EXECUÇÃO EXAUSTIVA DE STATECHARTS

Inês A. G. Boaventura
Departamento de Ciências de Computação e Estatística
IBILCE - UNESP - São José do Rio Preto
Rua Cristovão Colombo, 2265 - Jardim Nazareth
São José do Rio Preto - SP - CEP 15055

Paulo C. Masiero
Departamento de Ciências de Computação e Estatística
ICMSC - USP - São Carlos

8 de setembro de 1992

Sumário

Os Statecharts, devido a sua definição formal, permitem que os sistemas especificados com eles sejam validados de diversas maneiras. Neste trabalho, mostra-se como pode ser construída uma árvore de alcançabilidade para Statecharts que permite a simulação exaustiva de todos os possíveis estados alcançáveis, considerando toda a semântica dos Statecharts, inclusive história. Mostram-se os algoritmos desenvolvidos para avaliação de propriedades dinâmicas dos Statecharts, tais como: alcançabilidade, reiniciabilidade, deadlock, seqüência válida de eventos e uso de transições. Apresentam-se também exemplos de execução desses algoritmos.

Abstract

Statecharts, due to its formal definition, allow that systems specified using them, be validated of several ways. In this paper it is shonw how to build a reachability tree, which is the basis for exhaustive simulation of all state configurations. The simulation is based on the defined semantics for statecharts, including the history concept. It is shown algorithms developed for validating dynamic properties of statecharts, e.g., reachability, reversibility, deadlock, valid sequence of events and transition usage. Execution of the algorithms is also shown, through examples.

1 Introdução

Existe atualmente, dentro da área de Engenharia de Software, uma grande preocupação em construir teorias, técnicas e ferramentas para modelagem, análise e avaliação de sistemas complexos. Nesse contexto, a técnica denominada Statecharts é uma técnica adequada para o problema de especificação e validação do aspecto comportamental de sistemas reativos, e tem mostrado ser uma alternativa vantajosa, em comparação a outros métodos de especificação, principalmente aqueles baseados em máquinas de estados finitos convencionais.

Os Statecharts possuem um grande poder de representação e, além de facilitarem a manipulação e visualização de aplicações complexas, possuem uma definição formal, de maneira a possibilitar execução detalhada do sistema, permitindo sua validação através da Execução Exaustiva, produzindo assim todos os possíves cenários do sistema sob desenvolvimento. Segundo Harel, em [Hare92], a Execução Exaustiva é uma das técnicas de análise de especificações que constitui o estado da arte atual no desenvolvimento de sistemas complexos.

Por outro lado, as Redes de Petri, que existem há mais de vinte anos, também possuem um formalismo adequado para a modelagem de sistemas complexos, apresentando muitos recursos para estudos de propriedades desses sistemas. Este trabalho apresenta a definição de uma árvore de alcançabilidade para Statecharts, e sobre essa árvore de alcançabilidade definida, discute algoritmos para validação de especificações baseadas em Statecharts, verificando suas propriedades dinâmicas. A árvore de alcançabilidade proposta é inspirada na árvore semelhante, definida para Redes de Petri.

O trabalho está organizado da seguinte forma: na segunda seção faz-se uma breve introdução à sintaxe e semântica dos Statecharts; a seção três apresenta a definição de uma árvore de alcançabilidade para Statecharts, ilustrando sua construção através de um exemplo; mostra-se ainda a execução do algoritmo discutido, utilizando um exemplo real; na seção número quatro, baseadas na árvore de alcançabilidade, são definidas várias propriedades dinâmicas de Statecharts e mostra-se como as propriedades definidas podem ser verificadas, usando a árvore de alcançabilidade; a seção número cinco apresenta as conclusões do trabalho.

2 Introdução aos Statecharts

Os Statecharts (ou Estadogramas) foram introduzidos por Harel, em [Hare87a], como um formalismo visual para especificar o aspecto comportamental de sistemas reativos. Statechart é uma extensão das Máquinas de Estados Finitos (MEF), incluindo nestas os conceitos de hierarquia, paralelismo e mecanismo de comunicação, apropriando-as para o uso em aplicações grandes e complexas.

A sintaxe e semântica formais dos Statecharts podem ser encontradas em [Hare87b]. Nesta seção será apresentada informalmente a notação visual da técnica. Em um diagrama Statecharts, os retângulos com bordas arredondadas são utilizados para representar os **estados** em qualquer nível, utilizando o encapsulamento para expressar a relação de hierarquia. As setas direcionadas representam as **transições**, e podem se originar e terminar em qualquer nível. Uma seta é rotulada com um evento e, opcionalmente, também com uma condição.

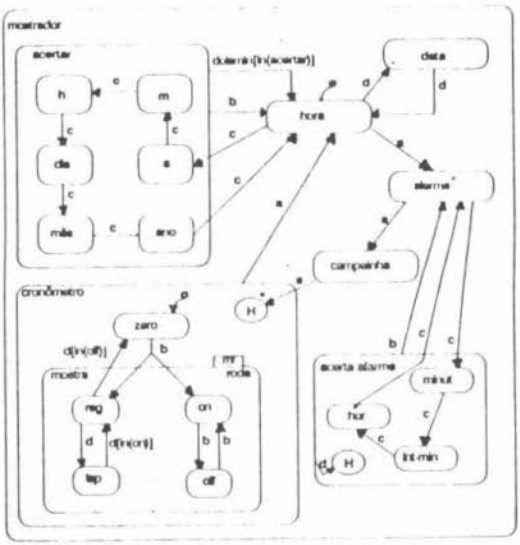

Figura 1. Statechart do Mostrador de um Relógio Digital.

A Figura 1 foi extraída de [Hare88a] e apresenta a especificação do mostrador de um relógio digital. Os eventos "a", "b", "c" e "d" representam o ato de pressionar os quatro botões externos do relógio. A semântica do estado **mostrador** é a de decomposição do tipo OU - exclusivo (XOR). Assim, estar no estado **mostrador**, significa estar apenas em um dos seus estados filhos: **acertar**, hora, data, alarme, campainha, cronômetro ou acerta-alarme.

O paralelismo (ortogonalidade) é representado pelos Statecharts através de linhas tracejadas delimitando os estados concorrentes, conforme mostra a Figura 1. O estado **mr** consiste de dois componentes ortogonais: os estados **mostra** e **roda**. Isso significa que quando o estado **mr** é ativado, os estados **mostra e roda** tornam-se ativos simultaneamente.

As transições podem ter condições associadas aos eventos, como mostram os eventos "doismin [in(acertar)]" e "d [in(off)]". Nestes casos o sistema somente transiciona se o evento ocorrer e se nesse momento a condição entre colchetes for verdadeira. Assim, observando a Figura 1, "d [in(off)]" possui a seguinte semântica: ocorrendo o evento d, o sistema transiciona de **reg** para **zero** apenas se na componente paralela **roda** o estado **off** estiver ativo.

Eventos de saída podem ser especificados nos Statecharts. Esses eventos são chamados de "ações". As ações, além de indicarem possíveis eventos para o mundo real, podem gerar novos eventos internos que se propagam ("broadcasting"), afetando outros componentes do diagrama. Por exemplo, suponha uma transição rotulada por "ev/a". Se o evento "ev" ocorrer, então a ação "a" será imediatamente executada, podendo causar novos eventos e, consequentemente, transições adicionais.

Uma característica interessante dos Statecharts é a capacidade de "lembrar" uma visita feita anteriormente a um estado. Essa característica é chamada **história**, e a notação utilizada é um **H** associado a uma transição de um estado (subestado ou superestado) para um superestado. Uma transição associada a um símbolo de história tem o significado de que, quando seu evento dispara, o próximo estado para o qual o sistema vai mudar, será aquele em que o sistema estava da última vez em que aquele superestado foi ativado. Usa-se o símbolo **H*** para indicar que a história deve ser lembrada recursivamente até os estados básicos.

No Statechart da Figura 1 aparecem exemplos da utilização dos símbolos **H** e **H***. Por exemplo, se o estado **campainha** estiver ativo e o evento "a" ocorrer, os estados **zero** ou **mr** serão ativados, dependendo da última configuração do superestado **cronômetro**. Se o estado **mr** é que será ativado, então esse mesmo procedimento será aplicado para todos os seus descendentes, até se atingir o nível básico. A transição *default* é utilizada apenas na primeira vez em que o sistema visita o superestado **cronômetro**.

3 Árvore de Alcançabilidade para Statecharts

Nesta seção apresentam-se alguns aspectos da construção de uma árvore de alcançabilidade para Statecharts. A árvore de alcançabilidade definida é semelhante à árvore de alcançabilidade definida para Redes de Petri, com algumas características próprias, tais como: a notação utilizada; a maneira de representar a hierarquia dos estados; o tratamento de história; e a forma de utilizá-la. Essa árvore de alcançabilidade foi definida com o objetivo de viabilizar a verificação de propriedades dinâmicas dos Statecharts.

3.1 Construção da Árvore de Alcançabilidade

A árvore de alcançabilidade é construída considerando-se todas as possíveis seqüências de eventos do Sistema Sob Desenvolvimento (SSD). Em cada passo todos os eventos são considerados como verdadeiros, assim todas as transições habilitadas num passo levam a uma nova configuração do sistema. As condições são incluídas nas expressões dos eventos e também são consideradas quando uma transição é tomada. A execução do SSD é uma seqüência representada pelo conjunto $\{(CS_i, \Upsilon_i, \Pi_i^*)\}$, $i \geq 0$, onde:

1. $CS_0 = (X_0, \Pi_0, \Theta_0, \xi_0)$, onde X_0 é a configuração de estado inicial e os outros símbolos são os estímulos externos que ocorrem no primeiro intervalo de tempo I_0. Π_0 é o conjunto de eventos externos gerados pelo ambiente, Θ_0 é o conjunto de condições primitivas cujo valor é TRUE e ξ_0 é uma função que para cada variável x no ambiente, calcula o valor de x.
2. Υ_i é um passo tomado em CS_i, no instante σ_{i+1}, para $i \geq 0$.
3. Π_i^* é o conjunto de eventos gerados por Υ_i, para $i \geq 0$.

Dessa forma, tem-se uma seqüência de configurações de estado onde CS_{i+1} é a configuração do sistema alcançada de CS_i, quando as transições em Υ_i, $i \geq 0$ são tomadas. É importante ressaltar

que no momento da construção da árvore, cada uma dessas transições é considerada separadamente, e não mais que uma transição ocorrendo ao mesmo tempo, como é a semântica dos Statecharts definida em [Hare87b].

Para cada um dos elementos dessa seqüência toma-se o conjunto $RS = \{(\Upsilon_j, \Pi_j^*)\}$ de possíveis reações do sistema.

Na árvore, cada nó é associado a uma configuração de estados do SSD. Os nós são classificados como: *Configuração nova*, *Configuração velha*, *Configuração terminal*, e *Configuração história*.

As *configurações novas* são aquelas configurações geradas e que ainda não foram inseridas na árvore. As *configurações velhas* são as configurações que já foram processadas e inseridas na árvore de alcançabilidade. As *configurações terminais* representam aquelas que possuem o conjunto $RS = \emptyset$, ou seja, não possuem nenhuma transição habilitada.

Os *nós história* representam todas as configurações alcançáveis a partir de transições ou estados associados ao símbolo de história. Os símbolos **H** e **h** aparecem nestes nós em substituição às configurações alcançáveis, naqueles estados que são ativados por história H^* e H, respectivamente. Após a criação da árvore, ao percorrê-la, quando um nó história for encontrado, substituem-se os símbolos **H** e **h** pela última configuração alcançada por aquele estado. Este procedimento será visto com mais detalhes, por meios de exemplos, na seção seguinte.

A árvore de alcançabilidade é construída tomando-se a configuração inicial de estados (ou default) e fazendo-a raiz da árvore. A raíz da árvore é uma *configuração nova* a ser processada. Para essa configuração consideram-se todos os eventos que podem ocorrer. Para cada um deles uma nova configuração é gerada, classificada e inserida na árvore. Este procedimento se repete até que não existam mais *configurações novas* a serem processadas, ou seja, todas as configurações alcançáveis pelo SSD foram classificadas em configurações velhas ou terminais, dando o tratamento adequado quando aparece o símbolo de história.

A notação utilizada para a representação de um nó da árvore de alcançabilidade é a seguinte:

- Parênteses "(" e ")" são utilizados para representar os superestados tipo **XOR**.
- Colchetes "[" e "]" são utilizados para representar os superestados tipo **AND**.
- Os estados átomos são representados por "1" ou "0", conforme estejam ligados ou desligados. Podem ser representados, eventualmente, pelo seu próprio nome.

3.2 Exemplo

Para o Statechart da figura 1, cada nó da árvore de alcançabilidade tem a forma :

(data, hora, alarme, campainha, (hor, int-min, minut), ([(lap, reg), (off, on)], zero), (ano, mes, dia, m, s, h)) Onde os parênteses mais externos representam o estado **mostrador**, que é um estado XOR composto por sete descendentes, os estados básicos **data, hora, alarme, campainha** e os estados **acertar-alarme, cronômentro** e **acertar** que são do tipo XOR e representados por parênteses. O estado **cronômetro** possui dois descendentes, o estado básico **zero** e o estado **mr** que é do tipo AND e é representado por colchetes e assim sucessivamente.

não apareceriam na árvore. Essa solução é determinística, apesar da existência desses dois nós: o nó default e o nó história.

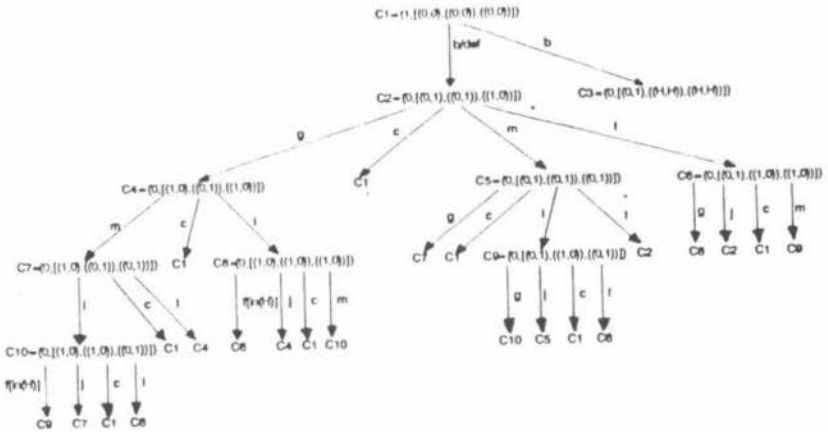

Figura 3. Árvore de Alcançabilidade para o Statechart da Figura 2.

Existem casos onde não há a necessidade da criação do arco rotulado com a extensão */def*, além daquele rotulado com o nome do evento que dispara a transição associada a um símbolo de história. No caso em que o estado destino dessa transição já tenha sido visitado anteriormente pelo sistema, e portanto já aparece representado na árvore de alcançabilidade, somente o *nó história* é criado.

Um outro aspecto importante e que merece ser discutido é a forma de tratar as transições que têm como estado origem um superestado. No Statechart da Figura 2 tem-se a transição *c* que tem como estado origem o superestado D. Essa organização hierárquica adotada pelos Statecharts significa que se o sistema estiver em qualquer um dos subestados de D e a transição *c* ocorrer, o sistema irá para o estado B.

Ao construir a árvore de alcançabilidade, a maneira de se organizar essa hierarquia é: para toda configuração de estados, onde são representados os subestados de D ligados, considera-se a transição *c*. Outra solução seria considerar essas transições apenas no primeiro nó da árvore onde o estado D estivesse ligado, mas essa solução dificultaria bastante no momento de percorrer a árvore. Na árvore de alcançabilidade da Figura 3, pode-se observar a solução adotada para representar a hierarquia de estados. Nota-se que essa solução acaba por deixar "plana" a representação hierárquica dos Statecharts.

3.3 Implementação

O algoritmo para a construção da árvore de alcançabilidade para Statecharts, CAAS, pode ser encontrado em [Boav92] e [Masi92]. Uma implementação foi desenvolvida em estações de trabalho SUN, utilizando-se a linguagem C, ligada ao ambiente STATSIM [Masi91].

não apareceriam na árvore. Essa solução é determinística, apesar da existência desses dois nós: o nó default e o nó história.

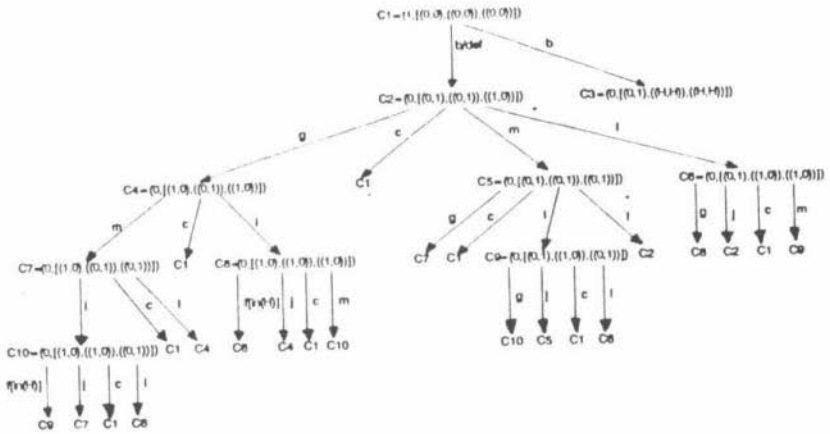

Figura 3. Árvore de Alcançabilidade para o Statechart da Figura 2.

Existem casos onde não há a necessidade da criação do arco rotulado com a extensão */def*, além daquele rotulado com o nome do evento que dispara a transição associada a um símbolo de história. No caso em que o estado destino dessa transição já tenha sido visitado anteriormente pelo sistema, e portanto já aparece representado na árvore de alcançabilidade, somente o *nó história* é criado.

Um outro aspecto importante e que merece ser discutido é a forma de tratar as transições que têm como estado origem um superestado. No Statechart da Figura 2 tem-se a transição *c* que tem como estado origem o superestado D. Essa organização hierárquica adotada pelos Statecharts significa que se o sistema estiver em qualquer um dos subestados de D e a transição *c* ocorrer, o sistema irá para o estado B.

Ao construir a árvore de alcançabilidade, a maneira de se organizar essa hierarquia é: para toda configuração de estados, onde são representados os subestados de D ligados, considera-se a transição *c*. Outra solução seria considerar essas transições apenas no primeiro nó da árvore onde o estado D estivesse ligado, mas essa solução dificultaria bastante no momento de percorrer a árvore. Na árvore de alcançabilidade da Figura 3, pode-se observar a solução adotada para representar a hierarquia de estados. Nota-se que essa solução acaba por deixar "plana" a representação hierárquica dos Statecharts.

3.3 Implementação

O algoritmo para a construção da árvore de alcançabilidade para Statecharts, CAAS, pode ser encontrado em [Boav92] e [Masi92]. Uma implementação foi desenvolvida em estações de trabalho SUN, utilizando-se a linguagem C, ligada ao ambiente STATSIM [Masi91].

A estrutura utilizada para a implementação da árvore de alcançabilidade para Statecharts é uma estrutura de árvore de 2-vias. A árvore é construída dinamicamente em memória principal pelo algoritmo CAAS.

4 Verificação de Propriedades

Mostra-se a seguir uma tabela onde são discutidas brevemente as propriedades de Redes de Petri [Pete81] e as propriedades de Statecharts, fazendo um relacionamento entre elas. Os algoritmos detalhados, para as propriedades ilustradas nesta seção, podem ser encontrados em [Boav92].

Propriedades	Redes de Petri	Statecharts
Limitabilidade	Uma RP é K-limitada se em cada um de seus lugares não há mais que K fichas	Não foi definida para Statecharts
Segurança	É um caso especial de limitabilidade, onde k = 1	Não foi definida para Statecharts
Conservação	Uma RP é conservativa se a soma das fichas, em cada uma de suas marcas, for sempre igual	Não foi definida para Statecharts
Vivacidade	Garante que o sistema nunca atingirá um estado de inatividade total, onde possa permanecer por tempo indefinido	Em Statecharts, com o nome de Impasse (deadlock) garante que a partir da configuração inicial, não existe nenhuma configuração alcançável na qual nenhuma transição está habilitada
Seqüência de Disparos	Pode-se verificar a possibilidade de disparo de uma seqüência específica de transições ou de algum subconjunto dessa seqüência.	Com o nome de Seqüência válida de eventos, é semelhante à definição de seqüência de disparos para Redes de Petri
Alcançabilidade	Garante, a partir de uma marcação inicial M0 se uma determinada marcação M pode ser alcançada ou não	Define-se alcançabilidade de configurações de estatos da mesma forma que alcançabilidade de marcações em Redes de Petri

Propriedades	Redes de Petri	Statecharts
Reiniciabilidade	Define-se reiniciabilidade em RP, quando o sistema retorna ao seu estado inicial após sua execução.	Um Statechart é reiniciavel se existem meios de retornar à sua conficuração inicial, a partir de qualquer configuração alcançável de CSo.
Uso de transição	A definição está relacionada com vivacidade. Uma transição está viva se pode ser disparada.	Pode-se definir da mesma forma que em Redes de Petri

Alcançabilidade de uma Configuração

Alcançabilidade em Statecharts é definida como sendo aquelas configurações de estado que o sistema pode alcançar. Essa propriedade pode ser útil quando um sistema está sendo especificado, ao fornecer meios para verificar se todos os estados especificados são ativados em algum momento, ou, se sob certas condições, alcança uma determinada configuração. Por exemplo, num Sistema Avionístico deseja-se saber se os assentos podem ser lançados para fora quando a aeronave se encontra em alguma situação crítica. Nesse exemplo, uma análise de alcançabilidade pode ser feita para verificar se o sistema alcança ou não essa determinada situação, partindo de uma situação específica.

Dois tipos de alcançabilidade são considerados:- a alcançabilidade de uma configuração a partir da configuração inicial e a alcançabilidade de uma configuração a partir de uma configuração qualquer.

A verificação da alcançabilidade de uma configuração a partir da configuração inicial é bastante simples, pois, uma configuração de estado é alcançável se for um nó da árvore. Portanto, o algoritmo, que verifica essa propriedade, percorre a árvore em busca de uma dada configuração. Se ela estiver na árvore, fornece a seqüência de eventos que leva da configuração inicial de estados até essa configuração de estados.

Exemplos:

```
Entre com a configuracao:
lap,off

A configuracao <lap,off> e' alcancavel a partir da configuracao inicial
atraves da seguinte sequencia de eventos:
Evento: a
Evento: a
Evento: a/def
Evento: b
Evento: d
Evento: b
```

```
Entre com a configuracao:
minut

A configuracao <minut> e' alcancavel a partir da configuracao inicial
atraves da seguinte sequencia de eventos:
Evento: a
Evento: c
```

Na alcançabilidade de uma configuração a partir de uma configuração qualquer, dada duas configurações de estados CS_1 e CS_2, verifica-se a existência de um caminho que leva de CS_1 a CS_2, e caso positivo, mostra-se a seqüência de eventos necessária para o sistema alcançar CS_2 a partir de CS_1.

Para este caso, ao fornecer as configurações CS_1 e CS_2, constrói-se uma nova árvore de alcançabilidade a partir da configuração CS_1, e então o problema se resume em verificar a alcançabilidade de CS_2 a partir da configuração inicial, que é CS_1.

Exemplos:

```
Entre com a primeira configuracao:
reg,on
Entre com a segunda configuracao:
mes

A configuracao <mes> e alcancavel a partir de <reg,on>
atraves da seguinte sequencia de eventos:
Evento: d
Evento: a
Evento: c
Evento: c
Evento: c
Evento: c
Evento: c


Entre com a primeira configuracao:
hora

Entre com a segunda configuracao:
lap,off


A configuracao <lap,off> e alcancavel a partir de <hora>
atraves da seguinte sequencia de eventos:
Evento: a
```

```
Evento: a
Evento: a/def
Evento: b
Evento: d
Evento: b
```

Reiniciabilidade

Para verificar se um Statechart é reiniciável, basta verificar se para toda configuração CS, alcançada a partir de CS_0, existe uma seqüência de eventos que a faça retornar à configuração CS_0.

O algoritmo discutido para alcançabilidade de uma configuração a partir de uma configuração qualquer pode também ser utilizado para verificar a propriedade de reiniciabilidade. Para essa finalidade, basta fornecer ao algoritmo todas as configurações alcançáveis de CS_0 como sendo a primeira configuração, CS_1 e, a segunda configuração, CS_2, a configuração CS_0. Se para toda configuração CS_1 fornecida existir uma seqüência de eventos que leva a CS_0, então o Statechart é reiniciável.

Impasse (Deadlock)

Um sistema modelado em Statecharts poderá ter possíveis deadlocks se atingir configurações de estados das quais nenhuma transição pode disparar e, portanto, em sua árvore de alcançabilidade aparecerão nós terminais. Aqui diz-se "possíveis deadlocks" pois existem sistemas que modelam algumas tarefas as quais possuem início e fim de processamento bem definido, e, neste caso, a árvore de Alcançabilidade gerada possuirá nós terminais, que não são deadlocks, mas que indicam o final de processamento dessas tarefas.

O algoritmo para verificação de "deadlocks" percorre a árvore toda em busca de configurações "terminais". Ao encontrar alguma configuração "terminal" é emitido um aviso de que existe um possível deadlock quando o sistema alcança essa configuração, e é exibida a seqüência de eventos que leva, a partir da configuração inicial, a essa situação. O Statechart da Figura 1 não possui situações de deadlock.

Uso de Transições

A propriedade de uso de transições garante que o sistema foi especificado corretamente, no sentido de não haver transições especificadas que nunca disparam, ou ainda, que nenhuma transição foi esquecida no diagrama Statechart.

Se uma transição nunca dispara, ou não foi especificada no modelo em Statecharts, ela não aparece como um arco da árvore de alcançabilidade. Dessa forma, dada uma transição, o algoritmo percorre a árvore recursivamente, em busca de arcos rotulados com o nome dessa transição. Se for encontrada é porque não foi esquecida, e em algum momento pode disparar, e portanto são fornecidas todas as configurações de estados das quais pode disparar. Caso contrário, haverá um aviso de que essa transição não foi especificada ou não é dispáravel.

Exemplo:

```
Entre com a transicao a ser verificada:
d [in(off)]
```

```
A transicao <d [in(off)]> pode disparar a partir da
configuracao <(0000(000)([(01)(10)]0)(000000))>
```

Seqüência Válida de Eventos

Diz-se que uma seqüência de eventos é válida se cada um dos eventos dessa seqüência causar uma mudança na configuração de estados do sistema modelado.

Essa propriedade tem como objetivo validar uma seqüência de eventos, isto é, dada qualquer seqüência de eventos, existe um algoritmo que percorre a árvore de alcançabilidade em busca de cada elemento dessa seqüência. A busca é feita em duas etapas: na primeira etapa obtém-se o primeiro elemento da seqüência fazendo uma busca recursiva na árvore, e caso esse elemento seja encontrado, a segunda etapa procurará os outros elementos na árvore. A segunda etapa será bem sucedida se, a partir do nó onde incide o arco cujo rótulo é o nome do primeiro elemento da seqüência, existir um caminho onde os eventos correspondem aos elementos restantes da seqüência dada. Se a segunda etapa da busca falhar, continua-se a buscar recursivamente o primeiro elemento da seqüência, passando para a segunda etapa, se for encontrado um novo arco com o seu nome. O algoritmo termina se conseguir validar a seqüência de eventos ou se chegar ao final do processamento recursivo.

Os nós história, criados pelo algoritmo de construção da árvore, serão úteis para verificar se uma seqüência de eventos é válida. O algoritmo que valida uma seqüência de eventos deve guardar o caminho percorrido numa lista, pois quando um nó história for encontrado, essa lista auxiliará no momento de restaurar a configuração de estados representada pelo nó história. Essa lista será também importante no momento da escolha entre um arco com um determinado nome e o arco com a extensão "/def" nesse nome. Isso tem a ver com o nó história, pois o evento que aparece nesses dois arcos pode levar a um nó história ou a uma configuração de estados "default", respectivamente, dependendo da informação contida na lista onde se encontra o caminho percorrido.

Exemplos:

```
entre com a sequencia de eventos:
a,c,c,c,d,b

A sequencia de eventos dada e' valida !


entre com a sequencia de eventos:
a,b,c

Sequencia invalida
```

5 Conclusões

As soluções encontradas neste trabalho para a verificação das propriedades dinâmicas de Statecharts, embora relativamente simples, são de certa forma originais, pois não existe nada publicado até o momento na literatura mostrando efetivamente como validar sistemas especificados com Statecharts. O sistema STATEMATE [Hare88a] faz isso, mas nao há nada publicado a respeito dos algoritmos utilizados.

Este trabalho é um ponto de partida muito importante, porque muitos estudos podem ser feitos a partir de agora, aproveitando suas idéias e criando novos algoritmos para verificar outras propriedades que possam vir a ser definidas sobre a árvore de alcançabilidade.

A existência desses algoritmos e sua integração ao ambiente STATSIM, contribuirá bastante para aumentar a funcionalidade desse ambiente, já que constituem uma nova alternativa para análise dos modelos de sistemas especificados em Statecharts, desenvolvidos dentro do ambiente.

Uma questão a ser considerada é o crescimento exponencial de uma árvore de alcançabilidade, dependendo da complexidade do sistema envolvido. Em Redes de Petri, esse aspecto tem sido discutido ao se utilizar as técnicas gráficas, baseadas na árvore de alcançabilidade, para a verificação de sistemas complexos. São sugeridos alguns métodos para gerenciar os limites dessa complexidade, que se resumem em fazer especificações e validações parciais do sistema [Boch80], [Leve87].

Em Statecharts existe também esse problema, pois para a construção da árvore de alcançabilidade todas as transições possíveis são consideradas, e para um sistema complexo, o número de possibilidades geradas pode ser extremamente grande. Um Statechart com muitos componentes ortogonais, cada um com muitos estados, tem um grande número de configurações de estados, o que torna impraticável a construção de sua árvore de alcançabilidade. Uma maneira de resolver esse problema é limitar o escopo dos testes, isto é, construir a árvore de alcançabilidade para partes críticas e isoladas do sistema [i-LO89]. Quando partes maiores do sistema necessitam ser testadas, uma possível solução seria eliminar algumas transições menos importantes e assim a árvore de alcançabilidade gerada teria um número menor de configurações. É importante ressaltar que essas restrições de limites do sistema devem ser feitas com muito cuidado, a fim de se garantir que os testes sejam realizados sem perda de informações e de forma eficiente.

Para um sistema complexo que pode ser modelado em vários níveis hierárquicos de detalhes, acredita-se que esse problema pode ser contornado. O algoritmo de construção da árvore de alcançabilidade poderá ser usado para construir uma árvore de alcançabilidade para cada um desses níveis hierárquicos, e a verificação desse sistema poderá ser feita parcialmente.

Referências

[Boav92] Boaventura, I. A. G. *Propriedades Dinâmicas de Statecharts.* Dissertação de Mestrado, ICMSC-USP, São Carlos, 1992.

[Boch80] Bochmann, G.; Sunshine, C.A. Formal Methods in Communication Protocol Design. *IEEE Trans. Comm.*, vol. COM-28(4), pp. 624-631, 1980.

[Hare87a] Harel, D. STATECHARTS: a visual formalism for complex systems. *Science of Comp.*

Programming, 8, pp.231-274, 1987.

[Hare87b] Harel, D. STATECHARTS: on the formal semantics os Statecharts. *IN Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*. Ithaca, New York, 1987.

[Hare88a] Harel, D. et al STATEMATE: a working environment for the development of complex reactive systems. *IN Proceedings of the Tenth International Conference on Software Engineering*. (Singapure, April), Washington D.C.,IEEE, 1988.

[Hare92] Harel, D. Biting the Silver Bullet: toward a brighter future for system development. *IEEE Computer*, pp.08-20, January 1992.

[i-LO89] i-LOGIX. *The STATEMATE Aproach to Complex Systems*. 1989.

[Leve87] Leveson, N. G.,Stolzy J. L. Safety Analysis using Petri nets. *IEEE Transactions on Software Engineering*, SE(13(3)), pp.386-397, 1987.

[Masi91] Masiero, P.C.; Fortes, R.P. de M.; Batista Neto, J.E.S. Edição e Simulação do Aspecto Comportamental de Sistemas de Tempo-Real. *Anais do XVIII Seminário Integrado de Hardware e Software, SBC*, Santos-SP, pp.45-61, 1991.

[Masi92] Masiero, P.C.; Boaventura, I.A.G.; Maldonado, J.C. Dynamic Properties of Statecharts: A Reachability Tree and Analysis of some Properties. *PANEL'92 XVIII Conferencia Latinoamericana de Informática*, Las Palmas de Gran Canaria, Espanha, 1992.

[Pete81] Peterson, J. L. *Petri Net Theory and The Modeling of Systems*. Prentice-Hall, Inc.,Englewood Cliffs, N.J., 1981.