

**UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA**

"VALIDAÇÃO DE ESPECIFICAÇÕES DE SISTEMAS REATIVOS: DEFINIÇÃO E ANÁLISE DE CRITÉRIOS DE TESTE"

SIMONE DO ROCIO SENGER DE SOUZA

**Tese apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo,
para obtenção do título de Doutor em Ciências "Física Aplicada – Opção Física
Computacional"**

OUL
Orientador:

Prof. Dr. José Carlos Maldonado

USP/IFSC/SBI



8-2-001356

Comissão Julgadora:

**Prof. Dr. José Carlos Maldonado (ICMC-USP)
Prof. Dr. Carlos Antonio Ruggiero (IFSC-USP)
Prof. Dr. Marcos José Santana (ICMC-USP)
Prof. Dr. Ricardo de Oliveira Anido (UNICAMP)
Dr. Plinio Roberto Souza Vilela (Telcordia-USA)**

**São Carlos – São Paulo
2000**

**IFSC-USP SERVIÇO DE BIBLIO / ECA
INFORMAÇÃO**

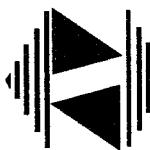
Souza, Simone do Rocio Senger de
Validação de Especificações de Sistemas Reativos: Definição e
Análise de Critérios de Teste/ Simone do Rocio Senger de Souza.
São Carlos, 2000.

p.264

Tese (Doutorado) – Instituto de Física de São Carlos, 2000.

Orientador: Prof. Dr. José Carlos Maldonado

- 1. Teste e Validação. 2. Critérios de Teste de Software.
3. Sistemas Reativos. 4. Especificações Formais. I. Título.**



**MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE SIMONE DO
ROCIO SENGER DE SOUZA APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO
CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 14 DE DEZEMBRO DE 2000.**

COMISSÃO JULGADORA:

Prof. Dr. José Carlos Maldonado/ICMC-SCE-USP

Prof. Dr. Marcos José Santana/ICMC-SCE-USP

Prof. Dr. Carlos Antonio Ruggiero/IFSC-FFI-USP

Prof. Dr. Ricardo de Oliveira Aaido/UNICAMP

Prof. Dr. Plinio Roberto Souza Vilela/Telcordia-USA

*Ao Paulo e
ao Felipe*

Agradecimentos

A Deus por guiar meus passos e me ensinar a ser paciente. Obrigada pelo conforto e proteção.

Ao meu orientador, Prof. Dr. José Carlos Maldonado, pelos ensinamentos, apoio, incentivo e profissionalismo que demonstrou durante os anos que trabalhamos juntos.

“O amor não consiste em duas pessoas olharem uma para outra, mas olharem juntas na mesma direção” (Saint-Exupéry). Ao Paulo, agradeço seu amor, carinho, compreensão e apoio constante.

Ao nosso querido filho Felipe, por nos brindar com sua existência durante a realização deste trabalho, nos ensinando a valorizar as coisas simples da vida e por ser uma fonte incansável de energia, entusiasmo e alegria.

À nossa família, pelo apoio e compreensão. Em especial, à minha mãe Leda que sempre nos auxiliou e motivou e, ao meu pai Silvio, pelo exemplo de dedicação e seriedade. Aos meus irmãos Silvia, Luciano, Jorge e Paula, obrigada pelo carinho e incentivo.

Ao Luciano e à Lilian, pelo carinho com que me acolheram na sua casa em minhas vindas a São Carlos.

À Sandra, pela amizade e por todo auxílio durante a realização deste trabalho. Obrigada pela troca de idéias e pelas revisões da tese.

Ao Adenilso, pela amizade, constante troca de idéias e apoio na revisão da tese.

A Tatiana, pela amizade, pelo apoio na revisão da tese e pelo auxílio na condução dos estudos de casos.

Ao Márcio Souza, pela amizade, companheirismo e pela ajuda nos detalhes finais da tese.

Aos meus amigos do Grupo de Engenharia de Software: Adenilso, Aline, André, Andrea, Antônio Carlos, Auri, Elisa, Elisandra, Ellen, Emerson, Janaína, Luciana, Márcio Delamaro, Maria Istela, Mayb, Reginaldo, Rejane, Rodrigo, Rosana, Rurik, Tatiana, Valéria, Vangrei e Willie. Obrigada pelo companheirismo e por tornarem o LabES um lugar agradável de trabalho.

Aos amigos do Grupo de Sistemas Distribuídos e Programação Concorrente: Aletéia, Álvaro, Ana Elisa, Andrezza, Arion, Célia, Masaki, Daniel, Edmilson, João Carlos (Boca), Jorge, Kalinka, Luciano, Márcio, Mário, Nivaldi, Omar, Roberta, Renata, Renato, Sarita, Silmara, Tatiana, Tomás e Vera, pela convivência e amizade. Em especial, ao Marcos e à Regina Santana, pelo incentivo, ensinamentos e auxílio em todos os momentos.

Ao Prof. Dr. Paulo Cesar Masiero, pelo auxílio na definição dos critérios de teste para Statecharts.

Ao Prof. Dr. Wanderley Lopes de Souza, pela ajuda com a linguagem Estelle.

To PhD Stanislaw Budkowski (Institut National des Telecommunications - France), who has kindly provided a version of the toolset EDT so that we could accomplish this work.

Aos demais amigos conquistados em São Carlos. Em especial, ao Ernesto, Garga, Jaque, Luis Carlos, Marisa, Mirla, Paquito, Rosângela e Rudinei.

À D.Maria, pelo amor e carinho que transmitiu ao Felipe enquanto estivemos em São Carlos. A sua ajuda foi fundamental para o desenvolvimento deste trabalho.

À Beth da Pós-Graduação do ICMC, pela amizade e ajuda em todos os momentos.

A todos os professores e funcionários do ICMC, em especial às bibliotecárias, ao Jacques, Paulinho e técnicos dos laboratórios, pelo suporte necessário para realização deste trabalho.

À Wladerez, pela atenção e simpatia dispensada.

À UEPG, em especial ao DeInfo e à PROPESP, pelo apoio aos nossos afastamentos.

À CAPES, pelo apoio financeiro.

Sumário

Lista de Figuras.....	iv
Lista de Tabelas	vi
Resumo	viii
Abstract.....	ix
Capítulo 1. Introdução.....	1
1.1. Contexto em que a Tese se Insere.....	1
1.2. Motivação	6
1.3. Objetivos.....	7
1.4. Organização do Trabalho	8
Capítulo 2. Revisão Bibliográfica	11
2.1. Considerações Iniciais	11
2.2. Terminologias e Conceitos Básicos	12
2.3. Técnicas de Especificação Formal de Sistemas.....	15
2.3.1. Statecharts.....	20
2.3.2. Estelle	24
2.3.3. <i>A Árvore de Alcançabilidade para Análise de Especificações Formais</i>	33
2.4. Técnicas e Critérios de Teste de Programas.....	39
2.4.1. <i>Critérios de Fluxo de Controle</i>	45
2.4.2. <i>Critérios de Fluxo de Dados</i>	47
2.4.3. <i>Teste de Mutação</i>	53
2.4.4. <i>Estudos Empíricos Relacionados ao Teste de Mutação</i>	62
2.5. Teste de Programas Concorrentes.....	67
2.6. Teste de Especificações Formais	71
2.7. Direções Futuras na Atividade de Teste de Software	81
2.8. Considerações Finais.....	84
Capítulo 3. Teste de Mutação Aplicado para Especificações Baseadas em Estelle.....	86
3.1. Considerações Iniciais.....	86
3.2. Teste de Mutação para Estelle	87
3.2.1. <i>Especificação Exemplo: Protocolo Bit-Alternante</i>	94
3.2.2. <i>Definição dos Operadores de Mutação nos Módulos</i>	99
3.2.3. <i>Definição dos Operadores de Mutação de Interface</i>	110
3.2.4. <i>Definição dos Operadores de Mutação na Estrutura</i>	116
3.3. Estratégias de Teste para Aplicação do Teste de Mutação para Estelle.....	124
3.4. Análise Teórica do Teste de Mutação para Estelle	129

3.4.1. Complexidade dos Operadores de Mutação nos Módulos	131
3.4.2. Complexidade dos Operadores de Mutação de Interface.....	136
3.4.3. Complexidade dos Operadores de Mutação na Estrutura.....	138
3.5. Aspectos para Implementação do Teste de Mutação para Estelle	140
3.5.1. Obtenção de um Conjunto Inicial de Casos de Teste	141
3.5.2. Geração dos Mutantes	143
3.5.3. Execução dos Mutantes.....	146
3.5.4. Análise dos Mutantes e Geração de Conjuntos de Casos de Teste Adequados.....	147
3.6. Considerações Finais.....	148
Capítulo 4. Critérios de Cobertura para Especificações Baseadas em Statecharts e em Estelle.....	150
4.1. Considerações Iniciais	150
4.2. FCCS e FCCE: Critérios de Cobertura para Statecharts e Estelle	152
4.3. Caracterização dos Requisitos e Seqüências de Teste dos Critérios de Cobertura	157
4.3.1. Árvore de Alcançabilidade para Statecharts	158
4.3.2. Árvore de Alcançabilidade para Estelle	161
4.3.3. Obtenção dos Requisitos de Teste dos Critérios FCCS e FCCE	166
4.3.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCS e FCCE	170
4.4. Um Exemplo de Aplicação dos Critérios FCCS.....	173
4.5. Análise Teórica dos Critérios de Cobertura: Relação de Inclusão	176
4.5.1. Relação de Inclusão dos Critérios FCCS	177
4.5.2. Relação de Inclusão dos Critérios FCCE	189
4.6. Estratégia Incremental de Aplicação dos Critérios de Cobertura	191
4.7. Aspectos para Implementação dos Critérios de Cobertura	193
4.8. Considerações Finais.....	196
Capítulo 5. Avaliação dos Critérios de Teste para Statecharts: Um Estudo de Caso	198
5.1. Considerações Iniciais	198
5.2. Estudo de Caso: Sistema de Controle de Passagem em Nível.....	199
5.3. Geração de Seqüências de Teste Adequadas ao Teste de Mutação	204
5.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCS.....	208
5.5. Strength dos Critérios Teste de Mutação e FCCS: Um Estudo de Caso.....	211
5.6. Avaliação de Seqüências de Teste Geradas pela Simulação Programada.....	214
5.7. Considerações Finais.....	223
Capítulo 6. Avaliação dos Critérios de Teste para Estelle: Um Estudo de Caso.....	225
6.1. Considerações Iniciais	225
6.2. Estudo de Caso: Especificação do Protocolo <i>Bit-Alternante</i>	226
6.3. Geração de Seqüências de Teste Adequadas ao Teste de Mutação	227
6.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCE	234
6.5. Avaliação de Seqüências de Teste Geradas pela Simulação: Ferramenta EDT	239
6.6. Considerações Finais.....	243

Capítulo 7. Conclusões.....	245
7.1. Ponderações Finais desta Tese.....	245
7.2. Contribuições desta Tese.....	249
7.3. Sugestões para Trabalhos Futuros	250
Referências Bibliográficas	252
Apêndice A – Descrição em Estelle do Protocolo <i>Bit_Alternante</i>	1

Lista de Figuras

Figura 3.1. Conexão entre Módulos em Estelle.....	92
Figura 3.2. Máquinas de Estados Finitos da Especificação do Protocolo Bit-Alternante, conforme descrito na ISO 9074 (1987).....	95
Figura 2.1. Exemplo de um Statechart (Masiero et al., 1994).....	23
Figura 2.2. Arquitetura de uma Especificação Descrita em Estelle (Lopes de Souza, 1989).....	26
Figura 2.3. Direções de Pesquisa na Área de Teste de Software (Vincenzi, 2000)...	81
Figura 3.1. Conexão entre Módulos em Estelle.....	92
Figura 3.2. Máquinas de Estados Finitos da Especificação do Protocolo Bit-Alternante, conforme descrito na ISO 9074 (1987).....	95
Figura 4.1. Árvore de Alcançabilidade do Statechart Exemplo (Masiero et al., 1994).	159
Figura 4.2. Árvore de Alcançabilidade para o Protocolo Bit-Alternante Especificado em Estelle.	166
Figura 4.3. Algoritmo para Obtenção dos Requisitos de Teste a partir da Árvore de Alcançabilidade.....	167
Figura 4.4. Os Critérios de Cobertura na Validação de Especificações.....	172
Figura 4.5. Statechart do Manipulador de Mouse (Cangussu et al., 1995).	174
Figura 4.6. Árvore de Alcançabilidade do Statechart Figura 4.5 (Masiero et al., 1994).	174
Figura 4.7. Relação Hierárquica dos Critérios FCCS.	177
Figura 4.8. Relação Hierárquica dos Critérios FCCS, considerando a Propriedade da <i>Reiniciabilidade</i>	187
Figura 4.9. Relação Hierárquica dos Critérios FCCE.	189

Figura 4.10. Relação Hierárquica dos Critérios FCCE, considerando a Propriedade da <i>Reiniciabilidade</i>	190
Figura 5.1. Diagrama do Sistema de Passagem em Nível (Barnard, 1998).....	200
Figura 5.2. Statechart do Sistema de Passagem em Nível.....	200
Figura 5.3. Árvore de Alcançabilidade do Sistema de Passagem em Nível.....	201
Figura 5.4. Statechart do Sistema de Passagem em Nível Corrigido.....	204
Figura 5.5. Telas da Execução Programada do Ambiente StatSim.	216
Figura 5.6. Programa para Simulação do Statechart da Figura 5.2.	217
Figura 5.7. Relatório com os Resultados Parciais dos Passos da Simulação Programada.....	218
Figura 5.8. Relatório Estatístico Gerado pela Simulação Programada.	219
Figura 6.1. Máquinas de Estados Finitos da Especificação do Protocolo Bit-Alternante, ilustrada na Figura 3.2.	227
Figura 6.2. Exemplo de Mutação nos Módulos: Operador <i>Substituição do Estado Origem</i>	231
Figura 6.3. Exemplo de Mutação de Interface: Operador <i>Substituição de Comando Output</i>	232
Figura 6.4. Exemplo de Mutação de Interface: Operador <i>Substituição de Variável de não Interface</i>	232
Figura 6.5. Exemplo de Mutação na Estrutura: Operador <i>Remoção de Conexões</i> . ..	233
Figura 6.6. Árvore de Alcançabilidade para a Especificação Estelle do Protocolo Bit-Alternante, apresentada na Figura 4.2.....	235
Figura 6.7. Exemplo de um Programa para Simulação de Especificações Estelle, disponível na Ferramenta EDT.....	240
Figura A.1. Arquitetura em Estelle do Protocolo <i>Bit-Alternante</i>	2

Lista de Tabelas

Tabela 3.1. Síntese dos Operadores de Mutação para Estelle.....	90
Tabela 3.2. Conjunto de Constantes Requeridas.....	94
Tabela 3.3. Conjunto Essencial de Operadores de Mutação para a Linguagem C (Teste de Unidade) e Operadores de Mutação de Módulos para Estelle Relacionados.....	128
Tabela 3.4. Conjunto Essencial de Operadores de Mutação de Interface para a Linguagem C (Teste de Integração) e Operadores de Mutação de Interface para Estelle Relacionados.....	128
Tabela 3.5. Complexidade dos Operadores de Mutação nos Módulos.....	131
Tabela 3.6. Complexidade dos Operadores de Mutação de Interface.....	136
Tabela 3.7 Complexidade dos Operadores de Mutação de Interface.....	139
Tabela 4.1. Primitivas de Entradas e de Saídas para as Transições dos Módulos do Protocolo Bit-Alternante.....	165
Tabela 4.2. Subconjunto de Requisitos de Teste (tr) e Seqüências de Teste (ts) para a Especificação da Figura 4.5.....	175
Tabela 4.3. Total de Requisitos e de Seqüências de Teste Gerados para os Critérios FCCS para o Statechart da Figura 4.5.....	176
Tabela 5.1. Classes de Equivalência para o Sistema de Controle de Passagem em Nível.....	203
Tabela 5.2. Total de Mutantes Gerados, Equivalentes e Anômalos para as Classes de Operadores de Mutação para o Statechart da Figura 5.2.....	206
Tabela 5.3. Tamanho dos Conjuntos de Seqüências de Teste Adequadas às Classes de Mutação para o Statechart da Figura 5.2.....	207
Tabela 5.4. Conjuntos de Seqüências de Teste Adequados ao Teste de Mutação para Statecharts: MEF-Adequado, MEFE-Adequado e ST-Adequado.	207

Tabela 5.5. Total de Requisitos de Teste e de Seqüências de Teste (geradas por construção) para os Critérios FCCS.....	209
Tabela 5.6. Parte dos Requisitos de Teste dos Critérios FCCS.....	210
Tabela 5.7. Seqüências de Teste Geradas por Construção para os Requisitos de Teste dos Critérios de Cobertura para Statecharts apresentados na Tabela 5.6.....	211
Tabela 5.8. Porcentagem de Cobertura das Seqüências FCCS-adequadas em Relação ao Critério Teste de Mutação para Statecharts.....	213
Tabela 5.9. Porcentagem de Cobertura das Seqüências AM-adequadas em Relação aos Critérios de Cobertura para Statecharts.	214
Tabela 5.10. Seqüências de Teste Geradas pela Simulação S _A	220
Tabela 5.11. Seqüências de Teste Geradas pela Simulação S _B	221
Tabela 5.12. Cobertura Obtida pelas Seqüências de Teste Geradas pela Simulação em Relação aos Critérios de Teste para Statecharts.	222
Tabela 6.1. Total de Mutantes para a Especificação do Protocolo Bit-Alternante..	229
Tabela 6.2. Seqüências de Teste Adequadas ao Teste de Mutação para a Especificação do Protocolo Bit-Alternante.	234
Tabela 6.3. Total de Requisitos e de Seqüências de Teste Gerado para os Critérios FCCE para a Especificação Estelle do Protocolo <i>Bit-Alternante</i>	236
Tabela 6.4. Subconjunto de Requisitos de Teste dos Critérios de Cobertura para a Especificação Estelle do Protocolo <i>Bit-Alternante</i>	237
Tabela 6.5. Subconjunto de Seqüências de Teste Adequado por Construção aos Critérios de Cobertura para a Especificação Estelle do Protocolo <i>Bit-Alternante</i>	238
Tabela 6.6. Seqüências de Transições Geradas pelo Programa de Simulação da Figura 6.7.	241
Tabela 6.7. Resultado da Aplicação de Seqüências de Teste geradas pela Simulação para os Mutantes Gerados para a Especificação do Protocolo <i>Bit-Alternante</i>	242
Tabela 6.8. Resultado da Avaliação da Cobertura das Seqüências de Teste Geradas pela Simulação em relação à FCCE para a Especificação do Protocolo <i>Bit-Alternante</i>	243

Resumo

Este trabalho investiga a aplicação de critérios de teste para o teste de especificações do aspecto comportamental de Sistemas Reativos, descritos em Estelle e em Statecharts. A utilização de Sistemas Reativos em várias atividades humanas requer uma maior qualidade tanto do produto como do processo de desenvolvimento, pois falhas nesses sistemas podem ocasionar riscos para vidas humanas e perdas econômicas. Os critérios de teste propostos nesta tese visam a fornecer uma medida de cobertura dos testes, permitindo que a qualidade da atividade de teste possa ser mensurada e avaliada. Esta tese apresenta contribuições para as três atividades fundamentais no contexto de teste de software, que são: definição de critérios de teste, desenvolvimento de estudos teóricos/empíricos e desenvolvimento de ferramentas. Com relação à definição de critérios de teste, é proposta a aplicação do Teste de Mutação para Estelle e a aplicação de critérios de Fluxo de Controle para Estelle e Statecharts. Para o Teste de Mutação, são identificados os tipos de erros em especificações Estelle, definindo-se os operadores de mutação, estratégias de teste incrementais e critérios de mutação alternativa que visam a diminuir o custo de aplicação desse critério. Para os critérios de Fluxo de Controle, foram definidas duas famílias de critérios: FCCS – *Família de Critérios de Cobertura para Statecharts* e FCCE - *Família de Critérios de Cobertura para Estelle*. Estudos teóricos são realizados visando a analisar a complexidade do Teste de Mutação para Estelle e a relação de inclusão dos critérios FCCS e FCCE. Estudos empíricos são realizados visando a comparar os critérios de teste definidos e a analisar a sua aplicação durante a simulação de especificações Estelle e Statecharts. Com relação ao desenvolvimento de ferramentas, a família de ferramentas *Proteum*, que apóia a aplicação do teste de Mutação, e os ambientes para simulação de especificações Estelle (EDT) e Statecharts (StatSim) fornecem uma base essencial para o desenvolvimento das ferramentas. São apresentadas algumas ponderações que devem ser consideradas para a definição de ferramentas de apoio à aplicação dos critérios propostos.

Abstract

Reactive Systems are applied to several human activities and as failures in these systems may cause human or economical losses, it is required the use of high-quality software development processes that would lead to the production of high-quality products. This thesis investigates criteria for testing of Reactive Systems' behavior specifications, specified either in Estelle or in Statecharts. These criteria systematize the testing activity and provide mechanisms for the software tests quality assessment. This thesis presents contributions to the three fundamental activities in the context of software testing, which are: definition of testing criteria, theoretical studies and tool development. In relation to the definition of testing criteria, it is proposed the use of Mutation Testing for Estelle specifications and the use of Control Flow Testing for Estelle and Statecharts specifications. For Mutation Testing, the errors types in Estelle specifications are identified; mutation operators are defined and incremental testing strategies are established. In this context, it is explored the alternative mutation criteria, which aim at reducing the cost of application of the Mutation Testing. For Control Flow Testing, two families of criteria are defined: SCCF – Statechart Coverage Criteria Family and ECCF – Estelle Coverage Criteria Family. Theoretical studies are accomplished to analyze the complexity of the Mutation Testing to Estelle and the inclusion relation for the FCCS and FCCE criteria. Case studies are conducted to evaluate the testing criteria defined in this thesis. The application of these criteria during the simulation of Estelle and Statecharts specifications is analyzed. The Proteum family tools, that supports the application of Mutation Testing, and the simulation environments to Estelle (EDT) and Statecharts (StatSim) supply an essential base for tools development. Considerations about the definition of supporting tools to the application of the proposed criteria are realized.

Capítulo 1. Introdução

Neste capítulo é apresentado o contexto em que o trabalho desta tese está inserido, procurando dar subsídios para caracterizar as contribuições desta tese. Apresentam-se também a motivação, os objetivos e a organização do trabalho.

1.1. Contexto em que a Tese se Insere

Sistemas reativos são sistemas que reagem a estímulos internos ou do ambiente, de forma a produzir resultados corretos dentro de intervalos de tempo previamente especificados e podem apresentar um comportamento bastante complexo (Harel e Politi, 1998). Alguns exemplos de sistemas reativos são: sistemas de caixas automáticos, de reservas aéreas, sistemas controladores de aeronaves, de automóveis, sistemas de telecomunicações, sistemas de monitoramento de pacientes, dentre outros.

Considerando as características dos sistemas reativos, observa-se que esses sistemas devem funcionar com alto grau de confiabilidade e segurança, pois falhas podem ocasionar riscos para vidas humanas e perdas econômicas. Para que esses riscos sejam evitados ou minimizados, é necessária a utilização de métodos e técnicas mais rigorosos para o seu desenvolvimento.

As técnicas formais contribuem nesse sentido, fornecendo uma linguagem gráfica e/ou descritiva para a especificação do sistema. Essas técnicas apresentam uma base matemática para a descrição das propriedades e do aspecto comportamental do sistema, permitindo a especificação, o desenvolvimento e a verificação de sistemas de uma maneira sistemática e rigorosa. A base matemática é dada, em geral, por uma linguagem de especificação formal que fornece condições para o desenvolvimento de sistemas livres de ambigüidades e inconsistências, possibilitando a verificação de propriedades sem a necessidade de executar o software (Pressman, 2000).

Quando os métodos formais são empregados no início do desenvolvimento do software, eles podem auxiliar na detecção de falhas que só seriam reveladas mais tarde durante os testes, na fase de implementação (Fabbri, 1996). Algumas técnicas de especificação formal utilizadas na descrição de sistemas reativos são: Redes de Petri (Peterson, 1977), Máquinas de Estados Finitos (Gill, 1962), Statecharts (Harel et al., 1987), Estelle (ISO, 1987; Budkowski e Dembinski, 1987), Lotos (Bolognesi e Brinksma, 1987), SDL (Turner, 1993), dentre outras. Em geral, essas técnicas procuram modelar a hierarquia, comunicação, sincronização e paralelismo do sistema.

Apesar de todo o rigor das técnicas formais, não se garante que a especificação formal esteja livre de erros e de acordo com os requisitos do usuário. Quanto mais cedo os erros são detectados no processo de desenvolvimento de software, mais fácil torna-se a tarefa de removê-los. Sendo assim, é fundamental a inserção de atividades de *Garantia de Qualidade* durante todo o processo de desenvolvimento de software.

As atividades de *Garantia de Qualidade de Software* têm o objetivo de garantir que a qualidade seja mantida em cada passo do processo de desenvolvimento. Alguns

exemplos dessas atividades são: uso de métodos e ferramentas para o desenvolvimento do software, de revisões técnicas, de estratégias de teste, controle de documentação e de mudanças.

A atividade de teste de software é um elemento crítico da Garantia de Qualidade, representando a última revisão da especificação, projeto e codificação. Para que a atividade de teste de software possa ser conduzida de forma planejada e sistematizada, técnicas e critérios de teste devem ser empregados. Um critério de teste estabelece os requisitos que devem ser cumpridos durante o teste e pode fornecer uma medida de cobertura que contribui para avaliar a qualidade e a adequação da atividade de teste.

Os critérios de teste são classificados nas seguintes técnicas: *Funcional* – o software é visto como uma “caixa preta”, sendo que os requisitos de teste são estabelecidos a partir das funções especificadas; *Estrutural* – os requisitos de teste são derivados a partir das características da implementação em teste e *Baseada em Erros* – os requisitos de teste são estabelecidos a partir do conhecimento dos erros típicos que podem ser cometidos no processo de desenvolvimento do software. No contexto desta tese, são investigados os critérios estruturais Baseados em Fluxo de Controle e o critério baseado em Mutações (Pressman, 2000).

As técnicas de teste devem ser vistas como complementares, pois, em geral, nenhuma delas é suficiente para garantir a qualidade da atividade de teste. A questão está em como utilizá-las de forma que as vantagens de cada uma sejam melhor exploradas em uma estratégia de teste (Maldonado et al., 1998).

No contexto de teste de programas, são encontradas várias iniciativas que propõem técnicas e critérios de teste (Herman, 1976; DeMillo et al., 1978; Myers, 1979; Laski e Korel, 1983; Ntafos, 1984; Rapps e Weyuker, 1985; Coward, 1988;

Ural e Yang, 1988; Beizer, 1990; Maldonado, 1991; Pressman, 2000). Esses critérios de teste foram definidos para programas seqüenciais, e não abordam características específicas de programas concorrentes, como execuções paralelas, comunicações e sincronizações. Nesse sentido, iniciativas de se estabelecer critérios para o teste de programas concorrentes podem ser identificadas (Yang e Chung, 1992; Taylor et al., 1992; Chung et al, 1996; Koppol e Tai, 1996; Yang et al., 1998; Silva-Barradas, 1998). Esses critérios são definidos com base nos critérios de teste para programas seqüenciais.

No contexto de especificação, podem ser conduzidos *testes de especificação* e *testes de conformidade* (ou teste baseado na especificação). O *teste de especificação* é análogo ao teste de programas e visa a encontrar erros na especificação e a garantir que a especificação esteja de acordo com os requisitos do usuário. O *teste de conformidade*, que na verdade é um teste funcional, visa a garantir que a implementação esteja em conformidade com a sua especificação, sendo essencial que a especificação esteja correta e retrate os requisitos do usuário. Em geral, teste de conformidade é aplicado no teste de protocolos de comunicação. Observa-se que os testes utilizados no teste de especificação podem ser posteriormente utilizados no teste de conformidade (Bourhfir et al., 1996).

Considerando o teste de especificações baseadas em Máquinas de Estados Finitos (MEF), existem vários métodos de geração de seqüências de teste, os quais são sintetizados por Ural (1992) e Bourhfir et al. (1996). Entretanto, esses métodos não englobam, necessariamente, os aspectos de dados da especificação.

Harel (1992) aponta que a validação de especificações de sistemas reativos é, em geral, realizada através de alguns tipos de simulação: *execução interativa, batch,*

programada e exaustiva. A partir da simulação, o usuário pode avaliar se o comportamento da especificação está de acordo com o esperado.

Conforme apontado por Petrenko e Bochmann (1996), é necessário que a análise de cobertura seja considerada também no contexto de teste de especificações de modo a contribuir para quantificar a qualidade da atividade de teste. Nesse sentido, podem-se destacar alguns trabalhos que utilizam o conhecimento adquirido no nível de programas para o teste de especificação, com uma perspectiva também de teste de conformidade. Esses trabalhos propõem critérios de Fluxo de Dados e de Fluxo de Controle para o teste de especificações baseadas em Máquinas de Estados Finitos Estendidas (MEFEs) e Estelle (Sarikaya et al., 1987; Ural, 1987; Ural e Yang, 1991; Vuong et al., 1994; Bourhfir et al., 1997; Kim et al., 1998) e aplicação do Teste de Mutação para Estelle (Probert e Guo, 1991), Máquina de Estados Finitos (MEF) (Fabbri et al., 1994), Redes de Petri (Fabbri et al., 1995) e Statecharts (1999a). Dado o aspecto complementar das técnicas e critérios de teste no nível de programas, é interessante também investigar mecanismos para analisar o aspecto complementar entre os critérios definidos para o teste de especificações.

Considerando o contexto acima, o trabalho desta tese é relacionado com o teste de especificações, investigando critérios para o teste de especificações baseadas em Statecharts e Estelle. O Teste de Mutação para Estelle é definido, abrangendo as características intrínsecas de Estelle, tais como: comportamento, comunicação, estrutura (ou arquitetura) e paralelismo. Os critérios de cobertura definidos para Estelle e Statecharts exploram informações sobre o fluxo de controle da especificação e são aplicados com base no comportamento da especificação, um aspecto relevante no contexto de sistemas reativos. Além disso, os critérios de teste propostos nesta tese visam a fornecer uma medida de cobertura da atividade de teste,

contribuindo para quantificar essa atividade. Outro aspecto importante é que os critérios de teste podem ser utilizados para complementar outras atividades de validação existentes, como por exemplo, simulações de especificações.

1.2. Motivação

A crescente utilização de software em aplicações consideradas críticas (sistemas reativos, por exemplo), exige uma maior qualidade do software. Apesar da utilização de técnicas formais para o desenvolvimento desses sistemas, não é possível garantir que o sistema seja livre de erros. Sendo assim, a atividade de teste, entre outras, é fundamental e deve ser empregada desde o início do desenvolvimento do sistema.

Harrold (2000) aponta algumas direções de pesquisa que são relevantes no contexto de teste de software e que devem ser investigadas de modo a fornecer métodos, processos e ferramentas de teste para o desenvolvimento de software de alta qualidade. Para atingir esse objetivo, é fundamental a definição de estratégias de teste que possam ser aplicadas durante todo o processo de desenvolvimento de software considerando, nessa estratégia, novos critérios de teste ou critérios de teste já existentes e que apresentam alta eficácia em revelar erros. Além disso, é necessário que o processo seja automatizado de modo que a atividade de teste possa ser realizada de forma eficiente. Para dar suporte ao desenvolvimento de ferramentas e de estratégias de teste, a realização de estudos teóricos e empíricos é essencial, pois fornece evidências da eficácia e custo dos critérios de teste existentes.

No contexto de teste de especificações, técnicas e critérios de teste são propostos, visando a auxiliar na seleção de conjuntos de casos de teste. Algumas dessas técnicas e critérios são definidos com base no teste de programas, mostrando

que é promissor utilizar os critérios definidos no nível de programas para o nível de especificação. Além disso, essas técnicas e critérios procuram considerar o aspecto de cobertura da especificação, o qual é relevante para avaliar a qualidade da atividade de teste.

Este trabalho visa a dar continuidade à linha de pesquisa do Grupo de Engenharia de Software do ICMC/USP, relacionada com a definição de critérios para o teste de especificações. O trabalho de Fabbri (1996), que define o Teste de Mutação para o teste de especificações, foi o primeiro trabalho de pesquisa do grupo nessa linha e motivou a definição desse critério para o contexto de Estelle, como também a caracterização dos critérios de cobertura para Statecharts. Este trabalho contribui com um dos objetivos do grupo, que é o estabelecimento de um ambiente integrado para apoiar a atividade de teste desde a especificação até a implementação.

1.3. Objetivos

O objetivo deste trabalho é investigar a aplicação de critérios que fornecem uma medida de cobertura da atividade de teste para a validação de especificações de Sistemas Reativos, descritos em Estelle e em Statecharts. São considerados os critérios de teste baseado em Fluxo de Controle e baseado em Mutações.

Nesse sentido, o interesse é estudar as técnicas de especificação formal Estelle e Statecharts, identificando as características dessas técnicas, de modo que os critérios de teste focalizem as características intrínsecas dessas técnicas.

Com relação ao Teste de Mutação, têm-se os seguintes objetivos:

- Explorar a aplicação desse critério no contexto de Estelle, a exemplo de Fabbri (1996) e Probert e Guo (1991), definindo os tipos de erros que

podem ser cometidos nessa técnica e, em seguida, definindo operadores de mutação;

- Definir estratégias de teste incremental para a aplicação desse critério;
- Analisar teoricamente o custo de aplicação desse critério no contexto de Estelle;

Com relação aos critérios de cobertura, têm-se os seguintes objetivos:

- Definição desses critérios de teste no contexto de especificações em Estelle e em Statecharts;
- Análise teórica da relação de inclusão desses critérios e o estabelecimento de estratégias incrementais para a aplicação dos critérios;
- Realização de um estudo de caso para analisar o *strength* (dificuldade de satisfação de um critério de teste) entre o Teste de Mutação e os critérios de cobertura no contexto de Statecharts.

1.4. Organização do Trabalho

Este trabalho está organizado em 7 capítulos e um apêndice. Neste capítulo foram apresentados: o contexto em que este trabalho se insere, a motivação e os objetivos para sua realização.

No Capítulo 2 é apresentada uma revisão bibliográfica relacionada ao tema desta tese, dando-se ênfase às fases de especificação e teste de software. Discutem-se com mais detalhes as técnicas formais Statecharts e Estelle e os critérios de teste estruturais (Fluxo de Controle e de Dados) e Teste de Mutação. São apresentadas as principais ferramentas que apóiam a aplicação de critérios de teste e os resultados de estudos empíricos relacionados com o Teste de Mutação. São descritos trabalhos que

definem critérios de teste para a validação de programas concorrentes e validação de especificações formais.

No Capítulo 3 é apresentada a definição do critério Teste de Mutação para validação de especificações em Estelle. Definem-se os operadores de mutação e uma estratégia incremental para aplicação desse critério de teste. É feita uma análise teórica da complexidade (custo) do Teste de Mutação para Estelle. São apresentadas também algumas ponderações sobre os aspectos que devem ser considerados para o desenvolvimento de uma ferramenta de apoio para aplicação desse critério de teste, no contexto de Estelle.

No Capítulo 4 são apresentados os critérios de cobertura, chamados de Família de Critérios de Cobertura para Statecharts (FCCS) e Família de Critérios de Cobertura para Estelle (FCCE), para validação de especificações em Statecharts e em Estelle, respectivamente. Descreve-se a utilização da árvore de alcançabilidade como mecanismo de apoio para aplicação dos critérios de cobertura. Apresentam-se também a análise teórica da relação de inclusão dos critérios de cobertura e os aspectos que devem ser considerados para o desenvolvimento de uma ferramenta de apoio para aplicação desses critérios.

No Capítulo 5 são apresentados os resultados de um estudo de caso, utilizando a especificação em Statecharts de um Sistema de Passagem em Nível (Barnard, 1998), para avaliação dos critérios de teste para Statecharts: Teste de Mutação (Fabbri et al., 1999a) e FCCS. É ilustrado também o emprego desses critérios de teste para complementar a atividade de simulação de Statecharts, utilizando o simulador de Statecharts do ambiente StatSim (Masiero et al., 1991).

No Capítulo 6 são apresentados os resultados de um estudo de caso, utilizando a especificação em Estelle do protocolo de comunicação *Bit_Alternante* (ISO, 1987),

para avaliação dos critérios de teste para Estelle: Teste de Mutação e FCCE. É ilustrado também o emprego desses critérios de teste para complementar a atividade de simulação de especificações Estelle, utilizando o simulador de especificações Estelle da ferramenta EDT (EDT, 2000).

No Capítulo 7 são apresentadas as conclusões desta tese e as sugestões para trabalhos futuros.

No Apêndice A é apresentada a especificação em Estelle do protocolo de comunicação *Bit_Alternante*, utilizado para ilustrar os critérios de teste para Estelle definidos nesta tese.

Capítulo 2. Revisão Bibliográfica

2.1. Considerações Iniciais

Neste capítulo é apresentada uma revisão sobre a atividade de teste de software, incluindo o teste de especificações, o qual constitui o principal objetivo deste trabalho. Com relação à atividade de especificação do software, são descritas algumas técnicas de especificação formal existentes, enfatizando as técnicas Estelle e Statecharts, técnicas alvo deste trabalho.

Com relação à atividade de teste de software, são apresentados os conceitos básicos, técnicas e critérios de teste. Segundo Maldonado (1997), os conhecimentos na área de teste de software podem ser caracterizados em três grupos: estudos teóricos, estudos empíricos e desenvolvimento de ferramentas. Seguindo essa divisão, neste capítulo são feitas algumas considerações sobre esses grupos de conhecimentos.

É apresentada uma descrição de trabalhos que definem técnicas e critérios de teste para a validação de programas concorrentes e para a validação de especificações de sistemas reativos. Esses trabalhos fornecem a base para a definição dos critérios de teste descritos nesta tese.

2.2. Terminologias e Conceitos Básicos¹

Nesta seção são apresentados alguns conceitos e terminologias fundamentais para o entendimento deste texto. Alguns conceitos mais específicos serão introduzidos à medida que forem citados no texto.

Em se tratando da atividade de teste de software, é importante diferenciar defeito, engano, erro e falha. Segundo a terminologia IEEE 610.12 (1990), *defeito (fault)* é um passo, processo (ou definição de dados) incorreto, por exemplo, uma instrução ou comando incorreto no programa; *engano (mistake)* é a ação humana que produz um resultado incorreto, por exemplo, uma ação incorreta feita pelo programador; *erro (error)* ocorre quando o valor esperado e o valor obtido não são os mesmos, ou seja, qualquer resultado inesperado na execução do programa constitui um erro e *falha (failure)* é a produção de uma saída incorreta em relação à especificação. Nesta tese, são utilizados os termos *erro* (causa) e *falha* (consequência), de modo que o termo erro engloba defeito, engano e erro. De uma forma geral, os erros podem ser classificados em *erros computacionais* – o erro provoca uma computação incorreta mas o caminho executado (seqüência de comandos) é igual ao caminho esperado e *erros de domínio* – o caminho efetivamente executado é diferente do caminho esperado, ou seja, um caminho errado é selecionado (Barbosa et al., 2000a).

Teste é uma atividade de verificação e validação. A atividade de *verificação* visa a assegurar consistência, completitude e corretitude do produto em cada fase e entre fases consecutivas do ciclo de vida do software. Nessa atividade, a seguinte questão é feita: “*O produto está sendo construído corretamente?*”. A atividade de *validação*

¹ As informações apresentadas nesta seção foram parcialmente extraídas de Maldonado et al., 1998, Barbosa et al., 2000a e Souza et al., 2000c.

visa a assegurar que o produto final corresponda aos requisitos do software. A seguinte questão é feita: “*O produto certo está sendo construído?*” (Pressman, 2000).

É importante também diferenciar *teste de programas*, *teste de especificações* e *teste baseado na especificação*. O *teste de programas* visa a encontrar erros no programa e procura garantir que ele esteja de acordo com os requisitos estipulados pelo usuário. O teste de especificações é análogo ao teste de programas, sendo que o alvo é validar a especificação, ou seja, procura garantir que a especificação do software esteja de acordo com os requisitos estipulados. *O teste baseado na especificação* ou *teste funcional*, utiliza como base a especificação para validar o software. Os requisitos de teste e casos de teste são derivados a partir da especificação e utilizados para testar o programa que implementa a especificação.

Um tipo particular de teste funcional é o *teste de conformidade de protocolos* que visa a garantir que a implementação esteja em conformidade com a sua especificação, ou seja, é utilizado para validar a implementação de protocolos de comunicação em relação à sua especificação visando a garantir que esse protocolo esteja compatível com outras implementações do mesmo protocolo (Bochmann e Petrenko, 1994). Para esse tipo de teste, é essencial que a especificação esteja correta e retrate os requisitos do usuário.

A atividade de teste de software apresenta uma série de limitações (Howden, 1987; Ntafos, 1988; Rapps e Weyuker, 1985), sendo que, de uma maneira geral, os seguintes problemas são indecidíveis: a) dados dois programas, se eles são equivalentes; b) dadas duas seqüências de comandos (caminhos) de um programa, ou de programas diferentes, se eles computam a mesma função; e c) dado um caminho se ele é executável ou não, ou seja, se existe um conjunto de dados de

entrada que levam à execução desse caminho. Outra limitação fundamental é a correção coincidente - o programa pode apresentar, coincidentemente, um resultado correto para um item particular de um dado $d \in D$, ou seja, um particular item de dado ser executado, satisfazer um requisito de teste e não revelar a presença de um erro (Maldonado et al., 1998).

As seguintes definições, extraídas de Maldonado et al. (1998), também são importantes no contexto desta tese: a) um programa P com domínio de entrada D é correto com respeito a uma especificação f se $f(d) = P^*(d)$ para qualquer item de dado d pertencente a D , ou seja, se o comportamento do programa está de acordo com o comportamento esperado para todos os dados de entrada; b) para dois programas P_1 e P_2 , se $P_1^*(d) = P_2^*(d)$, para qualquer $d \in D$, diz-se que P_1 e P_2 são **equivalentes**; c) na atividade de teste de software, pressupõe-se a existência de um oráculo - o testador ou algum outro mecanismo - que possa determinar, para qualquer item de dado $d \in D$, se $f(d) = P^*(d)$, dentro de limites de tempo e esforços razoáveis. Um oráculo decide simplesmente se os valores de saída são corretos ou não.

Sabe-se que o teste exaustivo é impraticável, ou seja, testar o programa para todos os elementos possíveis do domínio de entrada é, em geral, caro e demanda muito mais tempo do que o disponível (Myers, 1979).

Observa-se que não existe um procedimento de teste de propósito geral que possa ser usado para provar que um programa está correto, ou seja, se erros não são encontrados isso não significa que eles não existam. Apesar disso, os testes conduzidos de forma sistemática e criteriosa podem contribuir para aumentar a confiança de que o software desempenha as funções especificadas, podendo também evidenciar algumas características mínimas do ponto de vista de qualidade do produto.

A análise de cobertura é uma atividade realizada durante a condução e avaliação da atividade de teste e consiste em determinar o percentual de elementos requeridos por um dado critério de teste que foram exercitados pelo conjunto de casos de teste utilizado. A partir dessa informação, o conjunto de casos de teste pode ser aprimorado acrescentando-se novos casos de teste para exercitar os elementos ainda não cobertos. Nessa perspectiva, é fundamental o conhecimento sobre as limitações teóricas relacionadas à atividade de teste, pois os elementos requeridos podem ser não executáveis e, em geral, determinar a não executabilidade de um dado requisito de teste envolve a participação do testador.

2.3. Técnicas de Especificação Formal de Sistemas

A especificação é um modelo ou abstração de uma situação real que, dependendo da complexidade do problema modelado, pode ser descrita em níveis de detalhes. Durante a especificação são identificados os requisitos do sistema, as informações a serem processadas, as funções, desempenho desejado e o comportamento esperado do sistema.

Dentre as técnicas de especificação de sistemas, as técnicas formais são as mais indicadas para a modelagem de sistemas reativos, dado que esse tipo de sistema deve funcionar com alta confiabilidade e segurança. As técnicas de especificação formal possibilitam o desenvolvimento de especificações consistentes, completas e não ambíguas, empregando, para isso, uma linguagem com sintaxe e semântica formalmente definidas.

Em geral, as técnicas de especificação formal são empregadas na descrição de sistemas reativos, protocolos de comunicação e sistemas distribuídos. Conforme

definido anteriormente, Sistemas Reativos são sistemas baseados em eventos que reagem a estímulos internos ou do ambiente, interagindo com o mesmo de forma a produzir resultados corretos dentro de intervalos de tempo previamente especificados (Harel et al., 1987).

Protocolos de comunicação são regras que conduzem à comunicação entre os componentes de um sistema distribuído. Com o objetivo de reduzir a complexidade dessas regras de comunicação, usualmente, elas são divididas em uma hierarquia de camadas de protocolo. Segundo Bochmann e Petrenko (1994), a maioria dos protocolos de comunicação possuem uma natureza reativa, e as técnicas de especificação formais utilizadas para descrição de sistemas reativos são adequadas para especificá-los. Máquinas de Estados Finitos e técnicas de descrição formal como Estelle, Lotos e SDL são empregadas para a descrição formal do protocolo de comunicação.

Um sistema distribuído envolve tanto hardware como software e pode ser definido como um conjunto de computadores interligados por uma rede de comunicação e equipados com um sistema operacional distribuído, o qual é responsável por coordenar as atividades desenvolvidas e compartilhar os recursos do sistema (Coulouris et al., 1994). Segundo Tanenbaum (1996), é o sistema operacional distribuído que mantém as características necessárias do sistema distribuído (por exemplo, transparência para o usuário) utilizando uma rede de computadores como meio de comunicação.

Em síntese, sistemas reativos, protocolos de comunicação (que podem ser considerados sistemas reativos) e sistemas distribuídos, possuem as seguintes características em comum: concorrência, paralelismo, não determinismo,

comunicação e sincronização. Essas características precisam ser modeladas pela técnica de especificação utilizada para descrevê-los.

Dentre as técnicas formais, as que possuem apoio gráfico são bastante empregadas, pois facilitam a visualização e a descrição do sistema. Algumas que se destacam são: Redes de Petri (Peterson, 1977), Máquinas de Estados Finitos (Gill, 1962) e Statecharts (Harel et al., 1987).

As *Redes de Petri* foram criadas para modelar sistemas que apresentam concorrência, paralelismo, comunicação síncrona e assíncrona². Sua execução gera uma seqüência de eventos, obtidos a partir do disparo das transições, podendo ocorrer não determinismo no disparo das transições. Outro fator importante é que as Redes de Petri podem ser utilizadas para análise de alcançabilidade do sistema e para detecção de impasses (*deadlock*), sendo que a árvore de alcançabilidade é uma das principais técnicas para análise de Redes de Petri (Peterson, 1977).

A técnica *Máquinas de Estados Finitos (MEFs)* é muito utilizada para especificação do aspecto comportamental de sistemas reativos, particularmente, na área de protocolos de comunicação. O sistema é modelado como uma máquina, composto de estados e transições, estando em somente um de seus estados num dado momento (Gill, 1962). A partir de uma entrada, a máquina gera uma saída e muda de estado. A máquina pode ser representada por um diagrama de transição de estados ou por uma tabela de transição (Fabbri, 1996). Quando representada por meio de um diagrama de transições, círculos representam os estados e arcos direcionados representam as transições que podem ocorrer, levando de um estado para outro.

² Na *comunicação síncrona* o processo A envia uma mensagem para o processo B e fica esperando que B receba a mensagem. Na *comunicação assíncrona* o processo A envia uma mensagem para o processo B e continua sua execução normalmente. A mensagem enviada por A fica em uma fila até ser recebida por B.

O rótulo de cada arco representa a entrada e a saída gerada, no formato *e/s*. Quando representada por meio de uma tabela de transições, as colunas representam os estados da máquina e as linhas as possíveis entradas. Existem dois modelos que podem ser representados na tabela: o modelo de *Mealy* – interseção da linha com a coluna específica o próximo estado e a saída gerada; e o modelo de *Moore* – interseção da linha com a coluna contém apenas o próximo estado e existe uma coluna separada para indicar a saída associada com cada estado (Davis, 1988). Com o objetivo de aumentar o poder de representação das MEFs, algumas extensões dessa técnica são propostas (Bourhfir et al., 1996): *Máquinas de Estados Finitos Estendidas* (MEFEs), que representam o uso e definição de variáveis associadas às transições; e *Máquinas de Estados Finitos com Comunicação* (MEFCs), que representam os aspectos de comunicação entre MEFs através de canais de comunicação com filas associadas.

A técnica *Statecharts* é uma extensão de MEFs, baseada em três elementos: hierarquia, concorrência e comunicação. Statecharts permite descrever a dinâmica dos sistemas reativos, de forma clara e realística, e ao mesmo tempo formal e rigorosa, de maneira a possibilitar também uma simulação detalhada do sistema (Harel et al., 1987). Na Seção 2.3.1 essa técnica é descrita com maiores detalhes.

Uma área em que tem sido crucial o uso de especificações formais é a descrição e projeto de protocolos de comunicação. Com a definição do modelo de padronização de protocolos de comunicação pela ISO – *International Standard Organization (ISO OSI Reference Model)*, houve a necessidade de utilizar técnicas específicas para a descrição de protocolos e serviços de comunicação. Esse aspecto motivou o estudo para a definição de uma técnica de descrição formal (TDF) padrão para a área de protocolos de comunicação e áreas afins. Entretanto, foi observado

pelo grupo de estudo que muitas abordagens existentes poderiam ser adotadas. Assim, foi decidido que duas abordagens, *Baseada em Máquinas de Estados Finitos* e *Baseada em Álgebra*, seriam estudadas e uma técnica derivada de cada abordagem seria padronizada. Após 8 anos de trabalho, surgiram duas técnicas para descrição formal: Estelle e Lotos (Turner, 1993).

Segundo Bolognesi e Brinksma (1987), a idéia básica de *Lotos – Language of Temporal Ordering Specification* é que sistemas podem ser especificados definindo-se as relações temporais entre as interações que constituem o comportamento externo observável do sistema. Diferentemente do que o nome sugere, essa técnica é baseada na álgebra de processos e não em lógica temporal. Um sistema especificado em Lotos apresenta duas partes: 1) descrição do comportamento dos processos e interações, a qual é baseada em *Calculus of Communication Systems (CCS)* e *Communicating Sequential Processes (CSP)* e 2) descrição das estruturas de dados, a qual é baseada na linguagem de especificação para tipos abstratos de dados *ACTONE*. A técnica Lotos permite descrever o comportamento seqüencial, a concorrência e as comunicações do tipo síncrona e assíncrona (Turner, 1993).

Estelle – Extended State Transition Language é uma técnica de descrição formal que define um sistema como máquinas de estados finitos estendidas estruturadas hierarquicamente. As máquinas de estados se comunicam trocando mensagens através de canais de comunicação bidirecionais. Mensagens recebidas são armazenadas em filas de tamanho infinito. Estelle permite a descrição de paralelismo síncrono e assíncrono entre as máquinas de estado do sistema, permitindo evolução dinâmica do sistema. A especificação pode ser descrita em diferentes níveis de abstração, vindo da forma mais abstrata até aproximar-se da implementação

(Budkowski e Dembinski, 1987; Lopes de Souza, 1989). Na Seção 2.3.2 essa técnica é descrita com maiores detalhes.

Concomitante com a definição de Estelle e Lotos, o órgão CCITT – *Comité Consultatif International Téléphonique et Télégraphique*, o qual produz padrões no campo de telecomunicações, desenvolveu outra TDF chamada *SDL – Specification and Description Language*. A cooperação entre a ISO e CCITT no desenvolvimento de TDFs permitiu que SDL possuísse características de Estelle e Lotos: o conceito de máquinas de estados finitos de Estelle e tipos de dados algébricos de Lotos. SDL possui construções para representar a estrutura, o comportamento, as interfaces, a comunicação, abstrações, encapsulamento de módulos e refinamentos do sistema. Todas as características de SDL foram projetadas para adequarem-se à especificação de sistemas de telecomunicações, incluindo serviços e protocolos (Turner, 1993).

A partir das técnicas de descrição formal Estelle, Lotos e SDL, os órgãos CCITT e ISO têm desenvolvido uma notação padrão para descrever conjuntos de teste para teste de conformidade de padrões OSI. Essa notação é *chamada TTCN – Tree and Tabular Combined Notation*. Dois conjuntos de teste TTCN foram desenvolvidos pela ISO, ambos para o protocolo X.25 (Vuong et al., 1996). O objetivo é fornecer um conjunto de teste padrão que forneça uma certa garantia de que o protocolo implementado apresenta todas as regras definidas pela sua especificação.

Nas próximas seções são comentados os aspectos mais relevantes das técnicas Statecharts e Estelle visto que essas técnicas são alvo da proposta deste trabalho.

2.3.1. Statecharts

Statecharts é uma técnica de especificação formal proposta por Harel et al. (1987) para descrever o aspecto comportamental de sistemas reativos, com o uso de

uma hierarquia de MEFs. Algumas características dessa técnica que podem ser apontadas são (Harel et al., 1987; Harel e Politi, 1998):

- apresentação do modelo na forma de *diagrama de estados*, dispondo assim de todas as vantagens de se utilizar uma representação gráfica. Além disso, os diagramas de estados são apresentados de forma hierárquica e não plana, como ocorrem nas MEFs e MFEs. Isso tem a vantagem de tornar o modelo mais claro, sendo possível visualizar os aspectos de concorrência;
- a *decomposição* permite descrever o modelo do sistema de maneira modularizada e hierárquica, facilitando a representação de transições que ocorrem partindo de diferentes estados para um mesmo estado;
- a ortogonalidade possibilita descrever o paralelismo entre os componentes (MEFs) do modelo especificado. Componentes ortogonais (paralelos) são denominados de estados do tipo AND em oposição aos estados do tipo OR em que apenas um estado pode estar ativo em um dado instante;
- o *broadcasting* ou reação em cadeia permite descrever a sincronização entre os componentes ortogonais do modelo. *Broadcasting* ocorre quando uma transição que possui uma ação associada que é um evento (considerado *evento interno*) é disparada; esse evento interno possivelmente irá disparar transições em outros componentes do modelo, caracterizando assim a reação em cadeia; e
- *história*, que possibilita "lembrar" estados que foram visitados previamente. Quando um estado é ativado, seu estado *default* é ativado, a menos que a transição possua o símbolo *história*. Se isso ocorrer, será ativado o último estado visitado. A história pode ser especificada para ocorrer no primeiro nível de estados, representada por H, ou de forma recursiva, quando um estado é

decomposto em níveis, a história, representada pelo símbolo H^* , irá atuar em todos os níveis da hierarquia de estados.

Os aspectos de sincronização são representados por alguns tipos de condições e eventos que podem ser associados à transição (*evento[condição]/ação*) e que são avaliados no momento em que a transição estiver habilitada, ou seja, apta a ocorrer. Essas condições e eventos têm o objetivo de avaliar ou confirmar a ocorrência de uma dada situação em um componente paralelo que, se confirmada, possibilita que a transição seja disparada (Harel et al., 1987). Por exemplo, têm-se as condições $in(s)$ – verdadeira quando o estado s está ativo; $not-yet(e)$ ou $ny(e)$ – verdadeira caso o evento e ainda não tenha ocorrido; $current(c)$ ou $cr(c)$ – valor corrente da condição c na reação em cadeia e $cr(v)$ – valor corrente da variável ou expressão v na reação em cadeia. Têm-se também os eventos $exit(s)$ ou $ex(s)$ – ocorre quando o estado s deixa de estar ativo; $entered(s)$ ou $en(s)$ – ocorre quando o estado s passa a estar ativo.

Na Figura 2.1 é ilustrado um statechart simples mas que exemplifica as principais características dessa técnica. O estado A (tipo *OR*) é decomposto nos seguintes estados: B , C , D , G , H . O estado D , do tipo *OR*, é decomposto em dois estados: E e F , sendo que o estado F , do tipo *AND*, apresenta dois componentes ortogonais $F1$ e $F2$, ambos do tipo *OR*. Existem duas transições com história: a transição f e a transição d , essa última sendo história recursiva. *Broadcasting* ocorre na transição j/i do componente $F2$. Quando o evento j , ocorre essa transição poderá disparar também a transição i no componente paralelo $F1$, caso o estado $F12$ esteja ativo e i não tenha ocorrido no estado $F1$ no passo corrente.

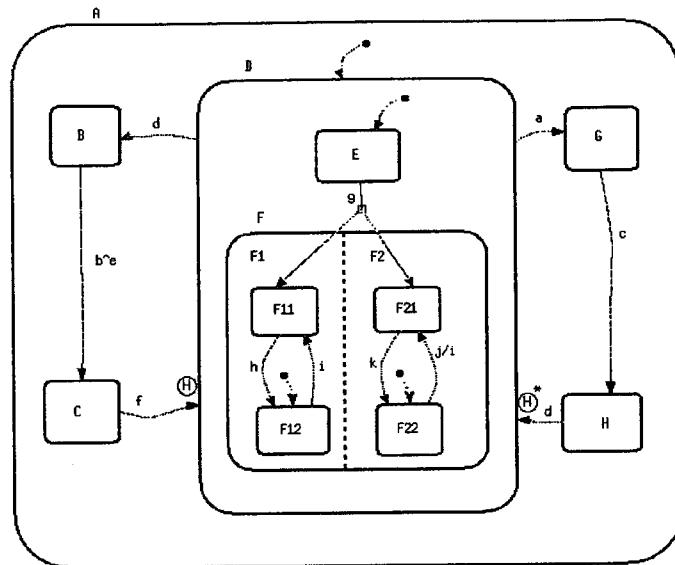


Figura 2.1. Exemplo de um Statechart (Masiero et al., 1994).

Além do poder de representação, a técnica Statecharts também possui sintaxe e semântica formalmente definidas, permitindo uma análise dos aspectos dinâmicos do modelo. Assim, a validação pode ser realizada das seguintes formas: 1) *Simulação Interativa*, quando o usuário gera, passo a passo, eventos que provocam alterações nos estados do sistema; 2) *Simulação em Batch*, quando a simulação é feita utilizando um conjunto de eventos pré-determinados; 3) *Simulação Programada*, quando o sistema pode ser observado sob condições aleatórias, o que é especificado previamente por meio de um programa de controle; 4) *Simulação Exaustiva*, quando o modelo é executado gerando-se todos os possíveis eventos e todas as alterações nos valores de condições e variáveis. Esse tipo de simulação é, em geral, impraticável e deve ser considerada como uma alternativa para partes pequenas e críticas do modelo.

Devido ao poder de representação da técnica Statecharts, ela tem sido investigada em diferentes tipos de aplicações, como por exemplo: para descrição de hardware (Drusinski e Harel, 1989), modelagem de hipertextos (Turine et al., 1997),

modelagem de aplicações hipermídia (Paulo et al., 1997), além da adaptação dessa técnica para sua aplicação dentro do paradigma de orientação a objetos (Coleman et al., 1992; Harel e Gery, 1996). Francês et al. (2000) apresentam uma extensão estocástica para a técnica Statecharts, associando probabilidades de ocorrência às transições. Utilizando cadeias de *Markov* para tratar essas probabilidades, é possível calcular medidas de desempenho em sistemas distribuídos estimando, por exemplo, tempos médios e utilização de recursos.

A seguir, são apresentadas sucintamente as ferramentas de apoio para aplicação da técnica Statecharts.

- **Ferramentas de Apoio para Aplicação da Técnica Statecharts**

Duas ferramentas disponíveis para apoiar a aplicação da técnica Statecharts são: *Statemate* e *StatSim*. *Statemate* é uma ferramenta comercial que permite a edição, simulação e análise de especificações Statecharts. (Harel et al., 1990).

O ambiente *StatSim – Statecharts Simulator* foi desenvolvido no Instituto de Ciências Matemáticas e de Computação - USP, pelo grupo de Engenharia de Software, e permite a edição e simulação de Statecharts, tanto na forma textual como na forma gráfica. É possível também a análise de propriedades da especificação através da árvore de alcançabilidade (Masiero et al., 1991; Fortes, 1991; Batista, 1991; Boaventura, 1992; Cangussu, 1993).

2.3.2. Estelle

Estelle – Extended State Transition Language é uma técnica de descrição formal desenvolvida pela ISO, para especificação de sistemas distribuídos, protocolos de

comunicação e serviços. O padrão ISO 9074 (1987) descreve a semântica e sintaxe dessa técnica.

Um sistema especificado em Estelle é estruturado em uma hierarquia de módulos que se comunicam através de troca de mensagens. O comportamento de cada módulo é descrito através de uma MEFE e utiliza, com algumas restrições, a linguagem Pascal (com algumas restrições), o que torna Estelle uma técnica de fácil aprendizagem. As mensagens recebidas pelos módulos são armazenadas em filas de tamanho infinito (tipo *FIFO – first in first out*) e são processadas de acordo com as condições, prioridades e atrasos associados às transições da MEFE.

A arquitetura de uma especificação em Estelle pode ser descrita através de três componentes principais: *módulos*, *canais* e *estrutura*. Na hierarquia de módulos existe o conceito de módulo pai e módulos filhos, sendo que o módulo pai sempre tem prioridade sobre os filhos. Isso significa que, se o módulo pai tiver transições aptas a ocorrerem em um passo, ele irá disparar uma de suas transições. Isso previne problemas de sincronização entre os módulos, visto que o módulo pai pode compartilhar as variáveis exportadas pelo módulo filho. Essa característica permite, por exemplo, que o módulo pai crie e finalize, dinamicamente, instâncias de módulos filhos.

A maneira em que os módulos são decompostos, sua comunicação e como esses módulos são iniciados e instanciados, caracteriza a estrutura da especificação. Em uma especificação Estelle, podem ser definidas várias instâncias de módulos e conexões entre módulos. Essas instâncias podem ser criadas de forma estática ou dinâmica. A estrutura estática é estabelecida quando a especificação é iniciada e não pode ser modificada em tempo de execução. A estrutura dinâmica é criada em tempo

de execução, de modo que as instâncias são criadas ou finalizadas com o disparo de transições.

Na Figura 2.2 é apresentada uma arquitetura de um sistema especificado em Estelle. Nesse exemplo, a especificação *Spec* possui dois módulos *U* e *M*. O módulo *U* possui duas instâncias *U[1]* e *U[2]* e o módulo *M* possui dois sub-módulos: módulo *M_I* e *M₂*. Para *M_I* existem duas instâncias: *M_I[1]* e *M_I[2]*. Existem canais de comunicação interligando os módulos *U[1]* com *M* e *U[2]* com *M* (duas instâncias do mesmo canal); e canais interligando *M_I[1]* e *M_I[2]* com *M₂*.

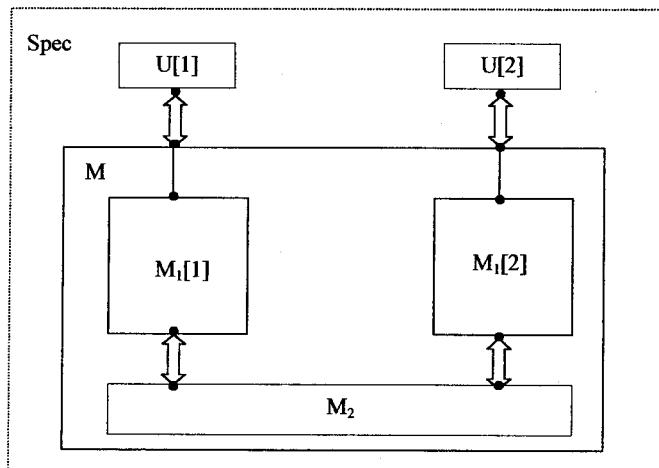


Figura 2.2. Arquitetura de uma Especificação Descrita em Estelle
(Lopes de Souza, 1989).

Em Estelle, é possível descrever o paralelismo síncrono, assíncrono e execução seqüencial dos módulos do sistema. Essas diferentes possibilidades de comunicação e sincronização são definidas através do atributo de cada módulo: *systemprocess*, *systemactivity*, *process* e *activity*.

Módulos com atributo *systemprocess* ou *systemactivity* são chamados *system* e entre eles ocorre paralelismo assíncrono. Dentro de um módulo *systemprocess*, pode ocorrer paralelismo síncrono ou execução seqüencial, dependendo do atributo de

seus módulos filhos, que pode ser *process* ou *activity*. Os filhos de um módulo *process* podem ser *process* ou *activity* e são executados sincronamente em paralelo; isso significa que uma transição de cada módulo é selecionada em cada passo e essas transições são executadas em paralelo. Por outro lado, os filhos de um módulo *activity* podem somente possuir o atributo *activity* e são executados seqüencialmente, ou seja, uma transição de um dos módulos é selecionada aleatoriamente para execução em cada passo. No caso de módulos *systemactivity*, seus módulos filhos podem possuir somente o atributo *activity* e com isso só ocorre execução seqüencial (Budkowski e Dembinski, 1987).

A técnica Estelle foi formulada para ser, na medida do possível, independente do tempo de execução, o qual fica a cargo do implementador. Entretanto, algumas transições espontâneas (que possuem evento nulo) podem possuir a cláusula *delay*, que estabelece um atraso mínimo e máximo para o disparo da transição. Esse atraso é definido dinamicamente por valores que indicam o número de unidades de tempo que a transição deve permanecer atrasada.

A semântica de Estelle permite que seja determinado o comportamento global do sistema especificado, definindo como as transições são selecionadas a partir dos módulos e quais são as regras para disparo dessas transições. O comportamento global de um sistema especificado em Estelle é definido pelo conjunto de todas as possíveis seqüências de *situações globais* geradas a partir de uma situação inicial, sendo que o disparo de uma transição faz com que o sistema mude de uma situação global para outra. Portanto, a semântica operacional de Estelle descreve de que forma essas seqüências de situações globais são geradas, ou seja, a maneira que as transições podem se intercalar para modelar o paralelismo entre os módulos do sistema. Como apontado por Budkowski e Dembinski (1987), essa semântica

operacional de Estelle auxilia na definição de ferramentas de apoio para a especificação dessa técnica, como por exemplo, simuladores, depuradores, interpretadores, compiladores, dentre outras.

A seguir, são descritas brevemente algumas ferramentas de apoio para a aplicação da técnica Estelle.

- **Ferramentas de Apoio para Aplicação da Técnica Estelle**

As ferramentas que fornecem apoio para a aplicação da técnica Estelle podem ser divididas em dois grupos:

- 1) ambientes que auxiliam na implementação de especificações Estelle, por exemplo, as ferramentas *PetDingo* e *EDT*; e
- 2) ambientes para verificação e validação de especificações Estelle, por exemplo, as ferramentas *ESTIM*, *EVEN*, *TESTVAL* e *TESTGEN*.

A ferramenta *PetDingo* foi desenvolvida nos Estados Unidos pelo *NIST – National Institute of Standards and Technology* (Sijelmasi, Strausser, 1990). É uma ferramenta de domínio público distribuída em um pacote contendo duas ferramentas principais: *PET - Portable Estelle Translator* e *Dingo – Distributed Implementation GeneratOr*. *PET* gera uma representação estática da especificação a partir da sua descrição, verificando sua sintaxe e semântica. O arquivo gerado pode ser processado por várias ferramentas para diferentes aplicações, como geração de código, tabelas de referências cruzadas, ou ferramentas de validação (Sijelmasi e Strausser, 1991a). A partir dos arquivos gerados pela ferramenta *Pet*, *Dingo* gera código na linguagem C++. As implementações geradas são distribuídas em diferentes processos do sistema e um protocolo de sincronização é utilizado para implementar a comunicação

entre esses processos de acordo com a semântica de Estelle (Sijelmasi e Strausser, 1991b).

A ferramenta *EDT – Estelle Development Toolset* foi desenvolvida na França, pelo *Institut National des Télécommunications*, sendo composta de um conjunto de ferramentas que permitem o desenvolvimento de implementações de sistemas especificados em Estelle (EDT, 2000). A maioria das ferramentas que constituem a *EDT* foi especificada no escopo do projeto europeu *ESPRIT – SEDOS – Software Environment for Distributed Open Systems*, e após o término do projeto muitas funcionalidades foram adicionadas à ferramenta. *EDT* permite: 1) geração de código C a partir da especificação; 2) simulação/depuração da especificação, fornecendo mecanismos para executar a especificação; 3) geração de tabelas de eventos e estados; e 4) geração de *drivers* para testes, que, a partir de um módulo *x* que está sendo descrito, gera *drivers* dos módulos que interagem com *x* (simulando o ambiente), possibilitando que sejam observadas as reações do módulo *x* com o ambiente. *EDT* permite que usuários possam projetar ferramentas que utilizam informações geradas por elas. Rolinski e Wytrebowicz (1998) apresentam duas extensões para a ferramenta *EDT*, definindo um analisador estático para especificações Estelle e um gerador de interações aleatórios, que possibilita a seleção aleatória de interações e de valores dos parâmetros de interações, fornecendo recursos para o gerador de *drivers* para testes.

Templemore-Finlayson et al. (1998) apresentam uma representação gráfica para Estelle, com base na representação gráfica da técnica SDL. Uma ferramenta, chamada *Estelle/GR*, apóia a utilização dessa representação gráfica para Estelle, possibilitando a geração da representação gráfica de Estelle a partir da descrição

textual da especificação, como também a obtenção da descrição textual a partir da representação gráfica.

Thees e Gotzhein (1998) apresentam *XEC – eXperimental Estelle Compiler*, uma ferramenta para geração automática de implementações em C++ a partir da técnica Estelle. Além da geração automática de implementações, essa ferramenta oferece outros recursos como, por exemplo, avaliação de desempenho e otimização de implementações.

Apesar de a técnica Estelle utilizar comandos da linguagem Pascal, a construção de uma ferramenta para validar especificações descritas nessa técnica é bastante diferente de uma ferramenta para validar programas escritos em Pascal. Isso ocorre devido às características de Estelle: seu comportamento é baseado na ocorrência de transições e, principalmente, pelo aspecto dinâmico que pode existir.

Courtiat e Saqui-Sannes (1992) descrevem a ferramenta *ESTIM – Estelle Simulator Based on an Interpretative Machine*, que permite a verificação de protocolos especificados em *Estelle**, uma variante de Estelle que utiliza o mecanismo de *rendezvous* (indica que a troca de mensagem entre dois processos é feita por uma sincronização de ambos, seguida pela troca de informações). A ferramenta restringe as especificações Estelle, considerando somente especificações contendo módulos com atributo *systemactivity*. Isso significa que o paralelismo síncrono não é considerado pela ferramenta. A verificação utiliza um grafo de alcançabilidade que é gerado a partir de eventos selecionados pelo usuário, ou seja, eventos que se desejam observar. Esse aspecto permite a construção de um grafo de alcançabilidade parcial, tentando diminuir o problema de explosões de estados. A partir do grafo, duas ferramentas, chamadas *PIPN* e *ALDEBARAN*, que são analisadores de sistemas de transições, são utilizadas para geração de um autômato

(quotient automaton) da especificação. Os resultados obtidos com esse autômato são comparados com os resultados esperados pelo usuário.

Vuong et al. (1994) descrevem um ambiente para seleção e geração de conjuntos de casos de teste para protocolos especificados em Estelle, chamado *TESTGEN – Test Generation and Selection Environment*. O método de geração de casos de teste é baseado em restrições e integra teste de fluxo de dados e de controle. As restrições são expressões booleanas que precisam ser satisfeitas para que o caso de teste seja gerado. São definidas restrições sobre estados, sobre variáveis (usos e definições) e sobre os construtores de tempo utilizados pelo protocolo. Por exemplo, a restrição sobre estados estabelece o número máximo e mínimo de vezes que cada estado pode ser alcançado pelo caso de teste. As restrições podem se sobrepor ou ser mutuamente exclusivas. A especificação Estelle do protocolo descrita em um único módulo (sem hierarquia de módulos) é transformada em um sistema de transições estendido (STE), de modo que os casos de teste são gerados a partir do STE e do conjunto de restrições definido.

Vuong et al. (1996) descrevem a ferramenta *TESTVAL – Test Validator* para auxiliar no teste de conformidade de protocolos, validando conjuntos de casos de teste no formato *TTCN (Tree and Tabular Combined Notation)*. TTCN é notação padrão para descrever conjuntos de teste para teste de conformidade de padrões OSI. A ferramenta *TESTGEN* é utilizada para produzir uma representação da especificação e a validação dos conjuntos de casos de testes é feita através de simulações e métodos de avaliações simbólicas. Um subconjunto da técnica Estelle é utilizado, denominado *Estelle.Y*, que é restrito a um único módulo e com a cláusula *delay* modificada para ficar consistente com os casos de teste especificados em

TTCN. Entretanto, os autores não descrevem quais características da cláusula *delay* que são modificadas.

Jirachiefpattana e Lai (1997) descrevem a ferramenta *EVEN – Estelle Verification Environment using NPNs*, a qual faz a verificação de especificações Estelle traduzindo-as em Redes de Petri Numérica (NPN), uma extensão de Redes de Petri que permite modelar, dentre outras coisas, variáveis e condições. O comportamento dinâmico, as variáveis exportadas e a maioria dos comandos de Estelle são tratados por essa ferramenta. As deficiências são que as cláusulas *priority* e *delay* e o atributo *systemprocess* não podem ser modelados. *EVEN* utiliza a ferramenta *Pet* do *NIST* para gerar o modelo estático da especificação e checar a sintaxe e semântica da especificação. A verificação utiliza a árvore de alcançabilidade de Redes de Petri Numéricas, proposta por Godefroid et al. apud Jirachiefpattana e Lai (1997)³.

Em síntese, pode-se observar que as ferramentas que apoiam a verificação e validação de especificações em Estelle fazem algum tipo de restrição quanto às características de Estelle. As principais características de Estelle não abordadas por essas ferramentas são: paralelismo síncrono, cláusula *delay* e *priority* das transições e o aspecto dinâmico da especificação. Além disso, nenhuma das ferramentas fornece medidas de cobertura para quantificar as atividades de verificação e validação de especificações Estelle.

Um mecanismo para analisar especificações formais é a árvore de alcançabilidade. A árvore de alcançabilidade é uma representação da especificação utilizada para analisar as propriedades da especificação (as propriedades são

³ GODEFROID, P.; HOLZMANN, G.J.; PIROTTIN, D.; State Space Caching Revisited. In: *IV International Workshop of Computer Aided Verification*, 1992.

discutidas a seguir). Dada a sua importância no contexto desta tese, as definições e limitações da árvore de alcançabilidade são apresentadas na próxima seção.

2.3.3. A Árvore de Alcançabilidade para Análise de Especificações Formais

A árvore de alcançabilidade, que consiste em gerar o comportamento possível do sistema modelado, tem sido utilizada com sucesso em algumas áreas de aplicação, como por exemplo, protocolos de comunicação, para validação e análise de propriedades do modelo (Pezzè et al., 1995). A árvore de alcançabilidade é uma das principais técnicas para análise de Redes de Petri (Murata, 1984), sendo utilizada também para análise de sistemas especificados através de MEF, MEFE, Statecharts e Estelle (Masiero et al., 1994; Huang e Hsu, 1994; Huang et al., 1997; Jirachiefpattana e Lai, 1997; Barnard, 1998).

Informalmente, pode-se dizer que a árvore de alcançabilidade é formada pelo conjunto de estados (ou configurações) possíveis do sistema. A partir do estado (configuração) inicial do sistema, todas as transições disparáveis são representadas, juntamente com os estados alcançados. Para cada novo estado inserido na árvore, são obtidas as transições disparáveis e os estados alcançados e assim, sucessivamente, até que todos os estados alcançáveis, a partir do estado inicial, sejam representados.

A utilização da árvore de alcançabilidade possibilita que algumas propriedades dinâmicas do sistema sejam investigadas. Masiero et al. (1994) e Barnard (1998) apresentam algumas propriedades:

- *validade de uma seqüência de eventos:* uma seqüência de eventos é válida se cada evento leva ao disparo de uma transição, produzindo uma mudança de configuração do sistema;
- *alcançabilidade de estados globais:* um estado global S_k é alcançável a partir de um estado global S_i se existe no mínimo uma seqüência de eventos válida seq_j e uma possível seqüência de estados intermediários tal que S_k possa ser obtida a partir de S_i , percorrendo a seqüência seq_j ;
- *reiniciabilidade:* um modelo é reiniciável quando para cada estado global S_i existe uma seqüência de eventos válida que levam ao estado global inicial S_0 ;
- *existência de deadlock:* um modelo possui *deadlock* se sua execução pode alcançar um estado global a partir da qual o sistema não evolui, porque não existe nenhuma transição que possa ser disparada; e
- *uso de transições:* uma transição é usada se ela aparece em, no mínimo, um caminho da árvore de alcançabilidade.

Masiero et al. (1994) definem a árvore de alcançabilidade para Statecharts com base na árvore de alcançabilidade convencional utilizada para análise de Redes de Petri (Murata, 1984). A construção da árvore de alcançabilidade considera todas as seqüências de computações possíveis para os Statecharts. Em cada passo, para cada transição apta para disparar, é suposto que a expressão do evento é verdadeira levando a uma nova configuração, sendo que as condições das transições também são consideradas no disparo das mesmas. As regras semânticas para calcular cada passo são baseadas na semântica definida por Harel et al. (1987). Os aspectos formais da definição da árvore de alcançabilidade para Statecharts são apresentados no Capítulo 4.

O documento ISO 9074 (1987), que descreve a técnica Estelle, define o comportamento global da especificação que, como apresentado por Budkowski e Dembinski (1987), representa a evolução dinâmica do sistema especificado. O comportamento global da especificação é formado por todas as **situações globais** do sistema, sendo que duas situações globais consecutivas correspondem à execução de uma transição. Cada **situação global** é formada pelas **situações locais** dos módulos do sistema e pelas transições selecionadas para execução; sendo que cada situação local de um módulo P é formada pelo estado atual de P e pela transição oferecida por P para execução. Jéron e Jard (1993) definem um sistema de transições que representa os aspectos de filas entre canais de comunicação e que pode ser utilizado para representar o comportamento dinâmico de especificações Estelle.

Muitas ferramentas para verificação de protocolos de comunicação têm sido construídas baseando-se na análise de alcançabilidade, visto a facilidade de automatização e eficácia dessa técnica (Lin et al., 1988; Chu e Liu, 1989; Courtiat e Saqui-Sannes, 1992; Huang e Hsu, 1994; Lai e Jirachiefpattana, 1996; Huang et al., 1997; Buchholz e Kemper, 1997; Jirachiefpattana e Lai, 1997; Nepommiaschy, et al., 1998; Cheung e Kramer, 1999). Entretanto, sua aplicabilidade é comprometida devido ao problema de explosão de estados, pois mesmo considerando sistemas pequenos, o número de estados possíveis pode ser muito grande, gerando um alto custo para construção da árvore de alcançabilidade. Procurando minimizar esse problema, pesquisadores têm apresentado algumas soluções para reduzir o tamanho da árvore de alcançabilidade, durante a sua construção. Observa-se que o problema de explosão de estados é investigado há algum tempo na área de protocolos de comunicação: Lin et al. (1988) apresentam uma revisão de várias estratégias para

redução propostas na década de 80, as quais são aplicadas em ferramentas para verificação de protocolos.

A seguir, são discutidas algumas técnicas para redução do tamanho da árvore de alcançabilidade. Essas técnicas são aplicadas durante a construção da árvore de alcançabilidade.

• Técnicas de Redução Durante a Construção da Árvore de Alcançabilidade

Lin et al. (1988) apresentam a estratégia *PROVAT – Protocol Validation Testing* para redução, durante a construção, do tamanho da árvore de alcançabilidade. Nessa estratégia são utilizadas heurísticas para determinar quais estados globais dos protocolos serão expandidos, quais transições serão disparadas, e quais estados globais serão descartados.

Barnard (1998) cita algumas técnicas de redução que podem ser aplicadas durante a construção da árvore de alcançabilidade:

- *nós duplicados* – utilizada para representar estados repetidos na árvore.

Quando um estado já existente é inserido, ele é considerado apenas um *link* para a primeira ocorrência desse estado, não sendo gerados novamente seus estados sucessores.

- *stubborn sets* – trata transições disparáveis que são independentes entre si, ou seja, transições que podem ser disparadas em qualquer ordem antes de ser obtido o próximo estado. Ao invés de serem consideradas todas as possíveis combinações dessas transições, apenas uma das possibilidades é considerada, chamada *stubborn set*.

- *conjunto de componentes* – considera apenas alguns componentes do sistema modelado durante a construção da árvore de alcançabilidade. Por exemplo, no

contexto de Statecharts, os componentes poderiam ser obtidos pela própria hierarquia de estados do modelo ou a partir dos componentes ortogonais em um determinado nível de hierarquia.

- *boundedness (limitador w)* – é utilizada para tratar as variáveis que fazem parte dos estados (ou configurações) da árvore de alcançabilidade. Para alguns tipos de variáveis (por exemplo, variáveis inteiros), os valores possíveis podem ser infinitos, levando também a um número infinito de estados do sistema. O *limitador w* restringe o número infinito de valores das variáveis para um valor único *w* que representa todos os possíveis valores; com isso o número infinito de estados é reduzido para um estado. Isso é muito útil para sistemas que possuem o mesmo comportamento independentemente dos valores das variáveis, entretanto, isso pode prejudicar a análise de sistemas que dependam dos valores dessas variáveis.

Chu e Liu (1989) apresentam uma técnica de redução para análise de alcançabilidade de protocolos especificados através de MEFEs. Essa técnica requer uma análise de fluxo de dados nas MEFEs para obter informações sobre as *variáveis mortas*. Para ilustrar o conceito de variável morta, considere que *x* é uma variável da MFE *M* e *s* é um estado de *M*. A variável *x* é morta em *s* se, a partir de *s*, *x* não é usada (ou referenciada) em nenhum dos possíveis caminhos de *M*, ou o próximo acesso a *x* é uma definição. Caso contrário, *x* é considerada uma variável viva. O comportamento de *M* a partir do estado *s* não depende do valor de *x* que é morta em *s*. Isso ocorre porque as variáveis vivas não serão influenciadas ou trocadas pela variável morta. Desse modo, dois estados globais são considerados equivalentes quando seus elementos são idênticos exceto, possivelmente, para os valores das variáveis mortas. Isso significa que o comportamento futuro desses estados globais é

o mesmo e somente um deles necessita ser considerado durante a análise de alcançabilidade.

Jéron e Jard (1993) propõem uma técnica de redução para sistemas que utilizam filas, como por exemplo, sistemas especificados em Estelle, SDL e em MEFC (MEFE com comunicação). Nessa técnica, durante a construção da árvore de alcançabilidade, são identificadas seqüências de transições que podem ser infinitamente repetidas e que, consequentemente, aumentam o tamanho da fila do canal de comunicação, resultando em uma árvore de alcançabilidade infinita. Essas seqüências de transições são detectadas pela determinação de uma relação, chamada *relação binária*, entre os dois estados globais que estão no início e no final da seqüência de transições. Informalmente, essa relação binária identifica dois estados globais S_1 e S_2 entre uma seqüência de transições, sendo que todos os elementos de S_1 e S_2 são idênticos, a menos do conteúdo das filas, em que o conteúdo das filas de S_2 é um prefixo ordenado do conteúdo das filas de S_1 . Essa seqüência de transições é identificada e marcada de forma que a árvore não seja mais explorada a partir dessa seqüência de transições. Segundo os autores, a técnica precisa de adaptações para ser aplicada em protocolos especificados em Estelle. Isso é necessário devido à cláusula *priority* que pode existir nas transições, a qual estabelece que o disparo de uma transição não depende somente do estado do sistema mas também das outras transições habilitadas. Com isso, o estado global e o prefixo ordenado são redefinidos de forma que sejam consideradas também as transições disparáveis de cada estado global.

Nesta tese é investigado o uso da árvore de alcançabilidade como um mecanismo para apoiar a seleção de seqüências de teste para especificações baseadas em Statecharts e em Estelle. Conforme será visto no Capítulo 4, a construção da

árvore de alcançabilidade para Statecharts e Estelle considera algumas das técnicas de redução discutidas nesta seção.

Na próxima seção, é discutida a atividade de teste de software. São apresentados os principais critérios de teste definidos para o teste de programas e um resumo dos resultados obtidos com a realização de estudos empíricos para a avaliação de critérios de teste.

2.4. Técnicas e Critérios de Teste de Programas

O processo de desenvolvimento de software envolve uma série de atividades em que, apesar dos métodos, técnicas e ferramentas empregados, erros no produto ainda podem ocorrer. Enganos podem ocorrer desde o início do desenvolvimento, por exemplo, especificando erroneamente os requisitos do sistema, como também nos estágios finais através da inserção de defeitos no projeto ou na implementação do sistema. Com isso, a etapa de teste é de grande importância para a identificação e eliminação de erros que persistem, representando a última revisão da especificação, projeto e codificação (Pressman, 2000).

Segundo Myers (1979), teste é o processo de executar um programa com o objetivo de encontrar erros e compreende os seguintes passos: 1) construção do conjunto de casos de teste; 2) execução do programa P com esse conjunto de casos de teste e 3) análise do comportamento de P para determinar se o mesmo está correto ou não. Esses passos se repetem até que se tenha confiança de que P realiza o esperado com o mínimo de erros possível.

A atividade de teste pode ser caracterizada em 3 níveis (ou fases), usualmente presentes em uma estratégia: *teste de unidade* – que concentra esforços nos módulos

do programa, os quais são as menores unidades do projeto de software, procurando identificar erros de lógica e de implementação; *teste de integração* – que visa a identificar erros na interação entre os módulos, sendo uma técnica sistemática para integrar os módulos que compõe a estrutura do software e *teste de sistema* – que é realizado após a integração do sistema e visa a identificar erros de funções e características de desempenho, que não estejam de acordo com a especificação.

O sucesso das atividades de teste e validação está relacionado com a qualidade do conjunto de casos de teste. Em princípio, o software deveria ser exercitado com todos os valores possíveis do domínio de entrada. Sabe-se, porém, que esse tipo de teste, conhecido como teste exaustivo, é impraticável devido a restrições de tempo e custo para sua realização. Nessa perspectiva, duas questões importantes são:

- *Como selecionar os casos de teste?*
- *Como garantir que um programa foi suficientemente testado?*

Para responder a essas questões definem-se: 1) *método de seleção de dados de teste*, como um procedimento para escolher casos de teste, e 2) *critério de adequação dos dados de teste* como um procedimento usado para avaliar um conjunto de casos de teste. Existe uma forte correspondência entre métodos de seleção e critérios de adequação, pois dado um critério de adequação C , existe um método de seleção M_c que estabelece: “Selecione um conjunto de teste que satisfaça o critério C ”. De forma análoga, dado um método de seleção M , existe um critério de adequação C_m que determina: “Um conjunto de dados de teste é adequado se ele foi gerado pelo método M ”. Desse modo, costuma-se utilizar o nome de critério de teste para designar as duas atividades (Maldonado, 1991).

Critérios de teste têm sido elaborados com o objetivo de fornecer uma maneira sistemática e rigorosa para selecionar um conjunto de teste e ainda assim ser

eficiente para apresentar os erros existentes, respeitando as restrições de tempo e custo associados a um projeto de software. Esses critérios são classificados em três técnicas de teste: *Funcional*, *Estrutural* e *Baseada em Erros*. A diferença entre essas três técnicas é a origem da informação usada para avaliar ou para construir os conjuntos de teste, de forma que cada técnica possui um conjunto de critérios de teste utilizados para esse fim. Além disso, nenhuma das técnicas de teste é completa, no sentido que nenhuma delas é, em geral, suficiente para garantir a qualidade da atividade de teste. Na verdade, essas diferentes técnicas se complementam e devem ser aplicadas em conjunto para assegurar-se um teste de boa qualidade (Maldonado, 1991). Essas três técnicas são descritas a seguir, dando-se ênfase aos critérios de Fluxo de Controle e o Teste de Mutação, os quais são utilizados neste trabalho, e aos critérios de Fluxo de Dados, dada a sua importância na atividade de teste de software e a perspectiva de se estender o presente trabalho com esses critérios.

• Técnica Funcional

Essa técnica também é conhecida como teste caixa preta (Myers, 1979), pelo fato de tratar o software como uma caixa em que o conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. As funções do sistema, identificadas a partir de sua especificação, são verificadas sem se preocupar com detalhes de implementação. Dessa forma, é essencial que a especificação esteja correta e de acordo com os requisitos do usuário (DeMillo, 1987).

Alguns critérios de teste funcional são:

- **Particionamento em Classes de Equivalência:** divide-se o domínio de entrada de um programa em classes de equivalência válidas e inválidas, a

partir das condições de entrada identificadas no programa. Seleciona-se o menor número possível de casos de teste, partindo-se do princípio que um elemento de uma classe é representativo para toda a sua classe. O uso de particionamento permite examinar os requisitos com mais detalhes e restringir o número de casos de teste existentes.

- **Análise do Valor Limite:** complementa o critério Particionamento em Classes de Equivalência, testando os limites de cada classe de equivalência. Ao invés de serem selecionados quaisquer elementos para cada classe, os casos de teste são selecionados das fronteiras de cada classe, pois é nesses casos que os erros costumam ocorrer com mais freqüência (Pressman, 2000).
- **Grafos de Causa e Efeito:** explora a combinação das condições de entrada do programa. São identificadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) as quais são combinadas dando origem a um grafo. A partir desse grafo é criada uma tabela de decisão a partir da qual são derivados os casos de teste.
- **Error Guessing:** nesse critério os possíveis erros são listados, com base nas características e no domínio de entrada do software, e os casos de teste são elaborados baseando-se nessa listagem.

Em geral, o teste funcional é uma técnica sujeita às inconsistências que podem ocorrer na especificação (DeMillo, 1987). Outro problema com essa técnica é a dificuldade de quantificar a atividade de teste, visto que não se pode garantir que partes essenciais ou críticas do programa sejam executadas. Como consequência, tem-se dificuldade em automatizar a aplicação de tais critérios, que ficam, em geral, restritos a aplicação manual. Porém, para aplicação desses critérios é essencial somente que se identifiquem as entradas, a função a ser computada e a saída do

programa, o que os torna aplicáveis praticamente em todas as fases de teste: unidade, integração e sistema (Delamaro, 1997).

- **Técnica Estrutural**

A técnica estrutural, também conhecida como teste caixa branca (em oposição à caixa preta), utiliza aspectos de implementação para derivar os requisitos de teste, baseando-se no conhecimento da estrutura interna da implementação (Myers, 1979).

A maioria dos critérios dessa técnica utiliza uma representação de programa conhecida como *Grafo de Fluxo de Controle* ou *Grafo de Programa* (Rapps e Weyuker, 1985). Um grafo de fluxo de controle é um grafo orientado sendo que cada vértice (ou nó) representa um bloco indivisível de comandos e cada aresta representa um desvio de um bloco para outro. Um bloco desse tipo tem as seguintes características: não existem desvios para o meio do bloco e uma vez que o primeiro comando do bloco seja executado, todos os demais comandos do bloco são executados seqüencialmente. Através do grafo de programa, escolhem-se os componentes que devem ser executados, caracterizando assim o teste estrutural.

Os critérios estruturais baseiam-se em tipos de estruturas diferentes para determinar quais partes do programa são requeridas na execução. Os critérios de teste estrutural são classificados em (Beizer, 1990):

- **Critérios Baseados na Complexidade:** utilizam informações sobre a complexidade do programa para determinar os requisitos de teste. Um critério bastante conhecido dessa classe é o *critério de McCabe* que utiliza a *complexidade ciclomática*⁴ para derivar os requisitos de teste.

⁴ Complexidade Ciclomática é uma medida que define o número de caminhos independentes no programa e, com isso, o número de casos de testes que devem ser elaborados para garantir que todos os comandos do programa sejam executados no mínimo uma vez (Pressman, 2000).

Essencialmente, esse critério requer que seja executado um conjunto de caminhos linearmente independente do grafo de programa (Pressman, 2000).

- **Critérios Baseados em Fluxo de Controle:** utilizam apenas características de controle da execução do programa, como comandos ou desvios, para derivar os requisitos de teste. Os critérios *Todos-Arcos*, *Todos-Nós* e *Todos-Caminhos* são os mais conhecidos dessa classe. Esses critérios são descritos com mais detalhes a seguir.
- **Critérios Baseados em Fluxo de Dados:** utilizam, como o próprio nome já diz, informações sobre o fluxo de dados do programa para derivar os requisitos de teste. Essa classe de critérios requer que sejam testadas as interações que envolvem definições de variáveis e referências a essas definições. Exemplos de critérios dessa classe são os critérios de Rapps e Weyuker (1985), de Ural e Yang (1988) e os critérios *Potenciais-Usos* (Maldonado, 1991). Dada a importância desses critérios na atividade de teste de programas, esses critérios são descritos com mais detalhes a seguir.

O conjunto de casos de teste obtido durante a aplicação dos critérios funcionais pode corresponder ao conjunto inicial para os testes estruturais. Como, em geral, o conjunto de casos de teste funcional não é suficiente para satisfazer totalmente um critério de teste estrutural, novos casos de teste são gerados e adicionados ao conjunto até que se consiga o grau de satisfação desejado, explorando-se, dessa forma, os aspectos complementares das duas técnicas de teste (Souza, 1996).

Conforme referenciado por Vergílio et al. (1993), um problema relacionado ao teste estrutural é a impossibilidade de se determinar automaticamente se um caminho é ou não executável, ou seja, não existe um algoritmo que dado um caminho completo qualquer decida se o caminho é executável e forneça o conjunto

de valores que causam a execução desse caminho. Sendo assim, é necessária a intervenção do testador para determinar quais são os caminhos não executáveis para o programa sendo testado.

2.4.1. Critérios de Fluxo de Controle

Conforme descrito anteriormente, esses critérios utilizam informações de controle do programa, obtidas a partir do *Grafo de Programa*, para derivar os requisitos de teste. Os critérios de fluxo de controle mais conhecidos são (Zhu et al., 1997):

- critério *Todos-Nós*: requer que todos os comandos do programa sejam executados no mínimo uma vez pelo conjunto de casos de teste. Isso significa cobrir todos os nós ou vértices do grafo de programa.
- critério *Todos-Arcos*: requer que toda transferência de controle do programa, ou todas as arestas do grafo de programa, sejam exercitadas pelo menos uma vez pelo conjunto de casos de teste. Entretanto, mesmo que todos os arcos do programa sejam testados não significa que todas as combinações de transferência de controle são testadas.
- critério *Todos-Caminhos*: requer que todos os caminhos possíveis do programa sejam executados, incluindo todas as combinações de arcos, ou de transferência. Um número infinito de caminhos pode existir em programas com comandos de repetição (*loops*), tornando esse critério impraticável.

Os critérios a seguir são similares ao critério *Todos-Caminhos*, entretanto apresentam algum tipo de restrição para selecionar os caminhos do programa a serem testados (Zhu et al., 1997):

- critério *Todos-Caminhos-Completos* – um *caminho completo* é um caminho P em que o primeiro nó do caminho é o nó inicial e o último nó é o nó final do grafo do programa. Esse critério requer que todos os *caminhos completos* sejam executados no mínimo uma vez pelo conjunto de casos de teste.
- critério *Todos-Caminhos-Simples* – um *caminho simples* é um caminho P tal que todos os nós que compõem esse caminho, exceto possivelmente o primeiro e o último, são distintos. Esse critério requer que todos os *caminhos simples* sejam executados no mínimo uma vez pelo conjunto de casos de teste.
- critério *Todos-Caminhos-Livres-Laços* – um *caminho livre de laço* é um caminho simples P tal que todos os nós são distintos, inclusive o primeiro e o último. Esse critério requer que todos os *caminhos livres de laços* sejam executados no mínimo uma vez pelo conjunto de casos de teste.

Comandos de repetição ou laços aumentam a complexidade do programa e ocasionam um número infinito de caminhos para serem testados (Zhu et al., 1997).

O critério a seguir é relacionado a programas contendo laços:

- Critério (*loop count- k*) – para um número inteiro k , esse critério requer que todo laço do programa em teste seja executado no máximo k vezes pelo conjunto de casos de teste (Howden, 1975).

Outros critérios de teste concentram-se nas condições dos programas:

- Critério *Todas-Decisões* – requer que toda condição existente nos comandos de decisão do programa, por exemplo, comandos *if*, *while*, seja avaliada com o valor verdadeiro e com o valor falso, ou seja, um conjunto de casos de teste adequado a esse critério possui um caso de teste em que a condição x é verdadeira e outro caso de teste em que x é falsa.

- Critério *Todas-Múltiplas-Condições* – esse critério é uma extensão do critério *Todas-Decisões* e considera as condições compostas por vários predicados, combinados por conectores como *and*, *or* ou *not*. Esse critério requer que um conjunto de casos de teste execute todas as possíveis combinações de valores verdadeiros dos predicados de todas as condições do programa (Myers, 1979).

Segundo Howden (1987), mesmo que se limite o número de iterações de um laço a duas ou três vezes, ainda assim, geralmente ter-se-á um número bastante grande de caminhos a serem testados. Nesse sentido, os critérios de Fluxo de Dados são mais seletivos pois, além de promoverem a concatenação de arcos, conduzem à seleção de caminhos com uma maior relação com os aspectos funcionais do programa.

Os principais critérios de Fluxo de Dados são apresentados a seguir, considerando a relevância desses critérios na atividade de teste, o uso deles no teste de programas concorrentes (Seção 2.5), no teste de especificações formais (Seção 2.6) e pela perspectiva futura de estender os critérios de teste propostos nesta tese com informações sobre o fluxo de dados de especificações.

2.4.2. Critérios de Fluxo de Dados

Os critérios de fluxo de dados utilizam informações sobre fluxo de dados para derivar os requisitos de teste. Ao invés de selecionar os caminhos somente baseando-se na estrutura de controle, como ocorre com os critérios de fluxo de controle, os critérios de fluxo de dados estabelecem associações entre definições de variáveis no programa e subsequentes referências a essas definições para derivar os caminhos a serem percorridos pelos casos de teste. Esses critérios estabelecem uma

“ponte” entre os critérios de fluxo de controle *Todos-Arcos* e *Todos-Caminhos* e tornam o teste estrutural mais rigoroso.

Uma variável é definida quando: i) ela está no lado esquerdo de um comando de atribuição, ii) ela está em um comando de entrada, ou iii) ela está em uma chamada de procedimentos como parâmetro de saída. Uma variável é referenciada quando seu valor é utilizado, podendo ser um uso computacional (*c-uso*), quando a variável é usada do lado direito de uma expressão, ou ser um uso predicativo (*p-uso*), quando a variável é utilizada em um predicado ou condição, afetando o fluxo de controle do programa.

As associações entre definições-usos de variáveis são obtidas a partir de um grafo, chamado *Grafo Def-Uso* (Rapps e Weyuker, 1985), que consiste de uma extensão do grafo de programa (descrito anteriormente). Para cada nó i do grafo *Def-Uso* são estabelecidos os conjuntos $c\text{-use}(i)$ – conjunto de variáveis com c-uso no nó i , e $def(i)$ – conjunto de variáveis com definições no nó i . Para cada arco (i,j) do grafo é estabelecido o conjunto $p\text{-use}(i,j)$ – conjunto de variáveis com p-uso no arco (i,j) .

Segundo Maldonado (1991), Herman (1976) pode ser considerado um dos precursores do uso de informações de fluxo de dados para o estabelecimento de critérios de teste. Outros trabalhos que definem critérios de teste baseado em fluxo de dados são: Laski e Korel (1983), Ntafos (1984), Rapps e Weyuker (1985), Ural e Yang (1988) e Maldonado (1991).

Os critérios básicos de Rapps e Weyuker (1985) são:

- **Todas-Definições (*all-defs*)**: requer que cada definição de variável seja exercitada pelo menos uma vez, não importa se por um *c-uso* ou um *p-uso*.
- **Todos-Usos (*all-uses*)**: requer que todas as associações entre uma definição de variável e seus subseqüentes usos (*c-usos* e *p-usos*) sejam exercitadas

pelos casos de teste, através de pelo menos um caminho livre de definição, ou seja, um caminho onde a variável não é redefinida.

A maior parte dos critérios baseados em fluxo de dados, para requererem um determinado elemento (por exemplo, associação, caminho, etc.) exige a ocorrência explícita de um uso de variável e não garante, necessariamente, a inclusão do critérios *Todos-Arcos* na presença de caminhos não executáveis, presentes na maioria dos programas (Maldonado, 1991).

Com a introdução do conceito **potencial-uso**, Maldonado (1991) define vários critérios de fluxo de dados, denominados critérios **Potenciais Usos**, cujos elementos requeridos são caracterizados independentemente da ocorrência explícita de uma referência (um uso) a uma determinada definição; se um uso dessa definição pode existir, ou seja, existir um caminho livre de definição até um certo nó ou arco – um potencial uso – a **potencial associação** entre a definição e o potencial uso é caracterizada, e eventualmente requerida. Na realidade, pode-se dizer que, com a introdução do conceito potencial uso, procura-se explorar todos os possíveis efeitos a partir de uma mudança de estado do programa em teste, decorrente de definição de variáveis em um determinado nó *i*. Da mesma forma como os demais critérios baseados na análise de fluxo de dados, os critérios Potenciais-Usos podem utilizar o Grafo *Def-Uso* como base para o estabelecimento dos requisitos de teste. Na verdade, basta ter a extensão do grafo de programa associando a cada nó do grafo informações a respeito das definições que ocorrem nesses nós, denominado de Grafo *Def* (Maldonado, 1991). Por definição, toda associação é uma potencial associação e dessa forma, as associações requeridas pelo critério *Todos-Usos* são um subconjunto das potenciais associações requeridas pelo critério *Todos-Potenciais-Usos* (Maldonado et al., 1998).

- **Todos-Potenciais-Usos:** requer, basicamente, para todo nó i e para toda variável x , para a qual existe uma definição em i , que pelo menos um caminho livre de definição com relação à variável (c.r.a) x do nó i para todo nó e para todo arco possível de ser alcançado a partir de i por um caminho livre de definição c.r.a. x seja exercitado.

Os critérios estruturais têm sido utilizados principalmente no teste de unidade, uma vez que os requisitos de teste por eles exigidos limitam-se ao escopo de unidade. Várias pesquisas que procuram estender o uso de critérios estruturais para o teste de integração podem ser identificadas. Linnenkugel e Müllerburg (1990) propõem uma série de critérios que estendem os critérios baseados em fluxo de controle e em fluxo de dados para o teste de integração. Harrold e Soffa (1991) apresentam uma técnica para determinar as estruturas de definição-uso interprocedimentais permitindo a aplicação dos critérios de fluxo de dados no nível de integração. Vilela (1998), com base no conceito potencial uso, estende os critérios *Potenciais-Usos* para o teste de integração.

Existem várias iniciativas de desenvolvimento de ferramentas de teste para apoiar a aplicação de critérios de Fluxo de Dados (Frankl e Weyuker, 1985; Maldonado et al., 1989; Horgan e Mathur, 1992; Herbert e Price, 1995).

A Poke-Tool – *Potential Uses Criteria Tool for Program Testing*, desenvolvida na Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, é uma ferramenta de teste que apóia a aplicação dos critérios *Potenciais-Usos*, *Todos-Nós* e *Todos-Arcos* (Maldonado et al., 1989). Devido a essa ferramenta ser multilinguagem, existem versões para o teste de unidade de programas na linguagem C (Chaim, 1991), Fortran (Fonseca, 1993) e Cobol (Leitão, 1992). A ferramenta Poke-Tool é orientada à sessão de trabalho, ou seja, o usuário entra com o programa a ser testado,

com o conjunto de dados de teste e seleciona todos ou alguns dos critérios suportados. Como saída, a ferramenta fornece ao usuário os seguintes resultados: conjunto de *arcos primitivos* (arcos que uma vez executados garantem a execução de todos os demais arcos do grafo de programa), grafo def-uso do programa, programa instrumentado para teste, conjunto de associações necessárias para satisfazer o critério selecionado e conjunto de associações ainda não exercitadas. Atualmente, essa ferramenta encontra-se disponível para os ambientes DOS e UNIX. A versão para DOS possui interface simples, baseada em menus. A versão para UNIX possui módulos funcionais cuja utilização se dá através de interface gráfica ou linha de comando (*shell scripts*).

A *Atac – Automatic Test Analysis for C* (Horgan e Mathur, 1992) é um dos esforços mais significativos no desenvolvimento de ferramentas de teste. Essa ferramenta foi desenvolvida na *Bell Communications Research* (hoje *Telcordia Technologies*) e apóia a aplicação de critérios de fluxo de controle e fluxo de dados no teste de programas nas linguagens C e C++. Basicamente, a *Atac* permite verificar a adequação de um conjunto de casos de teste, visualizar código não coberto pelos casos de teste, auxiliar na geração de casos de teste e reduzir o tamanho do conjunto de teste, através da eliminação de casos de teste redundantes. Atualmente a *Atac* está integrada ao *xSUDS* (*Telcordia Software Visualization and Analysis Toolsuite*), um ambiente de suporte às atividades de teste, análise e depuração. O ambiente *xSUDS* vem sendo comercializado pela IBM, sendo uma forte evidência de que o uso de critérios baseados em fluxo de dados constituirá, em um futuro próximo, o estado da prática no que diz respeito ao teste de software (Barbosa, 2000).

- **Técnica Baseada em Erros**

A técnica de teste baseada em erros utiliza informações sobre os tipos de erros mais comuns cometidos durante o processo de desenvolvimento de um software (DeMillo et al., 1978). A ênfase dessa técnica está nos erros que o programador ou o projetista pode cometer durante o processo de desenvolvimento, e nas abordagens que podem ser usadas para detectar a sua ocorrência.

Os critérios mais conhecidos dessa técnica são:

- **Critério Semeadura de Erros:** alguns erros são inseridos “artificialmente” no programa. Então, dos erros encontrados durante o teste verificam-se quais são naturais e quais são artificiais. A razão dos erros artificiais pelos naturais representa, teoricamente, o número de erros naturais ainda existentes no programa e serve para medir a adequação do conjunto de casos de teste empregado. Entretanto, os resultados são dependentes de como os defeitos são introduzidos, os quais, normalmente, são introduzidos manualmente. Além disso, nem sempre os defeitos artificiais são equivalentes aos defeitos naturais tornando difícil a identificação dos defeitos naturais. O critério Análise de Mutantes, descrito a seguir, procura amenizar esses problemas introduzindo defeitos no programa de uma maneira mais sistemática (Zhu et al., 1997).
- **Critério Análise de Mutantes:** a partir de pequenas modificações sintáticas, introduzidas sistematicamente no programa, são geradas versões ligeiramente diferentes do programa chamadas de *mutantes*. O objetivo é que sejam obtidos casos de teste que consigam revelar, através da execução

do programa, as diferenças de comportamento existentes entre o programa original e seus mutantes (DeMillo, 1987).

Esse critério de teste é descrito com mais detalhes a seguir, devido à sua utilização neste trabalho. Nesta tese, será utilizado o termo Teste de Mutação compreendendo sua proposta inicial, que é o critério Análise de Mutantes para o teste de unidade, e sua extensão, o critério Mutação de Interface para o teste de integração. Quando houver necessidade de distinção entre eles, os termos originais serão utilizados.

2.4.3. Teste de Mutação

O critério Análise de Mutantes surgiu na década de 70 na *YALE University* e *Georgia Institute of Technology*, possuindo um forte relacionamento com um método clássico para detecção de erros lógicos em circuitos digitais, conhecido como modelo de teste de falha única (DeMillo, 1980).

Um dos primeiros artigos que descrevem a idéia de teste de mutantes foi publicado em 1978 (DeMillo et al., 1978). Nesse artigo, DeMillo apresenta a filosofia básica da técnica, conhecida como *Hipótese do Programador Competente*: “Programadores possuem uma grande vantagem que é pouco explorada: eles costumam criar programas muito próximos do correto!”. Assumindo a validade dessa hipótese, o autor afirma que erros são introduzidos nos programas através de desvios sintáticos que, apesar de não causarem erros sintáticos, alteram a semântica do programa e, como consequência, conduzem o programa a um comportamento incorreto. Para revelar tais erros, o Teste de Mutação identifica os desvios sintáticos mais comuns e, aplicando pequenas transformações sobre o programa em teste,

encoraja o testador a construir casos de teste que mostrem que tais transformações conduzem a um programa incorreto (Agrawal et al., 1989).

Outra hipótese explorada na aplicação do Teste de Mutação é o *Efeito de Acoplamento*, a qual assume que erros complexos são “acoplados” a erros simples de modo que um conjunto de casos de teste capaz de detectar erros simples irá também detectar uma alta porcentagem de erros complexos (DeMillo et al., 1978). Nesse sentido, aplica-se uma mutação de cada vez no programa em teste, ou seja, cada mutante contém apenas uma transformação sintática. Estudos realizados por Offutt (1992) indicam a validade dessa hipótese reforçando a premissa de que o Teste de Mutação é bem fundamentada.

A aplicação do critério do Teste de Mutação envolve as seguintes atividades: execução do programa com um conjunto de casos de teste T , geração dos mutantes, execução dos mutantes com T e análise dos mutantes. Um programa P é testado com um conjunto de casos de teste T . Se o programa funciona corretamente, então P sofre pequenas perturbações, gerando mutantes que são executados com T . Caso o comportamento de P' (um mutante de P) seja diferente de P , então esse mutante é dito “morto”. Caso contrário, esse mutante está “vivo” devido a um dos dois motivos: 1) o conjunto T não é suficiente (adequado) para distinguir o comportamento de P e P' e, com isso novos casos de teste devem ser incluídos ao conjunto; ou 2) P' é dito equivalente a P , ou seja, para qualquer dado do domínio de entrada o comportamento dos dois programas não difere.

Os mutantes são gerados a partir de operadores de mutação que são específicos para uma linguagem de programação ou técnica de especificação e procuram sintetizar os erros mais comuns nessa linguagem ou técnica. Assim, operadores de

mutação são as regras que definem as alterações que devem ser aplicadas no programa em teste.

Conseguindo-se obter casos de teste que resultem em apenas mutantes mortos e equivalentes, tem-se um conjunto de casos de teste adequado ao programa em teste. Significa também que se esse programa possui erros, são erros pouco prováveis de ocorrerem, de acordo com o efeito de acoplamento, o que é fortemente dependente do tipo de mutante gerado para esse programa (DeMillo et al., 1978).

É importante observar que a equivalência entre programas é uma questão indecidível e requer a intervenção do testador. Alguns métodos e heurísticas têm sido propostos para determinar a equivalência de programas em uma grande porcentagem de casos de interesse (Budd, 1981; Offutt e Pan, 1996).

O critério Teste de Mutação fornece uma medida objetiva para avaliar a cobertura de um conjunto de casos de teste em relação ao programa em teste. Essa medida, chamada de *escore de mutação*, relaciona o número de mutantes mortos com o número de mutantes gerados e é calculada da seguinte maneira:

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

sendo que:

$DM(P, T)$: número de mutantes mortos pelo conjunto de casos de teste T;

$M(P)$: número de mutantes gerados para o programa P;

$EM(P)$: número de mutantes equivalentes ao programa P.

O escore de mutação varia no intervalo [0,1], sendo que quanto maior o escore mais adequado é o conjunto de casos de teste para o programa em teste. Percebe-se com essa fórmula que somente $DM(P, T)$ é dependente do conjunto de casos de teste

utilizado e que $EM(P)$ é obtido à medida que o testador decide, manualmente ou com o apoio de heurísticas, que determinado mutante vivo é equivalente.

Um dos maiores problemas para a aplicação do Teste de Mutação está relacionado ao seu alto custo, uma vez que o número de mutantes gerados, mesmo para pequenos programas, pode ser muito grande, exigindo um tempo de execução muito alto. Outro problema com a aplicação desse critério é a questão de equivalência: a determinação dos mutantes equivalentes é uma atividade custosa e que requer intervenção do testador.

Algumas soluções têm sido propostas objetivando viabilizar a aplicação do Teste de Mutação. A utilização de arquiteturas de hardware avançadas e o uso de análise estática de anomalias de fluxo de dados para reduzir o número de mutantes gerados são algumas dessas soluções (Krauser, et al., 1988; Mathur e Krauser, 1988; Marshall, et al., 1990; Choi e Mathur, 1993).

Critérios de teste alternativos para o teste de Mutação também têm sido definidos com o objetivo de reduzir os custos de aplicação desse critério. Esses critérios alternativos procuram selecionar apenas um subconjunto do total de mutantes gerados, reduzindo o custo, mas procurando não reduzir a eficácia em revelar erros. Offutt et al. (1993) definem a **mutação seletiva**, que seleciona alguns operadores de mutação para geração dos mutantes, por exemplo, descartando aqueles que geram um número elevado de mutantes em relação aos demais. Mathur e Wong (1993) apresentam dois critérios alternativos: **mutação aleatória** – que examina uma pequena porcentagem de mutantes selecionados aleatoriamente de cada operador de mutação e ignora os demais, e **mutação restrita** – que seleciona alguns tipos de operadores de mutação para geração dos mutantes. Outros trabalhos nessa linha, são os trabalhos de Offutt et al. (1996b), Barbosa et al. (2000b) e

Vincenzi et al. (1999) os quais definem um conjunto essencial de operadores de mutação para o teste de programas em Fortran e em C (teste de unidade e teste de integração), respectivamente.

Nessa mesma linha de trabalhos para reduzir os custos do Teste de Mutação, Offutt e Pan (1996) propõem uma técnica baseada em restrições para parcialmente detectar mutantes equivalentes. Os autores consideram que uma restrição é uma expressão algébrica que restringe o domínio de entrada do programa de modo que esse conjunto restrito satisfaça uma determinada meta. Por exemplo, $x > 0$ restringe o domínio de entrada para os valores em que x é positivo. Essa técnica é utilizada para geração automática de casos de teste, de modo que as restrições representam as condições que devem ser satisfeitas para distinguir os mutantes. A técnica proposta para detectar automaticamente os mutantes equivalentes utiliza o fato de que, se um caso de teste distingue um mutante, a restrição é verdadeira. Caso contrário, não existe nenhum caso de teste que pode distinguir o mutante e ele é considerado equivalente. Experimentos realizados pelos autores indicam que a técnica reduz o custo para a determinação de mutantes equivalentes, sendo possível detectar automaticamente 45% do total de mutantes equivalentes para os programas analisados.

Procurando explorar o critério Análise de Mutantes no contexto de teste de integração, Delamaro (1997) define o critério Mutação de Interface que explora erros de interface relacionados com a conexão entre as unidades de programas implementados na linguagem C. A idéia é caracterizar erros simples de integração que podem ocorrer na passagem de dados (por parâmetros ou por referência), no uso de variáveis globais, ou no comando *return* do módulo chamado. Considere um programa P , em que existam funções F e G , tal que F chama G . $SI(G)$ é o conjunto

de valores passados para G e $SO(G)$ os valores retornados por G . Dada essa situação, os erros de integração são classificados em três categorias: 1) erro tipo 1: quando os valores contidos em $SI(G)$ não são os esperados por G , influenciando a produção de saídas erradas antes do retorno de G ; 2) erro do tipo 2: quando os valores contidos em $SI(G)$ não são esperados em G e, desse modo, $SO(G)$ assume valores incorretos, fazendo com que F produza uma saída incorreta após o retorno da função G e, 3) erro do tipo 3: quando os valores contidos em $SI(G)$ são os esperados por G mas valores incorretos de $SO(G)$ são produzidos, fazendo com que F produza um resultado incorreto após o retorno de G . Segundo Delamaro (1997), essa classificação de erros é abrangente e não especifica o local do defeito que causa o erro; ela simplesmente considera a existência de um valor incorreto entrando ou saindo de uma função chamada. Observa-se então que os operadores de Mutação de Interface estão relacionados a uma conexão entre duas unidades, de modo que esse critério é aplicado ponto-a-ponto, de forma a testar cada conexão existente. Dois grupos de operadores de mutação são definidos: 1) operadores que são aplicados nos pontos relacionados com a comunicação entre unidades dentro da função chamada, e 2) operadores que são aplicados onde uma função é chamada. No primeiro caso, é necessário um mecanismo que permita identificar o ponto a partir do qual a função foi chamada, de modo que somente devem ser ativadas as mutações relacionadas com a conexão em teste.

Uma das vantagens da Mutação de Interface em relação ao critério Análise de Mutantes para o teste de unidades é que ela tende, por princípio, a reduzir o número de mutantes a serem executados e analisados pois sua aplicação se restringe aos pontos de conexão entre as unidades do programa (Delamaro e Maldonado, 1999). Estudos realizados por Vincenzi et al. (2000) indicam que os critérios de teste

Análise de Mutantes e Mutação de Interface são complementares e devem ser utilizados em conjunto para aumentar a qualidade da atividade de teste. Os autores apresentam uma estratégia de teste incremental para aplicação desses critérios de teste.

Com base na Mutação de Interface definida por Delamaro (1997), Ghosh e Mathur (2000) estendem esse critério para o teste de sistemas baseados em componentes. Um sistema baseado em componentes é composto de módulos que encapsulam dados e funcionalidades e podem ser configurados, por meio de parâmetros, em tempo de execução. Para aplicação da Mutação de Interface no contexto de aplicações distribuídas, é considerada a arquitetura CORBA – *Common Object Request Broker Architecture*⁵. Uma Linguagem de Descrição de Interface (em inglês, IDL) é utilizada para especificar os aspectos de interface dos componentes, descrevendo cada método que o componente oferece e especificando a declaração dos métodos (nome e parâmetros) e tratamento de exceções. O critério Mutação de Interface é aplicado na descrição das interfaces dos componentes, através de um conjunto de operadores de mutação definido para CORBA-IDL. Esses operadores de mutação são aplicados nos parâmetros dos métodos e nos valores de retorno. Por exemplo, o operador de mutação *troca (swap)* é aplicado na descrição do método e realiza a permutação entre os parâmetros de mesmo tipo do método. Para realizar o teste de mutação, o testador executa as entradas de teste a partir do cliente (usuário dos componentes) utilizando a interface original (sem mutação) e a interface mutante. Para a interface original, o cliente envia uma requisição para o servidor (que fornece os componentes) e uma resposta (*response-A*) é obtida. Para a interface mutante, o objetivo é obter uma resposta (*response-B*) diferente de

⁵ <http://cgi.omg.org/corba/beginners.html>

response-4. Uma ferramenta de teste chamada *TDS – Testing Distributed Systems*, foi desenvolvida para apoiar a aplicação da Mutação de Interface para o teste de componentes definidos usando CORBA-IDL (Ghosh et al. 2000).

Outra linha de trabalhos tem explorado o uso do Teste de Mutação para o teste de especificações, fazendo um mapeamento dos conceitos desse critério do nível de programas para o nível de especificação (Probert e Guo, 1991; Fabbri, 1996). Esses trabalhos serviram de base para a definição, nesta tese, do critério Teste Mutação para Estelle, e são discutidos com mais detalhes na Seção 2.6, que trata sobre teste de especificações.

Um aspecto importante para a aplicação do Teste de Mutação é a disponibilidade de ferramentas que automatizem esse critério. Nesse sentido, algumas ferramentas podem ser citadas. *Mothra* é um ambiente de teste baseado no critério Análise de Mutantes para programas na linguagem Fortran-77, com 22 operadores de mutação implementados (DeMillo et al., 1988). A interface utilizada é baseada em janelas o que facilita a realização dos testes; permite incorporar outras ferramentas como gerador de casos de teste, verificador de equivalência e oráculo.

A ferramenta *Proteum/C – Program Test Using Mutants*, desenvolvida pelo Grupo de Engenharia de Software do Instituto de Ciências Matemáticas e de Computação de São Carlos – USP, é outra ferramenta que apóia a aplicação do critério Análise de Mutantes (Delamaro, 1993). Essa ferramenta foi desenvolvida a partir dos operadores de mutação definidos por Agrawal et al. (1989) e dá suporte ao teste de unidade para programas na linguagem C. Devido ao seu aspecto de multilinguagem ela pode ser configurada para o teste de programas escritos em diferentes linguagens. Na *Proteum/C* a atividade de teste pode ser conduzida através de um ambiente gráfico ou na forma de *scripts*. Esta última forma é muito útil para a

realização de experimentos empíricos, pois reduz as interações do testador com a ferramenta diminuindo, com isso, o tempo consumido na atividade de teste.

A partir da ferramenta *Proteum/C* outras ferramentas foram desenvolvidas, compondo uma família de ferramentas para apoiar o teste de especificações e de programas baseado em Mutação (Maldonado et al., 2000a). A ferramenta *Proteum/IM* (Delamaro, 1999) apóia o teste de integração de programas em C. Ela possui 33 operadores de mutação, os quais são divididos em 2 grupos: 24 operadores do Grupo I, aplicados na função chamada, e 9 operadores do Grupo II, aplicados no ponto de chamada. A ferramenta *Proteum/IM 2.0* é uma evolução das ferramentas *Proteum* e *Proteum/IM*, fornecendo um ambiente integrado que facilita o desenvolvimento de estudos para definição de estratégias de teste, englobando o teste de unidade e de integração (Delamaro et al., 2000).

A ferramenta *Proteum-RS/FSM* apóia a aplicação do Teste de Mutação para validar especificações baseadas em Máquinas de Estados Finitos (Fabbri et al., 1999b). Essa ferramenta possui 9 operadores de mutação implementados, os quais foram definidos com base na classificação de erros para Máquinas de Estados feita por Chow (1978).

A ferramenta *Proteum-RS/ST* apóia a aplicação do Teste de Mutação para validar especificações baseadas em Statecharts (Fabbri et al., 1999a; Sugeta, 1999). Os operadores de mutação são divididos em três categorias: 9 operadores de mutação para os aspectos de Máquinas de Estados Finitos, 11 para os aspectos relacionados a Máquinas de Estados Finitos Estendidas e 17 operadores para os aspectos específicos da técnica Statecharts.

A ferramenta *Proteum-RS/PN* apóia a aplicação do Teste de Mutação para validar especificações baseadas em Redes de Petri (Fabbri et al., 1995; Simão et al.,

2000). Dois tipos de erros foram considerados para definição dos operadores de mutação para Redes de Petri: 8 operadores de mutação para alteração nos arcos da rede e 3 operadores de mutação para marcação inicial da rede. A ferramenta possui facilidades para geração de casos de teste e determinação de mutantes equivalentes (Simão e Maldonado, 2000).

As características gerais dessas ferramentas são apresentadas em Maldonado et al. (2000a) e retomadas no Capítulo 3 desta tese.

Dada a diversidade de critérios de teste existentes, decidir qual deles deva ser utilizado ou como utilizá-los de maneira complementar para obter melhores resultados não é uma tarefa fácil. Nesse contexto, o desenvolvimento de estudos empíricos e teóricos tem como objetivo comparar e analisar os critérios de teste, considerando, principalmente, o custo de aplicação e a eficácia em revelar erros, de modo a fornecer subsídios para que os critérios de teste possam ser aplicados na prática. Na próxima seção são apresentados os principais resultados obtidos com a realização de estudos empíricos que focalizam o critério Teste de Mutação, dada a sua eficácia em revelar erros (Mathur e Wong, 1993; 1994; Wong et al., 1994a; 1994b; Offutt et al., 1996a; 1996b; Souza, 1996, Delamaro, 1997; Barbosa et al., 2000b; Vincenzi et al., 1999).

2.4.4. Estudos Empíricos Relacionados ao Teste de Mutação⁶

Critérios de teste vêm sendo estudados teoricamente e empiricamente. Do ponto de vista de estudos teóricos, os critérios de teste são avaliados segundo dois aspectos: a *relação de inclusão*, e a *complexidade dos critérios*. A relação de inclusão estabelece uma ordem parcial entre os critérios caracterizando uma hierarquia entre

eles. Um critério C_1 inclui um critério C_2 se para qualquer programa P e qualquer conjunto de casos de teste T_1 , C_1 -adequado, T_1 for também C_2 -adequado e para um conjunto de casos de teste T_2 , C_2 -adequado, T_2 não é C_1 -adequado. A complexidade de um critério C_1 é definida como o número máximo de casos de testes requeridos no pior caso ou o número de elementos (ou requisitos) requeridos no pior caso (Rapps e Weyuker, 1985; Weyuker, 1984, Ntafos, 1988).

Do ponto de vista de estudos empíricos, os seguintes fatores são utilizados para avaliar os critérios de teste: *custo*, *eficácia* e *dificuldade de satisfação (strength)*. Entende-se por *custo* o esforço necessário para que o critério seja usado, o qual pode ser medido pelo número de casos de teste necessários para satisfazer o critério, ou por outras medidas dependentes do critério, tais como: o tempo necessário para executar todos os mutantes gerados, ou o tempo gasto para identificar mutantes equivalentes, caminhos e associações não executáveis, construir manualmente os casos de teste e aprender a utilizar as ferramentas de teste. *Eficácia* refere-se à capacidade que um critério possui em detectar um maior número de erros em relação a outro. *Strength* refere-se à probabilidade de satisfazer-se um critério tendo satisfeito outro critério. Seu objetivo é verificar o quanto consegue-se satisfazer um critério C_1 tendo satisfeito um critério C_2 (C_1 inclui C_2 , ou C_1 e C_2 são incomparáveis, ou seja, não possuem uma ordem de inclusão) (Wong, 1993).

O Teste de Mutação tem sido bastante investigado empiricamente, em função de sua eficácia em revelar erros (Mathur e Wong, 1993; 1994; Wong et al., 1994a; 1994b; Offutt et al., 1996a; 1996b; Souza, 1996, Delamaro, 1997; Barbosa et al., 2000b; Vincenzi et al., 1999). Esses estudos, além de indicarem como esse critério

⁶ Esta seção foi parcialmente extraída de Barbosa (2000).

se relaciona com outros critérios de teste, buscam novas estratégias a fim de reduzir os custos associados a esse critério de teste.

Os trabalhos de Mathur e Wong (1994), Wong et al. (1994a) e Offutt et al. (1996a) fazem um estudo comparativo entre o Teste de Mutação e o critério de Fluxo de Dados Todos-Usos. Os objetivos dos experimentos foram verificar o *strength* dos critérios, custos e eficácia em revelar erros, uma vez que esses critérios são incomparáveis do ponto de vista teórico. Os resultados obtidos indicam que o Teste de Mutação possui um custo maior (em termos do número de casos de teste necessários) mas apresenta uma eficácia maior para revelar erros. Além disso, foi obtido que, na prática, o Teste de Mutação inclui o critério *Todos-Usos* (Mathur e Wong, 1993). O trabalho de Wong et al. (1994a) indica que utilizar mutações alternativas, como Mutação Restrita e Aleatória, é uma abordagem útil na avaliação e construção de conjuntos de casos de teste. Desse modo, quando existir pouco tempo para efetuar os testes (devido ao prazo de entrega do produto) é possível aplicar o Teste de Mutação para testar partes críticas do software e utilizar alternativas mais econômicas, tais como Mutação Restrita e o critério Todos-Usos, para o teste das demais partes do software, sem comprometer significativamente a qualidade da atividade de teste.

Na mesma linha desses trabalhos, Souza (1996) faz um estudo comparativo entre o Teste de Mutação e os critérios Potenciais-Usos (Maldonado, 1991), visto que os critérios Potenciais-Usos incluem o critério Todos-Usos. Os resultados indicaram que esses critérios são incomparáveis, mesmo do ponto de vista empírico, o que motiva a investigar o aspecto complementar desses critérios quanto à eficácia em revelar erros.

Mathur e Wong (1993), Wong et al. (1994b), Souza (1996) e Wong et al. (1997) avaliam o custo de aplicação e eficácia em revelar erros da Mutação Aleatória e da Mutação Restrita. Os resultados obtidos indicam que a mutação restrita e mutação aleatória são igualmente eficazes. Wong et al. (1994b) e Souza (1996) estabelecem uma ordem incremental para o emprego de classes de mutação restrita (cada classe é composta por um subconjunto de operadores de mutação), com base no custo e eficácia de cada uma. Desse modo, os conjuntos de casos de testes podem ser construídos, inicialmente, de forma a serem adequados à classe com menor relação custo/eficácia. Na seqüência, quando as restrições de custo permitirem, esse conjunto pode ser melhorado de modo a satisfazer as classes de mutação com maior relação custo/eficácia.

Offutt et al. (1996b) e Barbosa et al. (2000b) realizaram estudos para determinar um conjunto essencial de operadores de mutação para o teste de programas em Fortran e em C, respectivamente. Um conjunto essencial de operadores de mutação é formado por um subconjunto de operadores de mutação capaz de modelar a maioria dos erros que se objetivam revelar durante a atividade de teste. A aplicação do conjunto essencial obtido por Offutt et al. proporcionou uma redução de custo da ordem de 77,5%, mantendo um escore de mutação de 0,995 em relação à Análise de Mutantes. Barbosa et al. define um procedimento para a determinação de um conjunto essencial de operadores de mutação e os resultados obtidos indicam que os conjuntos essenciais gerados por esse procedimento apresentam um alto grau de adequação em relação ao critério Análise de Mutantes, com escores de mutação acima de 0,995, proporcionando, em média, reduções de custo superiores a 65%. Esses estudos contribuem fortemente para a viabilização da aplicação do Teste de Mutação em ambientes comerciais de desenvolvimento de software.

A mesma linha de estudos empíricos investigando o critério Análise de Mutantes foi aplicada no âmbito do teste de integração, visando a avaliar empiricamente o critério Mutação de Interface. Como apresentado por Maldonado et al. (1998), com a proposição do critério Mutação de Interface torna-se evidente o aspecto positivo de se utilizar o mesmo conceito de mutação nas diversas fases do teste. Ainda, é natural a indagação sobre qual estratégia utilizar para se obter a melhor relação custo/eficácia quando são aplicados os critérios Análise de Mutantes e Mutação de Interface no teste de um produto. Assim sendo, Delamaro (1997) e Vincenzi et al. (2000) conduziram estudos envolvendo o critério Mutação de Interface e suas abordagens alternativas. Os resultados obtidos demonstraram que, assim como a Análise de Mutantes, o critério Mutação de Interface apresenta alta eficácia em revelar a presença de erros, porém com um alto custo de aplicação em termos do número de mutantes gerados. Vincenzi et al. investigaram o relacionamento entre os critérios Análise de Mutantes e Mutação de Interface e como utilizar tais critérios de forma complementar na atividade de teste, tendo como objetivo contribuir para o estabelecimento de uma estratégia de teste incremental, de baixo custo de aplicação e que garanta um alto grau de adequação em relação a ambos os critérios. Os resultados obtidos indicam que, mesmo com um número reduzido de operadores, é possível determinar conjuntos de casos de teste adequados ou muito próximos da adequação para ambos os critérios, a um menor custo (Vincenzi et al., 2000). Nessa mesma linha, Maldonado et al. (2000b) conduziram um estudo investigando a aplicação da Mutação Seletiva no teste de programas C, tanto em nível de unidade quanto de integração. De modo geral, para os conjuntos de programas utilizados, observou-se que com uma redução próxima a 80% no número de mutantes gerados, ainda assim seriam obtidos escores de mutação bastante significativos, em torno de

0.980, indicando que a Mutação Seletiva é uma abordagem promissora para aplicação do Teste de Mutação.

De uma maneira geral, esses estudos indicam que é possível viabilizar a aplicação do critério Teste de Mutação na prática. Para que isso seja possível, é necessário o aprimoramento das ferramentas de apoio e o estabelecimento das possíveis estratégias de teste que podem ser aplicadas, considerando as abordagens de mutação alternativas existentes. Além disso, outro aspecto fundamental é a formação do pessoal envolvido, estimulando o processo de transferência tecnológica entre universidade e indústria e contribuindo para a melhoria da qualidade dos produtos de software desenvolvidos. Esse último aspecto vem sendo investigado por Barbosa (2000).

Na próxima seção, são descritos alguns trabalhos sobre o teste de programas concorrentes. Esses trabalhos auxiliaram na definição dos critérios de teste apresentados nesta tese, em função das técnicas Statecharts e Estelle serem empregadas principalmente para especificar sistemas que apresentam concorrência e paralelismo.

2.5. Teste de Programas Concorrentes

Vários critérios de teste têm sido propostos para a atividade de teste de programas (Herman, 1976; DeMillo et al., 1978; Myers, 1979; Laski e Korel, 1983; Ntafos, 1984; Rapps e Weyuker, 1985; Coward, 1988; Ural e Yang, 1988; Beizer, 1990; Maldonado, 1991; Pressman, 2000). Essas técnicas são definidas para programas seqüenciais, os quais possuem um comportamento determinístico, ou seja, a partir de um valor de entrada D , toda vez que um programa seqüencial P é

executado com D a mesma saída é obtida. Isso ocorre porque os comandos do programa são executados seqüencialmente e os desvios são selecionados deterministicamente, de acordo com os valores de entrada. Entretanto, em programas concorrentes, dado o mesmo valor de entrada, diferentes saídas corretas podem ser produzidas. Características como execuções paralelas, comunicações e sincronizações são responsáveis por esse comportamento não determinístico. Essas características diferem os programas concorrentes dos programas seqüenciais e precisam ser consideradas durante a atividade de teste de software.

Alguns trabalhos são encontrados que estendem as técnicas e critérios de teste propostos para programas seqüenciais de forma que eles possam ser aplicados a programas concorrentes (Yang e Chung, 1992; Taylor et al., 1992; Chung et al, 1996; Koppol e Tai, 1996; Yang et al., 1998; Silva-Barradas, 1998).

Yang e Chung (1992) definem o critério de teste análise de caminhos para validar programas concorrentes. A partir de um programa concorrente, são gerados dois grafos: 1) *Grafo de Sincronização*, que apresenta o comportamento em tempo de execução do programa concorrente, modelando as possíveis sincronizações; e 2) *Grafo de Processo*, que apresenta a visão estática do programa, modelando o fluxo de controle entre os comandos do programa. Uma execução do programa irá sensibilizar uma rota *de sincronização (c-rota)* no grafo de sincronização e um caminho (*c-caminho*) no grafo de processo, sendo que um c-caminho pode possuir mais de uma c-rota. Os autores apontam três resultados práticos de sua pesquisa: seleção de c-caminhos, geração de casos de teste definitivos e execução dos casos de teste. Os autores apresentam também uma metodologia para auxiliar na fase de depuração, chamada de *execução controlada*, a qual fundamenta-se no seguinte teorema: “Se duas execuções de um programa concorrente com a mesma entrada

atravessam a mesma c-rota, então essas duas execuções devem produzir o mesmo comportamento". São consideradas duas execuções do programa: na primeira execução são obtidas as c-rotas que podem ser atravessadas para cada caso de teste; na segunda execução o dado de teste é formado pelo caso de teste e pela c-rota que deve ser atravessada, garantindo que a mesma c-rota seja atravessada.

Taylor et al. (1992) definem um conjunto de critérios estruturais para o teste de programas concorrentes, com base em um *Grafo de Concorrência*. Esse grafo mostra o comportamento do programa em execução, apresentando as possíveis sincronizações do programa. Os autores definem cinco critérios de teste: 1) *Todos-caminhos-concorrência*, 2) *Todas-histórias-concorrência*, 3) *Todos-arcos-entre-estados-concorrência*, 4) *Todos-estados-concorrência*, e 5) *Todas-possíveis-sincronizações*. A relação de inclusão (hierarquia) entre esses critérios é analisada.

Da mesma forma que Taylor et al. (1992), Chung et al. (1996) definem quatro critérios de teste para programas concorrentes, descritos na linguagem Ada: 1) *Todas-Entry-Call*, 2) *Todas-Possíveis-Entry-Acceptance*, 3) *Todas-Permutações-Entry-Call* e 4) *Todas-Dependências-de-Permutações-Entry-Call*. Esses critérios focalizam os aspectos de comunicação e sincronização entre as tarefas, expressos por intermédio de *rendezvous*. A relação de inclusão dos critérios também é apresentada pelos autores. Os critérios de teste são específicos para *rendezvous* e os autores apontam a necessidade de expandi-los para outros tipos de comunicação e sincronização.

Koppol e Tai (1996) apresentam uma abordagem incremental para o teste estrutural de programas concorrentes baseada na hierarquia de processos. Segundo os autores, sua abordagem ameniza o problema de explosão de estados. O modelo

básico utilizado é um *Sistema de Transições Rotuladas (LTS)*, que na verdade é um grafo de alcançabilidade para cada tarefa do programa concorrente.

Em uma outra linha de trabalho, auxiliando a demonstrar que, com algumas extensões, critérios de teste para programas seqüenciais são aplicáveis para o teste de programas paralelos, Yang et al. (1998) estendem o critério de fluxo de dados *todos-du-caminhos* para programas paralelos. Um *Grafo de Fluxo de Programa Paralelo (PPFG)* é construído a partir do programa, o qual é percorrido para obtenção dos *du-caminhos*. Todos os *du-caminhos* que possuem definição e uso de variáveis relacionadas ao paralelismo dos *threads* são requisitos de teste a serem executados, sendo que os *threads* são atividades de um programa paralelo que se comunicam entre si mas que são executadas independentemente (concorrentemente). A estratégia é automatizada através da ferramenta *della pasta (Delaware Parallel Sotware Testing Aid)*.

Silva-Barradas (1998) propõe a aplicação do critério Análise de Mutantes para programas concorrentes, considerando a linguagem Ada. É definido um conjunto de operadores de mutação para Ada e um mecanismo para realizar execução controlada, chamado *Análise de Mutantes Comportamental*. Nesse mecanismo, para cada execução do programa são consideradas a saída obtida e a seqüência de comandos executados, de forma que o comportamento do programa possa ser repetido a partir do valor de entrada e da seqüência percorrida. Esse aspecto é similar a execução controlada proposta por Yang e Chung (1992). Outro aspecto importante desse mecanismo é a possibilidade de analisar os mutantes vivos para verificar se existe um caso de teste no conjunto de casos de teste que poderia ter identificado o mutante, mas não o fez devido ao não determinismo. Isso é realizado através de execução simbólica e checagem de modelos.

Na próxima seção são apresentados alguns trabalhos sobre o teste de especificações baseadas em MEFEs, em particular, baseadas em Statecharts e em Estelle.

2.6. Teste de Especificações Formais

Por algum tempo, o teste de software foi considerado uma atividade a ser realizada somente na implementação do software e com isso, a maioria das pesquisas nessa área concentrou-se em estabelecer técnicas, critérios e ferramentas para validação de programas. Isso é observado pelo número de critérios de teste existentes no nível de programas. Entretanto, pesquisadores têm direcionado a atenção também para a validação de produtos, nas demais fases de desenvolvimento de software. Se um erro for inserido nas fases iniciais do desenvolvimento (por exemplo, durante a especificação do software), não é preciso, e não se deve, esperar até a fase de implementação do software para identificar e corrigir esse erro. Adiar a atividade de teste somente contribui para aumentar a dificuldade em se detectar o erro, o qual pode se propagar por diversas partes do programa, aumentando também o custo para a sua identificação e remoção.

Esta seção aborda o teste de especificações formais baseadas em MEFEs, pois mesmo utilizando técnicas formais para o desenvolvimento do software, não se garante que a especificação formal esteja livre de erros e de acordo com os requisitos do usuário.

Conforme mencionado anteriormente, duas atividades de teste são realizadas no escopo de especificações: 1) *teste de especificações*, que visa a encontrar erros na especificação e a garantir que ela esteja de acordo com os requisitos do usuário e 2)

teste de conformidade, também conhecido como teste baseado na especificação, é utilizado principalmente no teste de protocolos de comunicação e visa a garantir que a implementação do protocolo esteja em conformidade com a sua especificação.

Segundo Bourhfir et al. (1996), a especificação é composta por aspectos de dados e por aspectos de controle. Os trabalhos iniciais sobre teste de especificações concentraram-se em definir métodos para geração de seqüências de teste baseados em fluxo de controle, aplicados principalmente para MEFs. Esses métodos concentram-se em erros de transferência e erros de transições na MEF e são bastante empregados no teste de conformidade de protocolos de comunicações (Bochmann e Petrenko, 1994). Os métodos são: *Transition Tour (TT)*, *Distinguishing Sequence (DS)*, *Characterizing Set (W)*, *Unique Input/Output Sequence (UIO)* e *Partial W-Method (Wp)*. Uma síntese desses métodos pode ser encontrada em Ural (1992) e em Bourhfir et al. (1996). Para a aplicação da maioria desses métodos, a MEF precisa satisfazer um conjunto de propriedades e características: a) *estados equivalentes* – dois estados s_i (MEF M_1) e s_j (MEF M_2) são equivalentes se, quando exercitados por qualquer seqüência de entrada, produzem saídas idênticas; caso contrário, são denominados estados distinguíveis. M_1 e M_2 podem ser a mesma máquina; b) *especificação completa* – para cada estado existe uma transição para cada símbolo de entrada; c) *minimalidade* – nenhum par de estados é equivalente; d) *determinismo* – para toda entrada existe uma transição definida em cada estado; e e) *conectividade forte* – para todo par de estados (s_i, s_j) existe uma seqüência de símbolos de entrada tal que essa seqüência leva a máquina do estado s_i para o estado s_j . A aplicabilidade dos métodos de geração de seqüências de teste fica restrita a MEFs que satisfaçam as condições impostas pelos métodos o que, na prática, nem sempre ocorre.

Em se tratando de MEFE, os métodos anteriores para seleção de seqüências de teste não são adequados pois os aspectos de dados também precisam ser considerados.

Sarikaya et al. (1987) apresentam uma metodologia para gerar seqüências de teste, considerando os aspectos de fluxo de dados e de fluxo de controle de especificação baseadas em Estelle. A especificação em Estelle é transformada em uma forma equivalente chamada *especificação na forma normal (EFN)*. Basicamente, essa especificação não possui certas construções da linguagem Estelle, tais como: lista de estados (*stateset*), comandos condições (*if* e *case*) e chamadas de procedimentos e funções. Por exemplo, para remover um comando *if* de uma transição, são geradas duas transições, uma para cada possível valor do comando *if* (verdadeiro e falso). Na verdade, a condição do comando *if* é inserida na cláusula *provided* das transições geradas. A partir da EFN, são gerados grafos de fluxo de controle e de fluxo de dados e as seqüências de teste são geradas percorrendo-se esses grafos. Os autores ilustram a utilização do método *Transition Tour* (requer que todas as transições sejam executadas pelo menos uma vez) para gerar as seqüências de teste. Esse método não engloba a comunicação entre módulos, sendo aplicado a cada módulo da especificação.

Ural (1987) apresenta um método para selecionar seqüências de teste para protocolos de comunicação especificados em Estelle, com o objetivo de testar os aspectos de dados e de controle de implementações do protocolo. Assim como o trabalho Sarikaya et al. (1987), a especificação Estelle é transformada para a forma normal (EFS). Um grafo modela os fluxos de dados e de controle da especificação. Da mesma forma que é feito no teste de programas (Rapps e Weyuker, 1985), a partir desse grafo são identificadas as definições e usos de cada variável como também os

parâmetros das interações de entrada e de saída dos módulos. Baseado nessas informações, um conjunto de seqüências de teste é definido, o qual executa todos os pares de definições e usos, ou seja, que satisfaz o critério *Todos-Usos*. Dos critérios de Fluxo de Dados propostos por Rapps e Weyuker (1985) somente o critério *Todos-Usos* é empregado. Devido à EFN, esse método não considera a comunicação entre módulos.

Ural e Yang (1991) estendem o trabalho de Ural (1987) definindo um critério de Fluxo de Dados chamado *input-output dataflow chain (IO-df-chain)*. Esse critério requer que sejam executadas todas as associações entre cada saída (comando *output* da transição) e as entradas que influenciam a saída. Esse critério torna-se mais abrangente que o critério *Todos-Usos* pois requer, ao invés de associações isoladas entre definições e usos de variáveis, uma “cadeia” de definições e usos de variáveis entre uma saída e os valores que influenciam essa saída. Da mesma forma que em Ural (1987), a especificação Estelle é transformada em uma especificação na forma normal sendo gerado um grafo de fluxo de dados. Os autores apontam que o método proposto pode ser aplicado também para especificações compostas de mais de um módulo, sendo que neste caso é necessário estabelecer as associações entre as entradas e saídas de cada módulo e concatenar as associações resultantes de acordo com a comunicação existente entre os módulos. Esse método não considera as condições das transições e o fluxo de controle da especificação.

Henniger et al. (1995) apresentam um método para transformar MEFEs dos módulos na forma normal de especificações Estelle em *MEFEs desdobradas*. Nessas *MEFEs desdobradas* são retiradas as condições e prioridades das transições e a transição passa a ser habilitada somente pelo valor da entrada (mensagem recebida) e pelo estado origem da MEFE. Essa transformação aumenta o número de estados da

MEFE e permite que sejam aplicados os métodos para geração de seqüências de teste empregados para MEFs (*transition tour, método UIO, W, DS, Wp*). A transformação é possível quando as variáveis que ocorrem nas condições possuem valores finitos, o que, segundo os autores, é satisfeito na maioria das vezes pelas especificações de protocolos. Outra contribuição do trabalho de Henniger et al. (1995) é a classificação das variáveis de Estelle. As variáveis são classificadas em: *variáveis de contexto* – declaradas no corpo do módulo e usada nas ações das transições; *variáveis de interação* – são os parâmetros das primitivas de comunicação e *variáveis de controle* – variáveis que influenciam a seleção das transições, ou seja, são utilizadas na cláusula *provided*, ou na condição das transições. A abordagem apresentada pelos autores não considera especificações contendo mais de um módulo. Além disso, as *MEFEs desdobradas* podem apresentar um número elevado de estados (explosão de estados).

Bourhfir et al. (1997) apresentam um método para geração de seqüências de teste para especificações baseadas em MEFEs. Para a aplicação do método, a MEFE precisa satisfazer as propriedades de determinismo e reiniciabilidade (o estado inicial deve ser alcançado a partir de qualquer estado). O método pode também ser aplicado para Estelle, entretanto, a especificação deve estar na forma normal (EFN), conforme Sarikaya et al. (1987). O método consiste em percorrer a MEFE e estabelecer os caminhos (seqüências de transições) que devem ser percorridos. Esse método se diferencia das propostas apresentadas anteriormente porque descarta, durante a geração dos caminhos, os caminhos não executáveis. Para isso, um algoritmo analisa cada transição do caminho para verificar sua executabilidade. Essa análise é feita simbolicamente, sendo que cada predicado em cada transição é interpretado simbolicamente até conter somente constantes e parâmetros de entrada. Por exemplo,

para uma variável x definida em termos de outras variáveis, são feitos rastreamentos no caminho já percorrido, buscando recursivamente o valor de x definido em termos de parâmetros de entrada ou constantes. Quando esse valor é obtido, x é substituída por essa definição; caso contrário, a transição é considerada não executável. Os autores comentam que esse algoritmo é capaz de identificar a maior parte dos caminhos executáveis, facilitando a análise e a geração de seqüências de teste. Entretanto, essa abordagem não trata especificações em Estelle contendo vários módulos, pois é aplicada para cada MEFE.

Kim et al. (1998) propõem uma metodologia para geração de casos de teste para protocolos especificados através de MEFEs. A metodologia é composta de critérios de fluxo de dados, baseados nos critérios de Rapps e Weyuker (1985), e critérios de fluxo de controle, como *Todos-Caminhos*, *Todos-Nós*, *Todos-Arcos*. Os autores apresentam outros critérios, como *qualquer-uso*, *identificação-transições*, *identificação-estados*. Os autores mostram a relação entre seus critérios e os critérios de Rapps e Weyuker (1985), sendo que a relação de inclusão é adaptada para adequar-se ao contexto de MEFE. Nesse contexto, a definição e o uso de variáveis ocorre somente nos arcos (transições), e com isso, por exemplo, o critério *Todos-Uso*s não inclui o critério *Todos-Arcos*, ao contrário do que ocorre para esses critérios no nível de programas (Rapps e Weyuker, 1985). Simulação e análise de modelos são utilizadas concomitantemente para derivar os conjuntos de casos de teste com base nos critérios definidos. A metodologia não considera a comunicação entre módulos da especificação, gerando seqüências de teste para cada módulo isoladamente. Os critérios de teste propostos e a metodologia para geração dos casos de teste não são claramente definidos, deixando muitas questões em aberto.

Fecko et al. (2000) apresentam os resultados da aplicação de Estelle para a especificação do protocolo MIL-STD 188-220 (padrão militar das Forças Armadas Americanas) e a geração de seqüências de teste a partir da especificação formal. Para a geração de seqüências de teste, as MEFEs de Estelle são desdobradas e convertidas em MEFs, conforme Henniger et al. (1995), de forma que seja possível aplicar métodos de geração de seqüências de teste para MEFs. Para controlar o tamanho das MEFs geradas, as MEFEs são parcialmente desdobradas, determinando-se quais características de cada MFE devem ser desdobradas. Os autores apontam que determinar como as MEFEs devem ser desdobradas de modo a permitir que seqüências de teste efetivas possam ser geradas é um dos aspectos mais complexos do método. Os autores apresentam soluções para alguns problemas relacionados com a geração de seqüências de teste para protocolos de comunicação, tais como: diferença de tempo de execução das transições na especificação e na implementação, controle sobre as interfaces (entradas e saídas) com outras camadas do protocolo em teste e conflitos de tempos associados ao disparo de transições.

De uma maneira geral, observa-se que essas técnicas de geração de seqüências de teste para especificações em Estelle fazem algum tipo de restrição quanto às características de Estelle, principalmente, relacionadas ao paralelismo dos componentes e nas cláusulas das transições. Além disso, para a maioria das técnicas, a especificação em Estelle é transformada para uma especificação na forma normal, aumentando o número de transições das MEFEs dos módulos. Os critérios de cobertura para especificações Estelle apresentados nesta tese (Capítulo 4) se diferenciam dessas técnicas nos seguintes pontos: a) não fazem restrições quanto às características de Estelle; b) podem ser aplicados diretamente na especificação Estelle, não precisando que esta esteja na forma normal; c) podem ser aplicados para

mais de um módulo, ou seja, é possível selecionar um conjunto de componentes da especificação para aplicação dos critérios, permitindo também validar o paralelismo dos componentes selecionados; e d) os critérios de cobertura fornecem uma medida quantitativa do conjunto de seqüências de teste obtido em relação à cobertura da especificação.

Algumas iniciativas que empregam o Teste de Mutação para validar especificações formais são encontradas (Probert e Guo, 1991; Fabbri et al., 1994; 1995; 1999a). Esses trabalhos fazem um mapeamento da hipótese do programador competente definida no nível de programas, definindo a *hipótese do especificador ou projetista competente*: se a especificação em teste foi construída por um projetista ou especificador competente ou está correta ou está muito próxima do correto.

Probert e Guo (1991) definem a abordagem de teste *E-MPT – Estelle-directed Mutation-based Protocol Testing*, com base no Teste de Mutação, para validar especificações baseadas em Estelle. São definidos dois tipos de mutação: *mutação maior*, que testa as estruturas básicas de Estelle e *mutação menor* que testa as operações das transições. Estratégias são aplicadas para amenizar o custo de aplicação, como a *Mutação Ordenada* (Duncan, Robson, 1990), cuja idéia básica é ordenar os elementos (operadores de mutação, por exemplo) de forma a aumentar a velocidade da utilização do critério. São definidos também conjuntos de alternativas de mutação para cada elemento da especificação, por exemplo, variáveis, constantes e operadores matemáticos. Esses conjuntos são empregados para a geração das especificações mutantes. Os mutantes são gerados a partir da especificação e, através de um compilador para Estelle, tanto a especificação original como as especificações mutantes são traduzidas em programas na linguagem C. O compilador não gera códigos completos e dessa forma os programas gerados são completados

manualmente. A análise dos mutantes é feita executando os casos de teste com as implementações (da especificação original e das especificações mutantes) e comparando os resultados obtidos. O trabalho de Probert e Guo, diferentemente da proposta apresentada nesta tese (Capítulo 3), define o Teste de Mutação para validar somente o comportamento da especificação. Outra diferença é o fato da especificação ser transformada em programas C para posterior análise dos mutantes. Nessa transformação, juntamente com a interferência humana para completar os códigos, novos erros podem ser introduzidos, comprometendo a aplicação do Teste de Mutação.

Fabbri et al. definem o Teste de Mutação para validação de especificações baseadas em Máquinas de Estado Finito (1994), em Redes de Petri (1995) e em Statecharts (1999a). São definidos conjuntos de operadores de mutação para cada técnica de especificação, com base na classificação de erros de seqüenciamento para MEF (Chow, 1978): erros de saída (a saída de uma transição é incorreta), erros de transferência (o próximo estado de uma transição é incorreto) e erros de estados extras ou ausentes.

Para a técnica Statecharts, são definidos operadores de mutação para as MEFEs do statechart e para os aspectos intrínsecos dessa técnica, como: história, broadcasting e paralelismo (Fabbri, 1996; Fabbri et al., 1999a). O conjunto de operadores de mutação para MEFE engloba os operadores de mutação para MEF e operadores específicos para variáveis e condições. Esses últimos, foram definidos com base nos operadores de mutação para a linguagem C (Agrawal et al., 1989) e com base nos operadores de mutação para expressões booleanas (Weyuker et al., 1994). Os autores definem três estratégias de abstração incrementais para aplicação do Teste de Mutação para Statecharts: *Básica, Baseada em Ortogonalidade e*

Baseada em História. Essas estratégias permitem selecionar os componentes do Statecharts (MEFs) em diferentes níveis de hierarquia, possibilitando que a condução do teste seja realizada através das abordagens *top-down* ou *bottom-up*. A partir dos componentes, o teste da especificação pode partir do nível mais alto de abstração, que corresponde ao próprio Statecharts, ou partir do nível mais baixo de abstração, composto pelos componentes cujos estados são átomos ou básicos.

Petrenko e Bochmann (1996) apontam a relevância do aspecto de cobertura para a atividade de teste fazendo uma comparação entre a cobertura no teste de software e a cobertura no teste de protocolos especificados através de MEF. Nessa comparação são consideradas as técnicas: funcional e baseada em erros. O teste de especificações, considerando MEFs, em geral, é aplicado para realização do teste de conformidade (verificar se a implementação está de acordo com a especificação). Essa relação de conformidade é importante na análise de cobertura pois, a partir dela, determinam-se quais implementações devem ser consideradas errôneas. É definida uma medida de cobertura de erros, similar ao escore de mutação, em que um conjunto de teste é dito completo se existe um caso de teste que distingue, através da saída produzida, toda implementação que não esteja em conformidade com a especificação. Os autores salientam a necessidade de pesquisas que elaborem estratégias de teste combinando vários critérios de cobertura para validação de especificações em MEFE e outras técnicas de descrição formal.

Considerando os trabalhos comentados, observa-se um interesse da comunidade em estabelecer critérios de cobertura para sistemas concorrentes, tanto no nível de programa como no nível de especificação, sendo que no nível de especificação esses critérios se aplicam no teste de conformidade de protocolos de comunicação, em particular, para especificações baseadas em MEF e MEFE.

Os critérios de teste definidos nesta tese procuram responder às necessidades apontadas por Petrenko e Bochmann (1996), fornecendo medidas de cobertura para a atividade de teste de especificações e apresentando mecanismos para sistematizar a condução da atividade de teste. Além disso, os critérios de teste apresentados também podem fornecer subsídios para a realização de testes de conformidade.

2.7. Direções Futuras na Atividade de Teste de Software

Em seu trabalho, Harrold (2000) apresenta as perspectivas de pesquisas na área de teste de software, visando o desenvolvimento de métodos e ferramentas que permitam a transferência de tecnologia para a indústria (Figura 2.3). Pesquisas em diferentes áreas da atividade de teste têm obtido avanços significativos auxiliando no desenvolvimento de softwares de alta qualidade. Conforme ilustradas pelas setas da Figura 2.3, as diferentes áreas de pesquisa se relacionam e podem ser revisitadas durante o processo de desenvolvimento do software.

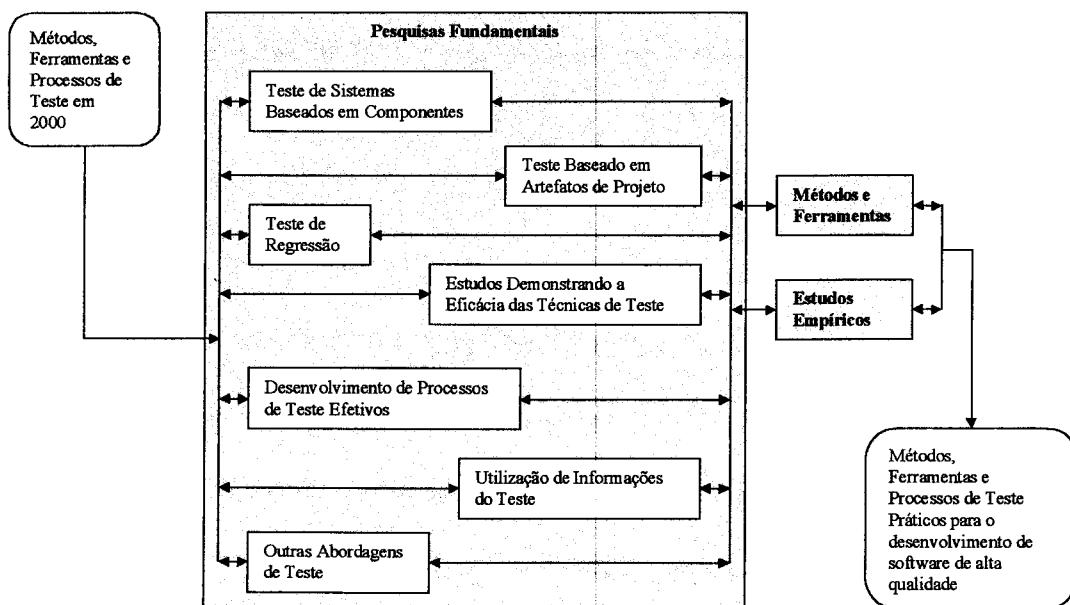


Figura 2.3. Direções de Pesquisa na Área de Teste de Software (Vincenzi, 2000).

Dentre as áreas de pesquisa fundamentais apontadas por Harrold podem-se destacar: *Teste de Sistemas Baseado em Componentes*, *Teste Baseado em Artefatos de Projeto*, *Utilização de Informações do Teste e Estudos Demonstrando a Eficácia das Técnicas de Teste*. O trabalho desta tese está bastante relacionado com essas atividades de pesquisas.

Com relação ao *Teste de Sistemas Baseado em Componentes*, devido ao aumento no tamanho e na complexidade dos softwares, a construção desses sistemas, principalmente sistemas distribuídos, tem sido feita baseada em componentes. Um sistema baseado em componentes é composto de módulos que encapsulam dados e funcionalidades e podem ser configurados, por meio de parâmetros, em tempo de execução. Com o aumento no desenvolvimento de sistemas baseados em componentes é necessária a definição de modos efetivos e eficientes de testá-los. É necessário também entender e desenvolver técnicas e critérios de teste que testem vários aspectos dos componentes, como por exemplo, segurança e tolerância a falhas. O trabalho de Ghosh e Mathur (2000), discutido na Seção 2.4.3, define o Teste de Mutação para o teste sistemas baseados em componentes.

As técnicas de *Teste Baseadas em Artefatos de Projeto* utilizam informações obtidas durante a definição do software, como por exemplo, o projeto, requisitos e a arquitetura do software para o desenvolvimento e o planejamento de casos de teste. A arquitetura do software envolve a descrição dos elementos do sistema, as interações entre esses elementos, padrões que guiam sua composição e restrições sobre os padrões. Através de abstrações do sistema, a arquitetura do software fornece uma maneira promissora de administrar grandes sistemas. Devido ao

tamanho e complexidade dos sistemas, são necessárias técnicas de teste para avaliar a qualidade dos sistemas no início do seu desenvolvimento. Esta tese apresenta contribuições nesse sentido, incluindo a definição do Teste de Mutação para validar os aspectos estruturais ou arquiteturais da especificação de sistemas.

A *Utilização de Informações do Teste* refere-se ao emprego de informações obtidas durante a atividade de teste, por exemplo, caminhos executados pelos casos de teste e resultado da execução dos casos de teste, em outras atividades do desenvolvimento do software. Essas informações podem ser utilizadas, por exemplo, durante os testes de regressão (realizados após a manutenção de software) (Harrold, 2000). De maneira similar, as informações obtidas durante a atividade de teste no nível da especificação (aplicando, por exemplo, os critérios de teste definidos nesta tese) podem ser utilizadas durante os testes de conformidade.

Com relação a *Estudos Demonstrando a Eficácia das Técnicas de Teste*, Harrold (2000) observa que existem alguns trabalhos investigando a eficácia em revelar erros das estratégias de teste, entretanto pesquisas adicionais são necessárias. Um aspecto importante é a identificação de classes de erros e quais os critérios mais efetivos para cada uma das classes. Desse modo, análises estatísticas ou empíricas são fundamentais para fornecer evidências tanto da eficácia de determinado critério em revelar certos tipos de erros como para a definição de estratégias de teste que combinem diversos critérios complementares e permita a detecção da maioria dos erros com um baixo custo de aplicação. No contexto dessa linha de pesquisa, nesta tese, são feitas algumas estudos de casos dos critérios de teste para especificações em Estelle e em Statecharts, fornecendo subsídios para o estabelecimento de estratégias de teste para validação dessas especificações.

Os critérios de teste definidos nesta tese estão encaixados no escopo das direções futuras da atividade de teste de software, pois se apresentam bastante relacionados com essas atividades, explorando os seguintes aspectos: definição de critérios de teste, desenvolvimento de estudos empíricos e subsídios para o desenvolvimento de ferramentas de apoio.

2.8. Considerações Finais

Este capítulo apresentou uma descrição das técnicas de especificação formal, enfatizando as técnicas Statecharts e Estelle, as quais são estudadas nesta tese. Foi apresentada também uma revisão dos principais conceitos relacionados à atividade de teste de software. Foram descritos as técnicas e os critérios para o teste de programas e os resultados de experimentos empíricos para avaliação desses critérios. Os estudos empíricos fornecem evidências do aspecto complementar das técnicas e critérios e fornecem também subsídios para o estabelecimento de estratégias de teste incrementais para a condução da atividade de teste de software.

Uma síntese dos trabalhos relacionados com a validação de especificações formais também foi apresentada. Foram apresentadas algumas técnicas para seleção de seqüências de teste para especificações baseadas em MEFs e MEFEs, utilizadas principalmente no teste de conformidade de protocolos de comunicação. Os critérios de teste propostos por Ural (1987), Probert e Guo (1991), Fabbri (1996), procuram utilizar o conhecimento adquirido no teste de programas, mapeando critérios de teste empregados no nível de programa para o nível de especificação. Essa abordagem é promissora e pode complementar as técnicas de simulação e análise de

alcançabilidade, normalmente empregadas para a validação de especificações baseadas em MEF, MEFEs, Redes de Petri, Statecharts, Estelle, entre outras.

Os critérios de teste propostos nesta tese vêm contribuir nesse sentido, fornecendo subsídios para guiar a seleção de seqüências de teste adequadas ao Teste de Mutação para Estelle, e adequadas aos critérios de Fluxo de Controle para Statecharts e Estelle. Esses critérios são apresentados nos próximos capítulos.

Capítulo 3. Teste de Mutação Aplicado para Especificações Baseadas em Estelle

3.1. Considerações Iniciais

Neste capítulo apresenta-se a definição do critério de teste baseado em mutações para validação de sistemas especificados em Estelle. A técnica Estelle permite a descrição do sistema considerando, essencialmente, as seguintes perspectivas: hierarquia dos módulos do sistema; comunicação entre os módulos e aspectos estruturais (arquiteturais) do sistema. Com base nessas perspectivas, foram identificados os principais tipos de erros que podem ser cometidos durante a especificação de um sistema em Estelle. Esses erros são divididos em três categorias: 1) erros de comportamento; 2) erros de interface; e 3) erros de estruturação (ou arquitetura) do sistema. A partir desses tipos de erros, operadores de mutação para Estelle foram definidos.

A definição desse critério de teste baseou-se nos trabalhos de Probert e Guo (1991) e Fabbri et al. (1994; 1995; 1999a), discutidos no Capítulo 2. Basicamente, esses trabalhos propõem a aplicação do teste de Mutação no contexto de especificações, considerando sistemas especificados em Estelle (Probert e Guo,

1991), Máquinas de Estados Finitos (Fabbri et al., 1994), Redes de Petri (Fabbri et al., 1995) e Statecharts (Fabbri et al., 1999a).

O teste de Mutação apresentado neste capítulo diferencia-se da proposta de Probert e Guo (1991) por apresentar um conjunto de operadores de mutação mais abrangente, considerando aspectos como a interface entre módulos, estruturação (arquitetura) e paralelismo da especificação.

Outra diferença importante entre o trabalho apresentado neste capítulo e o trabalho de Probert e Guo (1991) está na forma em que o Teste de Mutação é aplicado na especificação Estelle. No trabalho de Probert e Guo (1991), a especificação e seus mutantes são transformados em programas na linguagem C para serem executados pelos casos de teste. Essa estratégia está sujeita a erros que podem ser cometidos na tradução da especificação para o programa. Neste trabalho, é proposta a utilização de um simulador (por exemplo, o simulador da ferramenta EDT) para executar a especificação e seus mutantes com os casos de teste, evitando-se os problemas de transformação.

3.2. Teste de Mutação para Estelle

O teste de Mutação é baseado nos erros mais comuns que podem ser cometidos ao longo do desenvolvimento do sistema. Com o objetivo de modelar esses erros, um conjunto de operadores de mutação é definido o qual é aplicado ao produto em teste, gerando versões modificadas do produto, chamadas de mutantes. A definição dos operadores de mutação é um ponto fundamental para aplicação desse critério de teste. Entende-se por operadores de mutação as regras que definem as alterações que devem ser aplicadas ao produto em teste, caracterizando um conjunto de

implementações alternativas. Desse modo, o objetivo é obter um conjunto de casos de teste que garanta que o produto não possui os erros modelados pelos operadores de mutação.

Para a definição do teste de Mutação para Estelle, as hipóteses básicas desse critério no contexto de especificações foram consideradas: *hipótese do projetista competente* – uma especificação construída por um projetista competente está correta ou está próxima do correto; e *hipótese do efeito de acoplamento* – erros complexos são acoplados a erros simples de forma que casos de teste capazes de detectar erros simples são capazes de detectar a maioria dos erros (Fabbri, 1996). Na realidade, esses aspectos precisam ser validados no nível da especificação.

Conforme descrito por Fabbri (1996), a partir de uma especificação S , gera-se um conjunto de mutantes de S , $\phi(S)$, com base em um conjunto de operadores de mutação e diz-se que um conjunto de teste T é adequado para S em relação a $\phi(S)$ se para cada especificação $Z \in \phi(S)$, ou Z é equivalente a S , ou Z difere de S para pelo menos um caso de teste $t \in T$.

Segundo DeMillo e Offutt (1991), existem três condições que precisam ser satisfeitas para que um caso de teste t distinga um mutante M :

- 1) *alcançabilidade*: a execução do mutante M com t deve passar pelo comando St onde a mutação foi feita;
- 2) *necessidade*: o comportamento de M imediatamente após a execução de St (comando onde foi feita a mutação) deve ser diferente do comportamento do programa original neste mesmo ponto, para t ; e
- 3) *suficiência*: o comportamento final de M deve ser diferente do comportamento final do programa original, para t .

Para o critério Mutação de Interface, definido por Delamaro (1997), a condição de alcançabilidade é modificada de modo a refletir as características desse critério de teste que visa a validar cada conexão possível entre dois módulos conectados:

Condição de alcançabilidade para a Mutação de Interface: A execução de M , gerado para a conexão entre um módulo A e um módulo B , com um caso de teste t deve fazer com que o ponto onde a mutação foi feita seja alcançado através de uma chamada de A para B (Delamaro, 1997).

Essa alteração na condição procura garantir que t irá testar a conexão entre módulos que se deseja observar, pois o módulo B pode estar conectado com outros módulos além do módulo A . A idéia dessa alteração é procurar garantir que as mutações em B fiquem ativas somente para a conexão desejada. Essas condições também são consideradas no contexto de especificações Estelle.

O conjunto de operadores de mutação para Estelle foi definido tendo como objetivo torná-lo o mais completo possível, englobando as principais características que podem ser modeladas em Estelle. Os trabalhos relacionados foram revistos e utilizados para inspiração nesse processo. Apesar dos operadores de mutação serem bastante abrangentes, é possível aplicar somente um subconjunto desses operadores, de modo que esse critério possa ser adaptado às restrições de custo para realização da atividade de teste, ou aplicado de acordo com os aspectos que se deseja evidenciar na especificação em teste.

De acordo com as características da técnica Estelle, os tipos de erros e, consequentemente, os operadores de mutação são divididos em 3 categorias: Mutação nos Módulos, Mutação de Interface – Comunicação, Mutação na Estrutura (Arquitetura). Os operadores de mutação de cada classe são apresentados na

Tabela 3.1. A seguir são apresentados as características e os tipos de erros identificados para cada classe de mutação.

Tabela 3.1. Síntese dos Operadores de Mutação para Estelle.

Operadores de Mutação nos Módulos	
1. Substituição de estado inicial	12. Substituição de variável por variável
2. Substituição de estado origem	13. Substituição de variável por constante
3. Substituição de estado destino	14. Incremento e decremento de variáveis/ constantes
4. Remoção de estados	15. Inclusão de operadores unários nas variáveis
5. Remoção de transições	16. Substituição de ação das transições
6. Remoção de condição das transições	17. Cobertura de código
7. Substituição de evento de entrada	18. Remoção de prioridades das transições
8. Remoção de evento de entrada	19. Substituição de prioridades das transições
9. Negação de condição das transições	20. Remoção de atrasos das transições
10. Troca de operadores da condição	21. Substituição de atrasos das transições
11. Substituição de atribuição booleana	22. Substituição de política de fila
Operadores de Mutação de Interface	
<i>Grupo I: Ponto de chamada</i>	<i>Grupo II: Módulo chamado</i>
1. Substituição de parâmetros	7. Substituição de variáveis de interface
2. Incremento/decremento de parâmetros	8. Substituição de variáveis de não interface
3. Troca na ordem dos parâmetros	9. Incremento/decremento de variáveis
4. Inclusão de operadores unários	10. Inclusão de operadores unários nas variáveis
5. Substituição de comando <i>output</i>	11. Substituição de atribuição booleana
6. Remoção de comando <i>output</i>	
Operadores de Mutação na Estrutura	
1. Remoção de conexões	9. Substituição de <i>release</i> por <i>disconnect</i>
2. Inserção de desconexão	10. Substituição de <i>terminate</i> por <i>detach</i>
3. Remoção de desconexão	11. Substituição de <i>terminate</i> por <i>disconnect</i>
4. Remoção de comando <i>release</i>	12. Paralelismo síncrono por execução seqüencial
5. Remoção de comando <i>terminate</i>	13. Paralelismo síncrono por paralelismo assíncrono
6. Substituição de <i>release</i> por <i>terminate</i>	14. Execução seqüencial por paralelismo síncrono
7. Substituição de <i>terminate</i> por <i>release</i>	15. Execução seqüencial por paralelismo assíncrono
8. Substituição de <i>release</i> por <i>detach</i>	

• Mutação nos Módulos

Os operadores de mutação desse grupo modelam erros de comportamento dos módulos da especificação. Considerando que o comportamento dos módulos é expresso por meio de MEFEs, os tipos de erros para os módulos de Estelle foram definidos com base na classificação de erros de seqüenciamento para MEF (Chow, 1978) e com base no trabalho de Fabbri (1996). Fabbri define um conjunto de operadores de mutação para MEFE com base nos operadores de mutação definidos por Agrawal et al. (1989) para a linguagem C e nos operadores de mutação definidos

por Weyuker et al. (1994) para a validação de expressões booleanas. Com base nesses trabalhos e nos aspectos intrínsecos de Estelle, os erros relacionados ao comportamento de módulos em Estelle são:

- erros de transições, de saídas e de estados;
- erros em expressões, operadores, variáveis e constantes;
- erros nas filas dos canais de comunicação;
- erros no atraso para disparo das transições; e
- erros nas prioridades das transições.

• **Mutação de Interface - Comunicação**

Os operadores de mutação desse grupo modelam erros relacionados às interfaces entre os módulos, ou seja, modelam erros relacionados à comunicação entre módulos conectados. A definição dos operadores de mutação dessa classe baseou-se na Mutação de Interface definida por Delamaro (1997). A comunicação entre os módulos de uma especificação em Estelle ocorre através da troca de mensagens e do compartilhamento de variáveis. A troca de mensagens ocorre entre dois módulos conectados por um canal de comunicação bidirecional x , como ilustrado na Figura 3.1. O módulo A envia uma mensagem (uma primitiva de comunicação e seus parâmetros) para o módulo B . A mensagem recebida é armazenada em uma fila até que seja processada com o disparo de uma transição em B , de acordo com a semântica da linguagem Estelle (Budkowski e Dembinski, 1987). Da mesma maneira, o módulo B também pode enviar mensagens ao módulo A . As mensagens produzidas são descritas pelo comando $ip.primitive(parameters)$ associado às transições, onde ip representa o ponto de interação pelo qual a mensagem será enviada e $primitive(parameters)$ representa a primitiva de comunicação e seus

parâmetros (mensagem enviada). Como ocorre com a Mutação de Interface no nível de programas, o teste de mutação de interface em Estelle também é aplicado ponto-a-ponto. Considerando o exemplo da Figura 3.1, o módulo *B* pode ter mais de uma transição que pode processar a mensagem recebida do módulo *A*. Da mesma forma, o módulo *A* pode enviar mais de uma primitiva diferente para o módulo *B*. Assim sendo, tem-se mais de uma conexão entre os módulos *A* e *B*. O teste de mutação de interface considera todas as possíveis conexões entre os módulos.

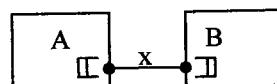


Figura 3.1. Conexão entre Módulos em Estelle.

O segundo tipo de comunicação (compartilhamento de variáveis) só ocorre entre um módulo pai e um módulo filho. A declaração de uma variável do tipo *export* em um dos módulos filhos significa que o módulo pai tem acesso a seu valor. O acesso simultâneo desse tipo de variável pelo pai e por seu filho não é possível por causa do princípio de prioridade de pai/filho: pai e filho nunca agem ao mesmo tempo; quando um módulo pai possui uma transição apta a disparar, ela tem prioridade sobre as transições dos módulos filhos e será disparada.

Nesse sentido, os operadores de Mutação de Interface procuram modelar os seguintes tipos de erros:

- erros no fluxo de dados entre os componentes do sistema;
- erros nas primitivas de comunicação enviadas e/ou recebidas pelos componentes; e
- erros na computação das primitivas recebidas.

- **Mutação na Estrutura (Arquitetura)**

O conjunto de operadores de mutação desse grupo procura modelar erros na arquitetura ou estrutura da especificação Estelle. A arquitetura representa a estrutura hierárquica dos componentes (ou módulos) do software, as interações entre os componentes e os dados que são usados pelos componentes. Shaw e Garlan (1995) descrevem um conjunto de propriedades que devem ser satisfeitas pela arquitetura:

- Propriedades Estruturais – define os componentes (módulos) do sistema e a maneira como esses componentes interagem entre si.
- Propriedades não-funcionais – a descrição da arquitetura do software incorpora características tais como: desempenho, confiabilidade, segurança e adaptabilidade.
- Família de sistemas relacionados – a arquitetura do software deve caracterizar componentes que podem ser utilizados no projeto de softwares similares, de maneira que certos componentes possam ser reutilizados.

Com base nessas propriedades estabelece-se os erros relacionados à arquitetura de especificações em Estelle. Observa-se que os aspectos de interface estão englobados pela arquitetura de software, de maneira que os erros de interface, definidos para Mutação de Interface, são considerados também no contexto de Mutação de Estrutura. Desse modo, quando se desejar validar os aspectos de estrutura ou arquitetura de especificações Estelle as duas classes de mutação (de Interface e de Estrutura) são consideradas. Assim sendo, os erros são divididos em dois grupos:

I. Erros de interface relacionados às conexões:

- erros na definição das conexões entre os componentes;
- erros no fluxo de dados entre os componentes do sistema;

- erros nas primitivas de comunicação enviadas e/ou recebidas pelos componentes;
- erros na computação das primitivas recebidas; e
- erros na finalização das conexões.

II. Erros de Modularização e Paralelismo:

- erros no particionamento dos componentes do sistema;
- erros na inicialização estática ou dinâmica dos componentes; e
- erros no fluxo de controle entre os componentes, isto é, na sincronização e no paralelismo dos componentes.

Os operadores de mutação são descritos nas próximas seções. Na Tabela 3.2 são apresentados os conjuntos de constantes requeridas definidos para serem utilizados nas mutações sobre variáveis. MAX_INT e MAX_REAL representam os valores máximos para variáveis inteiras e reais, respectivamente.

Tabela 3.2. Conjunto de Constantes Requeridas.

Tipo de Variável	Constantes Requeridas
Inteiro	-1, 1, 0, MAX_INT
Real	-1.0, 1.0, 0.0, -0.0, MAX_REAL

Antes de apresentar a definição dos operadores de mutação, é descrito o Protocolo Bit_Alternante que será utilizado para exemplificar cada operador de mutação.

3.2.1. Especificação Exemplo: Protocolo Bit-Alternante

Para ilustrar os operadores de mutação, a especificação do Protocolo Bit-Alternante é utilizada. Essa especificação foi escolhida por ser bastante conhecida e

utilizada. No Apêndice A é apresentada a descrição completa desse protocolo, conforme documento ISO 9074 (1987). A estrutura da especificação em Estelle desse protocolo é composta de um módulo principal, chamado de *system*, o qual possui três módulos filhos (Figura 3.2):

- **Usuário:** descreve os usuários conectados, os quais podem enviar e receber mensagens.
- **A_B:** descreve o protocolo *Bit-Alternante*, o qual procura fornecer uma comunicação confiável sobre um meio de comunicação não confiável. Esse protocolo associa um bit (0 ou 1) em cada mensagem para determinar quando essas mensagens devem ser retransmitidas.
- **Rede:** descrição do meio de comunicação que recebe as mensagens do usuário e envia-as para o usuário destino.

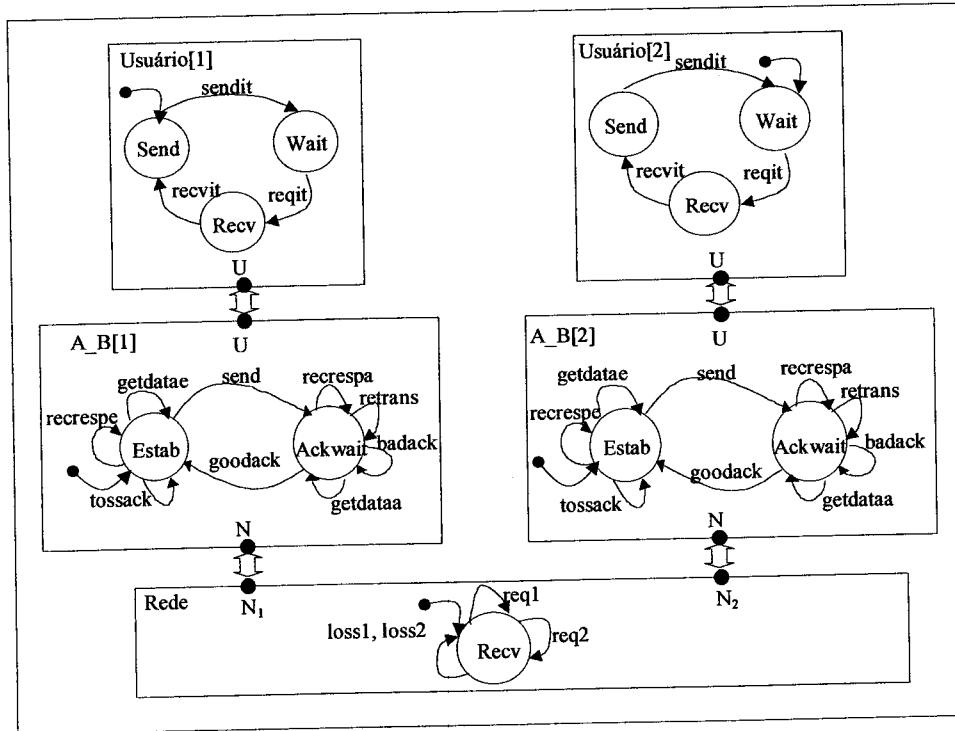


Figura 3.2. Máquinas de Estados Finitos da Especificação do Protocolo Bit-Alternante, conforme descrito na ISO 9074 (1987).

Na Figura 3.2 é representada a estrutura da especificação, contendo as instâncias de módulos, canais de comunicação entre os módulos e as MEFs de cada módulo. Para a descrição em Estelle desse protocolo, são definidas duas instâncias do módulo *Usuário*, sendo que o *Usuário[1]* começa enviando uma mensagem (estado *send*) e o *Usuário[2]* começa esperando receber uma mensagem (estado *wait*).

Para a descrição dos canais de comunicação, é necessário estabelecer quais primitivas de comunicação irão fluir pelo canal. Nesse exemplo existem dois tipos de canais de comunicação (canal *U* e canal *N*), descritos da seguinte forma:

```
channel Uaccesspoint (User, Provider);
  by User:
    SENDrequest (Udata: UDatatype);
    RECEIVErequest;
  by Provider:
    RECEIVEResponse (Udata: UDatatype);

channel Naccesspoint (User, Provider);
  by User:
    DATArequest (Ndata: Ndatatype);
  by Provider:
    DATAresponse (Ndata: Ndatatype);
```

Na declaração dos módulos, estabelecem-se os pontos de interação e o papel (*user* ou *provider*) do módulo em relação a esses pontos de interação:

```
...
{definição do cabeçalho dos módulos}

module Usertype activity (Connendptid : Ceptype);
  ip U: Uaccesspoint(User) common queue;
end;

module Alternatingbittype activity (Connendptid : Ceptype);
  ip {interaction point list }
    U: Uaccesspoint(Provider) common queue;
    N: Naccesspoint(User) individual queue;
end;

module Networktype activity;
  ip N : array[Ceptype] of Naccesspoint(Provider)
  individual queue;
end;
...
```

Para o módulo *A_B* existem dois pontos de interação: *U* – em que o papel de *A_B* é *provider*, ou seja, envia a primitiva de comunicação *RECEIVEResponse* e recebe as primitivas *SENDrequest* e *RECEIVErequest*, e *N* – em que o papel é *user*,

ou seja, A_B envia *DATArequest* e recebe *DATAresponse*. Para cada ponto de interação é definido também se a fila que armazena as mensagens recebidas é individual ao ponto de interação ou é comum aos pontos de interação do módulo.

Por exemplo, para o ponto de interação *N* é definida uma fila individual para armazenar as mensagens recebidas. Se a fila fosse comum (com a fila de *U*) ocorreria *deadlock* no sistema na seguinte situação: *SENDrequest* (para enviar dados) é enviada pelo módulo *Usuário* e armazenada na fila enquanto o módulo A_B está no estado *ACKWAIT* (esperando confirmação do recebimento). Neste caso, ocorre *deadlock* porque *SENDrequest* só é retirada da fila pela transição *SEND* (estado *ESTAB*). Como o critério da fila é *FIFO*, o módulo não é capaz de processar uma interação recebida pelo módulo *Rede* (*DATAresponse*) que determina a mudança de estado da MEFE, fazendo com que o sistema não possa evoluir. Esse aspecto ilustra um erro que poderia ocorrer na especificação caso a política de fila não fosse definida corretamente. Esse tipo de erro (mutação) é feito pelo operador *Substituição de política de fila*.

Para a descrição do comportamento dos módulos, são definidos os valores iniciais das variáveis e estados e o comportamento das transições da MEFE. Considerando o módulo *A_B*, a inicialização e as transições são descritas da seguinte forma:

```

    .. initialize {inicialização do módulo A_B}

    to ESTAB {estado inicial da MEFE de A_B}
    begin { inicializa variáveis }
        Sendseq := 0;
        Recvseq := 0;
        Emptybuf(Sendbuffer); {esvazia buffer de msg enviadas }
        Emptybuf(Recvbuffer); {esvazia buffer de msg recebidas }
    end;

    trans {início da descrição das transições da MEFE de A_B}
    from ESTAB
    to ACKWAIT
    when U.SENDrequest

```

```

name send:
begin
copy(P.Msgdata,Udata);           {copia dados do usuário em P}
P.Msgseq := Sendseq;
Store(Sendbuffer,P);            {armazena P no buffer }
Formatdata(P,B);
output N.DATArequest(B);
end;

...
from ESTAB
to same
when N.DATAreponse
provided Ndata.Id = DATA
name getdatae:
begin                                { transition 5 }
copy (Q.Msgdata,Ndata.Data);
Q.Msgseq := Ndata.Seq;
Formatack(Q,B);
output N.DATArequest(B);
if Ndata.Seq = Recvseq then
begin
Store(Recvbuffer,Q);
Increcvseq;
end
end;

```

Define-se o estado *ESTAB* como estado inicial da MEFE. Para as transições são definidos: estado origem da transição (*from*); estado destino (*to*); evento de entrada (*when*), condição (*provided*); ação da transição (bloco *begin-end*) e saída da transição (*output*). As cláusulas são opcionais; por exemplo, a transição *send* não possui condição (cláusula *provided*).

Os operadores de mutação de cada classe são apresentados e exemplificados a seguir. Para alguns operadores de mutação, não é possível realizar mutações na especificação do protocolo Bit_Alternante. Isso ocorre porque a especificação não apresenta os elementos necessários para que o operador seja aplicado. Quando necessário, são apresentadas as restrições que devem ser satisfeitas pela especificação para que o operador de mutação seja aplicado de modo a gerar somente mutantes sintaticamente corretos.

3.2.2. Definição dos Operadores de Mutação nos Módulos

1. Substituição de estado inicial

Esse operador substitui o estado definido como inicial da MEFE pelos demais estados de modo que em cada mutante um dos outros estados passa a ser o estado inicial.

Exemplo:

Especificação Original	Especificação Mutante
<pre>initialize to ESTAB begin Sendseq := 0; Recvseq := 0; Emptybuf(Sendbuffer); Emptybuf(Recvbuffer); end;</pre>	<pre>initialize → to ACKWAIT begin Sendseq := 0; Recvseq := 0; Emptybuf(Sendbuffer); Emptybuf(Recvbuffer); end;</pre>

2. Substituição de estado origem

Esse operador substitui o estado origem de cada transição pelos demais estados definidos para o módulo em teste. Considera também os estados declarados como *stateset*. Quando o estado origem S_i for trocado por *stateset* $S_x = \{S_j, \dots, S_n\}$ significa que a transição t terá como estado origem cada um dos estados pertencentes a S_x .

Restrição: o estado origem S_i de uma transição t só pode ser trocado pelo estado S_j se t não existe no conjunto de transições disparadas a partir de S_i , evitando, dessa forma, que seja inserido não-determinismo na especificação mutante.

Exemplo:

Transição Original	Transição Mutante
<pre>trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata, Udata); P.Msgseq := Sendseq; Store(Sendbuffer, P); Formatdata(P, B); output N.DATArequest(B); end;</pre>	<pre>trans → from ACKWAIT to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata, Udata); P.Msgseq := Sendseq; Store(Sendbuffer, P); Formatdata(P, B); output N.DATArequest(B); end;</pre>

3. Substituição do estado destino

Esse operador substitui o estado destino de cada transição pelos outros estados definidos para o módulo em teste.

Exemplo:

Transição Original	Transição Mutante
<pre> Trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> trans from ESTAB → to ESTAB when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>

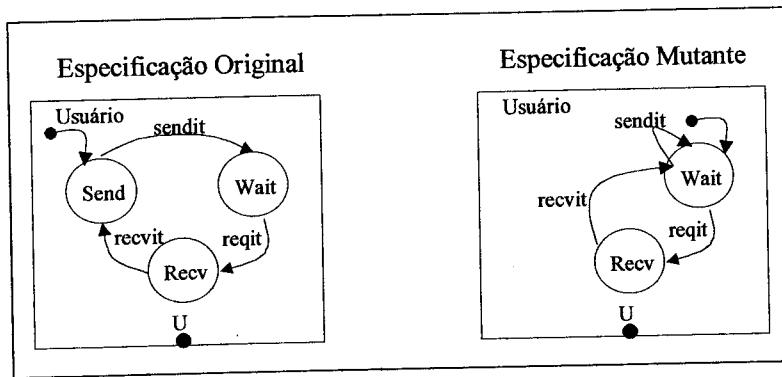
4. Remoção de estados

Esse operador remove os estados da MEFE. O estado a ser removido na verdade é unido a outro estado, desde que esses estados estejam conectados por meio de uma transição. O estado resultante passa a ser o estado origem e/ou destino de todas as transições que tinham como origem e/ou destino os estados que foram agrupados. Assim, todos os estados da MEFE são combinados para geração dos mutantes.

Restrição: é aplicado para MEFEs contendo mais de 2 estados e o estado origem S_i de uma transição t só pode ser trocado pelo estado S_j se t não existe no conjunto de transições disparadas a partir de S_j , evitando, dessa forma, que seja inserido não-determinismo na especificação mutante.

Exemplo: é removido o estado *Send* do módulo *Usuário*. No mutante gerado, *Send* é unido ao estado *Wait*.

A Figura que segue ilustra graficamente:



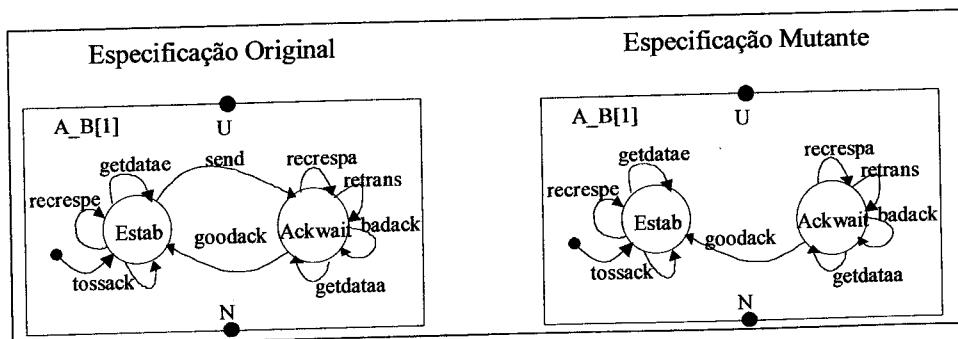
Especificação Original	Especificação Mutante
<pre> initialize to SEND provided Connendptid = 1 begin end; to WAIT provided Connendptid = 2 begin end; trans from SEND to WAIT delay(30) name SENDIT: begin udata.size := 5; udata.info := 'Hello '; output U.SENDrequest(udata); end; </pre>	<pre> initialize → to WAIT provided Connendptid = 1 begin end; to WAIT provided Connendptid = 2 begin end; trans → from WAIT to WAIT delay(30) name SENDIT: begin udata.size := 5; udata.info := 'Hello '; output U.SENDrequest(udata); end; </pre>

5. Remoção de transições

Esse operador elimina as transições da MEFE.

Exemplo: a transição *SEND* é removida da MEFE do módulo A_B.

A Figura que segue ilustra graficamente:



Transição Original	Transição Mutante
<pre> to ESTAB begin Sendseq := 0; Rcvseq := 0; Emptybuf (Sendbuffer); Emptybuf (Recvbuffer); end; trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> to ESTAB begin Sendseq := 0; Rcvseq := 0; Emptybuf (Sendbuffer); Emptybuf (Recvbuffer); end; trans → </pre>

6. Remoção de condição das transições

Esse operador remove a cláusula *provided* das transições, a qual define uma condição para que a transição possa ser disparada. Uma transição em que somente a cláusula *provided* não é satisfeita, ao sofrer a mutação por esse operador, passa a ser habilitada.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when N.DATAreponse provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATArequest(B); if Ndata.Seq = Rcvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATAreponse → name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATArequest(B); if Ndata.Seq = Rcvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

7. Substituição de evento de entrada

Esse operador substitui o evento de entrada de cada transição *t* pelos eventos de entrada das outras transições da MEFE, diferentes de *t*.

Restrição: esse operador só pode ser aplicado para eventos de entrada que possuem os mesmos tipos de parâmetros, evitando a geração de mutantes com erros sintáticos. Além disso, o evento de entrada E_i de uma transição t (com estado origem S) só pode ser trocado pelo evento E_j se E_j não existe no conjunto de transições disparadas a partir do estado origem S , evitando, dessa forma, que seja inserido não-determinismo na especificação mutante.

8. Remoção de evento de entrada

Esse operador elimina o evento de entrada de cada transição, sendo que cada transição passa a se comportar como uma transição espontânea.

Exemplo:

Transição Original	Transição Mutante
<pre>from ESTAB to same when U.RECEIVErequest provided not bufferempty(Recvbuffer) name recrespe: begin Q := Retrieve(Recvbuffer); output U.RECEIVEresponse(Q.Msgdata); Remove(Recvbuffer); end;</pre>	<pre>from ESTAB to same → provided not bufferempty(Recvbuffer) name recrespe: begin Q := Retrieve(Recvbuffer); output U.RECEIVEresponse(Q.Msgdata); Remove(Recvbuffer); end;</pre>

9. Negação de condição das transições

Esse operador é aplicado nas condições (cláusula *provided*) das transições. É inserido o operador de negação lógica (*not*) em frente da condição da cláusula *provided*.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATARESPONSE → provided not(Ndata.Id = DATA) name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

10. Troca de operadores da condição

Esse operador de mutação é aplicado nas condições (cláusula *provided*) das transições. É substituída cada ocorrência de um operador relacional {=, \neq , $>$, $<$, \leq , \geq } por cada um dos demais operadores relacionais. É substituída também cada ocorrência de um operador lógico {*and*, *or*} pelos operadores lógicos desse conjunto.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATARESPONSE → provided Ndata.Id < DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

11. Substituição de atribuição booleana

Esse operador troca cada atribuição de variáveis booleanas de *true* por *false* e vice-versa.

12. Substituição de variável por variável

Esse operador substitui cada ocorrência de uma variável nas transições e na inicialização pelas demais variáveis, de tipos compatíveis, declaradas pelo módulo ou por seus módulos hierarquicamente superiores.

Exemplo:

Transição Original	Transição Mutante
<pre> From ESTAB to same when N.DATAresponse provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREquest(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATAresponse provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREquest(B); if Ndata.Seq = Recvseq then begin → Store (Sendbuffer,Q); Increcvseq; end end; </pre>

13. Substituição de variável por constante

Esse operador substitui cada ocorrência de uma variável nas transições e na inicialização por constantes requeridas (Tabela 3.2) e por constantes de tipos compatíveis, declaradas pelo módulo ou por módulos hierarquicamente superiores.

Exemplo:

Transição Original	Transição Mutante
<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATAREquest(B); end; </pre>	<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); → P.Msgseq := 1; Store(Sendbuffer,P); Formatdata(P,B); output N.DATAREquest(B); end; </pre>

14. Incremento e decremento de variáveis e constantes

Esse operador de mutação é aplicado no uso de variáveis e constantes de tipos escalares. Esse operador faz um incremento (*var +1*) e um decremento (*var -1*) para cada uso de uma variável ou constante nas transições e na inicialização.

Exemplo:

Transição Original	Transição Mutante
<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); → P.Msgseq := (Sendseq + 1); Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>

15. Inclusão de operador unário nas variáveis

Esse operador inclui o operador de negação aritmética (símbolo $-$) em cada ocorrência de uma variável ou constante de tipo escalar nas transições e na inicialização.

Exemplo:

Transição Original	Transição Mutante
<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); → P.Msgseq := -Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>

16. Substituição de ação das transições

Esse operador de mutação modifica a ação das transições trocando o bloco de comandos entre *begin* e *end* de uma transição pelo bloco das demais transições.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when U.RECEIVErequest provided not bufferempty(Recvbuffer) name recrespe: begin Q := Retrieve(Recvbuffer); output U.RECEIVEResponse(Q.Msgdata); Remove(Recvbuffer); End; </pre>	<pre> from ESTAB to same when U.RECEIVErequest provided not bufferempty(Recvbuffer) name recrespe: → begin P := Retrieve(Sendbuffer); Formatdata(P,B); output N.DATAREquest(B); end; </pre>

17. Cobertura de Código

Os operadores de mutação desse grupo são do tipo instrumentado, procurando selecionar casos de teste que forneçam coberturas mínimas, relacionadas ao fluxo de controle dos comandos entre *begin* e *end* das transições. Existem dois operadores de mutação nesse grupo. O primeiro, insere uma função chamada *Trap-On-Execution* no início de cada bloco de comandos, onde a execução dessa função faz com que o mutante seja distinguido. O segundo operador de mutação garante a cobertura de todos os desvios. Isso é feito inserindo as funções *Trap-On-True(expr)* e *Trap-On-False(expr)* onde *expr* é uma expressão (ou condição) que precisa ser satisfeita para que o desvio (por exemplo, um comando *if*) seja executado. Para a primeira função, o mutante será distinguido quando *expr* tiver um valor diferente de zero; na segunda, o mutante será distinguido quando *expr* assumir um valor igual a zero. Assim, cada ramo do comando é executado pelo menos uma vez.

Exemplo:

Transição Original	Transição Mutante
<pre> From ESTAB to same when N.DATAreponse provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATArequest(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATAreponse provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATArequest(B); → if Trap_On_True(Ndata.Seq = Recvseq) then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

18. Remoção de prioridades das transições

Esse operador elimina a cláusula *priority* das transições, a qual estabelece uma prioridade de disparo para a transição. Essa mutação pode modificar a ordem de disparo das transições.

19. Substituição de prioridades das transições

Esse operador substitui o valor da prioridade das transições (cláusula *priority*) pelo valor da prioridade das demais transições e também pelos valores das constantes requeridas não negativas (Tabela 3.2), de modo que a ordem de disparo das transições seja modificada.

20. Remoção de atrasos das transições

Esse operador elimina a cláusula *delay* de cada transição espontânea (transições sem eventos de entrada). Essa cláusula estabelece um tempo mínimo e máximo que uma transição pode atrasar antes que ela seja disparada.

Exemplo:

Transição Original	Transição Mutante
<pre> from ACKWAIT to ACKWAIT delay (Retrantlyme) name retrans: begin P := Retrieve(Sendbuffer); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> from ACKWAIT to ACKWAIT → name retrans: begin P := Retrieve(Sendbuffer); Formatdata(P,B); output N.DATArequest(B); end; </pre>

21. Substituição de atrasos das transições

Esse operador altera o valor do tempo mínimo e máximo para atraso no disparo das transições espontâneas (cláusula *delay*). Para *delay(t1, t2)*, *t1* e *t2* são substituídos pelos valores do conjunto $\{t1, t2, 0, 1, *\}$ e pelas demais variáveis e constante de tipo escalar definidas.

Para entender o significado da cláusula *delay*, considere uma transição espontânea *T* com *delay(t1, t2)*. A partir do momento que *T* torna-se habilitada (apta para ser disparada), ela precisa permanecer habilitada por *t1* unidades de tempo até que ela possa ser oferecida para ser disparada. A execução de *T* depende das demais transições habilitadas no momento (considerando prioridade e não determinismo). Entre *t1* e *t2* unidades de tempo, *T* pode ou não ser disparada, dependendo das demais transições. Entretanto, quando *t2* unidades de tempo ocorrem, *T* deve ser oferecida pelo módulo para execução. Caso *t2* seja igual a ***, significa que não existe tempo máximo estipulado e, portanto, não existe garantia que *T* será oferecida para disparo em algum momento.

Exemplo:

Transição Original	Transição Mutante
<pre> from ACKWAIT to ACKWAIT delay (Retrantime) name retrans: begin P := Retrieve(Sendbuffer); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> from ACKWAIT to ACKWAIT → delay (Retrantime, *) name retrans: begin P := Retrieve(Sendbuffer); Formatdata(P,B); output N.DATArequest(B); end; </pre>

22. Substituição de política de fila

Esse operador modela erros relacionados com interações colocadas em filas erradas. Isto acontece quando um ponto de interação com fila individual é modificado para possuir uma fila comum, a qual será compartilhada com os outros pontos de interação. Esse operador altera a política de fila de cada ponto de interação do módulo. Quando a política de fila é comum, o mutante terá política de fila individual e vice-versa. Quando o ponto de interação não possui uma política de fila especificada, a troca é baseada na política de fila definida como *default* pelo módulo principal.

Exemplo:

Especificação Original	Especificação Mutante
<pre> module Alternatingbittype activity (Connendptid : Ceptype); ip {interaction point list } U: Uaccesspoint(Provider) common queue; N: Naccesspoint(User) individual queue; end; </pre>	<pre> module Alternatingbittype activity (Connendptid : Ceptype); ip {interaction point list } U: Uaccesspoint(Provider) common queue; → N: Naccesspoint(User) common queue; end; </pre>

3.2.3. Definição dos Operadores de Mutação de Interface

Os operadores de mutação desse grupo foram definidos com base no critério Mutação de Interface, definido por Delamaro (1997). Assim, dois grupos de operadores de mutação foram definidos:

- *Grupo I: Mutações no ponto de chamada:* os operadores deste grupo são aplicados onde ocorre uma chamada a um módulo, isto é, no comando *output*. Os operadores são aplicados principalmente na primitiva enviada e em seus parâmetros.
- *Grupo II: Mutações no módulo chamado:* os operadores deste grupo são aplicados somente nos comandos que executam computações com os dados recebidos.

O conceito de *variáveis de interface* – conjunto de variáveis passadas como parâmetros à função chamada e as variáveis globais usadas na função chamada – definido por Delamaro (1997) também é empregado no contexto de Estelle, sendo que em Estelle, o conjunto de variáveis de interface é formado pelos parâmetros das primitivas de comunicação enviados para um módulo e também pelas variáveis exportadas pelos módulos filhos.

Ghosh e Mathur (2000) investigam o uso da Mutação de Interface definida por Delamaro no contexto de teste de sistemas baseados em componentes. Os operadores de mutação definidos por Ghosh e Mathur correspondem aos operadores de mutação do Grupo I , que realizam as mutações no ponto de chamada.

Grupo I – Mutações Efetuadas no Ponto de Chamada (comando *output*)

1. Substituição de parâmetros

Esse operador substitui cada parâmetro da primitiva de comunicação do comando *output* por uma variável, constante requerida (Tabela 3.2) ou constante de tipo compatível, declarada pelo módulo ou por módulos hierarquicamente superiores.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when U.RECEIVErequest provided not bufferempty(Recvbuffer) name recrespe: begin Q := Retrieve(Recvbuffer); output U.RECEIVEresponse(Q.Msgdata); Remove(Recvbuffer); end; </pre>	<pre> from ESTAB to same when U.RECEIVErequest provided not bufferempty(Recvbuffer) name recrespe: begin Q := Retrieve(Recvbuffer); → output U.RECEIVEresponse(P.Msgdata); Remove(Recvbuffer); end; </pre>

2. Incremento e decremento de parâmetros

Esse operador incrementa e decrementa cada um dos parâmetros da primitiva de comunicação de tipos escalares. Isso é feito adicionando-se e subtraindo-se o valor 1 a cada parâmetro.

3. Operador de troca na ordem dos parâmetros

Esse operador troca a ordem em que os parâmetros aparecem na chamada. Isso é feito entre pares de parâmetros de tipos compatíveis.

4. Inclusão de operador unário nos parâmetros

Esse operador inclui o operador de negação aritmética (símbolo $-$) em cada parâmetro de tipo escalar da primitiva de comunicação.

5. Substituição de comando *output*

Esse tipo de mutação modifica a saída emitida pelas transições. Esse operador substitui o evento de saída do comando *output* de cada transição pelo evento de saída das demais transições.

Exemplo:

Transição Original	Transição Mutante
<pre>from SEND to WAIT delay(30); name sendit begin Udata.size := 5; Udata.info := 'Hello '; output U.Send_Request(Udata); end;</pre>	<pre>from SEND to WAIT delay(30); name sendit begin Udata.size := 5; Udata.info := 'Hello '; →output U.Receive_Request; end;</pre>

6. Remoção de comando *output*

Esse operador elimina o comando *output* de cada transição. Isso faz com que a execução da transição não emita saída.

Exemplo:

Transição Original	Transição Mutante
<pre>from SEND to WAIT delay(30); name sendit begin Udata.size := 5; Udata.info := 'Hello '; output U.Send_Request(Udata); end;</pre>	<pre>from SEND to WAIT delay(30); name sendit begin Udata.size := 5; Udata.info := 'Hello '; → end;</pre>

Grupo II – Mutações Efetuadas no Módulo Chamado

7. Substituição de variáveis de interface

Esse operador substitui cada uso de uma variável de interface (nas transições e na inicialização) por constantes requeridas (Tabela 3.2), por variáveis e por constantes, de tipos compatíveis, declaradas pelo módulo ou por módulos hierarquicamente superiores.

Exemplo:

Transição Original	Transição Mutante
<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>	<pre> trans from ESTAB to ACKWAIT when U.SENDrequest name send: begin → copy(P.Msgdata, Q.Msgdata); P.Msgseq := Sendseq; Store(Sendbuffer,P); Formatdata(P,B); output N.DATArequest(B); end; </pre>

8. Substituição de variáveis de não interface

Essa mutação atua nas variáveis que não são de interface, mas que podem afetar algum valor de interface. Assim, esse operador substitui cada ocorrência de uma variável de não interface, que aparece em expressões contendo pelo menos uma variável de interface, por variáveis e constantes do mesmo tipo declaradas pelo módulo ou por módulos hierarquicamente superiores, e por constantes requeridas (Tabela 3.2).

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to ACKWAIT when U.Send_Request name send begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_Request(B); end; </pre>	<pre> from ESTAB to ACKWAIT when U.Send_Request name send begin → copy(Q.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_request(B); end; </pre>

9. Incremento e decremento de variáveis

Esse operador incrementa e decrementa cada uso de variáveis de interface e não interface (tipo escalar), que aparecem em expressões contendo pelo menos uma

variável de interface. Isso é feito adicionando-se e subtraindo-se o valor 1 a cada variável.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); → if Ndata.Seq = (Recvseq + 1) then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

10. Inclusão de operador unário nas variáveis

Esse operador inclui o operador de negação aritmética (símbolo -) em cada uso de variável de interface (tipo escalar) no módulo. É aplicado também nas variáveis de não interface (escalares) que aparecem em expressões contendo pelo menos uma variável de interface.

Exemplo:

Transição Original	Transição Mutante
<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); if Ndata.Seq = Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>	<pre> from ESTAB to same when N.DATARESPONSE provided Ndata.Id = DATA name getdatae: begin copy (Q.Msgdata,Ndata.Data); Q.Msgseq := Ndata.Seq; Formatack(Q,B); output N.DATAREQUEST(B); → if Ndata.Seq = -Recvseq then begin Store(Recvbuffer,Q); Increcvseq; end end; </pre>

11. Substituição de atribuição booleana

Esse operador troca cada ocorrência de uma atribuição *true* em variáveis de interface booleanas por *false* e vice-versa, aplicado tanto na descrição das transições como na inicialização da especificação.

3.2.4. Definição dos Operadores de Mutação na Estrutura

Os operadores de mutação dessa classe foram divididos em 2 grupos: *estrutura* – operadores de mutação procuram validar os aspectos estruturais, ou seja, nas conexões definidas; e *parallelismo* – operadores de mutação que procuram validar o paralelismo estabelecido para execução dos módulos.

Grupo I – Mutações Efetuadas na Estrutura da Especificação

Quando um sistema é especificado em Estelle é possível descrever sua estrutura ou arquitetura, apresentando a hierarquia entre os módulos e suas conexões. A estrutura pode modificar-se em tempo de execução, caracterizando-se uma estrutura dinâmica. Isso ocorre, por exemplo, quando instâncias de módulos são criadas e destruídas em tempo de execução.

Antes de apresentar os operadores de mutação é necessário diferenciar alguns comandos Estelle para conexão e desconexão de módulos:

- ***connect/disconnect***: a operação *connect* é utilizada para realizar a conexão entre dois pontos de interação do mesmo tipo (os pontos de interação são declarados como canais de comunicação). Esses pontos de interação pertencem a módulos diferentes, os quais podem ser módulos irmãos ou módulo pai e filho. A operação *disconnect* realiza a desconexão de pontos de interação conectados através de uma operação *connect*, ou realiza a desconexão de todos

os pontos de interação de um módulo filho. Quando um ponto de interação é desconectado, as interações existentes na fila do ponto de interação continuam lá e podem ser processadas mesmo após realizada a desconexão.

- ***attach/detach***: a operação *attach* é um tipo especial de conexão que ocorre entre um módulo pai e um módulo filho. Quando a operação *attach* é realizada, as interações presentes na fila do ponto de interação do módulo pai são removidas da fila e inseridas na fila do ponto de interação do módulo filho que está sendo conectado. A partir dessa conexão todas as interações recebidas pelo ponto de interação do módulo pai são inseridas na fila do módulo filho. A operação *detach* é realizada pelo módulo pai e faz com que os pontos de interações conectados por um comando *attach* sejam desconectados. Quando a operação *detach* é realizada as interações presentes na fila do ponto de interação do módulo filho são removidas e adicionadas na fila do ponto de interação do módulo pai relativo à conexão.
- ***release e terminate***: esses comandos realizam a finalização de módulos filhos. Nos dois casos, antes do módulo filho ser finalizado, todas as conexões *connect* e *attach* são desconectadas (*disconnect* e *detach*). A diferença entre esses comandos está na forma em que as conexões *attach* são desconectadas. Considere dois pontos de interação *ip1* (módulo pai) e *ip2* (módulo filho) conectados pelo comando *attach*. No caso do comando *release*, as interações presentes na fila de *ip2*, recebidas por *ip1*, são removidas dessa fila e adicionadas à fila de *ip1*. No caso do comando *terminate*, ocorre uma operação do tipo *detach*: nenhuma interação de *ip2* é movida para *ip1*, sendo que essas interações são perdidas quando ocorre a destruição da conexão. Ou seja, a seqüência de comandos: *detach X;*

terminate X; é semanticamente equivalente ao comando *release X* (ISO, 1987).

A seguir são descritos e exemplificados os operadores de mutação desta classe.

1. Remoção de conexões

Esse operador elimina cada uma das conexões entre módulos.

Exemplo:

Especificação Original	Especificação Mutante
init Alternatingbit[Cep] with Alternatingbitbody(Cep);	init Alternatingbit[Cep] with Alternatingbitbody(Cep);
connect User[Cep].U to Alternatingbit[Cep].U;	→
connect Alternatingbit[Cep].N to Network.N[Cep];	connect Alternatingbit[Cep].N to Network.N[Cep];
end;	end;

2. Inserção de desconexão

Esse operador insere um comando de desconexão *disconnect* para cada comando de conexão *connect* e um comando de desconexão *detach* para cada comando de conexão *attach*. Para cada conexão é inserida uma desconexão nas transições do módulo; caso o módulo não possua transições a desconexão é adicionada na parte de inicialização do módulo.

Exemplo:

Especificação Original	Especificação Mutante
init Alternatingbit[Cep] with Alternatingbitbody(Cep);	init Alternatingbit[Cep] with Alternatingbitbody(Cep);
connect User[Cep].U to Alternatingbit[Cep].U;	connect User[Cep].U to Alternatingbit[Cep].U;
connect Alternatingbit[Cep].N to Network.N[Cep];	connect Alternatingbit[Cep].N to Network.N[Cep];
end;	→ disconnect User[Cep].U; end;

3. Remoção de desconexão

Esse operador de mutação remove cada um dos comandos de desconexão *disconnect* e *detach* da especificação. Esses comandos podem estar presentes na parte de inicialização dos módulos ou nas ações das transições dos módulos.

4. Remoção de comando *release*

O comando *release* remove todas as conexões (tanto *connect* como *attach*) do módulo e libera o módulo e todos os seus descendentes. Esse operador remove esse comando, fazendo com que um módulo que deveria ser destruído continue instanciado.

5. Remoção de comando *terminate*

O comando *terminate* remove todas as conexões (tanto *connect* como *attach*) do módulo e libera o módulo e todos os seus descendentes. Esse operador remove esse comando, fazendo com que um módulo que deveria ser destruído continue instanciado.

6. Substituição de *release* por *terminate*

Esse operador substitui todos os comandos *release* por comandos *terminate*.

7. Substituição de *terminate* por *release*

De forma inversa ao Operador 6, esse operador substitui cada ocorrência de um comando *terminate* por um comando *release*.

8. Substituição de *release* por *detach*

Esse operador substitui cada comando *release* por um comando *detach*. Isso faz com que o módulo que deveria ser liberado pelo comando *release* continue instanciado, sendo desconectados somente os seus pontos de interação.

9. Substituição de *release* por *disconnect*

Esse operador substitui cada comando *release* por um comando *disconnect*. Isso faz com que o módulo que deveria ser liberado pelo comando *release* continue instanciado, sendo desconectados somente os seus pontos de interação.

10. Substituição de *terminate* por *detach*

Esse operador substitui cada comando *terminate* por um comando *detach*. Isso faz com que o módulo que deveria ser liberado pelo comando *terminate* continue instanciado, sendo desconectados somente os seus pontos de interação.

11. Substituição de *terminate* por *disconnect*

Esse operador substitui cada comando *terminate* por um comando *disconnect*. Isso faz com que o módulo que deveria ser liberado pelo comando *terminate* continue instanciado, sendo desconectados somente os seus pontos de interação.

Grupo II – Mutações Efetuadas no Paralelismo da Especificação

Os operadores de mutação descritos a seguir foram definidos procurando englobar os diferentes tipos de execuções dos módulos. Devido aos aspectos sintáticos de Estelle, algumas regras para as modificações precisam ser estipuladas de forma que sejam gerados somente mutantes sintaticamente corretos.

O tipo de paralelismo para especificações em Estelle é determinado pelo atributo definido para os módulos, sendo que podem existir 3 tipos de execuções: 1) execução síncrona dos módulos; 2) execução seqüencial dos módulos; e 3) execução assíncrona dos módulos. A forma de execução é definida pelo atributo dos módulos, que pode ser: *systemprocess*, *systemactivity*, *process* e *activity*. O atributo do módulo pai influencia o paralelismo dos módulos filhos. Módulos com atributo *systemprocess* ou *systemactivity* são chamados *system* e entre eles ocorre paralelismo assíncrono. Dentro de um módulo *systemprocess*, ocorre paralelismo síncrono e o atributo de seus módulos filhos pode ser *process* ou *activity*. Os filhos de um módulo *process* podem ser *process* ou *activity* e são executados em paralelo sincronamente. Isso significa que uma transição de cada módulo é selecionada em cada passo e essas transições são executadas em paralelo. Por outro lado, os filhos de um módulo *activity* possuem somente o atributo *activity* e são executados seqüencialmente, ou seja, uma transição de um dos módulos é selecionada aleatoriamente para execução em cada passo. No caso de módulos *systemactivity* seus módulos filhos possuem somente o atributo *activity* e com isso só ocorre execução seqüencial (Budkowski e Dembinski, 1987).

12. Operador de troca de paralelismo síncrono por execução seqüencial

Esse operador troca o paralelismo síncrono dos módulos por execução seqüencial. Para gerar mutantes sintaticamente corretos, as seguintes modificações precisam ser feitas:

- i) trocar o atributo *systemprocess* do módulo *system* para *systemactivity*; e
- ii) trocar o atributo *process* de todos os módulos descendentes por *activity*.

13. Operador de troca de paralelismo síncrono por paralelismo assíncrono

Esse operador troca o paralelismo síncrono dos módulos por paralelismo assíncrono.

Restrição: esse operador só pode ser aplicado quando os módulos *system* (*systemprocess* e *systemactivity*) da especificação original não possuírem transições, porque com a mutação esses módulos ficarão sem atributo e módulos sem atributo não podem conter transições.

As seguintes modificações precisam ser feitas para gerar as especificações mutantes:

- i) trocar o atributo *systemprocess* do módulo *system* para sem atributo; e
- ii) trocar o atributo *process* e *activity* dos módulos descendentes do módulo *system* por *systemprocess* e *systemactivity*.

14. Operador de troca de execução seqüencial por paralelismo síncrono

Esse operador troca a execução seqüencial dos módulos por paralelismo síncrono. As seguintes modificações precisam ser feitas para gerar as especificações mutantes:

- i) trocar o atributo *systemactivity* do módulo *system* para *systemprocess*; e
- ii) trocar o atributo *activity* de todos os módulos descendentes por *process*.

Exemplo:

Especificação Original	Especificação Mutante
<pre> specification AB systemactivity; ... module Usertype activity (Connendptid : Ceptype); ip U: Uaccesspoint(User) common queue; end; module Alternatingbittype activity (Connendptid : Ceptype); ip U: Uaccesspoint(Provider) common queue; N: Naccesspoint(User) individual queue; end; module Networktype activity; ip N : array[Ceptype] of Naccesspoint(Provider) individual queue; end; ... </pre>	<pre> →specification AB systemprocess; ... → module Usertype process (Connendptid : Ceptype); ip U: Uaccesspoint(User) common queue; end; → module Alternatingbittype process (Connendptid : Ceptype); ip U: Uaccesspoint(Provider) common queue; N: Naccesspoint(User) individual queue; end; → module Networktype process; ip N : array[Ceptype] of Naccesspoint(Provider) individual queue; end; ... </pre>

15. Operador de troca de execução seqüencial por paralelismo assíncrono

Esse operador troca a execução seqüencial dos módulos por paralelismo assíncrono.

Restrição: esse operador só pode ser aplicado quando os módulos *system* (*systemprocess* e *systemactivity*) da especificação original não possuírem transições, porque com a mutação esses módulos ficarão sem atributo e módulos sem atributo não podem conter transições.

As seguintes modificações precisam ser feitas para gerar as especificações mutantes:

- i) trocar o atributo *systemactivity* do módulo *system* para sem atributo; e
- ii) trocar o atributo *activity* dos módulos descendentes do módulo *system* por *systemactivity*.

Exemplo:

Especificação Original	Especificação Mutante
<pre> specification AB systemactivity; ... module Usertype activity (Connendptid : Ceptype); ip U: Uaccesspoint(User) common queue; end; module Alternatingbittype activity (Connendptid : Ceptype); ip U: Uaccesspoint(Provider) common queue; N: Naccesspoint(User) individual queue; end; module Networktype activity; ip N : array[Ceptype] of Naccesspoint(Provider) individual queue; end; ... </pre>	<pre> → specification AB; ... → module Usertype systemactivity (Connendptid : Ceptype); ip U: Uaccesspoint(User) common queue; end; → module Alternatingbittype systemactivity (Connendptid:Ceptype); ip U: Uaccesspoint(Provider) common queue; N: Naccesspoint(User) individual queue; end; → module Networktype systemactivity; ip N : array[Ceptype] of Naccesspoint(Provider) individual queue; end; ... </pre>

3.3. Estratégias de Teste para Aplicação do Teste de Mutação para Estelle

Conforme já apontado anteriormente, um dos problemas relacionados com a aplicação do teste de Mutação é o custo, decorrente do número de mutantes gerados e que precisam ser analisados e executados pelos casos de teste. Outro fator que aumenta o custo desse critério é a determinação dos mutantes equivalentes. Um mutante equivalente é aquele que, para qualquer caso de teste, o comportamento dele e do programa (ou especificação) original é o mesmo. Esse tipo de mutante não contribui para a melhoria do conjunto de casos de teste, pois não contribui para geração de novos casos de teste. Além disso, é uma atividade que requer, em geral, a intervenção do testador, consumindo tempo e recursos para a sua realização.

Uma maneira de tentar reduzir o custo de aplicação do teste de Mutação é utilizar formas alternativas de aplicar esse critério. Essas formas alternativas visam a

reduzir o número de mutantes gerados. Outro aspecto que pode ser explorado também é a definição de uma estratégia incremental para aplicação dos operadores de mutação. Considerando as classes de mutação definidas para a validação de especificações Estelle, uma *Estratégia de Teste Incremental* pode ser estabelecida, composta dos seguintes passos (Souza et al., 2000a):

- i. para cada módulo de Estelle, aplicar os Operadores de Mutação nos Módulos e determinar um conjunto de teste adequado; e
- ii. determinar todas as conexões entre módulos:
 - para cada conexão, aplicar os Operadores de Mutação de Interface e determinar um conjunto de teste adequado.
- iii. aplicar os Operadores de Mutação na Estrutura (operadores número 1 a 11) para validar a estrutura estática e dinâmica de Estelle e selecionar um conjunto de teste adequado;
- iv. aplicar os Operadores de Mutação na Estrutura para Paralelismo (operadores número 12 a 15) e selecionar um conjunto de teste adequado;

Essa estratégia pode ser aplicada de forma *top-down* ou *bottom-up*, de acordo com os aspectos que se desejam explorar ou dar prioridade: comportamento dos módulos, comunicação, paralelismo e estruturação da especificação (estática ou dinâmica). Além disso, pode-se selecionar somente um subconjunto de operadores de mutação em cada passo da estratégia. Caso algum erro seja encontrado durante a aplicação da estratégia, a especificação pode ser corrigida e a estratégia aplicada novamente.

Para seleção de um subconjunto de operadores de mutação é possível utilizar abordagens de mutação alternativa, definidas no nível de programa. Essas abordagens são: **mutação aleatória** (*Randomly Selected Mutation*), **mutação**

restrita (*Constrained Mutation*) e **mutação seletiva** (*Selective Mutation*). Na **mutação aleatória** (Acree et al., 1979) é examinada uma pequena porcentagem de mutantes, selecionados aleatoriamente, de cada operador de mutação, ignorando os demais. Na **mutação restrita** (Mathur, 1991) são selecionados, intuitivamente, alguns operadores de mutação para geração dos mutantes, considerando os erros que se desejam revelar. Na **mutação seletiva** (Offutt et al., 1993) os operadores de mutação responsáveis por gerar um maior número de mutantes não são aplicados. Offutt et al. (1993) observaram que esses operadores de mutação podem ser desconsiderados pois não contribuem para aumentar a eficácia do conjunto de casos de teste.

Essas abordagens poderiam ser utilizadas para aplicação do teste de mutação para Estelle. Por exemplo, poderia ser aplicada a mutação aleatória, gerando 10% de mutantes de cada operador de mutação, combinando ou não com a Estratégia de Teste Incremental. Estudos empíricos realizados no nível de programas indicam que utilizar 10% para geração dos mutantes é satisfatório tanto em relação ao custo quanto em relação à eficácia em revelar erros (Mathur, 1991; Mathur e Wong, 1993; Wong, 1993). Para a mutação restrita poderiam ser selecionados os operadores de mutação mais relevantes de cada classe, de acordo com as características que se desejam validar, e utilizar a Estratégia de Teste Incremental para guiar a sua aplicação. Para a mutação seletiva, seria necessário identificar os N operadores de mutação que geram um número elevado de mutantes para serem descartados. Nesse caso, estudos empíricos precisam ser realizados para que sejam identificados os operadores de mutação que geram um maior número de mutantes. Por exemplo, considerando os resultados da aplicação desse critério no protocolo Bit_Alternante (Capítulo 6), poderia ser definido o critério 5-Seletivo, descartando os 5 operadores

de mutação que geraram um maior número de mutantes: *Troca de operadores da condição, Substituição de variável por variável, Substituição de ação das transições, Substituição de variáveis de interface e Substituição de variáveis de não interface.*

Outro trabalho que pode ser explorado no contexto de teste de mutação para Estelle, é a definição de um conjunto essencial de operadores de mutação (Offutt et al., 1996b; Vincenzi et al., 1999; Barbosa et al., 2000b). Para definição desse conjunto essencial são coletadas estatísticas sobre cada operador de mutação, a partir dos resultados de experimentos empíricos. A utilização de ferramentas que automatizam a aplicação do teste de Mutação favorece a realização desses estudos. Nessa linha de pesquisa, foram definidos conjuntos essenciais de operadores de mutação para a linguagem Fortran (Offutt et al., 1996b) e para a linguagem C, tanto no nível de unidade (Barbosa et al., 2000b), como no nível de integração (Vincenzi et al., 1999)

Considerando que o conjunto de operadores de mutação para Estelle é uma evolução natural dos operadores de mutação definidos para a linguagem C, o conjunto essencial de operadores para Estelle pode ser definido, inicialmente, baseando-se no conjunto essencial de operadores de mutação para a linguagem C. Nas Tabelas 3.3 e 3.4 são apresentados os operadores de mutação essenciais (linguagem C) para o teste de unidade e teste de integração, respectivamente, juntamente com os operadores de mutação para Estelle relacionados a cada operador de mutação.

Essa relação entre os operadores de mutação da linguagem C e da técnica Estelle foi determinada considerando apenas a definição (semântica) dos operadores de mutação e para resultados mais conclusivos é necessário aplicar, no contexto de Estelle, os procedimentos definidos no nível de programas (Offutt et al., 1996b;

Vincenzi et al., 1999; 2000; Barbosa et al., 2000b). Para que isso seja possível, é necessária uma ferramenta que automatize a aplicação do teste de Mutação para Estelle.

Tabela 3.3. Conjunto Essencial de Operadores de Mutação para a Linguagem C (Teste de Unidade) e Operadores de Mutação de Módulos para Estelle Relacionados.

Operadores de Mutação Essenciais para C	Operadores de Mutação para Estelle Relacionados
SWDD - troca comando <i>while</i> por <i>do-while</i>	—
SMTG - pára o comando <i>while</i> após 2 execuções	—
SSDL - remoção de comandos	Cobertura de código
OLBN - troca operador lógico por operadores <i>bitwise</i>	—
ORRN - troca operador relacional por relacional	Troca de operadores da condição
VTWD - incremento/decremento variáveis/constantes	Incremento e decremento variáveis/constantes
VDTR - força valores das variáveis para negativo, positivo e zero	Inclusão de operadores unários nas variáveis
Cccr - troca constante por constante	—
Ccsr - troca variável por constante	Substituição de variável por constante

Tabela 3.4. Conjunto Essencial de Operadores de Mutação de Interface para a Linguagem C (Teste de Integração) e Operadores de Mutação de Interface para Estelle Relacionados.

Operadores de Mutação de Interface Essenciais para C	Operadores de Mutação de Interface para Estelle Relacionados
II-ArgAriNeg acrescenta negação aritmética antes do argumento	Inclusão de operadores unários nos parâmetros
I-CovAllNod garante cobertura de nós	—
I-DirVarBitNeg acrescenta negação de bit em variáveis de interface	—
I-IndVarBitNeg acrescenta negação de bit em variáveis de não interface	—
I-IndVarRepGlo troca variáveis de não interface por variáveis globais utilizadas na função chamada	Substituição de variáveis de não interface
I-IndVarRepExt troca variáveis de não interface por variáveis globais não utilizadas na função chamada	Substituição de variáveis de não interface
I-IndVarRepLoc troca variáveis de não interface por variáveis locais do módulo chamado	Substituição de variáveis de não interface
I-IndVarRepReq troca variáveis de não interface por constantes requeridas	Substituição de variáveis de não interface

Outro aspecto que pode ser explorado nesse contexto é comparar o critério Mutação nos Módulos com o critério Mutação de Interface, na mesma linha do

trabalho realizado por Vincenzi et al. (2000). Vincenzi et al. definem uma estratégia de teste incremental para aplicação do critério Análise de Mutantes e Mutação de Interface, no nível de programas (considerando a linguagem C). Os autores observam que esses critérios são complementares e devem ser utilizados em conjunto para aumentar a qualidade da atividade de teste. Pretende-se investigar futuramente esse aspecto no contexto de especificações Estelle.

3.4. Análise Teórica do Teste de Mutação para Estelle

Nesta seção é feita uma análise teórica da complexidade do teste de Mutação no contexto de Estelle. Para analisar a complexidade de um critério de teste pode-se considerar o número máximo de casos de teste requerido no pior caso. Outra forma de analisar a complexidade é considerar o número de elementos requeridos (requisitos de teste) pelo critério no pior caso, como foi feito por Delamaro (1997) para analisar a complexidade do critério Mutação de Interface. Esses dois aspectos (casos de teste e requisitos de teste) estão fortemente relacionados.

No contexto de programas (nível de unidade), o número de mutantes gerados pelos operadores de mutação é determinado, principalmente, pelo número de variáveis vezes o número de referências às variáveis (Budd, 1981). No nível de interface o número de mutantes gerados é influenciado também pelo número de parâmetros das funções do programa (Delamaro, 1997).

Para determinar a complexidade do teste de mutação para Estelle, é identificado o número de mutantes gerados, no pior caso, para os operadores de mutação definidos, utilizando como base a análise teórica feita por Delamaro (1997) para

avaliar a complexidade do critério Mutação de Interface. Os seguintes conjuntos de elementos são utilizados:

$S(m)$ = conjunto de estados da MEFE do módulo m .

$I(m)$ = conjunto de eventos de entrada da MEFE do módulo m .

$C(m)$ = conjunto de constantes utilizadas no módulo m + constantes requeridas.

$L(m)$ = conjunto de variáveis declaradas no módulo m .

$G(m)$ = conjunto de variáveis declaradas por módulos hierarquicamente superiores ao módulo m .

$PI(m)$ = conjunto de pontos de interação do módulo m .

$P(m)$ = conjunto de parâmetros de cada primitiva de comunicação.

num_op = número máximo de operadores relacionais das condições das transições de m .

$varrefs$ = número máximo de ocorrências de variáveis ou constantes do módulo.

$interrefs$ = número máximo de ocorrência de variáveis de interface (variáveis recebidas pela primitiva de comunicação e variáveis exportadas).

max_elem = número máximo de arcos e arestas do grafo de Fluxo de Controle da ação de uma determinada transição.

Para essa análise, é considerado que a MEFE de um módulo m é determinística (para cada entrada existe uma transição definida em cada estado). Assim, o número máximo de transições é calculado em função de $S(m)$ e de $I(m)$, considerando também uma transição espontânea (sem evento de entrada) para cada estado de $S(m)$.

Tem-se que o número máximo T de transições é:

$$T = |S(m)| * |I(m)| + |S(m)|$$

A seguir é calculado o número máximo de mutantes que podem ser gerados para cada classe de mutação. É feita uma análise de cada operador de mutação e os resultados são sintetizados.

3.4.1. Complexidade dos Operadores de Mutação nos Módulos

A Tabela 3.5 apresenta as equações da complexidade de cada operador de Mutação nos Módulos. Nos próximos parágrafos, a complexidade de cada operador é comentada.

Tabela 3.5. Complexidade dos Operadores de Mutação nos Módulos.

Operadores de Mutação nos Módulos	Complexidade
1. Substituição de estado inicial	$ S(m) - 1$
2. Substituição de estado origem	$ I(m) * (S(m) - 1)$
3. Substituição de estado destino	$ I(m) * (S(m) - 1)$
4. Remoção de estados	$ S(m) * (S(m) - 1)$
5. Remoção de transições	T
6. Remoção de condição das transições	$ S(m) * I(m) $
7. Substituição de evento de entrada	$(I(m) - 1) * I(m) $
8. Remoção de evento de entrada	$ S(m) * I(m) $
9. Negação de condição das transições	$ S(m) * I(m) $
10. Troca de operadores da condição	$(num_op^*5) * S(m) * I(m) $
11. Substituição de atribuição booleana	$varrefs$
12. Substituição de variável por variável	$varrefs * (L(m) + G(m) - 1)$
13. Substituição de variável por constante	$varrefs * C(m) $
14. Incremento/decremento de variáveis/constantes	$varrefs * 2$
15. Inclusão de operadores unários nas variáveis	$Varrefs$
16. Substituição de ação das transições	$T*(T - 1)$
17. Cobertura de código	$T*max_elem$
18. Remoção de prioridades das transições	T
19. Substituição de prioridades das transições	$T*(T-1) + 2*T$
20. Remoção de atrasos das transições	$ S(m) $
21. Substituição de atrasos das transições	$(S(m) - 1) * S(m) + S(m) * (varrefs + 8)$
22. Substituição de política de fila	$ PI(m) $

1. *Substituição de estado inicial*: troca o estado inicial da MEFE pelos elementos do conjunto $S(m)$. Assim, tem-se que o número máximo de mutantes gerados, no pior caso, é igual ao número de estados da MEFE menos 1.

2. *Substituição de estado origem:* o número de mutantes gerados é influenciado pelo número de estados e eventos de entrada da MEFE. Assim, o número de mutantes gerados, no pior caso, é igual ao número de eventos de entrada, multiplicado pelo número de estados menos 1.
3. *Substituição de estado destino:* da mesma forma que o Operador 2, o número de mutantes gerados é influenciado pelo número de estados e eventos de entrada da MEFE, de modo que o número de mutantes gerados, no pior caso, é igual ao número de eventos de entrada, multiplicado pelo número de estados menos 1.
4. *Remoção de estados:* cada estado s_i será retirado e substituído por outro do conjunto $S(m)$ que esteja conectado a s_i . Assim sendo, considerando que todos os estados estejam conectados entre si, o número de mutantes gerados, no pior caso, é igual ao número de estados, multiplicado pelo número de estados menos 1.
5. *Remoção de transições:* o número de mutantes gerados é igual ao número de transições da MEFE
6. *Remoção de condição das transições:* o número de mutantes gerados é igual ao número de transições (não espontâneas) pois, no pior caso, todas as transições possuem condições.
7. *Substituição de evento de entrada:* substitui o evento de entrada de uma transição por todos os eventos de entrada da MEFE, resultando que o número de mutantes gerados, no pior caso, é igual ao número de eventos menos 1, multiplicado pelo número de transições não espontâneas.

8. *Remoção de evento de entrada:* o número de mutantes gerados é igual ao número de transições (não espontâneas), pois, no pior caso, todas as transições possuem eventos de entrada.
9. *Negação de condição das transições:* da mesma forma que o Operador 8, o número de mutantes gerados é igual ao número de transições (não espontâneas), pois, no pior caso, todas as transições possuem eventos de entrada.
10. *Troca de operadores da condição:* é considerado que cada transição (não espontânea) possui um número *num_op* de operadores relacionais, os quais serão substituídos por cada operador relacional possível (total de 6).
11. *Substituição de atribuição booleana:* é considerado que todas as ocorrências de *varrefs* são variáveis booleanas e, por isso, o número de mutantes gerados, no pior caso, é igual ao número de *varrefs*.
12. *Substituição de variável por variável:* considera-se que todas as ocorrências de *varrefs* serão trocadas por todos os elementos do conjunto $L(m)$ e $G(m)$.
13. *Substituição de variável por constante:* considera-se que todas as ocorrências de *varrefs* serão trocadas por todos os elementos do conjunto $C(m)$.
14. *Incremento/decremento de variáveis/constantes:* considera-se que todas as ocorrências de *varrefs* serão incrementadas e decrementadas, o que resulta em um número de mutantes gerados, no pior caso, igual a *varrefs*, multiplicado por 2.
15. *Inclusão de operadores unários nas variáveis:* considera-se que em todas as ocorrências de *varrefs* é possível incluir o operador unário "-", o que resulta em um número de mutantes gerados igual ao número de *varrefs*.

16. *Substituição de ação das transições*: substitui a ação de uma transição pela ação das outras transições da MEFE, resultando que o número de mutantes gerados é igual ao número de transições, multiplicado pelo número de transições menos 1, pois cada transição possui apenas uma ação (bloco *begin-end* da transição).
17. *Cobertura de código*: considera-se que esse operador será aplicado em cada arco e em cada aresta do Grafo de Fluxo de Controle da ação das transições, de modo que o número de mutantes gerados, no pior caso, é igual a *max_elem* multiplicado pelo número de transições.
18. *Remoção de prioridades das transições*: o número de mutantes gerados é igual ao número de transições pois, no pior caso, todas as transições possuem prioridade que podem ser removidas.
19. *Substituição de prioridades das transições*: substitui a prioridade de uma transição pela prioridade de todas as outras transições da MEFE e também pelos elementos 0 e 1 do conjunto de constantes requeridas, resultando que o número de mutantes gerados é igual ao número de transições multiplicado por 2 (elemento 0 e 1) somado com o número de transições, multiplicado pelo número de transições menos 1.
20. *Remoção de atrasos das transições*: é removida a cláusula *delay* de cada transição espontânea, de modo que o número de mutantes gerados é igual ao número de estados, pois, no pior caso, cada estado possui uma transição espontânea.
21. *Substituição de atrasos das transições*: considera-se que todas as transições espontâneas possuem cláusula *delay(t1,t2)*. Além disso, para cada cláusula *delay(t1,t2)*, no pior caso, é possível trocar *t1* e *t2* pelas variáveis e constantes

(*varrefs*) e pelos elementos do conjunto $d = \{t1, t2, 1, 0, *\}$. Por exemplo, para $delay(t1, t2)$ e para os elementos de d , tem-se as seguintes mutações: 1) $delay(t1, t1)$, 2) $delay(t2, t2)$, 3) $delay(1, t2)$, 4) $delay(t1, 1)$, 5) $delay(t1, 0)$, 6) $delay(0, t2)$, 7) $delay(0)$, 8) $delay(t1, *)$. Assim, o número de mutantes gerados para esse operador é igual ao número de estados multiplicado pelo número de estados menos 1 (uma transição espontânea por estado), somado com o número de estados, multiplicado por *varrefs* mais 8 (todas as mutações possíveis com os elementos do conjunto d).

22. *Substituição de política de fila*: a complexidade é igual a $PI(m)$, pois para cada ponto de interação é gerado um mutante.

A partir das equações apresentadas na Tabela 3.5, pode-se observar que a complexidade da classe Mutação nos Módulos está em função dos seguintes elementos da especificação: estados, eventos de entrada, transições, variáveis, operadores relacionais das condições, comandos e desvios das ações das transições e pontos de interação. Pode-se observar que, para alguns operadores de mutação, a complexidade é quadrática para $|I(m)|$ e $|S(m)|$ e para os demais elementos a complexidade é linear.

Entretanto, a complexidade do critério Mutação nos Módulos é um valor menor que o valor obtido com o somatório das equações da Tabela 3.5. Isso ocorre porque as características que precisam ser satisfeitas pela MEFE, de modo a maximizar o número de mutantes gerados para cada operador de mutação, são diferentes. Por exemplo, para o operador *Remoção de transições* considera-se que a MEFE é completamente especificada, ou seja, possui uma transição para todos os eventos de entrada em cada estado. Por outro lado, para maximizar o número de mutantes do

operador *Substituição de estado origem*, a MEFE não pode ser completamente especificada, senão esse não seria o pior caso para esse operador de mutação.

3.4.2. Complexidade dos Operadores de Mutação de Interface

Na Tabela 3.6 tem-se as equações da complexidade dos operadores de Mutação de Interface. O teste de interface considera uma conexão de cada vez para a geração dos mutantes, ou seja, uma conexão é formada pelo comando *output* de um módulo *A* e pelo processamento da mensagem por uma das transições do módulo *B*. Assim, as equações apresentadas na Tabela 3.6 são relativas a uma conexão, que será denotada por *A-B*.

Tabela 3.6. Complexidade dos Operadores de Mutação de Interface.

Operadores de Mutação de Interface	Complexidade
Grupo I – Ponto de Chamada	
1. Substituição de parâmetros	$ P * (L(m) + C(m))$
2. Incremento/decremento de parâmetros	$ P * 2$
3. Troca na ordem dos parâmetros	$2^{ P }$
4. Inclusão de operadores unários	$ P $
5. Substituição de comando <i>output</i>	$T * (T - 1)$
6. Remoção de comando <i>output</i>	T
Grupo II – Módulo Chamado	
7. Substituição de variáveis de interface	$interrefs * (L(m) + C(m) + G(m))$
8. Substituição de variáveis de não interface	$varrefs * (L(m) + C(m) + G(m))$
9. Incremento/decremento de variáveis	$(interrefs + varrefs) * 2$
10. Inclusão de operador unário nas variáveis	$Interrefs + varrefs$
11. Substituição de atribuição booleana	$Interrefs + varrefs$

1. *Substituição de parâmetros*: o número de mutantes gerados, no pior caso, é igual a P (número máximo de parâmetros) multiplicado pelas variáveis que podem substitui-lo, que são as variáveis do conjunto $L(m)$ e $C(m)$.
2. *Incremento/decremento de parâmetros*: para cada ocorrência de um parâmetro são feitos um incremento e um decremento, de modo que o número

de mutantes gerados, no pior caso, é igual ao número de parâmetros multiplicado por 2.

3. *Troca na ordem dos parâmetros*: no pior caso, todos os operadores de mutação podem ser trocados por todos os outros, de modo que o número de mutantes gerados é exponencial ao número de parâmetros.
4. *Inclusão de operadores unários*: considera-se que será inserido o operador “-“ em cada parâmetro, de forma que o número de mutantes gerados, no pior caso, é igual a P .
5. *Substituição do comando output*: considera-se que todas as transições possuem comando *output*, de modo que o número de mutantes gerados, no pior caso, é igual ao número de transições, multiplicado pelo número de transições -1.
6. *Remoção do comando output*: considera-se que todas as transições possuem comando *output*, de modo que o número de mutantes gerados, no pior caso, é igual ao número de transições.
7. *Substituição de variáveis de interface*: considera-se que é possível trocar cada ocorrência de *interrefs* por todas as variáveis dos conjuntos $L(m)$, $C(m)$ e $G(m)$.
8. *Substituição de variáveis de não interface*: semelhante ao operador anterior, considera-se que é possível trocar cada ocorrência de *varrefs* por todas as variáveis dos conjuntos $L(m)$, $C(m)$ e $G(m)$.
9. *Incremento/decremento de variáveis de interface*: o número de mutantes gerados, no pior caso, é igual ao número de *varrefs* e de *interrefs*, multiplicado por dois (um incremento e um decremento para cada variável de interface e de não interface).

10. Inclusão de operador unário nas variáveis: o número de mutantes gerados, no pior caso, é igual ao número de *varrefs* e *interrefs*, pois é assumido que cada ocorrência dessas variáveis pode ser trocada por “-“.

11. Substituição de atribuição booleana: da mesma forma que o operador anterior, o número de mutantes gerados, no pior caso, é igual ao número de *varrefs* e *interrefs*, pois se considera que cada ocorrência dessas variáveis é uma atribuição booleana.

A partir das equações apresentadas na Tabela 3.6, pode-se observar que a complexidade da classe Mutação de Interface está em função dos seguintes elementos da especificação: transições, variáveis e parâmetros das primitivas de comunicação. Pode-se observar que, para alguns operadores de mutação, a complexidade é quadrática para T e exponencial para P. Para os demais elementos a complexidade é linear.

3.4.3. Complexidade dos Operadores de Mutação na Estrutura

Considerando novamente a Figura 3.1, os módulos *A* e *B* possuem uma conexão entre eles, feita pelo canal de comunicação *x*. Assim sendo, *num_conexões* significa o número máximo de conexões do tipo *connect* ou *attach* na especificação. Na Tabela 3.7 é apresentada a equação da complexidade para cada operador de Mutação na Estrutura.

Tabela 3.7 Complexidade dos Operadores de Mutação de Interface.

Operadores de Mutação na Estrutura	Complexidade
Mutação na Estruturação	
1. Remoção de conexões	<i>num conexões</i>
2. Inserção de desconexão	<i>num conexões</i>
3. Remoção de desconexão	<i>num conexões</i>
4. Remoção de comando <i>release</i>	<i>num conexões</i>
5. Remoção de comando <i>terminate</i>	<i>num conexões</i>
6. Substituição de <i>release</i> por <i>terminate</i>	<i>num conexões</i>
7. Substituição de <i>terminate</i> por <i>release</i>	<i>num conexões</i>
8. Substituição de <i>release</i> por <i>detach</i>	<i>num conexões</i>
9. Substituição de <i>release</i> por <i>disconnect</i>	<i>num conexões</i>
10. Substituição de <i>terminate</i> por <i>detach</i>	<i>num conexões</i>
11. Substituição de <i>terminate</i> por <i>disconnect</i>	<i>num conexões</i>
Mutação no Paralelismo	
12. Paralelismo síncrono por execução seqüencial	1
13. Paralelismo síncrono por paralelismo assíncrono	1
14. Execução seqüencial por paralelismo síncrono	1
15. Execução seqüencial por paralelismo assíncrono	1

1. Operadores de Mutação na Estruturação: considera-se que, no pior caso, é possível realizar mutações sobre todas as conexões entre módulos que foram definidas.
2. Operadores de Mutação no Paralelismo: como é modificado o tipo de paralelismo da especificação como um todo e não de módulos isolados, no pior caso, irá ocorrer uma mutação para outro tipo de paralelismo. O atributo de todos os módulos é alterado na mesma mutação para evitar erros sintáticos na especificação.

A partir das equações apresentadas na Tabela 3.7, pode-se observar que a complexidade da classe Mutação na Estrutura está em função do número de conexões entre módulos, ou seja, conexões estruturais ou arquiteturais entre os módulos da especificação. Desse modo, a complexidade é linear em função do número de conexões da especificação.

3.5. Aspectos para Implementação do Teste de Mutação para Estelle

A aplicação do teste de Mutação envolve as seguintes atividades: geração de um conjunto de casos de teste T ; geração dos mutantes; execução dos mutantes com T ; e análise dos mutantes.

No contexto de Estelle, essas atividades ainda não foram automatizadas. Procurou-se, neste trabalho, detalhar alguns aspectos fundamentais para posterior implementação de uma ferramenta de apoio.

As ferramentas de apoio à aplicação do Teste de Mutação já desenvolvidas pelo Grupo de Engenharia de Software do ICMC/USP, fornecem uma base essencial para a definição dessa ferramenta. Essas ferramentas são:

- *Proteum e Proteum/IM (Program Testing Using Mutants)*, desenvolvidas para a validação de programas na linguagem C, para o teste de unidade (Delamaro, 1993) e integração (Delamaro et al., 1999), respectivamente. Atualmente, essas ferramentas encontram-se integradas em um ambiente de teste chamado *Proteum/IM 2.0* (Delamaro et al., 2000);
- *Proteum-RS/FSM*, para a validação de especificações baseadas em Máquinas de Estados Finitos (Fabbri et al., 1994; 1999b);
- *Proteum-RS/ST*, para a validação de especificações baseadas em Statecharts (Fabbri et al., 1999a; Sugeta, 1999); e
- *Proteum-RS/PN*, para a validação de especificações baseadas em Redes de Petri (Simão, 2000; Simão et al., 2000).

As principais funções dessas ferramentas correspondem às atividades básicas do teste de Mutação: definição dos casos de teste, execução da especificação, geração

dos mutantes, execução dos mutantes, análise dos mutantes, cálculo do escore de mutação e geração de relatórios. Uma base de dados mantém informações sobre os casos de teste e sobre os mutantes (situação de cada mutante). Essas ferramentas foram desenvolvidas para permitir ao usuário trabalhar com sessões de teste, iniciando uma sessão de teste e finalizando-a quando desejar, podendo retomá-la a qualquer momento. Para esse fim, os estados intermediários das sessões de teste são armazenados. Recursos para aplicar mutação aleatória e mutação restrita também são fornecidos, sendo que o testador pode selecionar um subconjunto de operadores de mutação ou especificar a porcentagem que deve ser aplicada para a geração dos mutantes.

As atividades relacionadas à aplicação do teste de Mutação são descritas a seguir, considerando a sua aplicação para especificações Estelle.

3.5.1. Obtenção de um Conjunto Inicial de Casos de Teste

Um conjunto de casos de teste (cada caso de teste é formado pela entrada para ser executada e pela saída esperada) é executado com a especificação original. Como acontece no nível de programas, o usuário (ou testador) faz o papel de oráculo, determinando se a saída obtida está correta. Caso a saída não esteja correta um erro foi revelado e a especificação deve ser corrigida. Após a correção do erro, o procedimento se repete até que todo conjunto inicial de casos de teste seja executado com a especificação original. No contexto de Estelle, as entradas de teste correspondem aos eventos definidos para o modelo especificado e as saídas correspondem aos resultados obtidos com a ocorrência dos eventos.

O conjunto inicial de seqüências de teste pode ser gerado manual ou automaticamente. O testador pode gerar as seqüências de teste de forma manual, com

base em algum critério de teste ou de forma intuitiva, a partir das características da especificação que ele deseja observar. De forma automática, pode-se utilizar ferramentas para geração automática de seqüências de teste ou mecanismos para geração aleatória de casos de teste. Para geração automática de seqüências de teste, pode-se considerar a possibilidade de integrar ferramentas já existentes como, por exemplo, aquelas apresentadas na Seção 2.3.2 (Capítulo 2).

Existem duas abordagens que podem ser exploradas para executar a especificação Estelle. A primeira abordagem é utilizar um compilador e transformar a especificação em um programa que representa o funcionamento da especificação. Utilizando essa abordagem, o conjunto de casos de teste inicial é executado com o programa gerado a partir da especificação.

Existem diversos compiladores que permitem a geração de código a partir de especificações em Estelle, tais como os compiladores das ferramentas *EDT*¹ – linguagem C, *PetDingo* – linguagem C++ (Sijelmasi e Strausser, 1991a), o ambiente de transformação de software *Draco*, que transforma especificações Estelle em implementações C++ (Santana et al., 1997), *XEC* – linguagem C++ (Thees e Gotzhein, 1998) e *UBC*² – linguagem C.

Outra abordagem que pode ser utilizada é executar a própria especificação Estelle, utilizando para isso um simulador. Com o simulador, os casos de teste podem ser construídos interativamente, ou seja, executando as transições de forma que os eventos desejados sejam disparados, ou através de programas de controle da simulação, os quais possibilitam que a simulação seja realizada sem o controle do usuário.

¹ <http://alix.int-evry.fr/~stan/edt.html>

² <http://www.estelle.org/>

O simulador *Edb* da ferramenta *EDT* permite que a simulação possa ser realizada de maneira interativa ou controlada. Na simulação interativa são fornecidas as transições aptas a disparar, sendo que o usuário pode selecionar também um subsistema ou um módulo a partir do qual as transições serão disparadas.

A execução da especificação utilizando um simulador parece uma abordagem mais promissora. A geração de código a partir da especificação nem sempre é completamente automatizada e erros podem ser inseridos nesse processo. Alguns estudos realizados no escopo desta tese utilizando-se a ferramenta *EDT* indicam que o simulador dessa ferramenta poderia ser empregado para execução da especificação original e das especificações mutantes e também para auxiliar na obtenção de seqüências de teste, que posteriormente poderiam ser aplicadas em uma particular implementação.

3.5.2. Geração dos Mutantes

A geração dos mutantes consiste em aplicar sistematicamente os operadores de mutação na especificação origem. Para auxiliar na aplicação dos operadores de mutação, alguns conjuntos de elementos são definidos. Esses conjuntos são construídos a partir dos elementos da especificação, como por exemplo, estados, variáveis, operadores, os quais podem sofrer mutações pela aplicação dos operadores de mutação, de modo a respeitar as características sintáticas da linguagem Estelle.

Na Mutação nos Módulos os conjuntos de elementos são obtidos da seguinte forma:

Para cada transição:

- Obter todos os elementos sintáticos que podem ser modificados e dividi-los em quatro categorias:

1. variáveis de estado (estados das cláusulas *from* e *to*): selecionar todas as variáveis de estado definidas pelo módulo que contém a transição.
2. eventos de entrada (associados à cláusula *when*): selecionar a partir das transições do módulo os eventos de entrada.
3. variáveis de contexto (utilizadas pela cláusula *provided* e no bloco *begin-end*): para cada variável de contexto, identificar seu tipo e selecionar todas as outras variáveis do mesmo tipo, declaradas pelo módulo em teste e pelos módulos hierarquicamente superiores.
4. operadores relacionais e lógicos relacionados às variáveis de contexto: selecionar todos os outros operadores do mesmo tipo.

Na Mutação de Interface os conjuntos de elementos são obtidos da seguinte forma:

- Para cada ponto de chamada (comando *output*), obter para cada conexão:
 - obter todos os parâmetros da cláusula *output*, e:
 - *Para cada parâmetro*: identificar seu tipo e selecionar todas as outras variáveis do mesmo tipo declaradas pelo módulo em questão e pelos módulos superiores.
 - *Para a primitiva de comunicação* do comando *output*: selecionar todas as outras primitivas definidas para o ponto de interação em questão.
 - *Para as variáveis exportadas*: identificar seu tipo e selecionar todas as outras variáveis do mesmo tipo declaradas pelo módulo em questão e pelos módulos superiores.
- Para cada módulo chamado, obter para cada conexão (transição):
 - obter todas as variáveis de interface: selecionar os parâmetros das primitivas de comunicação a partir da declaração de canais, para os papéis dos pontos de interação do módulo testado. Selecionar as variáveis exportadas a partir das variáveis declaradas em *export* no cabeçalho do módulo.

- Para cada variável de interface:
 - identificar seu tipo e selecionar todas as outras variáveis do mesmo tipo declaradas pelo módulo e pelos módulos superiores;
 - Obter as variáveis de não interface que aparecem em expressões que envolvem variáveis de interface:
 - identificar seu tipo e selecionar todas as outras variáveis do mesmo tipo declaradas pelo módulo e pelos módulos ancestrais.

Na Mutação na Estrutura não são definidos conjuntos de elementos pois as mutações são obtidas basicamente pela troca de comandos da linguagem Estelle.

Para geração dos mutantes, podem ser utilizados os mesmos mecanismos definidos e utilizados por Delamaro et al. (2000). Inicialmente, a partir da especificação Estelle podem-se gerar “descritores de mutação”, os quais possuem as informações necessárias para a geração dos mutantes, tais como: onde a mutação será feita, quantos caracteres serão substituídos e quais caracteres que irão substituí-los. A utilização do descritor de mutação auxilia na execução dos mutantes visto que a criação do mutante é fácil e rápida, pois requer somente a substituição de caracteres do texto, sem necessitar de informações sobre a estrutura e a sintaxe da linguagem. Desse modo, esse mecanismo de geração de mutantes pode ser explorado no contexto de especificações Estelle.

Outro aspecto que pode ser considerado durante a geração dos mutantes é a aplicação de estratégias de mutação alternativa: mutação restrita, seletiva e aleatória. A versão atual da família de ferramentas *Proteum* apresenta facilidades para aplicação dessas estratégias, permitindo que sejam selecionados alguns operadores para geração dos mutantes, ou permitindo que seja estipulada uma porcentagem de mutantes para serem gerados. Esses recursos devem ser disponibilizados também em

uma ferramenta para Estelle, tornando viável também a aplicação das estratégias de teste incrementais definidas para Estelle na Seção 3.3.

3.5.3. Execução dos Mutantes

Depois de gerados os mutantes, o próximo passo é executá-los com o conjunto de casos de teste inicial. As especificações mutantes podem ser executadas da mesma forma que a especificação original, utilizando ou geração de código a partir das especificações, ou utilizando um simulador para executá-las.

A execução dos mutantes é uma atividade “gargalo” do Teste de Mutação, dado o número elevado de mutantes que podem ser gerados. Para reduzir o tempo de execução dos mutantes, duas abordagens são utilizadas na ferramenta *Proteum/IM2.0* (Delamaro et al., 2000) e podem ser investigadas no contexto de Estelle. Na primeira abordagem um *meta-mutante* é utilizado, o qual consiste de um arquivo contendo vários mutantes, de modo que esses mutantes são compilados juntos. Isso é feito utilizando variáveis de controle de modo que todos os mutantes do meta-mutante sejam executados pelos casos de teste (Delamaro et al., 2000). Com isso é possível gerar um número menor de programas mutantes executáveis.

Outra abordagem utiliza informações sobre o fluxo de controle do programa para diminuir o tempo de execução dos mutantes. Quando um caso de teste t é incluído no conjunto de casos de teste, são armazenadas informações sobre quais nós do Grafo de Programa que são executados por t . Antes de tentar matar um mutante M com t , é checado se t pode realmente matar o mutante, ou seja, é verificado no descritor do mutante M se o nó (ou os nós) onde foi feita a mutação é executado por t (Delamaro et al., 2000). Caso não seja, M não é executado com t pois, certamente, M permaneceria vivo, de acordo com a propriedade de alcançabilidade, descrita na

Seção 3.2. Essas idéias, com as devidas adaptações podem ser exploradas no contexto de Estelle.

3.5.4. Análise dos Mutantes e Geração de Conjuntos de Casos de Teste Adequados

Este passo consiste em analisar os mutantes gerados para determinar se esses são equivalentes à especificação original ou se existe alguma seqüência de teste que distingue esses mutantes. Para os mutantes não equivalentes, é necessário adicionar seqüências de teste que evidenciam o comportamento incorreto do mutante. O objetivo é obter um conjunto de seqüências de teste que garanta que os erros modelados pelos operadores de mutação não estão presentes na especificação em teste.

Um problema nessa fase é a determinação dos mutantes equivalentes. Mutantes equivalentes não contribuem para melhoria do conjunto de casos de teste mas requerem tempo e atenção do testador para sua determinação. Em geral, eles têm sido detectados manualmente, aumentando o custo dessa atividade e restringindo a utilização prática do Teste de Mutação. Existem trabalhos, como por exemplo o trabalho de Offutt e Pan (1996) descrito no Capítulo 2, que procuram fornecer mecanismos para automatizar, mesmo que parcialmente, essa atividade. Para a versão atual da família de ferramentas *Proteum* a determinação dos mutantes equivalentes é feita manualmente.

As idéias apresentadas nesta seção têm por objetivo fornecer subsídios para o desenvolvimento de uma ferramenta para validação de especificações Estelle.

3.6. Considerações Finais

Neste capítulo foi apresentada a definição do teste de Mutação para a validação de especificações baseadas em Estelle. A principal contribuição deste capítulo foi a proposição de um conjunto de operadores de mutação para Estelle, o qual é um aspecto importante para o sucesso da aplicação desse critério de teste.

Trabalhos anteriores, tais como os trabalhos de Probert e Guo (1991), Fabbri et al. (1994;1995; 1999a) e de Delamaro (1997), forneceram as idéias básicas e a motivação para a definição dos operadores de mutação para especificações Estelle. O conjunto de operadores de mutação considera as características intrínsecas de Estelle, como por exemplo, a comunicação, o paralelismo e as estruturas dinâmicas. A divisão desses operadores de mutação em classes de aplicação: Mutação nos Módulos, Mutação de Interface e Mutação na Estrutura, possibilitou a definição de uma estratégia incremental para aplicação do teste de Mutação em Estelle (Souza et al., 2000a). Essa estratégia permite a condução da atividade de validação dando-se prioridade para erros específicos no modelo, além de possibilitar a aplicação sistemática do critério, de acordo com as condições disponíveis para realização dessa atividade (por exemplo, de acordo com o tempo e recursos disponíveis para sua condução).

A análise teórica da complexidade do teste de Mutação para Estelle mostrou que, em geral, o número de mutantes gerados é influenciado pelo número de variáveis, número de parâmetros enviados pelas mensagens, número de elementos da MEFE de cada módulo e pelo número de conexões entre os módulos da especificação. Esses resultados são semelhantes aos resultados apresentados por Budd (1981) e Delamaro (1997) para o teste de Mutação no nível de programas. Esses resultados indicam que o número de mutantes gerados pode ser elevado, o que ressalta a necessidade de

definir estratégias alternativas para sua aplicação, restringindo o número de mutantes utilizados, conforme é feito no nível de programas por Wong et al. (1994b), Offutt et al. (1996b) e Barbosa et al. (2000b).

A aplicação do teste de Mutação sem o suporte de uma ferramenta é impraticável. Nesse sentido, foram apresentadas ponderações sobre os aspectos necessários para a implementação de uma ferramenta que automatize a aplicação do teste de Mutação para Estelle. As ferramentas já existentes para esse critério de teste: *Proteum* (Delamaro, 1993), *Proteum/IM* (Delamaro, 1997), *Proteum/RS-FSM* (Fabbri et al., 1994; 1999b), *Proteum/RS-ST* (Sugeta, 1999; Fabbri et al., 1999a) e *Proteum/RS-PN* (Simão, 2000; Simão et al., 2000), fornecem uma base essencial para o desenvolvimento de uma ferramenta de validação para Estelle.

Como trabalhos futuros nesta linha de pesquisa, pretende-se refinar os operadores de mutação para Estelle, fazendo um estudo detalhado da capacidade de revelar erros de cada um. O desenvolvimento de uma ferramenta de apoio para aplicação do teste de Mutação é outro objetivo a ser realizado, como também a condução de experimentos para avaliar a capacidade em revelar erros desse critério de teste no contexto de Estelle. Para isso, pretende-se utilizar especificações Estelle reais e mais complexas.

No próximo capítulo é apresentada a definição de critérios de teste baseados em Fluxo de Controle para a validação de especificações descritas em Estelle e em Statecharts. Os critérios de teste para Estelle, definidos nesta tese, são aplicados na especificação do Protocolo *Bit_Alternante* e os resultados obtidos são apresentados no Capítulo 6.

Capítulo 4. Critérios de Cobertura para Especificações Baseadas em Statecharts e em Estelle

4.1. Considerações Iniciais

A Família de Critérios de Cobertura para Statecharts – FCCS e a Família de Critérios de Cobertura para Estelle – FCCE, apresentadas neste capítulo, auxiliam no processo de validação de especificações em Statecharts e em Estelle, fornecendo mecanismos para quantificar a atividade de teste. Através desses critérios, é possível analisar a cobertura da especificação pelas seqüências de teste e também guiar a geração de seqüências de teste, adequadas por construção, a esses critérios. Desse modo, esses critérios podem complementar as atividades normalmente empregadas para validação de especificações baseadas em Statecharts e em Estelle, como por exemplo, a atividade de simulação. Além disso, os conjuntos de casos de teste adequados a esses critérios podem ser utilizados também durante a realização de teste de conformidade.

A definição desses critérios de teste baseou-se em critérios definidos para programas paralelos e concorrentes, tais como, os trabalhos de Taylor et al. (1992), Yang e Chung (1992), Chung et al. (1996), Koppol e Tai (1996) e Yang et al. (1998). Isso foi possível dado que as técnicas Statecharts e Estelle podem modelar aspectos

de paralelismo e concorrência da especificação, tais como, comunicação, sincronização e não determinismo.

Conforme discutido no Capítulo 2, alguns pesquisadores exploram a definição de critérios de teste para validar especificações, fazendo um mapeamento de critérios definidos no nível de programas para o contexto de especificações e de programas concorrentes. Fabbri et al. (1994, 1995, 1999a) e Probert e Guo (1991) exploram o uso do Teste de Mutação, enquanto que Ural e Yang (1991) exploram o uso dos critérios baseados em Fluxo de Dados. Na mesma linha, as famílias de critérios FCCS e FCCE apresentam critérios de teste que exploram os aspectos de Fluxo de Controle (Beizer, 1990) da especificação que podem ser modelados a partir da árvore de alcançabilidade.

Os critérios de cobertura podem ser utilizados para guiar a seleção de seqüências de teste como também para avaliar a cobertura de seqüências de teste geradas por outros mecanismos, como por exemplo, a atividade de simulação. Dessa forma, esses critérios de cobertura podem complementar e quantificar outras formas de validação.

Em relação à técnica Statecharts, inicialmente, foram definidos os critérios FCCS, compostos de critérios que focalizam o fluxo de controle da especificação e critérios que focalizam as características específicas da técnica Statecharts, como por exemplo, história, paralelismo e broadcasting. A árvore de alcançabilidade para Statecharts, definida por Masiero et al. (1994), é utilizada para apoiar a aplicação dos critérios FCCS. Esses critérios foram analisados teoricamente, estabelecendo-se uma relação hierárquica entre eles e propondo-se uma estratégia incremental para aplicação dos critérios.

Com base nos resultados obtidos com a técnica Statecharts, explorou-se a possibilidade de definir esses critérios no contexto de especificações em Estelle,

surgindo, dessa forma, a FCCE, formada pelos critérios de Fluxo de Controle estabelecidos para Statecharts. Os critérios de cobertura no contexto de Estelle permitem validar os aspectos de comunicação entre os módulos, aspectos dinâmicos e as possíveis intercalações entre as transições disparadas, de acordo com o tipo de sincronismo do sistema. A validação desses aspectos é viabilizada a partir da definição de um modelo de árvore de alcançabilidade para Estelle. Esse modelo foi definido com base na semântica dessa linguagem e considerando algumas técnicas de redução descritas no Capítulo 2, tais como, *nós duplicados*, *conjunto de componentes* e *stubborn sets*.

4.2. FCCS e FCCE: Critérios de Cobertura para Statecharts e Estelle

Os critérios FCCS e FCCE, apresentados nesta seção, fornecem mecanismos para analisar a cobertura da especificação e também para guiar a geração de seqüências de teste, adequadas por construção, a esses critérios, ou seja, podem ser utilizados como critérios de adequação e como métodos de seleção de seqüências de teste.

A seguir são apresentados alguns conceitos que são essenciais para o entendimento dos critérios de cobertura definidos:

- **Configuração:** uma configuração C_i é um conjunto de estados que estão ativos em um passo da computação, sendo que C_0 é a configuração inicial. Em cada passo é suposto que os eventos associados à configuração atual são válidos e disparam as transições relacionadas, de modo que é possível modelar o espaço de configurações possíveis do sistema.

- **Caminho:** é uma seqüência finita de configurações ($C_0, C_1, C_2, \dots, C_m, C_k$), $k \geq 1$, tal que a primeira configuração é a configuração inicial da árvore (C_0), e existe uma transição de C_i para C_j , $\forall C_i, C_j \mid 0 \leq i < k$ e $j = i + 1$.
- **Caminho Simples:** é um caminho P tal que todas as configurações que compõem esse caminho, exceto possivelmente a primeira e a última, são distintas.
- **Caminho Livre de Laço:** é um caminho simples P tal que todas as configurações são distintas, inclusive a primeira e a última.
- **Caminho Reiniciável:** é um caminho P em que a primeira e a última configuração do caminho correspondem à configuração inicial C_0 , ou seja, são os caminhos que fazem com que o modelo retorne ao seu estado inicial.

Os critérios de cobertura que fazem parte da FCCS e da FCCE são descritos a seguir. Esses critérios são exemplificados na Seção 4.4, utilizando-se uma especificação descrita em Statecharts.

Em se tratando de sistemas baseados em transições, a cobertura mínima desejável é executar todas as configurações de estados e todas as transições. Assim, dois critérios de cobertura são definidos:

1. Critério **Todas-Configurações**: requer que todas as configurações do modelo sejam percorridas no mínimo uma vez pelo conjunto de seqüências de teste.
2. Critério **Todas-Transições**: requer que todas as transições sejam executadas no mínimo uma vez pelo conjunto de seqüências de teste.

Chow (1978) mostra que esses critérios não são apropriados para revelar erros típicos de especificações baseadas em Máquinas de Estados Finitos, tais como, *erros de transferência, erros de operação e erros de estados extras ou ausentes*. Isso pode

ser considerado também para especificações baseadas em Statecharts e em Estelle, visto que seus componentes básicos são máquinas de estados finitos estendidas.

3. Critério ***Todos-Caminhos***: requer que todos os caminhos sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste.

Do ponto de vista do teste estrutural, o critério *Todos-Caminhos* corresponde ao teste exaustivo, tornando-se impraticável, dada a possibilidade de existirem infinitos caminhos. Os critérios a seguir são mais rigorosos que os critérios *Todas-Transições* e *Todas-Configurações*, entretanto, menos dispendiosos que o critério *Todos-Caminhos*. Esses critérios estabelecem algum tipo de restrição para guiar a seleção de caminhos e também estabelecem uma "ponte" entre os critérios *Todos-Caminhos*, *Todas-Configurações* e *Todas-Transições*:

4. Critério ***Todos-Caminhos-k-Configuração***: requer que todos os caminhos contendo k repetições da configuração C_0 sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste. Para $k = 2$ cada caminho reinicia a especificação uma vez e para $k > 2$, cada caminho reinicia a especificação $k-1$ vezes.

É importante observar que esse critério só é aplicado efetivamente para especificações que são reiniciáveis. Uma especificação é reiniciável quando, para cada configuração C_i alcançada a partir de C_0 , existe uma seqüência de eventos que retorna a C_0 . Entretanto, não é possível alcançar C_0 a partir de C_i sem conhecer previamente o caminho a ser percorrido. Nesse sentido, o algoritmo desenvolvido por Boaventura (1992) para analisar se um statechart é reiniciável pode ser aplicado para verificar a aplicabilidade desse critério de teste, podendo ser estendido também para ser aplicado no contexto de especificações em Estelle.

5. Critério ***Todos-Caminhos-k-Configurações***: requer que todos os caminhos contendo no máximo k repetições de cada configuração sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste.
6. Critério ***Todos-Caminhos-com-um-Laço***: requer que todos os caminhos contendo no máximo 2 repetições de uma (somente uma) configuração C_i sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste.
7. Critério ***Todos-Caminhos-Simples***: requer que todos os caminhos simples sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste.
8. Critério ***Todos-Caminhos-livre-Laço***: requer que todos os caminhos livres de laços sejam percorridos no mínimo uma vez pelo conjunto de seqüências de teste.

A técnica Statecharts possui três características que precisam ser consideradas: paralelismo, *broadcasting* e história. O paralelismo é expresso pelas configurações contendo mais de um estado ativo, as quais estão incluídas nos requisitos que satisfazem o critério *Todas-Configurações*. Para *broadcasting* e história, os seguintes critérios são definidos, os quais são específicos para Statecharts:

9. Critério ***Todas-Reações-em-Cadeia***: requer que todas as transições com ações possuindo eventos sejam exercitadas pelo menos uma vez pelo conjunto de seqüências de teste. A partir de uma configuração, ocorre *broadcasting* ou reação em cadeia quando é disparada uma transição do tipo $e[c]/a$, com a sendo um evento. Dessa forma, todas as transições desse tipo devem ser percorridas no mínimo uma vez pelo conjunto de seqüências de teste para satisfazer este critério de teste.
10. Critério ***Todas-Configurações-História***: requer que para cada transição com símbolo história, todas as configurações possíveis de serem alcançadas a

partir da transição sejam percorridas pelo menos uma vez pelo conjunto de seqüências de teste.

Com relação à técnica Estelle, características específicas dessa técnica podem ser consideradas pelos critérios de cobertura. Por exemplo, pode-se testar a comunicação entre os módulos através do recebimento de mensagens nos pontos de interação, informação que pode ser obtida pelas configurações da árvore. Ou seja, as seqüências de teste adequadas ao critério *Todas-Configurações* executam, no mínimo uma vez, todo recebimento possível de interações pelos pontos de interação.

Outra característica importante de Estelle que pode ser considerada pelos critérios de cobertura é o paralelismo entre os módulos da especificação. Conforme descrito anteriormente, Estelle permite três tipos de execuções dos módulos (componentes): execução seqüencial, execução paralela síncrona e execução paralela assíncrona. Considerando mais de um componente para construção da árvore de alcançabilidade, as configurações representam as possíveis intercalações entre os estados de cada componente e, desse modo, o critério *Todas-Configurações* também executa todo paralelismo possível entre os módulos considerados. Dado que serão selecionados alguns componentes da especificação para análise e construção da árvore de alcançabilidade (reduzir a explosão de estados), a aplicação dos critérios de cobertura é restrita aos componentes selecionados, ou seja, a validação do paralelismo limita-se aos componentes selecionados.

Especificações Estelle contendo componentes dinâmicos (contendo módulos que podem ser criados e destruídos em tempo de execução pelos módulos hierarquicamente superiores) também podem ser validadas utilizando os critérios de cobertura. Neste caso, cada configuração da árvore conteria os estados (estado local + conteúdo das filas) dos módulos selecionados e dos módulos criados

dinamicamente. Para evitar a explosão de estados pode-se limitar o número de instâncias do mesmo módulo durante a construção da árvore de alcançabilidade. Da mesma forma que ocorre com a validação do paralelismo, a validação do aspecto dinâmico é restrito para os componentes selecionados para construção da árvore de alcançabilidade.

Em geral, os critérios de cobertura estabelecem os requisitos de teste mínimos que precisam ser executados pelo conjunto de seqüências de teste T , de forma que T seja adequado a esses critérios de cobertura. Um conjunto de seqüências de teste T é adequado em relação a um critério C_R (descrito como C_R -adequado) se T satisfaz ou executa todos os requisitos de teste impostos por C_R (Rapps; Weyuker, 1985).

Durante o estabelecimento dos requisitos de teste é possível derivar um conjunto de seqüências de teste adequado, por construção, aos critérios de cobertura. Um conjunto de seqüências de teste T é adequado por construção a um critério C_R quando os requisitos de teste estabelecidos por C_R guiam a geração de T , ou seja, T é construído de forma a cobrir cada requisito de C_R .

4.3. Caracterização dos Requisitos e Seqüências de Teste dos Critérios de Cobertura

Nesta seção são descritos como os requisitos de teste dos critérios de cobertura podem ser facilmente obtidos a partir da representação da especificação por meio da árvore de alcançabilidade. Conforme descrito no Capítulo 2, a árvore de alcançabilidade permite representar o comportamento dinâmico da especificação (ou do sistema), descrevendo todos os estados que podem ser alcançados a partir de um estado inicial. Entretanto, a explosão de estados é um fator que inviabiliza a sua

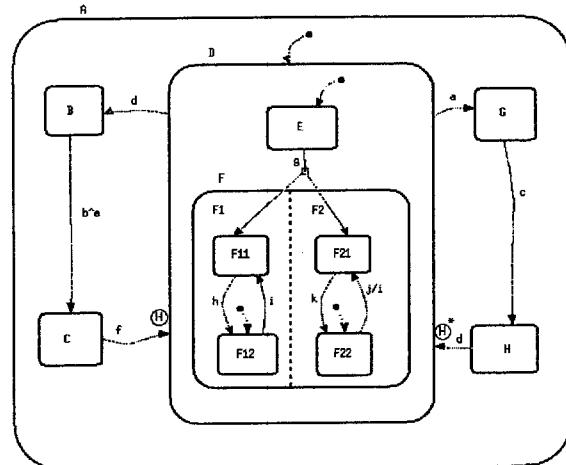
utilização. No Capítulo 2 foram descritos alguns trabalhos apresentando propostas de redução da árvore de alcançabilidade, durante a sua construção. Conforme descrito a seguir, algumas dessas propostas são consideradas na construção da árvore para especificações em Statecharts e Estelle.

4.3.1. Árvore de Alcançabilidade para Statecharts

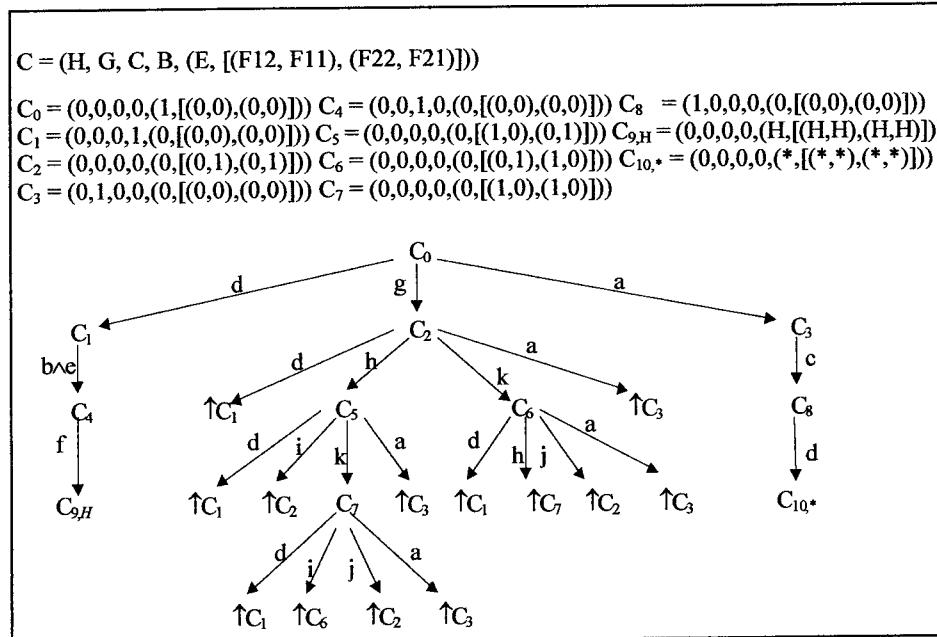
A árvore de alcançabilidade para Statecharts foi definida por Maseiro et al. (1994), tendo a seguinte notação: cada nó representa uma possível configuração de estados do modelo, sendo que podem existir 4 tipos de configurações: *configuração nova* – aquela que ainda não existe na árvore; *configuração velha* – aquela já existente à esquerda ou nos níveis superiores da árvore (essa configuração recebe o símbolo \uparrow , significando que a mesma representa um *link* para a sua primeira ocorrência); *configuração terminal* – quando não existe transição que possa ser habilitada a partir dela; e *configuração história* – representa todas as configurações que podem ser obtidas a partir de uma transição que possui o símbolo *história* (H ou H^* , que na árvore de alcançabilidade são denotados por H e *, respectivamente). Quando a árvore é percorrida, o símbolo história é trocado pela última configuração de estados que pertence ao escopo do símbolo história.

As seguintes técnicas de redução são consideradas durante a construção da árvore de alcançabilidade para Statecharts: *nós duplicados* – quando uma configuração já existente na árvore é inserida, ela é considerada apenas um *link* para a primeira ocorrência dessa configuração, não sendo geradas suas configurações sucessoras, e *stubborn sets* – trata transições independentes que são disparadas no mesmo passo. Essas transições podem ser disparadas em qualquer ordem antes de ser

obtida a próxima configuração, de modo que é possível considerar somente uma das possibilidades de ordem entre elas, chamada de *stubborn sets*. Na Figura 4.1 é ilustrada a árvore de alcançabilidade para Statecharts, construída a partir do statechart apresentado no Capítulo 2.



a) Statechart.



b) Árvore de Alcançabilidade.

Figura 4.1. Árvore de Alcançabilidade do Statechart Exemplo (Masiero et al., 1994).

Para facilitar a visualização, o statechart é apresentado novamente na Figura 4.1a. A configuração inicial (*default*) é o nó raiz da árvore, sendo considerada uma configuração nova. Todas as transições que fazem com que o statechart mude dessa configuração para outra são representadas, juntamente com as configurações obtidas com o disparo de cada transição. Essas configurações são classificadas e inseridas na árvore (Figura 4.1b). Esse procedimento é repetido até que todas as possíveis configurações sejam representadas. A notação utilizada para as configurações da árvore é que os estados que estão ativos na configuração recebem o número 1 e os estados que estão inativos recebem o número 0. Além disso, estados *OR* são colocados entre parênteses e estados *AND* são colocados entre colchetes. Para as configurações história (Figura 4.1) $C_{9,H} = (0,0,0,0,(H,[(H,H),(H,H)]))$ e $C_{10,*} = (0,0,0,0,(*,[(*,*),(*,*)]))$, os símbolos *H* e *** representam quais estados pertencem ao escopo da transição história.

Suponha que se deseja saber se a seqüência de eventos: *g, h, a, c, d, i* é válida ou não para o exemplo. A partir da árvore de alcançabilidade pode-se concluir que a seqüência é válida, sendo que as seguintes configurações são alcançadas com o disparo da seqüência de eventos: $C_0 \rightarrow C_2 \rightarrow C_5 \rightarrow \uparrow C_3 \rightarrow C_8 \rightarrow C_{10,*} \rightarrow C_5 \rightarrow C_2$. Esse exemplo ilustra dois aspectos. Primeiro, quando uma configuração velha é alcançada ($\uparrow C_3$) procura-se à esquerda ou nos níveis superiores da árvore a primeira ocorrência dessa configuração (sem o símbolo \uparrow) para prosseguir com a obtenção das próximas configurações. Segundo, quando uma configuração história é atingida, no exemplo $C_{10,*} = (0,0,0,0,(*,[(*,*),(*,*)]))$, a lista de configurações já atravessadas é percorrida de trás para frente, buscando a primeira ocorrência de uma configuração com estados ativos dentro do escopo da história. No exemplo, a lista começa a ser percorrida a

partir da configuração $C_8 = (1,0,0,0(0,[((0,0),(0,0)])))$, sendo que essa configuração não possui estados ativos dentro do escopo do símbolo história. Passa-se então para a próxima que é $C_3 = (0,1,0,0(0,[((0,0),(0,0)])))$, que também não possui estados ativos no escopo do símbolo história. A próxima configuração é $C_5 = (0,0,0,0(0,[((1,0),(0,1)])))$, que possui estados ativos dentro do escopo de história, a qual substitui a configuração $C_{10,*}$. Então C_5 , passa a ser a configuração atual. Finalmente, a partir de C_5 , é disparado o evento i , atingindo a configuração $C_2 = (0,0,0,0(0,[((0,1),(0,1)])))$.

Durante a construção da árvore de alcançabilidade para Statecharts não são consideradas as condições associadas às transições. Dessa forma, não é representado o broadcasting, que ocorre quando uma transição possui como ação um evento. Além disso, as variáveis utilizadas na especificação não são representadas na árvore de alcançabilidade, de forma que o fluxo de dados do modelo não pode ser extraído.

Os algoritmos para construção da árvore de alcançabilidade e análise de propriedades a partir dela, tais como: alcançabilidade de configurações, validade de uma seqüência de eventos, reiniciabilidade e existência de deadlock, encontram-se implementados e fazem parte do ambiente StatSim (Masiero et al., 1991). O ambiente StatSim é utilizado para edição e simulação de Statecharts e seu uso, juntamente com a aplicação dos critérios de cobertura para Statecharts, é exemplificado no Capítulo 5.

4.3.2. Árvore de Alcançabilidade para Estelle

Para a técnica Estelle, o documento ISO 9074 (1987) e Budkowski e Dembinski (1987) definem o comportamento do modelo especificado em Estelle caracterizando

estado global (ou situação global) e estados locais (ou situações locais) do sistema. Na verdade, um estado global é análogo a uma configuração na notação para Statecharts, sendo formado pelos *estados locais* dos módulos do sistema, estrutura hierárquica dos módulos, ligações entre os módulos, variáveis, conteúdo dos canais de comunicação, e pelas transições selecionadas para execução. Um estado local de um módulo P possui a notação (Sp, tp) , sendo que Sp é um dos estados de P e tp é uma transição oferecida por P . A partir de um estado Sp , mais de uma transição pode estar apta a disparar mas apenas uma é escolhida aleatoriamente (não determinismo), ou seja, pode existir mais de um estado local para Sp .

Jéron e Jard (1993) definem um sistema de transições, chamado *Sistema de Transições com Canais FIFO – TSFC*, para descrever o comportamento de sistemas especificados em Máquinas de Estados Finitos com Comunicação, em Redes de Filas, em Redes de Petri Estendidas e em Estelle, os quais têm em comum a utilização de canais de comunicação com filas. O sistema de transições *TSFC* é bastante similar às definições semânticas para Estelle do documento ISO 9074 (1987) e de Budkowski e Dembinski (1987). A diferença é que Jéron e Jard caracterizam o estado local Sp , como $Sp = \langle E(Sp), C(Sp) \rangle$, sendo que $E(Sp)$ representa o estado local da MEFE de Sp , e $C(Sp)$ representa o conteúdo das filas dos canais de comunicação de Sp . Para tratar a explosão de estados, os autores propõem uma técnica de redução que concentra-se nas filas dos canais de comunicação. Durante a construção da árvore são identificadas seqüências de transições que podem ser infinitamente repetidas e que aumentam o tamanho das filas dos canais de comunicação. Essas seqüências são identificadas e a árvore de alcançabilidade não é mais explorada a partir dessas seqüências de transições. Essa técnica de redução nem sempre consegueoccasionar uma redução no tamanho da árvore. Os autores

apresentam algumas situações em que a técnica não detecta que a árvore é infinita e com isso, não sendo possível reduzir o tamanho da árvore.

Nesta tese é proposta a construção da árvore de alcançabilidade para Estelle considerando a técnica de redução *conjunto de componentes*, que seleciona previamente alguns componentes do sistema modelado para a construção da árvore de alcançabilidade. Foi observado que a aplicação dessa técnica de redução, comparada com a técnica de Jéron e Jard (1993), leva a uma árvore de alcançabilidade de tamanho menor.

A obtenção da árvore de alcançabilidade selecionando apenas alguns componentes do sistema é muito útil em se tratando de especificações de protocolos de comunicação compostos de várias camadas. Em geral, as camadas do protocolo são especificadas (e implementadas) separadamente, descrevendo-se as interações entre elas.

Para a construção da árvore de alcançabilidade para Estelle, cada configuração C_i da árvore representa um possível estado do(s) componente(s) selecionado(s), possuindo as seguintes informações: estado $E(C_i)$ da(s) MEFE(s) do(s) componente(s) e conteúdo das filas $C(C_i)$ dos pontos de interação do(s) componente(s). O conteúdo de $C(C_i)$ é expresso pelas primitivas de comunicação recebidas. Uma configuração é equivalente a um estado local no sistema de transições de Jéron e Jard (1993). No restante deste capítulo será utilizado o termo “configuração” para designar os estados locais do modelo. Isso é feito para manter o mesmo padrão já utilizado para a árvore de alcançabilidade para Statecharts.

Para ilustrar a construção da árvore de alcançabilidade para Estelle é utilizado a especificação do Protocolo *Bit-Alternante*, utilizada no Capítulo 3 e descrita no Apêndice A. O módulo que especifica a funcionalidade do protocolo Bit-Alternante

(A_B) é o componente escolhido. Esse módulo possui dois pontos de interação: ponto de interação U – por onde é enviada a primitiva *receiveresponse* e por onde são recebidas as primitivas *sendrequest* e *receiverequest* do módulo *Usuário*; e ponto de interação N – por onde é enviada a primitiva *datarequest* e recebida a primitiva *dataresponse* do módulo *Rede*. A MEFE desse componente possui dois estados: *estab* e *ackwait*. Na Tabela 4.1 são apresentadas as primitivas de entrada e de saída para cada transição das MEFEs. Para definição de cada configuração da árvore foi necessário considerar também um *buffer* B das mensagens recebidas pelo protocolo. Isso foi feito devido aos aspectos funcionais desse protocolo: as transições *recrespa* e *recrespe* só estarão aptas a disparar se a primitiva *receiverequest* estiver na fila U e se o *buffer* (B) não estiver vazio. B armazena a mensagem recebida e que será enviada para o usuário destino. Assim, para esse exemplo, cada configuração da árvore de alcançabilidade possui os seguintes elementos:

$C_i = [estado, U, N, B]$ em que:

$estado = \{estab, ackwait\}$

$U = \{0, sendrequest, receiverequest\}$

$N = \{0, dataresponse\}$

$B = \{0, 1\}$

Na Figura 4.2 é apresentada a árvore de alcançabilidade do componente A_B do protocolo *Bit-Alternante*. São consideradas as transições dos componentes não selecionados (dos componentes *Usuário* e *Rede*) que são responsáveis por enviar entradas para as transições do componente A_B . Essas transições são representadas por arcos tracejados para diferenciar das transições do componente A_B e têm a descrição da primitiva que enviam. Essas transições indicam o recebimento das primitivas de comunicação nas filas U , N e no *buffer* B . São representadas também *configurações velhas*, as quais aparecem com o símbolo \uparrow , significando que a mesma

representa um *link* para a sua primeira ocorrência. Esse aspecto facilita a visualização da árvore.

Tabela 4.1. Primitivas de Entradas e de Saídas para as Transições dos Módulos do Protocolo Bit-Alternante.

Transições das MEFEs	Primitivas de Entrada	Primitivas de Saída
<i>Módulo Usuário</i>	<i>sendit</i>	-
	<i>reqit</i>	-
	<i>recvit</i>	<i>U.receiveresponse</i>
<i>Módulo A_B</i>	<i>send</i>	<i>U.sendrequest</i>
	<i>getdatae</i>	<i>D.dataresponse</i>
	<i>goodack</i>	<i>D.dataresponse</i>
	<i>recrespa</i>	<i>U.receiveresponse</i>
	<i>getdataaa</i>	<i>D.dataresponse</i>
	<i>badack</i>	<i>D.dataresponse</i>
	<i>recrespe</i>	<i>U.receiveresponse</i>
	<i>tossack</i>	<i>D.dataresponse</i>
<i>Módulo Rede</i>	<i>retrans</i>	<i>N.datarequest</i>
	<i>req1</i>	<i>N[1].datarequest</i>
	<i>req2</i>	<i>N[2].datarequest</i>
	<i>loss1</i>	<i>N[1].datarequest</i>
	<i>loss2</i>	<i>N[2].datarequest</i>

O controle do tamanho do árvore de alcançabilidade permite que os critérios de cobertura definidos neste capítulo possam ser aplicados para especificações mais complexas e sistemas reativos maiores. Entretanto, a construção da árvore considerando alguns componentes da especificação tem como desvantagem a impossibilidade de representar o comportamento global, pois o comportamento observável fica restrito aos componentes selecionados.

No Capítulo 6 é ilustrada a aplicação da FCCE utilizando o protocolo Bit-Alternante e a árvore de alcançabilidade exemplificada nesta seção. Nas próximas seções são descritos os procedimentos para obtenção dos requisitos e seqüências de teste para os critérios de cobertura para Statecharts e Estelle.

$C_0 = [\text{estab}, 0, 0, 0]$	$C_8 = [\text{ackwait}, 0, \text{dataresponse}, 0]$
$C_1 = [\text{estab}, \text{sendrequest}, 0, 0]$	$C_9 = [\text{ackwait}, 0, \text{dataresponse}, 1]$
$C_2 = [\text{estab}, \text{receiverequest}, 0, 0]$	$C_{10} = [\text{ackwait}, \text{receiverequest}, \text{dataresponse}, 0]$
$C_3 = [\text{estab}, \text{receiverequest}, 0, 1]$	$C_{11} = [\text{ackwait}, 0, 0, 0]$
$C_4 = [\text{estab}, \text{sendrequest}, \text{dataresponse}, 0]$	$C_{12} = [\text{ackwait}, \text{receiverequest}, \text{dataresponse}, 1]$
$C_5 = [\text{estab}, \text{receiverequest}, \text{dataresponse}, 0]$	$C_{13} = [\text{estab}, 0, 0, 1]$
$C_6 = [\text{estab}, \text{sendrequest}, 0, 1]$	$C_{14} = [\text{ackwait}, 0, 0, 1]$
$C_7 = [\text{ackwait}, \text{receiverequest}, 0, 0]$	$C_{15} = [\text{ackwait}, \text{receiverequest}, 0, 1]$

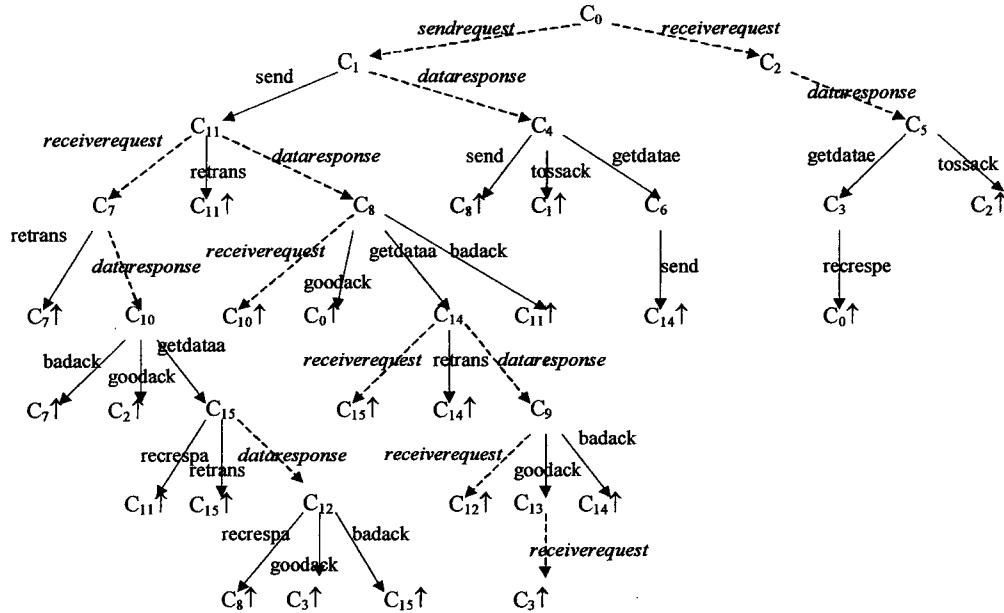


Figura 4.2. Árvore de Alcançabilidade para o Protocolo Bit-Alternante Especificado em Estelle.

4.3.3. Obtenção dos Requisitos de Teste dos Critérios FCCS e FCCE

De acordo com a definição de cada critério, a árvore de alcançabilidade é percorrida em profundidade até que todos os requisitos dos critérios sejam obtidos. Durante a construção da árvore dois conjuntos são definidos, os quais são utilizados pelos critérios de cobertura:

SC : conjunto de configurações da árvore.

T : conjunto de transições da árvore.

Para cada critério de cobertura, r_i é um requisito de teste e TR é o conjunto de requisitos de teste. Considera-se que toda vez que r_i é inserido em TR ($TR = TR \cup \{r_i\}$), um novo requisito de teste é buscado na árvore de alcançabilidade. r_i é formado por um conjunto de configurações que satisfazem as restrições do critério.

```

obtém_configuração( )
/* percorre a árvore (em profundidade) para buscar próxima
configuração a ser inserida*/
{
    se (in == 0) /* configuração não pode ser incluída em ri atual */
    se (config_anterior.prim_irmao <> -1) /* configuração possui irmão */
        configuração = config_anterior.irmao;
    se (configuração.retorno != -1) /* configuração é velha */
        configuração = configuração.retorno; /* primeira ocorrência de
                                                configuração */
        configuração.link = 1; /* marca que config. foi obtida a
                                partir de config. velha */
        lista_retorno = insere(configuração); /* insere config. em
                                                lista para backtrack */
    fim-se;
    senão /* configuração anterior não possui irmão */
        retorna(0); /* fim do requisito ri */
fim-se
senão se (in == 1) /* configuração anterior foi incluída em ri */
    se (config_anterior.prim_filho <> -1) /* configuração tem filho */
        configuração = config_anterior.prim_filho;
        se (configuração.retorno != -1) /* configuração é velha */
            configuração = configuração.retorno;
            configuração.link = 1;
            lista_retorno = insere(configuração);
        senão /* configuração já incluída não tem filho */
            retorna(0); /* fim do requisito ri */
        fim-se;
fim-se;
retorne(configuração);
}

obtém_requisito( )
/* inicia novo requisito de teste ri a partir de um requisito de teste já
percorrido e encerrado */
{
    configuração = config_anterior.pai;
    retira_ri(configuração); /* inicia outro ri retirando última config
                                da lista de ri anterior */
    enquanto (!encontrou) faça
        se (configuração.link == 1) /* config foi obtida de um link de
                                    uma config. velha */
            configuração = retira_retorno(lista_retorno); /* inicia
                                                    backtrack na lista_retorno */
        se (configuração.prim_irmao <> -1)
            configuração = configuração.prim_irmao;
            encontrou = 1;
        senão
            configuração = configuração.pai;
            se configuração == -1 /* percorrido acabou */
                retorna(0); /* não existe mais requisito novo */
                retira_ri(configuração);
            fim-se;
    fim-enquanto;
    retorna(configuração);
}

```

Figura 4.3. Algoritmo para Obtenção dos Requisitos de Teste a partir da Árvore de Alcançabilidade.

Para obter cada requisito de teste r_i , é implementado, para cada critério de cobertura, um algoritmo que determina se uma configuração pode ou não fazer parte do conjunto r_i . Esses algoritmos são descritos sucintamente a seguir. O procedimento da Figura 4.3 é chamado por todos os algoritmos dos critérios e retorna uma

configuração candidata a ser incluída em r_i . Esse algoritmo recebe como entrada um valor que significa se a configuração anterior (última configuração enviada) foi inserida ou não em r_i . Essa informação é utilizada para determinar a configuração candidata. O algoritmo trata as configurações velhas (com o símbolo \uparrow), fazendo os devidos retrocessos para encontrar a localização de configurações reais na árvore.

A seguir, descrevem-se resumidamente os procedimentos para obtenção dos requisitos de teste para cada critério de cobertura:

1. Critério *Todos-Caminhos*: cada requisito r_i é obtido percorrendo-se a árvore a partir do nó inicial C_0 tal que cada configuração C_j é inserida em r_i da seguinte forma:

```

Cj = obtem_configuração();
enquanto (Cj ≠ configuração terminal) faça
    ri = ri ∪ Cj;
    Cj = obtem_configuração();
fim-enquanto;
se (Cj == configuração terminal) então
    ri = ri ∪ Cj;
TR = TR ∪ {ri};

```

2. Critério *Todos-Caminhos-k-C₀-Configuração*: cada requisito r_i é obtido percorrendo-se a árvore a partir de C_0 e verificando cada nova configuração C_j :

```

k = 2; {número de vezes que C0 pode ocorrer em ri}
n = 0; {número de vezes que C0 ocorre em ri}
enquanto (n < k) faça
    Cj = obtem_configuração();
    enquanto (Cj ≠ C0) faça
        ri = ri ∪ Cj;
    fim-enquanto;
    n = n + 1;
    se (n ≤ k) então
        ri = ri ∪ Cj;
    fim-enquanto;
TR = TR ∪ {ri};

```

3. Critério *Todos-Caminhos-k-Configurações*: r_i é obtido percorrendo-se a árvore a partir de C_0 e verificando cada nova configuração C_j da seguinte forma:

```

k = 2; {número de vezes que  $C_j$  pode ocorrer em  $r_i$ }

enquanto ( $C_j \notin r_i$ ) ou ( $C_j \in r_i$  e num( $C_j$ ,  $r_i$ ) < k) faça
     $r_i = r_i \cup C_j$ ;
    incrementa( $C_j$ ,  $r_i$ ); {incrementa  $C_j$ }
     $C_j = obtém\_configuração()$ ;
fim-enquanto
TR = TR  $\cup \{r_i\}$ ;

```

4. Critério *Todos-Caminhos-com-um-Laço*: para esse critério somente uma das configurações pode aparecer 2 vezes em cada caminho, o que caracteriza um laço. Assim, cada r_i é obtido percorrendo-se a árvore a partir de C_0 e verificando cada nova configuração C_j :

```

flag = 0; {recebe 1 quando uma das configurações repete em
 $r_i$ }
enquanto (flag ≠ -1) faça
    enquanto ( $C_j \notin r_i$ ) faça
         $r_i = r_i \cup C_j$ ;
         $C_j = obtém\_configuração()$ ;
    fim-enquanto;
    se (flag == 0) então
         $r_i = r_i \cup C_j$ ;
        flag = 1;
    senão
        TR = TR  $\cup \{r_i\}$ ;
        flag = -1;
    fim-enquanto;

```

5. Critério *Todos-Caminhos-Simples*: r_i é obtido percorrendo-se a árvore a partir de C_0 , e verificando:

```

enquanto ( $C_j \notin r_i$ ) faça
     $r_i = r_i \cup C_j$ ;
     $C_j = obtém\_configuração()$ ;
fim-enquanto;
se ( $C_j == C_0$ ) então
     $r_i = r_i \cup C_j$ ;
    TR = TR  $\cup \{r_i\}$ ;

```

6. Critério *Todos-Caminhos-livre-Laço*: da mesma forma que o critério *Todos-Caminhos-Simples* os requisitos são obtidos da seguinte forma:

```

enquanto ( $C_j \notin r_i$ ) faça
     $r_i = r_i \cup C_j$ ;
     $C_j = obtém\_configuração()$ ;
fim-enquanto;
TR = TR  $\cup \{r_i\}$ ;

```

7. Critério *Todas-Transições*: o conjunto T de transições da árvore corresponde ao conjunto de requisitos desse critério.
8. Critério *Todas-Configurações*: o conjunto SC de configurações da árvore corresponde ao conjunto de requisitos desse critério.
9. Critério *Todas-Configurações-História*: identificam-se em T todas as transições com história t_h e percorre-se a árvore até que cada transição t_h seja alcançada.
10. Critério *Todas-Reações-em-Cadeia*: identificam-se em T todas as transições que desencadeiam *broadcasting* t_b e percorre-se a árvore até que as transições t_b sejam alcançadas.

Na Seção 4.4 esses procedimentos são aplicados em uma especificação em Statecharts. Na próxima seção, são descritos os procedimentos para geração de seqüências de teste adequadas, por construção, aos critérios FCCS e FCCE,

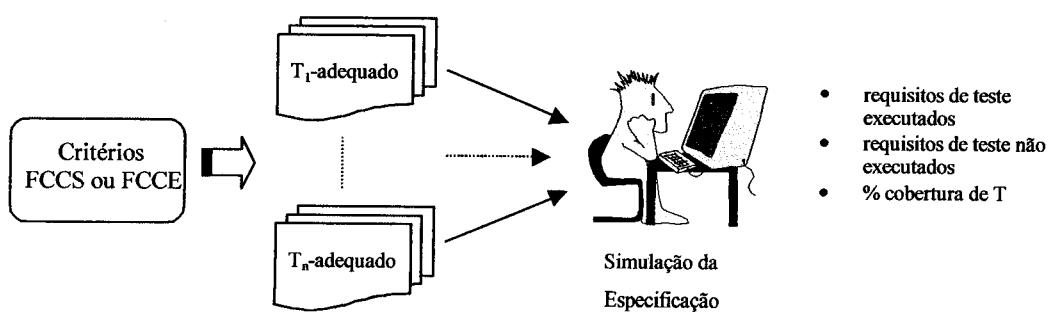
4.3.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCS e FCCE

Conforme descrito anteriormente, os critérios FCCS e FCCE podem ser empregados como critérios de adequação e como métodos de seleção de seqüências de teste. Esses aspectos são ilustrados na Figura 4.4. Na Figura 4.4a é ilustrada a utilização dos critérios FCCS ou FCCE para geração de seqüências de teste T_i adequadas. A partir das seqüências T_i 's, uma possível aplicação é utilizá-las para alimentar a simulação interativa ou *batch* da especificação, funcionando como conjuntos iniciais de seqüências de teste para a simulação. O usuário pode então avaliar o comportamento do modelo com os conjuntos T_i e inserir novas seqüências

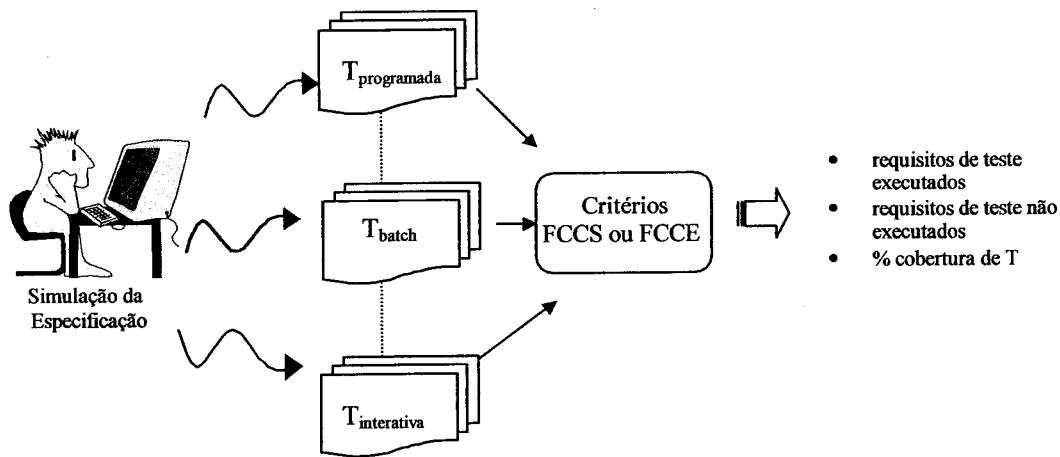
de teste conforme o andamento da simulação. Outra aplicação seria utilizar as seqüências T_i 's durante a realização de testes de conformidade.

A utilização dos critérios FCCS ou FCCE como critérios de adequação é ilustrada na Figura 4.4b. Nesse caso, os critérios são utilizados para avaliar a cobertura de seqüências de teste geradas pela simulação. A partir de um conjunto T , obtido durante a simulação, o usuário pode aplicar os critérios FCCS ou FCCE (dependendo do contexto) e obter a porcentagem de cobertura para as configurações, transições, paralelismo, história (no caso de Statecharts), enfim, para qualquer um dos critérios definidos anteriormente. Esse aspecto pode ser explorado também para seqüências de teste adequadas a outros critérios de teste (Teste de Mutação, por exemplo), para seqüências de teste geradas aleatoriamente ou geradas manualmente.

Nos dois casos (método de seleção e critério de adequação), as seguintes informações podem ser obtidas: requisitos de teste executados pelas seqüências de teste, requisitos que não foram executados pelo conjunto de seqüências de teste aplicado e porcentagem de cobertura obtida em relação aos critérios de cobertura.



a) Critérios de cobertura e a Geração de Seqüências de Teste.



b) Critérios de Cobertura e a Avaliação de Seqüências de Teste.

Figura 4.4. Os Critérios de Cobertura na Validação de Especificações.

As seqüências de teste adequadas por construção podem ser geradas à medida que a árvore é percorrida para obtenção dos requisitos de teste. Quando a configuração C_j é incluída em r_i a transição percorrida para encontrar C_j é incluída em um conjunto t_i que corresponde à seqüência de teste que percorre o requisito r_i . Analogamente à obtenção de TR , um conjunto TS é obtido, o qual é formado pelas seqüências de teste t_i que percorrem os requisitos r_i 's.

O conjunto de requisitos de teste para os critérios *Todas-Transições*, *Todas-Configurações*, *Todas-Configurações-História* e *Todas-Reações-em-Cadeia* é obtido sem percorrer a árvore de alcançabilidade. Através dos conjuntos T e SC , gerados durante a construção da árvore, é possível obter os requisitos desses critérios. Dessa forma, para obtenção de seqüências de teste adequadas por construção a esses critérios, foram definidos algoritmos para percorrer a árvore de modo a obter seqüências de teste que executem, no mínimo uma vez, cada requisito de teste desses critérios de cobertura.

Na próxima seção, é ilustrada a obtenção dos requisitos de teste para os critérios FCCS e a utilização desses critérios como métodos de seleção de seqüências de teste.

4.4. Um Exemplo de Aplicação dos Critérios FCCS

Para ilustrar a aplicação dos critérios de cobertura, a FCCS é aplicada a um statechart que descreve um Manipulador de Mouse simplificado. Esse statechart foi extraído de Cangussu et al. (1995) e registra o número de vezes que o botão do mouse foi pressionado (número de *clicks* do mouse), dentro de um intervalo de 4 unidades de tempo, descritas como *tops*. O statechart é apresentado na Figura 4.5, contendo 3 componentes ortogonais: *Numero_Top*, *Numero_Click* e *Emissao*, os quais são executados em paralelo. O disparo da transição *top/rst* do componente *Numero_Top*, irá gerar o evento *rst* que reinicia o componente *Numero_Click*, sincronizando os dois componentes. Condições foram associadas às transições do componente *Numero_Click* para garantir o funcionamento correto desse componente. Por exemplo, a condição *in(Três)* da transição *a2* garante que, se os eventos *click* e *top* ocorrerem juntos (estando o statechart no estado *Três* do componente *Numero_Top*), o número de *clicks* é incrementado antes da reinicialização do statechart. A árvore de alcançabilidade desse exemplo é apresentada na Figura 4.6.

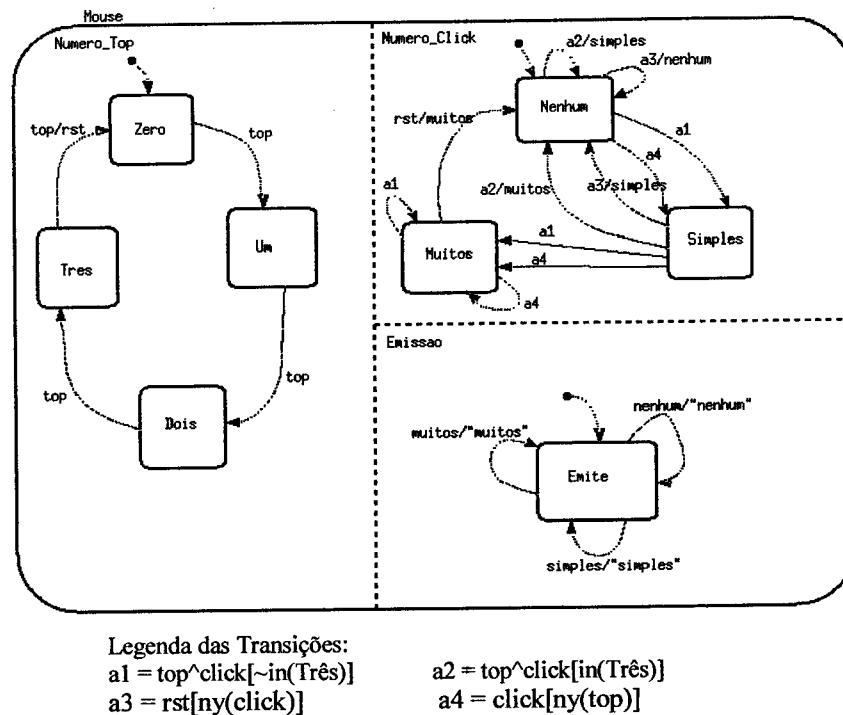


Figura 4.5. Statechart do Manipulador de Mouse (Cangussu et al., 1995).

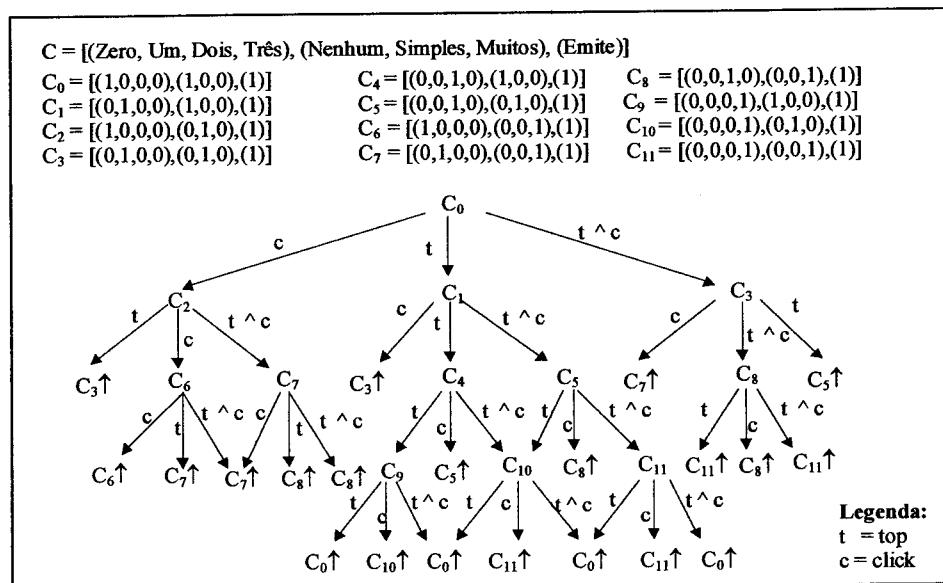


Figura 4.6. Árvore de Alcançabilidade do Statechart Figura 4.5 (Masiero et al., 1994).

Utilizando os procedimentos descritos na Seção 4.3, os critérios da FCCS foram aplicados. Na Tabela 4.2 são apresentados alguns requisitos e seqüências de teste gerados para cada critério de cobertura.

Tabela 4.2. Subconjunto de Requisitos de Teste (tr) e Seqüências de Teste (ts) para a Especificação da Figura 4.5.

Critérios FCCS	Requisitos de Teste e Seqüências de Teste
<i>Todos-Caminhos-k-C₀-Configuração</i>	$tr = \{(c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_0), (c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_{11}, c_0), (c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_0), \dots\}$ $ts = \{(c, t, c, c, t, t, t), (c, t, c, c, t, t, c, t), (c, t, c, c, t, t, \{t,c\}), (c, t, c, c, t, \{t,c\}, c, t), (c, t, c, c, t, \{t,c\}, \{t,c\}), (c, t, c, c, \{t,c\}, t, c, t), (c, t, c, c, \{t,c\}, t, \{t,c\}), \dots\}$
<i>Todos-Caminhos-k-Configurações</i>	$tr = \{(c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_2, c_3, c_8, c_{11}), (c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_0, c_2, c_3, c_5, c_{10}, c_{11}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_2, c_3, c_5, c_8, c_{11}), (c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_0, c_2, c_3, c_5, c_{10}, c_{11}), (c_0, c_2, c_3, c_7, c_7, c_8, c_{11}, c_0, c_2, c_6, c_6), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_3, c_8, c_{11}), \dots\}$ $ts = \{(c, t, c, c, t, t, t, c, t, \{t,c\}), (c, t, c, c, t, t, t, c, t, t, t, c), (c, t, c, c, t, t, t, c, t, t, t, t, c, t, t, \{t,c\}), (c, t, c, c, t, t, t, c, t, t, t, t, t, c, t, t, t, \{t,c\}), (c, t, c, c, t, t, t, c, c, c), (c, t, c, c, t, t, t, t, t, t, c, t, t, t, t, \{t,c\}), \dots\}$
<i>Todos-Caminhos-com-um-Laço</i>	$tr = \{(c_0, c_2, c_3, c_7, c_7, c_8, c_{11}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_4, c_9, c_{10}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_4, c_5, c_{10}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_5, c_{10}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_5, c_{11}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_6, c_{11}), (c_0, c_2, c_3, c_7, c_8, c_{11}, c_0, c_1, c_7, c_{11}), \dots\}$ $ts = \{(c, t, c, c, t, t), (c, t, c, t, t, t, t, t, t, c), (c, t, c, t, t, t, t, t, t, c, t), (c, t, c, t, t, t, t, t, t, t, t, c, t, t, t, \{t,c\}), (c, t, c, t, t, t, t, t, t, t, t, c, t, t, t, t, c, t, \{t,c\}), \dots\}$
<i>Todos-Caminhos-Simples</i>	$tr = \{(c_0, c_2, c_3, c_7, c_8, c_{11}, c_0), (c_0, c_2, c_3, c_8, c_{11}, c_0), (c_0, c_2, c_3, c_5, c_{10}, c_0), (c_0, c_2, c_3, c_5, c_{10}, c_{11}, c_0), (c_0, c_2, c_3, c_5, c_{10}, c_0), (c_0, c_2, c_3, c_5, c_8, c_{11}, c_0), (c_0, c_2, c_3, c_5, c_{11}, c_0), \dots\}$ $ts = \{(c, t, c, t, t, t), (c, t, \{t,c\}, t, t), (c, t, t, t, t, t), (c, t, t, t, c, t), (c, t, t, t, t, t, t, t, c, t), (c, t, t, t, t, t, t, t, t, c, t, t, t, \{t,c\}), (c, t, t, t, t, t, t, t, t, t, c, t, t, t, t, c, t, \{t,c\}), \dots\}$
<i>Todos-Caminhos-livre-Laço</i>	$tr = \{(c_0, c_2, c_3, c_7, c_8, c_{11}), (c_0, c_2, c_3, c_8, c_{11}), (c_0, c_2, c_3, c_5, c_{10}, c_{11}), (c_0, c_2, c_3, c_5, c_{11}, c_0), (c_0, c_2, c_3, c_5, c_{11}), (c_0, c_2, c_6, c_7, c_8, c_{11}), (c_0, c_2, c_7, c_8, c_{11}), (c_0, c_1, c_3, c_7, c_8, c_{11}), \dots\}$ $ts = \{(c, t, c, t, t), (c, t, \{t,c\}, t), (c, t, t, t, c), (c, t, t, c, t), (c, t, t, \{t,c\}), (c, c, t, t, t), (c, \{t,c\}, t, t), (t, c, c, t, t, t), (t, t, t, c, t, t, t, t, c, t), \dots\}$
<i>Todas-Transições</i>	$tr = \{(c_0, c_2), (c_2, c_3), (c_2, c_6), (c_2, c_7), (c_3, c_7), (c_7, c_7), (c_7, c_8), (c_8, c_{11}), (c_{11}, c_0), (c_0, c_1), (c_1, c_3), (c_3, c_5), (c_5, c_{10}), (c_{10}, c_0), (c_1, c_5), (c_5, c_8), \dots\}$ $ts = \{(c, t), (c, c, c), (c, c, t), (c, \{t,c\}, c), (c, \{t,c\}, t), (t, c), (t, t, t, t), (t, t, t, c), (t, t, t, t, t, t, t, c, t), \dots\}$
<i>Todas-Configurações</i>	$tr = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}\}$ $ts = \{(c, c), (c, \{t,c\}), (t, t, t), (t, t, \{t,c\}), (t, \{t,c\}, \{t,c\}), (\{t,c\}, \{t,c\})\}$
<i>Todas-Reações-em-Cadeia</i>	$tr = \{(c_9, c_0), (c_9, c_0), (c_{10}, c_0), (c_{10}, c_0), (c_{11}, c_0), (c_{11}, c_0)\}$ $ts = \{(t, t, t, t), (t, t, t, \{t,c\}), (t, t, \{t,c\}, t), (t, t, \{t,c\}, \{t,c\}), (t, \{t,c\}, \{t,c\}, t), (t, \{t,c\}, \{t,c\}, \{t,c\})\}$

Na Tabela 4.3 são apresentados o total de requisitos e o total de seqüências de teste (adequadas por construção). O critério *Todos-Caminhos* gera um número infinito de caminhos, consequentemente, um número infinito de requisitos de teste e, portanto, não foi aplicado. Como o statechart não possui história, o critério *Todas-Configurações-História* também não foi aplicado.

Tabela 4.3. Total de Requisitos e de Seqüências de Teste Gerados para os Critérios FCCS para o Statechart da Figura 4.5.

Critérios de Cobertura	# Requisitos de Teste	# Seqüências de Teste
<i>Todos-Caminhos-k-Configuração</i>	229	229
<i>Todos-Caminhos-k-Configurações</i>	16968	16968
<i>Todos-Caminhos-com-um-Laço</i>	622	622
<i>Todos-Caminhos-Simples</i>	41	41
<i>Todos-Caminhos-livre-Laço</i>	25	25
<i>Todas-Transições</i>	36	23
<i>Todas-Configurações</i>	12	06
<i>Todas-Reações-em-Cadeia</i>	06	06

Conforme descrito na Seção 4.3, o número de seqüências de teste gerado é igual ao número de requisitos de teste para a maioria dos critérios de cobertura (Tabela 4.3). Isso ocorre porque cada requisito de teste corresponde a um caminho distinto na árvore de alcançabilidade.

No Capítulo 5 é ilustrada a utilização dos critérios FCCS como critérios de adequação para avaliar a cobertura de seqüências de teste geradas pela simulação. Além disso, esses critérios são comparados com o Teste de Mutação para Statecharts (Fabbri, 1996; Fabbri et al. 1999a), avaliando-se o *strength* desses critérios.

A seguir é apresentada a análise teórica e o estabelecimento da relação de inclusão dos critérios FCCS e FCCE.

4.5. Análise Teórica dos Critérios de Cobertura: Relação de Inclusão

Como descrito no Capítulo 2, dois aspectos são considerados para avaliação teórica de critérios de teste: *relação de inclusão* e *complexidade dos critérios* (Rapps e Weyuker, 1985; Ntafos, 1988; Maldonado, 1991). A relação de inclusão estabelece uma ordem parcial entre os critérios de teste. A complexidade de um critério *C* é

definida como o número máximo de casos de testes requeridos no pior caso. Esta seção apresenta os resultados da análise de inclusão dos critérios de cobertura.

Um critério de teste C_{R1} *inclui* um critério C_{R2} se para qualquer conjunto de caminhos P que satisfaz C_{R1} implica que P também satisfaz C_{R2} , para qualquer programa ou especificação. C_{R1} *inclui estritamente* C_{R2} , representado por $C_{R1} \Rightarrow C_{R2}$, se C_{R1} *inclui* C_{R2} , mas C_{R2} não *inclui* C_{R1} . Os critérios C_{R1} e C_{R2} são *incomparáveis* se C_{R1} não *inclui* C_{R2} e C_{R2} não *inclui* C_{R1} (Rapps e Weyuker, 1985).

4.5.1. Relação de Inclusão dos Critérios FCCS

A relação de inclusão dos critérios FCCS é dada pelos seguintes teoremas:

Teorema 1: Os critérios FCCS para especificações em Statecharts obedecem à hierarquia da Figura 4.7.

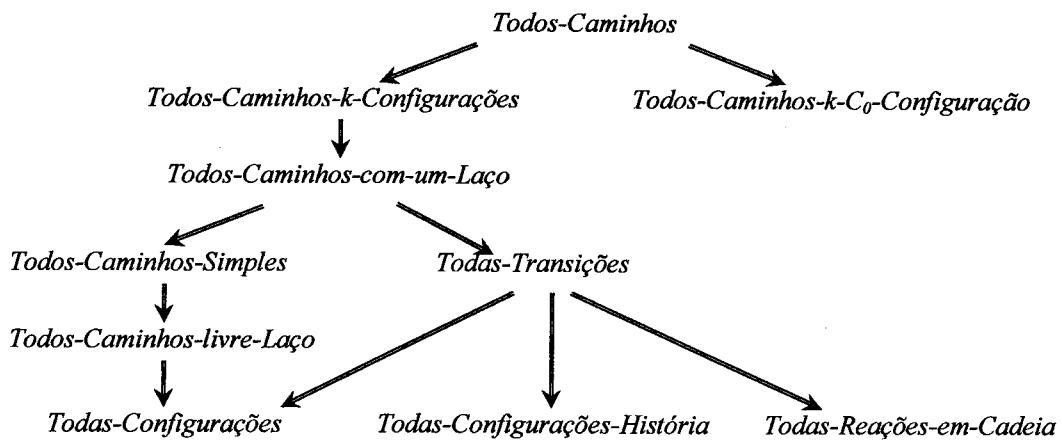


Figura 4.7. Relação Hierárquica dos Critérios FCCS.

Prova: Dado que a relação de inclusão é transitiva, basta demonstrar as seguintes relações:

- i) $\text{Todos-Caminhos} \Rightarrow \text{Todos-Caminhos-k-Configurações};$
- ii) $\text{Todos-Caminhos-k-Configurações} \Rightarrow \text{Todos-Caminhos-com-um-Laço}.$

- iii) Todos-Caminhos-com-um-Laço \Rightarrow Todos-Caminhos-Simples.
- iv) Todos-Caminhos-Simples \Rightarrow Todos-Caminhos-livre-Laço.
- v) Todos-Caminhos-livre-Laço \Rightarrow Todas-Configurações
- vi) Todos-Caminhos-com-um-Laço \Rightarrow Todas-Transições
- vii) Todas-Transições \Rightarrow Todas-Configurações
- viii) Todas-Transições \Rightarrow Todas-Configurações-História
- ix) Todas-Transições \Rightarrow Todas-Reações-em-Cadeia
- x) Todos-Caminhos \Rightarrow Todos-Caminhos-k-C₀-Configuração
- xi) Todos-Caminhos-k-Configurações e Todos-Caminhos-k-C₀-Configuração são incomparáveis.
- xii) Todos-Caminhos-Simples e Todas-Transições são incomparáveis. Todos-Caminhos-livre-Laço e Todas-Transições são incomparáveis.
- xiii) Todas-Configurações e Todas-Configurações-História são incomparáveis.
- xiv) Todas-Configurações e Todas-Reações-em-Cadeia são incomparáveis.
- xv) Todas-Configurações-História e Todas-Reações-em-Cadeia são incomparáveis.

i) *Todos-Caminhos \Rightarrow Todos-Caminhos-k-Configurações.*

Seja P_1 *Todos-Caminhos*-adequado. Assim, P_1 contém todos os caminhos possíveis, sendo que alguns desses caminhos podem ter comprimento infinito. Seja P_2 *Todos-Caminhos-k-Configurações*-adequado. Para todo caminho $p \in P_2$ existe pelo menos um caminho $m \in P_1$ que inclui p ; com isso p é incluído em P_1 . Pode-se concluir que P_1 também é *Todos-Caminhos-k-Configurações*-adequado, isto é, o critério *Todos-Caminhos* inclui o critério *Todos-Caminhos-k-Configurações*. Por

outro lado, P_2 não satisfaz o critério *Todos-Caminhos* porque basta considerar um caminho com k_l repetições de uma configuração C_i , $k_l > k$. Esse caminho não está necessariamente incluído em P_2 . Dessa forma, o critério *Todos-Caminhos* inclui estritamente o critério *Todos-Caminhos-k-Configurações*.

ii) *Todos-Caminhos-k-Configurações* \Rightarrow *Todos-Caminhos-com-um-Laço*.

Seja P_1 *Todos-Caminhos-k-Configurações*-adequado e P_2 *Todos-Caminhos-com-um-Laço*-adequado, ou seja, cada caminho $p \in P_2$ é formado por uma seqüência de configurações $C_0, C_i, \dots, C_j, C_i, C_k$, em que apenas uma configuração C_i pode aparecer repetida uma vez no caminho, caracterizando um laço. Para todo $p \in P_2$, existe pelo menos um caminho $m \in P_1$ tal que m inclui p , pois se assim não o fosse, P_1 não seria *Todos-Caminhos-k-Configurações*-adequado. Assim sendo, todo P_1 *Todos-Caminhos-k-Configurações*-adequado é *Todos-Caminhos-com-um-Laço*-adequado. Por outro lado, considere que P_2 inclui somente caminhos com um laço. P_2 não satisfaz o critério *Todos-Caminhos-k-Configurações*, pois basta considerar um caminho de P_1 com k repetições de, no mínimo, duas configurações diferentes C_j e C_k . Esse caminho não está necessariamente incluído em P_2 . Assim, existe pelo menos um conjunto P_2 adequado a *Todos-Caminhos-com-um-Laço* que não é *Todos-Caminhos-k-Configurações* adequado, de forma que o critério *Todos-Caminhos-com-um-Laço* não inclui o critério *Todos-Caminhos-k-Configurações*. Assim sendo, o critério *Todos-Caminhos-k-Configurações* inclui estritamente o critério *Todos-Caminhos-com-um-Laço*.

iii) *Todos-Caminhos-com-um-Laço* \Rightarrow *Todos-Caminhos-Simples*.

Seja P_1 *Todos-Caminhos-com-um-Laço*-adequado e P_2 *Todos-Caminhos-Simples*-adequado. Cada caminho $p \in P_2$ é composto de uma seqüência de

configurações $C_0, C_j, \dots, C_k, C_n$ sendo que todas as configurações no intervalo C_j, \dots, C_k são distintas (não repetidas) e as configurações C_0 e C_n podem ser repetidas (todo caminho inicia em C_0). Para todo $p \in P_2$ existe pelo menos um caminho $m \in P_1$ tal que m inclui p , pois se assim não fosse, P_1 não seria *Todos-Caminhos-com-um-Laço*-adequado. Assim, todo P_1 *Todos-Caminhos-com-um-Laço*-adequado é *Todos-Caminhos-Simples*-adequado. Por outro lado, P_2 não satisfaz o critério *Todos-Caminhos-com-um-Laço*, pois basta considerar um caminho de P_1 contendo um laço de uma configuração C_m , sendo $C_m \neq C_0$. Esse caminho não está necessariamente incluído em P_2 e, desse modo, existe pelo menos um conjunto P_2 adequado ao critério *Todos-Caminhos-Simples* que não é adequado ao critério *Todos-Caminhos-com-um-Laço*. Assim sendo, o critério *Todos-Caminhos-com-um-Laço* inclui estritamente o critério *Todos-Caminhos-Simples*.

iv) *Todos-Caminhos-Simples* \Rightarrow *Todos-Caminhos-livre-Laço*.

Seja P_1 *Todos-Caminhos-Simples*-adequado e P_2 *Todos-Caminhos-livre-Laço*-adequado, ou seja, todo caminho $p \in P_2$ é composto de uma seqüência de configurações C_0, C_i, \dots, C_n sendo que todas as configurações são distintas (não repetidas). Para todo $p \in P_2$ existe pelo menos um caminho $m \in P_1$ tal que m inclui p , pois se assim não o fosse, P_1 não seria *Todos-Caminhos-Simples*-adequado. Dessa forma, todo P_1 *Todos-Caminhos-Simples*-adequado é *Todos-Caminhos-livre-Laço*-adequado. Por outro lado, P_2 não satisfaz o critério *Todos-Caminhos-Simples*, pois para isso, basta considerar um caminho de P_1 da seguinte forma: C_0, C_j, \dots, C_0 . Esse caminho não está necessariamente incluído em P_2 . Assim sendo, o critério *Todos-Caminhos-Simples* inclui estritamente o critério *Todos-Caminhos-livre-Laço*.

v) ***Todos-Caminhos-livre-Laço \Rightarrow Todas-Configurações.***

P_1 é *Todos-Caminhos-livre-Laço*-adequado. Suponha que P_1 não seja *Todas-Configurações*-adequado. Portanto, existe pelo menos uma C_i tal que não existe caminho $p \in P_1$ em que C_i está incluída em p . Mas para toda configuração, em particular C_b , existe um caminho $m = C_0 \dots C_b$ tal que C_i está incluída em m e esse caminho é livre de laço. Pode-se concluir que P_1 não inclui m , assim sendo, P_1 não seria *Todos-Caminhos-livre-Laço*-adequado (absurdo). Assim sendo, todo caminho P_1 é *Todas-Configurações*-adequado. Por outro lado, um conjunto P_2 *Todas-Configurações*-adequado não é *Todos-Caminhos-livre-Laço*-adequado. Para isso, basta considerar o exemplo da Seção 4.4: o conjunto P_2^1 é *Todas-Configurações*-adequado mas não é *Todos-Caminhos-livre-Laço*-adequado pois não inclui o caminho livre de laço $C_0, C_2, C_3, C_7, C_8, C_{11}$. Assim sendo, conclui-se que o critério *Todos-Caminhos-livre-Laço* inclui estritamente o critério *Todas-Configurações*.

vi) ***Todos-Caminhos-com-um-Laço \Rightarrow Todas-Transições.***

P_1 é *Todos-Caminhos-com-um-Laço*-adequado. Suponha que P_1 não seja *Todas-Transições*-adequado. Portanto, existe pelo menos um arco (C_i, C_j) tal que não existe caminho $p \in P_1$ em que (C_i, C_j) está incluído em p . Entretanto, para todo arco, em particular para o arco (C_i, C_j) , existe um caminho m_i livre de laço de C_0 a C_i . Para $m = m_i \wedge C_j$, m contém o arco (C_i, C_j) e está incluído em P_1 . Portanto, P_1 é *Todas-Transições*-adequado. Por outro lado, um conjunto P_2 *Todas-Transições*-adequado não é *Todos-Caminhos-com-um-Laço*-adequado. Basta considerar o exemplo da

¹ P_2 *Todas-Configurações*-adequado = $\{(C_0, C_2, C_6), (C_0, C_2, C_7), (C_0, C_1, C_4, C_9), (C_0, C_1, C_4, C_{10}), (C_0, C_1, C_5, C_{11}), (C_0, C_3, C_8)\}$

Seção 4.4: é possível obter um conjunto P_2 ² *Todas-Transições*-adequado que não é *Todos-Caminhos-com-um-Laço*-adequado pois não inclui o caminho com um laço $C_0, C_3, C_8, C_8, C_{11}$. Assim sendo, conclui-se que o critério *Todos-Caminhos-com-um-Laço* inclui estritamente o critério *Todas-Transições*.

vii) ***Todas-Transições* \Rightarrow *Todas-Configurações*.**

P_1 é *Todas-Transições*-adequado. Suponha que P_1 não seja *Todas-Configurações*-adequado. Portanto, existe pelo menos uma configuração C_i tal que não existe caminho $p \in P_1$ em que C_i está incluída em p . Com isso, todo arco (C_i, C_j) ou (C_k, C_l) não existe em P_1 . Assim, P_1 não seria *Todas-Transições*-adequado (absurdo). Desse modo, todo caminho P_1 é *Todas-Configurações*-adequado. Por outro lado, um conjunto P_2 *Todas-Configurações*-adequado não é *Todas-Transições*-adequado. Para isso, basta considerar o exemplo da Seção 4.4: é possível obter um conjunto P_2 ¹ *Todas-Configurações*-adequado que não é *Todas-Transições*-adequado pois não inclui o arco (C_2, C_3) . Assim sendo, conclui-se que o critério *Todas-Transições* inclui estritamente o critério *Todas-Configurações*.

viii) ***Todas-Transições* \Rightarrow *Todas-Configurações-História*.**

P_1 é *Todas-Transições*-adequado e P_2 é *Todas-Configurações-História*-adequado, ou seja, para todo arco $(C_i, C_j) \in t_h$ (conjunto de transições com símbolo história), existe pelo menos um caminho $p \in P_2$, tal que (C_i, C_j) está incluído em p . Suponha que P_1 não é *Todas-Configurações-História*-adequado. Isso significa que para qualquer caminho $m \in P_1$ o arco (C_i, C_j) não está incluído em m . Se isso fosse verdadeiro, P_1 não seria *Todas-Transições*-adequado (absurdo). Assim sendo, todo

² P_2 *Todas-Transições*-adequado = $\{(C_0, C_2, C_3, C_7, C_7, C_8, C_{11}, C_0), (C_0, C_2, C_6, C_6, C_7, C_8, C_8, C_{11}), (C_0, C_2, C_6, C_7, C_8), (C_0, C_2, C_7, C_8, C_{11}, C_0), (C_0, C_1, C_3, C_8), (C_0, C_1, C_3, C_5, C_{10}, C_0), (C_0, C_1, C_4, C_9, C_0), (C_0, C_1, C_4, C_9, C_{10}, C_0), (C_0, C_1, C_4, C_5, C_8), (C_0, C_1, C_4, C_{10}, C_{11}), (C_0, C_1, C_5, C_{11}, C_0), (C_0, C_1, C_4, C_0), (C_0, C_3)\}$

caminho P_1 é é *Todas-Configurações-História*-adequado. Por outro lado, P_2 não é *Todas-Transições*-adequado. Para isso, basta considerar o statechart com história da Seção 4.3 (Figura 4.1a): é possível obter um conjunto P_2 ³ *Todas-Configurações-História*-adequado que não é *Todas-Transições*-adequado pois não inclui o arco (C_2, C_1) . Assim sendo, conclui-se que o critério *Todas-Transições* inclui estritamente o critério *Todas-Configurações-História*.

ix) *Todas-Transições* \Rightarrow *Todas-Reações-em-Cadeia*.

P_1 é *Todas-Transições*-adequado e P_2 é *Todas-Reações-em-Cadeia*-adequado, ou seja, para todo arco $(C_i, C_j) \in t_b$ (conjunto de transições com ação contendo evento), existe pelo menos um caminho $p \in P_2$, tal que (C_i, C_j) está incluído em p . Suponha que P_1 não é *Todas-Reações-em-Cadeia*-adequado. Isso significa que para qualquer caminho $m \in P_1$ o arco (C_i, C_j) não está incluído em m . Se isso fosse verdadeiro, P_1 não seria *Todas-Transições*-adequado (absurdo). Assim sendo, todo caminho P_1 é *Todas-Reações-em-Cadeia*-adequado. Por outro lado, P_2 não é *Todas-Transições*-adequado. Para isso, basta considerar o statechart da Seção 4.4: é possível obter um conjunto P_2 *Todas-Reações-em-Cadeia*-adequado = $\{(C_0, C_1, C_4, C_9, C_0, C_1, C_4, C_9, C_0), (C_0, C_1, C_4, C_{10}, C_0, C_1, C_4, C_{10}, C_0), (C_0, C_1, C_5, C_{11}, C_0, C_1, C_5, C_{11}, C_0)\}$ que não é *Todas-Transições*-adequado pois não inclui o arco (C_0, C_2) . Assim sendo, conclui-se que o critério *Todas-Transições* inclui estritamente o critério *Todas-Reações-em-Cadeia*.

x) *Todos-Caminhos* \Rightarrow *Todos-Caminhos-k-C0-Configuração*.

Seja P_1 *Todos-Caminhos*-adequado e P_2 *Todos-Caminhos-k-C0-Configuração*-adequado, ou seja, cada caminho $p \in P_2$ é composto de uma seqüência de

³ P_2 *Todas-Configurações-História*-adequado = $\{(C_0, C_3, C_8, C_0), (C_0, C_2, C_3, C_8, C_2), (C_0, C_2, C_6, C_3, C_8, C_6), (C_0, C_2, C_5, C_3, C_8, C_5), (C_0, C_2, C_5, C_7, C_3, C_8, C_7), (C_0, C_1, C_4, C_0), (C_0, C_2, C_5, C_7, C_1, C_4, C_7)\}$

configurações C_0, C_j, \dots, C_k , C_0 , e as configurações no intervalo C_j, \dots, C_k podem ser repetidas n vezes, até que seja encontrada a configuração C_0 . Para todo $p \in P_2$ existe pelo menos um caminho $m \in P_1$ tal que m inclui p , pois se assim não o fosse, P_1 não seria *Todos-Caminhos*-adequado. Assim sendo, todo P_1 *Todos-Caminhos*-adequado é *Todos-Caminhos-k-C₀-Configuração*-adequado. Por outro lado, considere que P_2 inclui somente caminhos em que C_0 . Por outro lado, P_2 não satisfaz o critério *Todos-Caminhos* porque seria necessário um caminho $p \in P_2$ com k_1 repetições da configuração C_0 , $k_1 > k$. Esse caminho não está necessariamente incluído em P_2 . Assim, existe pelo menos um conjunto P_2 *Todos-Caminhos-k-C₀-Configuração*-adequado que não é *Todos-Caminhos*-adequado, de forma que o critério *Todos-Caminhos-k-C₀-Configuração* não inclui o critério *Todos-Caminhos*. Assim sendo, o critério *Todos-Caminhos* inclui estritamente o critério *Todos-Caminhos-k-Configurações*.

xi) *Todos-Caminhos-k-Configurações* e *Todos-Caminhos-k-C₀-Configuração* são incomparáveis.

Seja P_1 *Todos-Caminhos-k-Configurações*-adequado e P_2 *Todos-Caminhos-k-C₀-Configuração*-adequado. P_1 não satisfaz o critério *Todos-Caminhos-k-C₀-Configuração* porque seria necessário que P_1 incluisse caminhos contendo k_1 repetições de uma configuração C_i , $k_1 > k$. P_2 também não satisfaz o critério *Todos-Caminhos-k-Configurações* porque seria necessário que P_2 incluisse necessariamente caminhos com uma seqüência de configurações $C_0, C_j, \dots, C_0, \dots, C_k$ em que a configuração C_0 é repetida mas não é a última configuração do caminho. Dessa forma, o critério *Todos-Caminhos-k-Configurações* e o critério *Todos-Caminhos-k-C₀-Configuração* são incomparáveis.

xii) ***Todos-Caminhos-Simples e Todas-Transições são incomparáveis.***

Todos-Caminhos-livre-Laço e Todas-Transições são incomparáveis.

P_1 é *Todos-Caminhos-Simples*-adequado, P_2 é *Todos-Caminhos-livre-Laço*-adequado e P_3 é *Todas-Transições*-adequado. P_1 não satisfaz o critério *Todas-Transições* porque seria necessário que P_1 incluisse caminhos $C_0, \dots C_j, C_j \dots C_m$, que significa incluir transições (C_j, C_j) . P_2 também não satisfaz o critério *Todas-Transições* porque, considerando que todo caminho $p \in P_2$ sempre inicia pela configuração C_0 , os arcos (C_i, C_0) não são incluídos necessariamente em P_2 , pois para isso seria necessário que a configuração C_0 estivesse repetida em P_2 . P_3 não satisfaz o critério *Todos-Caminhos-Simples* e o critério *Todos-Caminhos-livre-Laço*, pois, considerando o conjunto adequado ao critério *Todas-Transições*² para o exemplo da Seção 4.4, esse conjunto não inclui o caminho livre de laço $C_0, C_2, C_3, C_5, C_{11}$ e o caminho simples $C_0, C_2, C_3, C_7, C_8, C_{11}, C_0$. Assim sendo, conclui-se que o critério *Todas-Transições* e o critério *Todos-Caminhos-Simples* são incomparáveis e o critério *Todas-Transições* e o critério *Todos-Caminhos-livre-Laço* também são incomparáveis.

xiii) ***Todas-Configurações e Todas-Configurações-História são incomparáveis.***

P_1 é *Todas-Configurações*-adequado e P_2 é *Todas-Configurações-História*-adequado. Considerando o exemplo da Seção 4.3 (Figura 4.1a), $P_1 = \{(C_0, C_1, C_4, C_0, C_3, C_8, C_0), (C_0, C_2, C_5, C_7, C_6)\}$ *Todas-Configurações*-adequado não satisfaz o critério *Todas-Configurações-História*, pois P_1 não inclui o arco $(C_8, C_2) \in t_h$ (conjunto de transições com símbolo história). Da mesma forma, se nesse exemplo houvesse símbolo história H somente na transição f , o conjunto $P_2 = \{(C_0, C_1, C_4, C_0), (C_0, C_2, C_5, C_7, C_1, C_4, C_7)\}$ seria *Todas-Configurações-História*-adequado. Esse conjunto não inclui a configuração C_3 , podendo-se concluir que o critério *Todas-*

Configurações-História não inclui o critério *Todas-Configurações*. Assim sendo, o critério *Todas-Configurações-História* e o critério *Todas-Configurações* são incomparáveis.

xiv) *Todas-Configurações* e *Todas-Reações-em-Cadeia* são incomparáveis.

P_1 é *Todas-Configurações*-adequado e P_2 é *Todas-Reações-em-Cadeia*-adequado. Considerando o exemplo da Seção 4.3 (Figura 4.1a), o conjunto $P_1 = \{(C_0, C_1, C_4, C_0, C_3, C_8, C_0), (C_0, C_2, C_5, C_7, C_6)\}$ *Todas-Configurações*-adequado não satisfaz o critério *Todas-Reações-em-Cadeia*, pois não inclui os arcos (C_7, C_2) e $(C_6, C_2) \in t_b$ (conjunto de transições com ação contendo evento). Desse modo, o critério *Todas-Configurações* não satisfaz o critério *Todas-Reações-em-Cadeia*. Da mesma forma, o conjunto $P_2 = \{(C_0, C_2, C_5, C_7, C_2), (C_0, C_2, C_6, C_2)\}$ *Todas-Reações-em-Cadeia*-adequado, não satisfaz o critério *Todas-Configurações*, pois não inclui a configuração C_3 , podendo-se concluir que o critério *Todas-Reações-em-Cadeia* não inclui o critério *Todas-Configurações*. Assim sendo, o critério *Todas-Configurações* e o critério *Todas-Reações-em-Cadeia* são incomparáveis.

xv) *Todas-Configurações-História* e *Todas-Reações-em-Cadeia* são incomparáveis.

P_1 é *Todas-Configurações-História*-adequado e P_2 é *Todas-Reações-em-Cadeia*-adequado. Considerando o exemplo da Seção 4.3 (Figura 4.1a), o conjunto P_1^3 não satisfaz o critério *Todas-Reações-em-Cadeia*, pois esse conjunto não inclui os arcos (C_7, C_2) e $(C_6, C_2) \in t_b$ (conjunto de transições com ação contendo evento). Desse modo, o critério *Todas-Configurações-História* não satisfaz o critério *Todas-Reações-em-Cadeia*. Do mesmo modo, o conjunto $P_2 = \{(C_0, C_1, C_4, C_0, C_3, C_8, C_0), (C_0, C_2, C_5, C_7, C_6)\}$ *Todas-Reações-em-Cadeia*-adequado não é adequado ao critério *Todas-Configurações-História* pois não inclui o arco $(C_8, C_2) \in t_h$ (conjunto de

transições com símbolo história). Com isso, o critério *Todas-Reações-em-Cadeia* não satisfaz o critério *Todas-Configurações-História*, podendo-se concluir que esses critérios são incomparáveis.

A hierarquia entre os critérios FCCS da Figura 4.7 considera todos os tipos de statecharts, inclusive aqueles que não satisfazem a propriedade de *reiniciabilidade*: um statechart é reiniciável quando, para cada configuração C_i alcançável a partir da configuração C_0 , há uma seqüência de eventos que retorna à configuração C_0 , reiniciando-o. Por exemplo, quando o statechart possui transição com símbolo história, nem sempre é possível reiniciar o modelo. No exemplo da Figura 4.1a, a partir da configuração C_2 , não existe caminho que retorne à configuração C_0 . Desse modo, é estabelecido o seguinte teorema:

Teorema 2: Os critérios FCCS para especificações reiniciáveis obedecem à hierarquia da Figura 4.8.

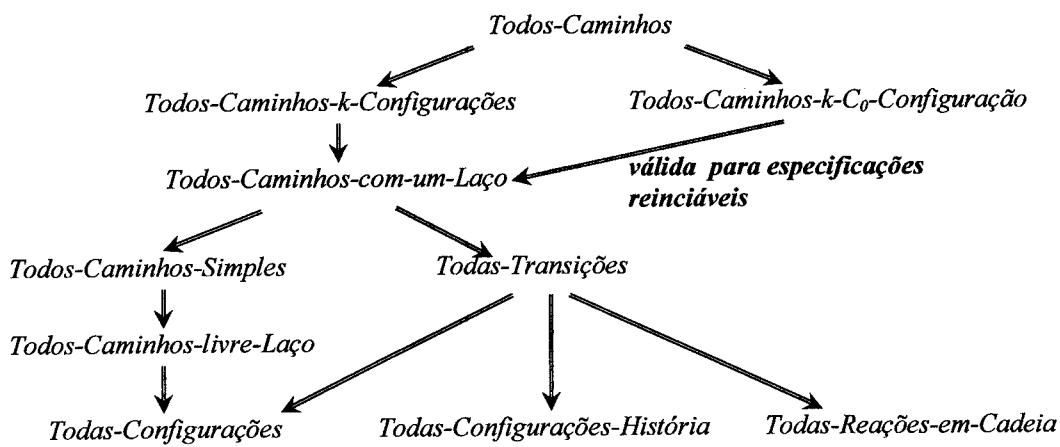


Figura 4.8. Relação Hierárquica dos Critérios FCCS, considerando a Propriedade da *Reiniciabilidade*.

Prova: Basta demonstrar a seguinte relação, dado que as demais já foram provadas:

i) $\text{Todos-Caminhos-}k\text{-}C_0\text{-Configuração} \Rightarrow \text{Todos-Caminhos-com-um-Laço.}$

Seja P_1 *Todos-Caminhos- $k\text{-}C_0\text{-Configuração}-adequado$* e P_2 *Todos-Caminhos-com-um-Laço-adequado*. Para todo caminho $m \in P_1$ $C_0, C_i C_j \dots C_b C_n C_0$, o subcaminho $C_i C_j \dots C_b C_n$ possui pelo menos um laço, caracterizado pelo arco (C_i, C_i) , de forma que existe pelo menos um caminho $m \in P_1$ tal que m inclui p , para todo $p \in P_2$. Com isso p é incluído em P_1 , de forma que P_1 também é *Todos-Caminhos-com-um-Laço-adequado*, ou seja, o critério *Todos-Caminhos- $k\text{-}C_0\text{-Configuração}$* inclui o critério *Todos-Caminhos-com-um-Laço*. Por outro lado, P_2 não satisfaz o critério *Todos-Caminhos- $k\text{-}C_0\text{-Configuração}$* porque seria necessário um caminho contendo k_1 repetições de uma configuração C_i , $k_1 > k$, sendo $C_i \neq C_0$. Esse caminho não está necessariamente incluído em P_2 . Dessa forma, o critério *Todos-Caminhos-com-um-Laço* não inclui o critério *Todos-Caminhos- $k\text{-}C_0\text{-Configuração}$* , podendo-se concluir que o critério *Todos-Caminhos- $k\text{-}C_0\text{-Configuração}$* inclui estritamente o critério *Todos-Caminhos-com-um-Laço*.

É importante ressaltar que os resultados apresentados pressupõe que todos os caminhos na árvore de alcançabilidade são executáveis. Entretanto, a não executabilidade, que é uma questão indecidível, pode alterar a relação de inclusão, a exemplo do que ocorre no nível de programas (Maldonado, 1991). A relação de inclusão deve ser revista considerando esse aspecto.

4.5.2. Relação de Inclusão dos Critérios FCCE

A relação de inclusão entre os critérios de cobertura para especificações em Estelle é dada pelos seguintes teoremas:

Teorema 3: Os critérios de cobertura para especificações em Estelle obedecem à hierarquia da Figura 4.9.

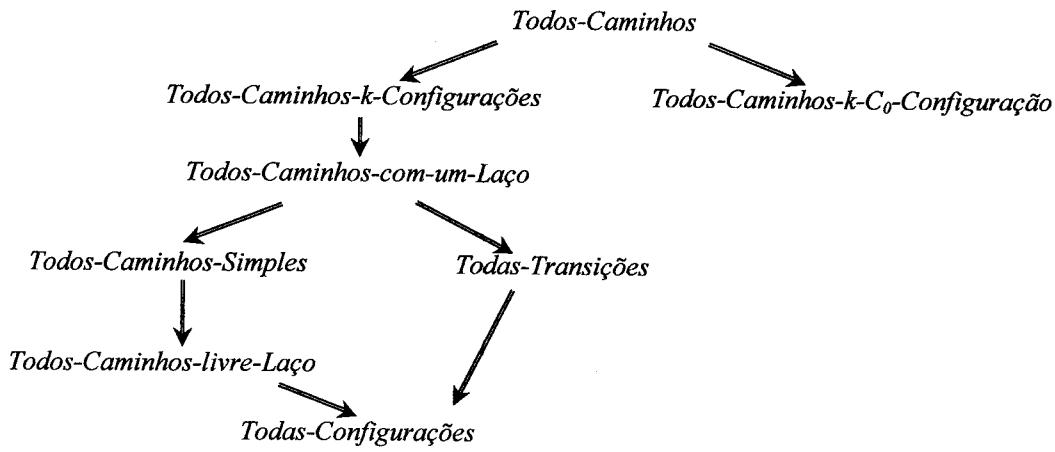


Figura 4.9. Relação Hierárquica dos Critérios FCCE.

Prova: Dado que a relação de inclusão é transitiva, basta considerar as relações justificadas para o Teorema1:

- i) Todos-Caminhos \Rightarrow Todos-Caminhos-k-Configurações.
- ii) Todos-Caminhos-k-Configurações \Rightarrow Todos-Caminhos-com-um-Laço.
- iii) Todos-Caminhos-com-um-Laço \Rightarrow Todos-Caminhos-Simples.
- iv) Todos-Caminhos-Simples \Rightarrow Todos-Caminhos-livre-Laço.
- v) Todos-Caminhos-livre-Laço \Rightarrow Todas-Configurações.
- vi) Todos-Caminhos-com-um-Laço \Rightarrow Todas-Transições.
- vii) Todas-Transições \Rightarrow Todas-Configurações.
- viii) Todos-Caminhos \Rightarrow Todos-Caminhos-k-C₀-Configuração.

- ix) Todos-Caminhos-k-Configurações e Todos-Caminhos-k-C₀-Configuração são incomparáveis.
- x) Todos-Caminhos-Simples e Todas-Transições são incomparáveis. Todos-Caminhos-livre-Laço e Todas-Transições são incomparáveis.

Para provar essas relações, o raciocínio é análogo à prova realizada para Statecharts e as provas não serão apresentadas nesta tese.

Teorema 4: Os critérios FCCE para especificações reiniciáveis obedecem à hierarquia da Figura 4.10.

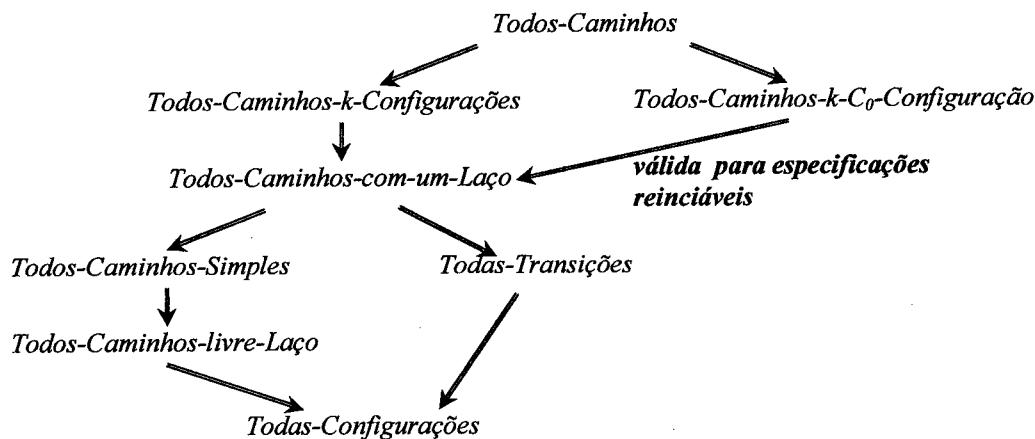


Figura 4.10. Relação Hierárquica dos Critérios FCCE,
considerando a Propriedade da *Reiniciabilidade*.

Prova: Dado que as demais relações já foram provadas, basta demonstrar a seguinte relação:

- i) Todos-Caminhos-k-C₀-Configuração \Rightarrow Todos-Caminhos-com-um-Laço.

Para provar essa relação, o raciocínio também é similar ao raciocínio utilizado para a técnica Statecharts e a prova não será apresentada nesta tese.

4.6. Estratégia Incremental de Aplicação dos Critérios de Cobertura

A relação de inclusão é um aspecto muito importante para avaliação de critérios de teste. Na prática, essas informações podem auxiliar na definição de uma estratégia de teste incremental para a aplicação dos mesmos. Considerando os resultados da seção anterior, uma possível estratégia de aplicação para os critérios de cobertura é:

- **Statecharts:**

Estratégia de Aplicação para os Critérios da FCCS:

1. Aplicar os critérios *Todas-Configurações* e determinar um conjunto de seqüências de teste adequadas;
2. Para Statecharts com história, aplicar o critério *Todas-Configurações-História* e determinar um conjunto de seqüências de teste adequadas;
3. Para Statecharts com *broadcasting*, aplicar o critério *Todas-Reações-em-Cadeia* e determinar um conjunto de seqüências de teste adequadas;
4. Aplicar o critério *Todas-Transições* e determinar um conjunto de seqüências de teste adequadas;
5. Aplicar o critério *Todos-Caminhos-livre-Laço* e determinar um conjunto de seqüências de teste adequadas;
6. Aplicar o critério *Todos-Caminhos-Simples* e determinar um conjunto de seqüências de teste adequadas;
7. Aplicar o critério *Todos-Caminhos-com-um-Laço* e determinar um conjunto de seqüências de teste adequadas;
8. Aplicar o critério *Todos-Caminhos-k-Configurações* e determinar um conjunto de seqüências de teste adequadas;

9. Aplicar o critério *Todos-Caminhos-k-Co-Configuração* e determinar um conjunto de seqüências de teste adequadas;
10. Aplicar o critério *Todos-Caminhos* e determinar um conjunto de seqüências de teste adequadas.

- **Estelle:**

Estratégia de Aplicação para os Critérios da FCCE:

1. Aplicar os critérios *Todas-Configurações* e determinar um conjunto de seqüências de teste adequadas;
2. Aplicar o critério *Todas-Transições* e determinar um conjunto de seqüências de teste adequadas;
3. Aplicar o critério *Todos-Caminhos-livre-Laço* e determinar um conjunto de seqüências de teste adequadas;
4. Aplicar o critério *Todos-Caminhos-Simples* e determinar um conjunto de seqüências de teste adequadas;
5. Aplicar o critério *Todos-Caminhos-com-um-Laço* e determinar um conjunto de seqüências de teste adequadas;
6. Aplicar o critério *Todos-Caminhos-k-Configurações* e determinar um conjunto de seqüências de teste adequadas;
7. Aplicar o critério *Todos-Caminhos-k-Co-Configuração* e determinar um conjunto de seqüências de teste adequadas;
8. Aplicar o critério *Todos-Caminhos* e determinar um conjunto de seqüências de teste adequadas.

Aplicando essas estratégias de teste, o conjunto de seqüências de teste irá, inicialmente, satisfazer critérios de cobertura mais fracos para identificar erros, como

os critérios *Todas-Transições* e *Todas-Configurações*, os quais são mais fáceis e baratos de serem satisfeitos, conforme demonstrado por Rapps e Weyuker (1985). A partir disso, se erros não forem encontrados, critérios de cobertura mais fortes como *Todos-Caminhos-livre-Laço* e *Todos-Caminhos-Simples*, podem ser aplicados, melhorando o conjunto de seqüências de teste. Dependendo das restrições de tempo e custo para realização dos testes, pode-se optar por aplicar critérios mais custosos, os quais geram um número grande de requisitos de teste, como *Todos-Caminhos-com-um-Laço*, *Todos-Caminhos-k-Configurações*, *Todos-Caminhos-k-C₀-Configuração* e *Todos-Caminhos*. Esses critérios podem ser considerados também para testar partes pequenas e críticas da especificação. Isso pode ser feito selecionado somente o(s) componente(s) crítico(s) da especificação para aplicação dos critérios de teste.

Considerando especificações em Statecharts, quando se deseja testar características específicas, como história e *broadcasting*, os seguintes critérios são adequados e também possuem um baixo custo para geração de seqüências de teste: *Todas-Configurações-História* e *Todas-Reações-em-Cadeia*.

Aplicando essas estratégias de teste, o conjunto de seqüências de teste pode ser incrementalmente melhorado, utilizando informações da relação de inclusão dos critérios de cobertura e respeitando restrições de tempo e custo, caso existam.

4.7. Aspectos para Implementação dos Critérios de Cobertura

Para que os critérios de cobertura sejam aplicados eficientemente é fundamental uma ferramenta de apoio. Nesta seção são apresentadas algumas ponderações sobre os aspectos de implementação da FCCS e da FCCE.

Os critérios de cobertura têm como objetivo auxiliar no processo de validação de especificações em Statecharts e em Estelle, fornecendo informações sobre a cobertura do fluxo de controle da especificação e também complementando e quantificando outras atividades de validação, como por exemplo, a atividade de simulação. Nesse sentido, uma ferramenta que automatize esses critérios de teste deve possuir as seguintes características básicas:

- permitir a derivação dos requisitos de teste dos critérios de cobertura a partir da árvore de alcançabilidade da especificação;
- possibilitar a geração de seqüências de teste adequadas por construção aos critérios de cobertura;
- permitir a avaliação de seqüências de teste geradas manualmente, pela simulação, ou por outros critérios de teste, como por exemplo, seqüências de teste geradas pelo Teste de Mutação;
- possibilitar a avaliação da cobertura de seqüências de teste geradas pela simulação em relação aos critérios de cobertura de maneira simultânea, fornecendo a cobertura obtida à medida que a simulação evolui.

Alguns aspectos já se encontram implementados, como por exemplo, a derivação dos requisitos de teste a partir da árvore de alcançabilidade e a geração de seqüências de teste adequadas por construção (Seção 4.3).

Com relação à FCCS, a automatização desses critérios pode ser vinculada ao ambiente StatSim. Como apresentado no Capítulo 2, o ambiente StatSim permite a edição e simulação de Statecharts, tanto na forma textual como na forma gráfica. É possível também a análise de propriedades da especificação através da árvore de alcançabilidade. A simulação pode ser realizada interativamente, em batch, de forma

programada ou exaustivamente. O simulador permite ao usuário acompanhar as ativações do modelo de forma animada.

Considerando a funcionalidade do ambiente StatSim, a automatização da FCCS utiliza como entrada a árvore de alcançabilidade e gera os requisitos de teste e seqüências de teste (adequadas por construção). O testador pode utilizar essas informações para verificar se a especificação satisfaz os requisitos do usuário e pode também utilizar essas informações como entrada para realizar a simulação da especificação, dessa forma, permitindo que o testador possa avaliar os resultados da simulação em relação à cobertura dos critérios da FCCS à medida que a simulação evolui.

Em relação à técnica Estelle, para a automatização da FCCE dois aspectos são necessários:

- uma ferramenta que construa, a partir da especificação em Estelle, sua árvore de alcançabilidade, conforme descrito na Seção 4.3.2. Para implementação podem ser considerados como base os algoritmos definidos para construção da árvore de alcançabilidade para Statecharts.
- uma ferramenta para simulação de especificações em Estelle, de modo a possibilitar que esses critérios possam ser utilizados para complementar essa atividade. Nesse processo, o simulador da ferramenta *EDT* pode ser considerado de forma que esses critérios possam ser aplicados de forma análoga à FCCS.

4.8. Considerações Finais

Este capítulo apresentou a definição dos Critérios de Cobertura para validação de especificações baseadas em Statecharts (FCCS) (Souza et al., 2000b) e em Estelle (FCCE). Esses critérios podem ser empregados tanto para construção de seqüências de teste como para complementar outras formas de validação, como por exemplo, a atividade de simulação. De acordo com o conhecimento da autora, este é o primeiro trabalho que procura utilizar, no contexto de Statecharts e de Estelle, a árvore de alcançabilidade para auxiliar na geração de seqüências de teste. A utilização da árvore de alcançabilidade para aplicação dos critérios de cobertura tornou-se viável devido ao emprego de técnicas de redução para diminuir a explosão de estados da árvore.

A FCCS e a FCCE são compostas por critérios que focalizam os aspectos de Fluxo de Controle da especificação representados pela árvore de alcançabilidade. A FCCS apresenta também critérios de teste para validar aspectos intrínsecos da técnica Statecharts, como paralelismo, história e *broadcasting*.

Estratégias de teste para aplicação dos critérios de cobertura foram apresentadas demonstrando que é possível aplicar esses critérios de maneira incremental: pode-se considerar inicialmente critérios de teste mais fracos e que apresentam menor custo de aplicação (em termos de número de requisitos de teste) e, na seqüência, considerar critérios mais fortes porém mais caros, de acordo com as restrições de tempo e recursos para a atividade de teste.

É importante observar que o conjunto de seqüências de teste adequado para testar a especificação pode ser empregado para a condução dos testes de

conformidade da implementação. Nesse sentido, pretende-se investigar o relacionamento entre esses níveis de abstração: especificação e implementação.

Como trabalhos futuros nessa linha de pesquisa, pretende-se estender os critérios de cobertura definindo critérios de Fluxo de Dados, na mesma linha de pesquisa de Ural e Yang (1991) e Yang et al. (1998). Isto requer que sejam adicionadas informações sobre o Fluxo de Dados na árvore de alcançabilidade. Além disso, pretende-se conduzir estudos empíricos para avaliar o custo e os benefícios dos critérios de cobertura (como eficácia em revelar erros da especificação) e também estudos comparativos entre esses critérios de cobertura e o Teste de Mutação no contexto de Statecharts (Fabbri, 1996; Fabbri et al., 1999a) e no contexto de Estelle (Souza et al., 2000a). Para isso, é fundamental a implementação de uma ferramenta de apoio à aplicação desses critérios de teste.

Nos capítulos a seguir, são feitos estudos dos critérios de teste propostos nesta tese. No Capítulo 5 são apresentados os resultados de um estudo comparativo entre a FCCS e o Teste de Mutação para Statecharts. No Capítulo 6, são apresentados os resultados de um estudo comparativo entre o Teste de Mutação para Estelle e a FCCE. Esses estudos ilustram a aplicação desses critérios de teste e avaliam o custo de aplicação e a utilização desses critérios para apoiar a atividade de simulação de especificações.

Capítulo 5. Avaliação dos Critérios de Teste para Statecharts: Um Estudo de Caso

5.1. Considerações Iniciais

Neste capítulo é realizada uma análise entre a Família de Critérios de Cobertura para Statecharts (Souza et al., 2000b) e o Teste de Mutação para Statecharts (Fabbri, 1996; Fabbri et al., 1999a). São geradas seqüências de teste adequadas aos critérios e então é analisado o *strength*: o quanto que as seqüências de teste adequadas ao Teste de Mutação (AM-adequadas) cobrem os requisitos dos critérios FCCS, e vice-versa, ou seja, o quanto que as seqüências adequadas aos critérios de cobertura (FCCS-adequadas) cobrem os requisitos do Teste de Mutação.

Os critérios FCCS e Teste de Mutação são utilizados também para avaliar a cobertura de seqüências de teste geradas pela Simulação Programada do ambiente StatSim, ilustrando, dessa forma, sua aplicação para complementar a atividade de simulação.

Para a realização deste estudo, utiliza-se a especificação do Sistema de Controle de Passagem em Nível. Barnard (1998) utiliza esse exemplo para ilustrar sua metodologia *COMX* para descrição formal de sistemas. Neste capítulo, esse exemplo foi modelado por meio de Statecharts utilizando a mesma nomenclatura empregada

por Barnard (1998), mantendo-se o erro inicial. Ilustra-se como os critérios de teste para Statecharts (FCCS e Teste de Mutação) podem auxiliar na identificação desse erro.

5.2. Estudo de Caso: Sistema de Controle de Passagem em Nível

O Sistema de Controle de Passagem em Nível é composto de duas estradas de ferro, cada uma em um sentido, duas cancelas e dois conjuntos de luzes e de motores para abaixar e levantar as cancelas (Figura 5.1). Nos trilhos dos trens existem sensores que monitoram a posição dos trens. Um controlador recebe sinais dos sensores e garante que cancelas, motores e luzes funcionam adequadamente: as cancelas devem abaixar quando o trem estiver passando e levantar quando o trem terminar de passar. Os sensores dos trilhos funcionam da seguinte forma (Barnard, 1998):

- Sensor 1: Se as cancelas estiverem abaixadas (*down*), deixa-as abaixadas, no caso de estar vindo um outro trem;
- Sensor 2: Abaixa as cancelas (se não estiverem abaixadas) dentro de n segundos;
- Sensor 3: Levanta as cancelas (*up*), a menos que o Sensor 1 da outra linha tenha sido disparado.

Abaixar as cancelas (*down*) significa “acender as luzes amarelas por x segundos e então acender as luzes vermelhas e ativar os motores para abaixar as cancelas”.

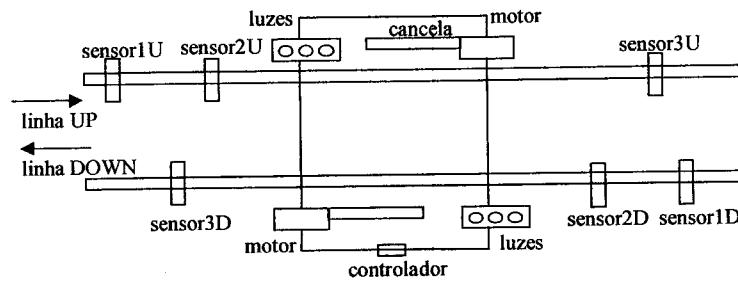


Figura 5.1. Diagrama do Sistema de Passagem em Nível (Barnard, 1998).

De acordo a descrição do sistema, o statechart da Figura 5.2 foi construído. O statechart é composto de 5 componentes ortogonais: *Controlador*, *Luzes*, *Cancelas*, *Trem_Linha_Up* (trem que atravessa a linha Up) e *Trem_Linha_Down* (trem que atravessa a linha Down).

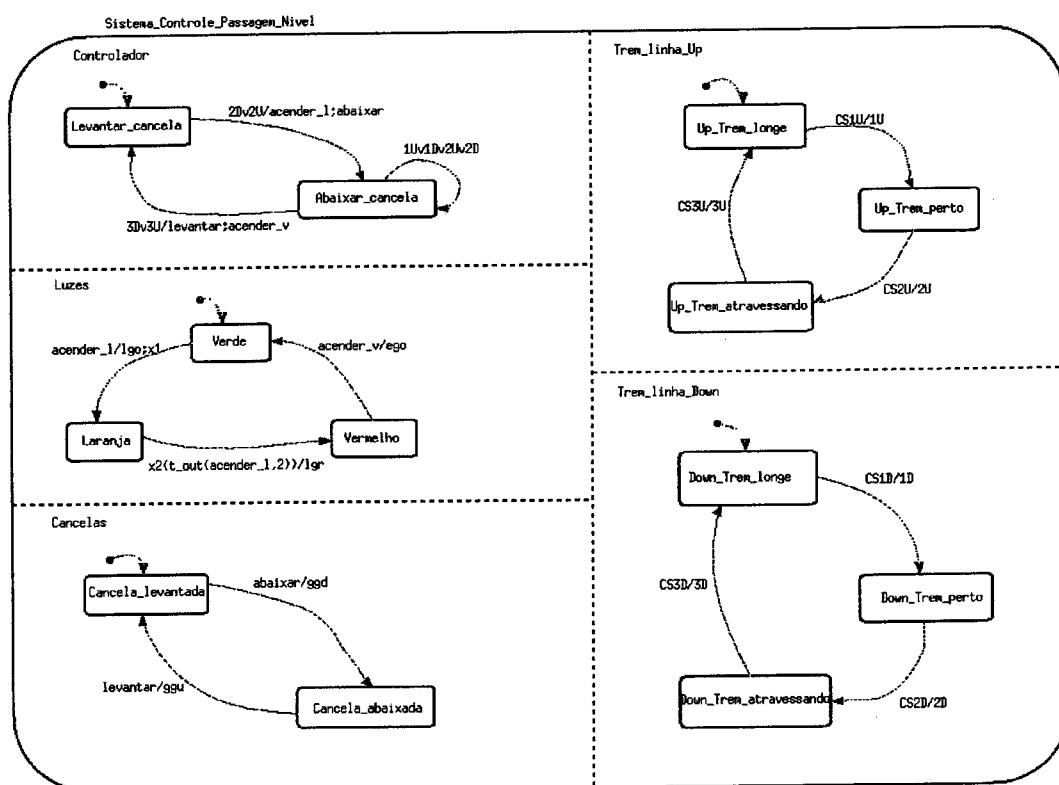


Figura 5.2. Statechart do Sistema de Passagem em Nível.

Para fins de simplicidade, foram selecionados os componentes *Cancelas*, *Trem_Linha_Up* e *Trem_Linha_Down* para construção da árvore de alcançabilidade.

Na Figura 5.3 é apresentada a árvore de alcançabilidade construída a partir dos componentes selecionados.

$$C = [(Levantar-cancela, Abaixar-cancela), (Up-Trem-longe, Up-Trem-perto, Up-Trem-atravessando), (Down-Trem-longe, Down-Trem-perto, Down-Trem-atravessando)]$$

$$\begin{array}{lll} C_0 = [(1,0), (1,0,0), (1,0,0)] & C_5 = [(0,1), (0,1,0), (0,0,1)] & C_9 = [(1,0), (1,0,0), (0,0,1)] \\ C_1 = [(1,0), (0,1,0), (0,1,0)] & C_6 = [(0,1), (0,0,1), (0,0,1)] & C_{10} = [(1,0), (0,0,1), (1,0,0)] \\ C_2 = [(1,0), (0,1,0), (1,0,0)] & C_7 = [(0,1), (0,0,1), (1,0,0)] & C_{11} = [(1,0), (0,0,1), (0,1,0)] \\ C_3 = [(1,0), (1,0,0), (0,1,0)] & C_8 = [(0,1), (1,0,0), (0,0,1)] & C_{12} = [(1,0), (0,1,0), (0,0,1)] \\ C_4 = [(0,1), (0,0,1), (0,1,0)] \end{array}$$

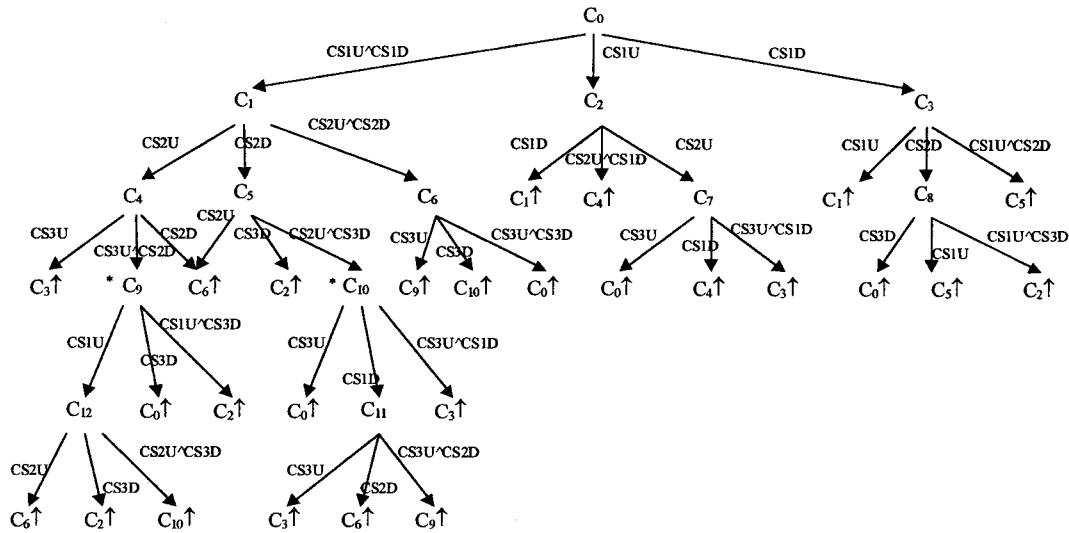


Figura 5.3. Árvore de Alcançabilidade do Sistema de Passagem em Nível.

Conforme apontado por Barnard (1998), de posse de informações semânticas da especificação, é possível observar pela árvore de alcançabilidade que configurações não desejadas são alcançadas (marcadas com * na árvore de alcançabilidade): $C_9 = (Levantar-cancela, Up-Trem-Longe, Down-Trem-atravessando)$ e $C_{10} = (Levantar-cancela, Up-Trem-atravessando, Down-Trem-Longe)$. Essas configurações significam que a cancela está levantada e que um dos trens ainda está passando. Essa situação ilustra a identificação de um erro no statechart a partir da árvore de alcançabilidade, o qual poderia não ser facilmente notado pela simples observação do statechart. Embora esse erro possa ser identificado pela árvore de alcançabilidade,

isso nem sempre é possível e fácil de ser feito considerando somente a árvore e a análise de propriedades.

Conforme descrito no Capítulo 2, a partir da especificação do sistema é possível a aplicação de critérios de teste funcionais. O critério funcional Particionamento em Classes de Equivalência é utilizado para exemplificar a geração de casos de teste a partir da descrição do sistema, dado que esse critério é um dos critérios funcionais mais utilizados. Para a geração dos casos de teste são identificadas as condições de entrada do programa e definidas as classes de equivalência válidas e inválidas. O objetivo é gerarem-se casos de teste que cubram essas classes de equivalência. Na Tabela 5.1 são apresentadas as condições e as classes de equivalência definidas para o exemplo utilizado. As condições são geradas a partir das entradas que são os trens movimentando-se pelos trilhos (sobre os sensores). É necessário gerar, no mínimo, uma seqüência de teste que satisfaz todas as classes válidas e uma seqüência de teste para cada classe inválida. Assim, o conjunto de seqüências foi gerado:

$$\begin{aligned} T_0 = \{ & (\text{sensor1U} \wedge \text{sensor1D}, \text{sensor2U} \wedge \text{sensor2D}, \text{sensor3U} \wedge \text{sensor3D}), \\ & (\text{sensor1U}, \text{sensor1D}, \text{sensor2U} \wedge \text{sensor2D}, \text{sensor1U}, \text{sensor2U}), \\ & (\text{sensor1U}, \text{sensor1D}, \text{sensor2U}, \text{sensor1D}, \text{sensor2D}, \text{sensor1D}, \text{sensor1U}) \}. \end{aligned}$$

T_0 foi aplicado ao statechart e o erro existente não foi revelado, pois para isso é necessário que o Sensor2 tenha sido ativado nas duas linhas (CS2D e CS2U) e, em uma das linhas, o Sensor3 (CS3U ou CS3D) seja, posteriormente, ativado. Entretanto, o erro até poderia ser revelado por esse critério, o que é dependente das seqüências de teste escolhidas para cobrir as classes de equivalência.

Esse critério de teste apresenta limitações, pois não se pode garantir que partes essenciais ou críticas do software foram executadas e, além disso, torna-se difícil quantificar a atividade de teste. Outra limitação desse critério é que não são

consideradas combinações de condições de entrada para geração das seqüências de teste, conforme é feito pelo critério Grafos de Causas e Efeitos, o que poderia ter propiciado a identificação do erro.

Tabela 5.1. Classes de Equivalência para o Sistema de Controle de Passagem em Nível.

Condições de Entrada	Classe Válida	Classe Inválida
Trem passando pelo Sensor 1 na linha Up	sim (1)	-
Trem passando pelo Sensor 2 na linha Up	sim (2)	-
Trem passando pelo Sensor 3 na linha Up	sim (3)	-
Trem passando pelo Sensor 1 na linha Down	sim (4)	-
Trem passando pelo Sensor 2 na linha Down	sim (5)	-
Trem passando pelo Sensor 3 na linha Down	sim (6)	-
Número (n) de trens na linha Up	n = 1 (7)	n > 1 (8)
Número (n) de trens na linha Down	n = 1 (9)	n > 1 (10)

Para corrigir o erro, uma possível solução é apresentada na Figura 5.4. Neste statechart é definida uma variável x que controla o número de trens que estão passando. A variável x é incrementada quando o trem passa do estado *longe* (*Up-Trem-longe* ou *Down-Trem-longe*) para *perto* (*Up-Trem-perto* ou *Down-Trem-perto*) e é decrementada quando o trem passa do estado *atravessando* (*Up-Trem-atravessando* ou *Down-Trem-atravessando*) para o estado *longe*. No componente *Controlador*, as cancelas serão levantadas quando nenhum trem estiver passando, ou seja, quando o valor de x for igual a zero.

Nas próximas seções, são apresentados os resultados da aplicação dos critérios de teste FCCS e Teste de Mutação no statechart com erro (Figura 5.2). É observado que esses critérios de teste contribuem na geração de seqüências de teste capazes de revelar o erro existente no statechart.

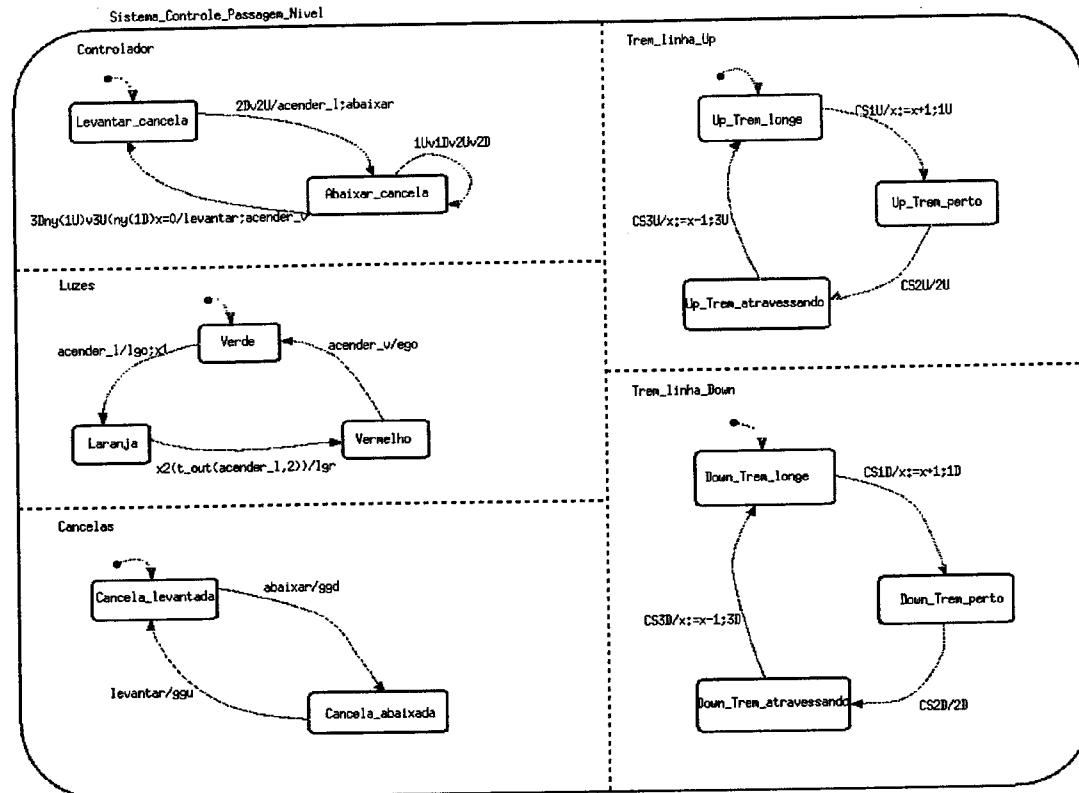


Figura 5.4. Statechart do Sistema de Passagem em Nível Corrigido.

5.3. Geração de Seqüências de Teste Adequadas ao Teste de Mutação

O Teste de Mutação para Statecharts (Fabbri, 1996) é composto por 37 operadores de mutação, os quais estão divididos em três classes, de acordo com as características do modelo que são validadas:

1. Conjunto de Operadores de Mutação para Máquinas de Estados Finitos (MEF): formado por 8 operadores de mutação que abordam aspectos de Máquinas de Estados Finitos no contexto de especificações baseadas em Statecharts.
2. Conjunto de Operadores de Mutação para Máquinas de Estados Finitos Estendidas (MEFE): formado por 11 operadores que tratam aspectos

relacionados a Máquinas de Estados Finitos Estendidas no contexto de Statecharts, ou seja, aspectos relativos ao uso de variáveis e condições.

3. Conjuntos de Operadores de Mutação para Aspectos Específicos da Técnica Statecharts (ST): formado por 17 operadores de mutação que tratam os aspectos intrínsecos da técnica Statecharts, como história, *broadcasting* e paralelismo.

Utilizando a ferramenta *Proteum-RS/ST* (Sugeta, 1999; Fabbri et al., 1999b), o critério Teste de Mutação foi aplicado ao statechart da Figura 5.2 e os resultados obtidos são apresentados na Tabela 5.2. Nessa tabela é apresentado o número de mutantes gerados para cada operador de mutação, número de mutantes equivalentes e anômalos. Na versão atual da ferramenta, os mutantes que apresentam não determinismo no seu comportamento não são tratados e são marcados como mutantes anômalos.

Na Tabela 5.3 são apresentados o total de mutantes gerados para cada classe de mutação (MEF, MEFE e ST) e o total de seqüências de teste adequadas a cada classe. As seqüências de teste foram geradas pelo testador da seguinte forma: um conjunto de seqüências de teste inicial foi criado, o qual foi elaborado sem utilizar critérios de teste, somente a partir da especificação. Os mutantes vivos foram executados com esse conjunto. Como alguns mutantes permaneceram vivos, novas seqüências de teste foram geradas a partir da análise desses mutantes. Para cada seqüência de teste inserida, todos os mutantes vivos eram executados e repetiu-se o processo até restarem somente mutantes mortos, anômalos e equivalentes.

Para a mutação no aspecto de MEFE do statechart foram gerados poucos mutantes porque a especificação não utiliza variáveis e também não possui condições

associadas às transições. Na Tabela 5.4 são apresentadas as seqüências de teste adequadas às classes de mutação para MEF, MEFE e ST, respectivamente.

Tabela 5.2. Total de Mutantes Gerados, Equivalentes e Anômalos para as Classes de Operadores de Mutação para o Statechart da Figura 5.2.

Operadores de Mutação		Número de Mutantes		
		Gerados	Equivalentes	Anômalos
MEF	1. alteração do estado default	08	-	-
	2. arco faltando	14	-	-
	3. evento faltando	19	02	-
	4. evento extra	56	12	08
	5. evento trocado	28	-	-
	6. destino trocado	23	01	-
	7. ação faltando	10	02	-
	8. ação trocada	02	-	-
MEFE	1. exclusão de expressão	0	-	-
	2. negação de expressão booleana	0	-	-
	3. troca associativa de termos	0	-	-
	4. troca operador aritmético por operador aritmético	0	-	-
	5. troca operador relacional por operador relacional	0	-	-
	6. troca operador lógico por operador lógico	05	03	-
	7. negação lógica	0	-	-
	8. troca variável por variável	0	-	-
	9. troca variável por constante	0	-	-
	10. troca constante por constante requeridas	03	02	01
	11. troca constante por variável escalar	0	-	-
ST	1. desassocia história da transição	0	-	-
	2. troca transição associada ao símbolo história	0	-	-
	3. exclui história do estado	0	-	-
	4. troca h por h^*	0	-	-
	5. troca h^* por h	0	-	-
	6. associa h ao estado	0	-	-
	7. associa h^* ao estado	0	-	-
	8. exclui condição in(s)	01	01	-
	9. troca estado da condição in(s)	02	-	-
	10. exclui condição not-yet(e)	0	-	-
	11. troca evento da condição not-yet(e)	0	-	-
	12. exclui evento exit(s)	0	-	-
	13. troca estado do evento exit(s)	0	-	-
	14. exclui evento entered(s)	0	-	-
	15. troca estado do evento entered(s)	0	-	-
	16. altera origem do broadcasting	20	02	08
	17. altera destino do broadcasting	18	3	2

Com relação ao erro existente, observa-se que algumas seqüências de teste geradas são *error-revealing*, ou seja, quando são executadas com a especificação original revelam o erro existente. As seqüências de teste *error-revealing* estão destacadas na Tabela 5.4. Entretanto, nenhum dos mutantes gerados é *error-*

revealing. Um mutante M , gerado a partir de um programa P , é dito *error-revealing* se, para qualquer caso de teste t , tal que $P^*(t) \neq M^*(t)$, pudermos concluir que $P^*(t)$ não está de acordo com o resultado esperado, ou seja, revela o erro. Por exemplo, a seqüência número 20 da classe MEF, é capaz de distinguir 56 mutantes. Desses mutantes, o mutante 13, gerado pelo operador *Arco Faltando*, é morto também pela seqüência de teste número 1 (classe MEF), que não é *error-revealing*.

Ilustrou-se assim como os critérios Teste de Mutação auxilia no teste de especificação.

Tabela 5.3. Tamanho dos Conjuntos de Seqüências de Teste Adequadas às Classes de Mutação para o Statechart da Figura 5.2.

Classes de Mutação	Total de Requisitos de Teste (mutantes)	Tamanho dos Conjuntos AM-Adequados
MEF	160	T_{MEF}
MEFE	08	T_{MEFE}
ST	41	T_{ST}
TOTAL	209	36

Tabela 5.4. Conjuntos de Seqüências de Teste Adequados ao Teste de Mutação para Statecharts: MEF-Adequado, MEFE-Adequado e ST-Adequado.

Classes de Mutação	Seqüências de Teste	
MEF	# 1: CS1U, CS2U # 2: CS1U, CS2U, CS3U # 3: CS1D, CS2D # 4: CS1D, CS2D, CS3D # 5: CS1U ^ CS1D, CS2U, CS2D # 6: CS1U # 7: CS1D # 8: CS2U # 9: CS2D # 10: CS3U # 11: CS1U, CS1U # 12: CS1U, CS3U # 13: CS1U, CS2U, CS1U # 14: CS1U, CS2U, CS2U # 15: CS3D # 16: CS1D, CS1D # 17: CS1D, CS3D	# 18: CS1D, CS2D, CS1D # 19: CS1D, CS2D, CS2D # 20: CS1U ^ CS1D, CS2U ^ CS2D, CS3U, CS3D # 21: CS1U ^ CS1D, CS2U ^ CS2D, CS3D, CS3U # 22: CS1U, CS2U, CS1D # 23: CS1U, CS1D, CS2D, CS2U # 24: CS1D, CS2D, CS1U
MEFE	# 1: CS1U ^ CS1D, CS2U, CS2D, CS3U	# 2: CS1U, CS2U
ST	# 1: CS1U ^ CS1D, CS2U # 2: CS1U, CS2U, CS3U # 3: CS1U # 4: CS1D # 5: CS1U ^ CS1D, CS2U, CS2D, CS3D, CS3U	# 6: CS1U ^ CS1D, CS2D, CS2U, CS3U, CS3D # 7: CS1U ^ CS1D, CS2D, CS2U # 8: CS1D, CS2D, CS1U # 9: CS1U ^ CS1D, CS2U, CS2D # 10: CS1U, CS2U, CS1D

5.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCS

Nesta seção são apresentados os resultados da aplicação da Família de Critérios de Cobertura para Statecharts na especificação exemplo da Figura 5.2. Os critérios de cobertura foram utilizados para geração de seqüências de teste adequadas, por construção, a esses critérios.

Os procedimentos apresentados no Capítulo 4 são utilizados para obtenção dos requisitos de teste dos critérios de cobertura e geração das seqüências adequadas por construção para esses critérios.

Na Tabela 5.5 é apresentado o total de requisitos e de seqüência de teste gerados. O critério *Todas-Transições* e *Todas-Reações-em-Cadeia* geram o mesmo número de requisitos de teste pois em todas as transições (dos componentes selecionados) ocorre *broadcasting*. Considerando que cada seqüência de eventos irá percorrer um caminho distinto, os critérios de cobertura que estabelecem caminhos (*Todos-Caminhos-com-um-Laço*, *Todos-Caminhos-Simples* e *Todos-Caminhos-livre-Laço*) requerem uma seqüência de teste para cada requisito de teste. Os critérios *Todos-Caminhos-k-Configurações* e *Todos-Caminhos-k-Co-Configurações* não foram considerados porque geraram um número muito elevado de requisitos de teste, apresentando um alto custo computacional para sua aplicação.

Na Tabela 5.6 é apresentada uma relação parcial dos requisitos de teste para os critérios de cobertura e na Tabela 5.7 as seqüências de teste adequadas para esses requisitos são apresentadas.

Tabela 5.5. Total de Requisitos de Teste e de Seqüências de Teste (geradas por construção) para os Critérios FCCS.

Critérios FCCS	Requisitos de Teste	Seqüências de Teste
<i>Todos-Caminhos</i>	infinito	—
<i>Todos-Caminhos-k-C₀-Configurações</i>	não aplicado	—
<i>Todos-Caminhos-k-Configurações</i>	não aplicado	—
<i>Todos-Caminhos-com-um-Laço</i>	5142	5142
<i>Todos-Caminhos-Simples</i>	1059	1059
<i>Todos-Caminhos-livre-Laço</i>	476	476
<i>Todas-Transições</i>	39	27
<i>Todas-Configurações</i>	13	05
<i>Todas-Configurações-História</i>	não aplicável	—
<i>Todas-Reações-em-Cadeia</i>	39	27
TOTAL	6768	6736

Durante a geração das seqüências de teste adequadas por construção, observou-se também a capacidade dos critérios de cobertura de revelar o erro existente na especificação. O erro é caracterizado pela existência das configurações C_9 e C_{10} , as quais representam configurações de estados não desejadas. Para todos os critérios de cobertura aplicados, essas configurações são percorridas por pelo menos um requisito de teste. Isso indica que os critérios FCCS conseguem auxiliar na identificação do erro existente na especificação. Considere, por exemplo, um requisito de teste do critério *Todos-Caminhos-Simples*: $C_0, C_1, C_4, C_3, C_8, C_5, C_6, C_9, C_2, C_7, C_0$. Esse requisito gera a seqüência de teste: $1u^{\wedge}Id, 2u, 3u, 2d, 1u, 2u, 3u, 3d^{\wedge}1u, 2u, 3u$. A transição em negrito não está de acordo com a especificação, pois o Sensor 3 só poderá levantar o portão se o Sensor 1 da outra linha não tiver sido ativado antes. Ou seja, analisando os requisitos de teste com o objetivo de gerar seqüências de teste observa-se uma situação não desejada pela especificação e assim é possível identificar esse erro.

Tabela 5.6. Parte dos Requisitos de Teste dos Critérios FCCS.

Critérios FCCS	Requisitos de Teste		
<i>Todos-Caminhos-com-um-laço</i>	1 - c0 - c1 - c4 - c3 - c1 - c5 - c6 - c9 - c12 - c2 - c7 2 - c0 - c1 - c4 - c3 - c1 - c5 - c6 - c9 - c12 - c10 - c11 3 - c0 - c1 - c4 - c3 - c1 - c5 - c6 - c9 - c2 - c7 4 - c0 - c1 - c4 - c3 - c1 - c5 - c6 - c10 - c11 - c9 - c12 - c2 - c7 5 - c0 - c1 - c4 - c3 - c1 - c5 - c6 - c10 - c11 - c9 - c2 - c7 6 - c0 - c1 - c4 - c3 - c1 - c5 - c2 - c7 7 - c0 - c1 - c4 - c3 - c1 - c5 - c10 - c11 - c6 - c9 - c12 - c2 - c7 8 - c0 - c1 - c4 - c3 - c1 - c5 - c10 - c11 - c6 - c9 - c2 - c7		
<i>Todos-Caminhos-Simples</i>	1 - c0 - c1 - c4 - c3 - c8 - c0 2 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c12 - c2 - c7 - c0 3 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c12 - c10 - c0 4 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c12 - c10 - c11 5 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c0 6 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c2 - c7 - c0 7 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c10 - c0 8 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c10 - c11 - c9 - c12 - c2 - c7 - c0		
<i>Todos-Caminhos-livre-Laço</i>	1 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c12 - c2 - c7 2 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c12 - c10 - c11 3 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c9 - c2 - c7 4 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c10 - c11 - c9 - c12 - c2 - c7 5 - c0 - c1 - c4 - c3 - c8 - c5 - c6 - c10 - c11 - c9 - c2 - c7 6 - c0 - c1 - c4 - c3 - c8 - c5 - c2 - c7 7 - c0 - c1 - c4 - c3 - c8 - c5 - c10 - c11 - c6 - c9 - c12 - c2 - c7 8 - c0 - c1 - c4 - c3 - c8 - c5 - c10 - c11 - c6 - c9 - c2 - c7		
<i>Todas-Transições</i>	1 - (c0,c1) 2 - (c1,c4) 3 - (c0,c2) 4 - (c2,c1) 5 - (c0,c3) 6 - (c3,c1) 7 - (c4,c3) 8 - (c1,c5) 9 - (c5,c6) 10 - (c1,c6) 11 - (c6,c9) 12 - (c2,c4) 13 - (c2,c7)	14 - (c7,c0) 15 - (c3,c8) 16 - (c8,c0) 17 - (c3,c5) 18 - (c4,c9) 19 - (c9,c12) 20 - (c4,c6) 21 - (c5,c2) 22 - (c5,c10) 23 - (c10,c0) 24 - (c6,c10) 25 - (c6,c0) 26 - (c7,c4)	27 - (c7,c3) 28 - (c8,c5) 29 - (c8,c2) 30 - (c12,c6) 31 - (c9,c0) 32 - (c9,c2) 33 - (c10,c11) 34 - (c11,c3) 35 - (c10,c3) 36 - (c12,c2) 37 - (c12,c10) 38 - (c11,c6) 39 - (c11,c9)
<i>Todas-Configurações</i>	c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13		
<i>Todas-Reações-em-Cadeia</i>	1 - (c0,c1) 2 - (c1,c4) 3 - (c0,c2) 4 - (c2,c1) 5 - (c0,c3) 6 - (c3,c1) 7 - (c4,c3) 8 - (c1,c5) 9 - (c5,c6) 10 - (c1,c6) 11 - (c6,c9) 12 - (c2,c4) 13 - (c2,c7)	14 - (c7,c0) 15 - (c3,c8) 16 - (c8,c0) 17 - (c3,c5) 18 - (c4,c9) 19 - (c9,c12) 20 - (c4,c6) 21 - (c5,c2) 22 - (c5,c10) 23 - (c10,c0) 24 - (c6,c10) 25 - (c6,c0) 26 - (c7,c4)	27 - (c7,c3) 28 - (c8,c5) 29 - (c8,c2) 30 - (c12,c6) 31 - (c9,c0) 32 - (c9,c2) 33 - (c10,c11) 34 - (c11,c3) 35 - (c10,c3) 36 - (c12,c2) 37 - (c12,c10) 38 - (c11,c6) 39 - (c11,c9)

Tabela 5.7. Seqüências de Teste Geradas por Construção para os Requisitos de Teste dos Critérios de Cobertura para Statecharts apresentados na Tabela 5.6.

Critérios FCCS	Seqüências de Teste	
<i>Todos-Caminhos-com-um-Laço</i> (T_u)	1 - 1u ^{1d} - 2u - 3u - 1u - 2d - 2u - 3u - 1u - 3d - 2u 2 - 1u ^{1d} - 2u - 3u - 1u - 2d - 2u - 3u - 1u - 3d ^{2u} - 1d 3 - 1u ^{1d} - 2u - 3u - 1u - 2d - 2u - 3u - 3d ^{1u} - 2u 4 - 1u ^{1d} - 2u - 3u - 1u - 2d - 2u - 3d - 1d - 3u ^{2d} - 1u - 3d - 2u 5 - 1u ^{1d} - 2u - 3u - 1u - 2d - 2u - 3d - 1d - 3u ^{2d} - 3d ^{1u} - 2u 6 - 1u ^{1d} - 2u - 3u - 1u - 2d - 3d - 2u 7 - 1u ^{1d} - 2u - 3u - 1u - 2d - 3d ^{2u} - 1d - 2d - 3u - 1u - 3d - 2u 8 - 1u ^{1d} - 2u - 3u - 1u - 2d - 3d ^{2u} - 1d - 2d - 3u - 3d ^{1u} - 2u	
<i>Todos-Caminhos-Simples</i> (T_s)	1 - 1u ^{1d} - 2u - 3u - 2d - 3d 2 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 1u - 3d - 2u - 3u 3 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 1u - 3d ^{2u} - 3u 4 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 1u - 3d ^{2u} - 1d 5 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 3d 6 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 3d ^{1u} - 2u - 3u 7 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3d - 3u 8 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3d - 1d - 3u ^{2d} - 1u - 3d - 2u - 3u	
<i>Todos-Caminhos-livre-Laço</i> (T_l)	1 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 1u - 3d - 2u 2 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 1u - 3d ^{2u} - 1d 3 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3u - 3d ^{1u} - 2u 4 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3d - 1d - 3u ^{2d} - 1u - 3d - 2u 5 - 1u ^{1d} - 2u - 3u - 2d - 1u - 2u - 3d - 1d - 3u ^{2d} - 3d ^{1u} - 2u 6 - 1u ^{1d} - 2u - 3u - 2d - 1u - 3d - 2u 7 - 1u ^{1d} - 2u - 3u - 2d - 1u - 3d ^{2u} - 1d - 2d - 3u - 1u - 3d - 2u 8 - 1u ^{1d} - 2u - 3u - 2d - 1u - 3d ^{2u} - 1d - 2d - 3u - 3d ^{1u} - 2u	
<i>Todas-Transições</i> (T_t)	1 - 1u ^{1d} - 2u - 3u 2 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 2u 3 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 3d 4 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 3d ^{2u}	5 - 1u ^{1d} - 2u - 3u ^{2d} - 3d 6 - 1u ^{1d} - 2u - 3u ^{2d} - 3d ^{1u} 7 - 1u ^{1d} - 2u - 2d 8 - 1u ^{1d} - 2d - 2u
<i>Todas-Configurações</i> (T_c)	1 - 1u ^{1d} - 2u - 3u ^{2d} - 1u 2 - 1u ^{1d} - 2d - 3d ^{2u} - 1d 3 - 1u ^{1d} - 2u ^{2d}	4 - 1u - 2u 5 - 1d - 2d
<i>Todas-Reações-em-Cadeia</i> (T_r)	1 - 1u ^{1d} - 2u - 3u 2 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 2u 3 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 3d 4 - 1u ^{1d} - 2u - 3u ^{2d} - 1u - 3d ^{2u}	5 - 1u ^{1d} - 2u - 3u ^{2d} - 3d 6 - 1u ^{1d} - 2u - 3u ^{2d} - 3d ^{1u} 7 - 1u ^{1d} - 2u - 2d 8 - 1u ^{1d} - 2d - 2u

5.5. Strength dos Critérios Teste de Mutação e FCCS: Um Estudo de Caso

Nesta seção são descritos os resultados da comparação entre a FCCS e o Teste de Mutação para Statecharts. O objetivo desse estudo é analisar o aspecto complementar desses critérios de teste. Para isso, os conjuntos de seqüências AM-adequados (obtidos na Seção 5.3) foram aplicados aos critérios de cobertura e analisou-se qual a porcentagem de cobertura obtida. O mesmo foi feito para os conjuntos de seqüências FCCS-adequados (obtidos na Seção 5.4) em relação ao Teste de Mutação.

Na Tabela 5.8 são apresentados os resultados da aplicação das seqüências FCCS-adequadas em relação ao Teste de Mutação. Para analisar o escore de mutação, foi utilizada a ferramenta *Proteum-RS/ST*, que possibilita a importação de seqüências de teste de outras sessões de teste. De posse de informações sobre mutantes equivalentes e anômalos, analisou-se a porcentagem de mutantes não equivalentes que são distinguidos pelas seqüências FCCS-adequadas. São utilizadas as siglas da Tabela 5.7 para os conjuntos de seqüências FCCS-adequadas. Observa-se que os conjuntos FCCS-adequados foram adequados para as mutações no aspecto de MEFs. O fato de não existirem variáveis e condições na especificação utilizada no estudo de caso não possibilita que essa classe de mutação seja aplicada integralmente. Com isso, faz-se necessário considerar uma especificação contendo variáveis e condições para fazer uma avaliação melhor do *strength* entre os critérios de cobertura e essa classe de mutação.

De uma maneira geral, pode-se observar que os conjuntos FCCS-adequados não conseguiram identificar todos os mutantes não equivalentes das classes de mutação em MEF e ST. Exceção ocorreu com os conjuntos adequados aos critérios *Todos-Caminhos-com-um-Laço* (T_w), *Todas-Transições* (T_t) e *Todas-Reações-em-Cadeia* (T_r), os quais obtiveram escore igual a 1 para a classe de mutação ST. Considerando a aplicação de todas as classes de mutação (*Todas-Mutações*), observa-se que o escore de mutação obtido foi alto, mas assim mesmo, nenhum conjunto FCCS-adequado conseguiu obter escore igual a 1. Este fato *per se* determina que o critério FCCS não inclui o critério Teste de Mutação para Estelle.

Tabela 5.8. Porcentagem de Cobertura das Seqüências FCCS-adequadas em Relação ao Critério Teste de Mutação para Statecharts.

Classes de Mutação	Escore de Mutação para Seqüências Adequadas aos Critérios FCCS					
	T_u	T_s	T_l	T_t	T_c	T_r
MEF	0.85	0.85	0.85	0.91	0.73	0.91
MEFE	1.0	1.0	1.0	1.0	1.0	1.0
ST	1.0	0.84	0.92	1.0	0.76	1.0
Todas-Mutações	0.88	0.85	0.86	0.93	0.73	0.93

Na Tabela 5.9 são apresentados os resultados da aplicação das seqüências AM-adequadas em relação aos critérios de cobertura. A análise de cobertura foi feita manualmente, pois ainda não existe um procedimento implementado para esse fim. Desse modo, para os conjuntos MEF-adequado, MEFE-adequado e ST-adequado são obtidas as configurações de estados atingidas e, de posse dos requisitos de teste de cada critério de cobertura, são analisadas quantas seqüências de teste cobrem esses requisitos. A porcentagem de cobertura é calculada pela equação:

$$\frac{\text{Número de Requisitos executados por } T}{\text{Total de Requisitos}} * 100 \quad (5.1)$$

sendo que, nesse caso, T representa um dos conjuntos adequados ao Teste de Mutação: MEF-adequado, MEFE-adequado e ST-adequado.

Devido ao rigor dos critérios de cobertura, os quais exigem que seqüências de teste percorram caminhos específicos na árvore de alcançabilidade, a cobertura obtida pelos conjuntos AM-adequados (considerando cada classe de mutação e todas as classes de mutação juntas) foi muito baixa. Além disso, as seqüências de teste AM-adequadas são curtas, em termos do número de configurações atingidas, e alguns critérios de cobertura, como por exemplo o critério *Todos-Caminhos-com-um-Laço*, exigem seqüências de teste mais longas para satisfazer cada requisito de teste, o que tornou baixa a cobertura das seqüências de teste AM-adequadas. Também,

neste caso, este fato determina que o Teste de Mutação para Estelle não inclui os critérios FCCS. Assim, pode-se concluir que esses critérios são incomparáveis.

Tabela 5.9. Porcentagem de Cobertura das Seqüências AM-adequadas em Relação aos Critérios de Cobertura para Statecharts.

Critérios FCCS	Porcentagem de Cobertura			
	T_{MEF}	T_{MEFE}	T_{ST}	TOTAL
<i>Todos-Caminhos-com-um-Laço</i>	0	0	0	0
<i>Todos-Caminhos-Simples</i>	0.4	0	0.3	0.6
<i>Todos-Caminhos-livre-Laço</i>	0	0	0	0
<i>Todas-Transições</i>	46.2	15.4	41.0	46.2
<i>Todas-Configurações</i>	84.6	53.8	84.6	84.6
<i>Todas-Reações-em-Cadeia</i>	46.2	15.4	41.0	46.2

5.6. Avaliação de Seqüências de Teste Geradas pela Simulação Programada

Nesta seção é ilustrada a utilização dos critérios FCCS e do Teste de Mutação como mecanismos para avaliar a cobertura de seqüências de eventos geradas pela simulação programada de Statecharts, dessa forma demonstrando como esses critérios podem auxiliar para melhorar a atividade de validação de especificações em Statecharts.

O procedimento foi realizado com o apoio de funções disponíveis no ambiente StatSim (Masiero et al., 1991), utilizando o *Módulo de Execução Programada*, desenvolvido por Cangussu (1993).

A Execução Programada é a simulação de um modelo, no caso Statecharts, conduzida através de um programa de controle, descrito em uma Linguagem de Controle de Execução (LCE), em que são definidas probabilidades de disparo dos eventos em cada passo da simulação. Esse tipo de simulação permite observar o comportamento do sistema reagindo sob condições aleatórias, com ou sem intervenção do usuário (Cangussu, 1993). Com a Linguagem de Controle de

Execução, o usuário constrói o programa que será executado durante a simulação programada. É possível especificar neste programa como os eventos serão gerados, podendo ser por meio de distribuição probabilística (normal, exponencial e uniforme); periodicidade fixa de ocorrência e arquivos de entrada (Cangussu, 1993). Após a simulação programada, são gerados dois relatórios: um contendo informações dos passos da simulação; e um segundo relatório que contém informações estatísticas do modelo executado. O relatório estatístico fornece uma visão geral de certas estatísticas do sistema modelado. Mesmo quando o sistema não necessita de uma análise estatística, o relatório constitui-se em um resumo da simulação, ou seja, nele é possível observar se um evento foi gerado um número correto de vezes, se uma transição em que a condição sempre deveria ser verdadeira foi disparada 100% das vezes que foi avaliada, etc. Esses números podem ajudar a detectar certos erros na especificação do statechart. Na Figura 5.5 é apresentada uma tela do Módulo de Execução Programada.

Para realizar a simulação, foram consideradas duas situações: no primeiro caso (simulação S_A) foi simulada uma situação na qual ocorre um fluxo de trens semelhante nas duas linhas férreas e no segundo caso (simulação S_B) uma situação na qual em uma das linhas férreas há um fluxo maior de trens. Na Figura 5.6 é apresentado o programa descrito em LCE para a simulação S_B , em que é fornecida uma probabilidade de disparo de 80% para os eventos do componente *Trem_linha_Down* e 20% para os eventos do componente *Trem_linha_Up*. Para a simulação S_A é fornecida uma probabilidade de 50% de disparo para os eventos dos componentes *Trem_linha_Down* e *Trem_linha_Up*. Na Figura 5.7 é apresentada uma parte do relatório gerado pela Simulação Programada com as informações em cada passo da simulação S_B , e na Figura 5.8 são apresentados os resultados do relatório

estatístico dessa simulação. Nas Tabelas 5.10 e 5.11 são apresentados os eventos gerados pelas simulações S_A e S_B , os quais representam as seqüências de teste obtidas pelas respectivas simulações.

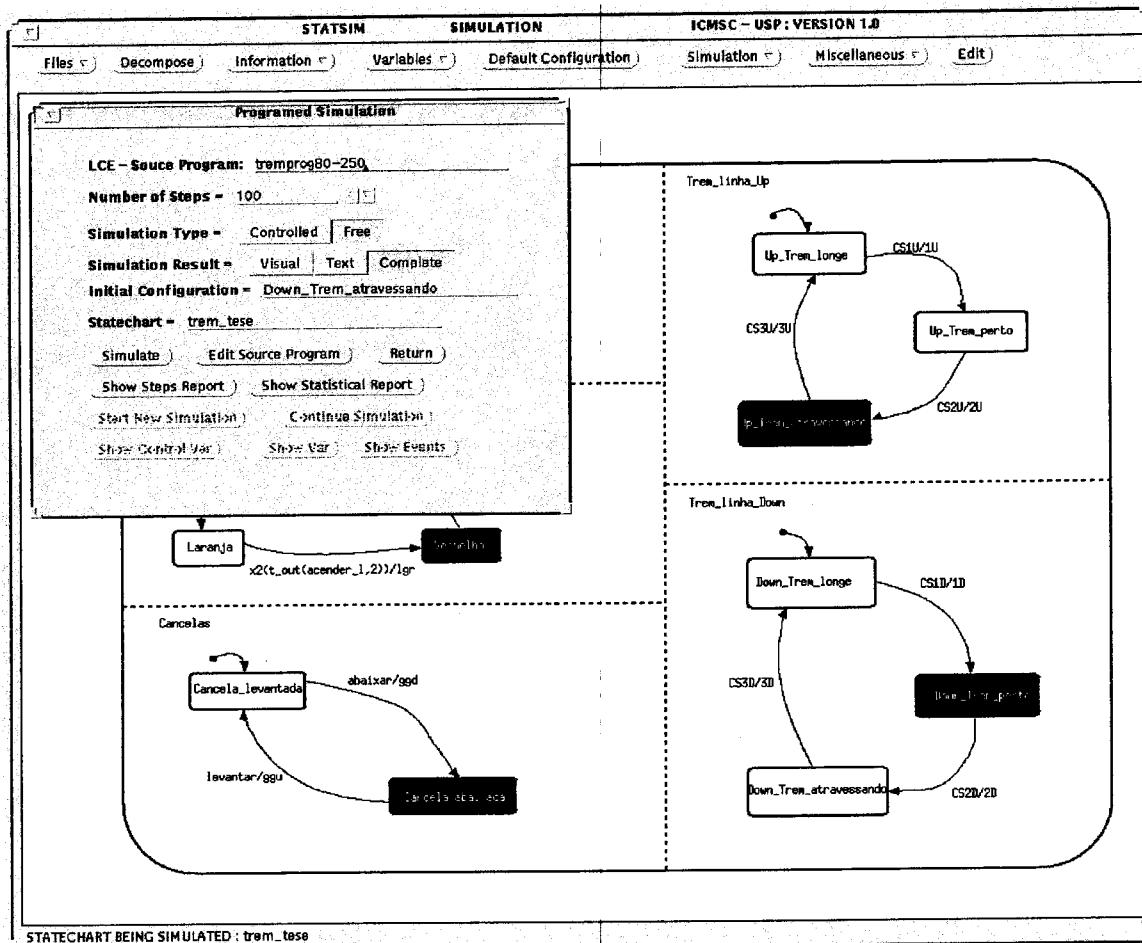


Figura 5.5. Telas da Execução Programada do Ambiente StatSim.

The screenshot shows a software interface with a title bar 'Edit File'. The main area contains a code editor with the following pseudocode:

```
GLOBAL PARAMETERS
  NUMBER_OF_STEPS = 100
  INITIAL_CONFIGURATION = DEFAULT
DECLARATIONS
  CONTROL_VAR %trem1, %trem2, %x, %y
EVENTS CS1D, CS2D, CS3D 20%
EVENTS CS1U, CS2U, CS3U 80%
SIMULATION
  ALL STEPS
  BEGIN
    IF (CONFIGURATION(Up_Trem_longo)) THEN
      BEGIN
        %x := 1
      END
    IF (CONFIGURATION(Down_Trem_longo)) THEN
      BEGIN
        %y := 1
      END
    IF (CONFIGURATION(Up_Trem_atravessando) AND (%x = 1)) THEN
      BEGIN
        %trem1 := %trem1 + 1
        %x := 0
      END
    IF (CONFIGURATION(Down_Trem_atravessando) AND (%y = 1)) THEN
      BEGIN
        %trem2 := %trem2 + 1
        %y := 0
      END
    MESSAGE("Numero de vezes que o trem UP passou = ", %trem1)
    MESSAGE("Numero de vezes que o trem Down passou = ", %trem2)
  END
END_SIMULATION
```

Figura 5.6. Programa para Simulação do Statechart da Figura 5.2.

Steps Report

[Return](#)

Report about source program: tremprog80-100

Statechart in Simulation: trem_tese
Number of Simulated Steps: 100
----- Step 1 -----

Configuration of Active States:

```

Sistema_Controlador_Passagem_Nivel
Controlador
Levantar_cancela
Luzes
Verde
Cancelas
Cancela_levantada
Trem_linha_Up
Up_Trem_perto
Trem_linha_Down
Down_Trem_longo

```

Events triggered : CS3U,CS2U,CS1U,U1
Internal Transcitions :
States Turned On : Up_Trem_perto
States Turned Off : Up_Trem_longo

Variable	Current Value	Average Value
x	0	0.0000

Messages :
Número de vezes que o trem UP passou = 0
Número de vezes que o trem Down passou = 0

----- Step 2 -----

Configuration of Active States:

```

Sistema_Controlador_Passagem_Nivel
Controlador
Levantar_cancela
Luzes
Verde
Cancelas
Cancela_levantada
Trem_linha_Up
Up_Trem_perto
Trem_linha_Down
Down_Trem_longo

```

Events triggered :
Internal Transcitions :
States Turned On :
States Turned Off :

Variable	Current Value	Average Value
x	0	0.0000

Messages :
Número de vezes que o trem UP passou = 0
Número de vezes que o trem Down passou = 0

----- Step 3 -----

Configuration of Active States:

Figura 5.7. Relatório com os Resultados Parciais dos Passos da Simulação Programada.

Statistical Report			
Return			
Statistical Report about source program: tremprog80-100			
STATECHART: trem_tese NUMBER OF STEPS: 100			
STATE	PERCENTAGE OF STEPS IN WHICH WAS ACTIVE	FINAL SITUATION	AVERAGE NUMBER OF CONSECUTIVE STEPS IN THIS STATE
Levantar_cancela	64.00	desligado	1.3704
Abaixar_cancela	36.00	ligado	0.3030
Verde	34.00	desligado	1.2667
Laranja	30.00	desligado	1.0000
Vermelho	36.00	ligado	1.4286
Cancela_levantada	64.00	desligado	1.3704
Cancela_abaixada	36.00	ligado	0.3846
Up_Trem_longo	34.00	desligado	0.2143
Up_Trem_perto	35.00	desligado	0.2500
Up_Trem_atravessando	31.00	ligado	0.1481
Down_Trem_longo	32.00	desligado	4.3333
Down_Trem_perto	31.00	ligado	4.8000
Down_Trem_atravessando	37.00	desligado	6.4000
EVENT S	NUMBER OF STEPS IN WHICH THIS EVENT HAPPENED	PERCENTAGE OF STEPS IN WHICH THIS EVENT WAS AVAILABLE	
abaixar	27	100.00	
Levantar	26	100.00	
acender_l	27	100.00	
acender_v	26	100.00	
D2	5	100.00	
U2	28	100.00	
U1	28	100.00	
D1	6	100.00	
U3	27	100.00	
D3	5	100.00	
CS1U	82	100.00	
CS2U	86	100.00	
CS3U	77	100.00	
CS1D	20	100.00	
CS2D	16	100.00	
CS3D	16	100.00	
TRANSITIONS	NUMBER OF STEPS IN WHICH THIS TRANSITION WAS FIRED	NUMBER OF STEPS IN WHICH THIS TRANSITION WAS EVALUATED	RATE OF FIRING SUCCESS
CS3D/3D	5	5	100.00
CS1D/1D	6	6	100.00
CS3U/3U	27	27	100.00
CS1U/1U	28	28	100.00
3Dv3U/levantar;acende...	26	28	92.86
CS2D/2D	5	5	100.00
CS2U/2U	28	28	100.00
1Uv1Dv2Uv2D	7	8	87.50
2Dv2U/acender_l;abaix...	27	28	96.43
acender_v/ego	14	14	100.00
x2(t_out(acender_l,2)...)	15	30	50.00
acender_l/lgo;x1	15	15	100.00
levantar/ggu	26	26	100.00
abaixar/ggd	27	27	100.00

Figura 5.8. Relatório Estatístico Gerado pela Simulação Programada.

Tabela 5.10. Seqüências de Teste Geradas pela Simulação S_A.

Seqüências de Teste Geradas pela Simulação S _A	
1: CS2D, CS3U	48: CS1D, CS3D, CS2U, CS3U, CS2D, 2D
2: CS3D, CS3U	51: CS1D, CS1U, CS2D, CS3U, CS2U, 2U
3: CS3U, CS2D	52: CS1D, CS3D, 3D, levantar
4: CS3U, CS2D, CS1D, 1D	53: CS1D, 1D, CS2U, CS1U, CS2D, CS3D
5: CS3D, CS2D, 2D, abaixar, acender_1	54: CS3U, 3U, CS1U, CS2D, 2D, abaixar
6: CS2D, CS3U	55: CS2U, CS3U, CS1U, 1U, CS1D
7: CS2D, CS2U	56: CS2U, 2U, CS1D, CS2D, CS3U
8: CS3U, CS3D, 3D, levantar, CS2U, CS2D	57: CS2D
9: CS2D	58: CS3D, 3D, levantar, CS2U
10: CS1U, 1U, CS3U, CS2U, CS3D	59: CS2U
11: CS3U, CS1U, CS3D, CS1D, 1D, CS2U, 2U	60: CS1U, CS1D, 1D
12: CS1U, CS3D, CS2D, 2D	61: CS2U, CS1D, CS3U, 3U
13: CS3U, 3U, levantar, CS3D, 3D	62: CS3U, CS1D
14:	63: CS1U, 1U, CS1D, CS3D
15: CS2U, CS2D, CS3D, CS3U	64: CS1D, CS2D, 2D, abaixar
16: CS2U, CS1U, 1U, CS2D	65: CS2U, 2U, CS2D, CS3U
17: CS1D, 1D, CS3U, CS2U, 2U, abaixar,	66: CS3U, 3U, levantar, CS1D
acender_1	67: CS1D, CS3D, 3D
18: CS1D, CS3U, 3U, levantar, CS1U, CS2D,	68: CS3U, CS2U
2D, CS3D	49: CS1D, CS2D, CS2U, CS3U
19: CS2U, CS3U, CS1D, CS3D, 3D	50: CS2D, CS1U, 1U, CS1D
20: CS1D, 1D, CS1U, 1U, CS2D, CS3D	69: CS3D, CS1D, 1D, CS2U, CS1U, 1U
21: CS2U, 2U, abaixar, CS1U, CS1D, CS3U	70: CS1U, CS2D, 2D, abaixar
22: CS1U, CS3U, 3U, levantar, CS2U	71: CS3D, 3D, levantar
23: CS2U	72: CS2D, CS1D, 1D, CS3U
24: CS2D, 2D, CS1U, abaixar, 1U, CS1D,	73: CS1U, CS2D, 2D, abaixar, CS2U, 2U,
CS2U	CS3D, CS3U, CS1D
25: CS2U, 2U, CS3U	74: CS3U, 3U, levantar, CS2U, CS3D, 3D
26: CS2D, CS3U, 3U, levantar, CS2U, CS1U	75: CS2D, CS1D, 1D, CS1U, 1U, CS3U
27: CS3D, 3D, CS3U, CS2U, CS1D	76: CS2D, 2D, abaixar, CS1D
28: CS1U, 1U, CS3D	77: CS3D, 3D, levantar, CS1U, CS1D
29: CS2U, 2U, abaixar, CS1U, CS1D, 1D,	78: CS3U
CS2D	79: CS2U, 2U, abaixar, CS3U, CS3D
30: CS3U, 3U, levantar, CS1U, CS2U, CS1D	80: CS3U, 3U, levantar, CS2D, CS1D, 1D
31: CS2D, 2D, abaixar, CS2U	81: CS1U, 1U, CS2U, CS2D, 2D, abaixar
32: CS3D, 3D, levantar	82: CS3D, 3D, levantar, CS2U, 2U, CS1U
33: CS3U, CS1D, 1D, CS1U, 1U, CS3D	83: CS1D, 1D, CS3D
34: CS2U, 2U, abaixar, CS1D, CS3D, CS1U,	84: CS2D, 2D, abaixar, CS3D, CS1U
CS2D, 2D	85: CS2U, CS2D, CS3U, 3U, levantar
35: CS2D, CS1U, CS3D, 3D, levantar	86: CS2U, CS3U, CS1U, 1U, CS1D
36: CS3D, CS2U, CS2D, CS3U, 3U	87: CS3U, CS3D, 3D, CS2U, 2U, abaixar
37: CS3U, CS1D, 1D, CS2D, CS1U, 1U, CS2U	88: CS1D, 1D, CS3U, 3U, CS2D
38: CS2D, 2D, abaixar, CS2U, 2U, CS3U	89: CS2D, 2D, CS1U, 1U, CS2U, CS3D, CS1D
39: CS1D, CS1U, CS2D	90: CS2U, 2U, CS1D
40: CS2D, CS1D, CS3U, 3U, levantar,	91: CS1U, CS2U, CS3U, 3U, levantar, CS2D
CS3D, 3D	92: CS1D, CS2D
41: CS1D, 1D, CS2D, CS2U, CS3D	93: CS3U, CS2U, CS1D, CS2D
42: CS3D	94: CS3D, 3D, CS2U, CS1U, 1U, CS2D, CS3U
43: CS1U, 1U, CS2D, 2D, abaixar, CS2U,	95: CS1D, 1D, CS2U, 2U, abaixar
CS1D, CS3D	96: CS3D, CS2U, CS1U
44: CS1U, CS2U, 2U, CS2D	97: CS2U, CS1U, CS2D, 2D
45: CS1U, CS3U, 3U, levantar	98: CS3U, 3U, levantar, CS1U, CS3D, 3D
46: CS3U, CS3D, 3D, CS2U, CS1D	99: CS2U, CS1U, 1U, CS3U
47: CS2D, CS2U, CS3D, CS3U, CS1D, 1D	100: CS1U, CS2D, CS2U, 2U, abaixar

Tabela 5.11. Seqüências de Teste Geradas pela Simulação S_B.

Seqüências de Teste Geradas pela Simulação S _B	
1: CS2D, CS2U, CS1D, 1D	51: CS3D, 3D, levantar, CS1D, CS2D
2:	52: CS1D, 1D, CS3D, CS2D, CS1U, CS2U
3: CS3U, CS3D, CS1D, CS2D, 2D, abaixar	53: CS3D, CS2D, 2D, abaixar, CS2U
4: CS1D, CS2D	54: CS1D, CS2U, CS2D
5: CS2D, CS1D	55: CS3U, 3U, levantar, CS3D, 3D, CS2D,
6: CS1U, 1U, CS3D, 3D, CS1D	CS2U, CS1D
7: CS3D, CS2D, CS1D, 1D	56: CS2D, CS1D, 1D
8: CS3D	57: CS3D, CS1D, CS1U, 1U
9: CS1D	58: CS3D
10: CS1D, CS2D, 2D	59: CS2D, 2D, abaixar, CS1D
11: CS1U, CS2D, CS3U, CS3D, 3D,	60: CS3D, 3D, levantar, CS2D, CS1D
levantar, CS1D	61: CS3D, CS2U, 2U, abaixar, CS1D, 1D,
12: CS3D, CS1D, 1D, CS2D	CS2D
13: CS1D, CS3D, CS2D, 2D, abaixar	62: CS1D, CS3D, CS2D, 2D
14: CS3D, 3D, levantar	63: CS3D, 3D, levantar, CS2D, CS1D, CS1U
15: CS1D, 1D, CS2D, CS3D	64: CS3D, CS1D, 1D
16: CS3D, CS1D, CS2D, 2D, abaixar	65: CS1D, CS3U, 3U
17: CS3D, 3D, levantar, CS1U, CS2D	66: CS3D, CS2D, 2D, abaixar, CS1D, CS3U
18: CS2D, CS3D	67: CS3D, 3D, levantar, CS2U, CS1D
19: CS1U, CS1D, 1D, CS2U, 2U, abaixar,	68: CS3D, CS1D, 1D, CS2U, CS2D
CS2D, CS3D	69: CS1D, CS2D, 2D, abaixar, CS3D
20: CS3D, CS1D, CS2D, 2D, CS1U	70: CS2D, CS3D, 3D, levantar, CS1U, 1U
21: CS2D, CS1U, CS3D, 3D, levantar, CS1D	71: CS1D, 1D, CS3D
22: CS1D, 1D, CS3D	72: CS2D, 2D, abaixar, CS1D, CS2U, 2U
23: CS1D, CS2D, 2D, abaixar, CS2U, CS3D	73: CS1D, CS3D, 3D, levantar, CS3U, 3U,
24: CS3D, 3D, levantar, CS1D	CS2D
25: CS1U, CS2D, CS1D, 1D	74: CS3D, CS1D, 1D, CS2D
26: CS2D, 2D, abaixar, CS1D, CS3U, 3U,	75: CS3U, CS1D, CS3D, CS2D, 2D, abaixar
CS3D	76: CS1U, 1U, CS2D, CS1D
27: CS1D, CS3U	77: CS1D, CS2D, CS3D, 3D, levantar
28: CS1D, CS2D	78: CS1D, 1D, CS3U, CS2D, CS3D, CS1U
29: CS1D, CS3D, 3D, levantar, CS2D	79: CS2D, 2D, abaixar, CS3D, CS1D
30: CS3U, CS3D	80: CS3D, 3D, levantar, CS2D, CS1D
31: CS3D, CS2D	81: CS2D, CS1U, CS3D, CS1D, 1D
32: CS3D, CS2D, CS1D, 1D	82: CS1D, CS1U
33: CS1D, CS3D, CS2U, CS2D, 2D, abaixar	83: CS3D, CS1D, CS2D, 2D, abaixar
34: CS1D, CS1U, 1U	84: CS3D, 3D, levantar, CS3U, CS1U, CS1D
35: CS3D, 3D, levantar	85: CS3U, CS3D, CS2D, CS1U, CS1D, 1D
36: CS1D, 1D, CS2D	86: CS3D, CS1D, CS2D, 2D, abaixar, CS2U,
37: CS1D, CS2D, 2D, abaixar, CS2U, 2U	2U
38: CS3U, 3U, levantar, CS2D, CS3D, 3D,	87: CS1D, CS3D, 3D, levantar, CS2D
CS1D	88: CS1D, 1D, CS3D, CS2D
39: CS1U, 1U, CS1D, 1D, CS3U, CS3D	89: CS1D, CS2D, 2D, abaixar, CS3U, 3U,
40: CS2U, 2U, abaixar, CS1D, CS2D, 2D	CS1U
41: CS3D, 3D, levantar, CS2D	90: CS3D, 3D, levantar, CS1D, CS2D
42: CS1U, CS2U, CS2D, CS3D	91: CS1D, 1D, CS2D, CS3U, CS3D
43: CS1D, 1D, CS2D	92: CS3D, CS1D, CS2U, CS3U
44: CS1D, CS2D, 2D, abaixar, CS1U	93: CS2D, 2D, abaixar, CS1D, CS3D
45: CS2D, CS1D, CS3D, 3D, levantar	94: CS2D, CS1D, CS3D, 3D, levantar
46: CS1D, 1D, CS3D, CS2D	95: CS3D, CS1D, 1D, CS3U, CS2D
47: CS2U, CS1D, CS1U, CS2D, 2D, abaixar,	96: CS1D, CS3D, CS2D, 2D, abaixar, CS3U
CS3D	97: CS1D, CS2D, CS3D, 3D, levantar
48: CS2U, CS3D, 3D, levantar, CS1U,	98: CS3D, CS2D, CS1D, 1D
CS2D, CS1D	99: CS2D, 2D, abaixar, CS3D
49: CS2U, CS1D, 1D, CS3D	100: CS1D, CS2D, CS3D, 3D, levantar
50: CS2D, 2D, abaixar, CS1D, CS3D	

Na Tabela 5.12 são apresentados os resultados da avaliação das seqüências geradas pelas simulações em relação aos critérios de cobertura FCCS e Teste de Mutação. Com relação aos critérios FCCS, observam-se que poucos requisitos de teste são satisfeitos pelas seqüências de teste geradas pelas simulações. A simulação

S_A apresentou uma cobertura melhor para os critérios *Todas-Configurações* e *Todas-Transições*. As seqüências de teste geradas pela simulação S_B não alcançam a configuração C_9 , de forma que o erro existente não pode ser identificado. Com relação ao Teste de Mutação, a cobertura (escore de mutação) obtida foi alta, porém não atingiu o escore de mutação 1 (na média). Conforme descrito anteriormente, o fato do statechart não utilizar variáveis e condições, foram gerados poucos mutantes para a classe MEFE. É interessante observar se esse resultado permanece para statecharts que empregam variáveis e condições.

Esses resultados poderiam ser melhorados modificando a simulação de forma que mais requisitos de teste (e mutantes) pudessem ser cobertos. Outra situação seria selecionar manualmente para disparo as transições, de modo a percorrer um número aceitável de requisitos de cada critério de cobertura e para distinguir mutantes que permaneceram vivos.

Tabela 5.12. Cobertura Obtida pelas Seqüências de Teste Geradas pela Simulação em Relação aos Critérios de Teste para Statecharts.

Critérios de Teste	Porcentagem de Cobertura	
	S_A	S_B
FCCS		
<i>Todos-Caminhos-com-um-Laço</i>	0.2	0.1
<i>Todos-Caminhos-Simples</i>	0.8	0.7
<i>Todos-Caminhos-livres-Laço</i>	0.4	0
<i>Todas-Transições</i>	89.7	51.3
<i>Todas-Configurações</i>	100.0	76.9
<i>Todas-Reações-em-Cadeia</i>	89.7	51.3
TOTAL	46.8	30.0
Teste de Mutação		
MEF	0.92	0.91
MEFE	1.0	1.0
ST	1.0	1.0
TOTAL	0.97	0.97

Embora esses resultados não sejam conclusivos, pois somente um exemplo foi empregado, eles fornecem evidências do aspecto complementar dos critérios Teste de

Mutação e FCCS para a atividade de validação, fornecendo parâmetros (requisitos de teste) para guiar essa atividade. Isso poderia ser feito através da utilização incremental dos critérios de teste para Statecharts: a partir de um conjunto de seqüências de teste obtido pela simulação, novas seqüências de teste podem ser geradas considerando, inicialmente, o Teste de Mutação, em seguida critérios de cobertura mais “fracos” (que geram menos requisitos de teste, como por exemplo, *Todas-Configurações* e *Todas-Transições*) e, de acordo com o tempo e recursos disponíveis, considerando critérios mais “fortes” e que também apresentam um custo maior para geração das seqüências de teste. Dessa forma, o conjunto de seqüências de teste pode ser melhorado incrementalmente.

5.7. Considerações Finais

Neste capítulo foram apresentados os resultados da aplicação dos critérios FCCS e do Teste de Mutação para Statecharts em uma especificação exemplo. A especificação exemplo descreve um Sistema de Controle de Passagem em Nível e foi escolhida por apresentar uma situação real e também por conter erros no modelo especificado. Esse último aspecto permitiu ilustrar como os critérios de teste para Statecharts podem contribuir para revelar o erro existente na especificação.

Considerando essa especificação exemplo, outros dois aspectos foram avaliados: 1) o *strength* dos critérios FCCS e do Teste de Mutação e 2) utilização dos critérios FCCS para avaliação de seqüências de teste geradas pela simulação.

O *strength* avalia o quanto é possível satisfazer um critério de teste tendo satisfeito outro. Para isso, é analisada a cobertura obtida (número de requisitos de teste exercitados) em relação a um critério C_i , aplicando-se um conjunto de casos de

teste adequado a um critério C_2 , e vice-versa. O resultado desse estudo com relação aos critérios Teste de Mutação e FCCS mostrou que esses critérios não conseguem satisfazer completamente os requisitos de teste de cada um, o que significa que esses critérios são incomparáveis. Esse resultado motiva o desenvolvimento de estudos considerando outros exemplos de especificações, de modo a avaliar os aspectos complementares desses critérios.

A utilização dos critérios Teste de Mutação e FCCS para avaliação de seqüências de teste geradas pela simulação permitiu ilustrar como esses critérios podem complementar a atividade de simulação. Tendo-se um conjunto de seqüências de teste inicial, obtido pela simulação programada, é possível melhorar esse conjunto incrementalmente, sendo que a definição de novas seqüências de teste pode ser guiada pelos requisitos desses critérios de teste. Nesse sentido, um aspecto que se pretende explorar é o desenvolvimento de um módulo que automatize a avaliação de seqüências de teste, considerando inicialmente a FCCS, o qual poderia estar acoplado ao ambiente StatSim, permitindo a avaliação das seqüências de teste à medida que a simulação evolui, sendo que o testador poderia selecionar os critérios de teste relevantes para os aspectos que deseja analisar no modelo simulado.

No próximo capítulo, esse mesmo estudo foi realizado considerando os critérios de teste para especificações Estelle. A partir da especificação em Estelle do Protocolo Bit-Alternante, são ilustradas a aplicação dos critérios Teste de Mutação para Estelle e FCCE e a utilização desses critérios para complementar a atividade de simulação de especificações em Estelle.

Capítulo 6. Avaliação dos Critérios de Teste para Estelle: Um Estudo de Caso

6.1. Considerações Iniciais

Neste capítulo ilustra-se a aplicação dos critérios: Teste de Mutação para Estelle e Família de Critérios de Cobertura para Estelle (FCCE). A especificação em Estelle do Protocolo *Bit-Alternante* é utilizada como estudo de caso. Essa especificação é simples mas abrange algumas características relevantes de Estelle, como por exemplo, comunicação, sincronização, paralelismo e hierarquia, e é adequada para a aplicação manual dos critérios.

É ilustrada a utilização dos critérios de teste como mecanismos para auxiliar na geração de seqüências de teste e como mecanismos para avaliação de seqüências de teste. Para o critério Análise de Mutantes, são apresentados exemplos de especificações mutantes e ilustrada a análise dessas especificações. Nesse estudo não foi avaliado o *strength* dos critérios, pois a realização desse estudo manualmente é inviável, devido ao número elevado de requisitos e de seqüências de teste gerados pelos critérios FCCE.

Os recursos para simulação disponíveis na ferramenta *EDT* são utilizados para a geração aleatória de um conjunto de seqüências de teste o qual é avaliado pelos

critérios de teste. Esse aspecto ilustra a utilização dos critérios para complementar a atividade de simulação de especificações em Estelle.

6.2. Estudo de Caso: Especificação do Protocolo *Bit-Alternante*

Para ilustrar a aplicação dos critérios de teste para Estelle a especificação do protocolo *Bit-Alternante* é utilizada. A descrição informal e formal desse protocolo foi apresentada no Capítulo 3. No Apêndice A encontra-se a especificação em Estelle completa desse protocolo, conforme documento ISO 9074 (1987).

A descrição desse protocolo em Estelle é composta de 3 módulos: *Usuário*, *Protocolo* e *Rede*. O módulo *Protocolo* descreve o comportamento do protocolo *Bit Alternante*, o módulo *Usuário* descreve os usuários conectados (especificado para 2 usuários) e o módulo *Rede* simula a rede de comunicação por onde as mensagens trafegam. Para facilitar o entendimento dos procedimentos realizados, a Figura 3.2 (Capítulo 3) é reproduzida na Figura 6.1.

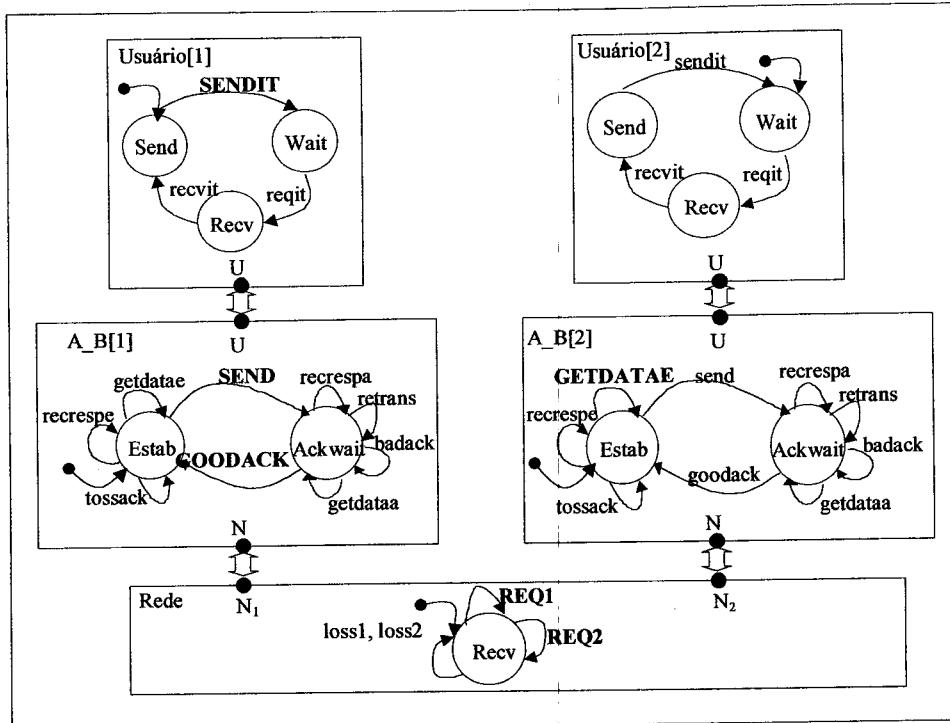


Figura 6.1. Máquinas de Estados Finitos da Especificação do Protocolo Bit-Alternante, ilustrada na Figura 3.2.

Na próxima seção, é ilustrada a aplicação do critério Teste de Mutação para o protocolo Bit_Alternante, descrito nesta seção.

6.3. Geração de Seqüências de Teste Adequadas ao Teste de Mutação

A partir da funcionalidade do protocolo de comunicação, uma seqüência de teste inicial ts é considerada. Essa seqüência descreve uma seqüência típica de operação, sem perdas de pacotes. A seqüência ts corresponde ao seguinte comportamento, sendo que as transições disparadas aparecem ressaltadas na Figura 6.1:

Usuário[1] envia uma mensagem para Usuário[2] (transição sendit). O protocolo de comunicação empacota a mensagem e envia a primeira seqüência de dados para a Rede (transição send). A rede recebe pacote de dados do protocolo e

envia para usuário destino (transição req1). O protocolo de comunicação do Usuário[2] recebe os dados da rede, armazena e envia confirmação para Usuário[1] (transição getdatae). A rede recebe confirmação de recebimento e envia para usuário[1] (transição req2). O protocolo do Usuário[1] recebe a confirmação (transição goodack).

Para simplicidade, visto que os procedimentos foram realizados manualmente, somente o módulo *A_B* é considerado para aplicação do Teste de Mutação. Quando somente alguns componentes são considerados, as entradas de teste correspondem aos eventos gerados pelos módulos conectados ao(s) componente(s) selecionado(s) e as saídas correspondem aos eventos e mensagens produzidos pelo(s) componente(s) selecionado(s). Dessa forma, selecionando-se o módulo *A_B*, ts é igual a:

entradas de teste = $\langle sendit1, req1, req2 \rangle$

saídas de teste = $\langle datarequest(B), datarequest(B) \rangle$

O critério Teste de Mutação foi aplicado e o total de mutantes gerados é apresentado na Tabela 6.1. Foram considerados todos os operadores de mutação, totalizando 363 mutantes. Desse total, 179 são mutantes equivalentes, o que representa 49%. Observam-se que os operadores que geraram um maior número de mutantes foram aqueles relacionados a variáveis, o que é um resultado semelhante ao obtido no nível de programas. Para a Mutação nos Módulos foi considerado somente o módulo *A_B*, e para a Mutação de Interface foram consideradas somente as conexões em que os módulos *Usuário* e *Rede* são os pontos de chamada para o módulo *A_B*.

Tabela 6.1. Total de Mutantes para a Especificação do Protocolo Bit-Alternante.

	Operadores de Mutação	# Mutantes	
		Gerados	Equivalentes
Módulos	1. Substituição de estado inicial	01	0
	2. Substituição de estado origem	18	10
	3. Substituição de estado destino	18	14
	4. Remoção de estados	0	0
	5. Remoção de transições	09	0
	6. Remoção de condição das transições	07	04
	7. Substituição de evento de entrada	0	0
	8. Remoção de evento de entrada	0	0
	9. Negação de condição das transições	07	03
	10. Troca de operadores da condição	26	23
	11. Substituição de atribuição booleana	0	0
	12. Substituição de variável por variável	67	34
	13. Substituição de variável por constante	02	0
	14. Incremento e decremento de variáveis/constantes	20	04
	15. Inclusão de operadores unários nas variáveis	10	02
	16. Substituição da ação das transições	22	12
	17. Cobertura de código	13	05
	18. Remoção de prioridades das transições	0	0
	19. Substituição de prioridades das transições	0	0
	20. Remoção de atrasos das transições	01	0
	21. Substituição de atrasos das transições	0	0
	22. Substituição de política de fila	02	01
Interface	1. Substituição dos parâmetros	0	0
	2. Incremento e decremento dos parâmetros	0	0
	3. Troca na ordem dos parâmetros	0	0
	4. Inclusão de operador unário nos parâmetros	0	0
	5. Substituição do comando <i>output</i>	04	0
	6. Remoção do comando <i>output</i>	04	0
	7. Substituição de variáveis de interface	64	36
	8. Substituição de variáveis de não interface	52	26
	9. Incremento e decremento de variáveis	0	0
	10. Inclusão de operadores unários nas variáveis	10	04
	11. Substituição de atribuição booleana	0	0
Estrutura	1. Remoção das conexões	02	0
	2. Inserção de desconexão	02	0
	3. Remoção de desconexão	0	0
	4. Remoção do comando <i>release</i>	0	0
	5. Remoção do comando <i>terminate</i>	0	0
	6. Substituição de <i>release</i> por <i>terminate</i>	0	0
	7. Substituição de <i>terminate</i> por <i>release</i>	0	0
	8. Substituição de <i>release</i> por <i>detach</i>	0	0
	9. Substituição de <i>release</i> por <i>disconnect</i>	0	0
	10. Substituição de <i>terminate</i> por <i>detach</i>	0	0
	11. Substituição de <i>terminate</i> por <i>disconnect</i>	0	0
	12. Paralelismo síncrono por execução seqüencial	0	0
	13. Paralelismo síncrono por paralelismo assíncrono	0	0
	14. Execução seqüencial por paralelismo síncrono	01	0
	15. Execução seqüencial por paralelismo assíncrono	01	01
TOTAL		363	179

É possível observar na Tabela 6.1 que alguns operadores de mutação não geram mutantes para essa especificação. Por exemplo, para os operadores de Mutação de Interface a maioria das primitivas de comunicação não possui parâmetros de forma que os operadores de 1 a 4 não geram mutantes. Os operadores de Mutação na Estrutura relacionados ao aspecto dinâmico também não geram mutantes, visto que esse tipo de estrutura não ocorre nessa especificação.

Aplicando-se a seqüência de teste *ts*, foram distinguidos 36 mutantes, o que representa um escore de mutação de 0.20. Como o escore de mutação obtido foi muito baixo, novas seqüências de teste são necessárias de modo a distinguir os mutantes que permaneceram vivos.

Para ilustrar a aplicação do Teste de Mutação com mais detalhes, é apresentado um mutante gerado para cada classe de mutação, comentando-se o resultado da execução de *ts* com esses mutantes.

Para exemplificar a Mutação nos Módulos, o operador *Substituição de estado origem* é utilizado. Ele é aplicado na transição *send* do módulo *A_B* e, conforme é mostrado na Figura 6.2, o estado origem *ESTAB* é trocado pelo estado *ACKWAIT*. Quando a especificação mutante é executada com *ts* nenhuma saída é emitida, pois a primitiva *send_request* é recebida pelo módulo *A_B*, mas não existe transição para processá-la, pois o estado inicial do módulo *A_B* é *ESTAB* e a transição mutante de *Send* só é habilitada a partir do estado *ACKWAIT*. Desta forma, esse mutante é distinguido pela seqüência de teste e é considerado morto.

Transição Send	Mutante da Transição Send
<pre> from ESTAB to ACKWAIT when U.Send_Request name send begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_Request(B); end; </pre>	<pre> → from ACKWAIT to ACKWAIT when U.Send_Request name send begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_Request(B); end; </pre>

Figura 6.2. Exemplo de Mutação nos Módulos: Operador *Substituição do Estado Origem*.

Para a Mutação de Interface, foram escolhidos dois operadores de mutação: o operador *Substituição de comando output*, que realiza a mutação no ponto de chamada da interface, e o operador *Substituição de variáveis de não interface*, que realiza a mutação no módulo chamado.

Para a conexão entre o módulo *Usuário* e o módulo *A_B*, definida pelo comando *output* da transição *sendit* do módulo *Usuário* e pela transição *send* do módulo *A_B*, os mutantes utilizados para ilustrar os critérios escolhidos são apresentados nas Figuras 6.3 e 6.4. Quando o mutante da Figura 6.3 é executado com *ts* nenhuma saída é obtida pois a primitiva *Send_request* não é recebida pelo módulo *A_B*, ficando este módulo esperando receber algum dado do usuário, ou algum dado da rede. Quando a primitiva *Receive_request* é recebida por *A_B* permanece na fila desse módulo, pois não existem dados para serem enviados. Dessa forma, essa especificação mutante é distinguida da especificação original.

Transição Sendit	Mutante da Transição Sendit
<pre>Module User ... from SEND to WAIT delay(30); name SENDIT begin Udata.size := 5; Udata.info := 'Hello '; output U.Send_Request(Udata); end;</pre>	<pre>Module User ... from SEND to WAIT delay(30); name SENDIT begin Udata.size := 5; Udata.info := 'Hello '; →output U.Receive_Request; end;</pre>

Figura 6.3. Exemplo de Mutação de Interface: Operador *Substituição de Comando Output*.

Transição Send (Módulo A_B)	Mutante da Transição Send
<pre>Module A_B ... from ESTAB to ACKWAIT when U.Send_Request name send begin copy(P.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_Request(B); end;</pre>	<pre>Module A_B ... from ESTAB to ACKWAIT when U.Send_Request name send begin → copy(Q.Msgdata,Udata); P.Msgseq := Sendseq; store(Sendbuffer, P); format_Data(P,B); output N.Data_request(B); end;</pre>

Figura 6.4. Exemplo de Mutação de Interface: Operador *Substituição de Variável de não Interface*.

Na Figura 6.4 é ilustrada a aplicação do operador de mutação *Substituição de variável de não interface*, sendo que a variável de não interface *P.msgdata* é trocada pela variável *Q.msgdata*, em um comando onde é utilizada uma variável de interface (*udata*). Esse mutante quando executado com *ts* produz a mesma saída obtida pela especificação original. Isto significa que a seqüência de teste não é adequada para distinguir esse mutante. Para distingui-lo é necessário projetar um caso de teste em que a mensagem é recebida pelo *Usuário[2]*, que corresponde a executar as transições *recrespe* (envia dados para usuário) e *recvit* (recebe os dados no módulo *Usuário[2]*). Nesse estágio, esse mutante permanece vivo.

Para ilustrar a Mutação na Estrutura, o operador de mutação *Remoção de conexões* é utilizado. Esse operador remove todas as ocorrências de comandos *connect* e *attach*, um de cada vez. Na Figura 6.5 é apresentada parcialmente a inicialização dos módulos e conexões da especificação original e de uma especificação mutante gerada por esse operador. Na especificação mutante foi retirado o comando para a conexão entre o módulo *Usuário* e o módulo *A_B*. Esse mutante quando executado com *ts* também não produz saída pois a primitiva *send_request* é enviada por meio da transição *sendit* mas, devido à falta da conexão entre os módulos *Usuário* e *A_B*, o módulo *A_B* não recebe a primitiva e as demais transição não são disparadas, ocorrendo *deadlock* no sistema. Desta forma, esse mutante é distinguido pela seqüência de teste *ts*.

Inicialização da Especificação Original	Inicialização da Especificação Mutante
<pre>... init Alternatingbit[Cep] with Alternatingbitbody(Cep); connect User[Cep].U to Alternatingbit[Cep].U; connect Alternatingbit[Cep].N to Network.N[Cep]; end; ...</pre>	<pre>... init Alternatingbit[Cep] with Alternatingbitbody(Cep); → connect Alternatingbit[Cep].N to Network.N[Cep]; end; ...</pre>

Figura 6.5. Exemplo de Mutação na Estrutura: Operador *Remoção de Conexões*.

Na Tabela 6.2 são apresentadas as seqüências de teste adequadas ao Teste de Mutação, juntamente com o número de mutantes que cada uma distingue e o escore de mutação obtido com o acréscimo de cada seqüência de teste. Conforme apresentado no Capítulo 2, o escore de mutação varia no intervalo de 0 a 1 e representa uma medida de cobertura do conjunto de teste em relação ao Teste de Mutação. As seqüências de teste foram obtidas manualmente e o resultado da

execução das especificações mutantes e da especificação original foi obtido com o apoio do simulador da ferramenta *EDT*. Esse simulador permite a execução da especificação, podendo-se selecionar as transições para o disparo e observar a saída obtida. Na Seção 6.5 esse simulador é descrito com maiores detalhes.

Tabela 6.2. Seqüências de Teste Adequadas ao Teste de Mutação para a Especificação do Protocolo Bit-Alternante.

Conjunto de Seqüências de Teste Adequado ao Teste de Mutação para Estelle	# Mutantes Distinguidos	Escore de Mutação
A = {sendit1, req1, req2}	36	0.20
B = {reqit1, sendit2, req2, recvit1, req1, reqit2, sendit1, req1, recvit2, req2}	32	0.37
C = {reqit1, sendit2, req2, reqit2, req1, recvit1, sendit1, req1, recvit2}	26	0.51
D = {reqit1, sendit2, req2, reqit2, recvit1, req1, sendit1, req1, recvit2, req2, reqit1, sendit2, req2, recvit1, reqit2, sendit1, req1, reqit1, recvit2}	70	0.89
E = {reqit1, sendit2, req2, reqit2, req1, recvit1, sendit1, req1, reqit1, recvit2, sendit2, req2, reqit2, recvit1, req1}	4	0.91
F = {reqit1, sendit2, req2, recvit1, reqit2, req1, sendit1, req1, recvit2, reqit1, sendit2, req2, recvit1, req1, reqit2, sendit1, req1, recvit2, reqit1, sendit2, req2, recvit1}	16	1.0

6.4. Geração de Seqüências de Teste Adequadas por Construção aos Critérios FCCE

Nesta seção é ilustrada a utilização dos critérios de cobertura para geração de seqüências de teste. Por motivos de aplicabilidade, é considerado somente o módulo *A_B* da especificação do Protocolo *Bit-Alternante*. Desse modo, os requisitos e seqüências de teste são obtidos a partir da árvore de alcançabilidade apresentada na Figura 4.2 do Capítulo 4. Para facilitar a visualização dos aspectos ilustrados nesta seção, essa Figura é reproduzida novamente na Figura 6.6.

Os procedimentos apresentados no Capítulo 4 são utilizados para obtenção dos requisitos de teste dos critérios de cobertura e geração das seqüências adequadas por construção para esses critérios.

$C_0 = [\text{estab}, 0, 0, 0]$	$C_8 = [\text{ackwait}, 0, \text{dataresponse}, 0]$
$C_1 = [\text{estab}, \text{sendrequest}, 0, 0]$	$C_9 = [\text{ackwait}, 0, \text{dataresponse}, 1]$
$C_2 = [\text{estab}, \text{receiverequest}, 0, 0]$	$C_{10} = [\text{ackwait}, \text{receiverequest}, \text{dataresponse}, 0]$
$C_3 = [\text{estab}, \text{receiverequest}, 0, 1]$	$C_{11} = [\text{ackwait}, 0, 0, 0]$
$C_4 = [\text{estab}, \text{sendrequest}, \text{dataresponse}, 0]$	$C_{12} = [\text{ackwait}, \text{receiverequest}, \text{dataresponse}, 1]$
$C_5 = [\text{estab}, \text{receiverequest}, \text{dataresponse}, 0]$	$C_{13} = [\text{estab}, 0, 0, 1]$
$C_6 = [\text{estab}, \text{sendrequest}, 0, 1]$	$C_{14} = [\text{ackwait}, 0, 0, 1]$
$C_7 = [\text{ackwait}, \text{receiverequest}, 0, 0]$	$C_{15} = [\text{ackwait}, \text{receiverequest}, 0, 1]$

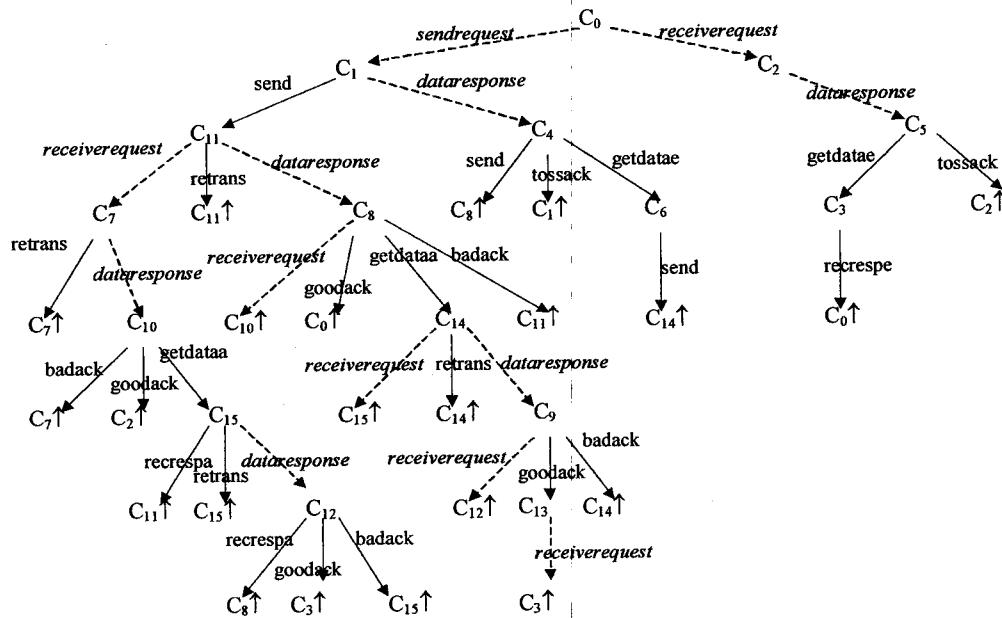


Figura 6.6. Árvore de Alcançabilidade para a Especificação Estelle do Protocolo Bit-Alternante, apresentada na Figura 4.2.

Na Tabela 6.3 são apresentados o total de requisitos e de seqüências de teste gerado para o módulo *A_B*. O critério *Todos-Caminhos* gera um número infinito de caminhos e por isso não foi considerado. O critério *Todos-Caminhos-k-Configurações* também não foi considerado porque gera um número muito elevado de requisitos de teste. Conforme mencionado no Capítulo 4, para a maioria dos critérios de cobertura, o número de seqüências de teste gerado é igual ao número de requisitos de teste porque cada requisito de teste corresponde a um caminho distinto

na árvore de alcançabilidade. Observa-se também que, como ocorre para os critérios de cobertura para Statecharts, o número de requisitos de teste gerado para cada critério de teste é bastante elevado, exigindo um número elevado de seqüências de teste para executar cada requisito de teste.

Tabela 6.3. Total de Requisitos e de Seqüências de Teste Gerado para os Critérios FCCE para a Especificação Estelle do Protocolo *Bit-Alternante*.

Critérios FCCE	Requisitos de Teste	Número de Seqüências de Teste
<i>Todos-Caminhos</i>	infinito	—
<i>Todos-Caminhos-k-C₀-Configurações</i>	17652	17652
<i>Todos-Caminhos-k-Configurações</i>	não aplicado	—
<i>Todos-Caminhos-com-um-Laço</i>	292	292
<i>Todos-Caminhos-Simples</i>	46	46
<i>Todos-Caminhos-livre-Laço</i>	39	39
<i>Todas-Transições</i>	37	22
<i>Todas-Configurações</i>	16	04
TOTAL	18082	18055

Na Tabela 6.4 são apresentados alguns requisitos de teste dos critérios de cobertura da FCCE. As seqüências de teste correspondentes a esses requisitos de teste são apresentadas na Tabela 6.5.

Na próxima seção, é ilustrada a aplicação dos critérios de teste para Estelle (FCCE e Teste de Mutação) para apoiar a atividade de simulação de especificações em Estelle.

Tabela 6.4. Subconjunto de Requisitos de Teste dos Critérios de Cobertura para a Especificação Estelle do Protocolo *Bit-Alternante*.

Critérios FCCE	Requisitos de Teste		
<i>Todos-Caminhos-k-Cr</i> ₀ <i>-Configurações</i>	1 - c0 - c1 - c11 - c7 - c7 - c10 - c2 - c5 - c3 - c0 2 - c0 - c1 - c11 - c7 - c7 - c10 - c2 - c5 - c2 - c5 - c3 - c0 3 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c2 - c5 - c3 - c0 4 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c2 - c5 - c2 - c5 - c3 - c0 5 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c15 - c12 - c8 - c0 6 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c15 - c12 - c8 - c14 - c9 - c12 - c3 - c0 7 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c15 - c12 - c8 - c14 - c14 - c9 - c13 - c3 - c0 8 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c11 - c8 - c10 - c15 - c12 - c8 - c14 - c9 - c12 - c3 - c0		
<i>Todos-Caminhos-com-um-Laço</i>	1 - c0 - c1 - c11 - c7 - c7 - c10 - c2 - c5 - c3 2 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c12 - c8 - c14 - c9 - c13 - c3 3 - c0 - c1 - c11 - c7 - c7 - c10 - c15 - c12 - c3 4 - c0 - c1 - c11 - c7 - c10 - c7 5 - c0 - c1 - c11 - c7 - c10 - c2 - c5 - c3 - c0 6 - c0 - c1 - c11 - c7 - c10 - c2 - c5 - c2 7 - c0 - c1 - c11 - c7 - c10 - c15 - c11 - c8 - c14 - c9 - c12 - c3 8 - c0 - c1 - c11 - c7 - c10 - c15 - c11 - c8 - c14 - c9 - c13 - c3		
<i>Todos-Caminhos-Simples</i>	1 - c0 - c1 - c11 - c7 - c10 - c2 - c5 - c3 - c0 2 - c0 - c1 - c11 - c7 - c10 - c15 - c12 - c8 - c0 3 - c0 - c1 - c11 - c7 - c10 - c15 - c12 - c8 - c14 - c9 - c13 - c3 - c0 4 - c0 - c1 - c11 - c7 - c10 - c15 - c12 - c3 - c0 5 - c0 - c1 - c11 - c8 - c10 - c7 6 - c0 - c1 - c11 - c8 - c10 - c2 - c5 - c3 - c0 7 - c0 - c1 - c11 - c8 - c10 - c15 - c12 - c3 - c0 8 - c0 - c1 - c11 - c8 - c0		
<i>Todos-Caminhos-livre-Laço</i>	1 - c0 - c1 - c11 - c7 - c10 - c2 - c5 - c3 2 - c0 - c1 - c11 - c7 - c10 - c15 - c12 - c8 - c14 - c9 - c13 - c3 3 - c0 - c1 - c11 - c7 - c10 - c15 - c12 - c3 4 - c0 - c1 - c11 - c8 - c10 - c7 5 - c0 - c1 - c11 - c8 - c10 - c2 - c5 - c3 6 - c0 - c1 - c11 - c8 - c10 - c15 - c12 - c3 7 - c0 - c1 - c11 - c8 - c14 - c15 - c12 - c3 8 - c0 - c1 - c11 - c8 - c14 - c9 - c12 - c3		
<i>Todas-Transições</i>	1 - (c0,c1) 2 - (c1,c11) 3 - (c0,c2) 4 - (c2,c5) 5 - (c11,c7) 6 - (c1,c4) 7 - (c4,c8) 8 - (c5,c3) 9 - (c7,c7) 10 - (c11,c11) 11 - (c11,c8) 12 - (c8,c10)	14 - (c4,c6) 15 - (c6,c14) 16 - (c3,c0) 17 - (c5,c2) 18 - (c7,c10) 19 - (c10,c7) 20 - (c8,c0) 21 - (c8,c14) 22 - (c14,c15) 23 - (c8,c11) 24 - (c10,c2) 25 - (c10,c15)	26 - (c15,c11) 27 - (c14,c14) 28 - (c14,c9) 29 - (c9,c12) 30 - (c15,c15) 31 - (c15,c12) 32 - (c12,c8) 33 - (c9,c13) 34 - (c13,c3) 35 - (c9,c14) 36 - (c12,c3) 37 - (c12,c15)
<i>Todas-Configurações</i>	c0, c1, c2, c11, c4, c5, c7, c8, c6, c3, c10, c14, c15, c9, c12, c13		

Tabela 6.5. Subconjunto de Seqüências de Teste Adequado por Construção aos Critérios de Cobertura para a Especificação Estelle do Protocolo *Bit-Alternante*.

Critérios FCCE	Seqüências de Teste	
<i>Todos-Caminhos-k-Configurações</i>	1 - sendit, send, reqit, retrans, req1, goodack, req1, getdatae, recrespe 2 - sendit, send, reqit, retrans, req1, goodack, req1, tossack, req1, getdatae, recrespe 3 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, goodack, req1, getdatae, recrespe 4 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, goodack, req1, tossack, req1, getdatae, recrespe 5 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, getdataa, req1, recrespa, goodack 6 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, getdataa, req1, recrespa, getdataa, retrans, req1, reqit, goodack, recrespe 7 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, getdataa, req1, recrespa, getdataa, retrans, req1, goodack, reqit, recrespe 8 - sendit, send, reqit, retrans, req1, getdataa, recrespa, req1, reqit, getdataa, req1, recrespa, getdataa, req1, reqit, goodack, recrespe	
<i>Todos-Caminhos-comum-Laço</i>	1 - sendit, send, reqit, retrans, req1, goodack, req1, getdatae 2 - sendit, send, reqit, retrans, req1, getdataa, req1, recrespa, getdataa, req1, goodack, reqit 3 - sendit, send, reqit, retrans, req1, getdataa, req1, goodack 4 - sendit, send, reqit, req1, badack 5 - sendit, send, reqit, req1, goodack, req1, getdatae, recrespe 6 - sendit, send, reqit, req1, goodack, req1, tossack 7 - sendit, send, reqit, req1, getdataa, recrespa, req1, getdataa, req1, reqit, goodack 8 - sendit, send, reqit, req1, getdataa, recrespa, req1, getdataa, req1, goodack, reqit	
<i>Todos-Caminhos-Simples</i>	1 - sendit, send, reqit, req1, goodack, req1, getdatae, recrespe 2 - sendit, send, reqit, req1, getdataa, req1, recrespa, goodack 3 - sendit, send, reqit, req1, getdataa, req1, recrespa, getdataa, req1, goodack, reqit, recrespe 4 - sendit, send, reqit, req1, getdataa, req1, goodack, recrespe 5 - sendit, send, req1, reqit, badack 6 - sendit, send, req1, reqit, goodack, req1, getdatae, recrespe 7 - sendit, send, req1, reqit, getdataa, req1, goodack, recrespe 8 - sendit, send, req1, goodack	
<i>Todos-Caminhos-livre-Laço</i>	1 - sendit, send, reqit, req1, goodack, req1, getdatae 2 - sendit, send, reqit, req1, getdataa, req1, recrespa, getdataa, req1, goodack, reqit 3 - sendit, send, reqit, req1, getdataa, req1, goodack 4 - sendit, send, req1, reqit, badack 5 - sendit, send, req1, reqit, goodack, req1, getdatae 6 - sendit, send, req1, reqit, getdataa, req1, goodack 7 - sendit, send, req1, getdataa, reqit, req1, goodack 8 - sendit, send, req1, getdataa, req1, reqit, goodack	
<i>Todas-Transições</i>	1 - sendit, send, reqit, retrans 2 - sendit, send, reqit, req1, badack 3 - sendit, send, reqit, req1, goodack 4 - sendit, send, reqit, req1, getdataa, recrespa 5 - sendit, send, reqit, req1, getdataa, retrans 6 - sendit, send, reqit, req1, getdataa, req1, recrespa 7 - sendit, send, reqit, req1, getdataa, req1, goodack 8 - sendit, send, reqit, req1, getdataa, req1, badack 9 - sendit, send, retrans 10 - sendit, send, req1, reqit 11 - sendit, send, req1, goodack	
<i>Todas-Configurações</i>	12 - sendit, send, req1, getdataa, reqit 13 - sendit, send, req1, getdataa, retrans 14 - sendit, send, req1, getdataa, req1, reqit 15 - sendit, send, req1, getdataa, req1, goodack, reqit 16 - sendit, send, req1, getdataa, req1, badack 17 - sendit, send, req1, badack 18 - sendit, req1, send 19 - sendit, req1, tossack 20 - sendit, req1, getdatae, send 21 - reqit, req1, getdatae, recrespe 22 - reqit, req1, tossack	
	1 - sendit, send, reqit, req1, getdataa, req1 2 - sendit, send, req1, getdataa, req1, goodack 3 - sendit, req1, getdatae 4 - reqit, req1, getdatae	

6.5. Avaliação de Seqüências de Teste Geradas pela Simulação: Ferramenta EDT

Nesta seção é ilustrada a aplicação dos critérios de Teste de Mutação e FCCE para avaliação de seqüências de teste geradas pela simulação, utilizando o simulador *Edb* da ferramenta *EDT* (EDT, 2000).

A ferramenta *EDT* fornece um método automático para obtenção de implementações distribuídas, na linguagem C, a partir de especificações em Estelle. Dentre as ferramentas que fazem parte da *EDT*, o simulador/depurador *Edb* permite que o usuário observe o comportamento dinâmico da especificação. O usuário pode controlar, observar e rastrear a execução da especificação através de comandos do simulador. A especificação pode ser simulada interativamente ou de maneira controlada. Na simulação interativa o usuário conduz a simulação escolhendo as transições entre as transições aptas a disparar ou determinando um módulo a partir do qual as transições serão selecionadas. Na simulação controlada o usuário pode definir, por meio de uma linguagem, um programa que controla a simulação (semelhante à Simulação Programada do ambiente StatSim).

Na Figura 6.7 é apresentado um programa disponível na ferramenta *EDT*, que pode ser utilizado para gerar as seqüências de teste aleatoriamente. Esse programa executa 5 vezes 20 transições escolhidas aleatoriamente, reiniciando o simulador para tentar obter seqüências de transições diferentes. As seqüências de transições obtidas com a execução desse programa são apresentadas na Tabela 6.6.

```

% this simulation scenario will execute
% 5 times 20 (most probably different) transitions.
%
#parameter_1:=20; \%number of transitions to be fired
#parameter_2:=5;   \%number of time the loop has to be repeated
%
#nb:=0;
#tfs:=$total_firing_steps;
do if #nb=>#parameter_2\
    then display #nb;
        display "successful end";
        exit
    else $firing_steps := #parameter_1 ;
        continue;
        if $total_firing_steps = #tfs + #parameter_1 \
            then display #nb;
                display $total_firing_steps;
                restart;
                #nb:=#nb+1;
                #tfs:=0;
            else if $is_break\
                then display "BREAK" \
                else display "non_successful end";
            fi;
            display #nb;
            display $current_firing_step;
            if $is_deadlock then display "DEADLOCK" fi;
            if $is_error then display "RUN TIME ERROR" fi;
            exit
        fi
    od

```

Figura 6.7. Exemplo de um Programa para Simulação de Especificações Estelle, disponível na Ferramenta EDT.

O critério Teste de Mutação foi utilizado para avaliar as seqüências de teste geradas pelo simulador. Para isso, os mutantes não equivalentes da especificação foram executados com essas seqüências de teste e os resultados são apresentados na Tabela 6.7. Para execução dos mutantes foi utilizado também o simulador da ferramenta *EDT*. Na tabela são apresentados somente os operadores de mutação que geraram mutantes e, para cada um, são listados: o número de mutantes não equivalentes, número de mutantes que foram mortos (ou distinguidos) pelas seqüências de teste e o escore de mutação obtido. É possível observar que as seqüências de teste atingiram um escore de mutação, na média, de 0.95 (considerando todos os operadores de mutação da Tabela 6.7), o que representa uma

boa cobertura. Entretanto, o escore de mutação foi baixo para alguns operadores de mutação, por exemplo, para os operadores de Mutação nos Módulos *Substituição de variável por variável* e *Cobertura de código*, e para os operadores de mutação de interface *Substituição de variável de interface*, *Substituição de variável de não interface* e *Inclusão de operadores unários nas variáveis*. Para melhorar a cobertura desses operadores de mutação pode-se optar por gerar novas seqüências de transições aleatoriamente e analisar se o escore de mutação aumenta ou então gerar manualmente seqüências de transições específicas para distinguir os mutantes que permaneceram vivos.

Tabela 6.6. Seqüências de Transições Geradas pelo Programa de Simulação da Figura 6.7.

Seqüências de Teste Geradas pela Simulação
seqüência A: REQIT, SENDIT, SEND, REQ1H, GETDATAE, LOSS2, RECRRESPE, RECVIT, REQIT, RETRANS, LOSS1, SENDIT, SEND, REQ2E, RETRANS, REQ1C, GETDATAA, RECRRESPA, REQ1H, RECVIT
seqüência B: REQIT, SENDIT, SEND, REQ1H, GETDATAE, RECRRESPE, RECVIT, REQ2D, GOODACK, REQIT, SENDIT, SEND, REQ2F, GETDATAE, REQ1D, RECRRESPA, GOODACK, RECVIT, REQIT, SENDIT
seqüência C: REQIT, SENDIT, SEND, REQ1F, GETDATAE, REQ2F, RECRRESPE, GOODACK, RECVIT, REQIT, SENDIT, SEND, REQ2G, GETDATAE, REQ1D, GOODACK, RECRRESPA, RECVIT, REQIT, SENDIT
seqüência D: REQIT, SENDIT, SEND, REQ1E, GETDATAE, RECRRESPA, RECVIT, REQ2H, GOODACK, REQIT, SENDIT, SEND, LOSS2, REQIT, RETRANS, REQ2F, GETDATAE, RECRRESPA, REQ1B, GOODACK,

Tabela 6.7. Resultado da Aplicação de Seqüências de Teste geradas pela Simulação para os Mutantes Gerados para a Especificação do Protocolo *Bit-Alternante*.

	Operadores de Mutação	# Mutantes		Escore de Mutação (%)
		Vivos	Mortos	
Módulos	1. Substituição de estado inicial	01	01	1.00
	2. Substituição de estado origem	08	08	1.00
	3. Substituição de estado destino	04	04	1.00
	5. Remoção de transições	09	09	1.00
	6. Remoção de condição das transições	03	03	1.00
	9. Negação de condição das transições	04	04	1.00
	10. Troca de operadores da condição	03	03	1.00
	12. Substituição de variável por variável	33	25	0.76
	13. Substituição de variável por constante	02	02	1.00
	14. Incremento/decremento de variáveis/constantes	16	16	1.00
	15. Inclusão de operadores unários nas variáveis	08	08	1.00
	16. Substituição de ação das transições	10	09	0.90
	17. Cobertura de código	08	06	0.75
	20. Remoção de atrasos das transições	01	01	1.00
	22. Substituição de política de fila	01	01	1.00
	Escore de Mutação			0.96
Interface	5. Substituição de comando <i>output</i>	04	04	1.00
	6. Remoção de comando <i>output</i>	04	04	1.00
	7. Substituição de variáveis de interface	28	22	0.79
	8. Substituição de variáveis de não interface	26	21	0.81
	10. Inclusão de operadores unários nas variáveis	06	05	0.83
Escore de Mutação			0.89	
Estrutura	1. Remoção de conexões	02	02	1.00
	2. Inserção de desconexão	02	02	1.00
	14. Execução seqüencial por paralelismo síncrono	01	01	1.00
Escore de Mutação			1.00	
TOTAL			0.95	

Os critérios de cobertura da FCCE também foram utilizados para avaliar as seqüências de teste geradas aleatoriamente. Os resultados foram obtidos manualmente: de posse dos requisitos de teste de cada critério foi avaliado quantos requisitos as seqüências de teste exercitam ou percorrem. A Equação 5.1, apresentada no Capítulo 5, é utilizada para calcular a cobertura das seqüências de teste em relação aos critérios de cobertura. Os resultados são apresentados na Tabela 6.8. Devido ao rigor dos critérios de cobertura, os quais exigem que seqüências de teste percorram caminhos específicos na árvore de alcançabilidade a cobertura foi muito baixa. A cobertura poderia ser melhorada se o tamanho (número de transições)

de cada seqüência fosse maior. Outra situação seria disparar manualmente as transições de modo a percorrer um número aceitável de requisitos de cada critério de cobertura.

Tabela 6.8. Resultado da Avaliação da Cobertura das Seqüências de Teste Geradas pela Simulação em relação à FCCE para a Especificação do Protocolo *Bit-Alternante*.

Critérios FCCE	# Requisitos Exercitados	Porcentagem de Cobertura
<i>Todos-Caminhos-k-C₀-Configurações</i>	01	0
<i>Todos-Caminhos-com-um-Laço</i>	01	0.3
<i>Todos-Caminhos-Simples</i>	02	4.3
<i>Todos-Caminhos-livre-Laço</i>	0	0
<i>Todas-Transições</i>	13	35.1
<i>Todas-Configurações</i>	10	62.5
TOTAL	27	17.0

Esses dados ilustram a utilização dos critérios de teste definidos nesta tese para guiar a atividade de simulação de especificações Estelle. Nesse contexto, os critérios poderiam ser utilizados para mensurar a cobertura das seqüências de teste obtidas durante a simulação em relação a esses critérios de teste. Esse processo poderia ser realizado incrementalmente selecionando os critérios relevantes e tentando simular novas situações até que todos os requisitos dos critérios de teste escolhidos fossem satisfeitos, contribuindo assim para efetivamente aprimorar a atividade de teste de especificações Estelle.

6.6. Considerações Finais

Neste capítulo foram apresentados os resultados da aplicação dos critérios Teste de Mutação e FCCE para Estelle, utilizando a especificação do protocolo *Bit-Alternante*. Dois aspectos foram ilustrados: a utilização dos critérios como mecanismos para geração de seqüências de teste e como mecanismos para avaliação

de seqüências de teste. Para esse último aspecto, o simulador da ferramenta EDT foi empregado para a geração aleatória de seqüências de teste.

Foi possível observar que, mesmo considerando um exemplo simples, o custo de aplicação (em termos do número de requisitos de teste dos critérios) é elevado, indicando que a aplicação sem apoio de uma ferramenta de teste é desapropriada, pois se torna uma atividade dispendiosa e propensa a erros.

Tendo ferramentas que automatizem os critérios de teste para Estelle será possível realizar novos estudos, considerando especificações mais complexas e reais. Um dos objetivos é avaliar como esses critérios de teste podem ser aplicados incrementalmente para auxiliar na validação de especificações Estelle e como reduzir os custos de aplicação.

A utilização dos critérios de teste para avaliar seqüências de teste geradas pela simulação permitiu ilustrar como esses critérios podem ser empregados para auxiliar a atividade de simulação. Da mesma forma que ocorre para os critérios de teste definidos para Statecharts, esses critérios podem guiar a simulação fornecendo, à medida que ela evolui, informações sobre a cobertura das seqüências em relação aos critérios de teste selecionados.

Pretende-se investigar, futuramente, os aspectos ilustrados neste capítulo, considerando outros exemplos de especificações em Estelle, mais complexos, de modo a analisar a eficácia em revelar erros desses critérios de teste.

Finalizando este trabalho, no próximo capítulo são apresentadas as conclusões e contribuições desta teste.

Capítulo 7. Conclusões

7.1. Ponderações Finais desta Tese

Este trabalho apresentou a definição de critérios para o teste de especificações de sistemas reativos, descritos em Estelle e em Statecharts. Essa definição baseou-se em critérios de Fluxo de Controle e no Teste de Mutação, originalmente definidos para o teste de programas.

Tendo como motivação o trabalho de Fabbri (1996), que define o Teste de Mutação para especificações baseadas em Máquinas de Estados Finitos, Redes de Petri e Statecharts, esta tese apresentou a definição desse critério para o teste de especificações em Estelle. Essa definição baseou-se na hipótese do projetista competente e do efeito de acoplamento, as quais foram estabelecidas através de um paralelo com as hipóteses para o Teste de Mutação no nível de programas (Fabbri, 1996).

Para aplicação do Teste de Mutação é necessário determinar um conjunto de operadores de mutação, o qual é baseado nos erros típicos que podem ser cometidos pelo programador/projetista do sistema. Os tipos de erros em especificações Estelle foram divididos em três classes: erros de comportamento, erros de interface/comunicação entre módulos e erros na estrutura (ou arquitetura) da especificação.

O conjunto de operadores de mutação considera as características intrínsecas de Estelle, como por exemplo, a comunicação, o paralelismo e as estruturas dinâmicas. A divisão desses operadores de mutação em classes de aplicação: Mutação nos Módulos, Mutação de Interface e Mutação na Estrutura, possibilitou a definição de uma estratégia incremental para aplicação do teste de Mutação em Estelle (Souza et al., 2000a). Essa estratégia permite a condução da atividade de validação dando-se prioridade para erros específicos no modelo, além de possibilitar a aplicação sistemática do critério, de acordo com as condições disponíveis para realização dessa atividade (por exemplo, de acordo com o tempo e recursos disponíveis para sua condução). Nessa estratégia é possível aplicar, inicialmente, os operadores de mutação nos módulos, de forma a testar o comportamento dos módulos da especificação, em seguida os operadores de mutação de interface para testar a comunicação entre os módulos e, finalmente, os operadores de mutação na estrutura para testar os aspectos estruturais da especificação. Além disso, é possível também aplicar somente alguns operadores de mutação de cada classe, selecionando os mais relevantes de acordo com os objetivos da atividade de teste.

Um dos problemas com a aplicação do Teste de Mutação é o seu alto custo, decorrente do número de mutantes que podem ser gerados e que precisam ser analisados. Foi apresentado que, no contexto de Estelle, o número de mutantes gerados é proporcional ao número de: variáveis, parâmetros das primitivas de comunicação, estados, transições e conexões entre módulos da especificação. Nesse sentido, foram apresentadas alternativas que podem ser utilizadas para viabilizar a aplicação desse critério em Estelle: Mutação *Restrita*, *Aleatória* e *Seletiva*.

A aplicação do Teste de Mutação sem o suporte de uma ferramenta é impraticável. Nesse sentido, foram apresentados alguns aspectos que devem ser

considerados no desenvolvimento de uma ferramenta de teste para apoiar esse critério. Esses aspectos são relacionados à geração, execução e análise dos mutantes. As ferramentas já existentes para esse critério de teste: *Proteum* (Delamaro, 1993), *Proteum/IM* (Delamaro, 1997), *Proteum/RS-FSM* (Fabbri et al., 1994; 1999b), *Proteum/RS-ST* (Sugeta, 1999; Fabbri et al., 1999a) e *Proteum/RS-PN* (Simão, 2000; Simão et al., 2000), fornecem uma base essencial para o desenvolvimento de uma ferramenta de validação para Estelle.

Outro aspecto explorado neste trabalho foi o estabelecimento de critérios baseados em Fluxo de Controle para o teste de especificações baseadas em Statecharts e em Estelle. Inicialmente, foi definida a FCCS – Família de Critérios de Cobertura para Statecharts, composta de critérios para validar os aspectos de fluxo de controle e as características intrínsecas da técnica Statecharts, como por exemplo, paralelismo, história e *broadcasting*.

A partir dessa família de critérios, foi definida a FCCE – Família de Critérios de Cobertura para Estelle, composta de critérios de cobertura para validar o fluxo de controle de especificações Estelle e também aspectos como paralelismo, comunicação entre módulos e aspectos dinâmicos.

A árvore de alcançabilidade é utilizada para aplicação dos critérios de cobertura FCCS e FCCE na especificação. A árvore de alcançabilidade tem a vantagem de fornecer uma visão comportamental do sistema modelado e também facilitar a implementação dos critérios de cobertura. Para Statecharts, foi utilizada a árvore de alcançabilidade definida por Masiero et al. (1994).

Para a técnica Estelle, a árvore de alcançabilidade foi definida, utilizando, durante a construção, as técnicas de redução empregadas na construção da árvore de alcançabilidade para Statecharts (Masiero et al., 1994) e também a redução por

seleção dos componentes (Barnard, 1998). A possibilidade de construir a árvore de alcançabilidade considerando alguns componentes do modelo especificado é uma característica que permite diminuir o tamanho da árvore e permite aplicar os critérios de teste considerando alguns módulos da especificação Estelle. Para a técnica Statecharts, esse aspecto não está implementado na construção da árvore de alcançabilidade, mas foi considerado para o estudo de caso apresentado no Capítulo 5.

Os critérios de cobertura FCCS e FCCE podem ser utilizados para geração de seqüências de teste e para avaliar a cobertura de seqüências de teste geradas por outros mecanismos de teste, como por exemplo, a simulação. Dessa forma, os critérios de cobertura podem ser utilizados para complementar a atividade de simulação de especificações em Statecharts e em Estelle, fornecendo mecanismos para quantificar essa atividade.

Através de estudos de caso, foi ilustrada a aplicação da FCCS e da FCCE para apoiar a atividade de simulação de Statecharts e de Estelle, respectivamente. Nesses estudos, avaliou-se a cobertura de seqüências de teste geradas pela simulação em relação aos critérios de cobertura. Os resultados desses estudos indicam que esses critérios podem auxiliar a melhorar as seqüências de teste geradas durante a simulação, pois permitem a sistematização dessa atividade.

Os critérios de teste definidos nesta tese foram avaliados através de dois estudos realizados. No primeiro estudo, foi feita uma comparação entre a FCCS e o Teste de Mutação para Statecharts (Fabbri et al., 1999a). Utilizou-se uma especificação em Statecharts de um Sistema de Passagem em Nível (Barnard, 1998), contendo pelo menos um erro e observou-se que tanto o Teste de Mutação como a FCCS conseguiram identificar o erro.

Nesse estudo analisou-se o *strength* dos critérios FCCS e Teste de Mutação para Statecharts. O resultado desse estudo indica que esses critérios são incomparáveis, pois as seqüências de teste adequadas ao Teste de Mutação não conseguiram percorrer todos os requisitos de teste dos critérios da FCCS e, da mesma forma, as seqüências de teste adequadas a FCCS também não conseguiram distinguir todos os mutantes não equivalentes gerados pelo Teste de Mutação.

No segundo estudo, foi ilustrada a aplicação dos critérios da FCCE e do Teste de Mutação para Estelle, considerando a especificação do protocolo de comunicação *Bit Alternante*. Nesse estudo não foi avaliado o *strength* dos critérios, pois a realização desse estudo manualmente é inviável, devido ao número elevado de requisitos e de seqüências de teste gerados pelos critérios FCCE.

Nesse estudo de caso, demonstrou-se como os critérios podem ser utilizados para geração de seqüências de teste e também a utilização desses critérios para apoiar a atividade de simulação, conforme mencionado anteriormente.

No final deste trabalho pode-se concluir que a definição de critérios de teste para validação do aspecto comportamental de especificações é uma abordagem promissora e contribui em pelo menos dois aspectos: 1) contribui para revelar os erros na fase inicial do desenvolvimento do software, facilitando sua identificação e eliminação e 2) os conjuntos de casos de teste obtidos com a aplicação dos critérios de teste na fase de especificação do sistema podem ser utilizados durante os testes de conformidade da implementação, facilitando essa atividade.

7.2. Contribuições desta Tese

A partir das considerações feitas na seção anterior, podem-se destacar as seguintes contribuições desta tese:

- Definição do Teste de Mutação para o teste de especificações baseadas em Estelle e, portanto, para validação de especificações de sistemas reativos. Foram definidos e classificados os tipos de erros que podem ser cometidos em especificações Estelle e, a partir desses erros, foi estabelecido um conjunto de operadores de mutação;
- Definição de uma estratégia de teste incremental para aplicação do Teste de Mutação para Estelle;
- Análise teórica da complexidade (em termos do número de mutantes gerados) do Teste de Mutação para Estelle e proposta de utilização de mutação alternativa nesse contexto, de modo a diminuir o custo de aplicação desse critério de teste, através da redução do número de mutantes gerados;
- Definição e avaliação de critérios de cobertura para o teste de especificações baseadas em Statecharts e em Estelle (FCCS e FCCE, respectivamente);
- Definição da árvore de alcançabilidade para Estelle, considerando algumas técnicas de redução, durante a sua construção, de modo a controlar a explosão de estados;
- Análise teórica da relação de inclusão entre os critérios FCCS e FCCE demonstrando a ordem parcial entre esses critérios de teste;
- Definição de uma estratégia de teste incremental para aplicação dos critérios de cobertura FCCS e FCCE, considerando a relação de inclusão entre esses critérios.

7.3. Sugestões para Trabalhos Futuros

A seguir, são relacionadas algumas sugestões de trabalhos para continuidade desta linha de pesquisa:

- Desenvolvimento de ferramentas de apoio para aplicação do Teste de Mutação para Estelle e dos critérios FCCS e FCCE;
- Definição dos critérios de teste abordados nesta tese no escopo de outras técnicas para descrição formal de protocolos de comunicação, tais como SDL e Lotos;
- Extensão da Família de Critérios de Cobertura de modo que sejam considerados também os aspectos de fluxo de dados da especificação;
- Adequar a árvore de alcançabilidade de Statecharts, disponível no ambiente StatSim, de modo a considerar aspectos de fluxo de dados da especificação e considerar também a implementação da técnica de redução seleção de componentes durante a construção da árvore;
- Condução de estudos empíricos para avaliar o custo de aplicação, a eficácia em revelar erros e o *strength* dos critérios de teste definidos nesta tese, com o objetivo de definir uma estratégia de teste incremental para a validação de especificações;
- Explorar o relacionamento entre os níveis de abstração: especificação e implementação com relação à atividade de teste de software. O conjunto de seqüências de teste adequado para testar a especificação pode ser empregado durante o teste de conformidade da implementação. Desse modo, faz-se necessária a realização de experimentos visando a avaliar o quanto que os conjuntos de seqüências de teste adequados à especificação são também adequados às suas possíveis implementações.

Referências Bibliográficas

- ACREE, A.T.; BUDD, T.A.; DEMILLO, R.A.; LIPTON, R.J.; SAYWARD, F.G. *Mutation Analysis*. Relatório Técnico GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, GA, setembro, 1979.
- AGRAWAL, H.; DEMILLO, R.; HATHAWAY, R.; HSU, Wm.; HSU, Wynne.; KRAUSER, E.; MARTIN, R.J.; MATHUR, A.; SPAFFORD, E. *Design of Mutant Operators for the C Programming Language*. Relatório Técnico SERC-TR-41-P, Software Eng. Research Center, Purdue University, março, 1989.
- BARBOSA, E.F. *Ensino, Aprendizado e Treinamento no Contexto de Teste e Validação de Software*. Qualificação de Doutorado, ICMC/USP, setembro, 2000.
- BARBOSA, E.F.; MALDONADO, J.C.; VINCENZI, A.M.R.; DELAMARO, M.E.; SOUZA, S.R.S.; JINO, M. Introdução ao Teste de Software. Minicurso apresentado no XIV SBES - Simpósio Brasileiro de Engenharia de Software, João Pessoa, PB, 4-6 outubro, 2000a.
- BARBOSA. E.F.; MALDONADO, J.C.; VINCENZI, A.M.R. Towards the Determination of Sufficient Mutant Operators for C. In: *First International Workshop on Automated Program Analysis, Testing and Verification*, Limerick, Irlanda, junho, 2000b.
- BARNARD, J. COMX: A Design Methodology Using Communicating X-Machines. *Information and Software Technology*, 40, p.271-280, 1998.
- BATISTA, J.E.S. *Um Editor Gráfico para Statecharts*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), 1991.
- BEIZER, B. *Software Testing Techniques*. 2a Edição, Van Nostrand Reinhold, New York, 1990.
- BOAVENTURA, I.A.G. *Propriedades Dinâmicas de Statecharts*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), 1992.

- BOCHMMAN, G.V.; PETRENKO, A. Protocol Testing: Review of Methods and Relevance for Software Testing. In: *International Symposium on Software Testing and Analysis (ISSTA'94)*, ACM-Software Engineering Notes, p.109-124, 1994.
- BOLOGNESI, T.; BRINKSMA, E. Introduction to the ISO Specification Language Lotos. *Computer Networks and ISDN Systems*, 14, p.25-59, 1987.
- BOURHFIR, C.; DSSOULI, R.; ABOULHAMID, E.M. Automatic Test Generation for EFSM-Based Systems. Publication Departamentale #1043, 1996 (disponível em: www.umontreal.ca/labs/teleinfo/PubListIndex.html).
- BOURHFIR, C.; DSSOULI, R.; ABOULHAMID, E.M.; RICO, N. Automatic Executable Test Generation for Extended Finite State Machine Protocols. Publication Departamentale #1060, 1997 (disponível em: www.umontreal.ca/labs/teleinfo/PubListIndex.html).
- BUCHHOLZ, P.; KEMPER, P. *Hierarchical Reachability Graph Generation for Petri Nets*. Publication 660, Universitt Dortmund, Fachbereich Informatik, 1997.
- BUDD, T.A. *Mutation Analysis: Ideas, Examples, Problems and Prospects*. Computer Program Testing, North-Holand Publishing Company, 1981.
- BUDKOWSKI, S.; DEMBINSKI, P. An Introduction to Estelle: A Specification Language for Distributed Systems. *Computer Network and ISDN Systems*, 14, p.3-23, 1987.
- CANGUSSU, J.W.L. *Execução Programada de Statecharts*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), 1993.
- CANGUSSU, J.W.L.; PENTEADO, R.A.D.; MASIERO, P.C.; MALDONADO, J.C. Validation of Statecharts Based on Programmed Execution. *Journal of Computing and Information*, vol. 1(2), Special Issue of the proceedings of the 7th International Conference on Computer and Information – ICCI'95, Ontario, Canada, jullho, 1995.
- CHAIM, M.J. *POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseados em Análise de Fluxo de Dados*. Dissertação (Mestrado), DCA/FEE/UNICAMP - Campinas, SP, Brasil, abril, 1991.
- CHEUNG, S.C.; KRAMER, J. Checking Safety Properties Using Compositional Reachability Analysis. *ACM Transactions on Software Engineering and Methodology*, 8(01), p.49-78, janeiro, 1999.
- CHOI, B.; MATHUR, A.P. High Performance Mutation Testing. *Journal Systems Software*, vol. 20(02), p.135-152, fevereiro, 1993.

- CHOW, T.S. Testing Software Design Modeled by Finite-State Machines. *IEEE Transaction on Software Engineering*, SE-4(3), maio, 1978.
- CHU, P-Y. M.; LIU, M.T. Global State Graph Reduction Techniques for Protocol Validation in the EFSM Model. In: *Proceedings of the IEEE Phoenix Conference on Computers and Communications*, p.371-377, 1989.
- CHUNG, C-M.; SHIH, T.K.; WANG, Y-H.; LIN, W-C.; KOU, Y-F. Task Decomposition Testing and Metrics for Concurrent Programs. In: *Fifth International Symposium on Software Reliability Engineering (ISSRE'96)*, p.122-130, 1996.
- COLEMAN, D.; HAYES, F.; BEAR, S. Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design. *IEEE Transactions on Software Engineering*, vol.18(01), p.9-18, janeiro, 1992.
- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems: Concepts and Design..* 2.ed., Wokingham, England, Addison-Wesley Publishing Company, 1994.
- COURTIAT, J-P.; SAQUI-SANNES, P. ESTIM: an Integrated Environment for the Simulation and Verification of OSI Protocols Specified in Estelle. *Computer Network and ISDN Systems*, 25, p.83-98, 1992.
- COWARD, P. A Review of Software Testing. *Information and Software Technology*, v. 30, n.3, p.189-198, abril, 1988.
- DAVIS, A.M. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM*, vol. 31(09), setembro, 1988.
- DELAMARO, M.E. *Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes.* Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), São Carlos, SP, Brasil, outubro, 1993.
- DELAMARO, M.E. *Mutação de Interface: Um Critério de Adequação Inter-Procedural para o Teste de Integração.* Tese (Doutorado) - Instituto de Física de São Carlos da Universidade de São Paulo (IFSC/USP), São Carlos, SP, 1997.
- DELAMARO, M.E.; MALDONADO, J.C. Interface Mutation: Assessing Testing Quality at Interprocedural Level. In: *International Conference of the Chilean Computer Science Society (19th SCCC)*, p.78-86, Talca, Chile, 1999.
- DELAMARO, M.E.; MALDONADO, J.C.; VINCENZI, A.M.R. Proteum/IM 2.0: An Integrated Mutation Testing Environment. In: *Mutation 2000 - A Symposium on Mutation Testing for the New Century*, San Jose, Califórnia, p.6-7, outubro, 2000.

- DEMILLO, R.A. Mutation Analysis as a Tool for Software Quality Assurance. In: *Proceedings of COMPSAC80*, Chicago - IL, outubro, 1980.
- DEMILLO, R.A. *Software Testing and Evaluation*. The Benjamin/Cummings Publishing Company, Inc., 1987.
- DEMILLO, R.A.; LIPTON, R.J.; SAYWARD, F.G. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer*, abril, 1978.
- DEMILLO, R.A.; OFFUTT, A.J. Constrained-Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, v. 17, n. 9, p.900-910, setembro, 1991.
- DEMILLO, R.A.; GUINDI, D.S.; KING, K.N.; MCKRAKE, W.N.; OFFUTT, A.J. An Extended Overview of the Mothra Testing Environment. In: *Proceedings of the Second Workshop on Software Testing, Verification and Analysis*, Banff - Canadá, 1988.
- DRUSINSKI, D.; HAREL, D. Using Statecharts for Hardware Description and Synthesis. *IEEE Transaction on Computer-Aided Design*, 8(7), julho, 1989.
- DUNCAN, I.M.M.; ROBSON, D.J. Ordered Mutation Testing. *ACM SIGSOFT Software Engineering Notes*, v.15(02), p.29-30, 1990.
- EDT. *ESTELLE DEVELOPMENT TOOLSET (versão 4.1)*. Institut National des Télécommunications, Evry, França, 2000 (Disponível em: www.alix.int-evry.fr/~stan/edt.html).
- FABBRI, S.C.P.F. *A Análise de Mutantes no Contexto de Sistemas Reativos: Uma Contribuição para o Estabelecimento de Estratégias de Teste e Validação*. Tese (Doutorado) - Instituto de Física de São Carlos da Universidade de São Paulo (IFSC/USP), São Carlos, SP, outubro, 1996.
- FABBRI, S.C.P.F.; MALDONADO, J.C.; DELAMARO, M.E.; MASIERO, P.C. Mutation Analysis Testing for Finite State Machine. In: *Fifth International Symposium on Software Reliability Engineering (ISSRE'94)*, Califórnia, p.220-229, novembro, 1994.
- FABBRI, S.C.P.F.; MALDONADO, J.C.; DELAMARO, M.E.; MASIERO, P.C. Proteum/FSM: A Tool to Support Finite State Machine Validation Based on Mutation Testing. In: *International Conference of the Chilean Computer Science Society (19th SCCC)*, p.96-104, Talca, Chile, 1999b.
- FABBRI, S.C.P.F.; MALDONADO, J.C.; MASIERO, P.C.; DELAMARO, M.E. Mutation Analysis Applied to Validate Specifications Based on Petri Nets. In: *8th IFIP Conference on Formal Descriptions Techniques for Distributed Systems and Communication Protocol (FORTE'95)*, p.329-337, Montreal, Canadá, 1995.

- FABBRI, S.C.P.F.; MALDONADO, J.C.; SUGETA, T.; MASIERO, P.C. Mutation Testing Applied to Validate Specifications Based on Statecharts. In: *International Symposium on Software Reliability Engineering (ISSRE'99)*, 1999a.
- FECKO, M.A.; UYAR, M.Ü.; AMER, P.D.; SETHI, A.S.; DZIK, T.; MENELL, R.; MCMAHON, M. A Success Story of Formal Description Techniques: Estelle Specification and Test Generation for MIL-STD 188-220. In: *Computer Communications: The Int'l Journal for the Computer and Telecommunication Industry*, special issue on FDTs in Practice, in press, summer 2000.
- FONSECA, R.P. *Suporte ao Teste Estrutural de Programas Fortran no Ambiente POKE-TOOL*. Dissertação (Mestrado), DCA/FEE/UNICAMP - Campinas, SP, Brasil, janeiro, 1993.
- FORTES, R.P.M. *Uma Ferramenta de Apoio à Utilização de Statecharts para Especificação do Comportamento de Sistemas de Tempo Real Complexos*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), 1991.
- FRANCÊS, C.R.L.; VIJAYKUMAR, N.L.; SANTANA, M.J.; SOLON, V. C.; SANTANA, R.H.C. Stochastic Extension to Statecharts for Representing Performance Models: an Application to a File System. In: *Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho (SBAC-PAD'2000)*, São Pedro, 2000.
- FRANKL, F.G.; WEYUKER, E.J. Data Flow Testing Tools. In: *Softfair II*, San Francisco, CA, p.46-53, dezembro, 1985.
- GHOSH, S.; GOVINDARAJAN, P.; MATHUR, A.P. TDS: A Tool for Testing Distributed Component-Based Applications. In: A Symposium on Mutation Testing for the New Century (Mutation 2000), San Jose, Califórnia, 6-7 de outubro, 2000.
- GHOSH, S.; MATHUR, A.P. Interface Mutation. In: *A Symposium on Mutation Testing for the New Century (Mutation 2000)*, San Jose, Califórnia, 6-7 de outubro, 2000.
- GILL, A. *Introduction to the Theory of Finite-State Machine*. New York, McGraw-Hill, 1962.
- HAREL, D. Bitting the Silver Bullet - Toward a Brighter Future for Systems Development. *Computer IEEE*, p.8-20, janeiro, 1992.
- HAREL, D.; GERY, E. Executable Objetc Modeling with Statecharts. In: *International Conference on Software Engineering (ICSE'96)*, IEEE, 1996.
- HAREL, D.; LACHOVER, H.; NAAMAD, A.; PNUELI, A.; POLITI, M.; SHERMAN, R.; SHTULL-TRAURING, A.; TRAKHTENBROT, M.

- STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transaction on Software Engineering*, 16(4), abril, 1990.
- HAREL, D.; PINNEL, A.; SCHMIDT, J.P.; SHERMAN, R. On the Fornal Semantics of Statecharts. In: *2nd IEEE Symposium on Logic in Computer Science*, Thaca, New York, 1987.
- HAREL, D.; POLITI, M. *Modeling Reactive Systems with Statecharts - The Statemate Approach*. McGraw-Hill, New York, 1998.
- HARROLD, M.J. Testing: A Roadmap. In: *22th International Conference on Software Engineering, in Future of Software Engineering (ICSE'2000)*, junho, 2000.
- HARROLD, M.J.; SOFFA, M.L. Selecting and Using Data for Integration Testing. *IEEE Software*, vol. 8(02), p.58-65, março, 1991.
- HENNIGER, O.; ULRICH, A.; KÖNIG, H. Transformation of Estelle Modules Aiming at Test Case Generation. In: *IFIP International Workshop on Protocol Test Systems (IWPTS'95)*, Paris, p.45-60, 1995.
- HERBERT, J.S.; PRICE, A.M.A. PROTESTE+: Ambiente de Validação Automática de Qualidade de Software através de Técnicas de Teste e Métricas de Complexidade. In: *IX Simpósio Brasileiro de Engenharia de Software - Caderno de Ferramentas*, Recife, PE, 3-6 de outubro, 1995.
- HERMAN, P.M. A Data Flow Analysis Approach to Program Testing. *Australian Computer Journal*, 8(3), novembro, 1976.
- HORGAN, J.R.; MATHUR, A.P. Assessing Testing Tools in Research and Education. *IEEE Software*, p.61-69, maio, 1992.
- HOWDEN, W.E. Methodology for the Generation Program Test Data. *IEEE Transactions on Computing*, vol.24(05), p.554-560, 1975.
- HOWDEN, W.E. *Functional Program Testing and Analysis*. McGraw-Hill, USA, 1987.
- HUANG, C-M.; HSU, J-M. An Incremental Protocol Verification Method. *The Computer Journal*, 37(8), 1994.
- HUANG, C-M.; HSU, J-M.; LAI, H-Y.; HUANG, D-T.; PONG, J-C. An Estelle-Based Incremental Protocol Design Systems. *J. Systems Software*, 36, p.115-135, 1997.
- IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. Padrão 610.12, 1990.

ISO/TC97/SC21/WG1/DIS9074. *Estelle – A formal Description Technique Based on an Extended State Transition Model.* 1987.

JÉRON, T.; JARD, C. Testing for Unboundedness of FIFO Channels. *Theoretical Computer Science*, 113, p.93-117, 1993.

JIRACHIEFPATTANA, A.; LAI, R. EVEN: A Software Environment for Estelle Specification Verification. *Journal of Systems Software*, 39, p.119-143, 1997.

KIM, T-H.; HWANG, I-S.; PARK, C-M.; LEE, J-Y.; LEE, S-B. Automatic Test Case Generation of Real Protocol: Framework and Methodology. In: *International Conference on Formal Description Technique for Distributed System and Communication Protocol and Protocol Specification, Testing and Verification*, Paris, França, 1998.

KOPPOL, P.V.; TAI, K-C. An Incremental Approach to Strutural Testing of Concurrent Software. In: *International Symposium on Software Testing and Analysis (ISSTA'96)*, ACM-Software Engineering Notes, p.14-23, 1996.

KRAUSER, E.W.; MATHUR, A.P.; REGO, V. High Performance Testing on SIMD Machines. In: *Second Workshop on Software Testing, Verification and Analysis*, Banff - Canadá, 1988.

LAI, R.; JIRACHIEFPATTANA, A. Verifying Estelle Protocol Specifications Using Numerical Petri Nets. *Computer Systems Science & Engineering*, 11(1), janeiro, 1996.

LASKI, J.W.; KOREL, B. A Data Flow Oriented Program Testing Strategy. *IEEE Transactions on Software Engineering*, SE-9(3), maio, 1983.

LEITÃO, P.S.J. *Suporte ao Teste Estrutural de Programas Cobol no Ambiente POKE-TOOL*. Dissertação (Mestrado), DCA/FEE/UNICAMP - Campinas, SP, Brasil, agosto, 1992.

LIN, F.J.; CHU, P.M.; LIU, M.T. Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies. *ACM SIGCOMM*, p.126-135, 1988.

LINNENKUGEL, U.; MÜLLERBURG, M. Test Data Selection Criteria for Software Integration Testing. In: *First International Conference on Systems Integration*, Morristown, Nova Jersey, p.709-717, abril, 1990.

LOPES DE SOUZA, W. Estelle: uma Técnica para Descrição Formal de Serviços e Protocolos de Comunicação. *Revista Brasileira de Computação*, 5(1), p.33-44, 1989.

MALDONADO, J.C. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese (Doutorado) - DCA/FEE/UNICAMP, Campinas, SP, Brasil, julho, 1991.

- MALDONADO, J.C. *Critérios de Teste de Software: Aspectos Teóricos, Empíricos e de Automatização.* Concurso de Livre Docência - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), janeiro, 1997.
- MALDONADO, J.C.; BARBOSA, E.F.; VINCENZI, A.M.R.; DELAMARO, M.E. Evaluating N-Selective Mutation for C Programs. In: *A Symposium on Mutation Testing for the New Century (Mutation 2000)*, San Jose, Califórnia, 6-7 de outubro, 2000b.
- MALDONADO, J.C.; CHAIM, M.L.; JINO, M. Arquitetura de uma ferramenta de teste de apoio aos critérios potenciais usos. In: *XXII Congresso Nacional de Informática*, São Paulo, SP, setembro, 1989.
- MALDONADO, J.C.; DELAMARO, M.E.; FABBRI, S.C.P.F.; SIMÃO, A.S.; VINCENZI, A.M.R.; MASIERO, P.C. Proteum: A Family of Tools to Support Specification and Program Testing Based on Mutation. In: *A Symposium on Mutation Testing for the New Century (Mutation 2000)*, San Jose, Califórnia, 6-7 de outubro, 2000a.
- MALDONADO, J.C.; VINCENZI, A. M.; BARBOSA. E.F.; SOUZA, S.R.S.; DELAMARO, M.E. Aspectos Teóricos e Empíricos de Teste de Cobertura de Software. *VI Escola de Informática da Sociedade Brasileira de Computação (SBC)* - Regional Sul, maio, 1998.
- MARSHALL, A.C.; HEDLEY, D.; RIDDELL, I.J.; HENNELL, M.A. Static Dataflow-Aided Weak Mutation Analysis (SDAWM). *Information and Software Technology*, Vol. 32(01), janeiro/fevereiro, 1990.
- MASIERO, P.C.; FORTES, R.P.M.; BATISTA NETO, J.E.S. Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo Real. In: *XI Congresso Nacional da Sociedade Brasileira de Computação (SBC)*, XVIII SEMISH, Santos, SP, p.45-61, agosto, 1991.
- MASIERO, P.C.; MALDONADO, J.C.; BOAVENTURA, I.G. A Reachability Tree for Statecharts and Analysis of Some Properties. *Information and Software Technology*, 36(10), p.615-624, 1994.
- MATHUR, A.P. Performance, Effectiveness, and Reliability Issues in Software Testing. In: *XV Annual International Computer Software and Applications Conference*, p.604-605, Tokio, Japão, setembro, 1991.
- MATHUR, A.P.; KRAUSER, E.W. Modeling Mutation on Vector Processor. In: *Second Workshop on Software Testing, Verification and Analysis*, Banff, Canadá, 1988.

MATHUR, A.P.; WONG, W.E. Evaluation of The Cost Alternate Mutation Strategies. In: *VII Simpósio Brasileiro de Engenharia de Software (VII SBES)*, Rio de Janeiro, RJ, outubro, 1993.

MATHUR, A.P.; WONG, W.E. An Empirical Comparison of Data Flow and Mutation-Based Test Adequacy Criteria. *The Journal of Software Testing, Verification and Reliability*, v. 4, n. 1, p.9-31, março, 1994.

MURATA, T. *Modeling and Analysis of Concurrent Systems*. Handbook of Software Engineering, Van Nostrand Reinhold Electrical, New York, 1984.

MYERS, G.J. *The Art of Software Testing*. John Wiley & Sons, New York, 1979.

NEPOMMIASCHY, V.A.; ALEKSEEW, G.I.; BYSTROV, A.V.; CHURINA, T.G.; MYLNIKOV, S.P.; OKUNISHNIKOVA, E.V. EPV-Petri Net Based Estelle Protocol Verifier. In: *1st International Workshop on the FDT - Estelle'98*, Evry, França, novembro, 1998.

NTAFOS, S.C. On Required Element Testing. *IEEE Transactions on Software Engineering*, SE-10(6), novembro, 1984.

NTAFOS, S.C. A Comparison of Some Structural Testing Strategies. *IEEE Transactions on Software Engineering*, vol.14(06), p.868-873, junho, 1988.

OFFUTT, A.J. Investigations of the Software Testing Coupling Effect. *ACM Transactions on Software Engineering Methodology*, vol. 1(01), p.3-18, janeiro, 1992.

OFFUTT, A.J.; PAN, J. Detecting Equivalent Mutants and the Feasible Path Problem. In: *Conference on Computer Assurance (COMPASS'96)*, IEEE Computer Society Press, Gaithersburg, MD, p.224-236, junho, 1996.

OFFUTT, A.J.; PAN, J.; TEWARY, K.; ZHANG, T.; An Experimental Evaluation of Data Flow and Mutation Testing. *Software Practice and Experience*, vol. 26(02), p.165-176, fevereiro, 1996a.

OFFUTT, A.J.; ROTHERMEL, A.J.; UNTCH, R.H.; ZAPF, C. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering Methodology*, vol. 5(02), p.99-118, abril, 1996b.

OFFUTT, A.J.; ROTHERMEL, A.J.; ZAPF, C. An Experimental Evaluation of Selective Mutation. In: *15th International Conference on Software Engineering (ICSE'93)*, Baltimore, Maryland, maio, 1993.

PAULO F.B.; MASIERO, P.C.; OLIVEIRA, M.C.F. Hypercharts: Extended Statecharts to Support Hypermedia Specification. In: ICECCS - Third IEEE International Conference on Engineering of Complex Computer Systems, Como, Itália, p.152-61, 8-12 de setembro, 1997.

- PETERSON, J.L. Petri Nets. *Computing Surveys*, vol. 9(03), setembro, 1977.
- PETRENKO, A.; BOCHMANN, G.V. On Fault Coverage of Tests for Finite State Specifications. *Computer Networks and ISDN Systems, Special Issue on Protocol Testing*, 1996 (disponível em: www.umontreal.ca/labs/teleinfo/PubListIndex.html).
- PEZZÈ, M.; TAYLOR, R.N.; YOUNG, M. Graph Models for Reachability Analysis of Concurrent Programs. *ACM Transaction on Software Engineering and Methodology*, 4(02), abril, 1995.
- PRESSMAN, R. S. *Software Engineering - A Practitioner's Approach*. 5^a Edição, McGraw-Hill, 2000.
- PROBERT, R.L.; GUO, F. *Mutation Testing of Protocols: Principles and Preliminary Experimental Results*. Protocol Test Systems, III, Ed. by I. Davidson and D.W. LitwackNorth-Holland, p.57-76, 1991.
- RAPPS, S.; WEYUKER, E.J. Selecting Software Test Data Using Data Flow Information. *IEEE Transaction on Software Engineering*, vol.11(04), p.367-375, abril, 1985.
- ROLINSKI, P.; WYTREBOWICZ, J. Analysis Tools for Estelle Specifications. In: *1st International Workshop on the FDT - Estelle'98*, Evry, França, novembro, 1998.
- SANTANA, A.C.L.; PRADO, A.F.; LOPES DE SOUZA, W. Utilização do Paradigma Draco para Implementar Especificações Estelle na Linguagem C++. In: *XV Simpósio Brasileiro de Redes de Computadores*, São Carlos, SP, p.118-134, 19-22 de maio, 1997.
- SARIKAYA, B.; BOCHMANN, G.V.; CERNY, E. A Test Design Methodology for Protocol Testing. *IEEE Transaction Software Engineering*, SE-13(5), maio, 1987.
- SHAW, M.; GARLAN, D. Formulations and Formalisms in Software Architecture. *Lecture Notes in Computer Science*, Springer-Verlag, vol.1000, 1995.
- SIJELMASSI, R.; STRAUSSER, B. NIST Integrated Tool Set for Estelle. In: *IFIP/FORTE'90*, 1990.
- SIJELMASSI, R.; STRAUSSER, B. *The Distributed Implementation Generator: an Overview and User Guide*. Relatório Técnico NCSL/SNA-91/3, 1991b.
- SIJELMASSI, R.; STRAUSSER, B. *The Portable Estelle Translator: an Overview and User Guide*. Relatório Técnico NCSL/SNA-91/2, 1991a.

SILVA-BARRADAS, S. *Mutation Analysis of Concurrent Software*. Tese (Doutorado), Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano, 1998.

SIMÃO, A. S. *Proteum-RS/PN: Uma Ferramenta para a Validação de Redes de Petri Baseada na Análise de Mutantes*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), São Carlos, SP, fevereiro, 2000.

SIMÃO, A.S.; MALDONADO, J.C. Mutation Based Test Sequence Generation for Petri Nets. In: *Workshop of Formal Methods (IVX SBES)*, João Pessoa, outubro, 2000.

SIMÃO, A.S.; MALDONADO, J.C.; FABBRI, S.C.P.F. , Proteum-RS/PN: A Tool to Support Edition, Simulation and Validation of Petri Nets Based on Mutation Testing. In: *Simpósio Brasileiro de Engenharia de Software (IVX SBES)*, João Pessoa, outubro, 2000.

SOUZA, S.R.S. *Avaliação do Custo e Eficácia do Critério Análise de Mutantes na Atividade de Teste de Software*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), São Carlos, SP, junho, 1996.

SOUZA, S.R.S.; MALDONADO, J.C.; FABBRI, S.C.P.F.; LOPES DE SOUZA, W. Mutation Testing Applied to Estelle Specifications. *Software Quality Journal*, vol. 8(04), to appear, artigo selecionado do 33rd Hawaii International Conference on System Sciences, Mini-track: Distributed Systems Testing, Maui, Hawaii, 4-7 de janeiro, 2000a.

SOUZA, S.R.S.; MALDONADO, J.C.; FABBRI, S.C.P.F.; MASIERO, PC. Statecharts Specifications: A Family of Coverage Testing Criteria. In: *Conferência Latino Americana de Informática (CLEI2000)*, Cidade de México, México, 18-22 de setembro, 2000b.

SOUZA, S.R.S.; MALDONADO, J.C.; FABBRI, S.C.P.F.; VINCENZI, A.M.; BARBOSA, E.F.; DELAMARO, M.E.; JINO, M. Introdução ao Teste de Software. *II Escola de Informática Norte da Sociedade Brasileira de Computação (EIN'2000 - SBC)*, Manaus-Belém, 18-20 de outubro, 2000c.

SUGETA, T. *Proteum-RS/ST: Uma Ferramenta para Apoiar a Validação de Especificações Statecharts Baseada na Análise de Mutantes*. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), São Carlos, SP, dezembro, 1999.

TANENBAUM, A.S. *Computer Networks*. 3. ed. New Jersey, Prentice-Hall Inc., 1996.

TAYLOR, R.N.; LEVINE, D.L.; KELLY, C.D. Structural Testing of Concurrent Programs. *IEEE Transaction Software Engineering*, 18(3), março, 1992.

- TEMPLEMORE-FINLAYSON, J.; RAFFY, J-L.; KRITZINGER, P.; BUDKOWSKI, S. A Graphical Representation and Prototype Editor for the FDT Estelle. In: *International Conference on Formal Description Technique for Distributed System and Communication Protocol and Protocol Specification, Testing and Verification*, Paris, França, 1998.
- THEES, J.; GOTZHEIN, R. The Experimental Estelle Compiler - Automatic Generation of Implementation from Formal Specifications. In: *2nd Workshop on Formal Methods in Software Practice (FMSP'98)*, Clearwater Beach, Flórida, USA, março, 1998.
- TURINE, M.A.S.; OLIVEIRA, M.C.F.; MASIERO, P.C. Designing Structured Hypertext with HMBS. In: *VIII International ACM Hypertext Conference*, Southampton, UK, p.241-256, 6-11 de abril, 1997.
- TURNER, K.J. *Using Formal Description Techniques – An Introduction to Estelle, Lotos and SDL*. Ed. by John Wiley & Sons, 1993.
- URAL, H. Test Sequence Selection Based on Static Data Flow Analysis. *Computer Communications*, v.10(5), p.234-242, outubro, 1987.
- URAL, H. Formal Methods for Test Sequence Generation. *Computer Communications*, 15(5), junho, 1992.
- URAL, H.; YANG, B. A Structural Test Selection Criterion. *Information Processing Letters*, vol.28, p.157-163, 1988.
- URAL, H.; YANG, B. A Test Sequence Selection Method for Protocol Testing. *IEEE Transaction on Communications*, 39(4), abril, 1991.
- VERGÍLIO, S.R.; MALDONADO, J.C.; JINO, M. Uma Estratégia para Geração de Dados de Teste. In: *Simpósio Brasileiro de Engenharia de Software (VII SBES)*, Rio de Janeiro, RJ, p.307-319, 1993.
- VILELA, P.R.S. *Critérios Potenciais Usos de Integração: Definição e Análise*. Tese (Doutorado), DCA/FEE/UNICAMP, Campinas, SP, Brasil, abril, 1998.
- VINCENZI, A.M.R.; MALDONADO, J.C.; BARBOSA, E.F.; DELAMARO, M.E. Interface Sufficient Operators: A Case Study. In: *Simpósio Brasileiro de Engenharia de Software (XIII SBES)*, p.373-391, Florianópolis, SC, outubro, 1999.
- VINCENZI, A.M.R.; MALDONADO, J.C.; BARBOSA, E.F.; DELAMARO, M.E. Unit and Integration Testing Strategies for C Programs Using Mutation-Based Criteria. In: *A Symposium on Mutation Testing for the New Century (Mutation 2000)*, San Jose, Califórnia, 6-7 de outubro, 2000.

VINCENZI, AM.R. *Orientação a Objeto: Definição e Análise de Recursos de Teste e Validação*. Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP), setembro, 2000.

VUONG, S.T.; JANSSEN, H., LU, Y.; MATHIESON, C.; DO, B. TESTGEN: an Environment for Protocol Test Suite Generation and Selection. *Computer Communications*, 17(04), abril, 1994.

VUONG, S.T.; LEE, S. L.; KIM, M.C. TESTVAL, a Tool for Protocol Test Validation and the Validation of a LAPB Test Suite as an Example. *Computer Communications*, 19, p.804-812, 1996.

WEYUKER, E.J. The Complexity of Data Flow Criteria for Test Data Selection. *Information Processing Letters*, vol.19(02), p.103-109, agosto, 1984.

WEYUKER, E.J; GORADIA, T.; SINGH, A. Automatically Generating Test Data from a Boolean Specification. *IEEE Transactions on Software Engineering*, vol.20(05), p.353-363, maio, 1994.

WONG, W.E. *On Mutation and Data Flow*. Tese (Doutorado) - Software Engineering Research Center - Purdue University, West Lafayette, Indiana, dezembro, 1993.

WONG, W.E.; MALDONADO, J.C.; DELAMARO, M.E.; MATHUR, A.P. Constrained Mutation in C Programs. In: *Simpósio Brasileiro de Engenharia de Software (VIII SBES)*, Curitiba, PR, p.439-452, outubro, 1994b.

WONG, W.E.; MALDONADO, J.C.; DELAMARO, M.E.; SOUZA, S.R.S. A Comparison of Selective Mutation in C and Fortran. In: *Workshop do Projeto Validação e Teste de Sistemas de Operação*, p.71-84, Águas de Lindóia, SP, janeiro, 1997.

WONG, W.E.; MALDONADO, J.C.; MATHUR, A.P. Mutation Versus All-Uses: An Empirical Evaluation of Cost, Strength, and Effectiveness. In: *Software Quality and Productivity - Theory, Practice, Education and Training*, Hong Kong, dezembro, 1994a.

YANG, C-S.; SOUTER, A.L.; POLLOCK, L.L. All-Du-Path Coverage for Parallel Programs. In: *International Symposium on Software Testing and Analysis (ISSTA'98)*, ACM-Software Engineering Notes, p.153-162, 1998.

YANG, R-D.; CHUNG, C-G. Path Analysis Testing of Concurrent Programs. *Information and Software Technology*, 34(1), janeiro, 1992.

ZHU, H.; HALL, P.A.V.; MAY, J.H.R. Software Unit Test Coverage and Adequacy. *ACM Computing Surveys*, v.29(4), dezembro, 1997.

Apêndice A – Descrição em Estelle do Protocolo

Bit_Alternante

Neste apêndice é apresentada a especificação do Protocolo Bit_Alternante descrita em Estelle. Essa descrição foi obtida no endereço:
<ftp://ftp.udel.edu/pub/grope/estelle-specs/>.

Informalmente, o protocolo *Bit-Alternante* pode ser descrito da seguinte forma:

“O protocolo é composto por um conjunto de usuários que trocam mensagens através de um meio de comunicação não confiável. Os dados a serem enviados são divididos em pacotes de dados e a cada pacote é associado um bit (0 ou 1), iniciando pelo bit 0. O pacote de dados é enviado e espera-se a confirmação do recebimento do pacote. Ao receber a confirmação, é verificado se é a mensagem de confirmação esperada. Em caso afirmativo, o pacote de dados na sequência é enviado; caso contrário, o pacote de dados é retransmitido. Após um tempo de espera, previamente estipulado, o pacote de dados é retransmitido. Quando um pacote de dados é recebido, é verificado se é o pacote esperado. Se for, o dado é formatado para envio ao usuário e envia-se uma mensagem confirmado o recebimento do pacote. Caso contrário ou após um tempo de espera, a mensagem de confirmação anterior é retransmitida”.

Na Figura A.1 é apresentada a arquitetura em Estelle do protocolo *Bit-Alternante*.

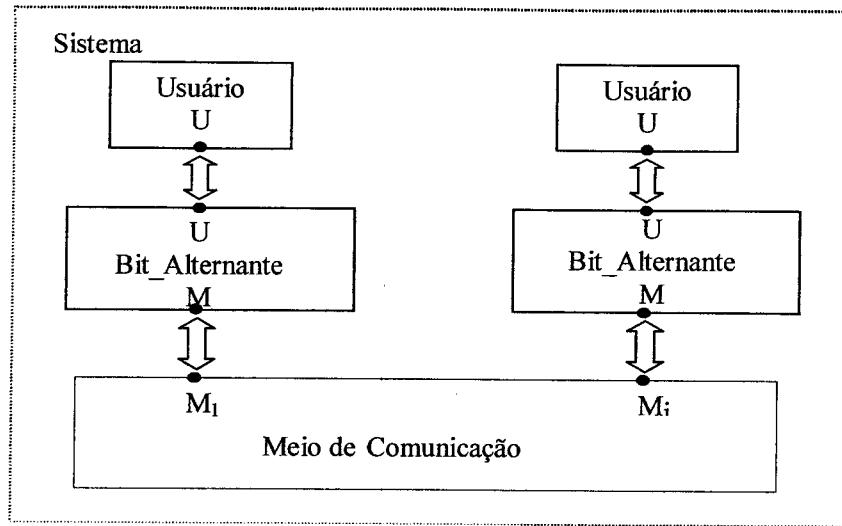


Figura A.1. Arquitetura em Estelle do Protocolo *Bit-Alternante*.

A seguir é apresentada a descrição em Estelle desse protocolo de comunicação.

```
{The classic Alternating Bit Protocol. Compilable!}

specification AB systemactivity;
timescale seconds;
{ This is the top level module body (specification)
  The specification has the attribute systemactivity
  and all its children ( user, ab, network ) are activityes.
  The time scale for delays is in seconds. }

const
  low  = 1; { Bounds of interaction point }
  high = 2; { subscripts. }
  Retrantlyme = 15; { retransmission time }

type
  Ceptype = low .. high;
  UDatatype = record { user data }
    size : integer;
    info : packed array [1..10] of char
  end;
  Seqtype   = 0 .. 1 ; { sequence number range }
  Idtype    = (DATA, ACK);
  Ndatatype = record
    Id: Idtype;        { type of message }
    Conn: Ceptype;     { cep of sender }
    Data: UDatatype;   { user data }
    Seq: Seqtype;      { sequence number }
  end;

{ Channel definitions for communication between the activities}

channel Uaccesspoint(User,Provider);
  by User:
    SENDrequest (Udata: UDatatype);
    RECEIVErequest;
  by Provider:
    RECEIVEresponse(Udata: UDatatype);

channel Naccesspoint(User,Provider);
  by User:
```

```

        DATArequest(Ndata: Ndatatype);
by Provider:
        DATAresponse(Ndata: Ndatatype);

{ Module header definitions }

module Usertype activity (Connendptid : Ceptype);
    ip U : Uaccesspoint(User) common queue;
end;

module Alternatingbittype activity (Connendptid : Ceptype);
    ip {interaction point list}
        U: Uaccesspoint(Provider) common queue;
        N: Naccesspoint(User) individual queue; end;

{ The module has two interaction points named U and N;
  the roles of the module are named:
  Provider with respect to U, and
  User with respect to N.

  Notice that there is an individual queue associated with the
  interactionpoint N. If the queue would have been common (with the queue of
  U) a SENDrequest interaction output by the user while the module is the in
  the ACKWAIT state would lead to a deadlock (since the module would not be
  able to activity a network interaction put in the same queue) }

module Networktype activity;
    ip N : array[Ceptype] of Naccesspoint(Provider) individual queue; end;

{ Body definitions for modules }

body Networkbody for Networktype;
{ There is no network body given in the original ISO document. In addition,
there is no routing information given to the network module other than the
originator. To resolve these problems, the messages are passed to exactly
one other body. }

trans
when N[1].DATArequest
    name req1a :
begin output N[2].DATAresponse(Ndata) end;
when N[2].DATArequest
    name req2a :
begin output N[1].DATAresponse(Ndata) end;
trans
when N[1].DATArequest
    name req1b :
begin output N[2].DATAresponse(Ndata) end;
when N[2].DATArequest
    name req2b :
begin output N[1].DATAresponse(Ndata) end;
trans
when N[1].DATArequest
    name req1c :
begin output N[2].DATAresponse(Ndata) end;
when N[2].DATArequest
    name req2c :
begin output N[1].DATAresponse(Ndata) end;
trans
when N[1].DATArequest
    name req1d :
begin output N[2].DATAresponse(Ndata) end;
when N[2].DATArequest
    name req2d :
begin output N[1].DATAresponse(Ndata) end;
trans
when N[1].DATArequest
    name req1e :

```

```

begin output N[2].DATAreponse(Ndata) end;
when N[2].DATArequest
  name req2e :
  begin output N[1].DATAreponse(Ndata) end;
trans
  when N[1].DATArequest
    name req1f :
    begin output N[2].DATAreponse(Ndata) end;
  when N[2].DATArequest
    name req2f :
    begin output N[1].DATAreponse(Ndata) end;
trans
  when N[1].DATArequest
    name req1g :
    begin output N[2].DATAreponse(Ndata) end;
  when N[2].DATArequest
    name req2g :
    begin output N[1].DATAreponse(Ndata) end;
trans
  when N[1].DATArequest
    name req1h :
    begin output N[2].DATAreponse(Ndata) end;
  when N[2].DATArequest
    name req2h :
    begin output N[1].DATAreponse(Ndata) end;
trans
  when N[1].DATArequest
    name loss1 :
    begin end;
  when N[2].DATArequest
    name loss2 :
    begin end;
end {Networkbody};

body Userbody for Usertype;

{ There is no body given for the User in the ISO document. This
 user merely checks whether it is user one or user two. It then
 starts sending and then receiving if it is user one or receiving
 and then sending if it is user two. }

state SEND, WAIT, RECV;

var udata : udatatype;

initialize
  to SEND
  provided Connendptid = 1
    begin end;
  to WAIT
  provided Connendptid = 2
    begin end;

trans
  from SEND
  to WAIT
  delay(30)
    name SENDIT:
    begin
      udata.size := 5;
      udata.info := 'Hello      ';
      output U.SENDrequest(udata);
    end;

```

```

trans
  from WAIT
  to RECV
  delay(5,5)
    name REQIT:
    begin
      output U.RECEIVErequest
    end;

trans
  from RECV
  to SEND
  when U.RECEIVEresponse
  name RECVIT:
  begin
  end;

end {Userbody};

{ The body for alternating bit is defined below: }

body Alternatingbitbody for Alternatingbittype;

type
  Msgtype =
    record { record introduces a data structure }
      Msgdata: UDatatype; { to be ... }
      Msgseq: Seqtype
    end;
  Buffertype =
    record
      size : integer;
      data : array [0..20] of Msgtype
    end;
var
  Sendbuffer, Recvbuffer: Buffertype;
  Sendseq,Recvseq: Seqtype;
  P,Q: Msgtype;
  B: Ndatatype;
state ACKWAIT, ESTAB; { state definition part }
stateset { state-set-definition-part }
  EITHER = [ACKWAIT, ESTAB];

function Ackok(Nd: Ndatatype): boolean;
  {notice that a function shall be demonstrably pure }
begin
  Ackok := (Nd.Id = ACK) and (Nd.Seq = Sendseq);
end;

procedure Copy(var ToData: UDatatype; Fromdata: UDatatype);
{ procedure provided by implementer: copy a user data variable }
begin ToData := Fromdata end;

{Note: Estelle is not sensitive to the case of letters; therefore,
subsequent references to "copy" rather than "Copy" refer to this procedure.
}

procedure Empty(var Data: UDatatype);
{ procedure provided by implementer:
  initialize a variable holding user data to the value no user data  }
begin Data.size := 0; Data.info := '???????????' end;

procedure Formatdata(Msg: Msgtype; var B: Ndatatype);
begin
  B.Id      := DATA;
  B.Conn    := Connendptid;

```

```

{ connection reference given in the instantiation }
copy( B.Data, Msg.Msgdata); { copy data }
B.Seq      := Msg.Msgseq;
end;

procedure Formatack (Msg: Msgtype; var B: Ndatatype);
begin
B.Id      := ACK;
B.Conn    := Connendptid;
empty (B.Data) ; { no data for an ACK }
B.Seq      := Msg.Msgseq;
end;

{ two variables of type buffertype are used to hold messages ( of type
msgtype):
  Sendbuffer for sending, Receivebuffer for receiving.
  The following procedure and functions are used
  manipulate buffertype variables }

procedure Emptybuf(var Buf: Buffertype);
{ procedure provided by implementer : set a buffer to empty i.e. contains no
messages}
var i : integer;
begin
Buf.size := 0;
for i := 0 to 20 do
begin
Buf.data[i].Msgseq := 0;
Buf.data[i].Msgdata.size := 0;
Buf.data[i].Msgdata.info := '??????????'
end
end;

procedure Store(var Buf: Buffertype; Msg: Msgtype);
{ procedure provided by implementer :
  store a message into a buffertype
  variable such that the messages can be
  retrieved or removed in a FIFO manner}
begin
Buf.data[Buf.size] := Msg;
Buf.size := Buf.size + 1
end;

procedure Remove(var Buf: Buffertype);
{ procedure provided by implementer :
  remove the first message}
var i : integer;
begin
for i := 1 to Buf.size do
  Buf.data[i - 1] := Buf.data[i];
Buf.size := Buf.size - 1
end;

function Retrieve(Buf: Buffertype): Msgtype;
{ function provided by implementer :
  retrieve the first message and return it;
  the message is not removed }
begin
Retrieve := Buf.data[0]
end;

function bufferempty(Buf:Buffertype) : boolean;
{ function provided by implementer : check if a buffer contains a message}
begin
bufferempty := Buf.size = 0
end;

```

```

procedure Incsendseq;
begin
  Sendseq := (Sendseq + 1) mod 2
end;

procedure Increcvseq;
begin
  Rcvseq := (Rcvseq + 1) mod 2
end;

initialize { initialization-part of the alternating bit activity }

to ESTAB { initialize major state variable to ESTAB }
begin { initialize variables }
  Sendseq := 0;
  Rcvseq := 0;
  Emptybuf (Sendbuffer); { implementation specific }
  Emptybuf (Recvbuffer); { implementation specific }
end;

trans { transition-declaration-part of the alternating bit activity }

from ESTAB
to ACKWAIT
when U.SENDrequest
  name send:
  begin
    copy(P.Msgdata,Udata);
    P.Msgseq := Sendseq;
    Store(Sendbuffer,P);
    Formatdata(P,B);
    output N.DATAREquest(B);
  end;

from ESTAB
to same
when N.DATAREsponse
provided Ndata.Id = ACK
  name tossack:
  begin
    { purge the queue to prevent deadlock }
  end;

from ESTAB
to same
when U.RECEIVErequest
provided not bufferempty(Recvbuffer)
  name recrespe:
  begin
    {transition 2}
    Q := Retrieve(Recvbuffer); { retreive received message }
    output U.RECEIVEResponse(Q.Msgdata);
    Remove(Recvbuffer) { remove message from receiving buffer }
  end;

from ACKWAIT
to same
when U.RECEIVErequest
provided not bufferempty(Recvbuffer)
  name recrespa:
  begin
    {transition 2}
    Q := Retrieve(Recvbuffer); { retreive received message }
    output U.RECEIVEResponse(Q.Msgdata);
    Remove(Recvbuffer) { remove message from receiving buffer }
  end;

from ACKWAIT
to ACKWAIT

```

```

delay (Retrantime)
  name retrans:
  begin
    P := Retrieve(Sendbuffer);           { transition 3 }
    Formatdata(P,B);                   { retreive message to be retransmitted }
    output N.DATArequest(B);          { format a network message }
  end;

from ACKWAIT
to ESTAB
when N.DATAreponse
provided Ackok(Ndata)
  name goodack:
  begin
    Remove(Sendbuffer);              { transition 4 }
    Incsendseq;
  end;

from ACKWAIT
to same
when N.DATAreponse
provided (Ndata.id = ACK) and (Ndata.seq <> Sendseq)
  name badack:
  begin
    { to prevent a livelock: a USENDrequest was sent
      and the alternatingbit instance receives a bad ack
      N.DATAreponse ack. }
  end;

from ESTAB
to same
when N.DATAreponse
provided Ndata.Id = DATA
  name getdatae:
  begin
    copy (Q.Msgdata,Ndata.Data);       { transition 5 }
    Q.Msgseq := Ndata.Seq;
    Formatack(Q,B);
    output N.DATArequest(B);
    if Ndata.Seq = Recvseq then
      begin
        Store(Recvbuffer,Q);
        Increcvseq
      end
  end;

from ACKWAIT
to same
when N.DATAreponse
provided Ndata.Id = DATA
  name getdataa:
  begin
    copy (Q.Msgdata,Ndata.Data);       { transition 5 }
    Q.Msgseq := Ndata.Seq;
    Formatack(Q,B);
    output N.DATArequest(B);
    if Ndata.Seq = Recvseq then
      begin
        Store(Recvbuffer,Q);
        Increcvseq
      end
  end;
end; { of the Alternatingbitbody}

modvar
  { module-variable-declaration-part of the specification }

```

```
User : array[CEPtype] of Usertype;
Alternatingbit: array [Ceptype] of Alternatingbittype;
Network : Networktype;

initialize { initialization-part of the specification }

begin { module initialization }
init Network with Networkbody;
all Cep : Ceptype do
begin
init User[Cep] with Userbody(Cep);
init Alternatingbit[Cep] with Alternatingbitbody(Cep);
connect User[Cep].U to Alternatingbit[Cep].U;
connect Alternatingbit[Cep].N to Network.N[Cep];
end;

end; { of module initialization within the
specification's initialization-part }

end. { End of specification; the specification has no transition part }
```