Adaptando Verificadores de Modelos para a Geração Automática de Objetivos de Teste para Sistemas Reativos *

Daniel Aguiar da Silva[1], **Patrícia D. L. Machado**[1]

[1]Grupo de Métodos Formais – DSC/Universidade Federal de Campina Grande – Brazil

{daguiar,patricia}@dsc.ufcg.edu.br

Abstract. *Test purposes are properties that specify behaviours an implementation under test (IUT) should exhibit. Such properties are usually specified through analyses performed over a model for the IUT. When performed over formal specifications, these analyses can be automated through formal techniques, such as model checking. This paper presents the adaption of the algorithms of a model checker in order to provide a new approach for model analyses aimed at the test purpose generation.*

Resumo. *Objetivos de teste são artefatos utilizados como guias da geração de casos de teste. Representam propriedades que definem os comportamentos a serem exibidos por uma implementação sob teste (IST). São especificados com base em análises sobre um modelo da IST. Quando realizadas sobre especificações formais, estas análises podem ser automatizadas por técnicas rigorosas, como a verificação de modelos. Este artigo apresenta uma adaptação dos algoritmos de um verificador de modelos para a realização de análises voltadas para a geração de objetivos de teste.*

1. Introdução

Sistemas reativos são caracterizados pela sua interação com o ambiente ao qual estão inseridos. Ambientes complexos geralmente exigem interações remotas, caracterizadas por distribuição, não-determinismo e concorrência, assim como a continuidade indefinida de sua execução. O correto desenvolvimento destes sistemas demanda grandes esforços nas atividades de especificação, implementação e de verificação e validação (v&v).

A alta complexidade envolvida na especificação de sistemas com tais características exige grande esforço para a produção de modelos corretos. Esta atividade tem sido auxiliada pela utilização de formalismos (e.g. [Jensen 1992, Hoare 1985]) que permitem a especificação precisa dos sistemas. Neste contexto, a verificação de modelos [Clarke et al. 1999] é uma importante técnica de v&v utilizada na correção das especificações formais, geralmente definidas como modelos para os sistemas. A verificação de modelos é uma rigorosa técnica de verificação automática de propriedades sobre modelos formais. As propriedades a serem verificadas são definidas por meio de fórmulas especificadas em formalismos de lógica temporal, como a CTL [Clarke et al. 1999]. A verificação é realizada por meio de algoritmos de busca sobre o espaço de estados de um modelo. O espaço de estados é uma representação abstrata de

*Este trabalho é financiado pela FAPESQ/CNPq (Projeto 060/03 e Processo CNPq 550466/2005-3). O primeiro autor é financiado pela CAPES.

menor nível do modelo e deve representar todos os estados do sistema e de suas transições por um grafo, como por exemplo sistemas de transições rotuladas (STR). A busca deve responder se o modelo satisfaz ou não a propriedade, retornando um caminho sobre o grafo como exemplo ou contra-exemplo, respectivamente.

Apesar de sua contribuição na produção de sistemas mais robustos e confiáveis, a verificação de modelos não assegura que a implementação dos sistemas corresponda de maneira fiel ao modelo especificado, pois, faltas podem ser inseridas durante a etapa de implementação. Portanto, técnicas de v&v devem ainda ser aplicadas sobre a implementação a fim de detectar possíveis faltas.

Dentre as técnicas de v&v aplicadas à implementação, teste é a mais popular. Sua aplicação é caracterizada pela realização de experimentos a partir da interação direta com o *software*. Para sistemas reativos estas interações são realizadas por meio de pontos de observação e controle (PCO's) [Jard and Jéron 2004], que são baseados em entradas e saídas do sistema, e produzem resultados que são então monitorados, permitindo posterior análise para validação. Entretanto, sua aplicação a sistemas reativos que operam em ambientes complexos pode tornar-se cara e ineficiente devido ao não-determinismo inerente aos mesmos.

A aplicação de técnicas de teste a partir de especificações formais compõe uma importante linha dos esforços para agregar maior rigor e eficiência aos testes de sistemas reativos [Tretmans 1996]. Casos de teste que têm como finalidade avaliar a conformidade entre especificação e implementação podem ser gerados de maneira automática com base em análises sobre a especificação. A esta técnica denomina-se teste de conformidade. A implementação sob teste (IST) é tratada como uma caixa preta, tendo o seu comportamento visível por meio de interações diretas, pelos PCO's.

Uma abordagem para a geração dos casos de teste baseia-se na especificação de propriedades desejáveis para o modelo, conhecidas como objetivos de teste, a partir das quais os casos de teste são projetados [de Vries and Tretmans 2001]. Os objetivos de teste são definidos com base em análises sobre modelos e geralmente provêem foco específico em partes destes. São utilizados como guias na geração de casos de teste, que por sua vez, devem avaliar a conformidade entre os comportamentos especificados e implementados.

O reconhecimento da característica complementar das técnicas de teste, em especial o teste de conformidade, e de verificação de modelos tem atraído muitos esforços para a aplicação conjunta e/ou fusão de ambas. Técnicas e ferramentas de teste de conformidade para sistemas reativos, baseadas na aplicação de técnicas de verificação de modelos, [de Vries and Tretmans 1998, Jard and Jéron 2004] têm sido desenvolvidas e aplicadas a projetos da indústria [Fernandez et al. 1997]. Em [Ammann et al. 1998] contra-exemplos fornecidos por verificadores de modelos são convertidos diretamente em casos de teste, no entanto sem basear-se em uma teoria sólida de testes. Contra-exemplos representam seqüências de execuções simples da IST, não sendo assim, adequados para a aplicação como casos de teste de sistemas reativos não-deterministas. Em [Jard and Jéron 2004] é apresentada uma ferramenta de geração automática de casos de teste. Esta ferramenta é baseada em uma sólida teoria de testes formais, apresentada em [Tretmans 1996], e baseia-se em objetivos de teste para guiar a geração de casos de teste. Entretanto, não provê mecanismos de geração dos objetivos de teste, especificados na forma de STR.

A geração de objetivos de teste é abordada em [Henniger et al. 2003, Silva and Machado 2006]. Em [Henniger et al. 2003] é apresentado um algoritmo para geração de objetivos de teste em MSC's (*Message sequence charts*). A técnica consiste em identificar em modelos, representados por máquinas de estados de comunicação assíncrona, os comportamentos definidos como significantes, dos quais definem-se os objetivos de teste. Um caso de teste deve ser gerado para cada objetivo de teste. Apesar da capacidade de identificar diretamente nos modelos os comportamentos significantes do sistema, o conjunto de casos de teste gerado tende a ser reduzido, não garantindo sua exaustão. Isto acontece pelo fato de o método não prover maior nível de abstração dos objetivos de teste em relação ao grafo de representação dos estados do modelo.

Em [Silva and Machado 2006] apresentamos uma técnica de geração de objetivos de teste, na forma de STR, com base em propriedades especificadas em CTL. A geração fundamenta-se na análise de propriedades CTL sobre um modelo, de modo a obter informações para a definição do STR relativo ao objetivo de teste. Estas informações são obtidas por meio da extração de caminhos do modelo relacionados a propriedades CTL, para a qual é necessária a adaptação do processo de verificação de modelos. Esta adaptação provê maior rigor e precisão à analise de modelos para a geração de objetivos de teste, compondo importante passo no processo de automação do teste de conformidade.

Neste trabalho abordamos a consolidação da aplicação da verificação de modelos como mecanismo de automação da geração de objetivos de teste. A adaptação algorítmica idealizada sobre algoritmos de busca e extração de exemplos e contra-exemplos de verificadores de modelos, necessária à implementação da técnica [Silva and Machado 2006], é apresentada em relação ao verificador de modelos Veritas [Rodrigues et al. 2004].

Este trabalho está organizado da seguinte maneira: A Seção 2 apresenta a técnica proposta em [Silva and Machado 2006]; A Seção 3 apresenta o verificador de modelos Veritas e a adaptação realizada sobre seus algoritmos para a implementação da técnica; A Seção 4 apresenta um estudo de caso, realizado com base na adaptação do Veritas, para a geração de um objetivo de teste e de casos de teste; A Seção 5 apresenta as considerações finais do trabalho.

2. Geração de Objetivos de Teste

A técnica apresentada em [Silva and Machado 2006] fundamenta-se no fato de que o que se deseja verificar em um modelo, deseja-se testar em uma implementação. Fica assim, estabelecida relação de equivalência entre as propriedades da verificação de modelos e dos testes de conformidade. Os eficientes mecanismos de análise, baseados em propriedades especificadas em lógica temporal, providos pela verificação de modelos são utilizados para a geração de objetivos de teste, provendo maior rigor a este processo.

A técnica prevê a realização de análises sobre um modelo, da mesma forma que a verificação de modelos realiza. No entanto, o processo é adaptado para obter informações sobre o modelo para realizar a geração dos objetivos de teste. A adaptação no processo de verificação de modelos dá-se com a mudança dos algoritmos referentes ao conectivo EU de um verificador de modelos. Este conectivo define fórmulas do tipo $EU(p, q)$, nas quais verifica-se a existência de ao menos um caminho em que p seja válida em todos os estados até que q o seja, ou que q seja válida inicialmente. Ao invés de extrair apenas um exemplo ou contra-exemplo relativo à fórmula, o verificador passará a extrair maior número de

ambos, exemplos e contra-exemplos, como mostrado na Figura 1. Assim, permite-se uma maior amostragem do espaço de estados para a extração de informações suficientes para a elaboração de um grafo abstrato que represente o objetivo de teste.

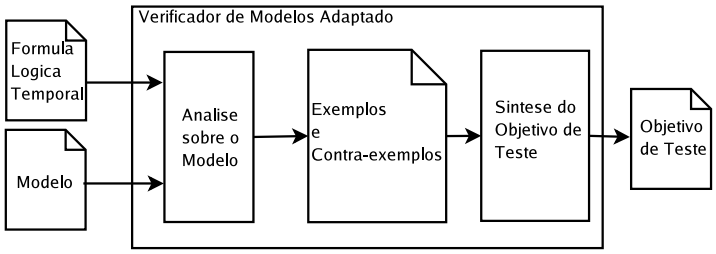


Figura 1. Processo de geração do objetivo de teste

Um objetivo de teste é representado por um STR. Formalmente, um objetivo de teste é uma tupla $M = (Q, A, \longrightarrow, q_0)$, onde Q é um conjunto finito não vazio de estados, A é o alfabeto de ações, $\longrightarrow \subseteq Q \times A \times Q$ é a relação de transição e $q_0 \in Q$ é o estado inicial. É munido de estados especiais de aceitação e rejeição (resp. *accept* e *refuse*). Estes estados são utilizados como base para a geração dos casos de teste, definindo seqüências de transições que representam os comportamentos que devem ou não ser selecionados, respectivamente.

2.1. Extração Exemplos e Contra-exemplos

Uma vez que um objetivo de teste é descrito por um sistema de transições rotuladas, as informações utilizadas para sua geração devem ser baseadas nas transições do modelo que sejam relevantes em relação à propriedade CTL. Desta forma, a extração de maior número de caminhos (i.e. exemplos e contra-exemplos do modelo) é realizada com a finalidade de obter uma maior amostragem do modelo, de modo a proporcionar maior exaustão das informações referentes às transições, que devem compor o STR do objetivo de teste.

Uma vez que os exemplos representam caminhos que satisfazem a propriedade, devem ser usados para prover informações sobre o comportamento aceito pelo objetivo de teste. Análises sobre suas transições devem classificá-las em relevantes ou irrelevantes em relação à propriedade. As transições relevantes devem definir as seqüências de aceitação do objetivo de teste. Já as irrelevantes devem ser abstraídas. Sua abstração é representada por transições especiais rotuladas com ”*”.

Uma vez que a técnica de verificação de modelos é definida sobre uma estrutura que considera estados e transições (i.e. através de estruturas de kripke), a utilização de abstrações de transições com ”*” em um STR pode originar objetivos de teste não representativos, quando baseados apenas em exemplos. Sua abstração pode ser maior que a desejada, uma vez que não se tem restrição sobre estados. Os casos de teste gerados a partir de tal descrição podem conter transições que levem à violação da propriedade. Desta maneira, caminhos que representem a violação da propriedade (i.e. contra-exemplos) devem ser utilizados para prover informações para restringir o objetivo de teste. As transições que causam a violação da propriedade devem definir as seqüências de transições que não devem gerar casos de teste, compondo as seqüências que levam ao estado de rejeição. Estas restrições são também importantes para o não-determinismo. Os casos de teste a serem gerados devem, com base nestas seqüências, prever a ocorrência de respostas do sistema que não correspondem ao objetivo de teste, de modo a não impactar nos resultados.

2.2. Análise e Abstração sobre Exemplos e Contra-exemplos

Para simplificar a análise realizada sobre os caminhos extraídos sobre o modelo, uma representação abstrata sobre seus estados é realizada. O modelo utilizado para tal representação baseia-se em máquina de estados finitos, formalmente definido pela tupla $(Q, \Sigma, \delta, q_0, F)$, onde Q é um conjunto finito e não vazio de estados, Σ um conjunto finito de símbolos que compõem o alfabeto aceito pela máquina, $q_0 \in Q$ o estado inicial, δ a função de transição $\delta : Q \times \Sigma \longrightarrow Q$ e $F \subseteq Q$ o conjunto de estados finais, denominados estados de aceitação. Os estados de cada caminho (exemplo ou contra-exemplo) são classificados em conjuntos definidos pelas proposições da fórmula CTL, de acordo com a relação de satisfação entre estados e proposições. A Figura 2 mostra um exemplo relacionado a uma fórmula CTL $EU(p, q)$. Os estados de um exemplo são classificados em dois conjuntos de estados, de tipos p e q. Os estados que satisfazem apenas a proposição p são denominados estados p. Os estados que satisfazem a proposição q são denominados estados q. Para fórmulas do tipo $EU(p, q)$, os estados q representam os estados de aceitação da máquina de estados.

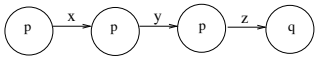


Figura 2. Representação simplificada de um exemplo relativo à fórmula $EU(p, q)$

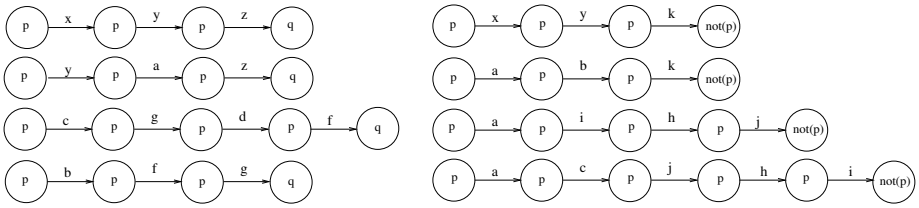
A análise realizada sobre os caminhos deve classificar suas transições como relevantes e irrelevantes em relação à propriedade, com base na detecção de mudança de estados sobre a representação simplificada dos caminhos. Esta classificação é efetuada pela identificação de transições e/ou suas seqüências de transições necessárias às mudanças de estados. A detecção de seqüências de transições necessárias às mudanças de estados é realizada pela interseção entre os dois conjuntos. Desta maneira, apenas seqüências que ocorrem em ordens alternadas são detectadas. Conjuntos de transições que compõem seqüências alternadas que causam mudança de estados devem sempre aparecer de forma conjunta nos caminhos. Desta forma, dois subconjuntos de transições relevantes são criados, um para as transições identificadas na interseção e outro para as demais.

Após a análise sobre os caminhos e classificação das transições, ocorre a geração do objetivo de teste. As transições dos subconjuntos de transições relevantes devem formar o STR do objetivo de teste. As transições dos exemplos compõem os caminhos de aceitação e as transições dos contra-exemplos compõem os caminhos de rejeição.

2.3. Exemplo de Aplicação da Técnica

A Figura 3 mostra um conjunto de caminhos relacionados a uma fórmula $EU(p, q)$. Os conjuntos de transições relevantes e irrelevantes definidos após a análise sobre os exemplos são $L = \{z, f, g\}$ e $N = \{x, y, a, b, c, d, f, g\}$, respectivamente. Algumas transições pertencem aos dois conjuntos (e.g. f e g). Podemos concluir que estas transições causam mudanças de estado de maneira conjunta. Assim, a interseção dos dois conjuntos compõe um terceiro conjunto $I = \{f, g\}$ para agrupá-las.

Após a identificação das transições da interseção entre L e N, as demais transições do conjunto L, definidas pela diferença entre L e I são utilizadas para conectar o estado inicial do objetivo de teste ao estado de aceitação. Neste exemplo, esta diferença é representada por $S = \{z\}$. Os caminhos que contêm transições que pertencem ao conjunto I

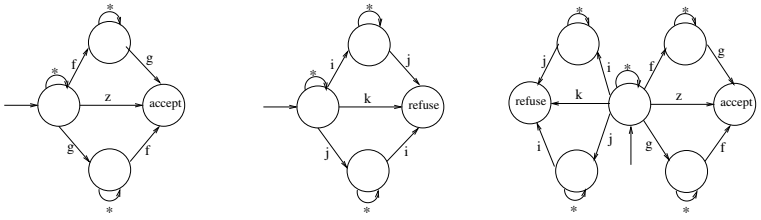


(a) (b)

Figura 3. Exemplos (3(a)) e contra-exemplos (3(b)) de uma fórmula $EU(p, q)$

são construídos com base em suas seqüências, que definem subconjuntos de I. Para cada subconjunto, ou seja, para cada seqüência, um caminho entre o estado inicial do grafo e o estado de aceitação é definido.

O grafo definido por este procedimento, sobre os exemplos extraídos do modelo, denomina-se grafo de aceitação (Figura 4(a)). O mesmo procedimento deve ser aplicado em relação aos contra-exemplos (Figura 3(b)) no intuito de construir o grafo de rejeição (Figura 4(b)). O grafo que representa o objetivo de teste deve conter as informações reunidas nos dois grafos. O objetivo de teste resultante é mostrado na Figura 4(c).



(a) Grafo de aceitação (b) Grafo de rejeição (c) Objetivo de teste resultante

Figura 4. Grafos obtidos no processo

3. Adaptação sobre o Veritas

O Veritas [Rodrigues et al. 2004] é um verificador de modelos CTL baseado em espaços de estados RPOO (Redes de Petri Orientadas a Objetos) [Guerra et al. 2004], que é uma linguagem de modelagem que adiciona às redes de Petri [Jensen 1992] características da orientação a objetos. Com a adaptação, delineada nesta seção, sobre seu algoritmo de busca relativo ao conectivo EU, que originalmente visa uma busca mínima por um espaço de estados, a finalidade do Veritas passa a ser prover informações para a geração dos objetivos de teste por meio de maior exaustão na busca de caminhos relativos à fórmula.

O Veritas implementa algoritmos de busca em profundidade baseados na abordagem definida em [Heljanko 1997, Vergauwen and Lewi 1993], que facilita a extração de exemplos e contra-exemplos durante a verificação do modelo. Nesta abordagem, cada tipo de conectivo (e.g. EU, EG) define a forma de pesquisa sobre o espaço de estados, que deve ser iniciada em um estado específico. Esta estratégia permite que subfórmulas não sejam

verificadas sobre todo o modelo, como a definida em [Clarke et al. 1999], minimizando a pesquisa realizada. Sua verificação pode ser realizada com base em estados encontrados após certa profundidade alcançada na busca, como por exemplo, na verificação de outras subfórmulas. Esta definição de verificação, em estados específicos, define um exemplo ou contra-exemplo durante a verificação ao passo em que o algoritmo evolui na busca.

A implementação sobre o algoritmo referente ao conectivo EU define que a pesquisa sobre o modelo deve parar ao encontrar um caminho que satisfaz a fórmula aplicada à verificação. Assim, apenas a extração de um exemplo é realizada. Nenhum contra-exemplo é apresentado em caso contrário.

O algoritmo 1 mostra o procedimento de busca definido pelo Veritas para o conectivo referido. O procedimento, denominado *check_EU*, recebe como argumento a fórmula f, o estado a ser verificado s e o caminho percorrido π. Caso o estado tenha sido previamente visitado e sua avaliação em relação à fórmula tenha sido verdadeira, a pesquisa deve ser finalizada (linhas 2,3 e 4, respectivamente).

Algoritmo 1 Algoritmo de busca relativo ao conectivo EU

```
 1: proc check_EU(f, s, π)
 2: if marked(f, s) then
 3:    if info(f, s) then
 4:       continue = false
 5:    end if
 6: else
 7:    mark(f, s)
 8:    if check(arg(f, 2),s,π) then
 9:       setInfo(f, s, true)
10:       insert(s, π)
11:       continue = false
12:    else
13:       if check(arg(f, 1),s,π) then
14:          for all t ∈ sucessors(s) ∧ continue do
15:             check_EU(f, t, π)
16:          end for
17:          if ¬ continue then
18:             insert(s,π)
19:             setInfo(f, s, true)
20:          end if
21:       end if
22:    end if
23: end if
```

O segundo argumento da fórmula, que denominaremos q, deve ser verificado em relação ao estado (linha 8). O procedimento *check* deve identificar o tipo de fórmula que recebe como argumento, neste caso q, devendo invocar o método de verificação correto correspondente. Neste caso, q pode ser uma proposição atômica, que deve ter sua validade verificada apenas no estado corrente (s), ou pode ser uma fórmula qualquer, definindo o aninhamento da fórmula que deve ser verificada por recursão.

Caso a avaliação de q seja positiva em s, a pesquisa é interrompida (linha 11),

uma vez que a condição de parada de $EU(p, q)$ é a satisfação de q, inicialmente, ou após uma seqüência de estados que satisfazem p. O estado s é adicionado ao caminho π (linha 9) e o resultado, utilizado pelo verificador em etapa posterior, é armazenado (linha 10). Esta última informação (função *info*) é utilizada pelo verificador para definir o resultado da verificação.

Caso q não seja verdadeira, o primeiro argumento, p, é verificado (linha 13). Caso a avaliação de p seja verdadeira, a busca deve continuar pelos estados sucessores de s, através de recursão (linha 15). Após a verificação dos estados sucessores, caso a avaliação tenha sido positiva para algum caminho alcançado pela recursão (linha 17), s é adicionado ao início do caminho (linha 18) e a fórmula é avaliada como verdadeira (linha 19).

3.1. Adaptação Algorítmica

A adaptação mostrada pelo algoritmo 2 restringe a verificação a fórmulas EU não aninhadas, ou seja, para as quais os argumentos p e q sejam apenas proposições atômicas. Nesta adaptação, a classificação das transições do modelo é efetuada ao passo em que realiza a busca e extração dos caminhos, que são utilizados para obtenção de informações sobre predecessão entre transições no algoritmo de síntese de objetivos de teste.

Algoritmo 2 Algoritmo de busca relativo ao conectivo EU

```
 1: proc check_EU(f, s, π)
 2: if ¬marked(f, s) then
 3:    mark(f, s)
 4:    if check(arg(f, 2),s,π) then
 5:       setInfo(f, s, true)
 6:       insert(s, π)
 7:       save(π,examples)
 8:       add(getTransition(last(π), s), acc, L)
 9:    else
10:       if check(arg(f, 1),s,π) then
11:          add(getTransition(last(π), s), #, N)
12:          insert(s,π)
13:          for all t ∈ sucessors(s) ∧ continue do
14:             check_EU(f, t, π)
15:          end for
16:       else
17:          add(getTransition(last(π), s), ref, L)
18:          insert(s, π)
19:          save(π,counter − examples)
20:       end if
21:    end if
22: else
23:    if add(getTransition(last(π), s), acc, L) ∧ info(f, s) then
24:       insert(s, π)
25:       save(π,examples)
26:    end if
27: end if
```

Sua condição de parada é a cobertura de todos os estados q sem que percorra ca-

minhos com auto-laços (linha 2). Caminhos que contêm estados q previamente cobertos serão extraídos caso haja transições não contidas em outros caminhos extraídos anteriormente. A Figura 5 mostra um esquema da busca realizada pelo algoritmo sobre um modelo. Os estados q são representados pelos estados 4 e 6. Os exemplos a serem extraídos são representados pela setas contínuas. O exemplo composto pelas setas tracejadas é descartado.

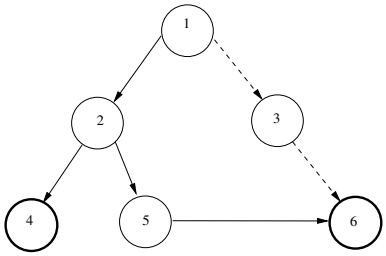


Figura 5. Estratégia de extração de caminhos

O controle realizado sobre a extração de caminhos com estados q previamente cobertos é apresentado entre as linhas 23 e 26. A função *info* passa a ser utilizada apenas para controle destes estados, deixando de ser acionada na avaliação de estados p. Assim, caso o estado visitado seja um estado q e a transição em questão não tenha sido previamente coletada (linha 23), o caminho é extraído (linhas 24 e 25).

Caso a avaliação de q seja positiva em relação ao estado s (linha 4), seu resultado é armazenado (linha 5), e s é adicionado ao caminho π (linha 6), que deve ser armazenado como exemplo (linha 7). A transição que ocorre entre s e seu antecessor é armazenada no conjunto de transições relevantes L com a informação sobre o estado resultante, que neste caso é o de aceitação (representado por *acc*, linha 8).

Caso q não seja verdadeira, p é verificado (linha 10). Caso a avaliação de p seja verdadeira, a transição que ocorre entre s e seu antecessor é armazenada no conjunto de transições irrelevantes N (linha 11). Para este conjunto, a informação de estado não é armazenada, uma vez que as transições são representadas por auto-laços de rótulo "*". A busca continua pelos estados sucessores de s (linha 14).

A avaliação negativa de p em s define um contra-exemplo, que deve ser extraído (linhas 17 e 18). A transição que ocorre entre s e seu antecessor é armazenada no conjunto de transições relevantes L, definindo o estado de rejeição (*ref*) como resultante (linha 17).

4. Estudo de Caso

Para demonstrar o funcionamento da técnica e dos algoritmos, um estudo de caso foi realizado sobre o protocolo IP Móvel [Perkins 2002]. Um objetivo de teste foi gerado e então, casos de teste foram gerados pela ferramenta TGV [Jard and Jéron 2004] a fim de validar o objetivo de teste.

Os protocolos de roteamento da Internet não operam com a possibilidade de migração de seus nodos de forma dinâmica, não prevendo alterações de endereços IP. A migração de um nodo pode causar a perda de conexão com a rede. O IP móvel foi desenvolvido para corrigir este problema, de modo que as migrações ocorram de forma transparente às aplicações mantendo a conexão dos dispositivos móveis (*mobile nodes*).

Para prover a transparência nas migrações, o protocolo propõe como solução que um dispositivo móvel (MN) possua dois endereços IP, o endereço original (*home address*), e um endereço que será atribuído a cada migração, denominado *care-of-address* (COA). O *home address* é obtido na rede de origem (*home network*), enquanto o COA é obtido na rede estrangeira (*foreign network*) para a qual o dispositivo esteja migrando. Assim, enquanto está em uma rede estrangeira, o dispositivo deve receber os pacotes através do roteador desta, o *foreign agent* (FA). Este roteador, por sua vez, recebe os pacotes destinados ao dispositivo através do roteador da rede de origem, o *home agent* (HA). Os pacotes enviados por um *host* ao dispositivo são destinados ao *home address*, que ao chegar ao *home agent* é encapsulado em outro pacote cujo endereço de destino é o COA.

O processo de migração de um dispositivo consiste na descoberta de um COA ao conectar-se à rede estrangeira e posterior registro deste junto ao *home agent*. Este registro é intermediado pelo *foreign agent*, que encaminha a mensagem informando o COA ao *home agent*, que por sua vez, atualiza os registros e confirma tal atualização.

4.1. Extração de Caminhos e Geração do Objetivo de Teste

Como um objetivo de teste deseja-se avaliar a conformidade entre uma IST e um modelo para o envio das mensagens ao *mobile node*. Inicialmente em sua rede local, enquanto não migrar-se, as mensagens devem ser encaminhadas pelo *home agent*. Uma fórmula $EU(p, q)$ é especificada, para a qual p é a proposição atômica "O *mobile node* encontra-se em sua *home network*" e q a proposição "O *home agent* entrega as mensagens ao *mobile node*". Um tipo específico de mensagem foi definido para prover maior restrição aos exemplos e contra-exemplos deste estudo de caso.

A análise realizada sobre o modelo, que possui um total de 1086 estados e 2502 transições, possibilitou a extração de 36 caminhos com 64 estados e 63 transições. A cada caminho em que a propriedade foi satisfeita, através da ocorrência de eventos em que o *home agent* entrega a mensagem o *mobile node*, definiu-se um exemplo a ser extraído. O número de exemplos extraídos foi 7. Suas transições foram classificadas durante o procedimento de busca do algoritmo, que encontrou uma transição relevante, representada pelo rótulo "*HA:MN.receiveDatagram(dat)*". As demais transições foram classificadas como irrelevantes, por não causarem mudanças de estados em relação à representação simplificada de estados. Estas transições são representadas por intermédio de abstrações em transições rotuladas com "*" no objetivo de teste.

Os contra-exemplos, em 29, corresponderam à migração do *mobile node* antes do recebimento da mensagem. O processo de classificação das transições definiu como relevante apenas uma transição, de rótulo "*FA:MN:receiveAdv(ip)*". O objetivo de teste gerado com base nestas informações é apresentado na Figura 6.

A geração de casos de teste correspondentes ao objetivo de teste da Figura 6 foi efetuada pela ferramenta TGV. Um grafo completo dos casos de teste (CTG) foi gerado. Este CTG contém 65 estados e 82 transições, abrangendo os estados e transições contidos nos caminhos extraídos pelo verificador. O conjunto de transições contidas no CTG foi igual ao dos caminhos extraídos do modelo. A diferença de números deve-se à estratégia de extração definida pelo algoritmo.

Como o algoritmo de busca apresentado tem como objetivo cobrir todos os estados q, a cobertura do CTG em relação a estes estados foi a mesma dos caminhos obtidos. En-

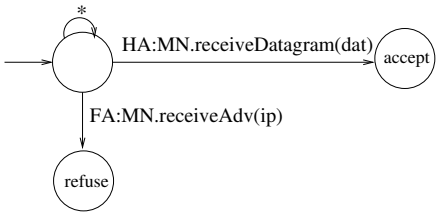


Figura 6. Objetivo de teste resultante

tretanto, além dos exemplos coletados, o CTG cobriu todas as possibilidades de caminhos sobre os estados q (e.g. o exemplo de seta tracejada da Figura 5 também foi coberto). Esta análise mostra uma exaustão dos casos de teste sobre o modelo em relação à propriedade CTL inicialmente especificada.

A combinação de estados e transições do CTG permite a definição de diferentes casos de teste que, em caso de laços no modelo, podem ser infinitos. Os casos de teste gerados devem prever situações em que o não-determinismo do sistema impossibilite a execução das seqüências de ações necessárias à validação do objetivo de teste, implicando em veredictos inconclusivos na sua execução. Desta forma, um caso de teste extraído do CTG pode ser composto pelas seqüências de uma parcela dos exemplos e contra-exemplos obtidos do modelo. Como exemplo, um caso de teste gerado conteve 36 estados e 38 transições, abrangendo boa parcela dos caminhos produzidos na busca, fato que mostra ser inapropriada a conversão direta de exemplos e contra-exemplos em casos de teste.

5. Conclusão

A adaptação sobre um verificador de modelos, realizada sobre o Veritas neste trabalho, possibilita a implementação da técnica de geração de objetivos de teste por meio da especificação direta de propriedades baseadas em lógica temporal CTL. A estratégia de extração de caminhos, adotada com o algoritmo, mostrou-se eficiente mecanismo de coleta de informações para o processo de geração. A abstração de determinados caminhos possibilitou uma otimização do processo, sem perda de informações.

Assim como diversos trabalhos têm se baseado na aplicação de técnicas de verificação de modelos para a geração de casos de teste, como em [Jard and Jéron 2004, de Vries and Tretmans 1998], a eficiência alcançada na geração de objetivos de teste mostra a viabilidade de ampla integração das técnicas, provendo maior rigor ao teste. Dentre os próximos passos desta integração estão a adaptação dos algoritmos baseados em outros operadores CTL (e.g. EU, EX), e a adaptação de algoritmos de verificação de modelos baseados em lógica temporal linear (LTL) [Clarke et al. 1999] para a geração de objetivos de teste. Uma vez que CTL e LTL têm poderes de expressão complementares, esta adaptação possibilita à técnica [Silva and Machado 2006] uma maior abrangência e poder de expressividade na especificação de propriedades para a geração de casos de teste.

Referências

Ammann, P. E., Black, P. E., and Majursky, W. (1998). Using model checking to generate tests from specifications. In Society, I. C., editor, *In Proceedings of the Second IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, pages 46–54.

Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

de Vries, R. G. and Tretmans, J. (1998). On-the-fly conformance testing using spin. In G. Holzmann, E. N. and Serhrouchni, A., editors, *Fourth Workshop on Automata Theoretic Verification with the Spin Model Checker, ENST 98 S 002, Paris, France. Ecole Nationale Supérieure des Télécommunications*, pages 115–128.

de Vries, R. G. and Tretmans, J. (2001). Towards formal test purposes. In *Proc. 1st International Workshop on Formal Approaches to Testing of Software (FATES), Aalborg, Denmark*, pages 61–76.

Fernandez, J.-C., Jard, C., Jéron, T., and Viho, G. (1997). An experiment in automatic generation of conformance test suites for protocols with verification technology. *Science of Computer Programming*, 29:123–146.

Guerra, F. V. d. A., de Figueiredo, J. C. A., and Guerrero, D. D. S. (2004). Timed extension of an object oriented petri net language. In *Simpósio Brasileiro de Métodos Formais - SBMF 2004*, pages 132–148, Recife - Brasil.

Heljanko, K. (1997). Model checking the branching time temporal logic ctl. Technical Report A45, Helsinki University of Technology.

Henniger, O., Lu, M., and Ural, H. (2003). Automatic generation of test purposes for testing distributed systems. In *Formal Approaches to Software Testing, Proceedings of FATES'03*, volume 2931 of *Lecture Notes in Computer Science*, pages 178–191. Springer.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Jard, C. and Jéron, T. (2004). TGV: theory, principles and algorithms – A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT)*, 6.

Jensen, K. (1992). *Coloured Petri Nets 1: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer-Verlag, Berlin, Alemanha.

Perkins, C. (2002). Rfc 3344:IP mobility support for IPv4. Status: Proposed Standard.

Rodrigues, C. L., Guerra, F. V., de Figueiredo, J. C. A., Guerrero, D. D. S., and Morais, T. S. (2004). Modeling and verification of mobility issues using object-oriented petri nets. In *Proc. of 3rd International Information and Telecommunication Technologies Symposium (I2TS2004)*.

Silva, D. A. and Machado, P. D. L. (2006). Towards Test Purpose Generation from CTL Properties for Reactive Systems. In *Proc. of the 2nd International Workshop on Model Based Testing (MBT), Vienna, Austria*. To appear.

Tretmans, J. (1996). Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software—Concepts and Tools*, 17(3):103–120.

Vergauwen, B. and Lewi, J. (1993). A linear local model checking algorithm for ctl. In Best, E., editor, *CONCUR'93: Proc. of the 4th International Conference on Concurrency Theory*, pages 447–461. Springer, Berlin, Heidelberg.