Estratégia para teste de Máquinas Finitas de Estados Estendidas

Flávio Rogério Uber *Eliane Martins*
Instituto de Computação - Universidade Estadual de Campinas - UNICAMP
e-mail: {flavio.uber, eliane}@dcc.unicamp.br
Campinas - SP

Resumo

Este artigo apresenta um método para melhorar a estratégia de geração de dados por parte da CONDADO [Sab98], uma ferramenta desenvolvida para testes baseados em máquinas finitas de estados estendidas(MFEE). Para isso pretende-se considerar os predicados associados às transições da máquina com o uso dos testes de domínios. Com essa modificação será melhorado o potencial para detecção de falhas, e ainda, será reduzido o número de casos de teste correspondentes a caminhos não executáveis.

Abstract

This paper presents a method to enhance the test generation strategy provided by CONDADO [Sab98], a tool developed for test generation based on extended finite state machines (EFSM). For that modification we will use domain testing taking into account the predicates of the state machine. With that modification we intend to increase fault detection capability and decrease number of test cases corresponding to non-executable paths.

1. Introdução

Para a representação de sistemas reativos, onde pode-se definir a ordem com que as interações ocorrem, é comum a utilização de máquinas finitas de estados (MFE). Assim, é comum o seu uso para representação de protocolos de comunicação e outros sistemas de tempo real, como caixas automáticos de bancos. No entanto, é comum que as mudanças de estados nesses sistemas estejam associadas não somente a um evento, mas também a uma condição (predicado). Como a MFE em seu modelo original não considera a existência de predicados, é necessário o uso das MFEEs, que consideram, além dos predicados, variáveis de contexto e ações (veja seção 2).

Devido à importância dos sistemas reativos, é necessário que existam técnicas que possam realizar testes a partir dessa especificação, a fim de melhorar a qualidade desses sistemas, e, consequentemente, sua confiabilidade.

Nesse intuito foi desenvolvida a CONDADO, uma ferramenta que implementa testes para a parte de controle e de dados somente com a utilização de técnicas caixa preta. Isso a difere da maioria dos métodos desenvolvidos para testes de MFEE, que utilizam técnicas caixa branca para a parte de dados [Ura91] [Bou96] [Bou97].

No entanto, como as demais técnicas, a CONDADO também enfrenta o problema da executabilidade dos caminhos a serem exercitados na implementação. A executabilidade está diretamente relacionada aos predicados que fazem parte de determinadas transições. Portanto existe a necessidade de se considerar esses predicados como forma de otimizar a geração dos casos de teste.

A seção 2 traz algumas definições e abordagens de teste para máquinas finitas de estados estendidas. A seção 3, faz algumas considerações sobre executabilidade, um problema característico nesse tipo de teste. A seção 4 descreve a CONDADO e mostra um exemplo de caso de teste não executável gerado por ela. A seção 5 traz os conceitos básicos de teste de domínio para identificá-la como uma técnica que considera os predicados para a geração dos testes. Finalmente a seção 6 dá um panorama geral do trabalho em desenvolvimento, bem como possíveis problemas que serão encontrados.

2. Máquinas Finitas de Estados Estendidas (MFEE)

Uma máquina finita de estados (MFE) é um modelo que serve para mostrar a variação de estados de um sistema ao longo do tempo. A máquina pode mudar de estado em resposta a uma

entrada (ou evento), podendo produzir uma saída. As mudanças de estado são denominadas de transições e as saídas são produzidas por ações.

Nesta seção será apresentada uma definição de máquinas finitas de estados estendidas, que diferem das MFEs pela inclusão de variáveis de contexto, predicados e ações. Em seguida serão apresentadas algumas abordagens de teste utilizadas para esse tipo de modelo.

2.1 Definição

Pode-se definir máquinas finitas de estados estendidas (MFEE) formalmente como uma 8-tupla <S, s0, I, O, V, P, A, g> onde:

S - conjunto não vazio de estados; s0 - estado inicial; I - conjunto finito de entradas; O - conjunto finito de saídas; V - conjunto de variáveis; P - conjunto de predicados que operam sobre as variáveis; A - conjunto de ações relacionadas às variáveis; g - função de transição de estado definida como g: S x I x P(V) (S x O x A(V)).

Uma transição pode ser dividida em duas partes: a parte da condição e a parte da ação. A parte da condição contém um evento de entrada e/ou um predicado. O predicado pode ser definido como uma expressão booleana que pode envolver as variáveis bem como os parâmetros das entradas. A parte da ação pode conter eventos de saídas e um número de comandos envolvendo as variáveis. Uma transição é disparada quando a condição é satisfeita, fazendo com que a ação correspondente à transição seja executada e a MFEE mude para o estado destino. A ação executada pode alterar os valores das variáveis associadas à transição.

2.2 Testes Baseados em MFEE

A seguir serão apresentadas as abordagens existentes para realização dos testes em uma MFEE [Boc97].

1. Separar fluxo de controle e fluxo de dados: Nesta abordagem, os aspectos controle e dados são separados na especificação. O fluxo de controle é representado por uma MFE, onde técnicas de teste de transição de estados são aplicadas. Através de critérios de testes orientados ao fluxo de dados, (como por exemplo todos-DU-caminhos, onde o objetivo é testar todo caminho livre de laços entre uma definição global de uma variável e todos os seus usos[Vil97]), testa-se os parâmetros das primitivas de entrada e saída e as variáveis de contexto [Bou97].

2. Transformar a especificação na forma normal: Uma especificação contendo apenas as chamadas transições na forma normal, não contêm instruções que influenciam o fluxo de controle do protocolo, tais como instruções condicionais (*if, case*) e instruções de repetição (*while, repeat*). Para transformar uma especificação em uma equivalente na forma normal cria-se uma nova transição para cada caminho distinto dentro de uma ação[Ura91].

3. Expandir a MFEE: Uma MFEE pode ser vista como uma notação compactada de uma MFE, já que essa transformação pode ser feita eliminando-se as variáveis de contexto através da criação de um conjunto de novos estados e novas transições. Os novos estados são formados pela combinação dos estados da MFEE com os valores das variáveis de contexto. Assim, a habilitação de uma transição dependerá somente do estado corrente e da entrada. Essa abordagem é chamada *unfolding*. Sua utilização pode levar a uma explosão de estados, o que dificulta sua manipulação. Além disso, quando uma variável não tem um domínio finito não é possível aplicar *unfolding* a uma MFEE [Cas87].

4. Transformar a especificação em gramática: A transformação da especificação em gramática permite representar todos os aspectos do protocolo (controle, dados, predicados e ações) usando uma única notação. A partir disso, técnicas de teste baseadas em gramática podem ser aplicados [Ura83].

Para a ferramenta CONDADO, foi utilizada uma abordagem sugerida por Poston [Pos96], que combina testes dirigidos aos eventos e testes dirigidos aos dados, apenas com a utilização de técnicas caixa preta.

Os testes dirigidos aos dados se concentram em encontrar falhas de manipulação de valores de dados como números, listas ou strings e falhas na manipulação de estruturas de dados como arrays,

registros ou arquivos. Este tipo de teste é dividido em cinco categorias: análise de limite, de partição, de domínio, de lista e de sintaxe [Pos96]. As categorias de testes dirigidos a dados implementadas pela CONDADO são análise de partição e análise de sintaxe.

A análise de partição é considerada quando trata-se o conjunto dos valores possíveis de um sistema como domínio. Este domínio é dividido em duas classes, uma para entradas válidas e outra para entradas inválidas. Em outros termos, se um caso de teste escolhido em uma determinada classe é capaz de encontrar uma determinada falha, assume-se que todos os outros casos de teste pertencentes à mesma classe são capazes de encontrar a mesma falha.

A análise de sintaxe trata do formato da entrada considerada. Por exemplo, um número de telefone pode ser definido como sendo a composição de três números de 0 a 9, um traço e mais quatro números de 0 a 9.

A seção 4 apresenta a ferramenta CONDADO com mais detalhes de seu funcionamento e de suas características.

3. CONDADO

A ferramenta CONDADO [Sab98] foi desenvolvida com a intenção de implementar uma estratégia que proporcionasse uma cobertura dos aspectos de controle (relacionado às transições) e dados (relacionado às variáveis e parâmetros) de um protocolo de comunicação a partir de técnicas de testes caixa preta, com a utilização dos testes de transição de estados, dos testes de sintaxe e dos testes de partição de equivalência.

O usuário pode impor algumas restrições à CONDADO. Essas restrições se referem por exemplo, às transições que serão testadas, o que é muito útil para testar funcionalidades específicas do protocolo. Outra possível restrição é quanto ao número de vezes que um ciclo será executado.

Estas restrições deixam a cargo do usuário a preocupação acerca da executabilidade de um determinado caminho. Um caminho é não executável se não existir um conjunto de valores para as variáveis de entrada, parâmetros e variáveis globais que causam a sua execução [Ver97a]. Caso o usuário não se preocupe com isso, corre-se o risco da ferramenta gerar casos de teste correspondentes a caminhos não executáveis, como será mostrado a seguir.

Considere a máquina de estados dada pela figura 1.

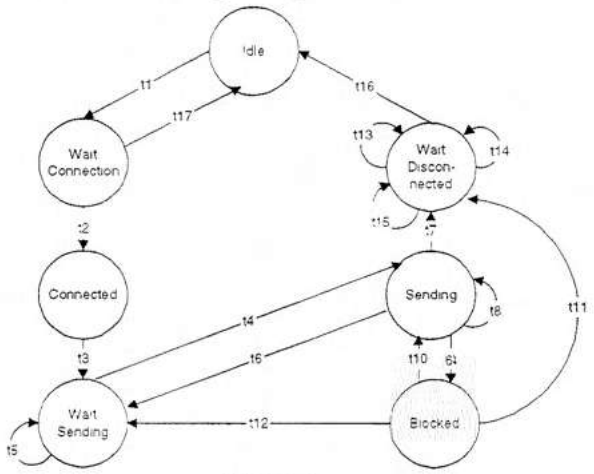

Figura 1

Quando é requisitada a geração de todos os casos de teste para as transições t5 e t12, serão obtidos oito casos de teste. Tomemos como exemplo o caso de teste composto pelas seguintes transições: t1, t2, t3, t4, t6, t4, t9, t10, t9, t12, t5, t4, t7, t13, t14, t15, t16. A única transição, para esse caso de teste, que incrementa a variável *number* é a transição t4. Portanto quando a transição t7 for atingida, t4 terá sido exercitada 3 vezes, e *number* terá valor 3. Para a habilitação de t7, é necessário que *number* seja igual a *number_of_segment*. Logo, trata-se de um caso de teste correspondente a um caminho não executável, já que *number* é uma variável inicializada em 0 e incrementada nas

transições t4 e t8 e *number_of_segment* é uma variável que tem seu domínio de entrada restrito aos valores 2, 4 e 8.

Como atualmente a CONDADO gera valores aleatórios (dentro do domínio de entrada) para as variáveis de entrada, torna-se uma tarefa difícil para o usuário restringir o número de vezes que a transição t4 deve ser exercitada para habilitar a transição t7.

Para minimizar este tipo de problema os predicados devem ser considerados para a geração dos casos de teste. Isso pode ser feito com a utilização do teste de domínio (veja seção 4).

4. Teste de Domínio

Os testes de domínio foram desenvolvidos inicialmente como uma técnica de teste caixa branca, com o intuito de se gerar casos de teste que façam com que o maior número possível de caminhos de um programa sejam seguidos. No entanto, seus conceitos são aplicáveis para teste caixa preta, quando a especificação é feita através de alguns modelos, como por exemplo uma máquina finita de estados estendida [Bei95].

Nesta seção são apresentados alguns conceitos básicos imprescindíveis para a compreensão da técnica de acordo com [Whi80]; em seguida são apresentadas algumas estratégias de teste de domínio que foram desenvolvidas.

4.1. Conceitos Básicos

Alguns dos principais termos empregados em testes de domínios são:

Domínio: Conjunto de dados de entrada que satisfazem a uma condição de caminho.

Condição de caminho: É o conjunto de condições que deve ser satisfeito pelos dados de entrada para que um caminho de controle seja escolhido.

Caminho de controle: É cada um dos caminhos de um programa. Se existem dados que fazem com que esse caminho seja seguido, ele é também um caminho de execução.

Predicado: Uma condição existente em um programa, associada com um valor verdadeiro ou falso, fazendo com que apenas uma das instruções seja seguida.

Segmento de borda: O limite de cada domínio é determinado por interpretações do predicado na condição de caminho e consiste de segmentos de borda. Cada segmento pode ser aberto (se os operadores relacionais forem $<$, $>$ ou $\neq$) ou fechado (se os operadores relacionais forem $=$, $\geq$, $\leq$).

Falha de domínio: Um caminho contém uma falha de domínio se uma falha em algum predicado provoca um deslocamento no segmento de borda ou se um predicado tem um operador relacional incorreto.

O teste de domínio particiona o espaço de entrada de um programa em um conjunto de domínios, sendo que cada domínio corresponde a um caminho e consiste de dados que fazem com que um caminho seja executado [Whi80].

A existência de predicados fazem com que a execução de um programa siga um determinado caminho [Bei90]. Um predicado é uma expressão composta de operadores relacionais ($<$, $>$, $=$, $\geq$, $\leq$, $\neq$). Por exemplo, se um programa possui uma instrução do tipo: ***se a>b então a:=b senão b:=a***, a condicional a>b particiona o programa em dois domínios.

A intenção portanto é fazer com que se consiga testar a maior quantidade possível dos caminhos. No entanto, é difícil testar todos os valores que correspondem a um determinado domínio. No exemplo anterior para que a instrução **a:=b** seja executada, é impossível que todos os valores tais que **a** seja maior que **b** sejam testados. Assim, o teste de domínio propõe que sejam testados valores críticos, ou seja, nos limites de um domínio.

4.2. Algumas estratégias

Ao longo do tempo foram propostas várias metodologias para teste de domínio. A tabela abaixo sumariza os principais itens abordados por algumas dessas técnicas bem como as diferenças entre elas.

A estratégia de teste de domínio tem o intuito de detectar falhas no fluxo de controle de um programa com a detecção das falhas de domínio.

A primeira delas foi proposta por White e Cohen [Whi80], na qual considera-se que o fluxo de controle é determinado pela combinação de expressões lógicas. Essa combinação é determinada em um predicado, pela possível ocorrência de operadores AND e OR.

White e Cohen propõem que no caso de predicados com desigualdade (<, >, ≥, ≤,), sejam escolhidos dois pontos ON, ou seja, dois pontos sobre a borda de um domínio e um ponto OFF, isto é, um ponto que está a uma pequena distância ε, fora do domínio. Se a borda é aberta ocorre o contrário: o ponto OFF pertence à borda do domínio sendo testado e os pontos ON estão a uma distância ε, já que a borda não pertence ao domínio.

Nessa estratégia os pontos são escolhidos de tal forma que a projeção do ponto OFF sobre a linha onde se encontram os pontos ON deve estar situada entre esses dois pontos. Para que a projeção do ponto OFF satisfaça esta condição White e Cohen propõem que os pontos ON sejam escolhidos nos extremos da reta que delimita o domínio.

Será obtido um espaço bi-dimensional, no caso de um predicado depender de duas variáveis de entrada. Conforme existirem predicados com mais variáveis teremos um espaço até mesmo N-dimensional. Esta estratégia pode ser estendida bastando que sejam escolhidos N pontos ON.

No entanto, a estratégia apresentou alguns pontos indesejados, como o problema da correção coincidente (*coincidental correctness*), ou seja, ignorou-se o fato de que mesmo seguindo um caminho errado, o programa pode produzir valores correspondentes à execução do caminho correto.

Dois anos mais tarde, em 1982, Clarke *et al* [Cla82], consideraram o trabalho anterior e propuseram algumas alterações que melhorariam a aplicabilidade da técnica. É proposta uma possível solução para correção coincidente: ao se testar os pontos ON de um domínio, caso sejam obtidos resultados corretos, deve-se verificar o domínio adjacente. Em caso de igualdade no resultado investiga-se a possibilidade da borda pertencer a este domínio adjacente.

Em 1994, Jeng e Weyuker [Jen94] propuseram uma estratégia simplificada para teste de domínio, onde foram removidas algumas limitações de estratégias anteriores (veja tabela).

Em 1998, Hajnal e Fogács [Haj98] propuseram um algoritmo que segue a mesma linha de [Jeng94]. Esse algoritmo parte de um único valore de entrada I_0 que atravessa um caminho conhecido P, e consegue os pontos ON e OFF para esse caminho (se possível), e os demais caminhos, com seus pontos ON e OFF.

Proposta	Desigualdade	Igualdade / diferença	Limitações	Espaço de entrada	Correção coincidente
White e Cohen	Requer N[1] pontos ON e um ponto OFF	Requer N pontos ON e dois pontos OFF	Não prevê elementos de entrada *array* e nem a ocorrência de predicados não lineares[2]	Apenas espaço de entrada contínuo	Considera a não ocorrência
Clarke, Hassel e Richardson	Requer V[3] pontos ON e V pontos OFF		Apenas predicados lineares	Pode-se adaptar a estratégia para espaços discretos	Propõe uma possível solução para correção coincidente
Jeng e Weyuker	Requer um ponto ON e um ponto OFF	Requer um ponto ON e dois pontos OFF	Considera *arrays* e predicados não lineares	Considera espaços discretos, já que o ponto ON não precisa estar sobre a borda	Considera a não ocorrência
Hajnal e Forgács	Requer um ponto ON e um ponto OFF	Requer um ponto ON e dois pontos OFF	Dificuldade de Implementação	Considera espaços discretos	Considera a não ocorrência

[1] Dimensão do espaço [2] Pred. linear possui variáveis com potência=1 [3] Núm. de vértices da borda

5. Estado Atual

Tendo como base esta dificuldade na análise dos predicados de uma máquina e as definições de teste de domínio, o cenário atual se caracteriza pelo estudo das abordagens de teste de domínios para que seja escolhida a mais adequada para o problema em questão.

Sabe-se de antemão que a aplicação dos testes de domínios para resolução dos predicados não é trivial. [Ver97b], após um estudo comparativo, aponta essa técnica como poderosa, mas de difícil automatização.

Bem é verdade que a intenção final do trabalho é a implementação dos testes de domínios na CONDADO, mas atualmente o que se deseja é um estudo que deverá produzir como resultado final um algoritmo descritivo para posterior implementação, levando-se em conta os diversos tipos de predicado [Cas87].

Referências Bibliográficas

[Bei90] Beizer, Boris. *Software Testing Techniques*. 1990.

[Bei95] Beizer, Boris. *Black-Box Testing. Techniques for Functional Testing of Software and Systems* . 1995.

[Boc97] Bochmann, G. v. & Dssouli, R. *Test Development for Distributed Systems: Towards Automation*. XV Simpósio Brasileiro de Redes de Computadores, São Carlos – SP. 1997.

[Bou96] Bourhfir, C.; Dssouli, R. & Aboulhamid E. M. *Automatic Test Generation for EFSM-based Systems*. http://www.iro.umontreal.ca/labs/teleinfo/PubListIndex.html. 1996.

[Bou97] Bourhfir, C.; Dssouli, R.; Aboulhamid E. M. & Rico, N. *Automatic Executable Test Case Generation for Extended Finite State Machine Protocols*. www.iro.umontreal.ca/labs/teleinfo/PubListIndex.html. 1997.

[Cas87] Castanet, R & Sijelmasi, R. *Methods and Semi-Automatic Tools for Preparing Distributed Testing*. IFIP. 1987.

[Cla82] Clarke, Lori A., Hassell, J. & Richardson, Debra J. *A Close Look at Domain Testing*. IEEE Trans. On Softare Engineering, vol.SE-8, Jul/82, pp. 380-390.

[Haj98] Hajnal, A. & Forgács, I. *An Applicable Test Data Generation Algorithm dor Domain Errors*. ISSTA. 1998.

[Jen94] Jeng, B. & Weyuker, Elaine J. *A Simplified Domain Testing Strategy*. ACM Trans. On Software Engineering and Methodology, 3(3), Jul/94, pp. 254-270.

[Pos96] Poston, R. M. *Automating Specification-based Software Testing*. IEEE Computer Society Press. 1996.

[Sab98] Sabião, Selma B. *Um Método para Geração de Testes Baseado em Máquina Finita de Estado Estendida combinando técnicas de teste caixa preta*. Tese de Mestrado. Universidade Estadual de Campinas, Campinas. 1998.

[Ura91] Ural H. & Yang, B. *A Test Sequence Selection Method for Protocol Testing*. IEEE Transaction On Communications, Vol 39, No. 4. 1991.

[Ura83] Ural, H. & Probert, R. L. *User-guided Test Sequence Generation*. Protocol Specification, Testing and Verification III. 1983.

[Ver97a] Vergílio, S. R. *Critérios Restritos de Teste de Software: Uma Contribuição para Gerar Dados de Teste Mais Eficazes*. Tese de Doutorado. FEEC. UNICAMP. 1997.

[Ver97b] Vergílio, S. R., Maldonado, J. C. & Jino, M. *Resultados de um Experimento de Aplicação de Diferentes Técnicas de Geração de Dados de Teste*. Workshop de Projeto, Validação e Teste de Sistemas de Operação. Janeiro, 1997.

[Vil97] Vilela, P. R. S., Maldonado, J. C., Jino, M. & Chain, M. C. *Uma Visão Sobre o Teste Estrutural Baseado em Análise de Fluxo de Dados*. Workshop do Projeto Validação e Testes de Sistemas de Informação. Águas de Lindóia, 1997.

[Whi80] White, Lee J. & Cohen, Edward I. *A Domain Strategy for Computer Program Testing*. IEEE Trans. On Software Engineering, vol.SE-6, mai/80, pp.247-257.