sempre bem-sucedida. Em tais condições, diz-se que a seleção de estados é realizada em **modo exato**, uma vez que todo estado q selecionado para a fusão com q_a é adjacente a todos os outros vértices do clique, não o sendo somente ao vértice q_a . Conseqüentemente, q é também incompatível com todos os estados, exceto q_a . Quando $m > 1$, a restrição (ii) pode ser violada, pois, nessas condições, não é possível distinguir de q dois ou mais estados do clique. Nesses casos, tal restrição deve ser previamente verificada.

No modo aproximado, há também a necessidade de se garantir o determinismo das MEFs reduzidas. Quando m está próximo do valor da cardinalidade do clique, pares são selecionados quase que arbitrariamente e, em casos muito específicos, dois estados com transições que produzem saídas diferentes para um mesmo símbolo de entrada podem acabar uma classe de equivalência comum. Uma MEF resultante de uma fusão que, embora bem-sucedida, seja dada nessas condições, passaria a ser não-determinística. Para evitar tal consequência, é suficiente verificar se nenhum dos estados em uma determinada classe é distinguível a outro desta mesma classe. Utilizando o grafo de distinção, a verificação pode ser facilmente realizada buscando-se pela existência das arestas correspondentes e, caso alguma seja encontrada, não efetuar a fusão. A verificação descrita é apresentada de maneira programática por meio do Algoritmo 6. A aplicação do procedimento *ValidarCompatibilidade* sobre o conjunto J , como construído para exemplificar o Algoritmo 4 da Seção 4.4, consiste na iteração por todos os pares de cada conjunto J_{q_a} , por exemplo, $J_{q_1} = \{q_1, q_4, q_6\}$, que resulta nos pares: (q_1, q_4) , (q_1, q_6) e (q_4, q_6) . A cada iteração, é verificada a pertinência do par em questão no conjunto de arestas E do grafo e, caso seja verdadeira, o procedimento é interrompido, retornando o valor falso. Se nenhum dos pares pertencer ao conjunto de arestas, o que é o caso dos três pares do exemplo acima, os laços terminarão e o algoritmo alcançará a linha 11, onde o valor verdadeiro é retornado.

Com base nas informações até aqui apresentadas e discutidas, define-se um procedimento iterativo de minimização, onde após cada iteração têm-se uma MEF M' , quasi-equivalente à MEF original M , com um número menor de estados. As iterações encerram-se quando o grafo de distinção de M' é completo ou quando a busca por estados compatíveis não gera mais resultados. Este procedimento está representado no Algoritmo 7.

Quando em modo exato, todos os pares definidos pelos conjuntos C_k são utilizados em operações de fusão dentro da mesma iteração. No entanto, quando em modo aproximado, apenas a primeira fusão bem-sucedida é realizada, sendo logo após iniciada a próxima iteração.

Uma chamada à função *BuscarProximoCliqueMaximal*, que não está descrita nos algoritmos anteriormente apresentados, foi utilizada. Esta função é considerada como parte da implementação do trabalho de Östergård (2002), através da qual são obtidos cliques na

Algoritmo 6 Verificação da Compatibilidade dos Estados nas Classes de Equivalência**Requer:** os conjuntos associados J_q e o grafo de distinção $G = (V, E)$ **Provê:** verdadeiro, quando a não há estados distinguíveis, ou falso, caso contrário.

```

1: function VALIDARCOMPATIBILIDADE( $J, G$ )
2:   for all  $q_a \in Q$  do
3:     for all  $q_i \in J_{q_a}$  do
4:       for all  $q_j \in J_{q_a}$  do
5:         if  $(q_i, q_j) \in E$  then
6:           return false
7:         end if
8:       end for
9:     end for
10:  end for
11:  return true
12: end function

```

ordem em que são no grafo encontrados. O primeiro parâmetro é o grafo a ser analisado, o segundo é o último clique encontrado. Quando este segundo parâmetro não é definido, a busca visa o primeiro clique. A ordem dos vértices, que implica na ordem dos cliques encontrados, pode ser definida sob algum critério específico. No entanto, essa ordem não é relevante para a presente aplicação.

Para exemplificar detalhadamente o funcionamento do Algoritmo 7, será utilizada a MEF M que foi apresentados na Figura 4.1(a) da Seção 4.2. Após copiar a MEF M em M' , monta-se seu grafo de distinção, tal como está representado na Figura 4.1(b). As duas operações anteriores são realizadas, respectivamente, pelas linhas 2 e 3. O laço principal, da linha 4, realizará iterações até que o grafo que está em G seja completo, o que não acontece com o grafo associado à M' . Dentro do laço principal, é iniciado o laço de busca por estados compatíveis em modo exato, ou seja, com $m = 1$. Considerando a ordem dos vértices, o primeiro clique maximal a ser encontrado pela linha 7 será $S = \{q_0, q_1, q_2\}$. A busca por estados compatíveis com $m = 1$, realizada na linha 8, retornará os conjuntos $C_{\{q_0\}} = \{\emptyset\}$, $C_{\{q_1\}} = \{q_6, q_8\}$ and $C_{\{q_2\}} = \{q_7\}$. Como os conjuntos $C_{\{q_1\}}$ e $C_{\{q_2\}}$ não estão vazios, a busca em modo exato foi bem sucedida e seu laço é interrompido, segundo a condição da linha 9. Não será necessário realizar seleção em modo aproximado, logo, o teste da linha 10 fará com que esta etapa seja ignorada.

O procedimento continua na linha 25, onde o conjunto P é inicializado como vazio. Este conjunto conterá todos os pares de estados que podem ser fundidos e é organizado no laço seguinte, da linha 26, que combina em pares os elementos dos conjuntos C_k com os elementos de cada conjunto k . O resultado para C_{q_1} é: (q_1, q_6) e (q_1, q_8) ; para C_{q_2} : (q_2, q_7) . Logo, $P = \{(q_1, q_6), (q_1, q_8), (q_2, q_7)\}$.

O laço da linha 33 vai iterar por todos os pares em P , e, para cada um deles, gerar o conjunto J e as classes de compatibilidade J_q . A chamada para o procedimento responsável por essas operações está na linha 34. Em seguida, o comportamento é dependente do valor

Algoritmo 7 Processo de Minimização**Requer:** uma MEF M a qual se deseja minimizar**Provê:** uma MEF M' , quasi-equivalente a M , com menor número de estados

```

1: function MINIMIZAR( $M$ )
2:    $M' \leftarrow M$ 
3:    $G \leftarrow \text{MontarGrafoDistincao}(M')$ 
4:   while not Completo( $G$ ) do
5:      $S \leftarrow \emptyset$ ; ▷ Iniciar busca em modo exato
6:     repeat
7:        $S \leftarrow \text{BuscarProximoCliqueMaximal}(G, S)$ 
8:        $C \leftarrow \text{BuscarEstadosCompativeis}(G, S, 1)$ 
9:     until  $S = \emptyset$  or ( $C_k \neq \emptyset, k \in K_1(S)$ )
10:    if ( $C_k = \emptyset, k \in K_1(S)$ ) then ▷ Sem estados compatíveis no modo exato
11:       $S \leftarrow \emptyset$ ; ▷ Iniciar busca em modo aproximado
12:      repeat
13:         $S \leftarrow \text{BuscarProximoCliqueMaximal}(G, S)$ 
14:         $m \leftarrow 2$ 
15:        repeat
16:           $C \leftarrow \text{BuscarEstadosCompativeis}(G, S, m)$ 
17:           $m \leftarrow m + 1$ 
18:        until  $m > |S|$  or ( $C_k \neq \emptyset, k \in K_m(S)$ )
19:         $m \leftarrow m - 1$ 
20:      until  $S = \emptyset$  or ( $C_k \neq \emptyset, k \in K_m(S)$ )
21:      if then ( $C_k = \emptyset, k \in K_m(S)$ ) ▷ Se ainda não encontrou estados compatíveis
22:        return  $M'$  ▷ Nada mais pode ser feito
23:      end if
24:    end if
25:     $P \leftarrow \emptyset$ 
26:    for all  $C_k, k \in K_m(S)$  do
27:      for all  $q_i \in k, q_j \in C_k$  do
28:        if  $(q_i, q_j) \notin P$  then
29:           $P \leftarrow P \cup \{(q_i, q_j)\}$ 
30:        end if
31:      end for
32:    end for
33:    for all  $(q_i, q_j) \in P$  do
34:       $J \leftarrow \text{ObterClassesEquivalencia}(M', q_i, q_j)$ 
35:      if  $m > 1$  and  $\text{ChecarRestricao}(J, M')$  and  $\text{ValidarCompatibilidade}(J, G)$  then
36:         $M' \leftarrow M' / (q_i, q_j)$ 
37:        break ▷ Modo aproximado, fundir apenas o primeiro par
38:      end if
39:       $M' \leftarrow M' / (q_i, q_j)$ 
40:    end for
41:     $G \leftarrow \text{MontarGrafoDistincao}(M')$ 
42:  end while
43:  return  $M'$ 
44: end function

```

de m : caso a iteração estivesse em modo aproximado, seriam realizadas as checagens de restrição e compatibilidade. Como este não é o caso, apenas é realizada a operação de fusão, como especificada na linha 39. Após o término do laço da linha 33, o grafo de distinção referente à MEF alterada é montado e o laço principal é retomado. Tal MEF e seu respectivo grafo são exibidos na Figura 4.2.

Figura 4.2: A MEF M' , após a primeira iteração, e seu respectivo grafo de distinção

A condição de saída do laço principal, que o grafo seja completo, não foi atendida, logo, mais uma iteração é iniciada. Esta segunda iteração ocorre de maneira idêntica à iteração anterior até a busca por estados compatíveis em modo exato, na linha 8. Com o único clique disponível $S = \{q_0, q_1, q_2\}$, tal busca resulta em todos os conjuntos $C_k = \emptyset$, $k \in K_1(S)$. Logo, o laço da linha 9 terminará sem estados compatíveis, o que tornará verdadeiro o resultado do teste da linha 10, fazendo iniciar a seleção de estados em modo aproximado.

A busca por cliques é reiniciada tornando S um conjunto vazio e, em seguida, efetuando uma chamada para a busca pelo próximo clique maximal, conforme as linhas 11 e 13. Após encontrar o primeiro clique, o valor da profundidade é alterado para $m = 2$ com a operação da linha 14. Inicia-se o laço para a seleção em modo aproximado, cujas iterações buscam por estados potencialmente compatíveis. A cada iteração, m também é incrementado. O laço termina segundo os critérios da linha 18: quando a seleção gerou resultados ou quando m superou a cardinalidade do clique. Nos casos em que este segundo critério é verdadeiro, a minimização não pode continuar, e o teste da linha 21 fará com que o procedimento termine retornando a MEF original ou resultante da iteração anterior. Na minimização da MEF em questão, o clique $S = \{q_0, q_1, q_2\}$ deverá ser mais uma vez o primeiro retornado, e a chamada para seleção em modo aproximado da linha 16 deverá retornar os seguintes conjuntos C_k : $C_{\{q_0, q_1\}} = \{q_4\}$, $C_{\{q_0, q_2\}} = \{q_3, q_5\}$ e $C_{\{q_1, q_2\}} = \{\emptyset\}$.

Uma vez que há estados selecionados, o laço de seleção termina e o procedimento continua com a organização do conjunto $P = \{(q_0, q_3), (q_0, q_4), (q_0, q_5), (q_1, q_4), (q_2, q_3), (q_2, q_5)\}$.

A primeira iteração do laço da linha 33 deverá utilizar o par (q_0, q_3) , segundo a ordem do conjunto P . A chamada para a obtenção das classes de compatibilidade resultará em $J = \{(q_0, q_3), (q_1, q_4), (q_2, q_5)\}$, com $J_{q_0} = \{q_0, q_3\}$, $J_{q_1} = \{q_1, q_4\}$ e $J_{q_2} = \{q_2, q_5\}$. Como $m = 2$, todos os demais testes da linha 35 também devem ser validados, o que implica em verificar a restrição (ii) e a compatibilidade. Após a validação, é realizada a operação de fusão. Uma vez que se está em modo aproximado, espera-se apenas uma fusão bem sucedida dentre todos os pares do conjunto P , como este é o caso para o par atual, o laço de fusões é interrompido na linha 37, e a iteração do laço principal é retomada.

Figura 4.3: MEF resultante da segunda iteração e seu respectivo grafo de distinção

A próxima operação é a construção de um novo grafo de distinção para a MEF resultante da fusão. Como é possível observar na Figura 4.3(b), o novo grafo é completo. Isso interromperá o laço principal e fará com que o procedimento retorne a MEF resultante, como exibida na Figura 4.3(a). Na seção seguinte são apresentados mais exemplos de minimização, porém com maior enfoque no método, abstraindo detalhes do algoritmo.

4.6 Aplicação do Método

Os exemplos que compõem esta seção foram selecionados de forma a exibir o comportamento do método em ambos os modos previstos no desenvolvimento. Os aspectos de cada caso são discutidos juntamente com a apresentação. Para cada exemplo, são exibidos, além da MEF inicial e sua versão reduzida, os grafos de distinção, MEFs e cliques gerados/utilizados em cada iteração.

4.6.1 Modo Exato

No exemplo a seguir, o método HSI (Petrenko e v. Bochmann, 1995b) foi utilizado para a geração de seqüências de teste sobre a MEF da Figura 4.4. As seqüências geradas estão listadas na Tabela 4.3. Este método é capaz de produzir conjuntos de teste m-completos

e, uma vez que se conhece tal característica, a aplicação da estratégia para verificação de completude, apresentada na Subseção 2.7.1, deve confirmar o poder declarado de detecção de falhas.

Figura 4.4: MEF mínima para geração de seqüências de teste

Tabela 4.3: Seqüências de Entrada/Saída geradas pelo método HSI para a MEF da Figura 4.4

Entradas/Saídas			
0/1	0/0	0/0	0/0
0/1	0/0	0/0	1/1
0/1	0/0	1/1	0/0
0/1	0/0	1/1	1/0
0/1	1/0	0/0	
0/1	1/0	1/1	

As seqüências de teste são então organizadas em uma MEF em árvore, utilizando a estratégia descrita por Yao et al. (1994). Tal MEF $M = (\Sigma, \Delta, Q, q_s, \delta, \lambda)$, com $\Sigma = \Delta = \{0, 1\}$, $Q = \{q_0, q_1, \dots, q_{10}, q_{11}\}$ é exibida na Figura 4.5.

O método *cosme* é então aplicado sobre a MEF em árvore. O grafo de distinção resultante da análise inicial é exibido na Figura 4.6. As seqüências que possibilitaram distinguir os estados adjacentes do grafo são exibidas na Tabela 4.4. Os estados não distinguíveis foram omitidos por não representarem informação relevante.

Tabela 4.4: Seqüências de entradas para distinção dos pares de estados da MEF da Figura 4.5

	q_0	q_1	q_2	q_3	q_6	q_9
q_0		0	0	0	0	0
q_1	0		1	1		1
q_2	0	1			1	
q_3	0	1			1	
q_6	0		1	1		1
q_9	0	1			1	

Em continuidade ao procedimento, realiza-se a seleção de estados compatíveis, iniciada pela busca por cliques maximais no grafo de distinção. Para o grafo da Figura 4.6, os maiores cliques possíveis possuem tamanho $n = 3$, sendo eles: $S_0 = \{q_0, q_1, q_2\}$,

Figura 4.5: MEF em árvore construída sobre as seqüências do método HSI

$S_1 = \{q_0, q_1, q_3\}$, $S_2 = \{q_0, q_2, q_6\}$, $S_3 = \{q_0, q_1, q_9\}$, $S_4 = \{q_0, q_3, q_6\}$, $S_5 = \{q_0, q_6, q_9\}$. Utiliza-se um dos cliques, S , para a análise de compatibilidade, no caso, $S = S_0$, o primeiro clique encontrado quando considera-se um algoritmo de busca em ordem seqüencial. Os vértices que pertencem ao clique S estão destacados na Figura 4.6.

Figura 4.6: Grafo de distinção construído sobre a MEF em árvore

Utilizando o clique S , inicia-se a seleção dos estados compatíveis. Uma vez que o modo exato ($m = 1$) está em vigor, cada vértice $q_i \in S$ figura em um conjunto $k_i \in K_1(S)$. Os conjuntos de estados compatíveis são então selecionados: $C_{k_0} = \emptyset$, $C_{k_1} = \{q_6\}$, $C_{k_2} = \{q_3, q_9\}$, onde $k_0 = \{q_0\}$, $k_1 = \{q_1\}$ e $k_2 = \{q_2\}$. Pode-se nele observar que o vértice q_6 é adjacente a somente dois elementos do clique, q_0 e q_2 , logo, q_6 é redundante ao estado q_1 , o único vértice do clique ao qual q_9 não é adjacente. O mesmo ocorre com q_3 e q_9 em relação à q_2 .

A seguir, cada elemento dos conjuntos C_{k_i} é selecionado para fusão com o estado $q_i \in k_i$. Neste caso, $P = \{(q_1, q_6), (q_2, q_3), (q_2, q_9)\}$ são selecionados. A fusão do primeiro par resultará nas fusões dos pares do conjunto $J = \{(q_1, q_6), (q_2, q_7), (q_8, q_9)\}$, pois sendo $I = \Omega(q_1) \cap \Omega(q_6) = \{0, 1\}$, para cada elemento de I tem-se que $\delta(q_1, 0) = q_2$, $\delta(q_6, 0) = q_7$ e $\delta(q_1, 1) = q_9$, $\delta(q_6, 1) = q_8$.

A operação $M' = M/(q_1, q_6)$ contará com as seguintes classes de equivalência sobre $\mathcal{Q}$: $\{[q_0], [q_1, q_6], [q_2, q_7], [q_8, q_9], [q_3], [q_4], [q_5], [q_{10}], [q_{11}]\}$. Sabe-se que as classes respeitam a restrição (ii), pois a seleção do par (q_1, q_6) ocorreu no modo exato. As classes tornar-se-ão estados em M' , sendo q'_i o estado de menor índice da classe $[q_i]$. Os estados q'_1 , q'_2 e q'_3 passam a ser, respectivamente, os representantes das classes $[q_1, q_6]$, $[q_2, q_7]$ e $[q_8, q_9]$. A MEF resultante é exibida na Figura 4.7.

Figura 4.7: MEF M' , resultante da fusão de q_1 e q_6

Uma vez que se está no modo exato, pode-se continuar a fusão dos pares selecionados. A nova operação deve ser realizada sobre a MEF M' , resultante da operação anterior. O segundo par selecionado em M é $(q_2, q_3) \in \mathcal{Q} \times \mathcal{Q}$, para fundi-los em M' , utiliza-se a relação de equivalência, definida na fusão anterior, para mapeá-los aos estados que os representam em $\mathcal{Q}'$, neste caso $(q'_2, q'_3) \in \mathcal{Q}' \times \mathcal{Q}'$.

Para a operação $M'' = M'/(q'_2, q'_3)$, constrói-se o conjunto $J = \{(q'_2, q'_3), (q'_3, q'_4), (q'_1, q'_5)\}$ pois, $\delta'(q'_2, 0) = q'_3$, $\delta'(q'_3, 0) = q'_4$ e $\delta'(q'_2, 1) = q'_1$, $\delta'(q'_3, 1) = q'_5$. São definidas as seguintes classes de equivalência sobre $\mathcal{Q}'$: $\{[q'_0], [q'_1, q'_5], [q'_2, q'_3, q'_4], [q'_8], [q'_{10}], [q'_{11}]\}$. De maneira análoga à descrita para a construção de M' , constrói-se a MEF M'' , onde q'_1 e q'_2 representam, respectivamente, as classes $[q'_1, q'_5]$ e $[q'_2, q'_3, q'_4]$. A MEF é ilustrada na Figura 4.8

Por fim, o terceiro e último par selecionado é fundido. Os estados $(q_2, q_9) \in \mathcal{Q} \times \mathcal{Q}$, são mapeados em $(q'_2, q'_8) \in \mathcal{Q}' \times \mathcal{Q}'$ e $(q''_2, q''_8) \in \mathcal{Q}'' \times \mathcal{Q}''$, logo, última operação dessa iteração é $M''' = M''/(q'_2, q'_8)$. É construído o conjunto $J = \{(q'_2, q'_8), (q'_2, q'_1 0''), (q'_1, q'_1 1'')\}$, a partir do qual são definidas as classes de equivalência sobre $\mathcal{Q}''$: $\{[q''_0], [q''_1, q''_{11}], [q''_2, q''_8, q''_{10}]\}$. Em $\mathcal{Q}'''$, ficam os estados q''_0 , q''_1 e q''_2 , respectivamente representantes das classes $[q''_0]$, $[q''_1, q''_{11}]$ e $[q''_2, q''_8, q''_{10}]$. A MEF é ilustrada na Figura 4.9.

Figura 4.8: MEF M'' , resultante da fusão de q'_2 e q'_3

Figura 4.9: MEF M''' , resultante da fusão de q''_2 e q''_9

Após a fusão de todos os pares selecionados na primeira iteração, o primeiro passo para a segunda é a construção de um novo grafo de distinção sobre a MEF resultante. Tal grafo é exibido na Figura 4.10. Como se pode observar, o grafo é completo, ou seja, todos os três estados da MEF resultante são distinguíveis entre si e, portanto, M''' é a versão mínima da MEF em árvore M . Uma vez que a MEF mínima foi encontrada, não são necessárias mais iterações.

Figura 4.10: Grafo de distinção da MEF resultante M'''

Com mesmo número de estados e transições, mesmo conjunto de entradas e saídas, M''' e a MEF da Figura 4.4 mostram comportamentos idênticos, ou seja, são equivalentes. A MEF M''' foi construída a partir do conjunto de seqüências de teste geradas pelo método HSI para a MEF da Figura 4.4, o que, segundo Yao et al. (1994), dá suporte à conclusão que tal conjunto apresenta cobertura completa de falhas.

4.6.2 Modo Aproximado

A MEF $M = (\Sigma, \Delta, Q, q_s, \delta, \lambda)$, agora com $\Sigma = \{a, b, c\}$, $\Delta = \{0, 1\}$, $Q = \{q_0, q_1, \dots, q_9, q_{10}\}$, exibida na Figura 4.11, foi selecionada para este exemplo de aplicação do método por representar uma situação em que o modo aproximado deve ser utilizado na eleição de estados para fusão.

Figura 4.11: MEF M para redução

A análise de distinção dos estados de M resulta nas seqüências listadas nas Tabela 4.5. O grafo de distinção G , construído sobre a tabela, é exibido na Figura 4.12.

Figura 4.12: Grafo de distinção G para M na primeira iteração

Os dois cliques maximais que se encontram em G são: $S_1 = \{q_0, q_3, q_9\}$ e $S_2 = \{q_0, q_4, q_8\}$. A seleção de estados no modo exato resulta em $C_{k_0} = C_{k_3} = C_{k_9} = \emptyset$ para S_1 e $C_{k_0} = C_{k_4} = C_{k_8} = \emptyset$ para S_2 , considerando $k_i = \{q_i\}$ para todo $q_i \in Q$. Uma vez que não

Tabela 4.5: Sequências de entradas para distinção dos pares de estados da MEF da Figura 4.11

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
q_0				a	b				b	c
q_1										c
q_2				a						
q_3	a		a			a		a		c
q_4	b								bc	
q_5				a						
q_6										c
q_7				a						
q_8	b				bc					
q_9	c	c		c			c			

há estados compatíveis selecionados, o método deve prosseguir no modo aproximado, elegendo os pares com maiores chances de resultar em fusões bem-sucedidas. A busca é reiniciada a partir do primeiro clique, S_1 , com profundidade $m = 2$, sendo $K_2(S_1) = \{\{q_0, q_3\}, \{q_0, q_9\}, \{q_3, q_9\}\}$. Partindo com $k = \{q_0, q_3\}$, é necessário encontrar ao menos um vértice adjacente à $S \setminus k = \{q_9\}$, o qual deverá resultar em uma fusão bem-sucedida com um dos estados de k . Uma breve análise do grafo G permite concluir que os vértices q_1 e q_6 atendem a tal condição. Logo, os pares eleitos para tentativa de fusão são o produto cartesiano $\{q_1, q_6\} \times k = \{(q_1, q_0), (q_1, q_3), (q_6, q_0), (q_6, q_3)\}$.

As classes de equivalência geradas por fusões entre esses pares serão analisadas, até que uma relação que implique em não-determinismo e respeite a restrição (ii) seja identificada. A análise ocorre na ordem em que os pares foram listados, então, a primeira tentativa de fusão parte do par (q_0, q_1) . Construído o conjunto $J = \{(q_0, q_1), (q_0, q_2), (q_1, q_3), (q_2, q_4)\}$, define-se as classes de equivalência sobre Q : $\{[q_0, q_1, q_2, q_3, q_4], [q_5], [q_6], [q_7], [q_8], [q_9], [q_{10}]\}$. A busca por estados incompatíveis sobre essa relação revela que a classe $[q_0]$ contém os pares (q_0, q_3) , (q_0, q_4) e (q_2, q_3) , cujos vértices são adjacentes em G . Dessa forma, a fusão entre q_0 e q_1 não pode ocorrer.

O próximo par indicado, segundo o produto cartesiano apresentado, é (q_1, q_3) , que implica no conjunto $J_{(q_1, q_3)} = \{(q_1, q_3), (q_2, q_4)\}$. As classes de equivalência geradas sobre Q são: $\{[q_0], [q_1, q_3], [q_2, q_4], [q_5], [q_6], [q_7], [q_8], [q_9], [q_{10}]\}$. As classes com múltiplos estados, $[q_1, q_3]$ e $[q_2, q_4]$, não possuem estados distinguíveis entre si e não violam a restrição (ii), logo, a fusão entre q_1 e q_3 é bem-sucedida. A MEF M' gerada após tal fusão é exibida na Figura 4.13.

Uma vez obtido o sucesso em uma fusão, a iteração que corria em modo aproximado termina, iniciando-se uma nova iteração com a MEF M' . Espera-se que, após a fusão bem-sucedida e a conseqüente disponibilização de mais informações para o grafo, uma próxima análise de distinção permita o prosseguimento do processo em modo exato. A

Figura 4.13: A MEF M' resultante da primeira iteração no modo aproximado

análise da segunda iteração deste exemplo é apresentada no grafo de distinção G' , exibido na Figura 4.14.

Figura 4.14: Grafo de distinção G' para M' na segunda iteração

O grafo G' contém um único clique maximal de tamanho $n = 3$, sendo $S'_1 = \{q'_0, q'_1, q'_2\}$. A seleção de estados no modo exato para S'_1 , com $k'_i = \{q'_i\}$, resulta em $C_{k'_0} = \emptyset$, $C_{k'_1} = \{q'_8\}$ e $C_{k'_2} = \{q'_9\}$. A MEF M'' , resultante das operações de fusão dos estados selecionados nesta iteração, é exibida na Figura 4.15.

Após a análise de distinção dos estados de M'' e a construção do grafo G'' , exibido na Figura 4.16, é possível concluir que a MEF não é mínima, portanto, ainda é possível reduzi-la com novas iterações. O grafo é exibido na Figura 4.16.

O único clique maximal disponível, $S''_1 = \{q''_0, q''_1, q''_2\}$, é utilizado nesta terceira iteração e, mais uma vez, a seleção no modo exato, com $k''_i = \{q''_i\}$, não produz resultados: $C_{k''_0} = C_{k''_1} = C_{k''_2} = \emptyset$. O método deve então prosseguir no modo aproximado, com $K_2(S''_1) =$

Figura 4.15: A MEF M'' resultante da segunda iteração

Figura 4.16: Grafo de distinção G'' para M'' na terceira iteração

$\{(q_0'', q_1''), \{q_0'', q_2''), \{q_1'', q_2'')\}$. Os pares de estados candidatos à fusão são: (q_0'', q_5'') , (q_0'', q_7'') , (q_0'', q_6'') , (q_1'', q_6'') , (q_2'', q_5'') e (q_2'', q_7'') .

A tentativa de fusão do primeiro par, q_0'' e q_5'' , com $J = \{(q_0'', q_5''), (q_1'', q_6''), (q_2'', q_7'')\}$, define sobre Q'' as classes de equivalência: $\{[q_0'', q_5''], [q_1'', q_6''], [q_2'', q_7''], [q_10'']\}$, nenhuma delas com estados distinguíveis entre si e sem violar a restrição (ii). Uma vez que a fusão é bem-sucedida, a terceira iteração termina, resultando na MEF M''' , exibida na Figura 4.17.

Figura 4.17: A MEF M''' resultante da terceira iteração

Mais uma vez, faz-se a análise de distinção de M''' . Os pares de estados distinguíveis estão organizados no grafo G''' . Como se pode observar, dos quatro estados, três são distinguíveis entre si: q_0''' , q_1''' e q_2''' , formando o único clique maximal $S_1''' = \{q_0''', q_1''', q_2'''\}$ em G''' .

O quarto estado, q_{10}''' , não pôde ser distinguido de nenhum outro, como se pode observar no grafo da Figura 4.18. Nesta situação, a seleção de estados no modo exato não surtirá resultados, também não o fará a seleção no modo aproximado com profundidade $m = 2$, mas somente a seleção no modo aproximado com $m = 3$, ou seja, q_{10}''' é elegível para fusão com qualquer outro estado do clique S_1''' .

Figura 4.18: Grafo de distinção G''' para M''' na terceira iteração

Sendo $\Omega(q_{10}''') = \emptyset$, é indiferente o estado do clique com o qual será fundido, pois todas as combinações resultarão em fusões bem-sucedidas. Pela ordem dos elementos de S_1''' , o par eleito para fusão será (q_0''', q_{10}''') , com $J = \{(q_0''', q_{10}''')\}$, gerando as classes sobre Q''' : $\{[q_0''', q_{10}'''], [q_1'''], [q_2''']\}$, sem agrupamento de estados distinguíveis. A MEF M'''' , gerada após a fusão, é exibida na Figura 4.19.

Figura 4.19: A MEF M'''' resultante da quarta iteração

A quarta iteração uniu o último estado que não pertencia ao clique maximal. O grafo $G'''' = \{q_0'''', q_1'''', q_2''''\}$, que é construído para uma nova iteração, é completo. Pode-se então concluir que a MEF M'''' é mínima.

4.7 Considerações Finais

Neste capítulo foi apresentada uma estratégia para a minimização de MEF parciais que explora a identificação de cliques maximais em grafos de distinção. Após a discussão de todos os aspectos considerados na elaboração desta estratégia, exemplos de sua aplicação foram apresentados.

O próximo capítulo contém relatos de experimentos e testes de bancada, realizados com o intuito de validar a contribuição do método apresentado. Uma vez que a proposta aborda uma solução heurística para um problema NP-completo, a qualidade do método é avaliada do ponto de vista teórico.

Experimentos

5.1 Considerações Iniciais

Para validar a contribuição que o método `cosme` representa na busca por formas mais eficientes de se minimizar MEFs parciais, alguns experimentos foram realizados e são neste capítulo descritos. Tais experimentos são baseados na medição do desempenho de uma implementação do método `cosme` quando utilizada em determinados cenários. Os desempenhos de implementações dos métodos `bica` e `stamina`, que compartilham o mesmo propósito, são também medidos e utilizados para comparações empíricas. No primeiro experimento, que figura na Seção 5.2, a avaliação foi conduzida com dados de entrada oriundos de um conjunto de exemplos utilizado em vários trabalhos neste tópico. A Seção 5.3 contém detalhes do segundo experimento onde, por meio da minimização de MEFs em árvore, as implementações são utilizadas na avaliação da cobertura de casos de teste. No terceiro experimento, descrito na Seção 5.4, as implementações foram utilizadas para minimizar MEFs geradas artificialmente, com variações em características como quantidades de estados, transições, entradas e saídas. Na Seção 5.5, um quarto experimento é apresentado, onde o poder de minimização do novo método é comparado com o método `bica`.

5.2 Comparação Empírica

O desempenho do método **cosme** foi empiricamente medido e comparado, sendo que a referência utilizada como base para essa comparação são dois outros métodos: **bica** (Pena e Oliveira, 1998) e **stamina** (Hachtel et al., 1991). A implementação do método **bica** foi obtida diretamente de um de seus autores. Quanto ao método **stamina**, utilizou-se uma implementação disponível em um conjunto de ferramentas de software para projetos de circuitos eletrônicos, desenvolvido na Berkeley University, chamado **SIS** (Sentovich et al., 1992).

Devido aos relatos de bom desempenho publicados pelos autores dos métodos **slim** (Higuchi e Matsunaga, 1996) e **chesmin** (Gören e Ferguson, 2002), seria desejável que estes fossem considerados no experimento empírico. No entanto, a dificuldade de se obter implementações funcionais impossibilitou a inclusão dos mesmos nas comparações. Como atenuante, o fato de os autores também utilizarem o método **stamina** em suas próprias comparações viabiliza uma comparação relativa com os resultados que são aqui apresentados.

As MEFs utilizadas nos experimentos foram geradas como passos intermediários de um procedimento assíncrono de síntese (Lavagno et al., 1992). O conjunto de exemplos foi originalmente utilizado por Kam et al. (1994) para avaliar seu algoritmo de minimização. Autores de trabalhos posteriores neste tópico de pesquisa, como Gören e Ferguson (2002); Higuchi e Matsunaga (1996); Pena e Oliveira (1998), também utilizaram o mesmo conjunto de exemplos para comparar desempenhos.

A avaliação foi realizada minimizando-se todos os exemplos através de cada um dos métodos. Em cada instância, mediu-se o tempo necessário para completar a execução e o número de estados resultantes, exibido na solução. Os valores observados estão disponíveis na Tabela 5.1, onde a primeira e a segunda coluna contêm, respectivamente, o nome da MEF e o número inicial de estados (N_i). Na terceira, quarta e quinta colunas, são exibidas as quantidades de estados obtidos nas soluções geradas por cada método, na seguinte ordem: **cosme** (C), **bica** (B) e **stamina** (S). Nas últimas três colunas é mostrado o tempo total de execução, medido para cada método, seguindo a mesma ordem anterior.

Situações em que o processo avaliado foi interrompido por erro interno, ou o tempo de execução excedeu 300 segundos, foram consideradas falhas e são indicadas na tabela pelo símbolo “—”. O limite de tempo foi estipulado arbitrariamente, muitas vezes maior que o tempo médio utilizado pelas implementações para minimizar MEFs semelhantes. Os tempos de execução são exibidos em milisegundos e foram medidos em um computador Intel Pentium 4, 3.4 GHz, 6800 bogomips. A área de troca da memória não foi utilizada por nenhum dos processos avaliados.

Tabela 5.1: Desempenho observado

FSM nome	número de estados				tempo de execução		
	N_i	cosme	bica	stamina	cosme	bica	stamina
alex1	42	6	6	6	21.7	982.8	208.7
intel-edge	28	5	4	5	13.2	94.8	23.0
isend	40	4	4	4	22.3	552.4	33.4
rcv-ifc	46	2	2	2	25.8	489.1	20.5
rcv-ifc.m	27	2	2	2	12.9	188.0	18.7
send-ifc	70	2	2	2	75.1	1776.8	29.9
send-ifc.m	26	2	2	2	14.1	611.7	18.9
vbe4a	58	3	3	3	52.6	1459.4	2058.2
vmebus	32	2	2	2	41.8	49.7e3	30.0
th.20	21	4	4	6	7.8	42.2	11.9e3
th.25	26	4	4	-	8.4	47.3	-
th.30	31	6	5	10	10.5	67.9	83.9e3
th.35	36	7	7	-	11.9	87.5	-
th.40	41	8	8	-	14.5	106.6	-
th.55	55	10	8	-	22.0	12.8e3	-
fo.16	17	3	3	3	6.5	33.3	8456.0
fo.20	21	3	3	4	7.0	38.2	36.8e3
fo.30	31	3	3	-	9.3	57.4	-
fo.40	41	6	4	-	13.8	388.3	-
fo.50	51	6	6	-	17.1	265.9	-
fo.70	71	9	-	-	35.7	-	-
ifsm0	38	3	3	3	32.1	194.2	20.2
ifsm1	74	14	14	-	56.1	569.6	-
ifsm2	48	9	9	9	89.1	11.0e3	10.7e3
e271	19	2	2	2	9.3	135.3	17.8
e285	19	2	2	2	9.0	52.5	17.8
e304*	19	19	2	2	9.8	49.3	17.9
e423*	19	19	2	-	335.0	74.1	-
e680*	19	19	2	2	15.1	56.0	18.0

É possível observar que a implementação do novo método exibiu um desempenho inferior ao **bica** em um único exemplo, (**e423**), o qual figura entre algumas MEFs de características especiais, discutidas mais adiante. Na maioria dos casos, diferença no tempo de execução é grande, culminando nos exemplos **ifsm2** e **th.55**, nos quais o tempo utilizado pelo método **bica** é algumas centenas de vezes maior que o tempo necessário ao método **cosme**. Em comparação com o método **stamina**, o novo método perde em eficiência apenas nos exemplos **rcv-ifc**, **send-ifc**, **vmebus** e **ifsm0**, somando 4 dentre os 29 totais.

No tocante à minimização obtida, **cosme** gerou MEFs maiores, com no máximo dois estados a mais, apenas nos exemplos: **intel-edge**, **th.30** e **th.55**.

Os últimos três exemplos da tabela, **e304**, **e423** e **e608**, são MEFs que contam com uma estrutura peculiar: a maioria das saídas não estão definidas. Nesses exemplos, o método **cosme** não foi capaz de obter qualquer minimização, pois desde o início não há

qualquer par de estados que possa ser utilizado em uma fusão sem que a condição de quasi-equivalência seja violada. Apesar destas circunstâncias, o método não manifesta uma falha, mas sim gera como saída a MEF original, sem redução, em tempo hábil.

5.3 Avaliação de Casos de Teste

Como discutido na Subseção 2.7.1, pode-se verificar a completude de um conjunto de casos de teste utilizando um processo de minimização para MEFs parciais. Neste sentido, o experimento descrito a seguir foi organizado para avaliar o comportamento dos métodos de minimização no contexto da avaliação da cobertura de falhas em conjuntos de seqüências de E/S, consideradas como casos de teste para sistemas reativos.

Segundo o procedimento proposto por Yao et al. (1994), descrito com detalhes na Subseção 2.7.1, uma MEF em árvore pode ser construída a partir de um conjunto de seqüências de E/S e, se essas seqüências constituem um conjunto de casos de teste completo, a minimização da MEF em árvore resultará na MEF da especificação, a partir da qual as seqüências foram derivadas.

No experimento, o método HSI (Petrenko e v. Bochmann, 1995b) foi utilizado para gerar seqüências de teste a partir de MEFs na forma mínima, aleatoriamente geradas. As seqüências são então organizadas em uma MEF em árvore, que deve ser minimizada. A MEF resultante é comparada com a MEF aleatória original e, sendo o método HSI capaz de gerar conjuntos completos, espera-se que ambas sejam idênticas. O método HSI foi escolhido para o experimento por ser capaz de gerar os conjuntos de casos de teste com bastante eficiência em comparação com outros métodos que também garantem a total cobertura de falhas.

Utilizou-se uma estratégia para geração de MEFs aleatórias completas, fortemente conectadas e reduzidas, proposta por Simão e Petrenko (2008), que é descrita a seguir. Primeiramente, são criados conjuntos de estados, entradas e saídas com o número requerido de elementos. A geração prossegue em três fases. A primeira fase inicia-se com a seleção de um estado para ser inicial, sendo também marcado como “alcançado”. Em seguida, para cada estado s não marcado, são selecionados, aleatoriamente, um estado s' marcado, uma entrada x e uma saída y . Uma transição que parte de s , com entrada x , saída y e atinge s' é adicionada. Marca-se s' como “alcançado”. Após completar esta fase, uma MEF inicialmente conectada foi gerada. Na segunda fase, o gerador adiciona transições (selecione aleatoriamente um par de estados, uma entrada e uma saída) até que a MEF torne-se completa ou o número de transições indicado seja atingido. Se a MEF obtida não é fortemente conectada, a mesma é descartada e a geração de uma nova MEF inicia-se. Na terceira fase, verifica-se se a MEF gerada é reduzida, e, caso negativo, a mesma é descartada e o procedimento de geração é reiniciado.

No experimento, 1700 MEFs reduzidas foram geradas, com número de estados n variando entre 3 e 20. Um total de 100 MEFs foram geradas para cada valor de n , com número de entradas $i = 2$, número de saídas $o = 2$ e transições $t = n \times 2$. Após a aplicação do método HSI, a organização das seqüências de teste em MEFs em árvore utilizou, em média, um número de estados variando linearmente em relação à n entre 14 e 142.

O tempo considerado nas comparação é a média dos tempos individuais necessários para reduzir cada uma das 100 MEFs em cada n . A Figura 5.1 contém os gráficos para a comparação dos desempenhos exibidos pelos três métodos em relação à n .

Figura 5.1: Comparação de desempenho entre os métodos na redução de MEFs em árvore

Pela natureza do problema, espera-se que o comportamento da variação no tempo de execução dos métodos, em relação ao número de estados na MEF original, seja próximo de exponencial. Para o experimento, sendo esse número linear em relação ao número de estados na MEF reduzida da especificação, o comportamento observado é semelhante ao esperado.

O método **bica** obteve o pior resultado, exigindo mais tempo na minimização em relação aos métodos comparados. O método **cosme** mostrou um desempenho intermediário, inferior ao método **stamina**. Este, por sua vez, embora tenha sido o mais eficiente dos métodos quanto ao desempenho, exibiu um número grande de falhas de execução, mostrando-se incapaz de resolver 65 instâncias, distribuídas tal como no gráfico exibido na Figura 5.2. Considerou-se falha quando uma implementação excedeu o tempo de execução em dois minutos ou o processo finalizou sem exibir resultados válidos. Os demais

métodos não apresentaram falhas. Todas as minimizações resultaram na MEF aleatória original.

Figura 5.2: Porcentagem de falhas exibidas pelo stamina para cada valor de n

5.4 Minimização de MEFs Aleatórias

Para observar a influência de parâmetros como as quantidades de estados, transições, entradas e saídas sobre o comportamento do método proposto, um experimento utilizando MEFs aleatoriamente geradas foi organizado. Para este experimento, um procedimento de geração como o descrito na seção anterior foi utilizado, diferenciado pelo fato de não adicionar transições suficientes para gerar MEFs completas. Sendo n o número de estados, i o número de entradas e t o número de transições em uma MEF, o valor de t pode variar entre $n - 1$, resultando em uma MEF árvore, e $i \times n$, resultando em uma MEF completa.

Novamente, além da implementação do **cosme**, as implementações dos métodos **bica** e **stamina** foram também avaliadas. Entretanto, um número muito grande de falhas apresentado nos testes pelo **stamina** inviabilizou o uso dos resultados coletados nas comparações. Desta forma, apenas os desempenhos dos métodos **cosme** e **bica** são comparados. Neste experimento, falhas são definidas da mesma maneira como no anterior, com base no mesmo tempo limite de dois minutos.

A medição do desempenho seguiu também a metodologia já descrita, sendo que 30 MEFs aleatórias foram geradas para cada valor do parâmetro em avaliação. O tempo considerado é o tempo médio utilizado para minimizar as 30 MEFs. A seguir, os resultados dos experimentos são exibidos e discutidos.

5.4.1 Número de Estados

Foram geradas MEFs com números de estados variando entre 3 e 100. Com o intuito de manter uma quantidade intermediária de transições, t foi definido em função de n , como $n \times 2$. O número de entradas e o número de saídas foram fixados em 4. O gráfico que contém os tempos de minimização está exibido na Figura 5.3.

Figura 5.3: Desempenho dos métodos na redução de MEFs aleatórias em função de n

Pode-se observar grande vantagem para a implementação do **cosme** em relação ao **bica**. Em um total de 2910 MEFs, a implementação do **bica** exibiu 1474 falhas, todas por ultrapassar o limite estipulado para o tempo máximo de execução. Este número de falhas, superior à metade do total de amostras, pode ser explicado pela natureza exponencial da abordagem utilizada no método **bica**. Como é possível observar na distribuição apresentada no gráfico da Figura 5.4, o tempo de execução do **bica** cresce rapidamente, até ultrapassar o valor máximo da escala do eixo Y . As falhas começam a surgir quando MEFs de tamanho superior a 30 estados são minimizadas. Considerando os limites de tempo definidos, a minimização torna-se inviável nas MEFs com 60 ou mais estados, para as quais a proporção de falhas é superior a 90%. Nenhuma falha foi exibida pelo método **cosme**, cujo tempo de execução ficou muito abaixo do limite estipulado, utilizando, no pior caso, 208ms.

Um total de 2214 falhas inviabilizou a inclusão do **stamina** na comparação de desempenho, sendo distribuídas nas proporções exibidas no gráfico da Figura 5.5.

Figura 5.4: Falhas exibidas pelo `bica` na redução de MEFs aleatórias em função de n

5.4.2 Número de Transições

Para este experimento, as MEF foram geradas com $n = 30$ estados. O número de entradas e o número de saídas foram fixados em 4. Variou-se o número de transições t entre 29 e 120, isto é, entre $n - 1$ e $n \times i$, os limites entre uma MEF em árvore e uma MEF completa. Uma vez que somente gerou falhas, o método `stamina` foi completamente excluído deste e dos próximos experimentos. O gráfico que contém os tempos de minimização está exibido na Figura 5.6.

É possível observar que o desempenho do `bica` é bastante influenciado pelo número de transições, variando o tempo de minimização entre algumas dezenas de milissegundos, até aproximadamente quatro segundos. Tempos maiores são necessários quando existe um número intermediário de transições. Poucas transições implicam em menos informações disponíveis e, conseqüentemente, menor esforço para a identificação de estados distinguíveis. Para um número maior de transições, quanto mais próximo de $n \times t$, mais o processo de minimização assemelha-se ao utilizado em MEFs completas.

O desempenho do `cosme` sofre uma influência semelhante, porém em proporções muito menores. Como se pode observar no gráfico da Figura 5.6, esse desempenho é próximo do linear quando comparado com aquele demonstrado pelo `bica`. Somente é possível notar tal influência quando o desempenho é observado em escala adequada, tal como exibido pelo gráfico da Figura 5.7.

Com base nos dados coletados, pode-se afirmar que a influência do número de transições sobre o desempenho do método `cosme` não é relevante.

Figura 5.5: Falhas exibidas pelo stamina na redução de MEFs aleatórias em função de n

5.4.3 Número de Entradas

No presente experimento foram geradas MEFs com $n = 30$ estados. Com o número de saídas fixado $o = 4$, variou-se o número i de entradas entre 2 e 20. No intuito de manter um número intermediário de transições, t foi definido em função de i como $((i + 1) \times 29)/2 - (10 \times i)$. Tal expressão foi definida de forma a gerar transições para menos da metade dos símbolos de entrada a partir de cada estado. O gráfico que contém os tempos de minimização está exibido na Figura 5.6.

Pode-se observar pelo gráfico que o desempenho do **bica** é influenciado de maneira exponencial em relação ao número de entradas. Quanto ao **cosme**, novamente a influência é pequena o suficiente para não ser notada na escala que acompanha o desempenho do **bica**. Para observá-la, é necessário utilizar uma escala de tempo muito menor, como a exibida no gráfico da Figura 5.9.

Por este experimento se pode concluir que a influência do número de entradas sobre o desempenho do **cosme** também é pequena o suficiente para ser ignorada.

5.4.4 Número de Saídas

As MEFs geradas para este experimento também contém um número $n = 30$ de estados. O número de entradas foi fixado $i = 4$ e o número de saídas o foi variado no intervalo entre 2 e 20. O número de transições foi definido como $t = 60$, que equivale ao dobro do número de estados. O gráfico que contém os tempos de minimização está exibido na Figura 5.10.

Figura 5.6: Desempenho dos métodos na redução de MEFs aleatórias em função de t

Segundo os dados, os desempenhos de ambos os métodos não sofrem grande influência do número de saídas. Apesar de algumas variações da ordem de 50ms no tempo médio de minimização apresentado pelo **bica**, o comportamento dos métodos mostrou-se próximo de uma constante em relação ao parâmetro sob avaliação.

A implementação do **cosme** gerou resultados em uma média de 10.8 ms (desvio-padrão de 0.44), mais rapidamente que o **bica**, que por sua vez executou, em média, por 113.5 ms (desvio-padrão de 13.33).

5.5 Comparação do Poder de Minimização

Uma vez que o método **cosme** faz uso de uma heurística para reduzir o tempo de minimização de MEFs parciais, espera-se que os resultados por ele produzidos não sejam exatos e, por essa razão, é importante analisar o quanto esses resultados são bons.

A maneira adotada para se realizar tal análise foi através da comparação empírica do poder de minimização com o método **bica**, sendo que este mostrou produzir os melhores resultados nas publicações em que foi submetido a testes (Gören e Ferguson, 2002; Pena e Oliveira, 1998).

Um experimento foi organizado da seguinte maneira: com base nos dados coletados nos experimentos anteriores, fixou-se alguns parâmetros de forma a exigir um esforço intermediário dos métodos. Os valores são: $n = 30$, $i = 4$, $o = 4$ e $t = 60$. Utilizando esses parâmetros, 1000 MEFs foram aleatoriamente geradas e em seguida minimizadas pelos

Figura 5.7: Desempenho do cosme na redução de MEFs aleatórias em função de t

métodos **cosme** e **bica**. O número de estados resultantes nas duas versões reduzidas de cada MEF foi comparado e a proporção de diferenças é exibida no gráfico da Figura 5.11.

Os valores utilizados na Figura 5.11 consistem na diferença entre o número de estados resultantes da minimização pelo **cosme**, subtraído do número de estados resultantes da minimização pelo **bica**. O **cosme** não reduziu mais que o **bica** nenhuma amostra. Em mais da metade das MEFs, um total de 574, não houve diferença no número de estados finais. Em pouco mais de um terço, um total de 339 MEFs, a minimização do **cosme** gerou um único estado a mais. Dois e três estados a mais ocorreram, respectivamente, em apenas 77 e 10 MEFs. Em média, as MEF reduzidas pelo **bica** utilizaram 15.22 estados, enquanto a redução pelo **cosme** resultou em 15.75 estados. Uma comparação ponderada pelo número de resultados revela uma diferença média de 0.523 estados a mais nas minimizações produzidas pelo método **cosme**.

Pode-se constatar que, em alguns casos, os resultados produzidos pelo método **cosme** são inferiores aos produzidos pelo **bica**. A importância da diferença no número de estados finais é relativa à aplicação na qual a MEF será empregada e, portanto, deve-se considerar a relevância do desempenho na minimização em detrimento da obtenção de MEFs com mais estados.

5.6 Considerações Finais

O comportamento exibido pela implementação do método **cosme** nos experimentos realizados demonstra a contribuição que esta nova abordagem representa na minimização de

Figura 5.8: Desempenho dos métodos na redução de MEFs aleatórias em função de i

MEFs parciais. Em comparação com os métodos **bica** e **stamina**, o **cosme** mostrou-se mais rápido e mais robusto na quase totalidade dos testes aos quais as implementações foram submetidas.

Constatou-se que as soluções resultantes de minimizações com o uso do método proposto são aproximadas, ou seja, as MEFs minimizadas por métodos exatos, como o **bica**, são menores. Tal comportamento já era esperado, pois se faz uso de uma heurística em substituição à enumeração exaustiva de classes de compatibilidade.

No capítulo seguinte são apresentadas as conclusões deste trabalho. A motivação é revista e as contribuições resultantes da solução proposta são discutidas. Também são propostas algumas direções para trabalhos futuros.

Figura 5.9: Desempenho do cosme na redução de MEFs aleatórias em função de i

Figura 5.10: Desempenho dos métodos na redução de MEFs aleatórias em função de o

Figura 5.11: Comparação do poder de minimização entre os métodos cosme e bica

Conclusões

Máquinas de Estados Finitos são amplamente utilizadas na computação para expressar algoritmos, protocolos de comunicações, circuitos de lógica seqüencial, entre outras aplicações. Em particular, 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 ambigüidades.

Em quaisquer aplicações dentre as citadas, a utilização de MEFs reduzidas, isto é, aquelas onde cada estado externa um comportamento único, significa mais simplicidade nos modelos e menor necessidade de recursos físicos em implementações, sem perda de funcionalidade. Entretanto, muitas vezes um modelo disponível contém estados redundantes, isto é, de mesmo comportamento. É comum que tais estados sejam inadvertidamente inseridos por projetistas, ou como resultado de processos automáticos de geração de MEFs, por exemplo, os baseados em descrições de alto nível. Com o intuito de remover esses estados redundantes, procedimentos de minimização podem ser aplicados. Quando uma MEF é completamente especificada, ou seja, estão definidas transições a partir de todos os estados para cada símbolo de entrada, sua redução pode ser obtida por algoritmos de complexidade polinomial. Por outro lado, quando há transições não definidas, a MEF é dita ser parcialmente especificada e sua redução passa a ser um problema NP-Completo.

Neste trabalho, um método que explora heurísticas e otimizações para a eliminação de estados redundantes em MEFs parciais foi proposto. Este método, que recebeu o nome de **cosme**, foi avaliado por meio de experimentos empíricos, a partir dos quais foi possível constatar que a abordagem apresentada, quando comparada a outros métodos de mesma finalidade, mostra eficiência superior em muitos cenários.

6.1 Contribuições

A abordagem que foi proposta é inédita: nenhum trabalho anterior faz uso de informações inferidas a partir de cliques maximais em grafos de distinção para agrupar estados redundantes em classes de compatibilidade. A única referência ao uso dessas estruturas na minimização de MEFs está no trabalho de Pena e Oliveira (1998), onde se utilizou a cardinalidade de um clique maximal do grafo de distinção para estimar o número mínimo de estados necessário à versão reduzida de uma MEF.

A partir dos resultados dos experimentos que foram apresentados no Capítulo 5, pode-se observar que, em vários cenários, o método *cosme* exibe um desempenho superior quando comparado a outros métodos de mesma finalidade.

A atividade de teste de software baseada em modelos beneficia-se dos resultados positivos referentes ao aumento da eficiência na minimização de MEFs, uma vez que a ausência de estados redundantes é requisito para a aplicação de muitas das técnicas de teste funcional baseadas em especificações formais. Há ainda a verificação da completude dos conjuntos de casos de teste, cuja técnica, proposta por Yao et al. (1994), reduz o problema à minimização de MEFs parciais.

6.2 Limitações

Como constatado no experimento apresentado na Seção 5.2, o método não foi capaz de minimizar duas MEFs que continham demasiadas transições com saídas indiferentes. Uma análise detalhada das razões desta limitação é necessária, objetivando entendê-las e, se possível, contorná-las.

A produção de resultados aproximados é inerente à abordagem proposta. O ganho de desempenho é obtido às custas da simplificação do procedimento utilizado para a seleção dos estados que compõe as classes de compatibilidade. Tal simplificação implica em desconsiderar combinações de estados as quais não são derivadas da heurística utilizada. Em certos casos, a combinação que resultaria na melhor minimização está entre as desconsideradas. Ainda assim, demonstrou-se experimentalmente que os resultados produzidos são muito próximos da minimização exata.

6.3 Trabalhos Futuros

A indisponibilidade de implementações executáveis inviabilizou a comparação do método proposto com outros métodos de interesse, como o *chesmin* e o *ism*. Os experimentos para tal comparação devem ser organizados em uma atividade futura. Novas tentativas para se

obter as implementações deverão ser realizadas e, no insucesso, considera-se a possibilidade de construí-las, utilizando como referência as informações contidas nos trabalhos originais.

A implementação poderá ser aprimorada em versões futuras. Na versão atual, já estão incorporadas funcionalidades que visam evitar a busca por mais cliques: observou-se que um mesmo clique maximal é, em muitos casos, utilizado para a seleção de estados em mais de uma iteração. Há indícios de que outras otimizações podem ainda ser inseridas, como estratégias para melhorar o aproveitamento das informações de incompatibilidade de estados que são fornecidas para cada clique.

A limitação referente às MEFs que utilizam grande número de transições com saídas indiferentes deverá receber atenção em trabalhos futuros. Espera-se verificar se tal limitação é solucionável ou inerente à abordagem utilizada no método. Neste segundo caso, é desejável entendê-la de forma mais clara, permitindo isolar com maior precisão as características das MEFs que impedem o prosseguimento das iterações do método.

Referências

- Adrion, W. R.; Branstad, M. A.; Cherniavsky, J. C. Validation, verification, and testing of computer software. *ACM Comput. Surv.*, v. 14, n. 2, p. 159–192, 1982.
- Angluin, D. Learning regular sets from queries and counterexamples. *Inf. Comput.*, v. 75, n. 2, p. 87–106, 1987.
- Barroca, L. M.; McDermid, J. A. Formal methods: Use and relevance for the development of safety-critical systems. *The Computer Journal*, v. 35, n. 6, p. 579–599, 1992.
- Battiti, R.; Protasi, M. Reactive Local Search for the Maximum Clique Problem. *Algorithmica*, v. 29, n. 4, p. 610–637, 2001.
- Biermann, A.; Feldman, J. On the synthesis of finite state machines from samples of their behavior. *IEEE Transactions on Computers*, v. 21, n. 6, p. 592–597, 1972.
- Bochmann, G. V.; Petrenko, A. Protocol testing: review of methods and relevance for software testing. In: *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA: ACM, 1994, p. 109–124.
- Bomze, I.; Budinich, M.; Pardalos, P.; Pelillo, M. The maximum clique problem. In: Du, D.-Z.; Pardalos, P. M., eds. *Handbook of Combinatorial Optimization, volume 4*, Kluwer Academic Publishers, 1999.
- Budd, T. A.; Angluin, D. Two notions of correctness and their relation to testing. *Acta Inf.*, v. 18, p. 31–45, 1982.
- Chan, W. Y. L.; Vuong, C. T.; Otp, M. R. An improved protocol test generation procedure based on uios. In: *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, New York, NY, USA: ACM, 1989, p. 283–294.

- Chow, T. S. Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Software Eng.*, v. 4, n. 3, p. 178–187, 1978.
- Dijkstra, E. W. *Structured programming*. London, UK, UK: Academic Press Ltd., 1–82 p., 1972.
- Frankl, P. G.; Weyuker, E. J. Testing software to detect and reduce risk. *Journal of Systems and Software*, v. 53, p. 275–286, 2000.
- Fujiwara, S.; Bochmann, G. V.; Khendek, F.; Mokhtar, A.; Ghedamsi, A. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, v. 17, n. 6, p. 13, 1991.
- Gendreau, M.; Soriano, P.; Salvail, L. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, v. 41, n. 4, p. 385–403, 1993.
- Gill, A. *Introduction to the theory of finite-state machines*. McGraw-Hill, iISBN 0070232431, 1962.
- Goodenough, J. B.; Gerhart, S. L. Toward a theory of test data selection. In: *Proceedings of the international conference on Reliable software*, New York, NY, USA: ACM, 1975, p. 493–510.
- Grasselli, A.; Luccio, F. A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Electronic Computers*, v. EC-14, n. 3, p. 350–359, 1965.
- Gören, S.; Ferguson, F. J. Chesmin: A heuristic for state reduction in incompletely specified finite state machines. In: *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, Washington, DC, USA: IEEE Computer Society, 2002, p. 248.
- Gören, S.; Ferguson, F. J. On state reduction of incompletely specified finite state machines. *Comput. Electr. Eng.*, v. 33, n. 1, p. 58–69, 2007.
- Hachtel, G. D.; Rho, J. K.; Somenzi, F.; Jacoby, R. Exact and heuristic algorithms for the minimization of incompletely specified state machines. In: *EURO-DAC '91: Proceedings of the conference on European design automation*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, p. 184–191.
- Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. 3, p. 231–274, 1987.
- Harel, D.; Pnueli, A. On the development of reactive systems, p. 477–498. 1985.

- Higuchi, H.; Matsunaga, Y. A fast state reduction algorithm for incompletely specified finite state machine. In: *DAC '96: Proceedings of the 33rd annual conference on Design automation*, New York, NY, USA: ACM, 1996, p. 463–466.
- Hopcroft, J. *Theory of machines and computations*, cap. An N Log N Algorithm for Minimizing States in a Finite Automaton Academic Press, 1972.
- Howden, W. E. *Functional program testing and analysis*. New York, NY, USA: McGraw-Hill, Inc., 1986.
- IEEE *Standard 610.12-1990: Glossary of software engineering terminology*, v. 1 IEEE Press, 1999.
- Kam, T.; Villa, T.; Brayton, R.; Sangiovanni-Vincentelli, A. A fully implicit algorithm for exact state minimization. In: *DAC '94: Proceedings of the 31st annual conference on Design automation*, New York, NY, USA: ACM, 1994, p. 684–690.
- Kannan, L.; Sarma, D. Fast heuristic algorithms for finite state machine minimization. In: *Design Automation. EDAC. Proceedings of the European Conference on*, 1991, p. 192–196.
- La Poutré, J. New techniques for the union-find problem. In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1990, p. 54–63.
- Lavagno, L.; Moon, C. W.; Brayton, R. K.; Sangiovanni-Vincentelli, A. L. Solving the state assignment problem for signal transition graphs. In: *DAC '92: Proceedings of the 29th ACM/IEEE conference on Design automation*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, p. 568–572.
- Maldonado, J. C.; Barbosa, E. F.; Vincenzi, A. M. R.; Márcio Eduardo Delamaro, Simone Rocio Senger Souza, M. J. Introdução ao Teste de Software. Nota Didática, 2004. Disponível em ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/not_did/ND_65.pdf.zip (Acessado em 20/01/2008)
- Moore, E. F. Gedanken-experiments on sequential machines. In: Shannon, C.; McCarthy, J., eds. *Automata Studies*, Princeton, NJ: Princeton University Press, p. 129–153, 1956.
- Myers, G. J. *Art of software testing*. New York, NY, USA: John Wiley & Sons, Inc., 1979.
- Myers, G. J. *The art of software testing*. 2th. ed. John Wiley & Sons, Inc., 2004.

- Naito, S.; Tsunoyama, M. Fault detection for sequential machines by transition tours. In: *IEEE Fault Tolerant Computing Conference*, 1981, p. 238–243.
- Naur, P.; Randell, B., eds. *Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7-11 oct. 1968, brussels, scientific affairs division, nato*. 1969.
- Olivier Coudert, J. C. M. New ideas for solving covering problems. In: *Design Automation, 1995. DAC '95. 32nd Conference on*, 1995, p. 641–646.
- Östergård, P. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, v. 120, n. 1-3, p. 197–207, 2002.
- Pardalos, P.; Rodgers, G. A branch and bound algorithm for the maximum clique problem. *Computers and Operations Research*, v. 19, n. 5, p. 363–375, 1992.
- Paull, M. C.; Unger, S. H. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Transaction on Electronic Computers*, v. EC-8, p. 356–367, 1959.
- Pena, J.; Oliveira, A. A new algorithm for the reduction of incompletely specified finite state machines. In: *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, 1998, p. 482–489.
- Perry, D. E.; Kaiser, G. E. Adequate testing and object-oriented programming. *J. Object Oriented Program.*, v. 2, n. 5, p. 13–19, 1990.
- Peterson, J. L. Petri nets. *ACM Comput. Surv.*, v. 9, n. 3, p. 223–252, 1977.
- Petrenko, A.; v. Bochmann, G. Selecting test sequences for partially-specified nondeterministic finite state machines. In: Luo, G., ed. *IWPTS '94: 7th IFIP WG 6.1 international workshop on Protocol test systems*, London, UK, UK: Chapman & Hall, Ltd., 1995a, p. 95–110.
- Petrenko, A.; v. Bochmann, G. Selecting test sequences for partially-specified nondeterministic finite state machines. In: Luo, G., ed. *IWPTS '94: 7th IFIP WG 6.1 international workshop on Protocol test systems*, London, UK, UK: Chapman & Hall, Ltd., 1995b, p. 95–110.
- Petrenko, A.; Yevtushenko, N.; Lebedev, A.; Das, A. Nondeterministic state machines in protocol conformance testing. *IFIP Transactions*, v. C-19, p. 363 – 378, 1993.
- Pfleeger, C. State reduction in incompletely specified finite-state machines. *IEEE Transactions on Computers*, v. C-22, p. 1099–1102, 1973.

- Pressman, R. S. *Engenharia de Software*. 5th. ed. McGraw-Hill, 2002.
- Pressman, R. S. *Software engineering: A practitioner's approach*. McGraw-Hill Professional, 2005.
- Sentovich, E. M.; Singh, K. J.; Lavagno, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; Stephan, P. R.; Brayton, R. K.; Sangiovanni-vincentelli, A. *Sis: A system for sequential circuit synthesis*. Relatório Técnico, University of California, Berkeley, 1992.
- Simão, A.; Petrenko, A. Generating Checking Sequences for Partial Reduced Finite State Machines. *Testing of Software and Communicating Systems: 20th Ifip Tc 6/Wg 6.1 International Conference, Testcom 2008 8th International Workshop, Fates 2008, Tokyo, Japan, June 10-13, 2008 Proceedings*, 2008.
- Wing, J. M. A specifier's introduction to formal methods. *Computer*, v. 23, n. 9, p. 8–23, 1990.
- Yao, M.; Petrenko, A.; v. Bochmann, G. Fault coverage analysis in respect to an fsm specification. In: *Proc. IEEE Networking for Global Communications. 13th INFOCOM '94*, 1994, p. 768–775 vol.2.