Pontifícia Universidade Católica do Rio Grande do Sul

Faculdade de Informática

Programa de Pós-Graduação em Ciência da Computação

Utilização de Diagramas de Decisão Multi-Valorada para representação do espaço de estados atingível em Redes de Autômatos Estocásticos

Ana Paula Salengue Scolari

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

Porto Alegre

2008

Dados Internacionais de Catalogação na Publicação (CIP)

S422u Scolari, Ana Paula Salengue
Utilização de diagramas de decisão multi-valorada para representação do espaço de estados atingível em redes de autômatos estocásticos / Ana Paula Salengue Scolari. – Porto Alegre, 2007.
81 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS
Orientador: Prof. Dr. Paulo Henrique Lemelle Fernandes

1. Informática. 2. Modelagem de Sistemas. 3. Redes de Petri. 4. Redes de Autômatos Estocásticos. 5. Avaliação de Desempenho (Informática). 6. Algoritmos. I. Título.

CDD 004.2

Ficha Catalográfica elaborada pelo Setor de Tratamento da Informação da BC-PUCRS

PUC

Campus Central
Av. Ipiranga, 6681 - prédio 16 - CEP 90619-900
Porto Alegre - RS - Brasil
Fone: +55 (51) 3320-3544 - Fax: +55 (51) 3320-3548

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Reconhecido pelo Parecer No. 930/98.C.N.E. Homologação Publicada no D.O.U. de 30/12/98.

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Utilização de Diagramas de Decisão Multi-valorada para Representação do Espaço de Estados Atingível em Redes de Autômatos Estocásticos***", apresentada por Ana Paula Salengui Scolari, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 31/03/2006 pela Comissão Examinadora:

Prof. Dr. Paulo Henrique Lemelle Fernandes – Orientador — PPGCC/PUCRS

Prof. Dr. Avelino Francisco Zorzo — PPGCC/PUCRS

Prof. Dr. Fernando Luís Dotti — PPGCC/PUCRS

Profa. Dra. Leila Ribeiro — UFRGS

Homologada em 18/03/08, conforme Ata No. 05/08 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

AV. IPIRANGA, 6681 - PRÉDIO 16/SALA 106
90619-900 - PORTO ALEGRE -RS -BRASIL
TELEFONE: 55 51 33203611 FAX 55 51 3320.3621
http: //www.pucrs.br/inf/pos e-mail: ppgcc@inf.pucrs.br

Resumo

Formalismos de modelagem são linguagens capazes de descrever sistemas de forma não ambígüa, permitindo a sua avaliação quantitativa. Os formalismos conhecidos como estruturados permitem a representação sistemática de modelos grandes e complexos. Entretanto, na prática, a modelagem de sistemas de tal porte pode apresentar o problema de explosão do espaço de estados. Em geral, a modelagem de sistemas através de formalismos estruturados, com espaço de estados discreto, resulta em um grande número de estados inatingíveis. A geração e armazenamento dos estados não atingíveis é um ônus não desejado ao processo de modelagem. Este trabalho apresenta um algoritmo capaz de gerar e armazenar somente o espaço de estados atingível (RSS) para o formalismo de Redes de Autômatos Estocásticos (SAN) utilizando Diagramas de Decisão Multi-Valorada (MDD). A idéia principal é apresentar uma primeira versão desse algoritmo, a fim de comprovar a sua viabilidade para o formalismo SAN.

Abstract

Modeling formalisms are languages capable of describing a system behavior in a non ambiguous way, allowing its quantitative evaluation. The structured formalisms can represent large and complex models in a systematic manner. However, the modeling of systems with such characteristics is still a problem, since the space state explosion is often a known issue. Usually, structured formalisms with discrete state space present a large number of unreachable states. This work presents an algorithm able to generate and storing the reachable space state of a Stochastic Automata Network (SAN) using Multi-valued Decision Diagram (MDD). This new technique aims to mainly verify that the applicability of MDD for SAN is a valuable approach.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

PN	Petri Net	15
SPN	Stochastic Petri Net	16
SAN	Stochastic Automata Network	16
PSS	Product State Space	16
RSS	Reachable Space State	16
MDD	Multi-valued Decision Diagram	16
TPN	Temporized Petri Net	23
DAG	Direct Acyclic Graph	32

1 Introdução

Um formalismo de modelagem é uma linguagem alfanumérica ou gráfica para representação de modelos [15]. Os formalismos de modelagem estocástica têm evoluído desde 1890, quando iniciaram os estudos sobre Cadeias de Markov [25]. Logo em seguida, no final da década de 1950, Jackson iniciou os estudos sobre Redes de Fila de Espera [11, 12], que foi complementado no final dos anos 70 por Little [14], Basket, Chandy, Muntz e Palacios [2], e Reiser e Lavenberg [23].

Modelos são abstrações de um problema do mundo real [5]. Formalismos como Cadeias de Markov (MC) são denominados de formalismos não estruturados, por não apresentarem uma maneira modular de representar um modelo. Conseqüentemente, formalismos não estruturados podem enfrentar dificuldades para descrever modelos grandes e complexos.

Os formalismos conhecidos como *estruturados* surgiram com o intuito de amenizar tais restrições. O formalismo de Redes de Petri (PN) foi introduzido no início de 1960 com a proposta de ser um formalismo capaz de descrever sistemas com espaço de estados discreto, representando realidades complexas através de modelos compactos [9]. O formalismo de Redes de Autômatos Estocásticos (SAN), proposto por Plateau [22] nos meados de 1980, tem basicamente o mesmo objetivo. No entanto, SAN permite a representação de um sistema como uma coleção de subsistemas tanto com comportamento independente, quanto com interdependência ocasional.

Com a existência de formalismos de modelagem capazes de representar sistemas grandes e complexos, emerge o problema da explosão do espaço de estados. Basicamente, a explosão do espaço de estados consiste em elevado consumo de memória para representar os diferentes estados aplicáveis a realidade de um modelo. Endereçar este problema começa a ser foco de estudo para vários formalismos como Redes de Petri e Redes de Autômatos Estocásticos. Soluções computacionais existentes utilizam estruturas de dados de tamanho proporcional ao espaço de estados produto (PSS), a fim de identificar posteriormente o espaço de estados atingível do

modelo (RSS). Essa abordagem assume que os estados não atingíveis do modelo também são armazenados, causando assim desnecessários gastos computacionais e de armazenamento.

Estudos aplicados ao formalismo de modelagem de Redes de Petri Estocásticas (SPN) demonstram que é possível descrever-se sistemas grandes e complexos de uma maneira compacta, levando-se em conta apenas a porção atingível do modelo (RSS). Basicamente, Buchholz e Kemper [7], e Miner e Ciardo [18, 17, 10] propuseram maneiras eficientes de descrever o espaço de estados atingível de modelos grandes e complexos para o formalismo de SPN, que anteriormente eram de difícil representação. Para alguns casos, a estrutura de dados Diagramas de Decisão Multi-Valorada, conhecida como MDD, foi utilizada, devido ao fato de ser compacta e eficaz para as operações comuns como inserções e buscas.

O objetivo desse trabalho é adaptar o conceito de MDD aplicado em SPN [18, 17] para o formalismo de modelagem SAN. A principal contribuição se fará pela descrição de um algoritmo capaz de encontrar e armazenar somente o espaço de estados atingíveis em uma estrutura MDD. Esta adaptação terá cuidado adicional com o uso de taxas constantes e funcionais apresentadas em eventos locais e sincronizantes, particularidade existente no formalismo de SAN. Porém, convém salientar que esta primeira versão do algoritmo preocupa-se com a viabilização de uso de MDD para o formalismo de SAN, ainda não focando algumas questões de performance. De fato, a análise da complexidade do algoritmo proposto é apresentada, a fim de estabelecer-se uma idéia do desempenho que este apresentará, conforme as variáveis que influenciam o sistema são aumentadas.

Obter-se a estrutura de MDD mais compacta possível que represente o modelo é um aspecto desejável. Para tanto, o conceito de ordem dos autômatos na estrutura de MDD é também explorado por esse estudo. Conclusões aqui apresentadas demonstram que a ordem com que os autômatos são dispostos no MDD influencia o número de nodos requeridos pela estrutura, bem como o número necessário de reduções (otimização).

Um comparativo do algoritmo atual para geração e armazenamento utilizado em SAN, Vetor de Booleanos, com a nova técnica que utiliza MDD é também apresentado. O quesito *consumo final de memória* é o alvo principal do comparativo estabelecido entre ambas as técnicas. O

PEPS, é a ferramenta atualmente utilizada para a resolução de sistemas descritos em SAN [3] e candidata a integração do algoritmo aqui proposto. Basicamente, a ferramenta PEPS possui 2 etapas distintas para obter a solução estacionária de um modelo SAN. A primeira etapa consiste em preparar as informações do modelo para que, na segunda etapa, as probabilidades de permanência em cada estado global possam ser calculadas. Dessa forma, a obtenção dos estados atingíveis do modelo, tópico aqui em estudo, é realizada na primeira etapa da solução de um modelo SAN, a qual o consumo de memória durante o processo de geração não é um fator muito relevante. Para o PEPS é importante que ao início da computação da segunda etapa, o consumo final de memória seja otimizado. Então, outras análises como consumo de memória durante o processo de geração (pico de memória) e consumo de CPU não são alvos de estudo desse trabalho.

O decorrer desse trabalho apresenta primeiramente uma breve descrição dos formalismos em estudo, que são SPN e SAN (Capítulo 2). O Capítulo 3 resume o uso de MDD para o formalismo de SPN, baseado em trabalhos realizados por Miner e Ciardo [18, 17]. Capítulo 4 apresenta o algoritmo capaz de gerar e armazenar o espaço de estados atingível utilizando MDD para o formalismo de SAN. A avaliação da complexidade deste algoritmo é também apresentada neste Capítulo. Ainda, um comparativo entre a técnica tradicional, Vetor de Booleanos, e a nova técnica baseada em MDD é apresentado no Capítulo 5, através de exemplos de modelagem. O Capítulo 6 apresenta a influência da ordem dos autômatos relacionada as tarefas de otimização da estrutura de MDD. E, finalmente o Capítulo 7 apresenta a conclusão desse trabalho ressaltando sua a principal colaboração assim como sugestões de futuros trabalhos a serem explorados.

2 Formalismos de Modegalem

Neste capítulo, apresenta-se um embasamento teórico a respeito dos formalismos utilizados como referência nesse estudo, compreendendo conceitos básicos sobre Redes de Petri (PN) e Redes de Autômatos Estocásticos (SAN).

2.1 PN - Redes de Petri

Redes de Petri *(PN - Petri Nets)* é um formalismo matemático baseado em grafos que tem o objetivo de descrever, modelar e analisar o comportamento de sistemas com espaço de estados discreto, sendo principalmente aplicáveis em sistemas concorrentes e assíncronos.

Este formalismo de modelagem tem sido amplamente utilizado em diversas áreas tais como protocolos de comunicação e rede, arquitetura de computadores, sistemas distribuídos, planejamento de manufaturas, verificação e síntese de circuitos digitais e quaisquer outros sistemas onde é possível fazer-se uma avaliação em alto nível do mesmo.

A estrutura de uma PN é composta por um conjunto de *lugares* denominados P, um conjunto de *transições* denominadas T e um conjunto de *arcos direcionados* formados por $(P \times T)$ e $(T \times P)$. Os arcos direcionados conectam os lugares às transições e as transições aos lugares. Além disso, o elemento denominado de *marca*, também chamado de *token*, é responsável por identificar as transições habilitadas a disparar. As marcas disponíveis nos lugares do modelo são denominados de *conjunto marcação*, que é representado pelo símbolo M. A cada disparo de uma transição, um número definido de marcas é consumido de todos os seus lugares de entrada[1] e um número definido de marcas é adicionado a todos os lugares de saída[2].

[1] Lugares predecessores que alimentam a transição t.

[2] Lugares sucessores a transição t.

Formalmente, pode-se denominar uma Rede de Petri como uma quádrupla do tipo:

$$N = (P, T, W, M_0) \tag{2.1}$$

Onde $P = \{p_1, ..., p_n\}$ é um conjunto finito e não vazio de lugares; $T = (t_1, ..., t_n)$ é um conjunto finito e não vazio de transições, sendo $P \bigcap T = \emptyset$; $W : (P \times T) \bigcup (T \times P) \rightarrow \mathbb{N}$ é o conjunto de arcos entre lugares e transições ou entre transições e lugares devidamente ponderados. Caso o peso de um arco seja diferente de zero, pode-se dizer por liberdade de linguagem que este arco existe. M_0 representa a marcação inicial, que é o conjunto de marcas disponíveis nos lugares antes do primeiro disparo de uma transição ocorrer.

Graficamente, os *lugares* são representados por círculos, as *transições* por barras e os *arcos direcionados* por setas. As *marcas* são representadas por *tokens* dentro dos círculos. Seja u um lugar qualquer conectado por uma transição t, se $W(u, t) > 0$, então existe um arco que parte de u e vai para t com peso $W(u, t)$. Seja v um lugar qualquer conectado por uma transição t, se $W(t, v) > 0$, então existe um arco que parte de t e vai para v com peso $W(t, v)$.

Um lugar é uma *entrada* para uma transição se existe um arco direcionado do lugar para a transição em questão. Uma *entrada* pode ser representada pela seguinte notação: $.u = v \in P \bigcup T \mid W(v, u) > 0$. Analogamente, um lugar é uma *saída* de uma transição se existe um arco direcionado da transição para o lugar em questão. Por sua vez, uma *saída* pode ser representada pela seguinte notação: $u. = v \in P \bigcup T \mid W(u, v) > 0$. A Figura 1 tem como lugares de entrada da transição t_1 apenas o lugar p_1 enquanto que os lugares de saída são representados por p_2 e p_3.

Quando um arco possui peso p maior que zero, é necessário o consumo de p marcas para que a transição t dispare. Na verdade, para que t possa disparar é necessário um número de marcas no mínimo igual ao valor do peso do arco, em todos os lugares que são entrada para esta transição.

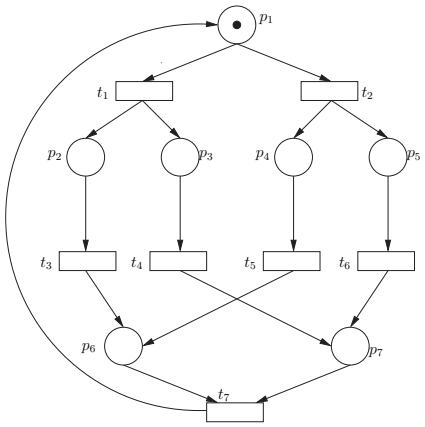


Figura 1 – Representação Gráfica de uma Rede de Petri

2.1.1 Disparando Transições

A execução de uma Rede de Petri acontece através do *disparo* de transições. Para que um disparo ocorra, é necessário que uma transição esteja *habilitada*. Para uma transição ser considerada habilitada é necessário que todos os lugares de entrada contenham um número de marcas no mínimo equivalente ao peso do arco em questão. Uma vez que uma transição t esteja habilitada em uma marcação M, um disparo pode ocorrer e uma nova marcação M' será atingida.

O disparo de uma transição t qualquer consome marcas dos lugares de entradas e gera marcas nos lugares de saída. Formalmente, pode-se representar essa operação como:

$$M' = M - W(p,t) + W(t,p) \tag{2.2}$$

Onde M' é o novo conjunto de marcações após o disparo da transição t ser efetuada.

Uma seqüência de transições quaisquer é dita uma *seqüência de disparos* se a partir de uma marcação M_1 atinge-se uma outra marcação M_k qualquer. Uma marcação M_k é dita *atingível* se existe uma seqüência de disparos a partir da marcação inicial M_0 que leva até a mesma. A *árvore de atingibilidade* de uma PN (Figura 2) representa o espaço de estados atingível como nodos da árvore, bem como todas as possíveis seqüências de disparos em forma de transições entre os nodos.

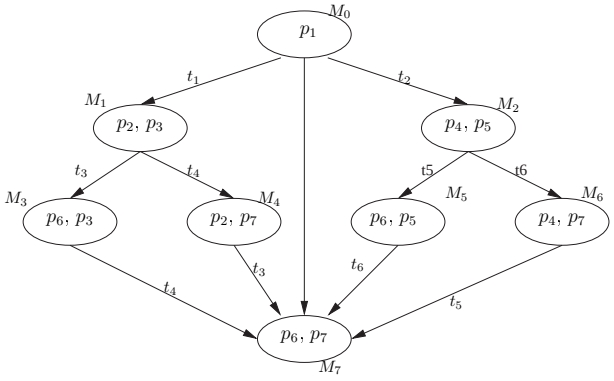


Figura 2 – Árvore de Atingibilidade da Rede de Petri da Figura 1

2.1.2 Limitação (*boundness*) e Segurança (*safeness*)

Uma Rede de Petri pode apresentar as propriedades de *limitação* e *segurança*.

Uma PN é dita *k-limitada*, ou apenas *limitada*, se para todos os estados atingíveis do modelo, todos os lugares da rede tem um limite k de marcas, sendo esse um valor finito.

Uma Rede de Petri é dita *segura* se o limite k desta for igual a 1. Logo, tem-se no máximo uma marca em todos os lugares marcados da rede, em qualquer estado atingível.

O Algoritmo *Symbolic Traversal Petri Net* apresentado em [21], é responsável por gerar a árvore de atingibilidade de uma PN e armazená-la em uma estrutura de BDD. Então, as combinações de caminhos armazenados na estrutura de BDD representam o espaço de estados atingível do modelo PN. Um exemplo prático é exibido pela Figura 2. Entretanto, este algoritmo tem como restrição a premissa de que a PN seja limitada e segura. Na verdade, existem extensões do Algoritmo *Symbolic Traversal Petri Net* para o caso de redes não seguras [21]. No entanto, por não ser objetivo de estudo desse trabalho, não será apresentado com mais detalhes.

2.1.3 Redes de Petri Estocásticas

O formalismo de Redes de Petri (PN) descreve somente a estrutura lógica de sistemas, pois tal modelagem não inclui conceitos de tempo. Todavia, o conceito de tempo possui um papel importante na descrição do comportamento de muitos sistemas e por essa razão faz-se necessário de ser representado.

A inclusão do conceito de tempo no formalismo de Redes de Petri, que foi primeiramente introduzido por Noe e Nutt [20], Merlin e Farber [16] e Zuberek [26], é conhecido como *Redes de Petri Temporizadas* (TPN). Essa primeira abordagem atribui um *tempo fixo* para o disparo das transições existentes, e devido a esse motivo tornou-se ineficiente para representação de todas as realidades. A atribuição de valores variáveis ao disparo das transições tornou-se necessário, pois esta permite a representação de um conjunto maior de realidades a qual o fator tempo variável é importante. Então, iniciou-se os primeiros estudos sobre Redes de Petri Estocásticas (SPN), detalhadamente documentados em [4, 19, 24].

Redes de Petri Estocásticas (SPN) é o formalismo de modelagem de Redes de Petri acrescido da associação de um tempo distribuído exponencialmente para o disparo de cada transição da rede. Nesse novo conceito, a modelagem e avaliação de sistemas envolvendo concorrência, não-determinismo e sincronização faz-se possível.

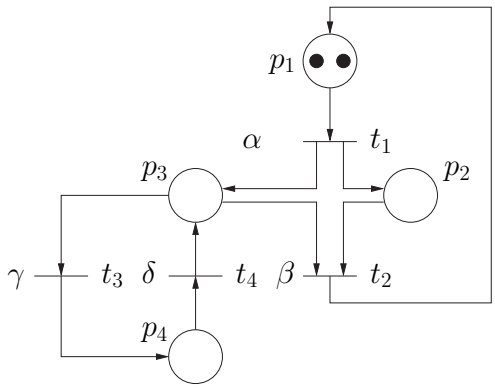


Figura 3 – Exemplo de uma Rede de Petri Estocástica

A estrutura de uma Rede de Petri Estocástica possui os mesmos elementos descritos anteriormente pela Equação 2.1, porém acrescido do conjunto $L = l_1, l_2, ..., l_m$ que representa as taxas de disparos das transições. A Figura 3 apresenta um exemplo de Rede de Petri Estocástica, onde cada p_i representa um lugar, cada t_i uma transição e finalmente α, β, γ e δ as taxas de disparo para as transições t_1, t_2, t_3 e t_4, respectivamente. Cabe salientar que para cada transição t do modelo, existe uma taxa de disparo que determina o intervalo de *tempo médio* entre o disparo das transições.

2.2 SAN - Redes de Autômatos Estocásticos

O formalismo de Redes de Autômatos Estocásticos (SAN) foi proposto por Plateau [22] em 1985, e sua idéia básica é representar um sistema como uma coleção de subsistemas com comportamento independente e interdependência ocasional. Cada subsistema é definido por um autômato estocástico e por transições entre os estados deste autômato, que são disparadas através de eventos a estas associadas.

Um autômato é um conjunto finito de estados juntamente com um conjunto finito de transições entre estes estados [13, 1]. A denominação estocásticos atribuída aos autômatos neste formalismo deve-se ao fato de que o tempo é tratado como uma variável aleatória, que na escala de tempo contínua obedece uma distribuição exponencial[1].

As próximas subseções deste Capítulo apresentarão conceitos básicos referentes ao formalismo SAN, a fim de demonstrar como esta abordagem é capaz de descrever sistemas atualmente ditos complexos.

2.2.1 Estados Locais e Globais

O formalismo SAN utiliza-se das noções de estado e de transições entre os estados para representação dos eventos. Existem transições que estão relacionadas a apenas um único autômato do modelo. Porém, transições que estão relacionados ao mesmo tempo a vários autômatos do modelo também podem existir, ocorrendo de acordo com taxas específicas. A próxima subseção explicará em detalhes os tipos de transições existentes em SAN.

O estado individual de cada autômato é chamado de *estado local*. Já o *estado global* de uma SAN é definido como a combinação de todos os estados locais de cada autômato componente da SAN.

2.2.2 Tipos de Transições

As ligações existentes entre os estados de um autômato são denominadas de *transições*. Associado as transições, existem eventos que determinam quando estas podem ocorrer. As transições e seus respectivos eventos podem ser classificadas em diferentes tipos, conforme explicado a seguir.

Dois tipos de transições podem modificar o estado global de um modelo SAN: *transições locais* e *transições sincronizantes*. Transições locais modificam o estado de apenas um autômato do modelo, e a estas estão associados *eventos locais*. Transições sincronizantes modificam simultaneamente o estado de mais de um autômato do modelo, e a estas estão associados *eventos sincronizantes*.

Ambas transições locais e sincronizantes podem também ser chamadas de *transições funcionais*. Isto ocorre por que associado ao disparo da transição, podem existir taxas não constantes que determinam o intervalo de disparo associado a esta.

Então, qualquer evento do modelo pode ter associado uma taxa constante *(número real positivo)* ou uma taxa funcional avaliada por uma função. Essa função depende do estado dos demais

autômatos e resulta em um número real positivo que determina se um determinado evento está ou não apto a disparar. Diferentemente de transições associadas a taxas constantes, taxas funcionais impõem mudanças de estado unidirecionais nos autômatos do modelo. Ou seja, taxas funcionais modificam apenas o estado do autômato a qual esta pertence e não os estados dos autômatos aos quais ela depende. O algoritmo proposto no Capítulo 4, levará em conta essa importante premissa a fim de determinar o conjunto de estados condição e conseqüência para gerar o RSS de um modelo SAN.

2.2.3 Estrutura de uma SAN

A estrutura de uma SAN compreende um conjunto de *autômatos* A; um conjunto de *eventos* E; e uma *função de atingibilidade* F. Formalmente, uma SAN é representada por uma tupla do tipo:

$$SAN = (A,\ E,\ F) \tag{2.3}$$

Onde:

- A é um conjunto de autômatos que compreende N autômatos nomeados $A^{(i)}$, onde $i \in [1..N]$ [3];
- E é um conjunto de eventos composto por E eventos nomeados e_j, onde $j \in [1..E]$ [4];
- F é uma função com $dom(\mathcal{S})$ e $codom([0..1])$, onde $dom(\mathcal{S})$ representa todos as possibilidades de estados globais do modelo, e $codom([0..1])$ o valor associado a função de 1 ou 0.

O espaço de estados produto (PSS) em SAN é definido como $\prod_{i=1}^{N} S^{(i)}$, onde S é o conjunto de

[3] No decorrer deste trabalho, é adotada a notação $[i..j]$ referindo-se a um número no intervalo de i até j, inclusive, pertencendo ao conjunto dos números naturais; e a notação $[i,j]$ referindo-se a um número pertencente ao intervalo i e j, inclusive, no conjunto dos números reais.

[4] No contexto deste trabalho, usa-se a letra e para se identificar um evento, porém qualquer outra letra ou palavra poderia ser usada sem perda de generalidade.

estados (locais) do autômato $A^{(i)}$ e $\mid S^{(i)} \mid$ é o número de estados (cardinalidade) do conjunto $S^{(i)}$.

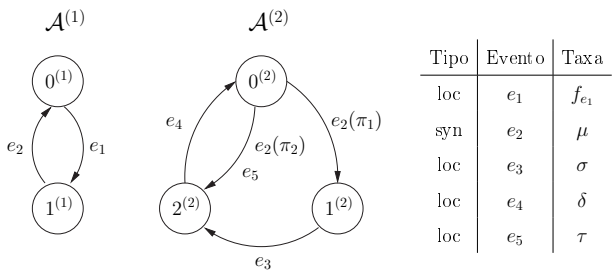
A definição de quais estados podem ser atingíveis ou alcançados em um modelo SAN é dada pela *função de atingibilidade*. A função de atingibilidade F associa aos estados globais de PSS o valor 1 se eles são *atingíveis*. Caso contrário, esta associa o valor 0, marcando-os como *inatingíveis*. Basicamente, a função de atingibilidade tem o intuito de eliminar os estados globais que não são atingíveis ao modelo, por não corresponderem com a realidade expressada por este. Por exemplo, um modelo SAN que representa um sistema de compartilhamento de recursos, onde tem-se N clientes disputando R recursos. É fácil de imaginar-se que se o número de recursos for menor que o número de clientes, o estado global que representa todos os clientes utilizando um recurso não poderá ser atingível.

RSS é um subconjunto de PSS que compreende todos os estados $\tilde{x}$, tais que $F(\tilde{x}) = 1$, denominados de conjunto de estados atingível.

2.2.4 Exemplo de Modelo SAN

A Figura 4 é um exemplo de modelo SAN com dois autômatos, um evento sincronizante (e_2) e quatro eventos locais (e_1, e_3, e_4, e_5). Neste exemplo, a taxa do evento e_1 não é constante e sim funcionalmente definida pela taxa f_{e1} descrita pela notação utilizada pela ferramenta PEPS [3]. A interpretação da função f define o disparo da transição do estado $0^{(1)}$ para o estado $1^{(1)}$ com taxa λ se o autômato $A^{(2)}$ está no estado $0^{(2)}$, ou taxa γ se o autômato $A^{(2)}$ está no estado $2^{(2)}$. Se o autômato $A^{(2)}$ está no estado $1^{(2)}$, a transição do estado $0^{(1)}$ para $1^{(1)}$ não ocorre (taxa igual a zero). É importante observar que o uso de funções disponibiliza uma maneira compacta e flexível de descrever comportamentos alternativos através do uso de um simples evento (local ou sincronizante).

A Figura 5 mostra a Cadeia de Markov (MC) equivalente para o modelo SAN apresentado pela Figura 4. Assumindo que o estado $0^{(1)}0^{(2)}$ é o estado inicial, somente 5 dos 6 estados desse modelo MC são atingíveis. Para representar tal característica, é necessário representar os estados



Tipo	Evento	Taxa
loc	e_1	f_{e_1}
syn	e_2	μ
loc	e_3	σ
loc	e_4	δ
loc	e_5	τ

Figura 4 – Exemplo de Modelo em SAN

globais atingíveis através do uso de uma função de atingibilidade. Todavia, em alguns casos, a obtenção de uma função que represente os estados atingíveis do modelo não é uma tarefa trivial. Na verdade, a tarefa de definição de uma função de atingibilidade para modelos criados em SAN talvez seja o aspecto menos encorajador para escolha do uso desse formalismo. Caso a função de atingibilidade não seja definida, ainda assim é possível encontrar-se todos os estados atingíveis do modelo, porém todas as possibilidades precisam ser computadas, trazendo assim um ônus não desejado ao sistema.

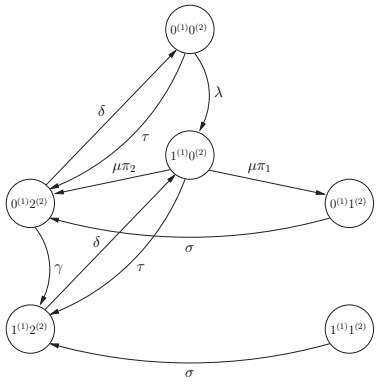


Figura 5 – Cadeia de Markov Equivalente ao Modelo SAN da Figura 4

Considerando o modelo exibido pela Figura 5, a função de atingibilidade deve obrigatoriamente excluir o estado global $1^{(1)}1^{(2)}$, e pode ser representada por:

$$F = !\left[\left(st\ \mathcal{A}^{(1)} == 1^{(1)}\right)\ \&\&\ \left(st\ \mathcal{A}^{(2)} == 1^{(2)}\right)\right]$$

Em SAN, o uso de expressões funcionais não está estritamente limitado a taxas de eventos. Probabilidades de roteamento do modelo podem também ser expressadas por funções. De fato, a possibilidade de uso de funções em SAN é considerado uma poderosa característica desse formalismo, uma vez que permite a descrição de sistemas complexos de uma forma compacta. Além disso, o custo computacional para gerenciamento de taxas funcionais tem significativamente decrescido com o desenvolvimento de soluções numéricas para modelos SAN (algoritmos para produtos tensoriais generalizados) [3].

Este Capítulo apresentou um embasamento teórico a respeito dos formalismos estudados neste trabalho. O formalismo de modelagem de Redes de Petri é o referencial de estudo para técnica de otimização do armazenamento do espaço de estados neste aplicada [18, 17]. O formalismo de Redes de Autômatos Estocásticos é o alvo principal deste trabalho, uma vez que é foco de estudo para proposição de técnica similar para otimização do armazenamento do espaço de estados atingível. Os próximos Capítulos irão apresentar em mais detalhes a técnica de otimização de espaço de estados existente para o formalismo de PN e a proposta de técnica similar para o formalismo de SAN.

3 Diagramas de Decisão Multi-valorada para SPN

Este capítulo apresenta a técnica utilizada como referência nesse trabalho, responsável por gerar o RSS para o formalismo de SPN e armazená-lo em uma estrutura de MDD. Este estudo está baseado nos trabalhos propostos por Miner e Ciardo [18, 17].

3.1 MDD para SPN

Dado o problema da explosão do espaço de estados existente nos formalismos de modelagem em geral, o estudo de técnicas alternativas, que não estejam associadas a necessidade de armazenamento de estruturas de dados de tamanho proporcional ao espaço de estados dos modelos, começam a ser foco de estudo.

Em SPN, a utilização de Diagramas de Decisão Binária (*BDD*) [6] surge como uma estrutura de dados alternativa, capaz de armazenar apenas a porção atingível de um modelo [21]. Basicamente, um BDD consiste em uma estrutura de dados que armazena apenas valores binários, tais como *(0, 1)*, em seus nodos. A relação entre os nodos da estrutura é dada por ponteiros de encadeamento, que determinam os nodos vizinhos mais ao lado e abaixo respectivamente. Devido a impossibilidade de armazenar-se valores diferentes de 0 e 1 em seus nodos, a estrutura de BDD, quando aplicada ao armazenamento de espaço de estados atingível em SPN, limita-se a representar somente Redes de Petri Limitadas e Seguras [21]. A estrutura de BDD é também chamada de grafo acíclico direto (DAG).
A Figura 6 representa o BDD equivalente a função f expressa pela Tabela 1. Os arcos que partem do nodo (ou estado) raiz x_1 para os nodos filhos mais a esquerda, representam o caminho a ser seguido quando o valor avaliado da função f para as variáveis x_1, x_2 ou x_3 é 0 (falso). Enquanto que os arcos que partem da raiz x_1 para os nodos filhos mais a direita, representam a avaliação da função para o valor 1 (verdadeiro). Partindo-se do nodo raiz, e seguindo-se os arcos

até chegar-se aos nodos filhos existentes (de acordo com o valor atribuído para a variável x_n em cada nível) é possível chegar-se ao valor 0 ou 1 avaliado pela função f, representado pelos nodos terminais 0 e 1 da estrutura de BDD .

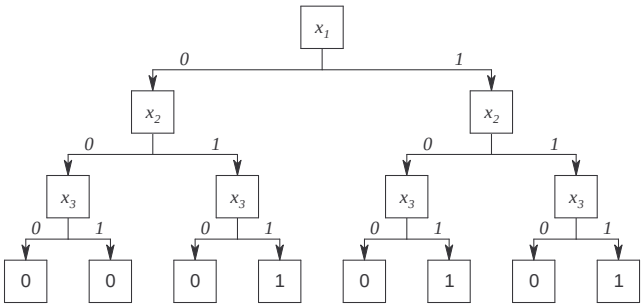


Figura 6 – Exemplo de Diagrama de Decisão Binária para Função f

Tabela 1 – Função f representada pelo Diagrama de Decisão Binária

f	0	0	0	1	0	1	0	1
x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1

O estudo de BDD aplicado ao formalismo de Redes de Petri Estocástica teve o objetivo de aliviar o problema da explosão do espaço de estados para sistemas grandes e complexos, pois este almeja o armazenamento de apenas o RSS do modelo.

De fato, trabalhos posteriores apresentados por Miner e Ciardo [18, 17], e inspirados nesse primeiro experimento realizado com a estrutura de BDD, propõem o uso de uma estrutura de dados mais poderosa, denominada de Diagramas de Decisão Multi-Valorada (*MDD*).

MDD é uma extensão da estrutura de BDD para lógica booleana não binária. Um MDD pode armazenar em seus nodos (ou estados), valores inteiros quaisquer, tais como *(0, 1, 2, ..., n)*. Analogamente a estrutura de BDD, os nodos da estrutura representam as variáveis do sistema enquanto que os arcos (ou linhas) representam os possíveis valores que estas podem assumir. A

grande diferença dá-se ao fato de o MDD não estar limitado a armazenar apenas valores binários (0 ou 1). A Figura 7 exemplifica uma estrutura MDD que representa a função mínimo entre três valores inteiros *min(a, b, c)*.

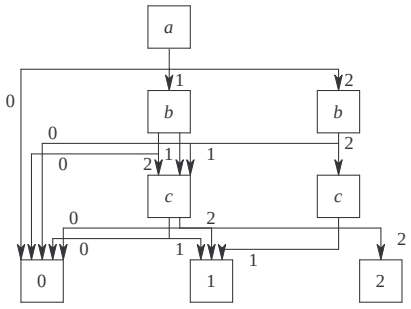


Figura 7 – MDD para função$min(a, b, c)$

Partindo-se do nodo raiz e seguindo os caminhos existentes representados pelos arcos até os nodos terminais é possível determinar-se qual o valor mínimo entre a, b e c. Por exemplo, quando a assume o valor 0, o arco com valor 0 liga a raiz a direto ao nodo terminal 0 (arco mais a esquerda), uma vez que não é necessário testar os demais valores de b e c. Outro caminhamento existente assume que a tem o valor 2 (arco mais a direita), e b também possui o valor 2 (arco mais a direita de b), portanto ainda é preciso avaliar o valor de c. Assumindo que o valor de c seja 0 (arco mais a esquerda) ou 1 (arco do meio), os valores mínimos são respectivamente 0 ou 1. Caso contrário, se o valor de c for 2, o mínimo entre a, b e c é 2.

Portanto, a característica diferencial apresentada pela estrutura de MDD, permite que valores inteiros quaisquer sejam armazenados nos nodos da estrutura. Este torna-a mais poderosa no âmbito de representação de espaço de estados para o formalismo de modelagem SPN. A Seção 3.2 irá apresentar exemplos de modelagem SPN utilizando a estrutura de MDD, a fim de ilustrar a técnica aqui referenciada.

Conceitos de otimização são sempre importantes de serem mencionados e avaliados, uma vez que refinam e melhoram técnicas em geral. Otimizações de ordenação e redução, previamente abordadas para BDDs, são igualmente importantes para que uma estrutura de MDD atinja sua

forma mais compacta possível. *Esta, é usualmente denominada de forma canônica.* É sabido que ambas estruturas de BDD e MDD só atingem a sua forma canônica, se estiverem propriamente ordenadas e reduzidas [6, 21]. O processo de compactação ocorre através da aplicação das regras de redução mencionadas em [21] sobre um modelo MDD já ordenado de acordo com algum critério estabelecido. De fato, não existe um método conhecido para definir qual o melhor critério de ordem a ser aplicado. Estudos realizados até então demonstram que a obtenção da melhor ordem (que torna a estrutura mais compacta ou com menor número de estados) é somente atingida através do teste de todas as possibilidades existentes [6]. O Capítulo 6 irá exercitar essa idéia em busca de uma heurística que determine o critério de ordem mais apropriado para o formalismo de SAN. A estrutura de MDD ordenada e reduzida (estados redundantes são eliminados sem alterar a representação do espaço de estados) é denominada de Diagramas de Decisão Multi-Valorada Ordenado e Reduzido (*ROMDD*). Todavia, no decorrer desse estudo a mesma será referenciada apenas como MDD, pois todas as estruturas aqui apresentadas estarão na sua forma canônica.

Em SPN, o critério de ordem é estabelecido sob um modelo previamente particionado de acordo com algum critério. Em geral, o critério de partição é considerado uma tarefa independente, que deve ser resolvida *a priori*. Segundo Miner e Ciardo, o estudo do critério de partição do modelo requer esforço adicional a definição do algoritmo de geração e armazenamento e deve ser tratado em escopo específico, por apresentar um tema vasto para estudo e experimento. De fato, o número de partições definidas para o modelo SPN representará o número de níveis da estrutura de MDD, influenciando diretamente na compactação da estrutura (e não na corretude do algoritmo). Ainda, a ordem em que os sub-modelos ou subsistemas são dispostos nos níveis da estrutura irá influenciar no número de estados final da estrutura de MDD. A Seção 3.2 irá apresentar um exemplo de particionamento e ordenamento para um modelo SPN.

O Algoritmo 1, proposto em [18], demonstra como obter o MDD através da simulação do disparo das transições de uma SPN. Este assume que o modelo SPN já foi particionado e ordenado de acordo com algum critério pré-estabelecido, não enfatizando dessa forma essas etapas.

O primeiro passo proposto pelo Algoritmo 1 (linha 3) é o disparo de transições locais, uma vez que estas afetam apenas o subsistema local a qual pertencem. As transições sincronizantes são então disparadas (linha 6) logo após, afetando simultaneamente mais de um subsistema.

Algorithm 1 MDDExplore(M_0)

1: $S \leftarrow M_0$; // inicializa S (marcação inicial)
2: **repeat**
3: $S \leftarrow DoLocals(S)$; //dispara trans. locais
4: $O \leftarrow S$; // guarda RSS antigo
5: $\varepsilon \leftarrow Intersect(S, \varepsilon(t))$; //$\varepsilon$ é o conjunto de marcas que habilitam a transição t
6: $F \leftarrow Fire(x_k, \varepsilon, t)$; //$F$ é o conjunto de marcas atingidas após a transição t disparar
7: $S \leftarrow Union(S, F)$; // adiciona F a S
8: **if** $(O == S)$ **then**
9: return S;
10: **end if**
11: **until** forever

Basicamente, o algoritmo permanece em um *loop* simulando o disparo de transições locais e sincronizantes até que o espaço de estados atingível (RSS) seja gerado e armazenado na estrutura de MDD.

3.2 Exemplo de Modelagem

Esta seção apresenta um exemplo de sistema utilizando modelagem em Redes de Petri Estocástica com a representação do espaço de estados atingível utilizando a estrutura de MDD.

A Figura 8 ilustra um sistema de compartilhamento de recursos. Nesse modelo, existem N processos compartilhando R recursos. Cada subsistema possui dois lugares: S_i que representa o estado *Sleeping* (ocioso) e U_i representando o estado *Using* (em uso). Marcas existentes no lugar RS representam o número de recursos disponíveis, enquanto que marcas disponíveis em RU representam o número de recursos em utilização. As transições ta_i e tr_i são sincronizantes entre os *i-ésimos* processos e os recursos.

A Figura 8 é genérica para N processos e R recursos, mas tomemos como exemplo os valores de $N = 4$ (4 processos) e $R = 2$ (2 recursos compartilhados) para a construção do espaço de

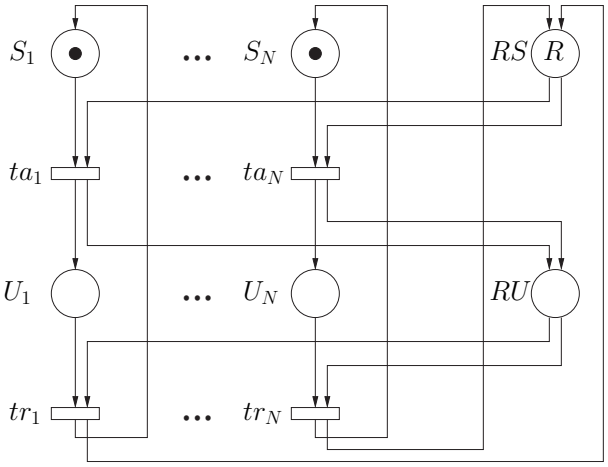


Figura 8 – Compartilhamento de Recursos - Modelo Redes de Petri

estados atingível. Primeiramente, a SPN precisa ser particionada em subsistemas para aplicação da técnica de MDD. Nesse caso, o critério de partição assume que cada processo representa um subsistema (lugares S e U) enquanto que os lugares que controlam os recursos representam outro subsistema (lugares RS e RU). Então, para valores de $N = 4$ E $R = 2$ tem-se um total de 5 subsistemas. Conseqüentemente, o número de níveis da estrutura de MDD terá um total de 5 *(K=5)*.

A fim de verificar se a ordem em que os subsistemas são representados na estrutura de dados influencia ou não o número de estados necessários, duas abordagens foram estudadas. A primeira abordagem assume que o subsistema que controla os recursos disponíveis é a raiz da estrutura *(Nível 5)*, seguido respectivamente pelos Processos 4 (*Nível 4)* , 3 (*Nível 3)*, 2 (*Nível 2)* e 1 (*Nível 1)* . A segunda abordagem assume que a raiz é iniciada pelo Processo 1 (*Nível 5)*, seguido respectivamente pelos Processos 2 (*Nível 4)*, 3 (*Nível 3)* e 4 (*Nível 2)* e finalmente o subsistema que controla os recursos disponíveis (*Nível 1)*.

A Tabela 2 apresenta a codificação utilizada na criação da estrutura de MDD apresentada pelas Figuras 9 e 10.

As cinco primeiro colunas da Tabela 2 representam cada um dos níveis da estrutura de MDD, e a última coluna apresenta a codificação utilizada em cada um dos níveis (0, 1 ou 2). O Processo 1 (coluna 1) representa os lugares S_1 e U_1 e sua codificação expressa basicamente dois estados

Tabela 2 – Codificação do Modelo Compartilhamento de Recursos para Construção do MDD

Processo 1	Processo 2	Processo 3	Processo 4	Recurso	Codificação
S_1, U_1	S_2, U_2	S_3, U_3	S_4, U_4	RS,RU	n/a
1,0	1,0	1,0	1,0	2,0	0
0,1	0,1	0,1	0,1	1,1	1
n/a	n/a	n/a	n/a	0,2	2

distintos. Quando Processo 1 está ocioso, o valor de codificação é representado por 0, ou seja S_1=1 e U_1=0. Porém, quando o Processo 1 está em uso, o valor da codificação é 1, ou seja S_1=0 e U_1=1. Essa mesma codificação é utilizada para todos os demais Processos 2, 3, e 4. A codificação 2 não é aplicável aos Processos, e é representada por n/a, uma vez que necessita-se apenas de dois valores distintos para representar as possibilidades existentes. A coluna Recurso necessita de três codificações distintas, expressas pelos valores 0, 1 ou 2, que significam respectivamente dois lugares disponíveis e zero em uso (2,0), um lugar disponível e um em uso (1,1) e zero lugares disponíveis e dois em uso (0,2).

Para ambos os casos, duas figuras de MDD representando o espaço de estados atingível são exibidas. As Figuras 9 (a) e 10 (a) apresentam o MDD em fase inicial, sem a aplicação das regras de otimização, para respectivamente a primeira (processo 1 é a raiz) e segunda disposição (subsistema de controle de recursos é a raiz). Já as Figuras 9 (b) e 10 (b) exibem o MDD em sua forma canônica para ambas.

Considerando que tanto as operações necessárias para realizar as reduções (otimização), como que a utilização de um maior número de nodos para representar-se o mesmo espaço de estados, são ônus não desejados ao sistema, avalia-se os distintos MDDs construídos. Percebe-se que a primeira disposição, onde o Processo 1 representa o nodo raiz, obteve um melhor resultado, uma vez que o número de reduções necessárias foi de apenas 16 *(35 estados iniciais - 19 estados finais)* comparado com o número de reduções necessárias para a segunda disposição que foi de 17 *(38 estados iniciais - 21 estados finais)*. Além disso, o número final de nodos da primeira disposição (referente a utilização de o menor número de nodos possível para representar-se um mesmo espaço de estados), comparado ao número final de estados apresentado pela segunda disposição é menor. (*19 estados* para a primeira disposição contra *21 estados* da segunda disposição). Então,

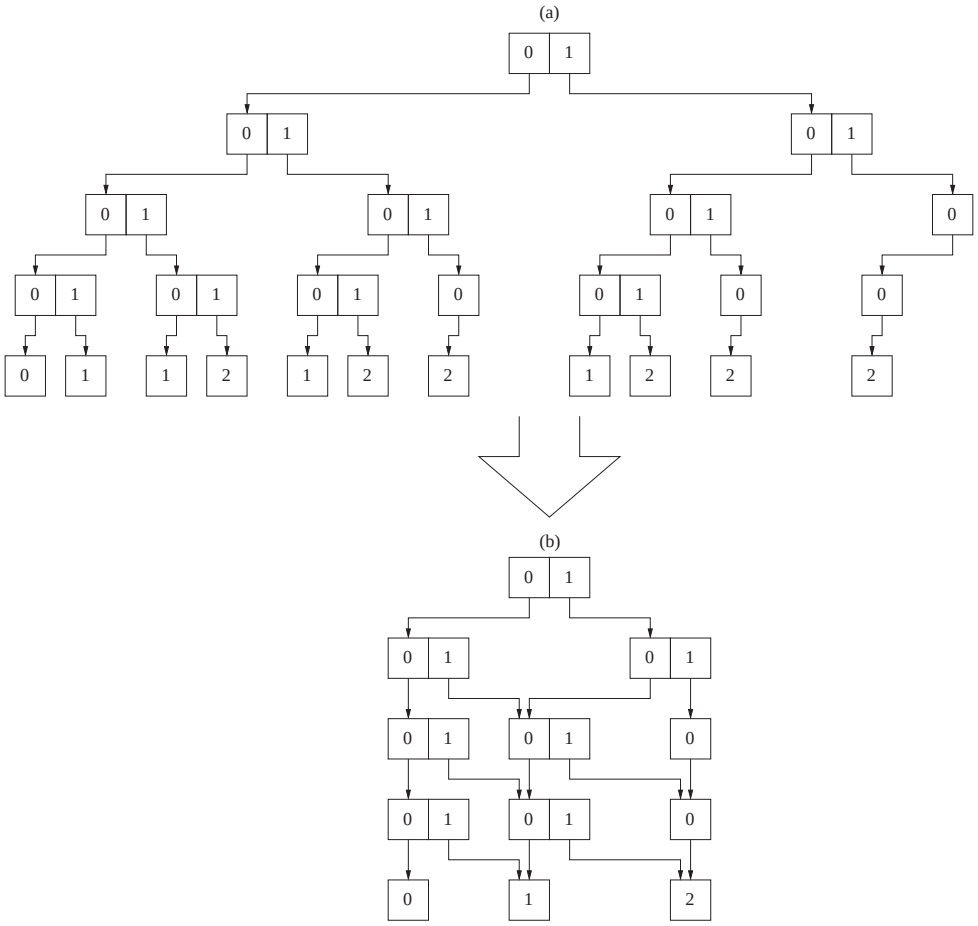


Figura 9 – Compartilhamento de Recursos - MDD - Primeira Disposição

dentre as duas abordagens testadas, a estrutura de MDD apresentada em 9 (b) expressa a forma mais otimizada de representação do espaço de estados atingível para o modelo SPN da Figura 8, segundo codificação estabelecida pela Tabela 2.

Finalmente, a fim de determinar-se o espaço de estados atingível expresso pelas estruturas de MDDs das Figuras 9 e 10, basta partir-se do nodo raiz destas e seguir os caminhos expressos pelos arcos existentes até atingir-se os nodos terminais. A utilização da forma reduzida da estrutura, ilustrada pelo desenho (b) de cada uma das Figuras MDDs mencionadas, é mais rápida e eficaz de ser utilizada, uma vez que esta não possui caminhos redundantes.

Este Capítulo referenciou conceitos importantes da técnica de armazenamento do espaço de estados que visa apenas a porção atingível do modelo para o formalismo de modelagem SPN.

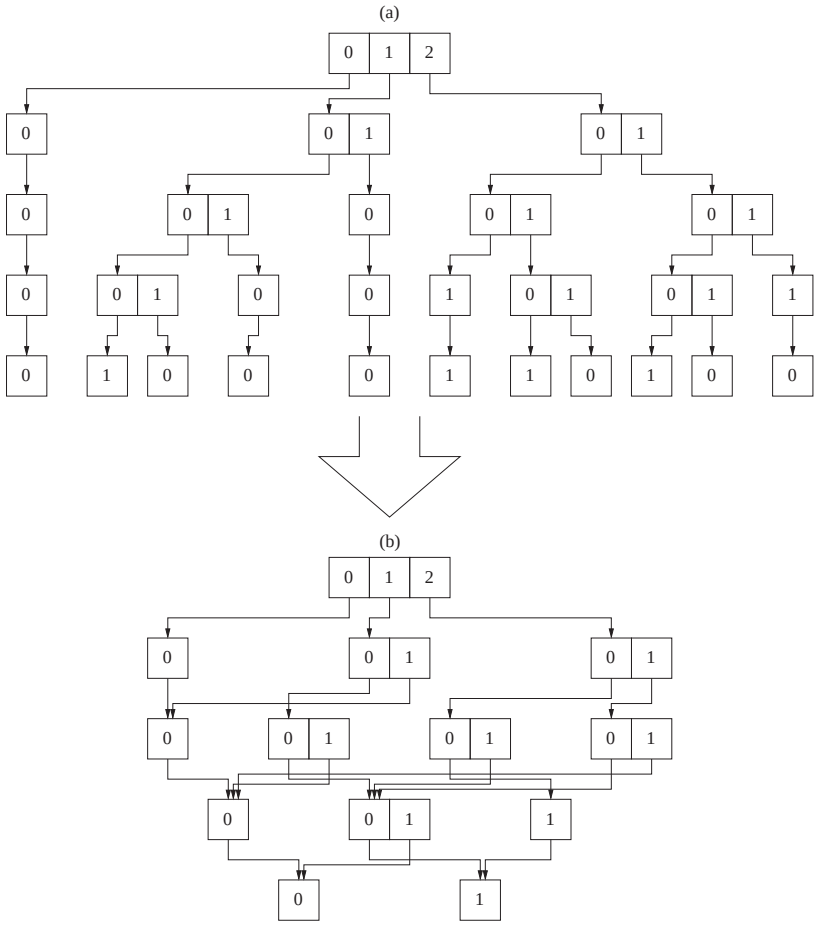


Figura 10 – Compartilhamento de Recursos - MDD - Segunda Disposição

Ainda, um pequeno histórico da evolução da técnica foi abordado, começando com a utilização de BDD e posteriormente partindo-se para uma estrutura de dados mais poderosa denominada de MDD. Os próximos Capítulos irão utilizar o embasamento aqui adquirido a fim de aplicar-se uma técnica análoga ao formalismo de SAN.

4 Aplicando Diagramas de Decisão Multi-Valorada para SAN

Apresenta-se nesse capítulo o algoritmo proposto para geração e armazenamento do espaço de estados atingível para o formalismo de modelagem SAN utilizando-se MDD. Primeiramente, uma breve referência sobre a atual técnica utilizada em SAN é apresentada, seguida da nova técnica baseada em MDD.

4.1 Atual Modelo de Armazenamento para SAN

A atual técnica de geração e armazenamento do espaço de estados atingível para o formalismo de SAN utiliza uma abordagem baseada em um Vetor de Booleanos. Cada possível estado do modelo requer uma posição que o represente nesse vetor. Se um estado é determinado como atingível, a posição correspondente do vetor é assinalada como verdadeiro. Caso contrário, a posição correspondente no vetor é assinalada como falso. Atualmente, esta técnica está implementada na ferramenta PEPS [3], e mesmo requerendo apenas um bit para representar cada estado, resulta em gastos enormes de armazenamento de memória para modelos grandes e complexos.

A fórmula para cálculo do consumo de memória para um modelo SAN que utiliza a técnica Vetor de Booleanos é obtida através da divisão do espaço de estados produto por 8, uma vez que cada estado é representado por um bit apenas. Esta é melhor detalhada no Capítulo seguinte 5, onde um comparativo entre ambas as técnicas Vetor de Booleanos e MDD é estabelecido.

Ainda, definir a função de atingibilidade associada a cada modelo SAN utilizando o método Vetor de Booleanos é, na maioria dos casos, um processo complexo, porém mandatório para tal abordagem. A função de atingibilidade é responsável por guiar o algoritmo de geração durante a

definição dos estados atingíveis do modelo, restringindo os estados não pertencentes a realidade por este representada.

4.2 MDD para SAN

A idéia de aplicar a estrutura de MDD em SAN surge como uma alternativa interessante para aliviar os problemas de explosão do espaço de estados encontrados na atual abordagem. Analogamente ao que é hoje aplicado em SPN, chega-se a um primeira versão da técnica que utilizará MDD para armazenar o espaço de estados atingível em SAN.

As etapas de partição do modelo em subsistemas e ordenamento na estrutura de MDD é um pré-requisito para a técnica a ser aplicada, assim como fundamentado pela técnica estudada em SPN. A característica de modularidade apresentada pelo formalismo SAN torna essa divisão mais fácil. Um critério simples de partição assume que cada autômato do modelo é um nível da estrutura de MDD. Alternativamente, o modelo poderia ser particionado de uma forma distinta, assumindo mais de um autômato por nível por exemplo. Todavia, esta alternativa não foi aqui estudada, mas de qualquer forma pode vir a ser explorada no futuro a fim de verificar-se se esta traria algum benefício adicional a técnica. De fato, os critérios de partição e ordem são variantes que podem influenciar na otimização da estrutura de MDD, mas que não implicam na corretude da técnica aqui proposta.

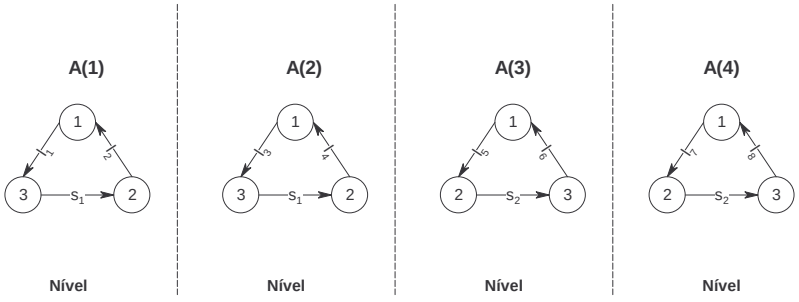


Figura 11 – Modelo SAN com 4 Autômatos

Então, selecionado o critério de partição um autômato para cada nível da estrutura de MDD, a ordem em que os autômatos serão dispostos nos níveis da estrutura MDD é guiada pelo identificador de cada autômato do modelo. Por exemplo, considerando um modelo SAN composto por 4 autômatos respectivamente rotulados pelos identificadores 1, 2, 3 e 4 (Figura 11), a estrutura MDD que representa esse modelo possuirá 4 níveis distintos.

Uma possível ordem a ser aplicada (Figura 12), seria partindo-se do nodo raiz até os nodos terminais da estrutura, dispor os autômatos identificados pelos rótulos 1, 2, 3 e 4. Outra possível ordem seria partir-se do nodo raiz dispondo-se os autômatos 4, 3, 2 e 1 conforme ilustrado pela Figura 13.

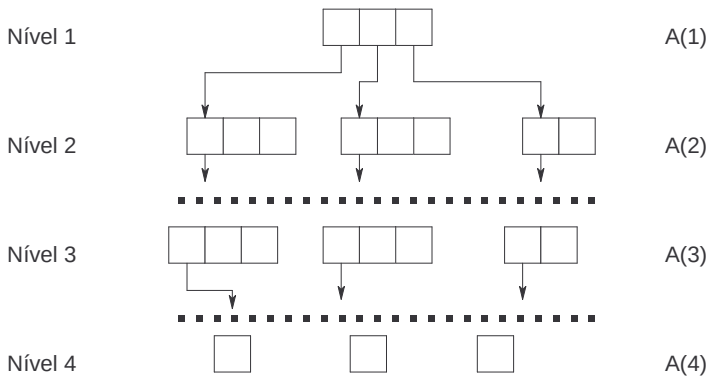


Figura 12 – Exemplo Ordem 1 MDD para Modelo SAN 4 Autômatos

As ordens aqui estabelecidas são apenas dois possíveis exemplos, dentre muitas outras combinações existentes. De fato, a ordem em que os autômatos são dispostos na estrutura de MDD é um fator muito importante de ser determinado *a priori*, uma vez que tal característica influencia diretamente na otimização da estrutura de MDD. Em razão disso, a Seção 6.1 apresenta diversos experimentos, onde a ordem dos autômatos na estrutura de MDD é variada, a fim de que estabeleça-se uma heurística que sugira um bom critério de ordem. Conclusões iniciais referenciadas no Capítulo 6 apontam as condições em que a estrutura de MDD torna-se mais otimizada (e igualmente eficaz).

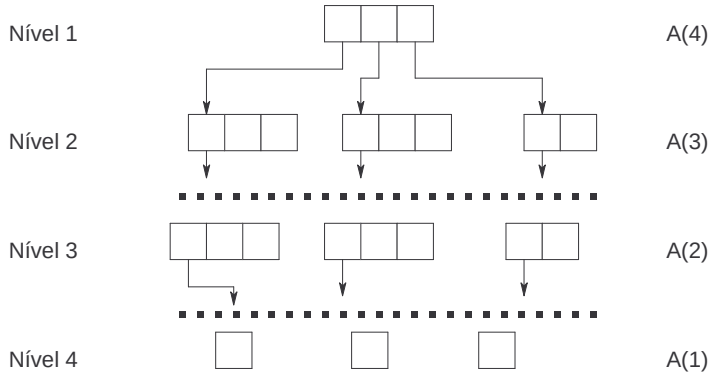


Figura 13 – Exemplo Ordem 2 MDD para Modelo SAN 4 Autômatos

Assim como fundamentado em SPN, o algoritmo de geração de espaço de estados atingível utilizando MDD em SAN, Algoritmo 2, baseia-se na simulação de disparo dos eventos locais e sincronizantes do modelo. O disparo de eventos locais afeta apenas o subsistema a qual ele pertence, enquanto que o disparo de eventos sincronizantes afeta simultaneamente dois ou mais subsistemas do modelo.

O primeiro passo do Algoritmo 2 define dois conjuntos de estruturas de MDDs, que respectivamente representam os estados condições (${C_d}^{(e)}$) e conseqüências (${C_q}^{(e)}$). Os estados condições e conseqüências são gerados para cada evento do modelo, independentemente destes serem eventos sincronizantes ou locais. *Os estados condições $d \in {C_d}^{(e)}$ representam os estados que habilitam o evento e a ser disparado. Os estados conseqüências $q \in {C_q}^{(e)}$ representam os estados atingidos depois que o evento e é disparado.*

A tarefa de geração dos estados condições e conseqüências está explicada em detalhes pelo Algoritmo 3. Todavia, para um melhor entendimento do primeiro passo do algoritmo, os estados condições e conseqüências relativos ao modelo SAN apresentado pela Figura 4, estão definidos na Tabela 3 e ilustrados na Figura 14.

Algorithm 2 MDDGeneration($Inicial$)

1: Geração $Cd^{(e)}$ e $Cq^{(e)}$;
2: $RSS \leftarrow Inicial$;
3: **repeat**
4: $modificado \leftarrow falso$;
5: **for all** evento e **do**
6: **for all** $d \in Cd^{(e)}$ e o correspondente $q \in Cq^{(e)}$ **do**
7: **if** ($d \in RSS$ e $q \notin RSS$) **then**
8: $RSS \leftarrow Union(RSS, q)$;
9: $modificado \leftarrow true$;
10: **end if**
11: **end for**
12: **end for**
13: **until** não $modificado$
14: **for** $i = N$ to 2 **do**
15: identifica nodos repetidos para i do RSS;
16: elimina nodos replicados redirecionando os ponteiros do nível $i - 1$ do RSS;
17: **end for**

Algorithm 3 Geração $Cd^{(e)}$ e $Cq^{(e)}$

for all evento e **do**
  for $i = 1$ to N **do**
    if $\mathcal{A}^{(i)}$ possui evento e **then**
      if evento e possui taxa funcional **then**
        $Cd^{(e)}$[nível i] $\leftarrow$ estados habilitadores de $\mathcal{A}^{(i)}$ onde a função é não-zero;
      else
        $Cd^{(e)}$[nível i] $\leftarrow$ estados habilitadores de $\mathcal{A}^{(i)}$;
      end if
      $Cq^{(e)}$[nível i] $\leftarrow$ estados destino de $\mathcal{A}^{(i)}$;
    else
      if event e possui taxa funcional e depende de $\mathcal{A}^{(i)}$ **then**
        $Cd^{(e)}$[nível i] $\leftarrow$ estados de $\mathcal{A}^{(i)}$ onde a função é não-zero;
      else
        $Cd^{(e)}$[nível i] $\leftarrow$ *don't care*;
      end if
      $Cq^{(e)}$[nível i] $\leftarrow$ *don't care*;
    end if
  end for
end for

Assumindo-se que o modelo apresentado pela Figura 4 foi particionado em 2 sub-modelos tem-se um sub-modelo para cada autômato. Ainda, o autômato $A_{(1)}$ representa o nível 1 da estrutura de MDD, enquanto que o autômato $A_{(2)}$ representa o nível 2.

Tabela 3 – Relação entre Eventos e Estados Condição e Conseqüência

evento	estados condição	estados conseqüência
e_1	d_1	q_1
$e_{2(\pi_1)}$	d_2	q_2
$e_{2(\pi_2)}$	d_3	q_3
e_3	d_4	q_4
e_4	d_5	q_5
e_5	d_6	q_6

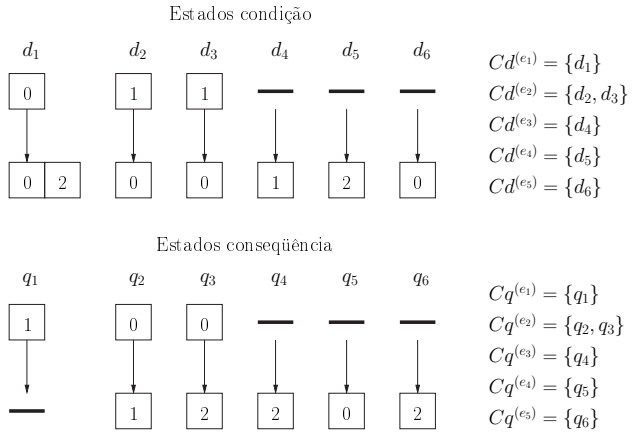


Figura 14 – Conjunto de Condição e Conseqüência

Analisando a Tabela 3 e Figura 14 observamos que os estados condição e conseqüência são respectivamente denominados por d_n e q_n. A variável n representa um valor inteiro seqüencial, que pode variar de $1..n$. Cada nível da estrutura de MDD representa um autômato do modelo, conforme definido pelo atual critério de partição. A existência de estados condição e conseqüência em um determinado nível das estruturas de MDDs iniciais está diretamente associada a existência do evento aos quais eles representam nos autômatos do modelo. Em outras palavras, no caso de um evento aparecer em apenas um autômato do modelo, os estados condição e conseqüência irão existir apenas no nível de MDD associado ao autômato específico. Por exemplo, o evento e_3 é local a $A_{(2)}$, e portanto os estados condição e conseqüência do mesmo, representados por d_4 e

q_4 aparecem apenas no nível 2 da estrutura de MDD. Em d_4 e q_4, o nível 1 está representado com um traço horizontal, que significa a não existência de estados condição e conseqüência para este nível da estrutura (considerando que o evento e_3 não aparece no autômato $A_{(1)}$). Além disso, caso um mesmo evento apareça mais de uma vez no mesmo autômato (existência de probabilidades), tal evento recebe diferentes identificadores d_n e q_n. Por exemplo, o evento e_2 aparece duas vezes no autômato $A_{(2)}$, em $e_{2(\pi_2)}$ e $e_{2(\pi_1)}$. Para tanto, a transição $e_{2(\pi_2)}$ é representado pelo estados d_2, q_2, enquanto que $e_{2(\pi_1)}$ é representado por d_3 e q_3. Em caso de eventos que representem taxas funcionais, os estados condição implicam na existência de estados alternativos para o disparo do evento. Por exemplo, o conjunto d_1 demonstra que o nível 2 da estrutura de MDD está condicionalmente associada a existência de 0 ou 2 na estrutura resultante de MDD, para que o evento e_1 possa disparar, uma vez que a função associada a e_1 assim o determina.

A segunda etapa do Algoritmo 2 (linha 2) é especificar o espaço de estados inicial para o conjunto RSS. O RSS pode ser inicializado com o estado global inicial do modelo ou com um conjunto de estados inicial (estados já conhecidos como atingíveis), através do uso de uma função.

De fato, a principal funcionalidade do Algoritmo 2 (linhas 3 a 13) consiste em um *loop* que permanece testando os estados condição para todos os eventos, e modificando o conjunto RSS através da adição dos estados conseqüência na estrutura de MDD resultante, quando necessário. A condição de parada do laço é verificada quando o conjunto RSS não é mais modificado e todos os eventos e tenham disparado no mínimo uma vez. Formalmente, esta etapa do algoritmo verifica se os estados condições $d \in {C_d}^{(e)} \in RSS$ para cada evento e do modelo. Caso a condição seja satisfeita, todos os estados conseqüências $q \in {C_q}^{(e)}$ correspondentes ao evento e são adicionados no conjunto RSS (caso $q \ni RSS$).

O tratamento das taxas funcionais durante o *loop* principal é um fator importante abordado pelo algoritmo, umas vez que é um diferencial para o formalismo de SAN. A ação tomada para modificar-se a estrutura de MDD resultante difere se o evento em questão possui ou depende de uma taxa funcional. Em tal situação, a seguinte característica é de fundamental importância quando um modelo utiliza taxas funcionais: *taxas funcionais impõem mudanças de estado unidirecionais nos autômatos do modelo. Ou seja, taxas funcionais modificam apenas o estado a qual ela pertence e não o estado a qual ela depende.* Essa premissa está representada no Algoritmo

3 pelas linhas linhas 4 e 11 respectivamente, impondo mudanças na estrutura MDD resultante apenas para os estados pertencentes e descartando quaisquer modificações para os estados dependentes.

As Figuras de 15 a 25 representam passo a passo a obtenção da estrutura de MDD resultante para o modelo SAN exibido pela Figura 4. O estado global utilizado é composto pelos estados iniciais $0^{(1)}0^{(2)}$, conforme também demonstrado pela Figura 15 Passo 1. Em particular, a Figura 15 também salienta o tratamento das taxas funcionais expressas pelo algoritmo, uma vez que o conjunto d_1 é disparado. Nesse caso, a existência de 0 no nível 2 da estrutura de MDD inicial habilita o disparo de d_1. Já, a estrutura MDD resultante após o disparo de d_1 tem alterado apenas o nível 1, uma vez que o nível 1 representa o autômato 1, a qual o evento e_1 pertence.

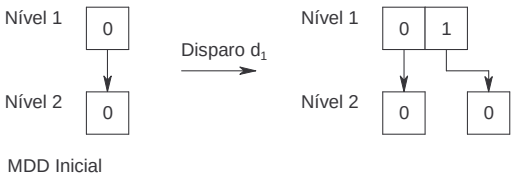


Figura 15 – Criação Estrutura de MDD Passo 1

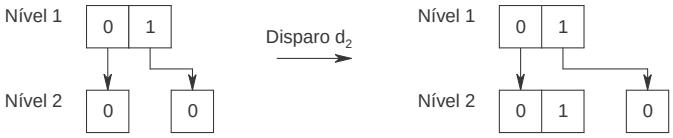


Figura 16 – Criação Estrutura de MDD Passo 2

Cabe salientar que do passo 1 ao passo 5 (Figuras de 15 a 19), a estrutura MDD resultante se modifica a cada nova iteração do algoritmo.

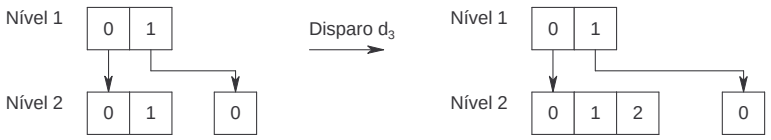


Figura 17 – Criação Estrutura de MDD Passo 3

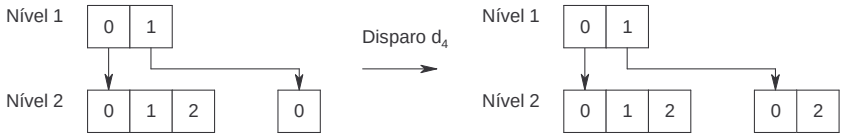


Figura 18 – Criação Estrutura de MDD Passo 4

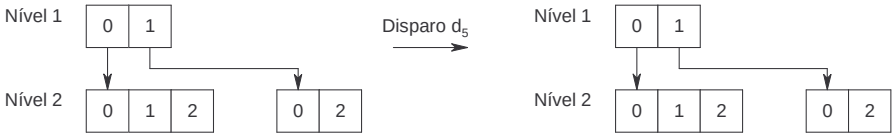


Figura 19 – Criação Estrutura de MDD Passo 5

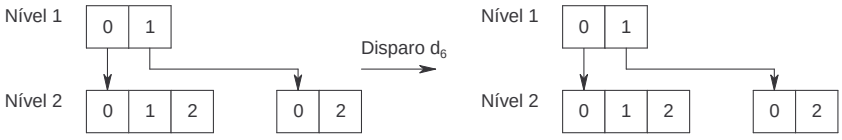


Figura 20 – Criação Estrutura de MDD Passo 6

Porém, do passo 6 ao passo 11 (Figuras de 20 a 25), todos os estados condição foram aptos a disparar, porém sem modificação nenhuma na estrutura MDD resultante. Então, o passo 11 (Figura 25) atinge a condição de parada proposta pelo algoritmo, uma vez que todos os estados

$C_d{}^{(e)}$ foram disparados e nenhuma modificação nova foi imposta com a adição dos estados $C_q{}^{(e)}$.

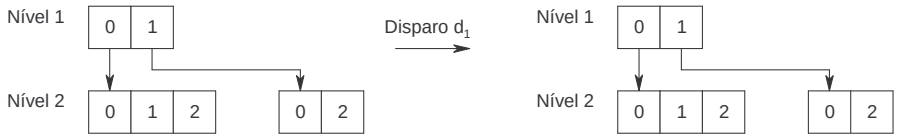


Figura 21 – Criação Estrutura de MDD Passo 7

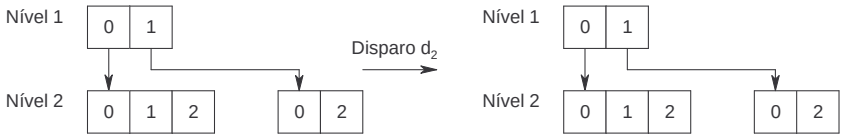


Figura 22 – Criação Estrutura de MDD Passo 8

Figura 23 – Criação Estrutura de MDD Passo 9

Dessa forma, a estrutura MDD resultante obtida após o laço principal do Algoritmo 2 representa o conjunto de estados atingíveis do modelo SAN (RSS). Todavia, ainda resta um último passo do Algoritmo 2, referente a redução da estrutura de MDD a fim de transformá-la em sua forma canônica (ROMDD). A etapa de redução representada pelas linhas 14 a 17 do Algoritmo 2 consiste na aplicação das regras de redução mencionadas em [21], e utilizadas para otimização do espaço de estados existente. Basicamente, o processo de redução consiste na varredura da estrutura de MDD a partir dos nodos terminais, eliminado-se nodos duplicados e redirecionando-se

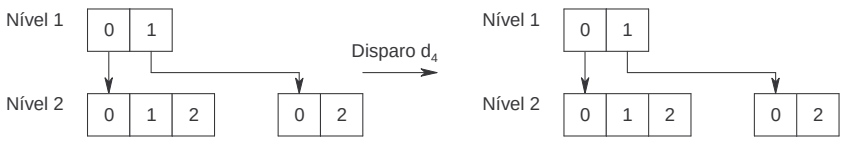


Figura 24 – Criação Estrutura de MDD Passo 10

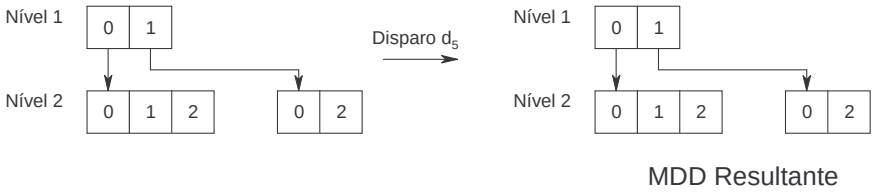


Figura 25 – Criação Estrutura de MDD Passo 11

os ponteiros dos nodos dos níveis superiores para um único exemplar de mesmo valor. A regra referente a eliminação de um nível inteiro, no caso deste assumir todos os valores possíveis não é aplicada, pois esta inclui uma complexidade maior para obtenção dos estados atingíveis representados pela estrutura de MDD.

O MDD representado pela Figura 26, já apresenta-se em sua forma canônica, pois após a aplicação das regras de redução, a estrutura permanece com o mesmo número de estados.

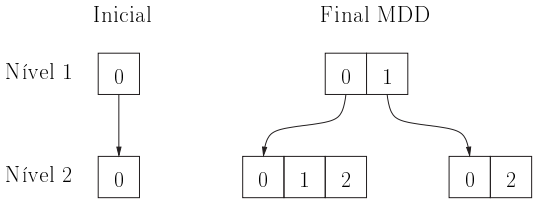


Figura 26 – Estrutura MDD Resultante

A aplicação das regras de redução do RSS como última etapa do algoritmo é ainda considerado um experimento inicial. Futuros estudos podem avaliar se a inclusão da etapa de redução no laço principal do algoritmo (depois do comando for, linha 14) traria benefícios adicionais ao

algoritmo, como menor consumo de memória durante a geração. Tal idéia pode vir a melhorar o desempenho do algoritmo, mas de qualquer forma, não foi explorada por este estudo. Cabe salientar, que a inclusão das regras de redução durante o loop principal poderá representar um consumo maior de CPU também. Portanto, uma avaliação de custo benefício faz-se se necessário, a fim de determinar se o ganho de memória comparado ao aumento do consumo de CPU compensará ou não.

4.3 Análise da Complexidade do Algoritmo

A análise da complexidade de um algoritmo refere-se ao tempo de execução que o mesmo apresenta para realização da tarefa proposta. O Algoritmo 2 será aqui avaliado através da Notação O, que representa a maior grandeza que este poderá assumir durante a geração do espaço de estados atingível para SAN.

Basicamente, o Algoritmo 2 é composto de três grandes etapas a serem avaliadas. A primeira parte é apresentada na linha 1, e refere-se a geração de estados $Cd^{(e)}$ e $Cq^{(e)}$, detalhadamente apresentada pelo Algoritmo 3. A segunda etapa importante a ser avaliada é o laço principal do Algoritmo 2, representado entre as linhas 3 e 13. Por fim, o laço de redução da estrutura de MDD, representado entre as linhas 14 e 17 do Algoritmo 2 também deve ser considerado. Pode-se dizer que estes três grande passos representam as principais variáveis associadas a execução da tarefa de geração e armazenamento do espaço de estados atingível em uma estrutura de MDD para o formalismo SAN. Portanto, a idéia é chegar-se a uma fórmula matemática expressa pela equação 4.1 abaixo e que representa a grandeza de tais variáveis:

$$O(x) + O(y) + O(z) \tag{4.1}$$

Onde x, y e z representam as variáveis associadas a cada uma das três etapas e que influenciarão diretamente no desempenho do algoritmo.

A primeira parte, linha 1 do Algoritmo 2, refere-se ao Algoritmo 3. Avaliando-se o mesmo, conclui-se que três sub-etapas são importantes e podem influenciar no seu desempenho. O co-

mando *for all evento e* apresentado na linha 1, enumera todos os eventos existentes no autômato, portanto *o número de eventos* está diretamente associado a complexidade aqui avaliada. A variável e representa este fator.

Além disso, o comando *for i=1 to N*, apresentado pela linha 2 também influenciará diretamente na grandeza apresentada pelo algoritmo. Este fator será representado pela variável N, que representa propriamente *o número de autômatos do sistema.*

Finalmente, esta primeira etapa também está diretamente associada a complexidade de execução do conjunto de instruções representados pelas linhas 3 a 16. Nessa fase, os MDDs iniciais são construídos através da atribuição de estados habilitadores e estados conseqüência do modelo. Então, a grandeza de ordem maior é aqui representada pelo *número máximo de estados de todos os autômatos do modelo.* Este aspecto será aqui representado pela variável μ.

Têm-se então a primeira parcela da fórmula matemática que representa a complexidade do algoritmo, conforme a equação 4.2:

$$O(e.N.\mu) + O(y) + O(z) \tag{4.2}$$

A segunda parcela a ser avaliada, diz respeito ao laço principal do Algoritmo 2, representado pelas linhas de 3 a 13. O comando *for all* $Cd^{(e)}$ $Cq^{(e)}$ referenciado na linha 6 testa todos os estados condição contra o RSS e aplica os estados conseqüência neste, caso necessário. Conclui-se então que este passo está diretamente associado a maior grandeza das estruturas de MDD que representam $Cd^{(e)}$ $Cq^{(e)}$ respectivamente. No pior caso, ambos MDDs terão o *tamanho máximo do número de estados existentes nos autômatos do modelo.* Portanto, considerando que a variável μ refere-se ao número máximo de estados do modelo para cada um dos conjuntos $Cd^{(e)}$ $Cq^{(e)}$, têm-se $(\mu.\mu)$.

Ainda, o laço *for all evento e* (linha 5) é facilmente identificado como diretamente associado ao número de eventos do modelo. Portanto, a variável e representa este aspecto.

O comando *repeat until* iniciado na linha 3 e finalizado na linha 13 ainda precisa ser considerado. Neste caso, o número de vezes que este laço será executado dependerá no pior caso,

do máximo número de eventos existentes no modelo. Então mais uma vez, a variável e deve ser considerada.

Dessa forma, chega-se a segunda parcela da fórmula matemática para complexidade do algoritmo, conforme representado em 4.3:

$$O(e.N.\mu) + O(e^2.\mu^2) + O(z) \tag{4.3}$$

O laço de redução da estrutura *for i=N to 2*, referenciado pelas linhas de 14 a 16 do Algoritmo 2, é responsável por identificar os nodos repetidos da estrutura de MDD resultante, eliminando-os através do redirecionamento dos ponteiros de encadeamento. Os fatores relevantes nesse caso, são portanto *o número máximo de autômatos*, representado pela variável μ, que determina de fato o valor de N para o laço *for i=N to 2*. Ainda, o número máximo de eventos, influencia no tamanho das estruturas de MDD iniciais($Cd^{(e)}$ $Cq^{(e)}$) que foram adicionadas ao MDD resultante, representado pelo conjunto RSS. Então, a variável e está também diretamente associada e deve ser levada em conta, conforme explicitado pela terceira parcela da equação 4.4, que também representa a equação final, ainda não reduzida:

$$O(e.N.\mu) + O(e^2.\mu^2) + O(e.N) \tag{4.4}$$

Finalmente, é necessário reduzir-se as grandezas repetidas apresentadas pela equação 4.4, uma vez que as mesmas se sobrepõe, não sendo necessário considerá-las mais de uma vez. A terceira parcela $O(e.N)$ está embutida na primeira parcela da equação 4.4, e portanto pode ser eliminada. Porém a primeira e segunda parcelas representam grandezas de ordens diferentes e devem ser mantidas. Chega-se portanto a equação final que representa a complexidade do algoritmo aqui estudado, representada pela equação 4.5:

$$O(e.N.\mu + e^2.\mu^2) \tag{4.5}$$

Considerações finais a respeito da equação 4.5 salientam que em caso o número de estados ou número de autômatos dobrar, o tempo de execução do algoritmo poderá aumentar em no máximo 16 vezes, conforme equação 4.6 representada abaixo:

$$(2e)^2.(2\mu)^2 = 4e^2.4\mu^2 = 16e^2.\mu^2 \tag{4.6}$$

4.4 Implementação do Algoritmo

Um protótipo do Algoritmo 2 aqui apresentado foi implementado no âmbito do grupo PEG (Performance Evaluation Group) por um bolsista também engajado em estudos da mesma área de interesse. A linguagem de programação C++ foi utilizada para a implementação deste. De fato, referimos a este como um protótipo, pois a implementação do uso de funções, mecanismo existente em SAN, ainda não foi coberto, assim como questões referentes a performance.

A implementação deste primeiro protótipo visa a realização de testes a fim de verificar a corretude do algoritmo aqui proposto, através de exemplos de modelagem apresentados nos próximos Capítulos. Esta primeira versão também permite a realização de alguns testes de performance a fim de identificar possíveis áreas a serem melhoradas no algoritmo. De fato é sabido através dos testes realizados para efeitos de comparação das técnicas MDD e Vetor de Booleanos (Capítulo 5) que a etapa de geração dos estados condição e conseqüência, primeira etapa imposta pelo Algoritmo 2, pode ser melhorada, uma vez que demonstrou consumir bastante memória durante execução do algoritmo. O Capítulo 7 referencia uma possível alternativa para melhoria de tal etapa.

Este Capítulo apresentou o Algoritmo capaz de gerar e armazenar espaço de estados atingível para o formalismo de modelagem SAN. Ainda, a análise da complexidade deste foi avaliada, com intuito de determinar quais as variáveis que influenciam no tempo de execução requerido durante a realização da tarefa proposta. Os próximos Capítulos irão utilizar o protótipo implementado para realização de testes de exemplos de modelagem, a fim de que a corretude do algoritmo proposto seja comprovada. Ainda, experimentos que determinam a influência da ordem dos autômatos com a compactação da estrutura de MDD também serão apresentados.

5 Comparativo entre MDD e Vetor de Booleanos

Apresenta-se nesse capítulo um comparativo do gasto de memória para as técnicas MDD e Vetor de Booleanos. Exemplos de modelagem são apresentados, estabelecendo-se conclusões relativas ao consumo final de memória para ambas técnicas.

5.1 Exemplos de Modelagem

Esta seção apresenta diversos exemplos SAN modelados com MDD. A idéia é ir aumentando o PSS de modelos SAN com RSS reduzidos (menores que 50%), a fim de estabelecer-se um comparativo entre as técnicas de MDD e Vetor de Booleanos com relação ao consumo final de memória para armazenar-se os estados atingíveis.

De fato, a fim de estabelecer-se um comparativo do gasto de memória para o uso de MDD e Vetor de Booleanos, primeiramente necessita-se entender como é calculado o consumo final de memória para ambos. No caso do Vetor de Booleanos, para calcular-se o consumo final de memória basta dividir-se o espaço de estados produto do modelo por 8 (lembrando-se que este consome apenas um bit para armazenar cada possível estado). No caso da técnica MDD, 2 bytes são utilizados para armazenar o valor de um nodo da estrutura, mais 2 ponteiros de encadeamento de 4 bytes cada, contabilizando um total de 10 bytes para armazenagem de cada nodo da estrutura de MDD. A Figura 27 exibe a estrutura de um nodo MDD para facilitar o entendimento do cálculo de memória aqui referido. Basicamente, cada nodo necessita armazenar um valor inteiro de 0 a N e o encadeamento com os nodos vizinhos ao lado e abaixo.

Dessa forma, as fórmulas a seguir representam como calcular-se o consumo final de memória para cada uma das técnicas:

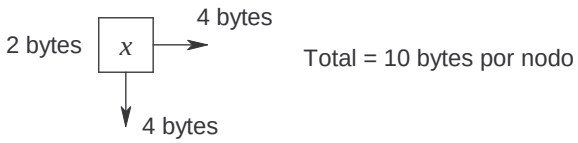


Figura 27 – Estrutura de um Nodo MDD

$$\text{Memoria\_Vetor\_Boleanos} = [PSS \div 8]\, bytes$$

$$\text{Memoria\_MDD} = [\text{num\_nodos\_MDD} \times 10]\, bytes$$

5.1.1 Modelos SAN

Modelos SAN com e sem semântica, que variam o número de autômatos e o conjunto de eventos sincronizantes e locais são apresentados nessa subseção.

O modelo SAN 1 apresentado pela Figura 28 possui 4 autômatos, 6 eventos sincronizantes e 8 eventos locais. O PSS do modelo é pequeno, apenas 81 estados e com 41 estados globais atingíveis (RSS). A estrutura MDD gerada pela técnica está apresentada pela Figura 29 e apresenta um total de 17 nodos.

O modelo SAN 2 apresentado pela Figura 30 possui 5 autômatos, 7 eventos sincronizantes e apenas 1 evento local. O PSS do modelo é um pouco maior, porém ainda considerado pequeno, 500 estados e 20 estados globais atingíveis (RSS). A estrutura MDD gerada pela técnica está apresentada pela Figura 31 e possui 19 nodos.

O modelo SAN 3, Pares de Autômatos (Figura 32), descreve um modelo com 6 autômatos,

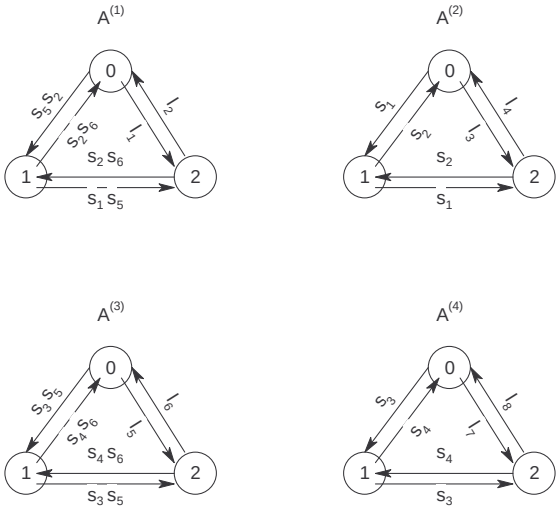


Figura 28 – Modelo SAN 1

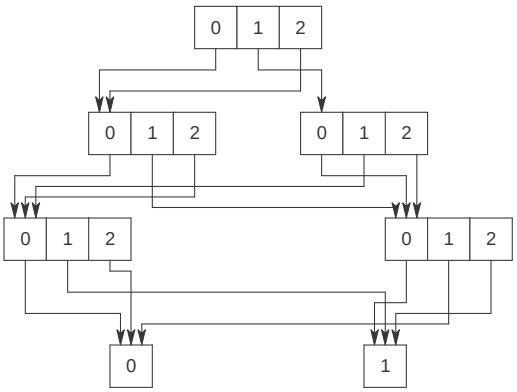


Figura 29 – MDD - Modelo SAN 1

sem taxas funcionais, 3 eventos locais e 6 eventos sincronizantes. Um modelo SAN compacto, com um espaço de estados produto de 729 estados, porém com um RSS de apenas 80 estados. Esse modelo SAN, possui uma interdependência aos pares, conforme explicado abaixo:

- autômato $A^{(1)}$ e $A^{(2)}$ possuem eventos sincronizantes s_1 e s_4;
- autômato $A^{(3)}$ e $A^{(4)}$ possuem eventos sincronizantes s_2 e s_5 ;
- autômato $A^{(5)}$ e $A^{(6)}$ possuem eventos sincronizantes s_3 e s_6;

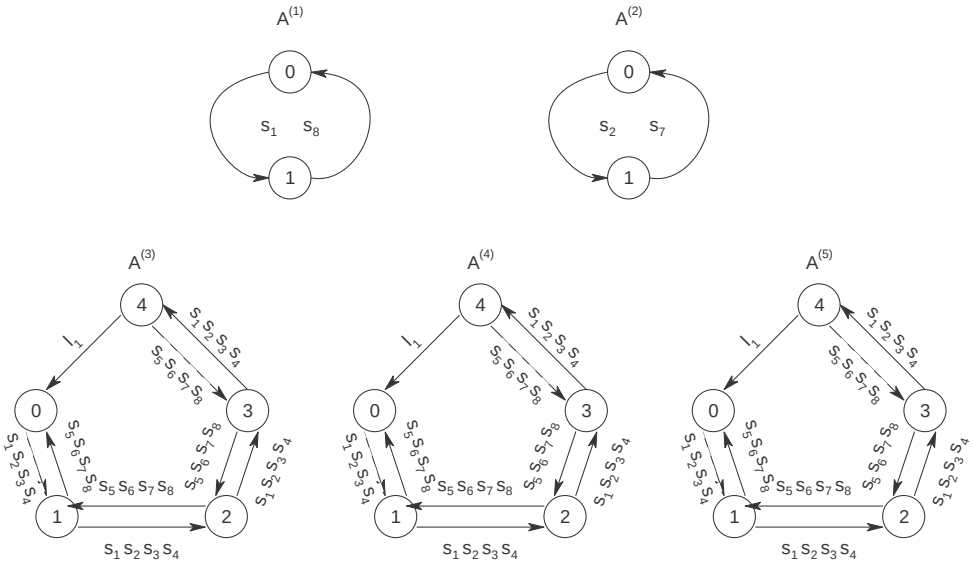


Figura 30 – Modelo SAN 2

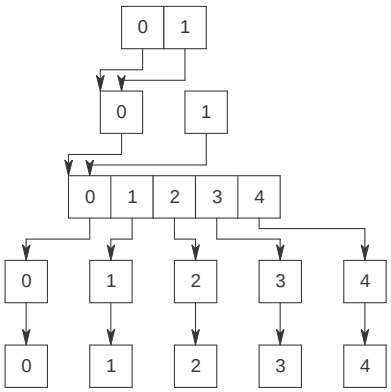
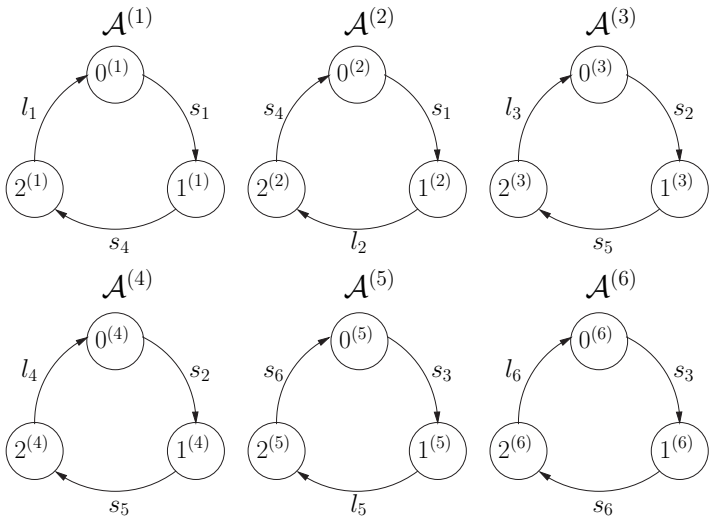


Figura 31 – MDD - Modelo SAN 2

O MDD resultante que representa o RSS do modelo é exibido pela Figura 33 e possui 18 nodos.

O modelo SAN 4 apresentado pela Figura 34 possui 7 autômatos, 14 eventos sincronizantes e apenas 1 evento local. O PSS do modelo é bem maior comparado aos exemplos anteriores, 131.072 estados com apenas 9 estados globais atingíveis (RSS). A estrutura MDD gerada pela técnica possui 46 nodos e está apresentada pela Figura 35.

A Tabela 4 faz um comparativo dos 4 modelos apresentados até aqui, listando da esquerda



Tipo	Evento	Taxa
loc	l_1	λ_1
loc	l_2	λ_2
loc	l_3	λ_3
loc	l_4	λ_4
loc	l_5	λ_5
loc	l_6	λ_6
syn	s_1	λ_7
syn	s_2	λ_8
syn	s_3	λ_9
syn	s_4	λ_{10}
syn	s_5	λ_{11}
syn	s_6	λ_{12}

Figura 32 – SAN 3 - Eventos Sincronizantes entre Pares de Autômatos

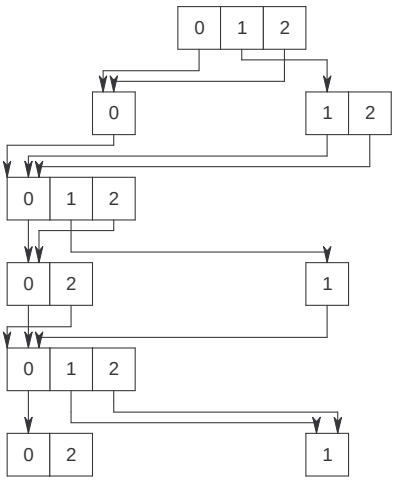


Figura 33 – MDD - Pares de Autômatos

para direita os seguintes dados: modelo SAN avaliado, PSS do modelo, RSS do modelo, percentual de estados globais atingíveis (percentual de RSS sob PSS), consumo final de memória para a técnica MDD e consumo final de memória para técnica Vetor de Booleanos.

Observando-se a Tabela 4 verifica-se que modelos SAN com PSS pequenos, mesmo apresentando um elevado número de estados inatingíveis, a técnica MDD não compensa. Por exemplo, tomando-se o modelo SAN 3, que possui apenas 4% de estados atingíveis sob o percentual de todo o PSS, o consumo final de memória da técnica MDD é ainda em torno de 50% maior.

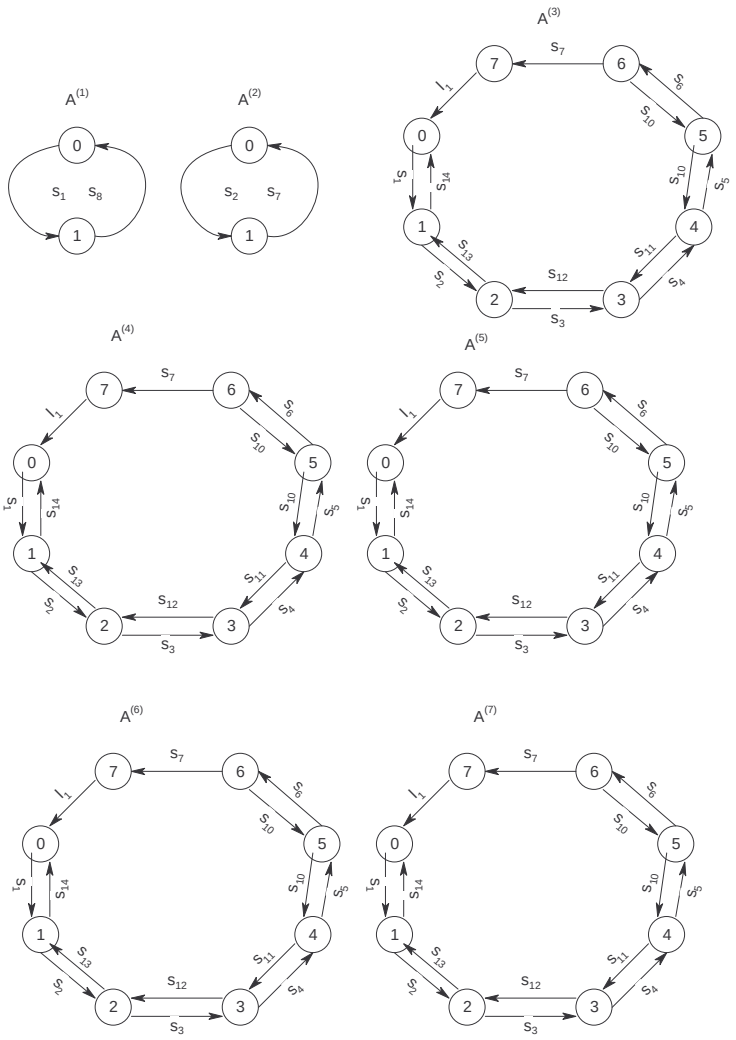


Figura 34 – Modelo SAN 4

Tabela 4 – Comparativo entre MDD e Vetor de Booleanos

Modelo	PSS	RSS	% RSS	MDD	Vetor de Booleanos
SAN 1	81	41	50%	180 bytes	11 bytes
SAN 2	500	20	4%	190 bytes	63 bytes
SAN 3	729	80	11%	180 bytes	92 bytes
SAN 4	131.072	9	0.006%	460 bytes	16.384 bytes

Porém, considerando o modelo SAN 4, onde o PSS já é um pouco maior, e o número de estados atingíveis menor que 1%, a economia de memória utilizando-se MDD é de cerca de 97%.

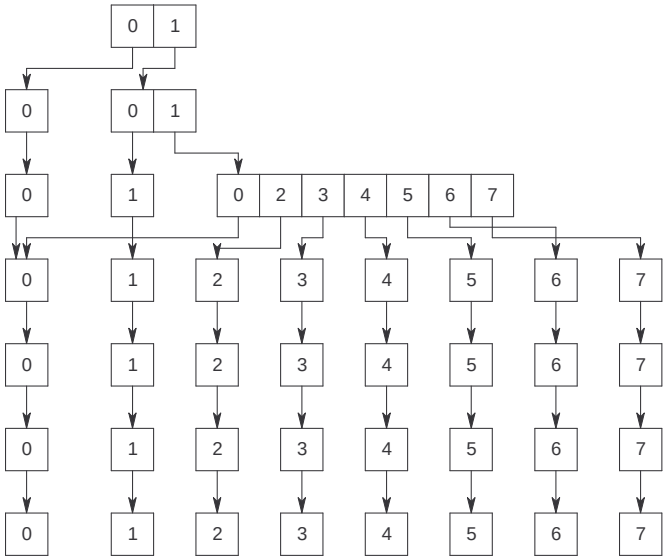


Figura 35 – MDD - Modelo SAN 4

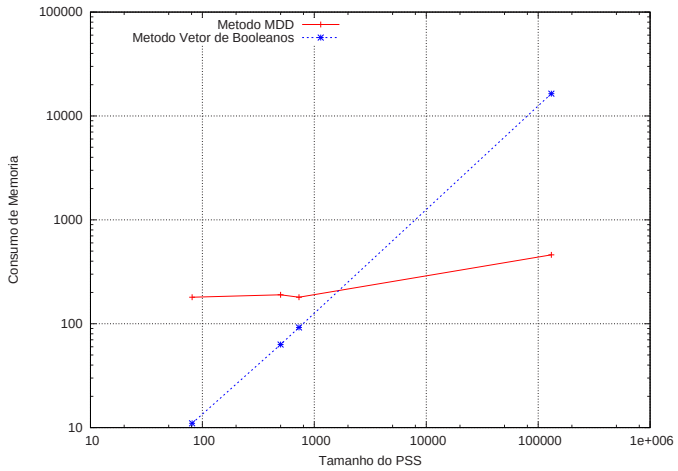


Figura 36 – Comparação do Consumo de Memória para Modelos SAN diversos

O Gráfico 36 também representa o comparativo de memória para os exemplos SAN mencionados acima. Novamente, percebe-se que o uso de MDD para modelos SAN com espaço de estados pequeno não é eficiente, mesmo o modelo possuindo um grande percentual de estados inatingíveis.

Continuando a coleta de dados para o estabelecimento do comparativo entre as técnicas,

avalia-se a seguir o clássico modelo Jantar dos Filósofos. Em particular, a avaliação deste modelo permite aumentar-se o número de filósofos a fim de variar-se bastante os valores para PSS e RSS.

O clássico modelo do Jantar dos Filósofos, amplamente citado na literatura SAN [8] foi avaliado com valores de 3, 4, 5, 6, 7, 8, 9, 10, 11 e 12 filósofos. A cada novo filósofo adicionado, 3 novos eventos sincronizantes mais um novo autômato são adicionados ao modelo, aumentando-se dessa forma as variáveis que trazem mais complexidade a esse modelo SAN.

No caso de 3 filósofos, o modelo SAN possui 9 eventos sincronizantes e está exibido pela Figura 37. Os lugares D e E representam o uso do garfo com a mão direita e esquerda. E o autômato com os estados U e L representam que o recurso garfo está em Uso ou $Livre$. O espaço de estados produto é representado por 216 estados, com apenas 12 estados atingíveis (RSS). Esse modelo assume que o primeiro filósofo pega o garfo com a mão esquerda primeiramente, enquanto que os outros pegam o garfo com a mão direita. Tal comportamento assimétrico é necessário para que o sistema não entre em *deadlock*. Este modelo representa a restrição de uso do recurso garfo.

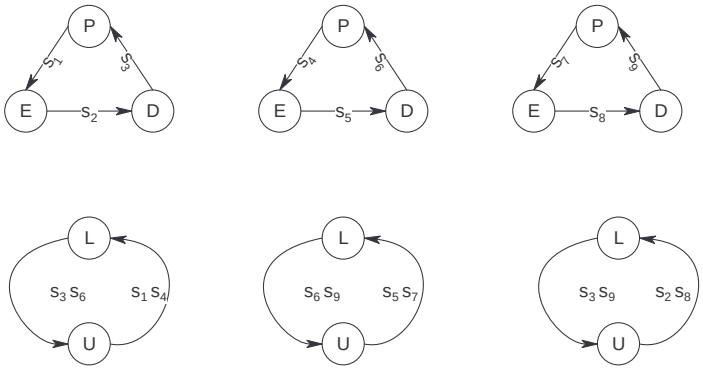


Figura 37 – Jantar dos Filósofos

O MDD resultante que representa o RSS do modelo é exibido pela Figura 38 e requer 30 nodos para armazenar o RSS.

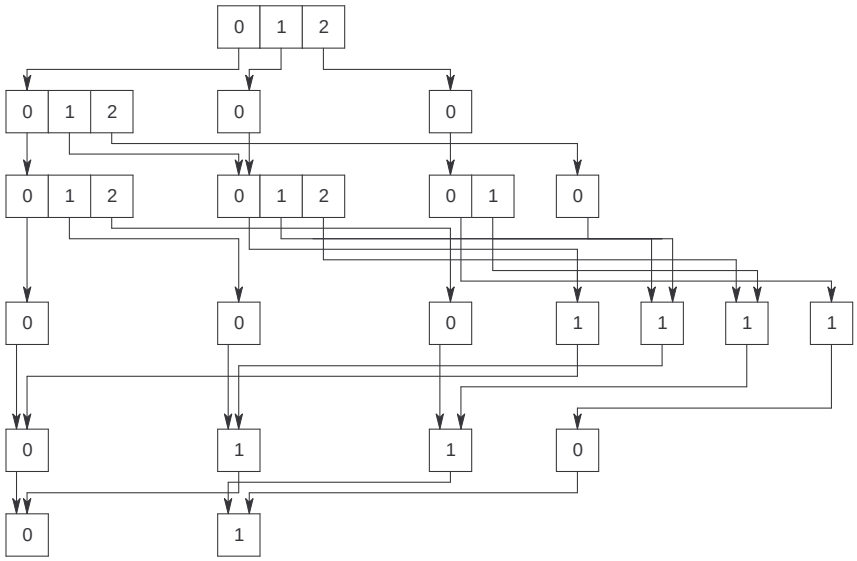


Figura 38 – MDD Resultante para Modelo Jantar dos (3) Filósofos

A Tabela 5 apresenta o comparativo do gasto de memória para as técnicas de MDD e Vetor de Booleanos, para o modelo SAN com 3, 4, 5, 6, 7, 8, 9, 10, 11 e 12 filósofos. Conforme o número de filósofos aumenta (conseqüente crescimento do PSS e diminuição do RSS), o método de MDD demonstra-se mais eficiente, tendo um gasto de memória bem inferior comparado ao Vetor de Booleanos. De fato, esse ganho dá-se principalmente pela característica do modelo em questão apresentar um grande número de estados inatingíveis aplicados a um espaço de estados produto consideravelmente elevado.

O Gráfico 39 também representa o comparativo de memória para o exemplo dos filósofos com o consumo final de memória ambos métodos. Este demonstra a eficiência do uso de MDD para o modelo SAN dos filósofos somente quando o modelo apresenta um PSS grande e um valor baixo de estados atingíveis (sempre menor que 1% do total apresentado pelo PSS). Esta análise conclui que o ganho efetivo da técnica inicia-se com o número de filósofos maior ou igual a 6, que possui PSS com cerca de 46.656 estados ou mais. A economia do consumo final de memória para 6 filósofos está em torno de 50%. Considerando o maior número de filósofos testado, que foi de 12, a economia do consumo final de memória fica em 99% (179.150 bytes gastos pela técnica MDD contra 272.097.792 bytes gastos pela técnica Vetor de Booleanos).

Tabela 5 – Consumo de Memória para Modelo Jantar dos Filósofos

Filósofos	PSS	RSS	% RSS	MDD	Vetor de Booleanos
3	216	12	5.5%	300 bytes	27 bytes
4	1.296	29	2.3%	650 bytes	162 bytes
5	7.776	70	0.9%	1.350 bytes	972 bytes
6	46.656	169	0.3%	2.750 bytes	5.832 bytes
7	279.936	408	0.15%	5.550 bytes	34.992 bytes
8	1.679.616	985	0.06%	11.150 bytes	209.952 bytes
9	10.077.696	2.378	0.0003%	22.350 bytes	1.259.712 bytes
10	60.466.176	5.741	0.009%	44.750 bytes	7.558.272 bytes
11	362.797.056	13.860	0.004%	89.950 bytes	45.349.632 bytes
12	2.176.782.336	33.461	0.002%	179.150 bytes	272.097.792 bytes

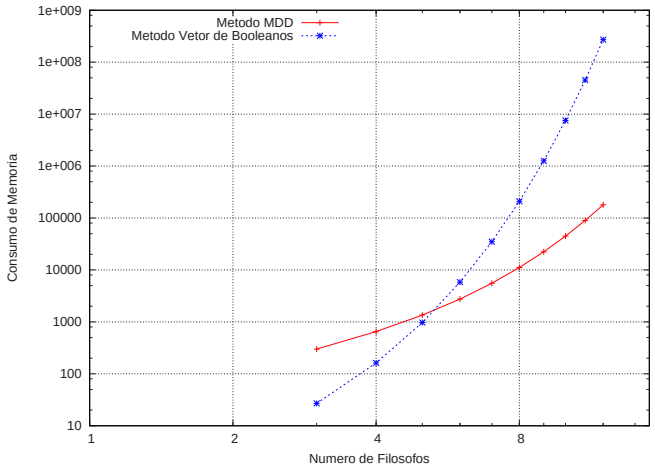


Figura 39 – Comparação do Consumo de Memória para Modelo Jantar dos Filósofos

O próximo modelo SAN apresentado é o modelo de Compartilhamento de Recursos. Este modelo representa N processos compartilhando R recursos. O modelo aqui avaliado possui 16 processos e número de recursos será variado de 2 a 6 exercitando-se assim diferentes valores de PSS e RSS.

O conjunto de autômatos que representam o modelo de Compartilhamento de Recursos é composto por um autômato para cada processo e outro autômato que controla o número de

recursos. O autômato que representa os processos possui 2 estados, que representam respectivamente *Ocioso* (*Sleep*) e *Em Uso* (*Using*). O autômato que representa os recursos possui um estado para cada recurso compartilhado. No caso de 2 recursos compartilhados, teremos 3 estados R_0, R_1, R_2 que representam respectivamente 0 recursos em uso, 1 recurso em uso e 2 recursos em uso. Eventos sincronizantes a_n entre os autômatos de processos e o autômato de recursos *alocam* recursos quando um processo passa do estado *Ocioso* para o estado *Em Uso*. Eventos sincronizantes r_n liberam recursos em uso quando autômatos que representam os processos passam do estado *Em Uso* para *Ocioso*. A Figura 40 exibe o modelo SAN para o exemplo em estudo.

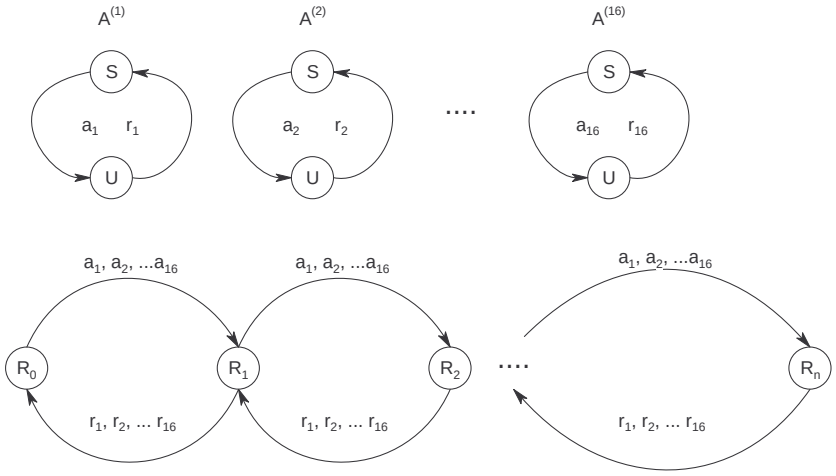


Figura 40 – Compartilhamento de Recursos

A Tabela 6 apresenta o comparativo do gasto de memória para as técnicas de MDD e Vetor de Booleanos, para o modelo de compartilhamento de recursos com 16 processos, variando-se o número de recursos para 2, 3, 4, 5, 6 recursos compartilhados. Além disso, esta apresenta os valores de PSS e RSS e percentual de RSS sob PSS para todas as configurações testadas. Conforme o número de recursos aumenta (conseqüente aumento do PSS e diminuição do RSS), o método de MDD demonstra-se mais eficiente, tendo um gasto de memória sempre bem inferior comparado ao Vetor de Booleanos. De fato, esse ganho dá-se principalmente pela característica do modelo em questão apresentar um grande número de estados inatingíveis aplicados a um espaço de estados produto grande, da mesma forma que o exemplo anterior dos filósofos.

Tabela 6 – Consumo de Memória para Modelo Compartilhamento de Recursos

Recursos	PSS	RSS	% RSS	MDD	Vetor de Booleanos
2	262.144	137	0.05%	790 bytes	32.768 bytes
3	393.216	697	0.2%	1.070 bytes	49.152 bytes
4	524.288	2.517	0.5%	1.330 bytes	65.536 bytes
5	655.360	6.885	1.05%	1.570 bytes	81.920 bytes
6	786.432	14.893	2.0%	1.790 bytes	98.304 bytes

O Gráfico 41 também representa o comparativo para o exemplo do compartilhamento de recursos destacando o consumo final de memória ambos métodos. Este demonstra a eficiência do uso de MDD para este modelo SAN para todos os valores de recursos testados. O ganho máximo se dá para o número de recursos 6, a qual o método MDD obteve um ganho de 98% comparando-se o consumo final de memória para ambos métodos. (98.304 bytes para o Vetor de Booleanos contra apenas 1.790 bytes para a técnica MDD).

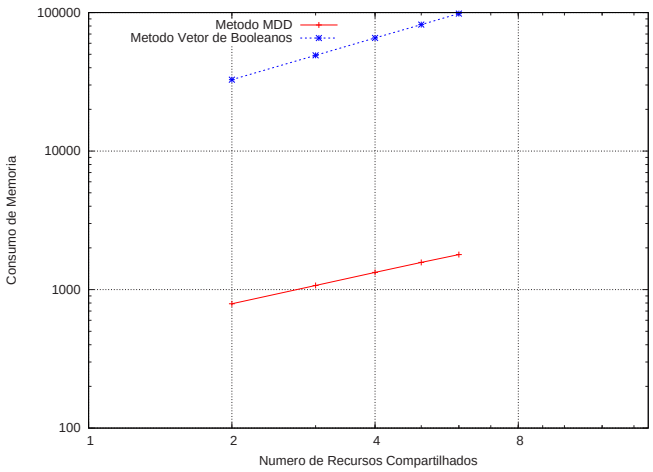


Figura 41 – Comparação do Consumo de Memória para Modelo Compartilhamento de Recursos

Este capítulo apresentou uma série de modelos SAN avaliados pelas técnicas de Vetor de Booleanos e MDD ao que diz respeito ao consumo final de memória. Avaliando os modelos apresentados, constatou-se que o ganho efetivo da técnica de MDD sob a técnica Vetor de Booleanos está associado a modelos SAN com um percentual muito baixo de estados atingíveis, juntamente

com modelos que apresentem espaço de estados produto consideravelmente grande. Verificou-se que apenas a existência de um elevado número de estados inatingíveis não constitui-se por si só característica indicativa para o uso da técnica de MDD. De fato, a combinação de duas importantes características que são elevado percentual de estados inatingíveis (menos de 2% para poucos e menor que 1% para a maioria) e um elevado tamanho de PSS (varia de acordo com o modelo, mínimo de 7.776 estados) são prerrogativas determinantes para a escolha da técnica de MDD. Os próximos Capítulos irão explorar como a variação de ordens dos autômatos na estrutura de MDD influencia a compactação desta. Ainda o Capítulo final traz as principais colaborações apresentadas pelo presente trabalho assim como sugestões de trabalhos futuros.

6 Análise do Reordenamento dos Autômatos

Apresenta-se neste capítulo como a ordem de disposição dos autômatos nos níveis da estrutura de MDD pode influenciar no número final de nodos resultantes. Para tanto, alguns modelos SAN foram testados e as conclusões são aqui apresentadas.

6.1 Exemplos de Modelagem

Para execução dos testes de ordem, foram escolhidos alguns modelos SAN, e testados com o Algoritmo 2 implementado. Várias ordens existentes foram exploradas, a fim de determinar-se uma heurística que aponte a estrutura MDD mais compacta possível que represente o RSS do modelo.

O resultado dos testes aponta que uma boa ordem para disposição dos autômatos na estrutura de MDD seria *manter os autômatos que têm interdependência o mais perto possível nos níveis da estrutura.* Autômatos que possuem interdependência, são autômatos que possuem eventos sincronizantes ou dependências funcionais entre si.

Basicamente três modelos SAN foram testados e avaliados nas seguintes condições: para cada combinação de ordem proposta, avaliou-se o número de nodos antes da aplicação da etapa de redução *(Núm. Máximo)*, e o número de nodos após a aplicação da etapa das regras de redução (Algoritmo 2) *(Núm. Final)*. O objetivo desse teste é encontrar qual a combinação de ordem que resulta em um MDD mais compacto (número final de nodos menor) e com o menor número de reduções necessárias (*Num. Final - Num. Máximo*).

O primeiro exemplo (Figura 32) descreve um modelo com 6 autômatos, sem taxas funcionais,

3 eventos locais e 6 eventos sincronizantes. Um modelo SAN compacto, com um espaço de estados produto de 729 estados, porém com um RSS de apenas 80 estados. Esse modelo SAN, possui uma interdependência aos pares, conforme explicado na seção anterior 5.1.

Tabela 7 – Exemplo de Eventos Sincronizantes entre Pares de Autômatos

ordem	max.	final
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}$	179	18
$\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}$	183	18
$\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}$	183	18
$\mathcal{A}^{(5)}\mathcal{A}^{(2)}\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}$	168	24
$\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(4)}\mathcal{A}^{(2)}$	200	30
$\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(4)}\mathcal{A}^{(1)}\mathcal{A}^{(6)}\mathcal{A}^{(2)}$	212	30
$\mathcal{A}^{(3)}\mathcal{A}^{(1)}\mathcal{A}^{(5)}\mathcal{A}^{(2)}\mathcal{A}^{(6)}\mathcal{A}^{(4)}$	203	42
$\mathcal{A}^{(1)}\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(2)}\mathcal{A}^{(4)}$	203	42
$\mathcal{A}^{(3)}\mathcal{A}^{(1)}\mathcal{A}^{(5)}\mathcal{A}^{(2)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}$	215	42
$\mathcal{A}^{(3)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(4)}\mathcal{A}^{(1)}\mathcal{A}^{(6)}$	224	42
PSS: 729 - RSS: 80		

Avaliando o *Número Final* de nodos gerados (Tabela 7), verifica-se que as melhores ordens são aquelas em que a interdependência entre os pares de autômatos estão aproximadamente dispostas nos níveis da estrutura de MDD (pares $\mathcal{A}^{(1)}\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}\mathcal{A}^{(4)}$ e $\mathcal{A}^{(5)}\mathcal{A}^{(6)}$) *(Número Final = 18)*.

O segundo exemplo apresentado (Figura 42), descreve um modelo de 7 autômatos, sem taxas funcionais, 9 eventos locais e 4 eventos sincronizantes. O espaço de estados produto do modelo contabiliza 512 estados, enquanto que o o espaço de estados atingíveis RSS representa apenas 64 estados. Conforme exibido na Figura 42, neste exemplo, o quinto autômato $A^{(5)}$ possui interdependência individual com quatro autômatos do modelo $A^{(1)}$, $A^{(2)}$, $A^{(3)}$ e $A^{(4)}$, enquanto que os dois últimos autômatos $A^{(6)}$ e $A^{(7)}$ possuem apenas eventos locais.

Observando-se o *Número Final* de nodos gerados (Tabela 8), verifica-se novamente que as melhores ordens são aquelas em que a interdependência entre os pares de autômatos estão apro-

Tabela 8 – Exemplo com Múltiplos Eventos Sincronizantes

ordem	max.	final
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	146	25
$\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(7)}$	166	25
$\mathcal{A}^{(6)}\mathcal{A}^{(7)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}$	206	25
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	142	26
$\mathcal{A}^{(1)}\mathcal{A}^{(5)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	150	26
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(6)}\mathcal{A}^{(4)}\mathcal{A}^{(7)}$	162	27
$\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(7)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}$	206	27
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	142	29
$\mathcal{A}^{(5)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	153	29
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(7)}$	170	29
$\mathcal{A}^{(1)}\mathcal{A}^{(6)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(7)}\mathcal{A}^{(4)}$	198	29
$\mathcal{A}^{(6)}\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(7)}\mathcal{A}^{(5)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}$	206	29
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(6)}\mathcal{A}^{(5)}\mathcal{A}^{(7)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}$	214	33
PSS: 512 - RSS: 64		

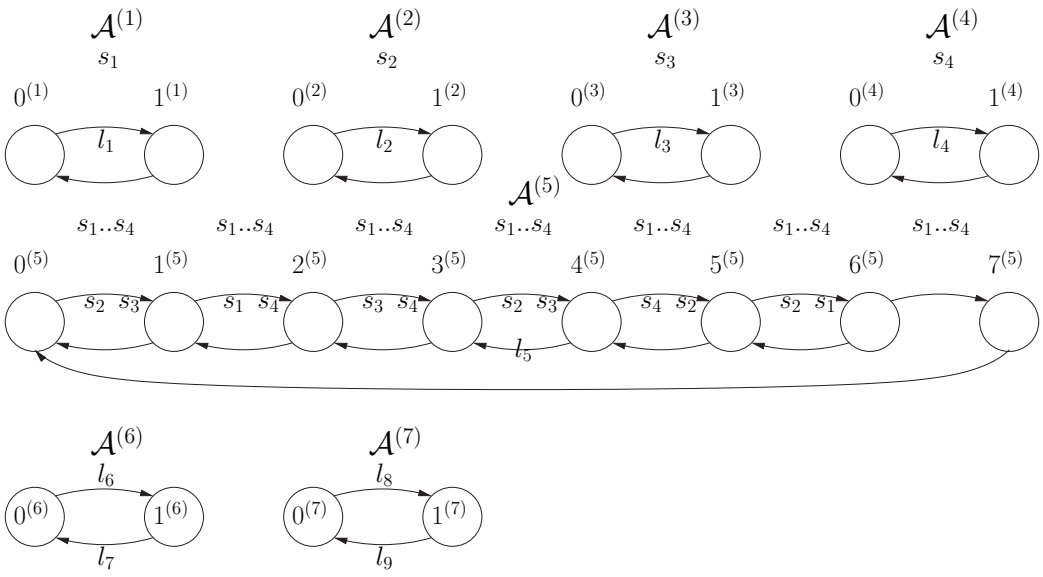
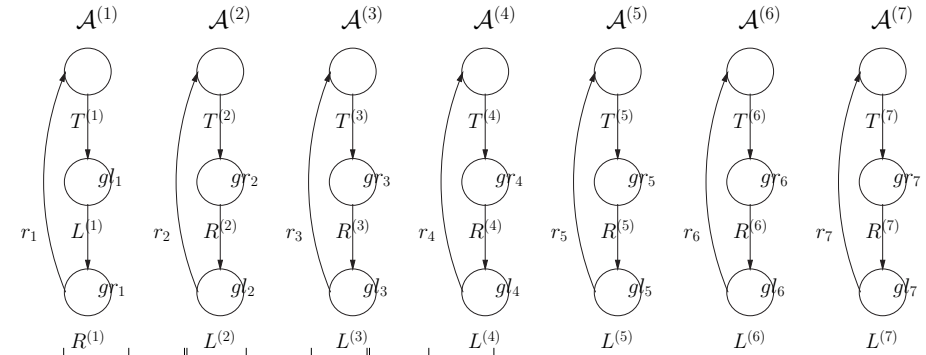


Figura 42 – Exemplo com Múltiplos Eventos Sincronizantes

ximadamente dispostas. *(Número Final = 25)*. Também observa-se que a posição do autômato independente não afeta o número final de nodos da estrutura MDD. Ou seja, contanto que os pares $\mathcal{A}^{(1)}\mathcal{A}^{(2)}$ e $\mathcal{A}^{(3)}\mathcal{A}^{(4)}$ estejam próximos, o número final de nodos permanece com o menor valor de 25.

O terceiro e último exemplo avaliado é novamente o clássico modelo do Jantar dos Filósofos. O modelo SAN desse exemplo (Figura 43) não possui nenhum evento sincronizante. Porém, ele representa 7 filósofos, com 14 eventos locais e com taxas funcionais, mais 7 eventos locais com taxas constantes. O espaço de estados produto é representado por 2187 estados, com apenas 408 estados atingíveis (RSS). Esse modelo assume que o primeiro filósofo pega o garfo com a mão esquerda primeiramente, enquanto que os outros pegam o garfo com a mão direita. Tal comportamento assimétrico é necessário para que o sistema não entre em *deadlock*. Este modelo representa a restrição de uso do recurso garfo apenas com taxas funcionais. Claramente, todos os autômatos do modelo possuem interdependência funcional com os seus dois vizinhos imediatos.



Type	Event	Rate	Type	Event	Rate	Type	Event	Rate
loc	gl_1	f_{gl_1}	loc	gr_1	f_{gr_1}	loc	r_1	μ
loc	gr_2	f_{gr_2}	loc	gl_2	f_{gl_2}	loc	r_2	μ
loc	gr_3	f_{gr_3}	loc	gl_3	f_{gl_3}	loc	r_3	μ
loc	gr_4	f_{gr_4}	loc	gl_4	f_{gl_4}	loc	r_4	μ
loc	gr_5	f_{gr_5}	loc	gl_5	f_{gl_5}	loc	r_5	μ
loc	gr_6	f_{gr_6}	loc	gl_6	f_{gl_6}	loc	r_6	μ
loc	gr_7	f_{gr_7}	loc	gl_7	f_{gl_7}	loc	r_7	μ

$f_{gl_1} = (st\ \mathcal{A}^{(2)} == T^{(2)}) * \lambda$ $\quad f_{gr_1} = (st\ \mathcal{A}^{(7)}\ != L^{(7)}) * \lambda$
$f_{gr_2} = (st\ \mathcal{A}^{(1)} == T^{(1)}) * \lambda$ $\quad f_{gl_2} = (st\ \mathcal{A}^{(3)} == T^{(3)}) * \lambda$
$f_{gr_3} = (st\ \mathcal{A}^{(2)}\ != L^{(2)}) * \lambda$ $\quad f_{gl_3} = (st\ \mathcal{A}^{(4)} == T^{(4)}) * \lambda$
$f_{gr_4} = (st\ \mathcal{A}^{(3)}\ != L^{(3)}) * \lambda$ $\quad f_{gl_4} = (st\ \mathcal{A}^{(5)} == T^{(5)}) * \lambda$
$f_{gr_5} = (st\ \mathcal{A}^{(4)}\ != L^{(4)}) * \lambda$ $\quad f_{gl_5} = (st\ \mathcal{A}^{(6)} == T^{(6)}) * \lambda$
$f_{gr_6} = (st\ \mathcal{A}^{(5)}\ != L^{(5)}) * \lambda$ $\quad f_{gl_6} = (st\ \mathcal{A}^{(7)} == T^{(7)}) * \lambda$
$f_{gr_7} = (st\ \mathcal{A}^{(6)}\ != L^{(6)}) * \lambda$ $\quad f_{gl_7} = (st\ \mathcal{A}^{(1)}\ != R^{(1)}) * \lambda$

Figura 43 – Exemplo Jantar dos Filósofos

Mais uma vez, observando-se o *Número Final* de nodos gerados (Tabela 9), verifica-se novamente que as melhores ordens são aquelas em que a interdependência entre os autômatos vizinhos estão dispostos aproximadamente *(Número Final = 45)*. Qualquer outra disposição de ordem

Tabela 9 – Exemplo Jantar dos Filósofos

ordem	max.	final
$\mathcal{A}^{(1)}\mathcal{A}^{(2)}\mathcal{A}^{(3)}\mathcal{A}^{(4)}\mathcal{A}^{(5)}\mathcal{A}^{(6)}\mathcal{A}^{(7)}$	716	45
$\mathcal{A}^{(1)}\mathcal{A}^{(7)}\mathcal{A}^{(2)}\mathcal{A}^{(6)}\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(4)}$	757	53
$\mathcal{A}^{(1)}\mathcal{A}^{(4)}\mathcal{A}^{(7)}\mathcal{A}^{(3)}\mathcal{A}^{(6)}\mathcal{A}^{(2)}\mathcal{A}^{(5)}$	836	118
$\mathcal{A}^{(1)}\mathcal{A}^{(3)}\mathcal{A}^{(5)}\mathcal{A}^{(7)}\mathcal{A}^{(2)}\mathcal{A}^{(4)}\mathcal{A}^{(6)}$	835	142
PSS: 2.187 - RSS: 408		

aplicada resultou em uma estrutura de MDD muito pior, considerando ambos o número de estados antes da redução (Núm. Máximo) e após a redução (Núm. Final).

7 Considerações Finais

Este estudo apresentou um algoritmo para armazenamento do RSS para o formalismo SAN utilizando a estrutura de MDD. No caso do formalismo de SPN, a utilização da estrutura de MDD para armazenamento do RSS já demonstrou ser claramente viável e vantajosa [17, 18]. Ferramentas SPN, que utilizam métodos baseados em estruturas de dados de tamanho proporcional ao espaço de estados, são capazes de representar modelos entre no máximo 10^4 e 10^6 [18] número de estados. O método baseado em MDD, demonstrou ser capaz de representar modelos com grandeza de 9.18×10^{626} e acredita-se que modelos ainda maiores são possíveis de ser representados [18].

O algoritmo aqui proposto para o formalismo de SAN demonstrou através de exemplos de modelagens ser igualmente viável. Testes apresentados no Capítulo 4 demonstraram que o consumo de memória final é bem inferior para modelos que possuem PSS maiores que 10^4 e com um vasto número de estados inatingíveis (cerca de menos de 1% de estados atingíveis). De fato, a economia final de memória para o exemplo dos Filósofos (12 filósofos) com PSS em torno de 10^{10} foi superior a 99% e para o modelo de compartilhamento de recursos com PSS de 10^6 ficou em torno de 98%.

Além disso, a obtenção do RSS do modelo SAN sem necessitar da geração do espaço de estados produto bem como a definição da função de atingibilidade do modelo, também pode ser considerado uma melhoria para o tratamento computacional desse formalismo. A técnica MDD exige apenas que um conjunto de estados iniciais seja informado *a priori*.

O fato de a ferramenta PEPS requerer uma quantidade elevada de memória disponível durante a avaliação da solução estacionária do modelo (segunda etapa da execução de um modelo SAN no PEPS que avalia as probabilidades de permanência nos estados globais), e não durante a fase de geração do espaço de estados atingível do modelo, levou este trabalho a não avaliação

formal do consumo de memória utilizado durante a execução do algoritmo. Os testes aplicados aos exemplos de modelagem focaram apenas o consumo final de memória, pois é deste que a segunda etapa do PEPS inicia. Porém, notou-se que o pico de memória gasto durante a execução do algoritmo proposto (Algoritmo 2) pareceu ser elevado. Primeiras análises apontam como causas desse problema a etapa de geração de estados condição e conseqüência deste algoritmo. Além disso, a realização de reduções da estrutura de MDD apenas após o laço principal deve ser reavaliada como opção para a redução do uso de memória durante a geração. Ambos temas constituem possibilidades de trabalhos futuros de pesquisa e que não foram aqui exploradas por tratar-se de melhorias que não interferem na formalização da corretude da técnica MDD aqui proposta.

A etapa de geração dos estados condições e conseqüências, apresentado pelo Algoritmo 3 propõe o armazenamento *a priori* de todos os estados $(C_d{}^{(e)})$ e $(C_q{}^{(e)})$ para todos os eventos do modelo. Uma possível otimização seria armazenar os eventos sincronizantes em uma estrutura que identifique em quais autômatos estes ocorrem, bem como seus estados origem. Essas informações são necessárias para determinação do conjunto $(C_d{}^{(e)})$ e $(C_q{}^{(e)})$ *a posteriori*. Além disso, outra estrutura com as informações topológicas do modelo SAN, por exemplo número de autômatos, eventos locais, número de estados faz-se necessário. Dessa forma, durante a exploração do espaço atingível do modelo, ambas estruturas permitirão a geração dos conjuntos condição e conseqüência durante a análise, remediando assim o consumo excessivo de memória requerido pelo seu atual armazenamento *a priori*.

A redução de nodos repetidos na estrutura de MDD, apresentado pelo Algoritmo 2 (linhas 14 a 17) também pode ser reavaliada visando melhorias de performance. Uma idéia seria realizar ciclos de reduções durante o laço principal deste algoritmo, a fim de verificar se o pico de memória diminuiria. De fato, o modelo MDD para SPN explora tal premissa. Todavia, a adição dos passos de redução no laço principal do Algoritmo 2 , deve cuidadosamente avaliar se o ganho de memória comparado ao aumento do consumo de CPU será válido ou não.

A real eficiência do algoritmo aqui proposto, com relação a quesitos de performance, não pode ainda ser avaliada completamente, pois a implementação de todas as funcionalidades formalmente propostas pelo algoritmo ainda não estão disponíveis. Especificamente, o tratamento

direto de funções (taxas funcionais) proposto pelo Algoritmo 3 ainda não foi implementado. Os exemplos aqui apresentado com taxas funcionais, foram tratados de forma implícita, através da utilização da técnica de conversão dos autômatos com taxas funcionais para autômatos correspondentes sem o uso de taxas funcionais. Cabe salientar que esta primeira versão do algoritmo tem como objetivo provar sua viabilidade e corretude de uso para o formalismo SAN, e tais prerrogativas foram comprovadas através dos exemplos de modelagens testados através do protótipo implementado.

O critério de ordem em que os autômatos são dispostos na estrutura de MDD é também um fator importante para otimização desta. Este foi exercitado através de exemplos de modelagens apresentados no Capítulo 6. Constatou-se que os autômatos que possuem interdependência entre si, como transições sincronizantes e taxas funcionais, devem ser dispostos o mais próximo possível nos níveis da estrutura de MDD, a fim de representar-se o espaço de estados atingível de uma maneira mais compacta.

A complexidade do algoritmo proposto foi cuidadosamente avaliada e demonstrou que o tempo de execução deste está quadraticamente associado a duas importantes variáveis do sistema, que são tamanho do autômato e número total de eventos do modelo. Através da Notação de maior grandeza O, provou-se que em caso o número de eventos e número de autômatos dobrar, o tempo de execução do algoritmo ficará no máximo 16 vezes maior.

Dentre demais tópicos de interesse, igualmente válidos a serem aprofundados, porém considerados melhorias a longo prazo, pode-se citar a integração do algoritmo proposto na ferramenta PEPS. Todavia, seria interessante que as melhorias aqui elencadas sejam primeiramente integradas no protótipo existente do algoritmo. Conseqüentemente, a proposta de inclusão do mesmo na ferramenta PEPS daria-se apenas após esta etapa de melhorias e consolidação da versão independente.

Conforme ressaltado, muito ainda precisa ser feito, mas acredita-se que esse estudo inicial seja uma promissora fonte de informação a respeito de técnicas de otimização do espaço de estados atingível para o formalismo de SAN, que quando consolidadas trarão inquestionáveis benefícios a implementação de realidades complexas utilizando-se o formalismo de SAN.

Referências

[1] K. Atif and B. Plateau. Stochastic Automata Networks for modelling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[2] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22(2):248–260, 1975.

[3] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In *Performance TOOLS 2003*, Urbana and Monticello, Illinois, USA, 2003. Springer-Verlag.

[4] B. Beyaert, G. Florin, P. Lonc, and S. Natkin. Evaluation of computer system dependability using Stochastic Petri Nets. In *Proceedings of the* 11^{th} *International Symposium on Fault Tolerant Computing*, pages 24–26, Portland, June 1981. IEEE Computer Society Press.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, New York, NY, USA, 1998.

[6] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[7] P. Buchholz and P. Kemper. Hierarchical reachability graph generation for Petri nets. *Formal Methods in Systems Design*, 21(3):281–315, 2002.

[8] M.-Y. Chung, G. Ciardo, S. Donatelli, N. He, B. Plateau, W. J. Stewart, E. Sulaiman, and J. Yu. A Comparison of Structural Formalisms for Modeling Large Markov Models. In *IPDPS Next Generation Software Program - NSFNGS - PI Workshop*, Santa Fe, New Mexico, USA, April 2004. ACM Press.

[9] G. Ciardo. What a structural world. In *PNPM '01: Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, page 3, Washington, DC, USA, 2001. IEEE Computer Society.

[10] G. Ciardo and A. S. Miner. Storage Alternatives for Large Structured State Spaces. In *9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1245 of *LNCS*, pages 44–57, St. Malo, France, 1997. Springer-Verlag.

[11] J. R. Jackson. Networks of waiting lines. *Operations Research*, 5:518–521, 1957.

[12] J. R. Jackson. Jobshop-like queueing systems. *Management Science*, 10:131–142, 1963.

[13] J.D.Ullman J.E.Hopcroft. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

[14] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operating Research*, 9:383–387, 1961.

[15] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor systems.* The MIT Press, 1986.

[16] P. M. Merlin and D. J. Farber. Recoverability of Communication Protocols - Implications of a Theorical Study. *IEEE Transactions on Communications*, 24(9):1036–1043, 1976.

[17] A. S. Miner. Effcient state space generation of GSPNs using decision diagrams. In *International Conference on Dependable Systems and Networks (DSN 2002)*, pages 637–646, Washington, DC, USA, June 2002. IEEE Computer Society.

[18] A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *Proceedings of the* 20^{th} *International Conference on Applications and Theory of Petri Nets*, pages 6–25, Williamsburg, VA, USA, June 1999. Springer-Verlag.

[19] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, C-31(9):913–917, 1982.

[20] J. D. Noe and G. J. Nutt. Macro E-nets representation of parallel systems. *IEEE Transactions on Computers*, C-22(8):718–727, 1973.

[21] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia. Petri net Analysis Using Boolean Manipulation. In *15th International Conference on Application and Theory of Petri Nets*, volume 815 of *LNCS*, pages 416–435, Zaragosa, Spain, 1994. Springer-Verlag.

[22] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, United States, 1985. ACM Press.

[23] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, 1980.

[24] F. J. W. Symons. The description and definition of Queueing Systems by numerical Petri nets. *Australian Telecommunication Research*, 13:20–31, 1980.

[25] K. S. Trivedi. *Probability & statistics with reliability, queuing, and computer science applications*. Englewood Cliffs: Prentice-Hall, New York, NY, USA, 1982.

[26] W. M. Zuberek. Timed Petri Nets and Preliminary Performance Evaluation. In *Proceedings of the* 7^{th} *Annual Symposium on Computer Architecture*, La Baule, France, 1980. ACM Press.