UNIVERSIDADE DE SÃO PAULO

Notas Didáticas

Métodos e Técnicas para Análise e Projeto de Sistemas Reativos

Lúcio Felippe de Mello Neto

Adenilso da Silva Simão

São Carlos

Novembro de 2007

Observações

Esta Nota Didática está sendo elaborada a partir das anotações realizadas em sala de aula pelos alunos do primeiro semestre de 2006 e do primeiro semestre de 2007 da disciplina Métodos e Técnicas para Análise e Projeto de Sistemas Reativos. Esta Nota Didática contém textos e figuras extraídas das anotações dos alunos e não está finalizada.

Sumário

1 Definições Iniciais

Nesta seção são definidos e apresentados alguns conceitos relacionados aos Sistemas Reativos.

1.1 Definições de Sistemas

Sistema: Um sistema é um conjunto de elementos inter-relacionados que realizam um objetivo. Cada elemento do sistema não pode ser explicado de forma isolada e as transformações ocorridas em uma parte influenciam as demais. Um subsistema é um sistema que faz parte de outro sistema. Exemplos: Sistema Nervoso, Sistema Solar, Sistema Operacional, etc.

Sistema Reativo: Um sistema reativo (SR) é um paradigma de sistemas que reagem a estímulos (entradas) fornecidos pelo ambiente. O sistema reativo é contínuo, ou seja, deve estar sempre pronto para fornecer uma resposta. Seu comportamento é fortemente baseado em estados de forma que saídas diferentes são produzidas a partir de uma mesma entrada, pois a resposta depende do estado em que o sistema se encontra. Exemplos de Sistemas Reativos: Sistemas Embarcados (celulares, automóveis); Sistemas de Controle (metroviário); Sistemas de Supervisão e Monitoramento (controle de qualidade industrial).

1.2 Desenvolvimento de Software

Na elaboração de um software, é necessário que várias etapas sejam realizadas para se chegar ao produto final. Na Figura 1 são ilustradas as etapas de desenvolvimento do software. Uma das características do software de boa qualidade ocorre quando a sua implementação atende às expectativas do mundo real.

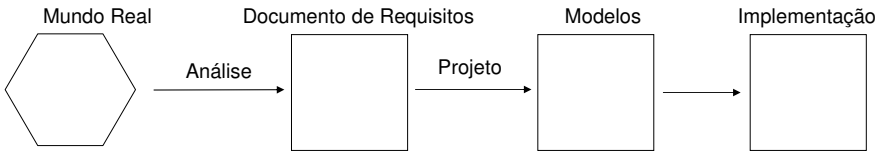

Figura 1: Etapas de Desenvolvimento.

Ao elaborarmos um software, existem diversas técnicas a serem empregadas a fim de garantir que o produto final seja de boa qualidade. A Engenharia de Software é um ramo da ciência da computação que utiliza métodos, processos e ferramentas com o intuito de se criar um software de melhor qualidade.

A Verificação e a Validação (V&V) (Figura 2) é um conjunto de atividades da Engenharia de Software que se aplica ao longo do processo de desenvolvimento do software para averiguar a qualidade do produto.

A verificação consiste em um conjunto de ações cuja meta é certificar se o modelo conceitual foi transcrito de forma adequada quando do uso da linguagem de programação. Já a validação consiste em procedimentos empregados para certificar se os valores gerados pelo modelo apresentam coerência com os gerados pelo sistema real.

Podemos caracterizar as técnicas de verificação com a seguinte frase: “Am I doing the things right?” (Estou fazendo as coisas de forma certa, de modo correto?). Toda técnica em que essa pergunta se encaixa é uma forma de verificação. Por exemplo, se todas as variáveis declaradas foram inicializadas, etc. Já as técnicas de validação podem ser caracterizadas pela seguinte sentença: “Am I doing the right things?” (Estou fazendo as coisas certas, as coisas que eram para serem feitas?). Por exemplo, as técnicas de validação visam garantir que a implementação cumpra com seus objetivos.

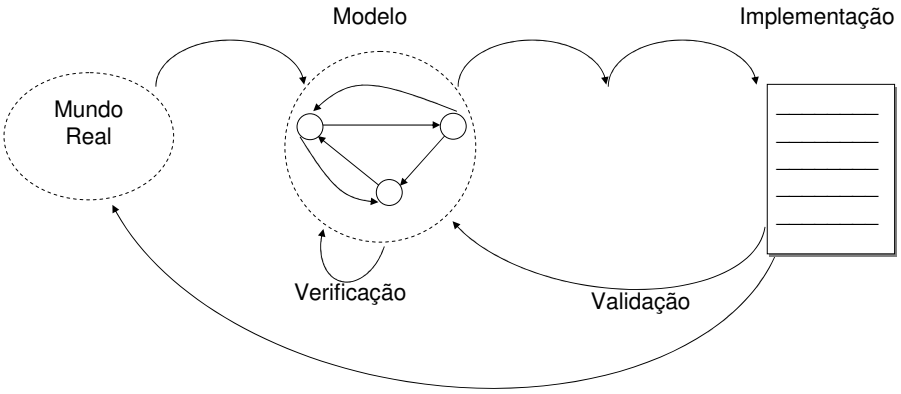

Figura 2: Verificação e Validação.

No contexto da validação, uma atividade muito importante é o teste de software. O teste é uma das formas de se fazer a validação e se destaca pela sua importância. Por essa razão que o teste é citado como um caso à parte: verificação, validação e teste (VV&T). O teste é um tipo de validação que envolve execução.

É importante lembrar que os modelos são essenciais, pois realizar a validação com o mundo real é extremamente difícil. Por essa razão, é interessante que os

testes se apliquem aos modelos. Estes testes são realizados diversas vezes a fim de se alcançar o modelo mais apropriado que sirva de base para a implementação.

Quando a etapa de teste não é realizada de forma adequada podem ocorrer danos ao patrimônio e à vida. A seguir são citados alguns dos bugs mais famosos:

- Therac 25 (radioterapia): reuso de código; pacientes mortos.
- Patriot (interceptação de mísseis): erro de arredondamento; 28 soldados mortos.
- Pentium (microprocessador): erro na unidade aritmética; prejuízo de 500 milhões de dólares.
- Ariane (veículo espacial): conversão de uma variável de ponto flutuante de 64 bits para 16 bits; explosão logo após o seu lançamento.
- Cobalt 60 (radioterapia): sem verificação, testes inadequados, especificação e documentação pobres.

1.3 Técnicas de Especificação de Sistemas Reativos

A especificação de Sistemas Reativos pode ser realizada por meio de diferentes técnicas:

- Máquinas de Transições de Estados
  - Máquinas de Estados Finitos
  - Statecharts

  - Redes de Petri

- Diversas extensões: Redes de Petri Temporais, Máquinas de Estados Finitos Estendidas, etc.

1.4 Máquina de Transição de Estados (MTE)

As máquinas de estados consistem em uma representação do sistema a partir dos estados em que o sistema pode estar e a transição entre eles. Em geral, esses sistemas possuem um número muito maior de possibilidades de resposta do que o número de entradas. Por exemplo, o bichinho virtual, o relógio digital, etc. Aperta-se um botão e obtém-se uma resposta. Em seguida, aperta-se o mesmo botão e a resposta é diferente da primeira. Isto é, a entrada pode ter sido a mesma, mas os estados em que a máquina se encontrava (na primeira e na segunda vez em que o botão foi apertado) eram diferentes.

A MTE é um modelo abstrato e pode-se representá-la por meio de uma dupla $< S, T >$, onde S é o conjunto de estados e T representa as transições de estados. Pode-se representar também uma MTE por meio de um tupla (s, e, s'), que significa que o estado s está ligado ao estado s' através do evento e.

As chances de se fazer validação em uma MTE é nula devido à complexidade inerente em uma representação. Por exemplo, um simples programa que possua uma variável do tipo inteiro contém mais de 65.000 estados. Ou quando há um sistema que possua infinitos estados. Desse modo, não há a possibilidade de realizar testes suficientes para garantir que a MTE infinita está correta ou incorreta.

Assim sendo, estudaremos três modelos para a representação de Máquinas de Transição de Estados: as Máquinas de Estados Finitos (MEFs), os Statecharts e as Redes de Petri.

1.4.1 Máquina de Estados Finitos

A Máquina de Estados Finitos (MEF) possui um conjunto finito de estados e transições. Normalmente os estados que representam um sistema não são finitos sendo necessário realizar uma abstração do sistema a fim de que ele possa ser representado por um número finito de estados. Essa abstração deve ser realizada sem que o novo sistema seja descaracterizado do sistema original.

Uma MEF é uma tupla $(X, Y, S, \delta, \lambda, s_0)$, onde:

- X é o alfabeto de entradas;
- Y é o alfabeto de saída;
- S é o conjunto finito de estados;
- δ é a função de transição, $X \times S \rightarrow S$;
- λ é a função de saída, $X \times S \rightarrow Y$;
- s_0 é o estado inicial.

Considerando a MEF da Figura 3, tem-se:

- $X = \{a, b\}$.
- $Y = \{0, 1\}$.

- $S = \{S_1, S_2, S_3\}$.
- $\delta = \{(S1, a) \rightarrow S2,\ (S1, b) \rightarrow S3,\ (S2, a) \rightarrow S1,\ (S3, a) \rightarrow S2,\ (S3, b) \rightarrow S1\}$.
- $\lambda = \{(S1, a) \rightarrow 0,\ (S1, b) \rightarrow 1,\ (S2, a) \rightarrow 0,\ (S3, a) \rightarrow 1,\ (S3, b) \rightarrow 1\}$.
- $s_0 = S_1$.

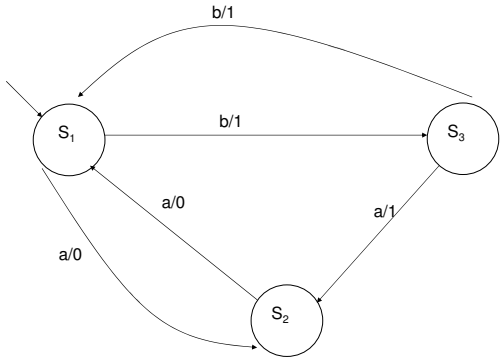

Figura 3: Representação Gráfica da MEF.

Existem diversas formas de representar uma MEF. A forma mais conhecida é por meio de um diagrama de estados (Figura 3). A MEF também pode ser representada por meio das tabelas de transições (Tabela 1). A MEF da Figura 3 pode ser representada por duas tabelas (Tabelas 1(a) e 1(b), que representam respectivamente o próximo estado atingido e a saída produzida pelas entradas) ou por apenas uma tabela (Tabela 1(c)).

Em algumas situações não há transições (por exemplo, o estado S_2 com a entrada b) definidas pela máquina. Nesses casos, pode-se ignorar a entrada, levar a MEF para um estado de erro ou produzir uma saída que é uma indicação do erro ocorrido. Quando não há transições definidas para todas as entradas, diz-se

Tabela 1: Representação Tabular da MEF.

δ	a	b
S_1	S_2	S_3
S_2	S_1	-
S_3	S_2	S_1

(a)

λ	a	b
S_1	0	1
S_2	0	-
S_3	1	1

(b)

δ / λ	a	b
S_1	S_2/0	S_3/1
S_2	S_1/0	-
S_3	S_2/1	S_1/0

(c)

que a MEF é *parcial*, enquanto que a MEF é *completa* quando a máquina possui uma transição definida para todas as entradas.

As MEFs são ditas *determinísticas* quando, a partir de uma entrada, tem-se apenas uma opção de transição de estado. Já nas MEFs *não-determinísticas*, podem existir duas ou mais possibilidades diferentes de transição para uma mesma entrada.

É possível obter uma MEF determinística a partir de uma MEF não-determinística, porém o número de estados irá aumentar. Em determinadas situações é preferível utilizar uma máquina não determinística de quatro estados a trabalhar com uma máquina determinística de 600 estados.

1.4.2 Statecharts

Os Statecharts são ótimos para a modelagem, mas ruins para a análise. Eles se resumem apenas à simulação. São muito utilizados por indústrias a fim de facilitar a representação do sistema para os funcionários do projeto.

É uma técnica que constitui uma extensão das Máquinas de Estados Finitos com a adição de características de *decomposição*, *ortogonalidade* e *broadcasting*. Os estados podem ser decompostos em subestados. Na Figura 4 é ilustrado um exemplo de statechart.

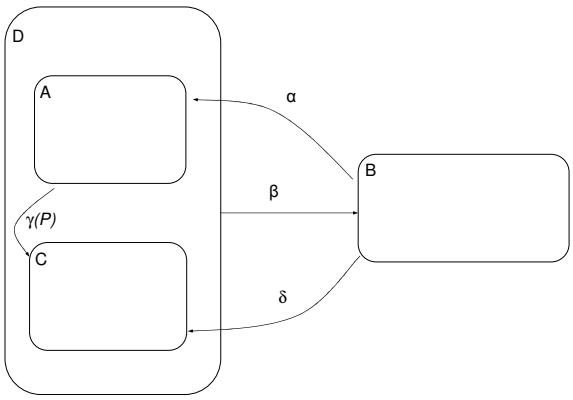

Figura 4: Exemplo de Statechart.

1.4.3 Redes de Petri

Há situações em que a representação é feita facilmente por uma Rede de Petri e difícil de ser feita por uma MEF, e vice-versa. Deve-se analisar a forma mais apropriada para se usar em determinado caso. As Redes de Petri entram como uma forma complementar na representação.

As Redes de Petri são representadas por um grafo (Figura 5). Dois tipos de nós: lugares (representados pelos círculos) e transições (representadas pelas barras). A execução da Rede de Petri é controlada pela posição e movimento de um marcador, chamado de *token*, que é representado por um ponto preto.

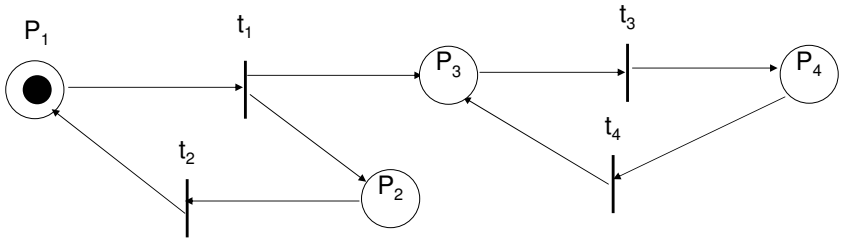

Figura 5: Exemplo de Rede de Petri.

2 Máquinas de Estado Finito

Como vimos anteriormente, uma MEF é uma máquina composta por estados e transições. As MEFs podem ser de dois tipos: Máquinas de *Mealy* e *Moore*. Uma máquina de *Mealy* depende da entrada e de seu estado atual, ou seja, as saídas são representadas a partir dos estados e transições. O uso de uma máquina de Mealy geralmente leva a uma redução de estados. A máquina de Mealy é mais flexível e pode-se obter uma máquina de Mealy a partir de uma máquina de Moore. Uma aplicação comum é o projeto de diálogo entre um programa (de computador) e o seu usuário. Neste caso, o diálogo poderia se dar de duas maneiras: ser comandado pelo programa ou pelo usuário. Apenas as máquinas do tipo Mealy serão utilizadas ao longo do texto.

Uma máquina de *Moore* depende somente do seu estado atual, ou seja, as saídas são representadas nos estados. Com isso, geralmente torna-se necessário aumentar o número de estados para representar saídas diferentes. Um exemplo comum de aplicação é o desenvolvimento de Analisadores Léxicos de compiladores

ou tradutores de linguagens em geral. Na Figura 6 é representada uma MEF de Moore onde em cada estado há uma saída (0 ou 1).

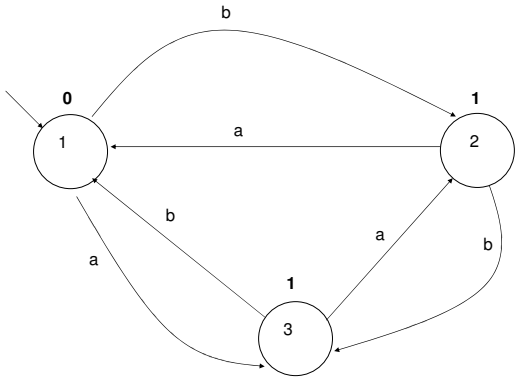

Figura 6: Exemplo de MEF de Moore.

2.1 Conceitos de MEFs

Considere a MEF da Figura 7 para ilustrar os conceitos relacionados as MEFs.

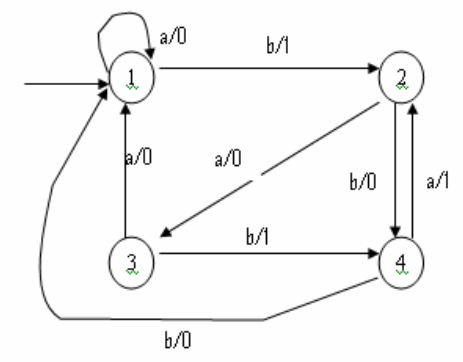

Figura 7: Exemplo de MEF.

Conectividade: Refere-se à conectividade entre os estados, ou seja, se todos os estados da MEF estão interconectados. Em relação à conectividade da MEF, há dois tipos:

- Fortemente Conexa: Uma MEF é fortemente conexa quando a partir de qualquer estado se consegue atingir um outro estado qualquer da MEF, mesmo que por intermédio de outros estados. A MEF da Figura 7 é fortemente conexa, por exemplo: $3 \rightarrow 1 = a, 1 \rightarrow 4 = bb, 2 \rightarrow 4 = b$.

- Inicialmente Conexa: Uma MEF é inicialmente conexa quando a partir de seu estado inicial consegue-se atingir todos os outros estados da MEF, mesmo que por intermédio de outros estados. Por exemplo, para a MEF da Figura 7 a partir do estado inicial 1 consegue-se atingir todos os outros estados: $1 \rightarrow 2 = b, 1 \rightarrow 3 = ba, 1 \rightarrow 4 = bb$.

Para descobrir se uma MEF é inicialmente conexa pode-se utilizar a Árvore Sucessora. A idéia da árvore sucessora é representar cada estado da MEF em um nó da árvore. O nó inicial é representado pelo estado inicial da MEF e as arestas representam as entradas. A partir do nó inicial, a árvore é estendida até que os nós sejam marcados como "velhos". Duas técnicas são utilizadas para marcar os nós como "velhos":

1. se o nó atingido já existe na árvore. Então o nó atingido é marcado como "velho" (representado com um *).

2. se o nó atingido já existe no caminho que leva ao nó inicial. Então o nó atingido é marcado como "velho" (representado com um *).

Na Figura 8 é ilustrada a árvore sucessora da MEF da Figura 7 utilizando a técnica 1.

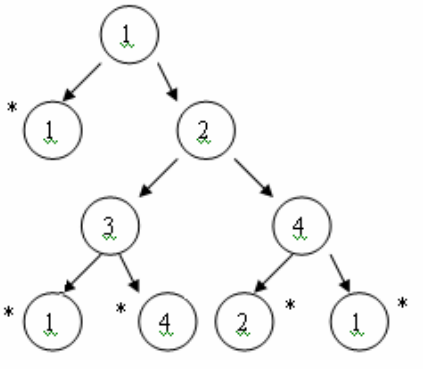

Figura 8: Árvore sucessora.

A partir do estado inicial 1, os estados 1 e 2 são atingidos com as entradas a e b, respectivamente. Como o nó 1 já existe na árvore, então o nó 1 é marcado com um "*" ("velho") e esse ramo não será mais expandido. A partir do estado 2, os estados 3 e 4 são atingidos com as entradas a e b, respectivamente. A árvore é expandida até que todos os nós folhas são marcados como "velhos".

A diferença entre as duas técnicas pode ser representada pelo nó folha 4. Utilizando a técnica 1, o nó é marcado, uma vez que ele já está presente na árvore. Utilizando a técnica 2, o nó 4 não seria marcado, uma vez que ele não está no caminho que leva ao nó inicial, que seria: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Com a primeira técnica é possível encontrar o menor caminho até o nó desejado.

Estados Equivalentes: Dois estados são ditos equivalentes quando possuem o mesmo comportamento para todas as entradas definidas. Dessa forma, não se consegue determinar em qual estado o sistema se encontrava antes de acontecer uma transição. Na Figura 9 é ilustrada uma MEF onde os estados 2 e 5 são equivalentes.

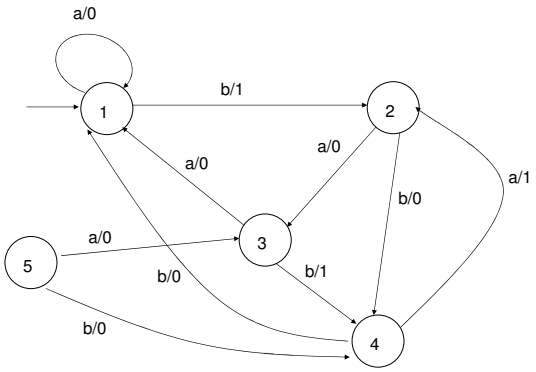

Figura 9: MEF com estados equivalentes.

Observa-se que, para a MEF da Figura 7, os estados 3 e 1 não são equivalentes, embora produzam as mesmas saídas. Isso pode ser visto com a entrada ba. Para o estado 1, a saída produzida é 10, enquanto que para o estado 3 a saída é 11. Desse modo, se observarmos as saídas podemos identificar em qual estado o sistema se encontrava.

Máquina Minimal: Uma MEF é minimal quando não possui estados equivalentes. É possível transformar uma MEF não-minimal em uma minimal realizando a junção dos estados equivalentes.

Seqüência de Sincronização (SS): Uma seqüência de sincronização é uma seqüência básica de símbolos de entrada que faz a MEF, partindo de qualquer estado, atingir um estado desejado. Essa seqüência pode não existir e não é necessariamente única.

Por exemplo, para a MEF da Figura 7 uma SS para o estado 1 seria $\{aaa\}$. Desse modo, a seqüência $\{aaa\}$ leva a MEF para o estado 1 partindo de qualquer estado em que a MEF possa estar.

Se existe uma SS para um determinado estado e a MEF for fortemente conexa então há uma SS para todos os estados da MEF. Por exemplo, as SS para os estados 2, 3 e 4 são $\{aaab\}$, $\{aaaba\}$ e $\{aaabb\}$, respectivamente.

2.2 Cálculo das Seqüências de Sincronização

Para se calcular as SS, constrói-se uma árvore em que arestas representam as possíveis transições e os nós representam os possíveis estados atingidos pelas transições. Uma SS é obtida percorrendo-se a árvore da raiz até o nó que possui apenas um estado. Para o nós repetidos, o procedimento é interrompido. Na Figura 10 está ilustrada a árvore que encontra uma SS do estado 1 para a MEF da Figura 7.

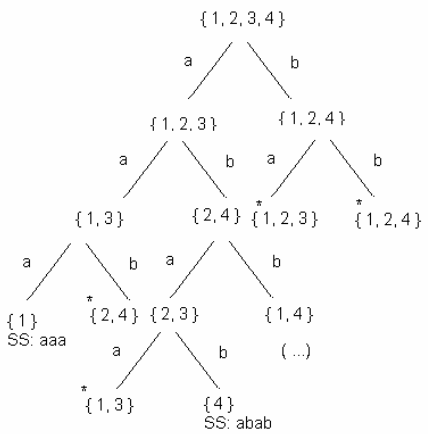

Figura 10: Cálculo da Seqüência de Sincronização.

As seqüências de sincronização encontradas foram $\{aaa\}$, que leva o sistema até o estado 1, e $\{abab\}$ que leva ao estado 4, não importando em qual estado a MEF se encontrava. O nó folha $\{1, 4\}$ poderia ser expandido e uma nova seqüência de sincronização poderia ter sido encontrada. A seguir, são ilustrados mais dois

exemplos. Considere a MEF da Figura 11, sendo que o cálculo da SS é ilustrado na Figura 12.

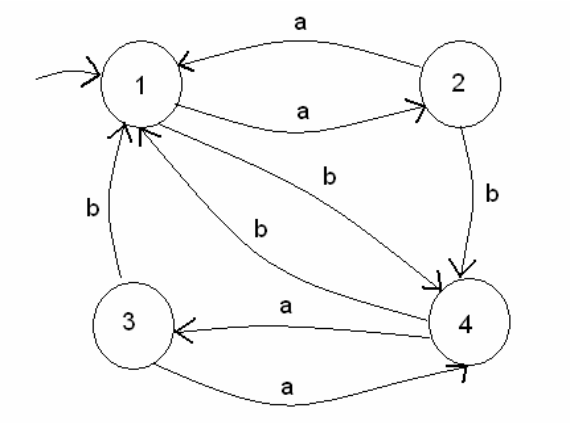

Figura 11: Exemplo de MEF.

Observando a Figura 12, nota-se que a MEF (Figura 11) não possui nenhuma SS.

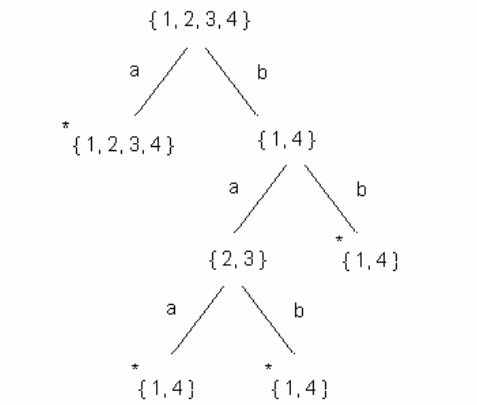

Figura 12: Cálculo da SS para a MEF da Figura 11.

Agora, considere uma MEF modificada (Figura 13) a partir da MEF da Figura 11. Como ilustrado na Figura 14, a MEF (Figura 13) possui uma seqüência de sincronização dada por $\{bb\}$.

Seqüência de Separação: é uma seqüência básica de símbolos de entrada que, quando aplicada em dois estados distintos, produz duas saídas distintas,

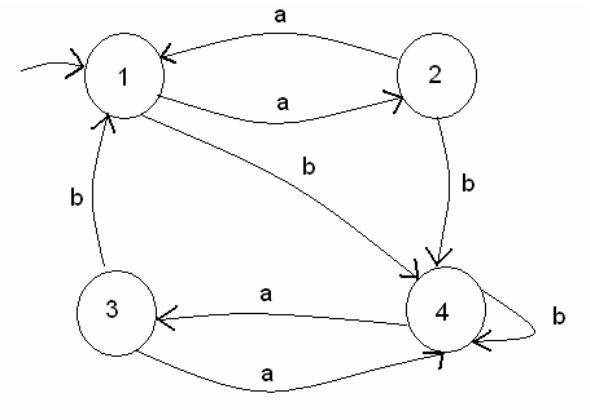

Figura 13: Exemplo de MEF.

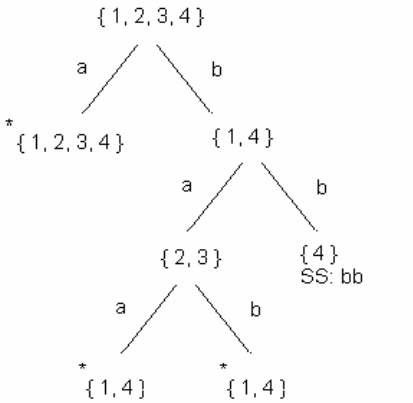

Figura 14: Cálculo da SS para a MEF da Figura 13.

ou seja, essa seqüência distingue dois estados. Sempre há seqüências de separação entre dois estados de uma MEF minimal e não são necessariamente únicas.

Por exemplo, a seqüência b é uma seqüência de separação entre os estados 1 e 2, pois as saídas resultantes (1 e 0, respectivamente) são diferentes. Dessa forma, conseguimos descobrir em qual estado o sistema se encontrava observando as saídas produzidas.

Seqüência de Distinção (DS): é uma seqüência básica de símbolos de entrada que diferencia todos os estados da MEF. Com a aplicação da DS pode-se

concluir em qual estado a MEF se encontrava antes de sua aplicação. Essa seqüência não é suficiente para se determinar qual estado a MEF atinge e nem toda MEF possui a DS.

Na Tabela 2 é ilustrada uma DS para a MEF da Figura 7. Para cada estado da MEF, a aplicação da seqüência ba resulta em saídas diferentes. A seqüência ba é uma DS e quando aplicada, podemos saber em qual estado a MEF se encontrava.

Tabela 2: Seqüência de Distinção.

	b	**a**
1	1	0
2	0	1
3	1	1
4	0	0

Seqüência Única de Entrada/Saída (UIO): é uma seqüência utilizada quando a MEF não possui a DS. Por definição, a seqüência DS é uma seqüência UIO para qualquer estado. A seqüência UIO pode não existir.

Na Tabela 3 é ilustrada a menor seqüência UIO para o estado 4. A seqüência a diferencia o estado 4 dos demais e, com isso, é possível saber em qual estado a MEF se encontrava antes de sua aplicação.

Como visto, a seqüência a é uma seqüência UIO para o estado 4 ($UIO(4) = a$), pois a saída produzida é 1, o que diferencia o estado 4 dos demais. Por exemplo, para os demais estados, poderíamos ter as seqüências: $UIO(1) = ba, UIO(2) = bb$ e $UIO(3) = bbb$.

Tabela 3: Seqüência UIO do estado 4.

	a
1	0
2	0
3	0
4	1

Conjunto W: é um conjunto de seqüências distintas que, quando aplicadas, são capazes de distinguir todos os estados da MEF. O conjunto W existe para todas as MEFs minimais. A seqüência DS é um conjunto W com apenas uma única seqüência. O conjunto formado por todas as seqüências UIO forma um conjunto W.

Na Tabela 4 são ilustradas as seqüências $\{a, ba\}$ que formam o conjunto W para a MEF da Figura 7.

Tabela 4: Conjunto W.

	a	**ba**
1	0	10
2	0	01
3	0	11
4	1	00

Como visto anteriormente, a seqüência UIO é utilizada para saber se a MEF se encontrava em um estado específico. Já a DS é utilizada para saber em qual estado a MEF se encontrava. A seguir, são apresentadas as técnicas para os cálculos das seqüências UIO, DS e do conjunto W.

2.3 Cálculo da Seqüência DS

O cálculo da seqüência DS também é realizado por uma árvore sucessora. A árvore é utilizada para calcular a menor seqüência possível. A DS é obtida quando o nó possui somente conjuntos unitários. Se um nó contiver um grupo de estados com estados repetidos, a expansão da árvore termina.

O nó raiz é obtido pelo conjunto de todos os estados da MEF. Em seguida, as entradas são aplicadas para obter os filhos sucessores do nó raiz. Para os novos nós obtidos, os estados subseqüentes à entrada aplicada são agrupados em conjuntos distintos considerando a saída que os estados produzem. Por exemplo, se a saída do estado 1 com a entrada a é diferente da saída do o estado 3 com a entrada a, então os estados 1 e 3 têm de serem separados em conjuntos diferentes enquanto que se o estado 1 e 2 produzirem saídas iguais para a entrada a eles devem ser colocados em um mesmo conjunto. Para cada grupo obtido, novos grupos são calculados formados pelos estados atingidos com as entradas. Se existir estados repetidos dentro de um mesmo grupo, a expansão da árvore termina. Novamente são aplicadas as entradas para a obtenção dos nós sucessores e o processo se repete até que em um nó possua apenas conjuntos unitários. Na Figura 15 está ilustrado o cálculo da DS para a MEF da Figura 7, onde a seqüência DS encontrada é ba.

Aplicando a seqüência ba para cada um dos estados presentes na MEF, verifica-se que as saídas obtidas são diferentes. Dessa forma, pode-se concluir em qual estado a máquina se encontrava. Na Tabela 5 5 são ilustradas as saídas produzidas com a entrada ba.

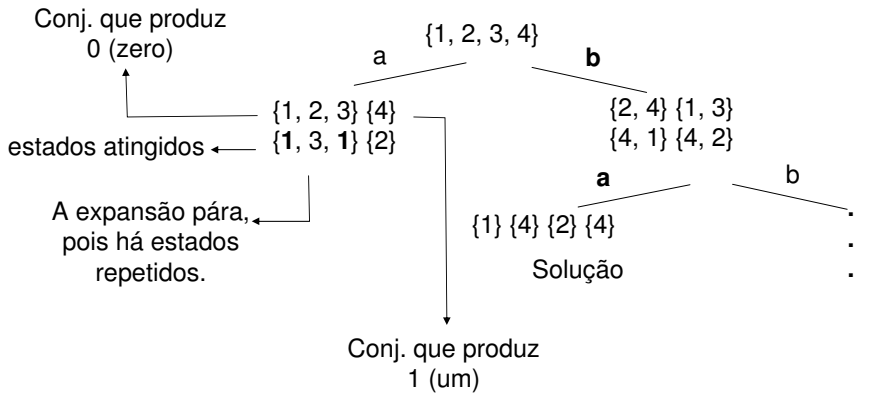

Figura 15: Cálculo da Seqüência de Distinção.

Tabela 5: Saídas obtidas com ba.

Estado	**Saída Obtida com** ba
1	10
2	01
3	11
4	00

2.4 Cálculo da Seqüência UIO

Para o cálculo da seqüência UIO uma árvore sucessora também é utilizada. O nó raiz é formado pelo estado que se deseja encontrar o UIO (estado escolhido) e por um conjunto formado pelos demais estados presentes na MEF. Em seguida, as entradas são aplicadas para obter os filhos sucessores do nó raiz. Os novos nós são formados por duas linhas. A primeira linha é formada pelos estados da MEF que produzem a mesma saída do estado escolhido. A segunda linha é formada

pelos estados atingidos aplicando-se a entrada nos estados da primeira linha. Esse procedimento é repetido até que o conjunto dos nós seja vazio. A expansão de um nó termina quando um estado escolhido também aparecer dentro do conjunto dos demais estados. Na Figura 17 é ilustrada a árvore para a obtenção da seqüência UIO do estado 1 para a MEF da Figura 16. Na Figura 18 é ilustrada a árvore para a obtenção da seqüência UIO do estado 5 da mesma MEF.

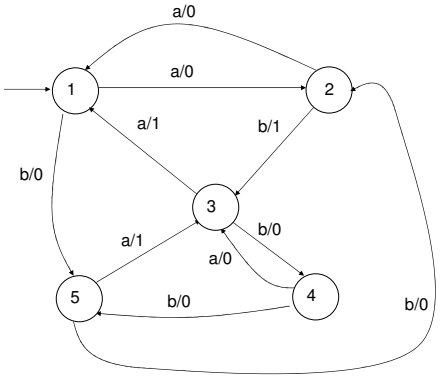

Figura 16: Exemplo de MEF.

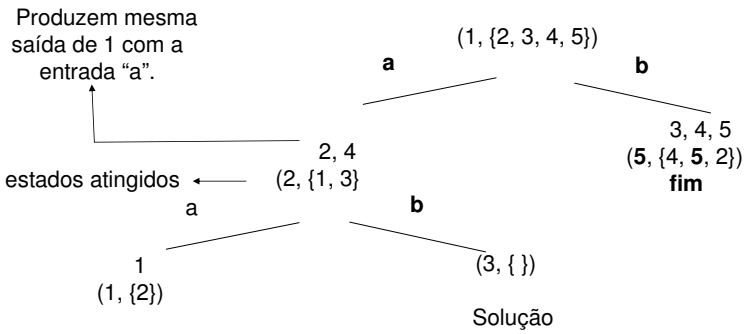

Figura 17: Cálculo da Seqüência UIO para o estado 1.

No exemplo da Figura 17, para a entrada a os estados 2 e 4 produzem a mesma saída do estado 1. Já os estados 3 e 5 produzem saídas diferentes e, portanto, são

retirados do conjunto. Em seguida, os estados atingidos são representados, sendo que a partir dos estados 2 e 4 os estados 1 e 3 são atingidos. A árvore termina com a entrada b, pois não é possível diferenciar o estado 5 dele mesmo. A solução encontrada foi $UIO(1) = ab$, pois há um conjunto vazio, ou seja, a seqüência ab diferenciou o estado 1 de todos os demais. Na Tabela 6, tem-se as saídas obtidas com a aplicação de ab.

Tabela 6: UIO do estado 1.

Estado	**Saída Obtida com** ab
1	**01**
2	00
3	10
4	00
5	10

Desse modo, consegue-se saber se MEF estava no estado 1, aplicando-se a seqüência $UIO(1) = \{ab\}$ e verificando se a saída obtida é 01. Se afirmativo, é a MEF estava no estado 1, caso contrário, pode-se apenas afirmar que a MEF não se encontrava no estado 1.

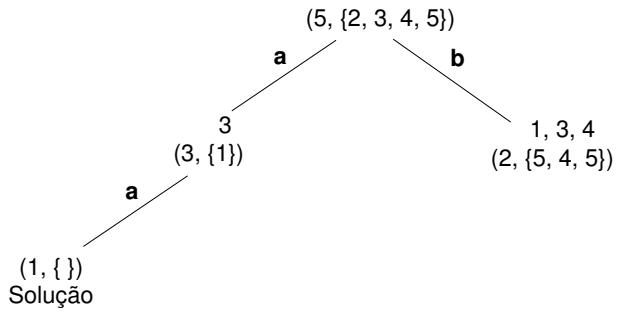

Figura 18: Cálculo da Seqüência UIO para o estado 5.

2.5 Cálculo do Conjunto W

Para a construção do conjunto W a representação em grafos é melhor em relação à representação em árvore. Toda MEF minimal possui o conjunto W, ou seja, se o conjunto W não puder ser encontrado a MEF possui estados equivalentes. A seguir é apresentada a construção do conjunto W para a MEF da Figura 16. A idéia é distinguir todos os pares de estados da MEF.

$W_0 = \{\}$. São todos os pares de estados que podem ser distinguidos com uma seqüência de tamanho 0 (zero).

$W_1 = \{(1,2,b),\ (1,3,a),\ (1,5,a),\ (2,3,a),\ (2,4,b),\ (2,5,a),\ (3,4,a),\ (4,5,a)\}$. Pares de estados que podem ser distinguidos com uma seqüência de tamanho 1.

$W_2 = W_1 \cup \{(1,4,aa),(3,5,aa)\}$.

$W_3 = W_2 \cup \{\} = W_2$. Fim do algoritmo.

O algoritmo termina quando não há mais pares a considerar ou quando ainda restam pares que podem ser distinguidos no próximo passo. Após o término do algoritmo o conjunto W é formado pelas seqüências distintas. Para esse exemplo, o conjunto W seria $\{a,b,aa\}$. As seqüências que são prefixas de outras podem ser retiradas. Dessa forma, a seqüência a pode ser retirada do conjunto, pois ela é prefixo de aa. O conjunto W obtido é $\{b,aa\}$.

O problema para se obter o menor conjunto W é NP-completo. Um conjunto $W_i \in W$ é um conjunto de seqüências que diferenciam o estado i dos demais. Por exemplo, para a MEF da Figura 16, os subconjuntos poderiam ser:

$W_1 = \{b, aa\}$. As seqüências b e aa são capazes de diferenciar o estado 1 dos demais estados.

$W_2 = \{b\}$. A seqüência b é capaz de diferenciar o estado 2 dos demais estados.

$W_3 = \{aa\}$. A seqüência aa é capaz de diferenciar o estado 3 dos demais estados.

3 Análise de Máquinas de Estado Finito

As seqüências UIO e DS são grandezas que nos permitem obter informações de uma determinada implementação para saber se ela está em conformidade com a sua especificação. Um dos problemas que existem quando se pensa em análise de sistemas reativos é como derivar seqüências de testes boas o suficiente para garantir que a implementação está satisfatória.

No contexto das MEFs, várias perguntas podem ser feitas em relação ao teste e análise do sistema: "A MEF é inicialmente conexa?", "A MEF é fortemente conexa?", "Existe alguma seqüência de entrada que produz um resultado inesperado?", "Quando podemos terminar de realizar o teste e saber se ele garante que a MEF foi implementada corretamente?".

Em relação aos testes em uma MEF, podemos considerar três símbolos: *reset*, *set* e o *status*. O *reset* tem por finalidade fazer com que a MEF volte ao estado inicial. O $set(s)$ tem por objetivo posicionar a máquina em um determinado estado s. O *status* tem por finalidade retornar o estado em que a máquina se encontra. Porém, normalmente estas funções não são muito confiáveis, pois o projetista mui-

tas vezes pode implementar essas operações de maneira errada. Dessa forma, as seqüências UIO e DS podem ser utilizadas para a realização dos testes. Por exemplo, a função *status* pode ser substituída pela DS, pois com a DS pode-se concluir em qual estado a MEF se encontra. Alguns dos testes possíveis de se realizar estão ilustrados na Tabela 7.

Tabela 7: Possíveis testes.

Teste	**Objetivo**
Reset status	Para saber se o estado inicial é o correto.
$Set(1)$ *status* $Set(2)$ *status* ... $Set(n)$ *status*	Para saber se a MEF realmente possui n estados.
$Set(1)$ $a/0$ *status* $Set(1)$ $b/0$ *status* ... $Set(n)$ $a/1$ *status* $Set(n)$ $b/0$ *status*	Para saber se a saída e o estado atingido com todas as entradas são os esperados.

Em determinadas situações a operação de *status* pode não existir. Dessa forma, para a realização dos testes, a seqüência de distinção (DS) poderia ser empregada. Na Tabela 8, são ilustrados os mesmos testes, mas com a utilização da DS ao invés da operação de *status*.

3.1 Utilização da DS

Como dito anteriormente, as operações $set(n)$, *reset* e *status* podem ter sido empregadas incorretamente ou não existir em determinadas implementações. Assim, uma forma mais confiável para a realização dos testes seria com a utilização da

Tabela 8: Possíveis testes com a DS.

Teste	**Objetivo**
$Reset\ DS$	Para saber se o estado inicial é o correto.
$Set(1)\ DS$ $Set(2)\ DS$ ... $Set(n)\ DS$	Para saber se a MEF realmente possui n estados.
$Set(1)\ a/0\ DS$ $Set(1)\ b/0\ DS$ ... $Set(n)\ a/1\ DS$ $Set(n)\ b/0\ DS$	Para saber se a saída e o estado atingido com todas as entradas são os esperados.

DS. A partir da DS podemos criar uma seqüência de verificação. Uma *seqüência de verificação* é uma seqüência de símbolos de entrada que serve para realizar os testes de forma completa, ou seja, se a seqüência de verificação é aplicada a uma implementação qualquer, e esta produzir as saídas esperadas, tem-se a garantia de que a implementação está correta. A seguir, são descritos os passos para a construção de uma seqüência de verificação utilizando uma DS.

A idéia é reconhecer (distinguir) todos as transições da MEF observando as saídas produzidas. Considere a MEF 4 da Figura 19 com a $DS = aa$. Os estados reconhecidos são marcados com seus respectivos números e os estados representados por X são os estados que faltam reconhecer. As transições são reconhecidas quando o estado anterior e o estado atingido são reconhecidos. Começa-se aplicando a DS no estado inicial para reconhecê-lo. Se o estado atingido não estiver reconhecido, então novamente a DS deve ser aplicada, senão a transição que não foi reconhecida é aplicada.

Começa-se pelo estado inicial:

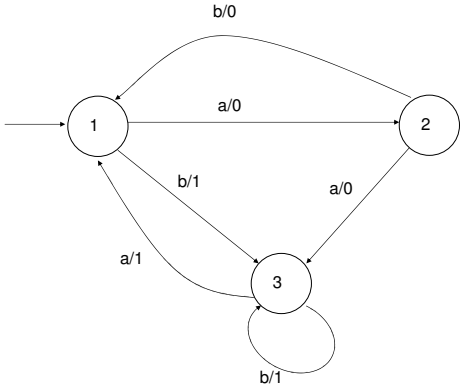

Figura 19: MEF 4.

1

Primeiramente, deve-se reconhecer o estado inicial aplicando-se o DS:

1 a X a X

Saída: 00 então o estado inicial está reconhecido.

O último X também é reconhecido pelo DS, pois a saída produzida é 10:

1 a X a 3 a X a X

Aplica-se DS no último X para reconhecê-lo:

1 a X a 3 a X a 2 a X a X

Aplica-se DS no último X para reconhecê-lo:

1 a X a 3 a X a 2 a X a 1 a X a X

Agora, sabe-se que o estado 1 com a entrada aa atinge o estado 3, desse modo, o último X já está reconhecido:

1 a X a 3 a X a 2 a X a 1 a X a 3

Como o último estado atingido já está reconhecido, deve-se escolher uma transição que ainda não foi reconhecida. Nesse caso, as transições do estado 3 com as entradas a e b não estão reconhecidas e qualquer uma poderia ser escolhida. Escolhe-se a transição com a entrada a:

1 a X a 3 a X a 2 a X a 1 a X a 3 a X

Aplica-se DS no último X para reconhecê-lo:

1 a X a 3 a X a 2 a X a 1 a X a 3 a 1 a X a X

Nesse momento, a transição do estado 3 com a entrada a foi reconhecida. Todos os anteriores devem ser reconhecidos:

1 a X a 3 a 1 a 2 a X a 1 a X a 3 a 1 a X a X

Conseqüentemente, a transição do estado 1 com a entrada a foi reconhecida. Todos os anteriores devem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a X a 1 a 2 a 3 a 1 a 2 a X

Conseqüentemente, a transição do estado 2 com a entrada a foi reconhecida. Todos os anteriores devem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3

O último estado (estado 3) já está reconhecido com a transição a. Escolhe-se então o b:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2

Faltam ainda as transições 2 com b e 1 com b. O último estado (estado 2) já está reconhecido com a transição a. Escolhe-se então o b:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3

Falta ainda a transição 1 com b. Então o estado 1 é atingido com a transição do estado 3 com a, que já está reconhecida:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1

Como o último estado (estado 1) já está reconhecido com a transição a, escolhe-se o b:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b 3 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b 3 a 1 a 2

Nesse momento, todas as transições estão reconhecidas e a seqüência de verificação obtida é: *aaaaaaaaaaabaabaaabaa*.

3.1.1 Otimização

Uma otimização pode ser realizada para a obtenção da seqüência de verificação com a utilização da DS. Essa otimização é ilustrada a seguir.

Começa-se pelo estado inicial:

1

Primeiramente, deve-se reconhecer o estado inicial aplicando-se o DS:

1 a X a X

Saída: 00 então o estado inicial está reconhecido.

Nesse momento, não é necessário reconhecer o último X, pode-se reconhecer o segundo X aplicando apenas mais um a:

1 a 2 a X a X

Aplica-se mais um a no segundo X para reconhecê-lo:

1 a 2 a 3 a X a X

Aplica-se mais um a no segundo X para reconhecê-lo:

1 a 2 a 3 a 1 a X a X

Nesse momento, os X restantes já estão reconhecidos:

1 a 2 a 3 a 1 a 2 a 3

O último estado (estado 3) já está reconhecido com a transição a. Escolhe-se então o b:

1 a 2 a 3 a 1 a 2 a 3 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 b 3 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2

Faltam ainda as transições 2 com b e 1 com b. O último estado (estado 2) já está reconhecido com a transição a. Escolhe-se então o b:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3

Falta ainda a transição 1 com b. Então o estado 1 é atingido com a transição do estado 3 com a, que já está reconhecida:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1

Como o último estado (estado 1) já está reconhecido com a transição a, escolhe-se o b:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b X

Aplica-se DS no último X para reconhecê-lo:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b 3 a X a X

Os dois X restantes já podem ser reconhecidos:

1 a 2 a 3 a 1 a 2 a 3 b 3 a 1 a 2 b 1 a 2 a 3 a 1 b 3 a 1 a 2

Nesse momento, todas as transições estão reconhecidas e a seqüência de verificação obtida é: *aaaaabaabaaabaa*.

3.2 Utilização de $UIOs$

A existência da DS facilita a realização dos testes em uma MEF, pois o conjunto de teste é formado pela seqüência de verificação. No entanto, a DS nem sempre existe. Nesse caso, as seqüências UIO podem ser utilizadas para a construção do conjunto de teste. Considere a MEF da Figura 20 com os valores: $UIO(1) = ab, UIO(2) = ba, UIO(3) = b, UIO(4) = a$. A operação *reset* é representada pelo símbolo r no início de cada seqüência de teste.

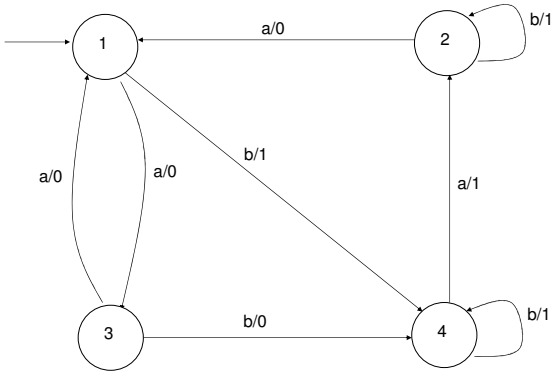

Figura 20: MEF 5.

Um conjunto de teste poderia ser criado para verificar todas as transições da MEF da seguinte forma. Cada transição é verificada adicionando-se a seqüência

UIO do estado atingido. Na Tabela 9 são ilustradas as seqüências obtidas com a utilização de $UIOs$.

Tabela 9: Conjunto de Teste com $UIOs$.

(Estado,entrada)	**Seqüências Obtidas**
$(1, a)$	$r\ a\ UIO(3) = r\ a\ b$
$(1, b)$	$r\ b\ UIO(4) = r\ b\ a$
$(2, a)$	$r\ b\ a\ a\ UIO(1) = r\ b\ a\ a\ ab$
$(2, b)$	$r\ b\ a\ b\ UIO(2) = r\ b\ a\ b\ ba$
$(3, a)$	$r\ a\ a\ UIO(1) = r\ a\ a\ ab$
$(3, b)$	$r\ a\ b\ UIO(4) = r\ a\ b\ a$
$(4, a)$	$r\ b\ a\ UIO(2) = r\ b\ a\ ba$
$(4, b)$	$r\ b\ b\ UIO(4) = r\ b\ b\ a$

No exemplo da Tabela 9, com a retirada das seqüências que são prefixos de outras, o conjunto de testes obtido é: $\{raba,\ rbaaab,\ rbabba,\ raaab,\ rbaba,\ rbba\}$. No entanto, esse conjunto não é completo, ou seja, ele não consegue verificar todos os erros de uma implementação. Por exemplo, suponha que uma implementação seja criada conforme a Figura 21 com uma alteração da transição $a/0$ do estado 3 de forma que atingisse o próprio estado 3.

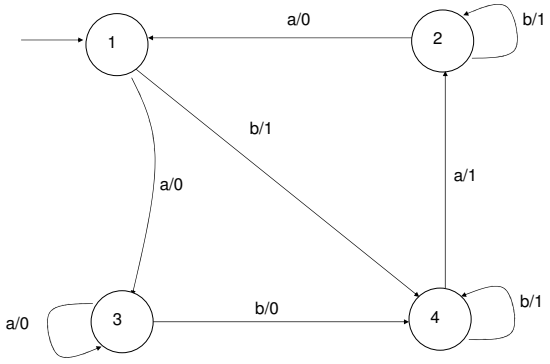

Figura 21: MEF 6.

O conjunto de teste criado anteriormente com a utilização das seqüências UIO não acusaria nenhum erro na implementação da Figura 21. Essa implementação passaria pelos testes mesmo com um erro, pois a alteração resultou em $UIO(1) = UIO(3)$, ou seja, a seqüência $UIO(1) = ab$ serviria para o estado 3 também. Para que esse erro fosse revelado, bastaria que a seqüência aab fosse adicionada ao conjunto de teste. Com essa seqüência o erro seria revelado, pois a saída correta de aab seria 001 e não 000.

Conclui-se que a utilização das seqüências UIO nem sempre produz um conjunto completo, pois podem existir máquinas diferentes com os mesmos $UIOs$. Uma forma para se construir um conjunto de teste completo seria aplicar todos os $UIOs$ em todos os estados. Com isso, a cardinalidade do conjunto de teste aumenta, mas esse conjunto se torna completo. Por exemplo, considere a MEF original (Figura 20). Na Tabela 10 está ilustrado parte do conjunto de teste produzido.

3.3 Utilização do Conjunto W (Método W)

Para a construção de um conjunto de teste, ainda pode haver situações em que a MEF não possua DS e a aplicação de todas as seqüências UIO em todos os estados pode resultar em um conjunto de alta cardinalidade. Nesse caso, o conjunto W pode ser utilizado. O conjunto W sempre existe para as MEFs minimais e sua aplicação produz um conjunto de teste completo. O conjunto W pode ser aplicado de forma análoga às seqüências UIO. Cada transição é verificada com a adição do conjunto W. Para evitar-se uma falha de inicialização, o conjunto W também

Tabela 10: Conjunto de Teste com todas as $UIOs$.

(Estado,entrada)	**Seqüências Obtidas**
$(1, a)$	$r\ a\ UIO(1) = r\ a\ ab$ $r\ a\ UIO(2) = r\ a\ ba$ $r\ a\ UIO(3) = r\ a\ b$ $r\ a\ UIO(4) = r\ a\ a$
$(1, b)$	$r\ b\ UIO(1) = r\ b\ ab$ $r\ b\ UIO(2) = r\ b\ ba$ $r\ b\ UIO(3) = r\ b\ b$ $r\ b\ UIO(4) = r\ b\ a$
$(2, a)$	$r\ b\ a\ a\ UIO(1) = r\ b\ a\ a\ ab$ $r\ b\ a\ a\ UIO(2) = r\ b\ a\ a\ ba$ $r\ b\ a\ a\ UIO(3) = r\ b\ a\ a\ b$ $r\ b\ a\ a\ UIO(4) = r\ b\ a\ a\ a$
$(2, b)$	$r\ b\ a\ b\ UIO(1) = r\ b\ a\ b\ ab$ $r\ b\ a\ b\ UIO(2) = r\ b\ a\ b\ ba$ $r\ b\ a\ b\ UIO(3) = r\ b\ a\ b\ b$ $r\ b\ a\ b\ UIO(4) = r\ b\ a\ b\ a$
...continua para os demais estados.	

deve ser aplicado após o *reset*. Por exemplo, considerando a MEF da Figura 20, o conjunto de teste obtido com a utilização de W está ilustrado na Tabela 11.

Como visto, o conjunto W deve ser aplicado em todas as transições de todos os estados. Dessa forma, o conjunto de teste produzido é completo. Para facilitar o entendimento, pode-se construir o conjunto de teste com a utilização de uma árvore de teste (Figura 22) construída da seguinte forma. Os nós representam os estados da MEF e as arestas representam as entradas da MEF. O nó inicial representa o estado inicial da MEF. Quando um nó atingido no nível i já existe nos níveis menores que i, então a expansão termina. Desse modo, a árvore obtida contém os caminhos parciais que percorrem todas as transições. Para cada nó da

Tabela 11: Conjunto de Teste com o conjunto W.

$reset$	$r\ W$
1	$r\ a\ W$ $r\ b\ W$
2	$r\ b\ a\ a\ W$ $r\ b\ a\ b\ W$
3	$r\ a\ a\ W$ $r\ a\ b\ W$
4	$r\ b\ a\ W$ $r\ b\ b\ W$

árvore, o conjunto W deve ser aplicado. Na Figura 22 é ilustrada a árvore de teste para a MEF da Figura 20.

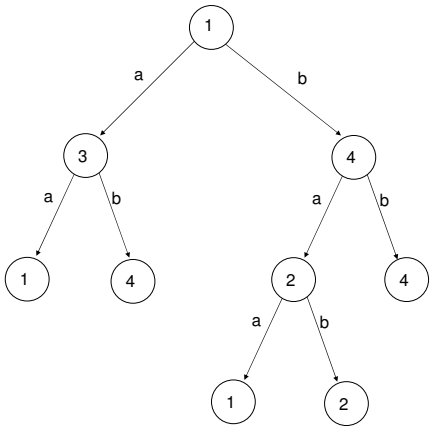

Figura 22: Árvore de Teste.

3.4 Representação de Sistemas por meio das MEFs

Após uma visão dos conceitos sobre as MEFs, verifica-se que elas possuem uma facilidade tanto para a representação de sistemas quanto para a realização de sua análise. Nesse contexto, tem-se a impressão de que as MEFs são capazes de representar qualquer sistema real. No entanto, é importante observar: As MEFs

possuem limitações e quais são elas? Se tais limitações não existem, por que existem outros métodos de representação e análises de sistemas reais, como por exemplo os Statecharts?

Para responder essas perguntas, utilizaremos um exercício prático para observar as dificuldades. Nas Figuras 23 e 24 estão representadas as modelagens de um relógio digital e de uma máquina de refrigerantes, respectivamente. As MEFs foram construídas pelos alunos em sala de aula.

3.4.1 Relógio Digital

A MEF da Figura 23 representa um relógio digital com suas principais funções: luz, alarme, cronômetro e data. Foram criadas quatro entradas: $E1, E2, E3$ e $E4$ para representar os quatro botões do relógio. Foram criadas 14 saídas: *luz*, *cronmetro*, *ajustehor*, *ajustemin*, *ajustedia*, *ajustemes*, *ajusteal*, *data*, *alarme*, *start*, *stop*, *reset*, *mudavalor* e *mudacampo*. Os estados representam: $S1 = hora$, $S2 = cronmetro$, $S3 = cronmetrocorrendo$, $S4 = ajustedoalarme$, $S5 = ajustedominuto$, $S6 = ajustedahora$, $S7 = ajustedodia$, $S8 = ajustedoms$. Observa-se que a seqüência "$E4$ $E3$" é uma DS.

3.4.2 Máquina de Refrigerantes

A MEF da Figura 24 representa uma máquina de refrigerantes que aceita moedas de R$ 0,25; R$ 0,50 e R$ 1,00. Foram criadas cinco entradas: $0,25$; $0,50$; $1,00$ (representando as moedas que a máquina aceita); *escolher* (escolha do tipo de refrigerante) e *cancelar* (botão cancelar ou timeout). Foram criadas três saídas:

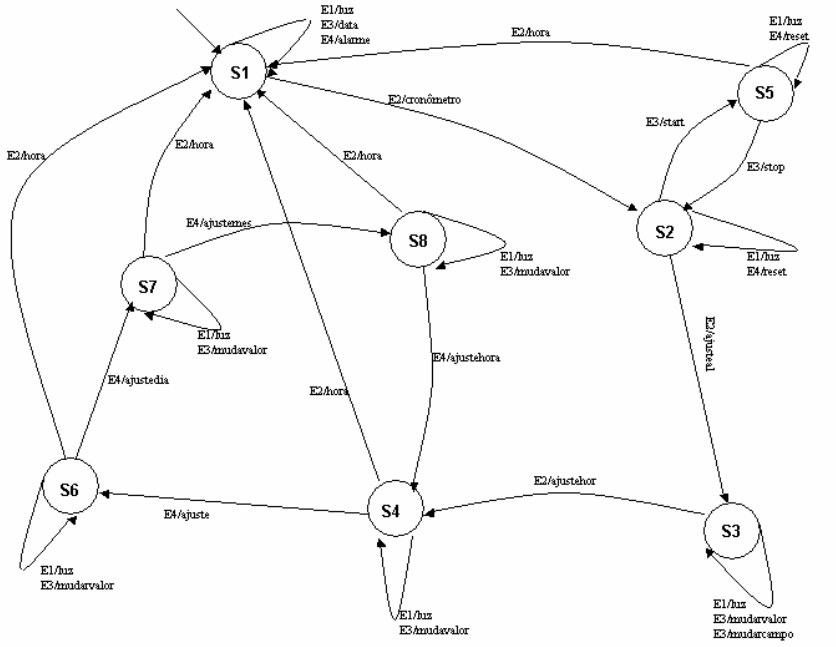

Figura 23: Relógio Digital.

$falta\ X$ (exibe na tela que faltam X reais); $troco\ X$ (devolve X reais); *Liberar* (libera refrigerante). A seqüência "$1,00$" é uma DS.

A partir da modelagem desses dois sistemas podem-se observar quais limitações existem com as MEFs. Primeiramente, observa-se que as MEFs não representam o tempo de cada operação, o que pode ser importante em determinadas situações.

Um problema relacionado às MEFs é a grande quantidade de estados necessários para representar o sistema. Por exemplo, se fosse necessário alterar a máquina de refrigerante para funcionar também com moedas de 10 centavos, oito estados deveriam ser adicionados (para funcionar com moedas de 10, 20, 30, 40, 60, 70, 80 e 90 centavos). Para que a máquina funcionasse com todas as moedas, seus estados aumentariam absurdamente.

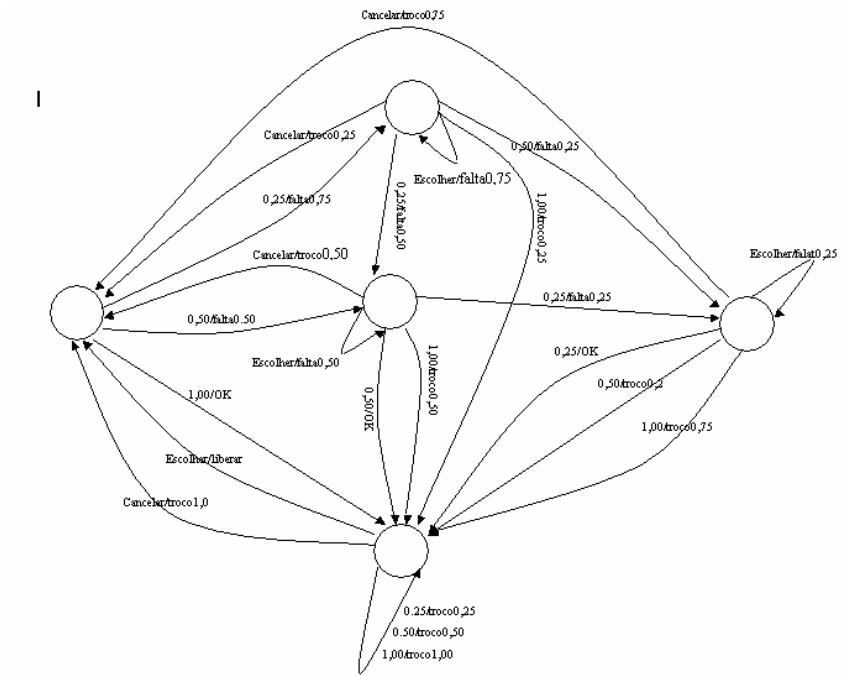

Figura 24: Máquina de Refrigerantes.

Para o relógio digital, se fosse necessário representar a função de $TIMER$, por exemplo, teríamos que colocar para cada estado existente mais um estado para representar o $TIMER$ ligado ou desligado. Isso dobraria o número de estados.

Esse aumento do número de estados é exponencial, conhecido como "Explosão de Estados". A Explosão de Estados é um fator que muitas vezes dificulta a representação de um sistema por meio das MEFs, já que aumenta muito o número de estados resultante da adição de novas funcionalidades. Outra característica que limita a utilização das MEFs é que não há a representação de paralelismo.

3.5 Explosão de Estados

Um dos problemas apresentados pelas MEFs é o da explosão de estados. O número de estados tende a aumentar muito na medida em que se aumenta o nível

de detalhe do que se deseja especificar. Assim, é necessário realizar simplificações para que a modelagem seja feita. Considere um exemplo do interruptor de luz ilustrado na Figura 25.

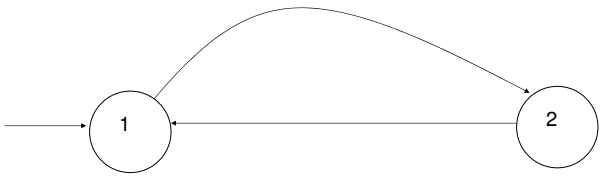

Figura 25: Estados do Interruptor.

Os 2 estados possíveis são: *Ligado* e *Desligado*. Com a adição de uma variável que representa a intensidade da luz com os valores ($fraca, mdia$ e $forte$) tem-se quatro possíveis estados (Figura 26):

- 1 = Luz apagada.
- 2 = Luz acesa com nível médio de intensidade.
- 3 = Luz acesa com nível baixo de intensidade.
- 4 = Luz acesa com nível alto de intensidade.

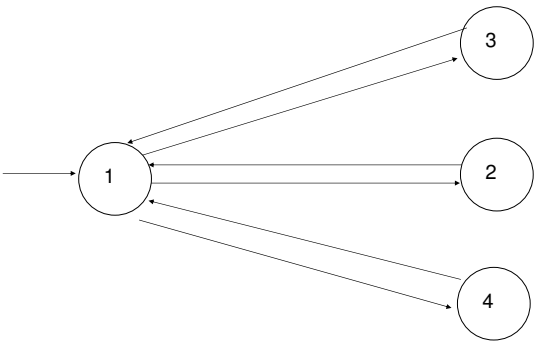

Figura 26: Explosão de Estados.

Se fosse acrescentada mais uma variável para representar, por exemplo, a cor da luz com três possíveis cores, então haveria a necessidade de dez estados.

4 MEFs Estendidas e Statecharts

Nesta seção serão apresentadas duas técnicas de modelagem de sistemas reativos: Máquinas de Estado Finito e Statecharts.

4.1 MEFs Estendidas

Para tentar-se evitar a explosão de estados, uma outra forma de representação pode ser utilizada para a modelagem de sistemas, chamada de Máquinas de Estados Finitos Estendidas. As MEFs Estendidas adicionam funcionalidades às MEFs originais, permitindo um maior poder de representação. No entanto, da mesma forma que há um ganho na representatividade, há uma perca em relação à validação do sistema.

As MEFs Estendidas possibilitam caracterizar os estados de um sistema de forma agrupada minimizando o problema da explosão de estados. Além disso, as MEFs não determinísticas podem ser transformadas em MEFs Estendidas determinísticas. As entradas da MEF podem ser parametrizadas permitindo a passagem de dados adicionais, que podem produzir resultados diferentes.

As MEFs Estendidas possuem variáveis de contexto. Essas variáveis são utilizadas para representar alguns estados da MEF e a configuração é dada pela combinação do estado mais o valor das variáveis de contexto. Por exemplo, a

representação do interruptor de luz de quatro estados (Figura 26) seria realizada com apenas dois estados por meio da MEF estendida (Figura 27). Os estados são: 1 = interruptor apagado; e 2 = interruptor acesso com nível = (intensidade).

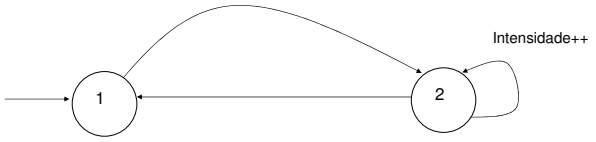

Figura 27: Interruptor com MEF estendida.

Outros conceitos relacionados às MEFs estendidas são: *Condição de Guarda*, *Parametrização* e *Clock*. Uma condição de guarda é uma condição que impõe um limite a uma variável de contexto. São úteis para descrever uma necessidade, um fator de permissão ou uma escolha. Por exemplo, na Figura 28 se a MEF está no estado A e recebe como entrada o símbolo a, ela vai para o estado B (produzindo 1 como saída) se a variável de contexto x for menor que 2, caso contrário a MEF continua no estado A (produzindo 2 como saída).

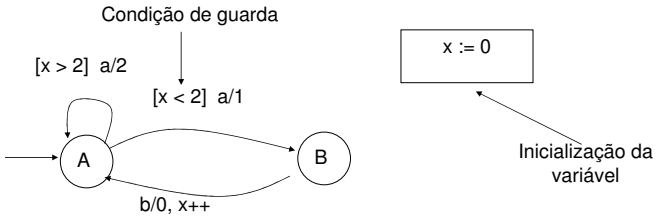

Figura 28: Exemplo de condição de guarda.

Com a utilização da condição de guarda, é possível transformar uma MEF não determinística em uma MEF estendida determinística. Por exemplo, a MEF da FiguraFigura 29 pode ser transformada na MEF estendida da Figura 30.

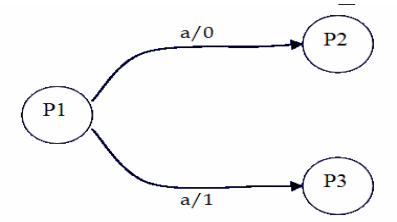

Figura 29: MEF não determinística.

A MEF da Figura 29 não é determinística, pois para a mesma entrada a no estado $P1$, é possível obtem como saída as respostas 0 ou 1.

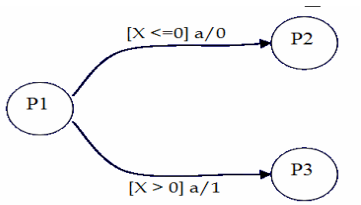

Figura 30: MEF estendida determinística.

A MEF da Figura 30 é determinística, pois com a entrada a no estado $P1$, tem-se a saída 0 se $X <= 0$ e a saída 1 se $X > 0$. Se não existir valores definidos para uma variável de contexto, então a MEF é parcialmente especificada. Por exemplo, considere a Figura 31. Observa-se que não está determinado qual saída deveria ser produzida quando $X = 0$.

A parametrização é utilizada quando a entrada pode ser utilizada com parâmetros. Por exemplo, na Figura 32 o parâmetro i é utilizado tanto na entrada quanto na condição de guarda.

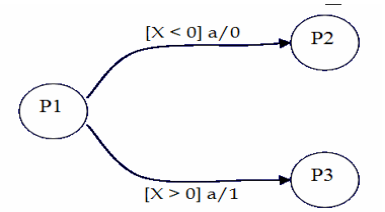

Figura 31: MEF estendida parcialmente especificada.

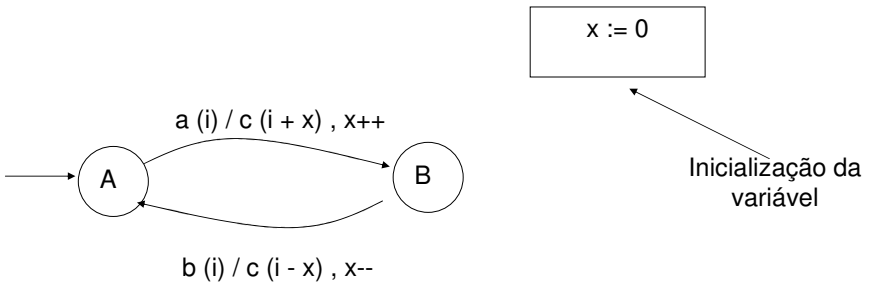

Figura 32: MEF com entrada parametrizada.

O clock é uma variável contínua ou discreta que representa o tempo e que está implícita no sistema (Figura 33).

4.2 Statecharts

Os Statecharts são extensões das MEFs com a adição de outros recursos, tais como, *decomposição*, *ortogonalidade*, *broadcast* e *história*. Com esses recursos adicionais, o paralelismo pode ser representado. Devido ao seu poder de representação, os statecharts têm sido muito utilizados na indústria.

A decomposição permite que a especificação possa ser construída de forma modularizada. É possível criar diversos estados sendo tratados como um único

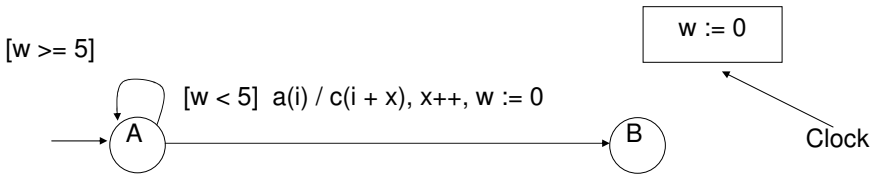

Figura 33: Exemplo de clock.

superestado ou que um superestado seja decomposto em vários subestados chamados de estados ortogonais. Cada um dos estados ortogonais é composto por um ou mais estados. Os superestados podem ser do tipo AND ou OR. No superestado OR, apenas um estado ortogonal pode estar ativo em cada instante, da mesma forma que nas MEFs. No superestado AND todos os estados ortogonais são ativados simultaneamente agindo como máquinas independentes. A característica de ortogonalidade refere-se a essa independência dos estados ortogonais. É necessário identificar no estado ortogonal qual será seu estado inicial. Por exemplo, na Figura 34 é ilustrado um statechart composto por um superestado, três estados ortogonais onde cada um deles possui dois, três e dois estados, respectivamente. Quando o superestado é ativado, os estados $P0$, $B0$ e $C1$ são ativados simultaneamente.

Devido à existência dos estados ortogonais, podem ocorrer situações em que mais de uma saída é produzida com a mesma entrada. Por exemplo, quando dois estados ortogonais possuem saídas com a entrada x. Qual das saídas deveria ser considerada? Ou quando um estado ortogonal e o superestado apresentam saída

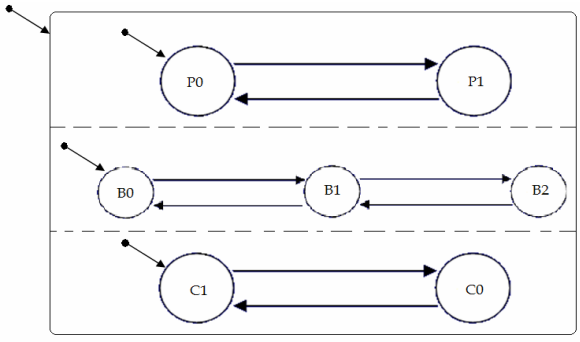

Figura 34: Exemplo de Statechart.

para a entrada x. Nesse caso, costuma-se considerar a hierarquia dos estados, dando preferência à saída gerada pelo superestado.

O mecanismo de *broadcasting* permite que um evento interno desencadeie ações nos diversos componentes ortogonais, possibilitando a modelagem da concorrência. O mecanismo de história permite ao modelo "lembrar" onde estava após sair de um superestado. Por exemplo, quando se ativa um superestado do tipo AND, todos os seus estados ortogonais ativam-se simultaneamente por default. Com a utilização da história, é possível ativar somente o estado ortogonal que estava ativo da última vez. Para ilustrar o uso da história considere o statechart da Figura 35. Quando o sistema é inicializado no superestado M o estado $P0$ é inicializado por default. No entanto, quando ocorre a transição do estado B para primeiro estado ortogonal do superestado, a transição é feita por história (símbolo H). Isto significa dizer que o estado a ser ativado, dentre $P0$ e $P1$, será aquele que estava ativo na última vez que o superestado M se encontrava ativo.

Uma outra opção de se utilizar a história é a história hierárquica (símbolo $H*$). Quando se utiliza a história, a recuperação do último estado só vale para o nível atual. Já no caso da história hierárquica, todos os níveis sofrem a recuperação

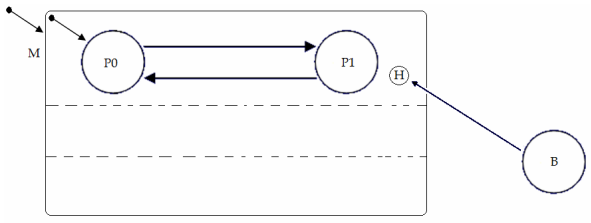

Figura 35: Exemplo de Statechart com história.

do último estado, inclusive os subníveis. Desse modo, ocorre uma recuperação em cadeia em todos os níveis existentes no modelo. Em resumo, existem três formas de ativação dos estados ortogonais quando o superestado é ativado:

1. Por default: quando os estados iniciais contidos em cada estado ortogonal são ativados. Ocorre quando o sistema é ativado, por exemplo.

2. Por estado específico atingido: quando um subestado (contido no estado ortogonal) é atingido fazendo com que o estado ortogonal seja ativado, e os outros ortogonais sejam ativados nos estados default.

3. Por história: em que o subestado ativado será aquele que estava ativo da última vez.

Os statecharts podem ser convertidos em MEFs. Por exemplo, para o statechart da Figura 36, tem-se a MEF da Figura 37.

Em comparação com as MEFs, os statecharts aumentam a capacidade de se modelar sistemas devido aos seus recursos adicionais. No entanto, há maior dificuldade para realizar-se a validação. Na Figura 38 é ilustrado um sistema simplificado de um relógio digital modelado com statechart.

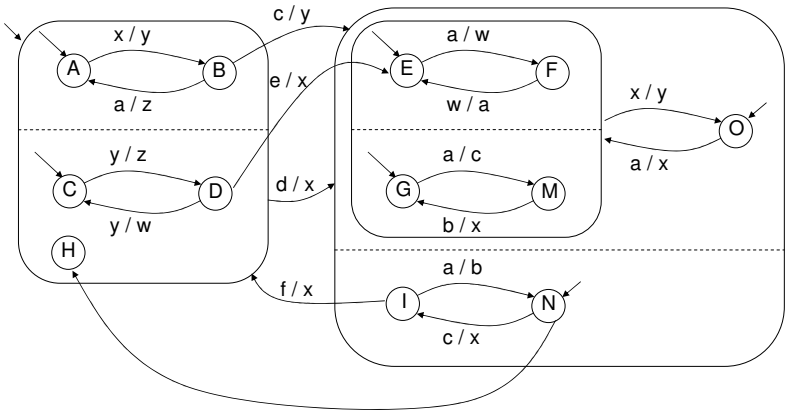

Figura 36: Exemplo de Statechart com história.

4.2.1 Métodos de Validação para Statecharts

Os Statecharts não possuem muitos métodos para análises conclusivas. Há duas formas utilizadas:

- Simulação: é realizada com a utilização de uma ferramenta de suporte. A simulação pode ser realizada de diferentes formas:
  - Interativa – onde as entradas são inseridas, uma a uma, interativamente. Desse modo, o comportamento do statechart pode ser observado em cada momento.
  - Em lote (batch) – onde são inseridas seqüências de eventos e o comportamento do sistema é observado posteriormente.
  - Exaustiva – onde todas as possíveis execuções do sistema são realizadas para posterior observação.

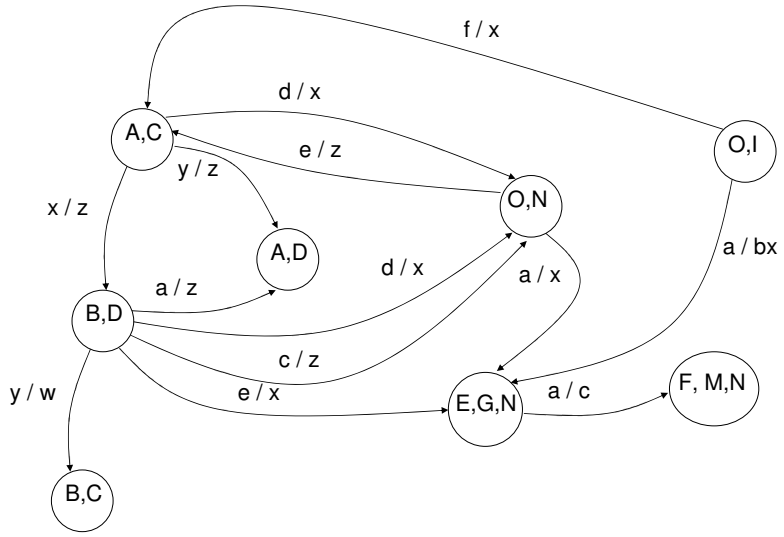

Figura 37: MEF a partir de Statechart (sem história).

  - Programada – similar à simulação em lote, mas pode-se delimitar quais seqüências serão utilizadas e quantas vezes serão executadas.

- Alcançabilidade: é realizada por meio de uma árvore de alcançabilidade (Figura 40)

A partir da Figura 39, é possível construir uma árvore de alcançabilidade, ilustrada na Figura 40.

Na árvore de alcançabilidade o nó inicial representa a configuração inicial do statechart. Para o exemplo citado, tem-se os estados $\{A, C\}$ ativos e a variável $i = 0$. Para cada nó atingido, todas as possibilidades de entrada devem ser ilustradas. A expansão termina quando há um nó já existente na árvore (nó velho). Na Figura 40, os nós velhos são marcados com um X.

Pode-se também produzir um grafo a partir da árvore de alcançabilidade. Se o grafo for finito, tem-se uma MEF, onde as técnicas relacionadas às MEFs podem

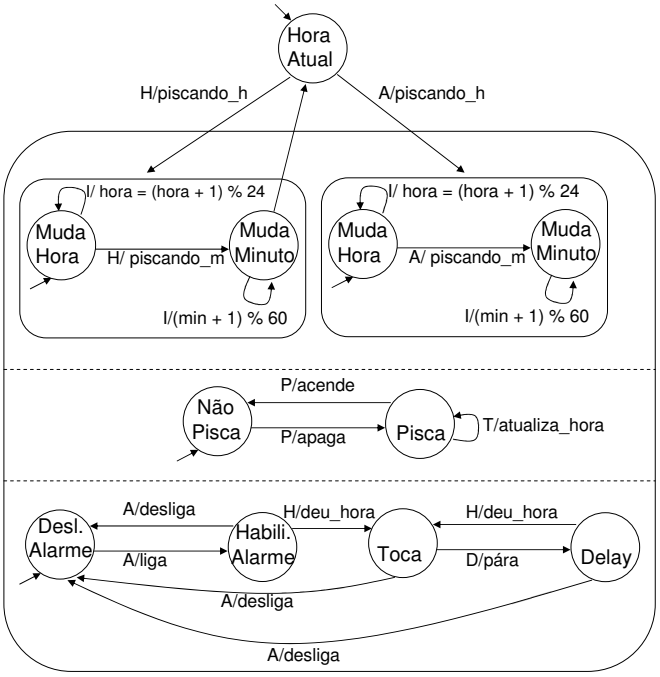

Figura 38: Statechart do Relógio Digital.

ser aplicadas. No entanto, o grafo pode ser muito grande e às vezes infinito, o que dificulta a obtenção de uma MEF. Na Figura 41 é ilustrado o grafo obtido a partir da árvore da Figura 40.

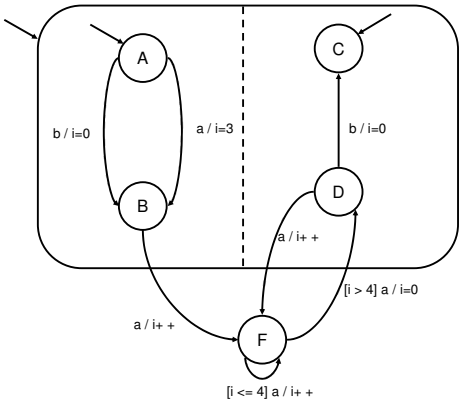

Figura 39: Statechart.

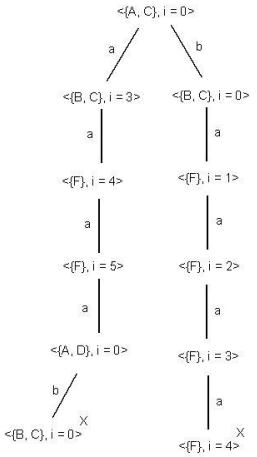

Figura 40: Árvore de Alcançabilidade do Statechart.

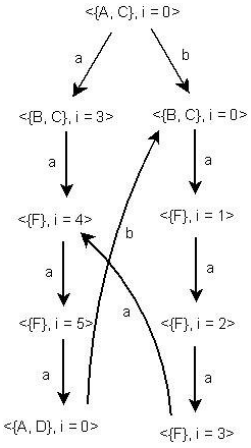

Figura 41: Grafo de alcançabilidade do Statechart.

5 Redes de Petri

As Redes de Petri (RP) foram introduzidas em 1962 com a tese de Carl Adam Petri cujo objetivo era a criação de um modelo para realizar a comunicação entre as máquinas de estados. Atualmente há diversas variações sendo utilizadas para a modelagem de sistemas.

Uma RP consiste em uma técnica para a especificação de sistemas que possibilita uma representação matemática e possui mecanismos poderosos de análise. As RP permitem a verificação de propriedades e da corretude do sistema especificado. As RP podem ser utilizadas, por exemplo, para descrever sistemas que lidam com concorrência, compartilhamento e sincronização.

As RP são representadas com um grafo (Figura 42). O grafo da RP modela as propriedades estáticas do sistema e possui dois tipos de nós: lugares (representados pelos círculos) e transições (representadas pelas barras). Os lugares e as transições são conectados por arcos direcionados. Se existe um arco partindo do nó i ao nó j (o arco parte de um lugar e chega a uma transição ou vice-versa), então i é uma entrada de j, e j, uma saída de i. Além das propriedades estáticas, a RP possui propriedades dinâmicas que são representadas pela simulação do grafo. A execução da RP é controlada pela posição e movimento de um marcador (chamado de *token*). Os tokens situam-se nos lugares, são representados por pontos pretos e são movimentados por meio das transições.

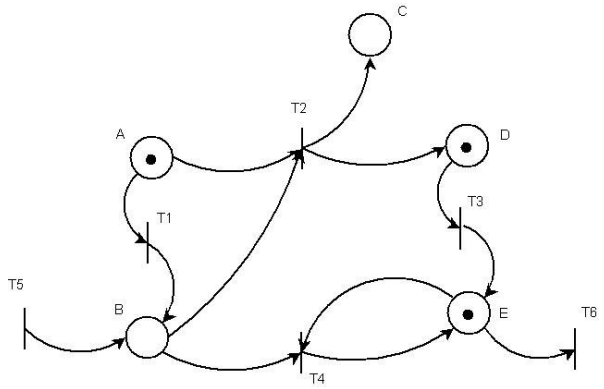

Figura 42: Exemplo de Rede de Petri.

Há diferentes tipos de RP que serão vistos a seguir.

5.1 Redes de Petri C/E (Condição/Evento)

Os eventos são elementos ativos e lugares são elementos passivos. Os lugares representam as condições. Na Figura 43a, o evento T1 "pode" ocorrer se a condição em S for satisfeita. Uma condição está satisfeita quando o lugar está marcado (43b).

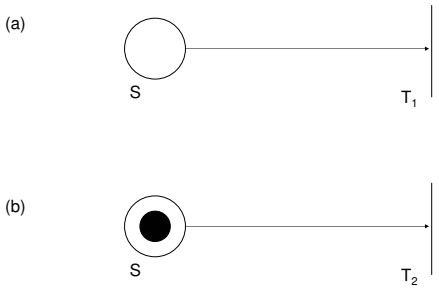

Figura 43: Exemplo de Eventos.

Regra de habilitação: para habilitar um evento é necessário que as pré-condições estejam satisfeitas e todas as pós-condições estejam não-satisfeitas. Por exemplo, na Figura 42 tem-se:

- $T1$: está habilitada para disparar porque o lugar antecessor (pré-condição) à transição possui token e o sucessor não.
- $T6$: está habilitada para disparar porque o lugar antecessor à transição possui token;
- $T5$: está habilitada para disparar porque o lugar antecessor à transição possui token (os lugares que não aparecem no grafo são considerados marcados);
- $T4$: é uma transição morta, pois não há nenhuma configuração que faça $T4$ disparar.

Na Figura 44, estão habilitados os eventos $T2$ e $T5$ e não estão habilitados os eventos $T4$, $T1$ e $T3$. O evento não necessariamente ocorre, mesmo que esteja habilitado.

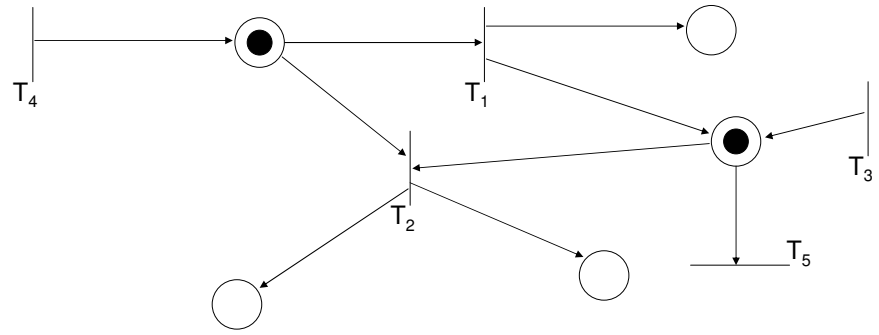

Figura 44: Eventos habilitados: $T2$ e $T5$.

5.2 Redes de Petri L/T (Lugar/Transição)

A RP lugar/transição é o modelo mais utilizado sendo uma generalização do tipo anterior. Para habilitar uma transição é necessário apenas verificar se o lugar antecessor (de entrada) possui *token*. Os lugares podem ter mais de um *token*. A transição está ativada quando todos os lugares de entrada possuem um *token*. A transição "retira" os *tokens* dos lugares de entrada e cria novos tokens nos lugares de saída da transição, independente da quantidade de *tokens* existentes nos lugares de saída. Na Figura 45 é ilustrado um exemplo de modelagem para o problema do produtor/consumidor.

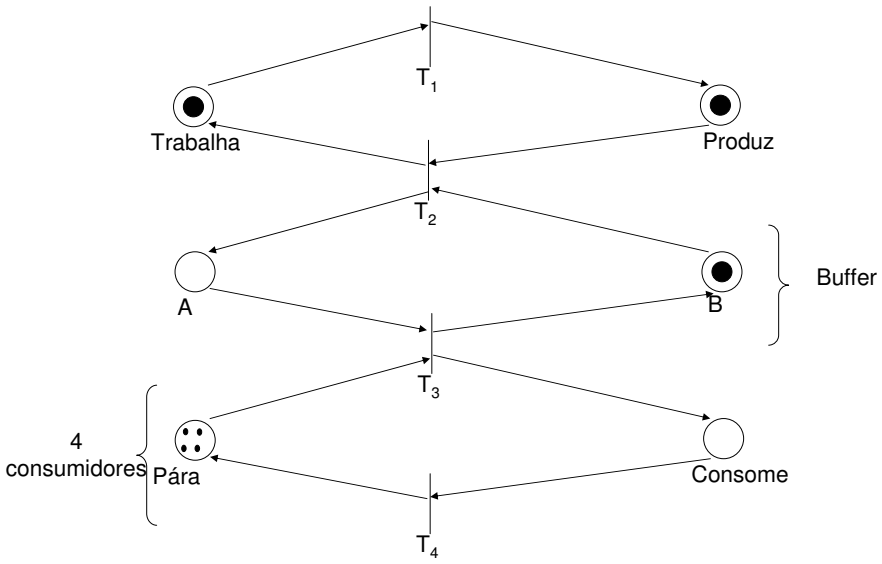

Figura 45: Exemplo Rede de Petri para Produtor/Consumidor.

Um problema clássico que pode ser modelado por uma RP é o da disputa de recursos. Considere o exemplo da Figura 46. O sistema não atinge a região crítica e isso é garantido pois não há a possibilidade de se ter *tokens* em $A2$ e $B2$ simultaneamente. De acordo com o modelo, se $A2$ possuir um *token*, $T3$ não estará habilitada, pois não haverá *token* em $R1$. Desse modo, $B2$ não possuirá *tokens*. Em seguida, se $T2$ for disparada, $T3$ estará habilitada. Se $T3$ for disparada, $B2$ possuirá token, mas $A2$ não possuirá *token*. Logo, $A2$ e $B2$ nunca possuirão marcas ao mesmo tempo, o que evita o acesso mútuo à região crítica.

Na Figura 47 tem-se uma modelagem falha com duas regiões críticas. Se disparar $T1$ e $T4$, não é possível dispara nenhuma outra transição. Isso representa uma situação de *deadlock*.

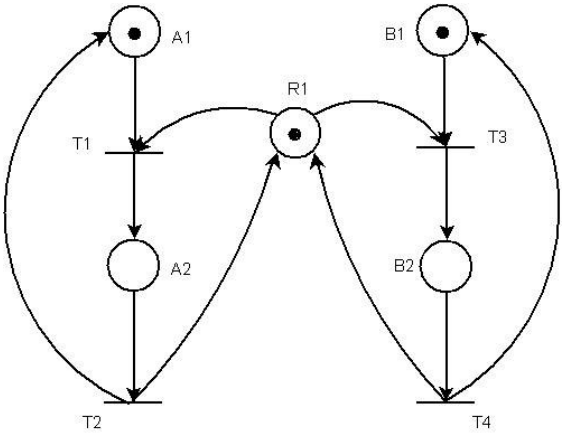

Figura 46: Modelagem do acesso à região crítica.

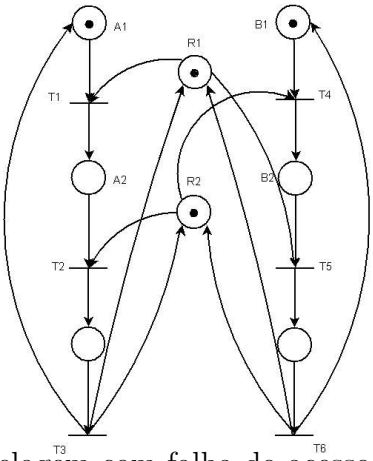

Figura 47: Modelagem com falha do acesso à região crítica.

Uma opção para evitar-se o deadlock seria a modelagem ilustrada na Figura 48, pois o recurso R2 nunca será acessado sem que R1 também seja acessado pelo mesmo sistema.

Outro problema clássico é o produtor/consumidor representado na Figura 49. Há dois lugares que representam o consumidor e dois lugares que representam o produtor.

Considerando o exemplo da Figura 49, há duas formas para modelar o sistema com dois produtores. A primeira consiste na criação de outro conjunto de lugares e transições (igual ao do produtor já existente) e ligá-lo ao buffer (semelhante

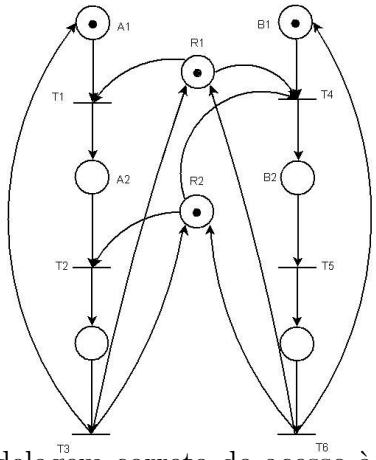

Figura 48: Modelagem correta do acesso à região crítica.

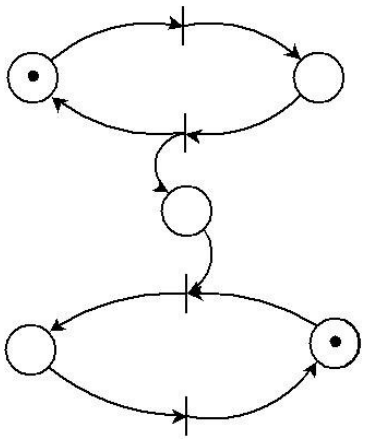

Figura 49: Modelagem do produtor/consumidor.

à Figura 42). A segunda consiste na inserção de mais um *token* no lugar que contém o *token*. Para criar mais produtores, basta inserir a quantidade referente de tokens. O mesmo ocorre para o consumidor. Basta inserir mais *tokens* no lugar que contém o *token* para aumentar o número de consumidores.

5.3 Rede de Petri Ponderada

As RP ponderadas seguem os mesmos conceitos das redes lugar/transição. Nas RP ponderadas, os arcos possuem pesos que representam a quantidade de tokens necessária para que a transição seja ativada. Na Figura 50 está representada uma RP ponderada antes e depois de um movimento. Para que ocorra a transição $T2$, é necessário que no lugar S tenha no mínimo três *tokens* e no lugar A tenha no mínimo duas *tokens*.

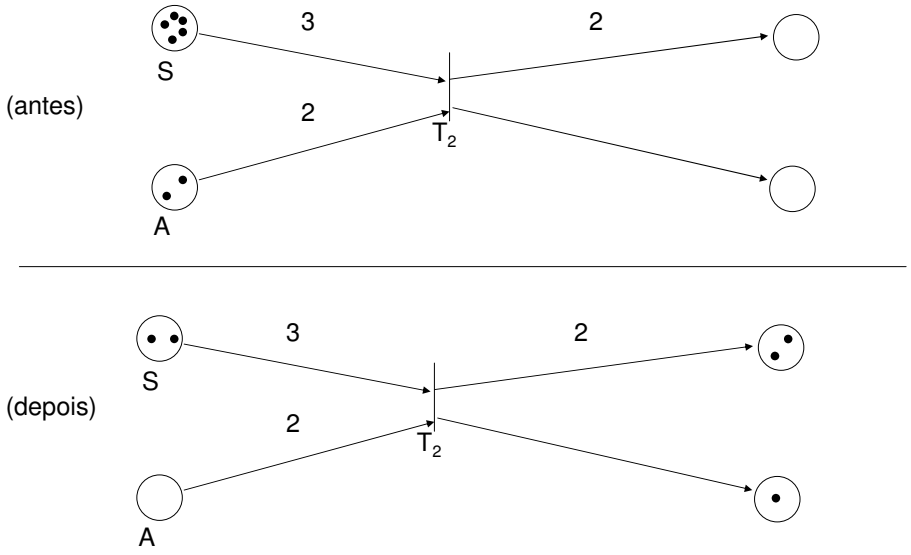

Figura 50: Exemplo Rede de Petri Ponderada.

Na Figura 51 está ilustrada uma modelagem do problema leitor/escritor com seis leitores. Nessa modelagem há uma falha, pois a prioridade é dos leitores e quando um deles está lendo, os escritores ficam impedidos de escrever. Neste caso, os leitores podem ler consecutivamente e impedir os escritores de realizar seu trabalho. Uma solução para o problema poderia ser resolvido com a modelagem da Figura 52.

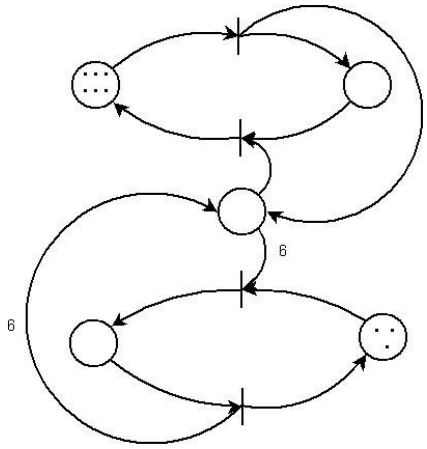

Figura 51: Rede de Petri Ponderada com falha para o leitor/escritor.

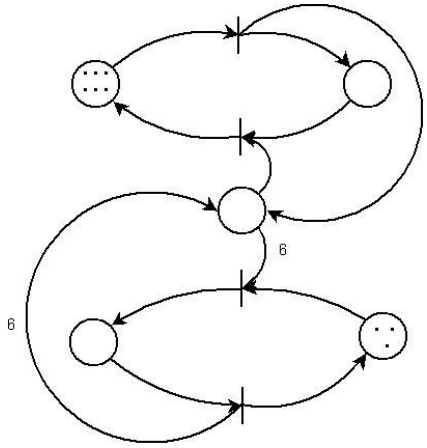

Figura 52: Rede de Petri Ponderada correta para o leitor/escritor.

5.4 Rede de Petri com capacidade de lugares

As RP com capacidade de lugares seguem os mesmos conceitos das RP ponderadas mas há uma quantidade máxima de *tokens* que o lugar pode conter. Para que uma transição ocorra é necessário que os lugares de entrada tenham a quantidade de tokens requerida e que o lugar de chegada não ultrapasse a quantidade de *tokens* permitida. Na Figura 53 é ilustrado um exemplo de RP com capacidade de lugares.

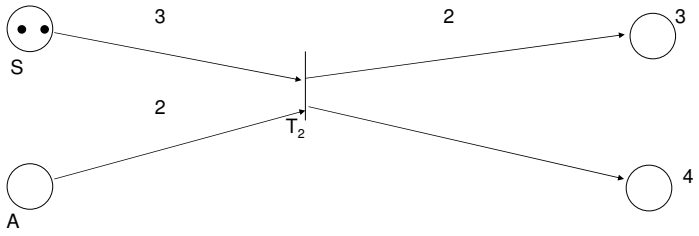

Figura 53: Exemplo de Rede de Petri com Capacidade de Lugares.

Na Figura 54 é ilustrada a modelagem de um parquímetro utilizando as RP com capacidade de lugares.

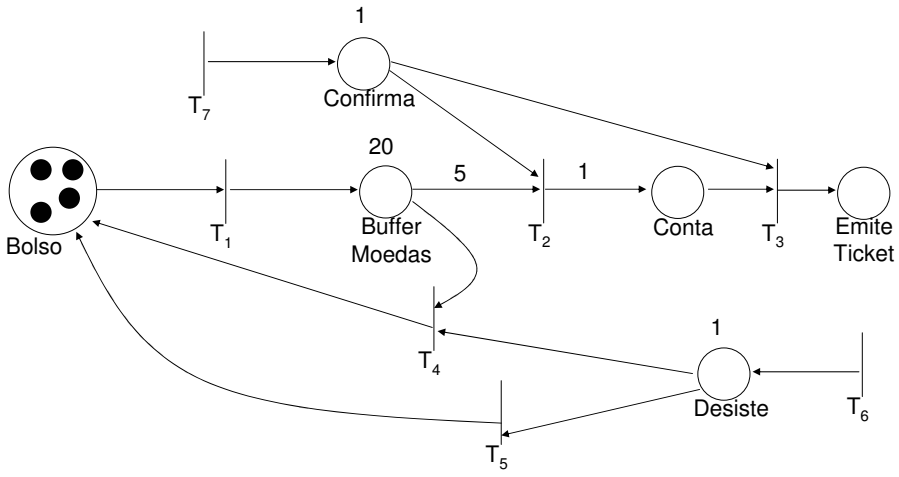

Figura 54: Rede de Petri do Parquímetro.

Relacionando as RP com capacidade de lugares com as RP de outros tipos pode-se concluir que:

- Nas redes do tipo condição/evento o peso dos arcos e o limite de tokens dos lugares são iguais a 1.
- Nas redes do tipo lugar/transição o peso dos arcos é 1 e o limite de tokens dos lugares é infinito.

As RP evitam a explosão de estados em determinados problemas. Por exemplo, na modelagem do produtor/consumidor, para aumentar-se a quantidade de produtores ou consumidores basta adicionar *tokens* nos seus respectivos lugares e a ponderação dos arcos permite representar o número de *tokens* que deve passar em cada disparo.

No entanto, há situações em que as RP também possuem o problema da explosão de estados, assim como nas MEFs. Para o problema do produtor/consumidor, haveria uma explosão de estados se houvessem dois ou mais tipos de produtos, ou se existissem diferentes tipos de consumidores. Isso ocorre, pois os tokens não podem ser diferenciados. Por exemplo, na Figura 55 é representada uma modelagem de uma produtora de canetas, mas com três produtores diferentes: carga, tubo e tampa. O consumidor necessita de pelo menos um produto de cada produtor para consumir e montar a caneta.

É possível verificar que, mesmo utilizando as RP, há situações onde a explosão de estados ocorre. Nesse caso, podem-se utilizar as Redes de Petri Coloridas.

5.5 Redes de Petri coloridas

As Redes de Petri Coloridas (*CPN - Color Petri Nets*) possuem marcas diferentes, ou seja, os *tokens* são de tipos diferentes. Para a construção de uma CPN,

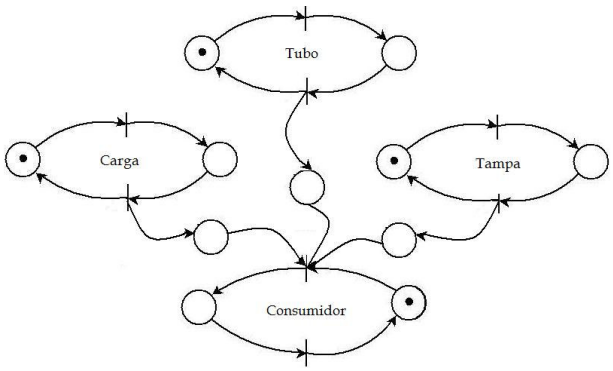

Figura 55: Exemplo de Explosão de Estados em Redes de Petri.

é necessário definir as cores e as variáveis que assumirão uma cor. Cada lugar está associado a uma cor, ou seja, o lugar poderá conter apenas elementos de uma determinada cor. Na Figura 56 tem-se um exemplo de CPN.

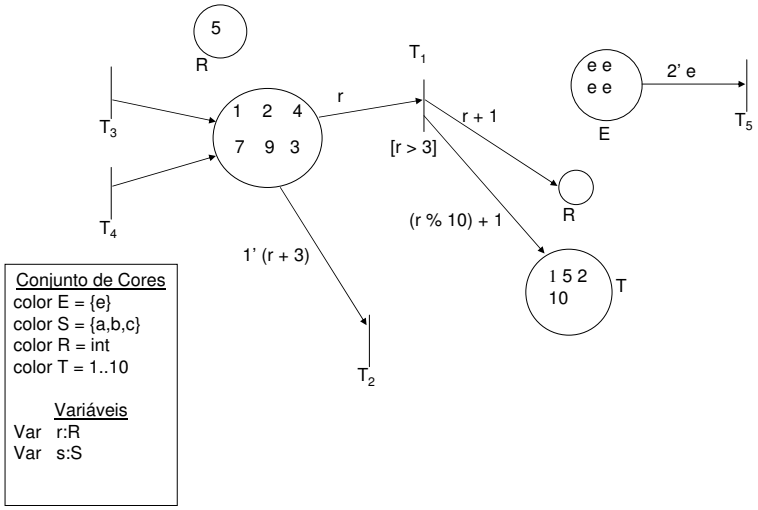

Figura 56: Exemplo de Rede de Petri Colorida.

Nas CPN os valores dos *tokens* podem ser testados e manipulados por meio de uma linguagem de programação funcional. Inicialmente, os *tokens* eram representados por cores ou por padrões que possibilitavam a distinção entre eles. Atualmente, os *tokens* são representados por estruturas de dados complexas e podem

conter informações. Os arcos realizam operações sobre os *tokens* e as transições são consideradas "expressões de guarda".

Cada lugar possui as seguintes especificações:

- Cor (especifica os tipos de tokens que podem residir no lugar). Exemplo: int, $\{d1, d2, d3\}$, $string$.
- Multi-Set marcação inicial (multi-conjunto de *tokens* coloridos). Exemplo: [1,1,1,2,3,4,4,5] $\rightarrow$ 3‘1 + 1‘2 + 1‘3 + 2‘4 + 1‘5.

No Multi-Set é possível realizar diversas operações de conjuntos:

- Soma: [1,1,2,3] + [1,4,4,5] = [1,1,1,2,3,4,4,5]
- Subtração: Para subtrair o conjunto B do conjunto A é necessário que B seja subconjunto de A. Exemplo: [1,1,2,3] - [2,3] = [1,2].
- Maior ou Menor: Para verificar se o conjunto $A > B$, o conjunto A deve conter todos os símbolos do conjunto B em quantidades iguais ou maiores. Exemplo: [1,1,2,3,3] > [1,1,3].
- Igual ou Diferente: Um conjunto A é igual a um conjunto B se possui todos os elementos de B nas mesmas quantidades. Exemplo: [1,1,2,3] = [1,1,2,3].

Cada transição possui as seguintes especificações:

- Variáveis: (para identificação). Exemplo: $P:\ prod$; $Q:\ prod$;.
- Guarda: (expressão que contém variáveis) Exemplo: [Q<>P].

Cada arco possui as seguintes especificações:

- Expressão (contém variáveis). Exemplo: "$if\ P\ =\ P1\ then\ C1\ else \ldots$".

Quando a expressão do arco é avaliada, um multi-conjunto de *tokens* coloridos é gerado. As expressões podem conter constantes, variáveis e funções. Na Figura 57 tem-se uma CPN para o problema do produtor/consumidor.

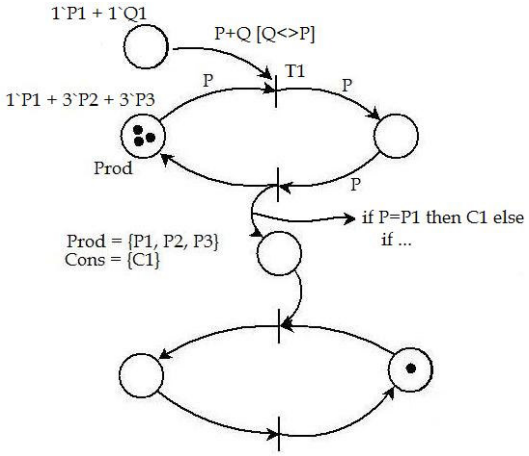

Figura 57: Exemplo de Rede de Petri Colorida para produtor/consumidor.

5.6 Propriedades de Redes de Petri

No contexto de RP, tem-se o conceito de "marcação". Uma marcação representa o estado de um sistema modelado por uma RP em um determinado instante, ou seja, é a configuração da RP em um determinado momento. Uma marcação da RP pode ser representada por meio de uma tupla formada pelos lugares da RP. Se o lugar não possui *token*, recebe o valor "0", caso contrário, recebe o valor representado pela quantidade de *tokens*. Por exemplo, a marcação da (Figura 58) é <1 0 0 1 0 0>, pois nos lugares $P1$ e $P4$ há apenas um *token*.

As RP possuem algumas propriedades básicas que permitem a análise do sistema modelado. As propriedades podem ser classificadas em dois tipos: estruturais ou comportamentais. As propriedades estruturais são aquelas que dependem da estrutura topológica da RP, ou seja, não dependem da marcação inicial. As propriedades comportamentais são aquelas que dependem da marcação inicial da RP, ou seja, sua ocorrência depende da configuração inicial da RP. A seguir são apresentadas as principais propriedades de uma RP.

- Limitação: um lugar é limitado se o número de *tokens*, em cada um dos lugares existentes, não exceda um número N finito. Para qualquer seqüência de ações executadas, nunca o número de *tokens* será maior que N. Se todos os lugares de uma rede são limitados, a rede é limitada.

Considerando a Figura 58, $P6$ é 1-limitado.

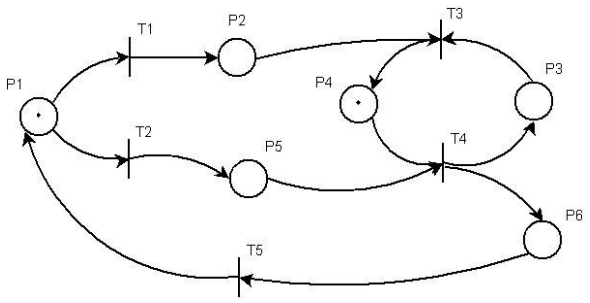

Figura 58: Rede de Petri A.

Na Figura 59 os lugares $P1$ e $P2$ são 3-limitados enquanto que $P3$ é ilimitado. Em uma modelagem, deve-se tomar cuidado em relação aos lugares ilimitados. Por exemplo, o lugar $P3$ não poderia representar um $buffer$.

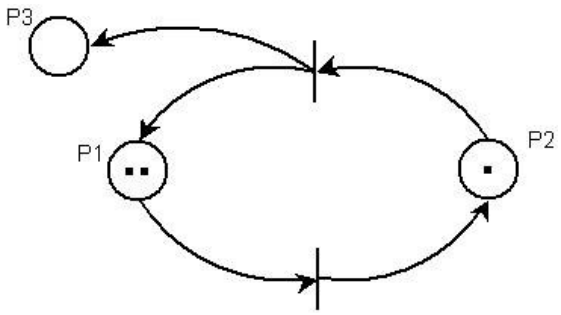

Figura 59: Rede de Petri B.

- Alcançabilidade: determina se uma marcação pode ou não ser alcançada. Desse modo, é possível verificar se uma determinada configuração pode ser obtida.

Na Tabela 12 são ilustradas algumas marcações da RP A (Figura 58).

Tabela 12: Marcações.

<1 0 0 1 0 0>	marcação inicial.
<0 0 0 1 0 1>	marcação não alcançável.
<1 0 1 0 0 0>	marcação alcançável após os disparos de $T2$, $T4$ e $T5$

Os algoritmos para o cálculo da alcançabilidade são decidíveis, porém são muito custosos.

- Cobertura: determina se uma marcação pode ou não ser coberta, ou seja, se existe alguma configuração que possua, no mínimo, determinada marcação. Desse modo, é possível verificar se um sistema pode atingir pelo menos a tal configuração.

Considerando a Figura 58, a marcação <1 1 1 0 0 0> cobre a marcação <1 0 1 0 0 0>. Na Figura 60, a marcação inicial é <0 0 1>. A marcação <3 0 1> não é alcançável, porém é coberta pela marcação <4 0 1>.

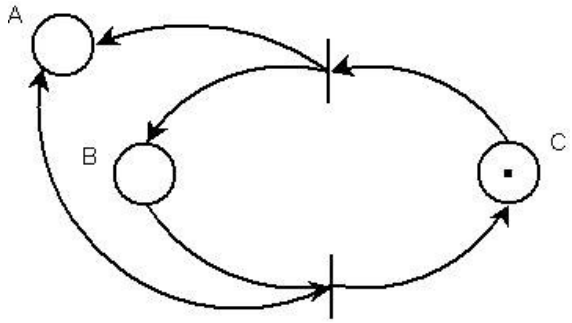

Figura 60: Rede de Petri C.

- Reversibilidade: é a capacidade da RP voltar para uma marcação anterior. Uma RP reversível possui uma marcação possível de ser atingida a partir de qualquer outra marcação em que a RP se encontre.

Na Figura 61 é ilustrado o exemplo mais simples de uma RP reversível.

Sistemas periódicos ou cíclicos sempre são reversíveis. Há um caso particular de reversibilidade, chamado de reversibilidade à marcação inicial. A RP da Figura 61 também é reversível à marcação inicial. Na Figura 62 tem-se uma RP reversível, porém não reversível à marcação inicial.

- Persistência: ocorre quando duas transições habilitadas podem ser sempre disparadas, não importando a ordem dos disparos. Em outras palavras, uma das transições é disparada a outra transição não fica desabilitada.

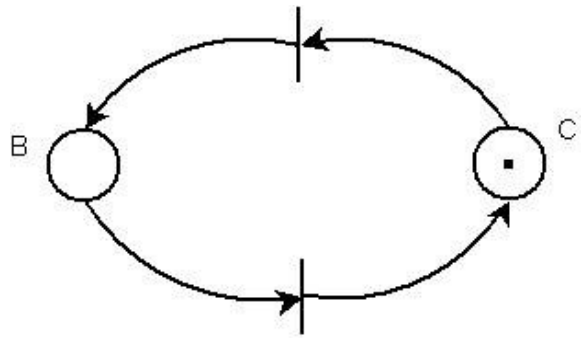

Figura 61: Rede de Petri reversível.

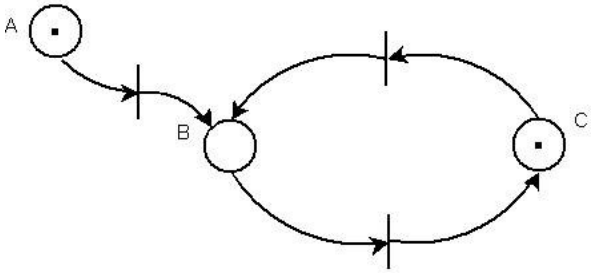

Figura 62: Rede de Petri não reversível à marcação inicial.

Na Figura 58 tem-se uma RP não persistente, pois ao disparar $T1$, inabilita-se $T2$. Já na Figura 62, tem-se uma RP persistente, pois sempre é possível disparar as duas transições habilitadas, independente da ordem de disparo.

- *Liveness*: determina se as transições estão vivas ou mortas. Tem-se as seguintes classificações:
  - $L0 - live$: morta, nunca pode ser disparada.

– $L1 - live$: viva, potencialmente disparável.

– $L2 - live$: k-viva, pode ser disparada k vezes, com k finito.

– $L3 - live$: ∞-viva, pode ser dispara infinitas vezes.

– $L4 - live$: pode ser disparada em qualquer marcação.

Na Figura 63 é ilustrado um exemplo de transição $L4$.

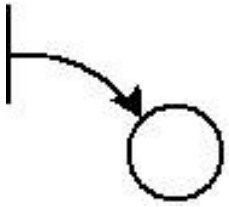

Figura 63: Exemplo de transição $L4$.

Na Figura 64, as transições $T0$, $T1$, $T2$ e $T3$ são dos tipos $L0$, $L1$, $L2$ e $L3$, respectivamente.

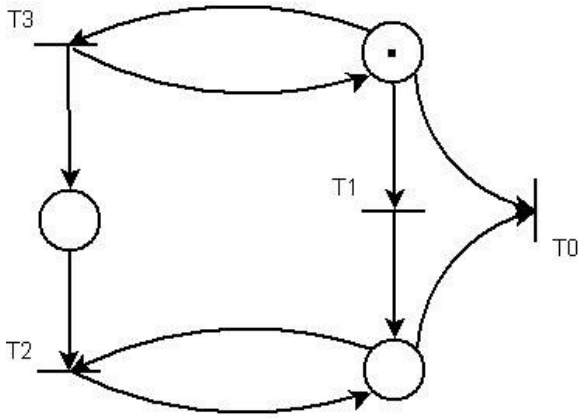

Figura 64: Rede de Petri F.

- Conservação: ocorre quando a soma de *tokens* de um conjunto é sempre constante. Se os lugares aparecem em um conjunto conservativo, podemos

classificá-los como limitados. A conservação é uma propriedade estrutural, enquanto que as demais propriedades são comportamentais.

Na Figura 58, os lugares $P3$ e $P4$ são conservativos. No entanto, a RP como um todo é não conservativa. Já na Figura 62, os lugares B e C não são conservativos. No entanto, a RP como um todo é conservativa.

5.7 Técnicas para Análise de Redes de Petri

Uma das técnicas utilizadas para a análise das RP é realizada pela matriz de incidência. Dada uma RP (Figura 65), pode-se representá-la por meio de uma matriz de incidência $C = O - I$ (Figura 66).

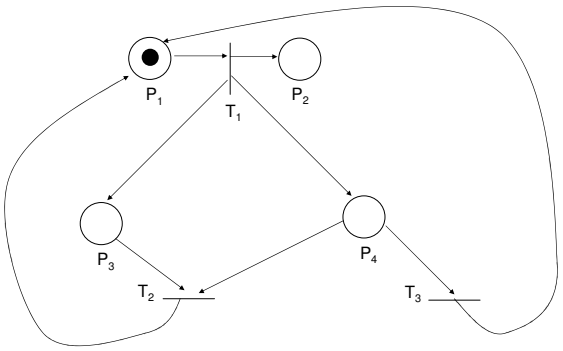

Figura 65: Rede de Petri G.

A matriz de entrada I representa quais lugares são entradas das transições. Se um lugar é uma entrada de uma transição, o valor na matriz é um, senão é zero. Por exemplo, o lugar P_1 é entrada para T_1.

A matriz de saída O representa quais lugares são saídas das transições. Se um lugar é uma saída de uma transição, o valor na matriz é um, senão é zero. Por exemplo, o lugar P_1 é saída para as transições T_2 e T_3.

A matriz de incidência C é calculada pela subtração da matriz O pela matriz I. Por exemplo, o lugar P_3 é saída para a transição T_1, é entrada para a transição T_2 e não possui relação com a transição T_3.

$$I = \begin{array}{c} \begin{matrix} P_1 & P_2 & P_3 & P_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix} \qquad O = \begin{array}{c} \begin{matrix} P_1 & P_2 & P_3 & P_4 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix}$$

$$C = \begin{array}{c} \begin{matrix} P_1 & P_2 & P_3 & P_4 \end{matrix} \\ \begin{pmatrix} -1 & 1 & 1 & 1 \\ 1 & 0 & -1 & -1 \\ 1 & 0 & 0 & -1 \end{pmatrix} \end{array} \begin{matrix} T_1 \\ T_2 \\ T_3 \end{matrix}$$

Figura 66: Matriz de Incidência.

A partir da matriz de incidência, é possível realizar alguns cálculos para a análise da RP. Considere a RP da Figura 65. A marcação atual é dada por $\mu =$ <1 0 0 0>. Para saber a próxima configuração da rede após o disparo de uma transição T_i soma-se μ com a linha i da matriz C. Por exemplo, para disparar T_1: <1 0 0 0> + $C[1]$ = <0 1 1 1>. Considerando que a RP não contém *loop*, se o resultado da soma não contiver nenhum valor negativo, significa que a transição estava habilitada.

Uma outra técnica para a análise das RP é com a utilização da árvore de alcançabilidade. A árvore é construída da seguinte forma. A marcação inicial da RP é a raiz da árvore. Os nós representam as marcações atingidas após o disparo

das transições. Os arcos representam o disparo de uma determinada transição. A expansão termina quando não há transições a serem disparadas ou quando um nó for "velho". Na Figura 67 é ilustrada a árvore de alcançabilidade para a RP da Figura 65.

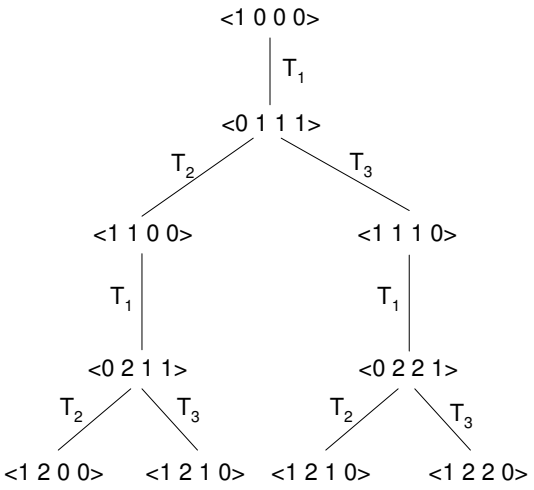

Figura 67: Árvore de Alcançabilidade.

Em relação à árvore de alcançabilidade podem ser feitas algumas observações. Se a transição aparece na árvore, a transição não está morta. Se todas as transições de uma RP aparecem na árvore, então a RP é viva. Se a árvore possui algum nó morto, pode haver situações de *deadlock*. Se a árvore é finita, os lugares da RP são limitados.

Considere a RP H da Figura 68. Na Figura 69 é ilustrada a árvore de alcançabilidade da RP H (Figura 68).

Pode-se observar que a árvore (Figura 69) é infinita, pois o lugar $P4$ é ilimitado. Para tal problema, há duas soluções. A primeira, é expandir a árvore até um certo

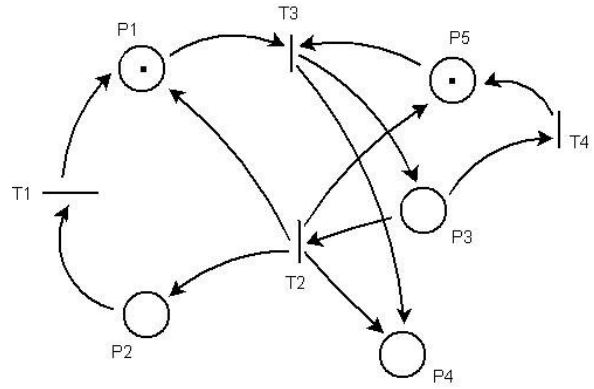

Figura 68: Rede de Petri H.

ponto para que se obtenha uma resposta concisa. No entanto, as respostas serão parciais já que a árvore não foi expandida totalmente.

A segunda solução é transformar a árvore infinita em uma árvore finita. Para isso, cria-se um outro tipo de árvore, conhecida como *árvore de cobertura*. No exemplo da Figura 69, pode-se observar que o nó "1 1 0 2 1" cobre o nó "1 0 0 0 1", pois todos os seus elementos, tomados um a um, são maiores ou iguais. Pode-se observar também que o lugar $P4$ contém o valor 2, sendo que esse valor é maior obtido em relação ao caminho do nó atual até a raiz. Dessa forma, pode-se substituir o valor do nó $P4$ pelo símbolo w, onde $n < w$ e $w + n = w - n = w$ para todo $n \in \mathbb{N}$. A expansão termina quando todos os lugares possuírem o símbolo w. Desse modo, tem-se uma representação finita da RP.

Considere a RP I ilustrada na Figura 70. Na Figura 71 é ilustrada a árvore de cobertura da RP I.

Com a árvore de cobertura não é possível responder questões sobre alcançabilidade e não é possível saber se a RP possui *deadlock*. No entanto, outras questões

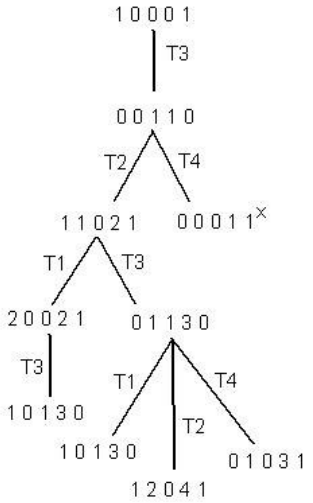

Figura 69: Árvore de Alcançabilidade.

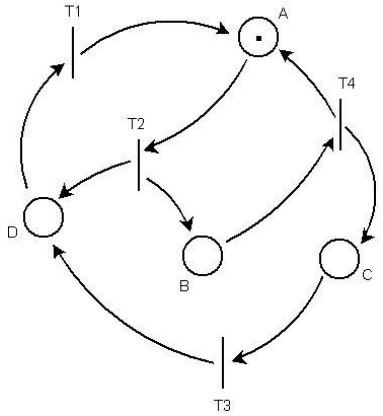

Figura 70: Rede de Petri I.

podem ser respondidas. Se uma transição não aparecer na árvore, a transição é $L0$. Se não aparecer o símbolo w em um lugar, esse lugar é limitado, caso contrário é ilimitado.

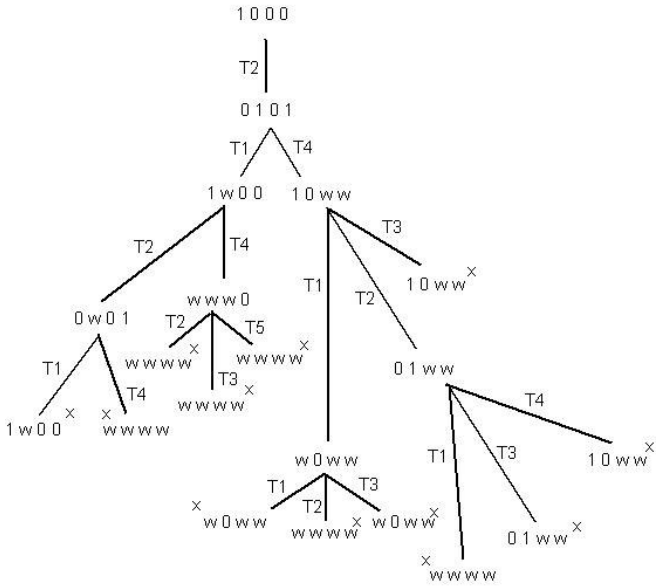

Figura 71: Árvore de Cobertura.