PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SOLUÇÃO NUMÉRICA DE DESCRITORES MARKOVIANOS A PARTIR DE RE-ESTRUTURAÇÕES DE TERMOS TENSORIAIS

RICARDO MELO CZEKSTER

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Paulo Henrique Lemelle Fernandes

Porto Alegre
2010

Dados Internacionais de Catalogação na Publicação (CIP)

C998sr Czekster, Ricardo Melo
Solução numérica de descritores markovianos a partir de re-estruturações de termos tensoriais / Ricardo Melo Czekster. – Porto Alegre, 2010.
195 f.

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

1. Informática. 2. Simulação e Modelagem em Computadores. 3. Cadeias de Markov – Computação. 4. Redes de Autômatos Estocásticos. I. Fernandes, Paulo Henrique Lemelle. II. Título.

CDD 003.3

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

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "Solução Numérica de Descritores Markovianos a partir de Reestruturações de Termos Tensoriais", apresentada por Ricardo Melo Czeksster, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, Processamento Paralelo e Distribuído , aprovada em 29/03/2010 pela Comissão Examinadora:

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

Prof. Dr. Fernando Luís Dotti - — PPGCC/PUCRS

Prof. Dr. Dalcidio Moraes Claudio – — FAMAT/PUCRS

Prof. Dr. Nicolas Maillard – — UFRGS

Homologada em 18/05/10, conforme Ata No. 08/10 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Para Elizabeth e Waldemar.

AGRADECIMENTOS

Agradeço o orientador Paulo Fernandes pelo incentivo, apoio, disponibilidade e dedicação durante o doutorado. Foi um período de aprendizagem constante e gratificante. Obrigado.

Sou também grato pela co-orientação durante o período *sanduíche* do doutorado em Grenoble/France dada por Bruno Gaujal, Florence Perronnin e Jean-Marc Vincent bem como em Edinburgh/Scotland por Jane Hillston e Stephen Gilmore.

Meu sincero obrigado por aceitar o convite para compor a banca de defesa de doutorado aos professores e pesquisadores Nicolas Maillard, Dalcidio Moraes Claudio e Fernando Luís Dotti.

Gostaria de incluir nos agradecimentos a ajuda do professor Avelino Zorzo pelas considerações efetuadas na proposta de tese e sugestões para a defesa.

Agradeço o apoio financeiro obtido durante o período, fornecido por Hewlett-Packard (HP) para a bolsa taxas, Projeto STGSD/Siemens (Março a Novembro 2006), CAPES (processo número 3322/06-8 – bolsa sanduíche de 12 meses em 2007), EPSRC CODA Project (*Edinburgh funding*, Fevereiro a Julho 2008) e Projeto Paleoprospec/Petrobras (Abril de 2009 a Abril 2010).

Aos funcionários do PPGCC/PUCRS, particularmente Sandra Rosa, Tiago Lingener, Régis Escobal da Silva e Talita Utteich, agradeço o apoio e a paciência durante toda a duração do doutorado.

Agradeço meus "orientadores" da vida, Elizabeth Melo Czekster e Waldemar Czekster. Obrigado, do fundo do coração por sempre acreditar e confiar no meu potencial. Eu dedico este trabalho a vocês, pelo empenho constante e por ter me ensinado os valores e caráter que tenho hoje. Obrigado meus queridos *braticzeks* Gustavo e Clarissa, eu não poderia ter irmãos melhores nesta vida.

Agradeço do fundo d'alma a minha esposa, Thais Webber dos Santos pela paciência e por tudo que passamos juntos neste período de crescimento profissional. Sem a tua ajuda e orientação, com certeza não teria conseguido. Muito obrigado pelas correções, por ter programado os algoritmos comigo, por ter me explicado os conceitos e pela paciência que sempre demonstraste. Obrigado também Maria Helena, Geraldo Webber dos Santos e Elisa Webber dos Santos, pelo apoio.

Um agradecimento aos demais membros da minha família, Rafael Guimaraes da Silva e Michele Dorneles Valent Czekster, bem como meus tios, tias, primos e primas e membros recém chegados, em especial Vera Melo Ferreira, Daniel Philomena Melo e pessoal do Projeto Paleoprospec.

Obrigado aos grandes amigos do PPGCC, em especial Luciano Copello Ost, Edson Ifarraguire Moreno, Afonso Sales, Leonel Pablo Tedesco, Rafael Iankowski Soares e Frederico Bartz Möller.

Obrigado aos amigos de longa data Luiz Pedro Sório de Carvalho, Frederico Sório de Carvalho, Carina Bello de Carvalho, Fausto Richetti Blanco, Marco Aurélio Stelmar Netto, Régis Poli Kopper, André Benvenuti Trombetta, Felipe Bacim de Araujo, Pedro Velho, Leonardo Brenner, Osmar Marchi dos Santos, Simone Regina Ceolin e Alessandro Ide Noriaki.

Je voudrais remercier mon ami Daouda Traoré et mon collègue de bureau Éric Saule.

I would like to thank my international friends, specially Michael David Pedersen, Allan Clark, Mirco Tribastone, Adam Duguid and everyone else at LFCS (Edinburgh).

SOLUÇÃO NUMÉRICA DE DESCRITORES MARKOVIANOS A PARTIR DE RE-ESTRUTURAÇÕES DE TERMOS TENSORIAIS

RESUMO

Os formalismos estruturados foram definidos ao longo dos anos com o objetivo de aumentar o nível de abstração e oferecer uma alternativa de modelagem mais sofisticada do que a proporcionada pelas tradicionais Cadeias de Markov. Exemplos de formalismos estruturados que utilizam álgebra tensorial para o armazenamento de seus descritores são as Redes de Autômatos Estocásticos, as Redes de Petri Estocásticas Generalizadas Superpostas e as Álgebras de Processo. Tais descrições utilizam primitivas de modelagem entre seus componentes capturando sua semântica operacional e permitindo a sua análise ao retornarem índices quantitativos de desempenho quando são resolvidos numericamente. Os mecanismos atuais de solução usam propriedades da Álgebra Tensorial (clássica ou generalizada) para multiplicar termos tensoriais de eventos entre os estados dos modelos (*i.e.*, um *descritor Markoviano*) por um vetor de probabilidade, que contém a solução estacionária ou transiente. Esta operação é chamada de *Multiplicação Vetor-Descritor* (MVD) e é realizada de três maneiras básicas: de forma esparsa (ineficiente em memória, eficiente em tempo), utilizando o Algoritmo *Shuffle* (eficiente em memória, ineficiente em tempo para algumas classes de modelos) ou através do Algoritmo *Split*, que é uma combinação das duas primeiras abordagens. A principal contribuição deste último foi a proposição de um método híbrido onde incrementa-se a memória (de forma razoável) para acelerar o cálculo efetuado por iteração. Entretanto, o principal desafio do Algoritmo *Split* é relativo à determinação de cortes de cada termo tensorial e em como re-estruturá-lo para reduzir o custo computacional por iteração, acelerando a convergência de modelos estruturados. Este trabalho aborda estes problemas, baseando-se em três eixos: i) na discussão das primitivas de modelagem para composição de sistemas através de formas mais abstratas de descrição, ii) nas diferentes formas de tratamento de termos tensoriais de descritores Markovianos para execução mais otimizada da MVD a partir de re-estruturações das ordens originais, e iii) na execução do Algoritmo *Split* com taxas constantes ou funcionais demonstrando os resultados obtidos para diversas classes de modelos. Para os casos observados, foi demonstrado através de experimentos que o melhor ganho, balanceando-se tempo e memória, é verificado quando as matrizes dos termos tensoriais são reordenadas, tratando as do tipo identidade na parte estruturada e avaliando-se os elementos funcionais uma única vez na parte esparsa. Ao avaliar as funções somente uma vez em todo o processo de MVD, converte-se os descritores generalizados para clássicos em tempo de execução e promove-se ganhos consideráveis em tempo para determinadas classes de modelos. Observou-se também que as atividades de sincronização ou comunicação entre os módulos ou partições envolvidas bem como o total de parâmetros das dependências funcionais realizam um papel crucial no desempenho obtido.

A presente tese é finalizada identificando as classes de modelos mais adequadas para a utilização do Algoritmo *Split*, propondo formas de re-estruturação de descritores Markovianos que privilegiem a esparsidade e a existência de matrizes do tipo identidade para balancear os custos em memória e tempo de execução.

Palavras-chave: Modelagem computacional; Formalismos estruturados; Soluções numéricas; Cadeias de Markov; Redes de Autômatos Estocásticos; Descritores Markovianos; Ferramentas computacionais; Álgebra Tensorial.

NUMERICAL SOLUTION OF MARKOVIAN DESCRIPTORS BASED ON RESTRUCTURATIONS OF TENSOR TERMS

ABSTRACT

Several formalisms have been defined throughout the years aiming the enhancement of the abstraction level which offer a more sophisticated modeling alternative than traditional Markov Chains. Examples of formalisms that use tensor algebra for descriptor storage are Stochastic Automata Networks, Superposed Generalized Stochastic Petri Nets, and Process Algebras. These descriptions employ modeling primitives among their components by capturing their operational semantics, and allow analysis by returning quantitative performance indexes when subjected to numerical solution. Solution mechanisms use both classic and generalized Tensor Algebra properties to multiply tensor terms of events among the states of the models (*i.e.*, *a Markovian descriptor*) by a probability vector, using stationary or transient measurements. This operation is called Vector-Descriptor Multiplication (VDM), and can be performed by three different methods: sparsely (memory inefficient, time efficient), using the *Shuffle* Algorithm (memory efficient, time inefficient, depending on the model) or through the *Split* Algorithm, a combination between the two former approaches. The main contribution of the Split approach was the proposition of a hybrid method where memory increments (in a reasonable fashion) are used to accelerate the calculations per iteration. On the other hand, the main challenge of the *Split* Algorithm is the determination of each division point (e.g. the cut parameter), and how the tensor terms must be restructured to reduce computational costs per iteration, allowing quicker convergence for structured models. The present work addresses these problems in three distinct ways: i) by discussing the modeling primitives for system composition through more abstract ways of description, ii) by treating each tensor term of Markovian descriptors in different manners for more optimized VDM solution restructuring the original orders, iii) by executing the Algorithm *Split* having both constant or functional rates, demonstrating the results for a variety of models. The experiments discussed here demonstrate that the best gain considering time and memory is verified when the matrices of the tensor terms are reordered, treating the identity ones in the structured part and evaluating functional elements just once in the sparse part. When the functions were evaluated only once in all VDM process, a conversion of generalized to classic descriptor took place in execution time, with considerable gain in time for some classes of models. In addition, it was observed that synchronization or communication activities between each module or system partition and the total number of functional parameters play a crucial role in the overall performance. The present thesis is finalized with the identification of the most suitable class of models for the utilization of the Split Algorithm, and the proposition of a restructuration of Markovian that privileges sparsity and identity matrices to balance memory costs and execution time.

Keywords: Computational modeling; Structured formalisms; Numerical solutions; Markov Chains; Stochastic Automata Networks; Markovian Descriptors; Computational tools; Tensor Algebra.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

ASP	*Alternate Service Patterns*
ATC	*Álgebra Tensorial Clássica*
ATG	*Álgebra Tensorial Generalizada*
AUNF	*Additive Unitary Normal Factor*
CPN	*Coloured Petri Nets*
CTMC	*Continuous Time Markov Chains*
DTMC	*Discrete Time Markov Chains*
EMPA	*Extended Markovian Process Algebra*
FAS	*First Available Server*
GD	*Grafo de Derivação*
GDC	*Grau de Dependência Constante*
GDG	*Grau de Dependência Generalizado*
GMRES	*Generalized Minimal Residual Algorithm*
GSPN	*Generalized Stochastic Petri Nets*
ipc	*Imperial PEPA Compiler*
MARCA	*Markov Chain Analyzer*
MC	*Markov Chains*
MDD	*Multivalued Decision Diagrams*
MIT	*Massachussets Institute of Technology*
MPA	*Markovian Process Algebras*
MT	*Matriz de Transição*
MVD	*Multiplicação Vetor-Descritor*
NUMA	*Non-Uniform Memory Access*
OQN	*Open Queueing Network*
PEPA	*Performance Evaluation Process Algebra*
PEPS	*Performance Evaluation of Parallel Systems*
PMC	*Processos Markovianos com Comunicação*
PMCG	*Processos Markovianos com Comunicação Generalizados*
PN	*Petri Nets*
PNML	*Petri Net Markup Language*
PSS	*Product State Space*

pwb	*PEPA Workbench*
RSS	*Reachable State Space*
RW	*Redes Wireless ad hoc*
SAN	*Stochastic Automata Networks*
SGSPN	*Superposed Generalized Stochastic Petri Nets*
SMART	*Symbolic Model checking Analyzer for Reliability and Timing*
SPN	*Stochastic Petri Nets*
UML	*Unified Modeling Language*
VDM	*Vector-Descriptor Multiplication*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

SUMÁRIO

1. INTRODUÇÃO

Um dos objetivos ao se mapear realidades em modelos consiste na verificação de medidas de desempenho que atestam a semântica operacional dos sistemas sob análise. Este mapeamento envolve diferentes etapas, onde a qualidade associada ao modelo varia de acordo com o tempo investido na sua caracterização. As etapas consideradas para um estudo computacional de desempenho através de modelagem analítica podem ser descritas das seguintes formas: a) *descrição*, através da utilização de abstrações, b) *parametrização*, que reflete as variáveis envolvidas e c) *avaliação*, resultando na extração e análise de índices quantitativos de desempenho.

A parte de *descrição* do sistema é realizada através da utilização de formalismos os quais definem cada funcionalidade com diferentes primitivas de modelagem. Um exemplo de formalismo são as Cadeias de Markov (*Markov Chains*, ou MC) que trabalham com a descrição dos estados e transições que um sistema assume. As transições conectam os estados e informam tanto as comunicações envolvidas quanto a frequência das mudanças. Tais frequências são mapeadas através da *parametrização* do modelo que envolve o estabelecimento das taxas observadas entre os estados adotados. Esta parte da modelagem é crucial pois corresponde à associação de valores às transições, podendo ser obtidas a partir de medidas realizadas previamente ou através de monitoramentos. A parte da *avaliação* trata da extração, interpretação e análise dos índices de desempenho do sistema sob estudo, onde caracteriza-se seu comportamento e operação.

O mapeamento de sistemas para modelos analíticos permite que sejam inferidos índices de desempenho que atestam a sua operação e o seu comportamento. Estas características auxiliam o processo de tomada de decisões ao usar abstrações para os problemas existentes. Neste sentido, a área de Avaliação de Desempenho e Modelagem Estocástica de sistemas torna possível a realização de análises com base nos resultados numéricos produzidos na solução dos modelos computacionais descritos pelos modelos. Estes mecanismos são utilizados na descoberta de eventuais gargalos antes que estes sejam colocados fisicamente em produção, ou seja, antes de se precisar comprar máquinas para um *cluster* ou aumentar o número de servidores em uma empresa.

Apesar da simplicidade de MC para modelar diferentes realidades e avaliar sistemas para extração de probabilidades de permanência em cada estado, observa-se que mesmo sistemas reais de tamanho restrito podem vir a possuir um conjunto de estados total elevado. Os problemas relacionados às representações do espaço de estados torna igualmente difícil a modelagem de realidades de grande porte. Este fator, conhecido por *explosão do espaço de estados* é recorrente modelagens puramente baseadas em MC e motiva pesquisas para amenizá-lo. Com o passar do tempo constatou-se que uma das formas de mitigá-lo ocorreu através da descrição de formalismos com um maior grau de *estruturação*, *i.e.*, abstrações para decompor um sistema em partes mais gerenciáveis. Exemplos de tais formalismos são as Redes de Autômatos Estocásticos (*Stochastic Automata Networks* - SAN) [PLA85,PLA91], as Redes de Petri [PET77,BAL07] e as Álgebras de Processos [HIL96,BAE05], entre outros.

Uma vez que o sistema é modelado com um determinado formalismo, escolhido a partir do conjunto de primitivas disponíveis, a próxima etapa consiste na determinação da existência ou não de suas características estacionárias. O objetivo é detectar se o equilíbrio foi atingido ou, caso contrário, analisar sua solução transiente, *i.e.*, quando o processo é executado em uma quantidade de tempo pré-determinada. Este processo é realizado através do mapeamento das taxas de ocorrência contidas nas transições considerando escala de tempo contínua ou das probabilidades de ocorrência, no caso de escala de tempo discreta [STE09]. Os valores são colocados em uma matriz que mapeia um sistema linear de equações contendo um número de variáveis correspondente ao total do número de estados. A solução do sistema é realizada através de métodos diretos ou iterativos [STE94].

A determinação da solução computacional de modelos trata de uma parte importante da análise de desempenho de realidades complexas, responsável pelo cálculo de índices que atestam de forma numérica e quantitativa, o desempenho e a operação dos sistemas modelados. Um exemplo que pode ser expresso através de modelagens analíticas são, entre outros, problemas de alocação, disponibilidade e utilização de recursos. Sistemas paralelos e distribuídos estão sujeitos às aplicações de tais mecanismos, permitindo, por exemplo, a detecção de gargalos, questões sobre validação de modelos e análises que visem a otimização ou o estudo dos estados que o sistema assume.

Em modelagem Markoviana, um dos principais eixos de pesquisa em Avaliação de Desempenho alinha-se à solução otimizada do sistema linear que descreve o modelo quando este é transposto para representações matriciais que capturam a mudança dos estados. Neste sentido, deve ser cuidadosamente observada a quantidade de memória exigida tanto para armazenar tais matrizes quanto os mecanismos que provêem a determinação do regime estacionário ou transiente dos modelos.

Técnicas tradicionais de armazenamento e solução através de abstrações tensoriais [PLA85], tais como as presentes no formalismo das SAN são consideradas um avanço significativo, pois a maneira de representação utilizada *nunca* armazena a matriz de transição completamente (como as MC). O formalismo de SAN utiliza representações tensoriais para os modelos, compostas por um conjunto finito e restrito de matrizes que compõem o denominado *descritor* Markoviano ou Kronecker. Este formalismo é, portanto, eficiente em termos de memória necessária para armazenar o modelo pois salva apenas matrizes esparsas de baixa dimensão, dependendo das transições de cada autômato[1].

A solução numérica de tais descritores Markovianos é igualmente foco de novas pesquisas e os algoritmos especializados para efetuar tal processo são chamados de algoritmos para a *Multiplicação Vetor-Descritor* (MVD). As operações complexas de multiplicação que devem ser efetuadas no nível do descritor Markoviano podem ser otimizadas de diferentes maneiras. Os descritores são formados de matrizes com elementos constantes ou funcionais, que são tipos mais complexos para o tratamento computacional apesar de serem simples primitivas de modelagem. Logo, dada a formação das matrizes que compõem o descritor, este pode ser classificado respectivamente como clássico ou generalizado. Neste último, as matrizes contêm elementos funcionais e são melhor explicados ao longo do trabalho.

[1] A dimensão de cada matriz é dependente do modelo estudado. Como será explicado ao longo do trabalho, estas matrizes possuem a dimensão igual ao número de estados de cada autômato.

As formas de MVD tratam cada termo tensorial (*i.e.*, uma representação para cada evento do modelo) de forma independente e permitem a permutação das suas matrizes internas, alterando a sua ordem original, para promover ganhos em termos de tempo para efetuar cada iteração. Um termo tensorial é composto por matrizes de diferentes tipos, tais como identidades, matrizes contendo um elemento ou matrizes com elementos funcionais, entre outros. Estas matrizes estão sujeitas a serem trocadas de posição entre si, entretanto, desconhece-se os efeitos de se fazer esta re-estruturação dos termos de descritores tensoriais. Esta tese analisará especificamente os algoritmos clássicos de MVD tais como o Algoritmo *Esparso* [STE94] e o Algoritmo *Shuffle* [FER98b] em perspectiva com o Algoritmo *Split* [CZE07]. Uma maior atenção será direcionada para este último pois, apesar de propor uma divisão dos termos tensoriais para aplicar uma solução híbrida, não se conhece ainda a melhor forma de realização destas divisões assim como suas consequências quando aplicadas a descritores generalizados.

A tese está direcionada para a descoberta das implicações das avaliações de taxas funcionais e a presença de matrizes do tipo identidade de cada termo tensorial para operar com menos operações de multiplicações em ponto-flutuante (relacionadas diretamente à complexidade envolvida para a MVD). O objetivo destes estudos aliado ao fato de serem permitidos reordenamentos dos termos é flexibilizar ao máximo o Algoritmo *Split* para trabalhar com descritores Markovianos com taxas funcionais, comparando com a abordagem atual do Algoritmo *Shuffle*. Como será mostrado ao longo da tese, a adoção de uma estratégia para dividir os termos tensoriais impacta no tempo gasto para descoberta do regime estacionário ou transiente.

1.1 Motivação

Os métodos de MVD dividem-se em duas classes distintas, conforme o custo em memória. De um lado, o Algoritmo *Esparso* é frequentemente inviabilizado pelos gastos em memória necessários para armazenamento da matriz de transição. Já o Algoritmo *Shuffle* é eficiente em memória e otimizado para tratar de elementos funcionais realizando ou não permutações na ordem lexicográfica[2] das matrizes para otimizar as avaliações e demais mecanismos para ganho de desempenho pesquisados em outros trabalhos [FER98b]. Neste sentido, o Algoritmo *Split* surgiu como uma alternativa viável ao proporcionar uma maneira balanceada em termos de tempo e memória de realização do processo de MVD [CZE07]. Esta abordagem mostrou-se ser bastante flexível, armazenando e agregando partes do descritor Markoviano.

Entretanto, desconhece-se as implicações das re-estruturações dos termos tensoriais dos descritores quando o Algoritmo *Split* é utilizado para realizar a MVD. Dadas as limitações impostas pelos descritores generalizados, aliadas à estrutura interna dos descritores clássicos e os tipos de matrizes envolvidas, observa-se a importância de se mapear tais consequências e antever potenciais problemas na solução dos modelos. Estes estudos devem proporcionar um maior entendimento sobre como o algoritmo pode executar para fazer uso das reordens, parâmetros de elementos funcionais e tipos de

[2]Na ordem original das matrizes descritas pelos autômatos.

matrizes para a solução de modelos. Por fim, deseja-se determinar a flexibilidade do Algoritmo *Split* e descobrir a melhor relação custo/benefício em termos de tempo e memória para sua execução.

1.2 Objetivo

O principal objetivo deste trabalho é propor um algoritmo de re-estruturação de termos tensoriais que, dadas as características de termos clássicos ou generalizados de sistemas representados através de descritores Markovianos, forneça uma solução mais acelerada em termos de tempo face aos métodos de MVD presentes atualmente. Em resumo, a memória a ser gasta não deve constituir-se em um impedimento para utilização do Algoritmo *Split*.

Para a proposição deste algoritmo é necessário definir as características que mais influenciam os termos tensoriais, bem como as formas de permutação que podem ser adotadas, a quantidade de matrizes do tipo identidade e o total de parâmetros para avaliação de cada função presente em um modelo. A partir destas informações será possível determinar as classes de modelos que o Algoritmo *Split* é melhor aplicado e questões relativas às limitações da sua flexibilidade.

1.3 Contribuição

Ao alcançar o objetivo, são esperados resultados relevantes no contexto da solução numérica de sistemas baseados em representações tensoriais presentes em formalismos estruturados para avaliação quantitativa de sistemas. Em um primeiro momento, a pesquisa auxiliará na determinação das classes de modelos melhor adaptadas para utilização do Algoritmo *Split* quando comparado com as abordagens clássicas de MVD. A partir destes resultados será construído um algoritmo que observe as características dos termos tensoriais de descritores clássicos e generalizados bem como suas matrizes internas e informe o quanto de memória será gasto para a busca dos índices de desempenho.

Em um segundo momento, ao se conhecer a estrutura interna de descritores Markovianos e os detalhes de execução do processo de MVD e seus principais algoritmos poderá ser definido um conjunto de regras de mapeamento para que diferentes formalismos estruturados que possuam uma representação tensorial sejam beneficiados com uma solução MVD otimizada. Isto ampliará as maneiras de se analisar o desempenho de sistemas já que será possível modelá-lo utilizando-se as primitivas existentes em um determinado formalismo e resolvê-lo com métodos que encontram-se no estado-da-arte de MVD com representação tensorial, dado que uma conversão para este formato exista e seja válida. Este formato descritivo fornecerá o conhecimento necessário para a proposta da implementação de uma ferramenta que resolva descritores Markovianos re-estruturando seus termos tensoriais para utilizar o Algoritmo *Split*.

1.4 Organização

A tese está organizada da seguinte forma: o Capítulo 2 aborda diferentes formalismos presentes na literatura, seguido pelo Capítulo 3 que demonstra formas mais abstratas de descrição de sistemas.

As definições são usadas no Capítulo 4, que trata sobre as formas de MVD e a operação dos seus algoritmos principais. Após abordar a MVD, demonstram-se as estratégias para divisão de termos tensoriais clássicos e generalizados no Capítulo 5, considerando-se as implicações teóricas e as características mais importantes. A seguir, explicam-se as funcionalidades da ferramenta GTAexpress no Captítulo 6 para, no Capítulo 7 apresentar os resultados obtidos de diferentes experimentos. A tese é finalizada pelo Capítulo 8 com um resumo das contribuições e as perspectivas futuras.

2. DESCRIÇÃO DE SISTEMAS ATRAVÉS DE FORMALISMOS

Uma das maneiras de se descrever realidades complexas é através da sua representação com um formalismo. Um formalismo é um conjunto de regras que mapeiam propriedades e características de sistemas a modelos e definem, de forma não ambígua, as entidades relacionadas e o funcionamento das suas interações. Pode-se, por exemplo, observar como os elementos de cada modelo transitam de condição em condição, trocando ou não de estado, de acordo com taxas ou frequências de ocorrência. O objetivo final de se realizar tais mapeamentos é permitir que sejam extraídos índices computacionais de desempenho tais como indicadores correspondentes à utilização dos recursos ou informações para inferir a capacidade média de um *buffer*[1], entre outros exemplos. Os formalismos para avaliação de desempenho são compostos por diferentes primitivas de modelagem que auxiliam na descrição das realidades estudadas. Para o caso dos formalismos de descrição com vistas à análise de sistemas complexos e estudo da solução analítica é comum constatar a existência de duas entidades principais: estados e transições. Os formalismos definidos na literatura distinguem-se entre si de diversas formas, entretanto, todos são fundamentalmente baseados no formalismo das Cadeias de Markov [STE94]. Este capítulo discutirá diferentes maneiras para abstração de sistemas em representações analíticas através da aplicação das primitivas de modelagem presentes em alguns dos formalismos Markovianos da atualidade. No contexto deste trabalho assume-se que o objetivo da descrição com formalismos é encontrar uma solução estacionária ou transiente para um dado sistema sem recorrer a outras práticas igualmente importantes porém menos precisas, dependendo do caso, tais como simulação.

O capítulo é iniciado com considerações preliminares sobre abstração e modelagem de sistemas em geral na Seção 2.1, seguido da definição de Cadeias de Markov na Seção 2.2, Redes de Petri e uma de suas variantes, as Redes de Petri Coloridas na Seção 2.3. A seguir, o capítulo aborda as Redes de Autômatos Estocásticos na Seção 2.4, Álgebra de Processos na Seção 2.5 e é finalizado com uma breve análise comparativa com as vantagens e desvantagens de cada formalismo na Seção 2.6.

2.1 Modelagem de sistemas

A modelagem computacional de realidades com múltiplos estados e transições é uma parte importante da avaliação analítica pois transforma descrições alto nível e primitivas em informações que atestam a operação e a qualidade dos sistemas sob estudo. A necessidade de métodos formais para avaliação computacional de desempenho foi melhor investigada com o objetivo de prover raciocínios (*reasoning*) em sistemas e auxiliar na descoberta de gargalos, erros e eventuais problemas antes da sua disponibilidade física. O objetivo inicial da análise de sistemas foi o de adicionar determinismo e enumerar meios de descrever realidades com transições intrincadas matematicamente, ou seja, de forma não ambígua. A ideia era aplicar os conceitos formais presentes e permitir a construção de

[1] Uma região auxiliar de memória.

modelos que representassem realidades com o maior número de detalhes possível, baseando-se nas primitivas de modelagem existentes. A seguir, tentaria-se descobrir as propriedades da estacionariedade do sistema, atestando o ponto onde ocorreu um regime de equilíbrio nas probabilidades de permanência de cada estado. Igualmente importante era conseguir realizar estas tarefas de maneira precisa, utilizando os recursos computacionais disponíveis da melhor forma possível e analisando os resultados obtidos [KLE75, SAV81, LAV83, JAI91, BOL98, STE09].

As observações iniciais indicaram que simples modelos poderiam ser analiticamente resolvidos em uma quantidade finita e razoável de tempo e mesmo assim continuariam capturando as características primordiais do sistema mesmo quando estes fossem ampliados em escala. Tratou-se de um resultado importante apesar desta ser, à primeira vista, uma maneira não-trivial de atacar o problema da modelagem computacional de sistemas pois provou que era suficientemente simples adicionar estados e transições em um dado modelo e este, mesmo assim, retornaria respostas consistentes [STE07]. Os resultados demonstrariam, por exemplo, se uma dada realidade estaria degradada e até mesmo quando começaria a desempenhar suas atividades de forma insatisfatória. Estes números, quando devidamente interpretados, indicariam a necessidade de se comprar mais máquinas ou outros componentes físicos de sistemas *antes* que recursos financeiros fossem utilizados. Estas constatações demonstraram que a análise de desempenho de sistemas é, de fato, uma maneira crucial de se evitar o desperdício de recursos pois trata da modelagem analítica em alto nível e solução computacional antes do sistema físico estar operacional [STE09].

Sistemas para extração de relações de causa e efeito podem beneficiar-se das vantagens da modelagem estocástica. Técnicas de análise baseadas em "força bruta" devem ser postas de lado e darem lugar a enfoques probabilísticos, baseados em abstrações da realidade (através da modelagem estocástica dos sistemas). O mapeamento de diferentes realidades para modelos permite a determinação precisa das interconexões das entidades envolvidas, capturando as suas nuances e respondendo questões críticas, analisadas a partir dos índices de desempenho calculados. O objetivo é, dado um contexto de aplicação, abstrair a realidade em um modelo e calcular seus índices, extraindo informações dos grandes volumes de dados e apresentando as descobertas de forma organizada.

Os passos para inspecionar sistemas envolvem as suas operações fundamentais, ou seja, captura-se a essência das tarefas realizadas pelo sistema e as trocas de estados existentes. Quando apropriadamente definidos, estes passos aumentarão as chances de se produzirem resultados relevantes. Antes de modelar uma realidade complexa é importante definir sua operação principal em um modelo restrito, ou seja, o mais simples possível, definindo quais estados este possui e como transita de estado para estado. A seguir, é realizada uma análise mais minuciosa descartando estados que não são importantes para a operação do sistema como um todo. A partir de abstrações ou *refinamentos* dos modelos é possível caracterizar a operação fundamental do sistema. Este processo está normalmente associado ao esforço dos modeladores ao representar o sistema através de modelos. Estas melhorias também são utilizadas para a otimização do sistema como um todo. Ao proporcionar a alteração de estados e transições, pode-se potencialmente descobrir uma melhor configuração para sua operação ou formas simplificadas que onerem menos recursos.

Após verificar os cenários a serem modelados, estuda-se qual formalismo pode ser usado e qual oferece a melhor maneira de descrever a realidade, dadas as suas características e as formas pelas quais evolui e interage com as suas partes constituintes (seus componentes). A escolha de um formalismo que apresente o melhor conjunto de ferramentas computacionais para solução e verificação é igualmente importante, pois não basta um formalismo ser eficaz em termos de descrição mas ineficaz ao calcular os índices de desempenho. Dando seguimento ao processo, o próximo estágio trata de buscar o equilíbrio do sistema onde este é analiticamente resolvido através de cálculos sobre os quais é possível inferir os índices de desempenho para o modelo.

Para resolver um modelo, calcula-se o vetor de probabilidades correspondente à estacionariedade do sistema, *i.e.*, supõe-se que este foi 'simulado' até um ponto onde suas mudanças não mais afetam o seu estado inicial. Pode-se dizer que o sistema chegou em um momento onde não mais evolui atingindo um equilíbrio, dadas as taxas de ocorrência definidas para as transições. Assim, extraem-se os índices computacionais de desempenho comparando-os com outras modelagens do mesmo problema, adicionando ou excluindo componentes e estados, inspecionando o reflexo das mudanças nos índices.

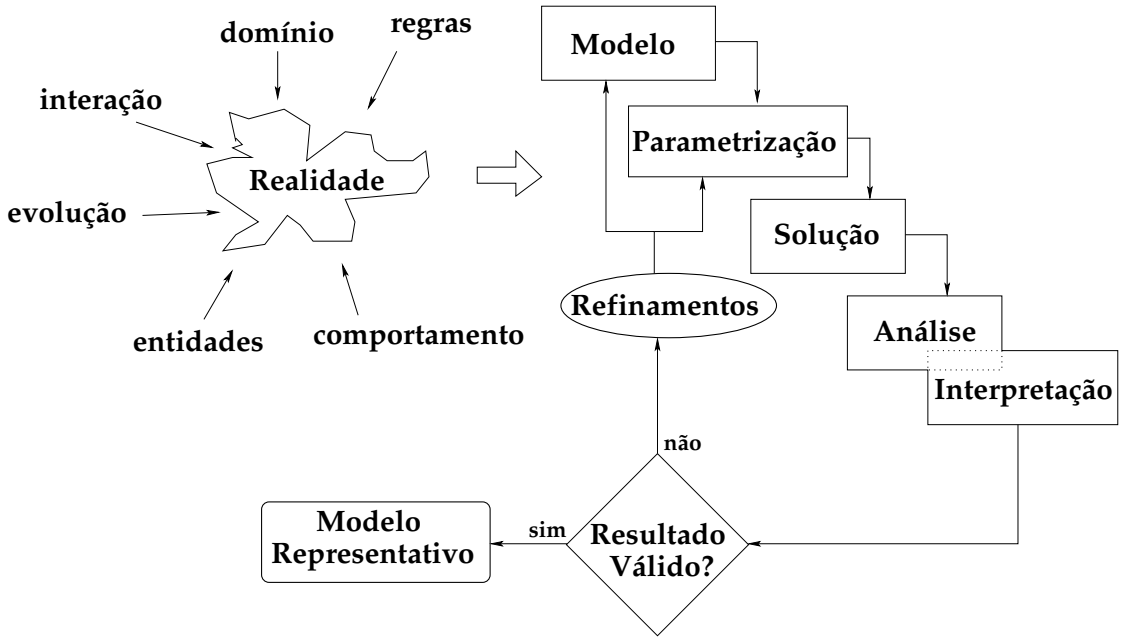


Figura 2.1: Processo de modelagem, abstração e captura de informações em um sistema.

A descrição do processo de modelagem, abstração e mapeamento de informações de sistemas é mostrada pela Figura 2.1. Uma dada realidade pode ser pensada como um conjunto de entidades dentro de um certo domínio que interagem através de um conjunto de regras. Estas entidades possuem estados e transitam entre estes com uma certa frequência com o disparo de eventos, que podem ser considerados operações sobre um estado do sistema que deslocam-se para outros estados. Após a consideração destes aspectos, inicia-se a formalização propriamente dita através da criação de um modelo com primitivas não-ambíguas de modelagem, compreendendo o processo que resultará

no cálculo e descoberta dos índices de desempenho. Ressalta-se a importância de abstrair detalhes e capturar apenas os aspectos mais relevantes do sistema sob análise [JAI91, STE09].

Estas abstrações serão determinantes para a obtenção dos índices calculados na fase de solução. Após esta fase, os índices são inspecionados e interpretados no estágio de análise, seguida por etapas de modificações do modelo, utilizando a retro-alimentação (*feedback*) obtida nos estágios anteriores.

Para determinar que uma dada modelagem é válida podem ser utilizadas, entre outros mecanismos, técnicas de simulação de eventos discretos [JAI91]. Todo o processo de modelagem computacional é iterativo, ou seja, apenas para quando o modelador determina que o modelo construído mapeou a realidade satisfatoriamente. A análise dos modelos ajuda a prevenir inconsistências e comportamentos indesejados bem como detecta condições de disparo de transições.

Tabela 2.1: Definição de formalismos e datas dos principais trabalhos.

Formalismo	Data dos primeiros trabalhos	Trabalhos relevantes para Avaliação de Desempenho
Cadeias de Markov	1906	[BOL98, NOR98, CHU04, HOW07, STE09]
Redes de Filas de Espera	1909	[ERL09, KLE75, LAZ84, GEL87, BOL98, STE09]
Redes de Petri	1962	[PET66, AJM84, CHI93a]
Redes de Petri Coloridas	1981	[JEN81, JEN96, JEN07]
Redes de Autômatos Estocásticos	1985	[PLA85, PLA91]
Redes de Atividades Estocásticas	1985	[SAN01]
Redes Bem-Formadas	1990	[CHI90, CHI93b, CHI97]
Álgebras de Processos	1970	[GOT95, BER98, BAE05]
PEPA	1994	[HIL96, HIL05a]
PEPA nets	2001	[GIL01, GIL03, HIL04]

A Tabela 2.1 lista, de forma ampla, os principais formalismos existentes na literatura e as datas dos primeiros trabalhos onde foram aplicados bem como algumas de suas extensões. A seguir, serão discutidos em um maior nível de detalhe os seguintes formalismos: Cadeias de Markov, Redes de Petri, Redes de Autômatos Estocásticos e Álgebra de Processos.

2.2 Cadeias de Markov

Cadeias de Markov (MC ou *Markov Chains*) são processos estocásticos[2] em tempo discreto ou contínuo com entidades bastante simples: estados e transições associadas. O formalismo foi proposto com um interesse particular, baseado em textos literários [HOW07]. Esta aplicação não usual demonstra a versatilidade do formalismo, podendo ser utilizado para mapear inúmeras realidades [STE94, NOR98, STE09].

O objetivo da aplicação original de MC foi, dado um texto e uma determinada posição, inferir a probabilidade de uma próxima letra ser uma vogal dado que a letra atual é uma consoante [HOW07].

[2]Um processo estocástico ou processo randômico é o oposto de um processo determinístico. Ao invés de se lidar com apenas uma realidade possível de como um processo evolui no tempo, em um processo estocástico existe uma qualidade indeterminada da sua evolução e é descrita através de distribuições de probabilidades [BHA97, BOL98, HOW07].

Trata-se de uma propriedade primordial de MC, a chamada *memoryless property*. Esta propriedade estabelece que toda a informação relevante está contida no estado atual, não interessando os estados anteriores. MC é utilizado devido à sua simplicidade e ao fato de possibilitar inferências de raciocínios e inspeções de desempenho para realidades variadas através de primitivas básicas de modelagem.

As MC permaneceram desconhecidas até serem utilizadas no contexto de avaliação de desempenho de sistemas no MIT (*Massachussets Institute of Technology*). A aplicação em questão foi direcionada a sistemas de compartilhamento de tempo (*time sharing systems*) [STE07]. A principal observação realizada nesse trabalho foi a de que, mesmo o modelo sendo extremamente simples, ainda conseguia capturar a essência de sua operação e prover respostas mesmo quando adaptado ou escalado para refletir outros comportamentos.

Hoje em dia, as MC são utilizadas em inúmeros contextos tais como simulação [PRO96, HAG02] ou análises de riscos no mercado de ações, para citar algumas aplicações. Um outro exemplo da utilidade deste formalismo são os sistemas de indexação e procura da Internet, onde o conjunto de informações existentes é amplamente desorganizado e de proporções massivas. A dificuldade desta classe de problema é a de encontrar páginas com conteúdo relevante para o que se esteja procurando, às vezes, observando contexto, ou seja, também retornando outras páginas com conteúdo similar. O principal conceito pode ser modelado como cada página na Internet sendo um estado e construir uma abstração onde considera-se uma entidade que 'salta' de página em página, visitando-as de forma aleatória e simulando uma caminhada randômica (*random walking*). O processo de visitação das páginas é realizada de forma aleatória onde, a partir de uma página inicial, deseja-se visitar as outras páginas conectadas à esta (em terminologia da Internet, cada página possui diversos *links*, ou seja, transições para outras páginas).

Ao modelar este problema como sendo uma MC e resolver o sistema de equações que o representa onde as variáveis são as páginas, o resultado prático é a probabilidade (a relevância) de páginas similares. Ao descobrir e ordenar estas probabilidades de ocorrência em ordem decrescente (as com maiores probabilidades primeiro), são retornadas as páginas mais prováveis para o usuário. Esse conceito é utilizado pelo sistema de buscas da empresa *Google* e o processo neste contexto é descrito de forma superficial [LAN06][3].

No contexto de MC, é possível representar um sistema usando tempo contínuo (também chamadas de Cadeias de Markov em escala de tempo contínuo – *Continuous Time Markov Chains*, ou CTMC) ou tempo discreto (Cadeias de Markov em escala de tempo discreto – *Discrete Time Markov Chains*, ou DTMC). A Figura 2.2 mostra um exemplo de Cadeia de Markov em escala de tempo contínuo com $N = 9$ estados e uma taxa para cada transição, definidas como λ, μ ou ν, dependendo do estado observado.

Os índices de desempenho são computados da distribuição estacionária ou transiente da Cadeia de Markov, resolvendo-se as equações de balanço global do sistema através da ferramenta

[3]Esta descrição encontra-se bastante simplificada neste trabalho, pois são utilizados algoritmos complexos de mapeamento, indexação e associação de *ranks* a páginas, normalmente através de implementações proprietárias inacessíveis à comunidade externa por questões mercadológicas.

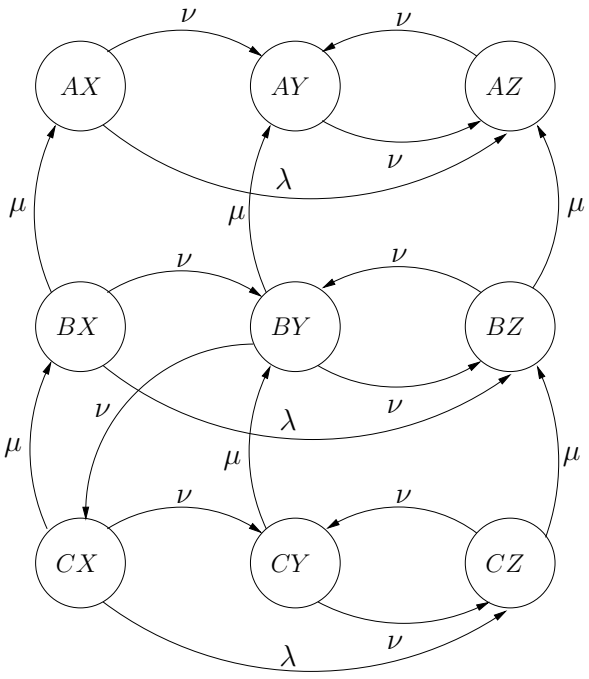


Figura 2.2: Um exemplo de Cadeia de Markov com nove estados.

MARCA [STE91] (*Markov Chain Analyzer*), entre outras. As principais distinções versam sobre a duração deste processo a partir de um determinado tempo t sobre o tempo total T. São chamadas de probabilidades *estacionárias* quando atingiram o equilíbrio em T e *transientes* quando interrompe-se o processo em t e analisam-se os índices de desempenho [STE94].

Estas variáveis de entrada são extraídas da matriz de transição P da MC, onde a probabilidade p_{ij} significa a probabilidade condicional para sair do estado i e ir para o estado j, onde $i \neq j$. Para o caso da solução estacionária do sistema, a equação pode ser escrita para DTMC como: $\pi P = \pi$, ou para o caso de CTMC como: $\pi \tilde{\mathcal{Q}} = 0$, onde $\tilde{\mathcal{Q}} = \Delta T + I$ e $\Delta T \leq (max_i(|p_{ij}|))^{-1}$. Esta operação irá converter a representação em tempo contínuo representado pelas *taxas de transição* da matriz $\tilde{\mathcal{Q}}$ para uma representação baseada em tempo discreto com as *probabilidades de transição* da matriz P. Neste sistema discreto, as transições acontecem em intervalos de tempo ΔT. Este parâmetro é normalmente escolhido para que a probabilidade existente *entre* duas transições seja negligenciável [STE94, STE07].

Logo, uma MC pode ser vista como um sistema de transição de estados onde o modelador define quais estados são acessíveis a partir de outros estados e configura uma taxa ou probabilidade das transições do sistema. Existem diversos métodos para resolver o sistema linear de equações, tais como métodos diretos (eliminação Gaussiana, decomposição LU), iterativos (Método da Potência, Jacobi ou Gauss-Seidel), de bloco (Gauss-Seidel de bloco) ou de projeção (como *Generalized Minimal Residual Algorithm* – GMRES), para citar alguns exemplos [STE94, SAA95].

Métodos de solução direta deste sistema de equações tais como eliminação Gaussiana não são eficientemente aplicáveis em modelos com número de estados elevado, pois requerem igualmente elevadas quantidades de memória para armazenar a matriz de transição. Já técnicas iterativas de

solução são melhor aproveitadas, principalmente as que armazenam a matriz $\tilde{\mathcal{Q}}$ de forma esparsa, entretanto, existem limites quanto aos modelos que podem ser resolvidos desta forma. O resultado da multiplicação de uma matriz por um vetor π traduz a probabilidade de estar em um estado em um certo tempo, depois que o sistema está operacional [STE94].

A Figura 2.3 mostra como é gerada a matriz $\tilde{\mathcal{Q}}$ com as taxas de transição da MC do exemplo da Figura 2.2. Para este caso, cada estado da cadeia leva a um ou mais estados de acordo com uma taxa (μ, ν ou λ). Para transformar essa matriz em um *Gerador Infinitesimal* correspondente à matriz $\tilde{\mathcal{Q}}$ é necessário que a soma de cada linha resulte em zero. Para tanto, precisa-se corrigir a diagonal principal para refletir essa exigência.

	AX	AY	AZ	BX	BY	BZ	CX	CY	CZ
AX	0	ν	λ	0	0	0	0	0	0
AY	0	0	ν	0	0	0	0	0	0
AZ	0	ν	0	0	0	μ	0	0	0
BX	μ	0	0	0	ν	λ	0	0	0
BY	0	μ	0	0	0	ν	ν	0	0
BZ	0	0	μ	0	ν	0	0	0	0
CX	0	0	0	μ	0	0	0	ν	λ
CY	0	0	0	0	μ	0	0	0	ν
CZ	0	0	0	0	0	μ	0	ν	0

Figura 2.3: Processo de construção da matriz das taxas de transição da MC.

A seguir é mostrada esta correção, preparando a matriz para posterior multiplicação por um vetor de probabilidades:

$$\left(\begin{array}{ccc|ccc|ccc}
-(\nu+\lambda) & \nu & \lambda & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\nu & \nu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \nu & -(\nu+\mu) & 0 & 0 & \mu & 0 & 0 & 0 \\
\hline
\mu & 0 & 0 & -(\mu+\nu+\lambda) & \nu & \lambda & 0 & 0 & 0 \\
0 & \mu & 0 & 0 & -(\mu+2\nu) & \nu & \nu & 0 & 0 \\
0 & 0 & \mu & 0 & \nu & -(\mu+\nu) & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & \mu & 0 & 0 & -(\mu+\nu+\lambda) & \nu & \lambda \\
0 & 0 & 0 & 0 & \mu & 0 & 0 & -(\mu+\nu) & \nu \\
0 & 0 & 0 & 0 & 0 & \mu & 0 & \nu & -(\mu+\nu)
\end{array}\right)$$

Para a CTMC desta matriz, calcula-se $\pi\tilde{\mathcal{Q}} = 0$ onde π é um vetor de probabilidades inicial e resolve-se o sistema de equações lineares. O resultado informa as probabilidades de permanência de cada estado do sistema. A partir destes resultados, extraem-se os índices de desempenho. Um critério de parada utilizado nos métodos iterativos clássicos é verificar que a variação entre duas iterações elemento a elemento seja menor que $1E^{-10}$. Supondo que sejam escolhidos os valores $\mu = 1, \nu = 2, \lambda = 5$, o seguinte produto vetor-matriz ($\pi\tilde{\mathcal{Q}} = 0$) corresponde à:

$$\overbrace{(\pi_0 \quad \pi_1 \quad \pi_2 \quad \pi_3 \quad \pi_4 \quad \pi_5 \quad \pi_6 \quad \pi_7 \quad \pi_8)}^{\text{vetor de probabilidades } \pi} \times \overbrace{\left(\begin{array}{ccc|ccc|ccc} -7 & 2 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -3 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & -8 & 2 & 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -5 & 2 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & -3 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & -8 & 2 & 5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & -3 \end{array}\right)}^{\text{Gerador Infinitesimal}} = \left(\begin{array}{c} 0 \ldots 0 \end{array} \right)$$

Uma alternativa é utilizar métodos diretos, definindo o sistema linear de equações com nove incógnitas ($\pi_{0..8}$), onde busca-se encontrar o conjunto de valores que satisfaçam:

$$\begin{cases} -7\pi_0 + \pi_3 = 0 \\ 2\pi_0 - 2\pi_1 + 2\pi_2 + \pi_4 = 0 \\ 5\pi_0 + 2\pi_1 - 3\pi_2 + \pi_5 = 0 \\ -8\pi_3 + \pi_6 = 0 \\ 2\pi_3 - 5\pi_4 + 2\pi_5 + \pi_7 = 0 \\ \pi_2 + 5\pi_3 + 2\pi_4 - 3\pi_5 + \pi_8 = 0 \\ 2\pi_4 - 8\pi_6 = 0 \\ 2\pi_6 - 3\pi_7 + 2\pi_8 = 0 \\ 5\pi_6 + 2\pi_7 - 3\pi_8 = 0 \\ \pi_0 + \pi_1 + \pi_2 + \pi_3 + \pi_4 + \pi_5 + \pi_6 + \pi_7 + \pi_8 = 1 \end{cases}$$

Outro meio de solução é utilizar, por exemplo, o *Método da Potência* [STE09]. Ao utilizar esta técnica, para este modelo e para os valores de taxas de transição escolhidos, são necessárias 118 iterações para obtenção da solução estacionária, resultando em valores da probabilidade da permanência em cada estado. Por exemplo, a Tabela 2.2 mostra que o estado AY tem $30,52\%$ de probabilidade de permanência enquanto que o estado AZ tem $26,25\%$. Caso os estados desta MC significassem, por exemplo, processadores e a métrica observada fosse processamento, os de 'rótulo' AY e AZ estariam ocupados $\approx 56\%$ do tempo na operação desse sistema. A tabela também mostra, na coluna π, o índice do vetor de probabilidades para cada estado global do sistema (total de estados do sistema é $N = 9$). É igualmente possível modificar as taxas do modelo trocando-as para os seguintes valores: $\mu = 1, \nu = 5, \lambda = 8$. Estas mudanças alteram a convergência do método fazendo com que sejam necessários 186 passos. Os resultados obtidos são mostrados na parte direita da Tabela 2.2, que mostra que as maiores probabilidades de permanência estão concentradas nos estados AY, AZ, CY e CZ.

O presente trabalho não discutirá todos os detalhes pertencentes a Cadeias de Markov. Para maiores informações, existem diversos outros trabalhos na literatura sobre MC [STE94, BOL98, BUC98, NOR98, STE01, HAG02, STE07, STE09].

Tabela 2.2: Análise estacionária e refinamento da MC resultando no vetor de probabilidades.

		$\mu = 1, \nu = 2, \lambda = 5$		$\mu = 1, \nu = 5, \lambda = 8$	
π_i	Rótulo do Estado i	Índice (I)	Probabilidade ($I \times 100$)	Índice (I)	Probabilidade ($I \times 100$)
π_0	AX	0,00037804325	0,04 %	0,00014600307	0,01 %
π_1	AY	0,30523211288	30,52 %	0,21929713481	21,93 %
π_2	AZ	0,26251322707	26,25 %	0,20427049965	20,43 %
π_3	BX	0,00264630278	0,26 %	0,00189803999	0,19 %
π_4	BY	0,08468168888	8,47 %	0,07440316752	7,44 %
π_5	BZ	0,17518524215	17,52 %	0,12796930560	12,80 %
π_6	CX	0,02117042225	2,12 %	0,02657255985	2,66 %
π_7	CY	0,06774535366	6,77 %	0,16909811384	16,91 %
π_8	CZ	0,08044760705	8,04 %	0,17634517563	17,63 %
Total		$\sum_{i=0}^{N-1} \pi_i = 1,0$	100%	$\sum_{i=0}^{N-1} \pi_i = 1,0$	100%

Como mencionado anteriormente, a modelagem mapeia os estados que um sistema adota, constituindo o chamado *Espaço de Estados* da realidade. Entretanto, quando são necessários muitos estados para compor um determinado sistema, diz-se que ocorre uma *explosão do espaço de estados*. Dependendo do número de estados, não é possível resolver o sistema de equações em um tempo razoável por falta de recursos computacionais tais como memória ou poder de processamento. Esta limitação de modelagem e consequentemente de solução numérica foi a responsável pela pesquisa de mapeamentos mais estruturados das cadeias, como será abordado a seguir.

2.2.1 Emergência dos formalismos estruturados

Apesar de amplamente utilizadas, as MC possuem problemas fundamentais, sendo o principal quando um modelo possui um elevado número de estados tornando-o de difícil análise. Esta *explosão do espaço de estados* também torna impraticável a solução do sistema de equações via métodos diretos e, às vezes, até mesmo métodos iterativos em um tempo razoável. Este problema foi constatado desde a definição inicial de MC, onde verificou-se que mesmo simples descrições criavam muitos estados gerando desafios relacionados tanto às questões de armazenamento da matriz de transição quanto aos mecanismos de solução envolvidos.

Para reduzir os efeitos problemáticos da explosão do espaço de estados foram consideradas outras formas de representação dos sistemas, mas permanecendo com a visão das Cadeias de Markov como alternativa válida de modelagem. A alternativa encontrada foi a definição de *formalismos estruturados*, usados para representar um sistema em um nível superior de abstração, através de uma Cadeia de Markov correspondente de forma implícita ou subjacente. Este novo tipo de visão criou novas definições, tais como modelar partes do sistema como entidades ou componentes autônomos mas que eventualmente sincronizavam atividades. Ao efetuar o produto cartesiano dos espaços de

estados de cada subsistema, ou seja, ao combinar todos os estados passíveis de serem compostos dos componentes de um dado formalismo estruturado, por exemplo, autômatos em Redes de Autômatos Estocásticos (Seção 2.4) ou processos em PEPA (Seção 2.5.1), trabalha-se, na verdade, com uma Cadeia de Markov correspondente. Este produto cartesiano também é chamado de Espaço de Estados Produto $\mathcal{X}$ (*Product State Space*, ou PSS) e dita a quantidade de estados total no sistema.

Cabe ressaltar que os formalismos estruturados reduzem significativamente o problema da explosão do espaço de estados, mas não o eliminam completamente. Alguns formalismos estruturados apresentam também a problemática relacionada à existência de estados inatingíveis, pois, ao combinar os estados entre si e usar as transições definidas no modelo, são gerados estados que não podem ser alcançados dependendo do estado inicial escolhido. Este problema é melhor evidenciado na modelagem Markoviana, bastando verificar a existência de estados sem saídas (estados *absorventes*) ou chegadas (estados *inatingíveis*). Dependendo do caso, o Espaço de Estados Atingível $\mathcal{X}^R$ (*Reachable State Space*, ou RSS) é um subconjunto do PSS, *i.e.*, $\mathcal{X}^R \subseteq \mathcal{X}$ [MIN99b, BUC02a], entretanto, ainda permanece um problema em aberto a solução eficiente de modelos complexos onde todos, ou quase todos, os estados são atingíveis. Entretanto, estes problemas podem ser mitigados utilizando-se estruturas de dados sofisticadas de manipulação simbólica tais como Diagramas de Decisão Multi-valorados (*Multivalued Decision Diagrams* ou MDD) [MIN99b] ou através do uso de vetores reduzidos [BEN06].

Na prática, representações estruturadas da matriz $\tilde{\mathcal{Q}}$ podem ser obtidas através de diferentes formalismos de modelagem: Redes de Petri Estocásticas Generalizadas Superpostas [DON93, DON94], Redes de Autômatos Estocásticos [PLA85, PLA91] ou até mesmo descrições modulares e composicionais como PEPA [HIL96]. O princípio geral de utilizar tensores para representação implícita desta matriz existe desde os primórdios da definição das Redes de Autômatos Estocásticos, mas recentemente observou-se a aplicação com êxito em outros formalismos estocásticos [DON93, HIL01, HIL07]. As vantagens e desvantagens ao se adotar um formalismo variam de caso para caso e é necessário saber previamente as funcionalidades presentes em cada definição, maximizando-se, assim, as análises que são permitidas. O objetivo do restante deste capítulo é introduzir outras importantes descrições para modelagem de sistemas. O próximo formalismo a ser descrito refere-se às Redes de Petri, seguido por Redes de Autômatos Estocásticos e Álgebra de Processos.

2.3 Redes de Petri

Redes de Petri, (PN ou *Petri Nets*) [BAL07] são representações gráficas de modelagem para descrição de sistemas através do uso de anotações. Estruturas comuns de Redes de Petri são nodos que correspondem a lugares (*place nodes*), nodos que correspondem a transições (*transition nodes*) e arcos dirigidos que representam conexões entre lugares e transições. Dentro de cada lugar, existem marcas (*tokens*) e a distribuição de tais marcas dentro de uma rede é conhecida como uma marcação (*marking*). A maior vantagem da adoção de PN para modelagem está no fato de possibilitar uma visão clara de causalidade e conflito nas regiões que compõem a rede, devido à maneira utilizada para

representar o sistema. Uma extensão de PN são as chamadas Redes de Petri Estocásticas (SPN ou *Stochastic Petri Nets*) [FLO85], definidas pela introdução de tempo de disparo não-determinístico entre transições.

A Figura 2.4 mostra um exemplo de uma PN. A rede descrita pela figura contém quatro lugares (P_1, P_2, P_3, P_4) e três transições (T_1, T_2, T_3) mostrando como as marcas podem se movimentar de um lugar ao outro dentro da rede.

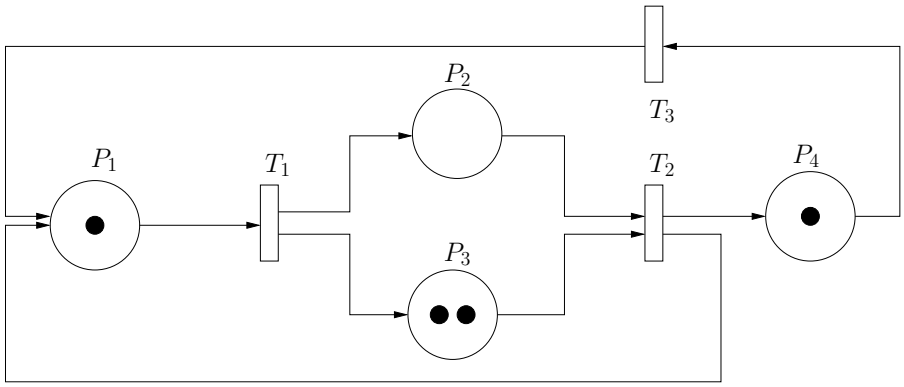


Figura 2.4: PN com lugares (P_1, P_2, P_3, P_4), transições (T_1, T_2, T_3) e marcas (círculos pretos).

Uma importante vantagem de uma PN é sua característica ao descrever como funciona o fluxo de informações na rede e como operam as estruturas de controle e restrição de movimentação dos tokens entre os lugares. A rede possui uma estrutura e um sistema de marcação. A estrutura representa a parte estática do sistema. Os dois tipos de nodos são representados por componentes gráficos, círculos para os lugares e retângulos para as transições. Os lugares correspondem aos estados do sistema e as transições a ações que forçam a mudança entre os estados. Os arcos conectam lugares a transições e transições a lugares, sendo chamados, respectivamente, de *arcos de entrada* e *arcos de saída*.

Cabe ressaltar que, com o tempo, as SPN foram estendidas para modelar apenas algumas transições de um sistema com tempo de disparo não-determinístico. Essa extensão foi chamada de Redes de Petri Estocásticas Generalizadas (GSPN ou *Generalized Stochastic Petri Net*) [CHI93a, AJM95,BAL07] e permite a modelagem de realidades que possuem atividades de durações variáveis e que podem trocar de estado muitas vezes. Uma GSPN permite duas classes diferentes de transições: imediatas e temporizadas. Uma transição imediata é disparada imediatamente (em tempo zero) assim que estiver habilitada e uma transição temporizada é disparada após um tempo aleatório regido por uma distribuição exponencial associada. A estrutura de uma GSPN segue a definição de uma SPN, porém o conjunto das taxas de disparo contém somente os elementos das transições temporizadas.

Entretanto, observou-se a necessidade de se compartimentalizar (ou modularizar) os modelos definidos com PN, onde existem interações entre esses componentes. As Redes de Petri Estocásticas Generalizadas Superpostas (SGSPN ou *Superposed Generalized Stochastic Petri Net*) [DON93, DON94] foram definidas permitindo a interação entre os componentes através da superposição de

transições. A solução de modelos desta classe de redes pode ser tratada pois deriva uma expressão tensorial similar à adotada pelas Redes de Autômatos Estocásticos (maiores detalhes na Seção 2.4, a seguir, ou no Capítulo 4) para o seu Gerador Infinitesimal. A modelagem em SGSPN é feita através do uso de módulos com transições locais e que também interagem através de sincronizações. As fórmulas tensoriais (melhor explicadas na Seção 4.3) são bastante similares às obtidas através das manipulações tensoriais presentes em Redes de Autômatos Estocásticos.

Um modelo descrito em SGSPN é mostrado na Figura 2.5. Este modelo possui dois componentes chamados $C^{(1)}$ e $C^{(2)}$ (mostrado na figura com uma separação "abstrata" conforme as linhas pontilhadas com os elementos à esquerda e à direita), com sete lugares ($p_{1..7}$) e cinco transições ($t_{1..5}$) sendo que os dois componentes estão sendo sincronizados pelo evento t_3. Esta rede possui duas marcas, uma em p_1 e a outra em p_6.

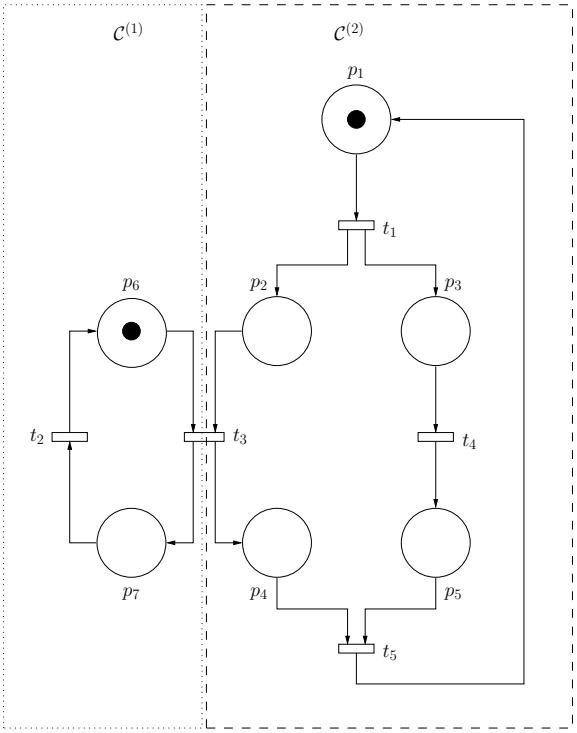


Figura 2.5: Uma SGSPN mostrando seus componentes e sincronizações.

A *marcação* de uma Rede de Petri é a associação de *marcas* a *lugares*. Para uma definição formal de uma PN é necessário especificar a estrutura da rede e a marcação inicial, *i.e.*, onde as marcas serão inicialmente colocadas dentro da rede. A dinâmica do sistema segue um conjunto básico de regras sendo que uma transição ocorre quando o lugar de entrada cumpre as condições expressas pelas inscrições dos arcos [BAL07].

O objetivo deste trabalho não é especificar formalmente PN, apenas oferecer uma breve descrição deste formalismo. A solução de PN pode ser realizada através do uso da ferramenta *SMART* [CIA01]. Para maiores informações sobre PN e suas extensões indica-se [PET77, MOL82, AJM84, MUR89, BAL07], que mostram o funcionamento e explicam PN com um maior nível de detalhe.

2.3.1 Redes de Petri Coloridas

PN possuem extensões para capturar outros tipos de comportamento de sistemas. Uma extensão bastante conhecida são as Redes de Petri Coloridas (CPN ou *Coloured Petri Nets*). A principal distinção entre uma Rede de Petri tradicional e uma Rede de Petri Colorida reside nas informações contidas nas *marcas*. Em PN elas são indistinguíveis (possuem uma mesma cor), enquanto que em CPN, cada marca pode possuir uma característica distinta associada tal como, por exemplo, uma cor.

Uma CPN é um formalismo estruturado usado em realidades onde sincronizações e comunicações são necessárias, combinando as vantagens de Redes de Petri com conceitos de linguagens de programação de alto nível. CPN utilizam as ideias que descrevem como as entidades cooperam entre si e proveem mecanismos para manipulação de estruturas de dados e associação a variáveis, ou seja, conceitos de linguagens de programação. Tais redes, como PN tradicionais, possuem lugares, transições e arcos. Adicionalmente, possuem um outro componente chamado páginas, que podem potencialmente conter lugares. O conjunto destes elementos formam uma rede que, quando usados em conjunto e mostrados graficamente, permitem a manipulação de estruturas complexas fornecendo auxílios visuais para reconhecer como as entidades estão conectadas entre si e como interagem.

CPN podem ser usadas para verificação formal, pois oferecem um conjunto de ferramentas desenvolvidas para este propósito específico. Existem dois tipos de métodos de verificação em CPN: análise do espaço de estados e análise de invariantes. O objetivo da verificação formal é provar matematicamente que um sistema possui um conjunto de propriedades comportamentais. À medida que sistemas atingem uma escala industrial, também torna-se difícil sua análise. As CPN podem ser usadas para estes propósitos, aliadas a outros mecanismos de validação utilizando-se simulação [JEN07].

Dentre as vantagens de se adotar CPN para modelar sistemas, destaca-se: primitivas para estabelecer e realizar inferências sobre a interação de componentes dentro de um sistema, modelagem de sub-componentes (também conhecida por modelagem hierárquica), descrição gráfica de sistemas e ferramentas para solução do sistema em questão (através de um *software* chamado *CPN Tools* [WEL06]). Para outras informações sobre CPN, sugere-se os seguintes trabalhos [JEN81, JEN96] e mais recentemente [JEN07].

2.4 Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos (SAN ou *Stochastic Automata Networks*) [PLA85, PLA91] é um formalismo estruturado utilizado para o cálculo de índices de desempenho na área da avaliação quantitativa de sistemas. Este formalismo baseia-se em entidades semi-autônomas chamadas autômatos e como estes componentes se relacionam para sincronizar transições ou trocar de estados. O conjunto destes autômatos forma uma rede, onde é permitido que sejam analisadas diferentes realidades. O formalismo foi originalmente construído para visar o mapeamento de atividades de

paralelismo e distribuição onde elementos tem comportamentos independentes mas eventualmente necessitam sincronizar tarefas entre si. Em SAN, descrevem-se os sistemas de maneira estruturada, implicitamente gerando-se a MC correspondente. Essa estruturação implica no armazenamento eficiente das estruturas de dados, facilitando a representação de sistemas com múltiplas transições e produzindo índices de desempenho para posterior análise e interpretação.

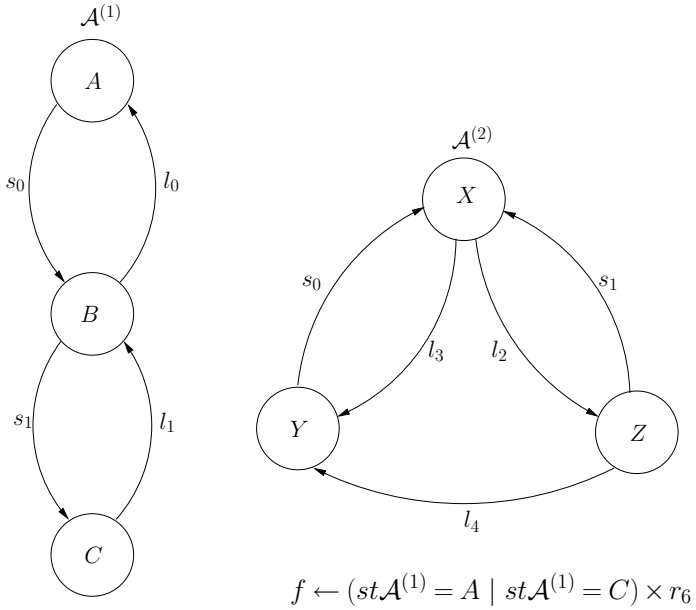
Mais especificamente, SAN é um formalismo estruturado que descreve interações entre entidades conhecidas por *autômatos*. Cada autômato apresenta transições específicas na sua estrutura, trocando de estados através de *transições* que, por sua vez, são regidas por um ou mais *eventos*. Um autômato altera seu comportamento local através de eventos locais ou interage com outros autômatos através de eventos sincronizantes. Um evento sincronizante envolve, no mínimo, dois autômatos e representa uma mudança de estado que acontece de forma concomitante. Um evento é disparado de acordo com a definição de uma taxa de ocorrência ou taxa de transição. A taxa de ocorrência de um determinado evento é classificada de duas maneiras distintas: constante ou funcional. Uma taxa constante é uma frequência média com a qual a transição ocorre. Uma taxa funcional tem seu valor médio determinado segundo o estado de outros autômatos, de acordo com fórmulas de consulta a estados pré-estabelecidas pelos usuários quando modelam a abstração do sistema. A vantagem em se utilizar taxas funcionais consiste, entre diversos fatores, em simplificar a modelagem e, dependendo do caso, reduzir o número de eventos ou, até mesmo de autômatos, para representar uma realidade [STE94, BEN04b].

A Figura 2.6 mostra um exemplo de uma SAN simples com eventos locais (*loc*) e sincronizantes (*syn*), contendo taxas constantes e funcionais. A figura mostra uma rede composta por dois autômatos, cooperando sobre um espaço de estados produto de tamanho $|\mathcal{X}| = 3 \times 3 = 9$. O exemplo define cinco eventos locais (chamados l_0, l_1, l_2, l_3, l_4) e dois eventos sincronizantes (s_0, s_1). Uma das taxas é funcional (f) sendo as demais taxas constantes. A taxa funcional é avaliada para r_6 quando o estado do autômato $\mathcal{A}^{(1)}$ for igual a A ou igual a C, ou avaliada para $zero$ (*i.e.*, não ocorrendo), caso contrário (nesse caso, igual a B)[4].

Um modelo em SAN é transformado em um *descritor Markoviano* (uma representação tensorial para o Gerador Infinitesimal $\tilde{\mathcal{Q}}$, também chamado de *descritor Kronecker*) [PLA85, STE94, FER98b], *i.e.*, um conjunto de matrizes responsáveis por definir a ocorrência dos eventos locais e sincronizantes de forma tensorial, melhor descritas na Seção 4.3. Após a definição do descritor, é utilizado um método especializado para multiplicá-lo por um vetor de probabilidade, ou seja, efetuar a Multiplicação Vetor-Descritor (MVD, maiores informações na Seção 4.4).

Apesar das suas diferenças algorítmicas, o processo de multiplicar um vetor por um descritor pode ser resumido em encontrar uma forma adequada de multiplicar uma estrutura usualmente grande (o vetor) por outra reduzida mas com estruturação complexa (o descritor Markoviano). Algumas soluções para SPN [AJM84] traduzem a representação dos modelos em uma única matriz esparsa. Naturalmente, esta proposta esparsa é difícil de ser utilizada para modelos com muitos

[4] O símbolo '|' na função f representa um teste lógico "OU" e o símbolo '$=$' representa um teste lógico que retorna verdadeiro caso o estado seja igual à parte da direita da fórmula.



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

Figura 2.6: Um exemplo de SAN com eventos locais/sincronizantes, e taxas constantes/funcionais.

estados (*e.g.*mais de 500 mil estados), pois normalmente necessita armazenar esta matriz contendo muitas vezes diversos elementos não-nulos (*e.g.* mais de 4 milhões). Isto é possível apenas com soluções sofisticadas para armazenar a matriz usando práticas baseadas em disco (*disk-based approaches*) [DEA98a], ou técnicas de geração *on-the-fly* [DEA98b] aliadas à computação paralela e distribuída [DON93, TAD99, BAL04].

Mais genericamente, os modelos de SAN são convertidos para descritores Markovianos e resolvidos através da ferramenta PEPS (*Performance Evaluation of Parallel Systems*) [BRE07]. Novos desenvolvimentos estão atualmente direcionados a três eixos de pesquisa: utilização de representação simbólica do RSS (usando-se estruturas mais complexas como MDD [CIA97]), aceleração da MVD com o Algoritmo *Split* [CZE07, CZE09] usando-se a ferramenta GTAexpress e técnicas de Simulação Perfeita [FER08]. Maiores informações sobre métodos de solução de MVD encontram-se no Capítulo 4 e sobre a GTAexpress no Capítulo 6.

2.5 Álgebras de Processos

Com o passar dos anos constatou-se um crescente aumento na adoção de Álgebras de Processos Markovianas (MPA ou *Markovian Process Algebras*) voltadas para a construção de modelos que representem sistemas complexos. Exemplos incluem *Performance Evaluation Process Algebra* [HIL96] (PEPA) e *Extended Markovian Process Algebra* [BER98] (EMPA), ou seja, formalismos desenvolvidos para capturar o comportamento de sistemas e permitir análise funcional através da solução da CTMC associada. Uma das principais atrações para o uso de álgebras de processos é a habilidade de se construir um modelo composicional baseado em uma definição em alto-nível. Tais álgebras permitem que sejam usadas em diversas aplicações, onde são facilmente modeladas através de sim-

ples primitivas que mostram como o sistema evolui de acordo com seu comportamento interno e de acordo com as interações existentes entre os componentes modelados.

2.5.1 PEPA – *Performance Evaluation Process Algebra*

Como em todas as álgebras de processos, sistemas são representados em PEPA como a composição de componentes que realizam ações. Em PEPA, as ações ocorrem conforme uma duração, logo, a expressão $(\alpha, r).P$ denota um componente que realizou uma ação α com taxa r e evoluiu para um componente P. Aqui $\alpha \in \mathcal{A}$, onde $\mathcal{A}$ é o conjunto de tipos de ações e $P \in \mathcal{C}$, onde $\mathcal{C}$ é o conjunto de tipos de componentes.

PEPA possui um conjunto restrito de "combinadores", e as descrições dos sistemas são construídas a partir da execução concorrente e interação simples de componentes sequenciais. A sintaxe destes combinadores está informalmente descrita abaixo, entretanto, maiores detalhes são encontrados em [HIL96, HIL05b].

- *Prefix*: o combinador *prefix* marca uma parada completa, dando ao componente a sua primeira ação designada. Como explicado anteriormente, $(\alpha, r).P$ executa uma ação α com taxa r, e subsequentemente começa a se comportar como P;

- *Choice*: o componente $P + Q$ representa um sistema que pode se comportar tanto como P ou como Q. As atividades de ambos P e Q estão autorizadas (*enabled*). A primeira atividade a acabar distingue-se (ou seja, é autorizada), a outra é descartada;

- *Constant*: é conveniente associar nomes a padrões comportamentais de componentes e constantes realizam essa tarefa definindo uma equação, *i.e.*, $X \stackrel{def}{=} E$. O nome X está no escopo da expressão do lado direito, significando que, por exemplo, $X \stackrel{def}{=} (\alpha, r).X$ desempenha α com taxa r para sempre;

- *Hiding*: a possibilidade de abstrair alguns aspectos do comportamento de componentes é realizado pelo operador *hiding*, denotado por P/L. O conjunto L identifica aquelas atividades as quais são consideradas internas ou privadas ao componente e quais irão aparecer como o tipo desconhecido chamado τ;

- *Cooperation*: escreve-se $P \underset{L}{\bowtie} Q$ para descrever cooperação entre P e Q sobre L. O conjunto L é o conjunto de cooperação e determina as atividades com as quais os componentes devem sincronizar atividades. Para tipos de ação fora de L, os componentes procedem independentemente e concorrentemente com suas atividades autorizadas. Escreve-se também $P \parallel Q$ como uma abreviação para $P \underset{L}{\bowtie} Q$ quando o conjunto L está vazio.

As atividades do conjunto de cooperação definido somente realizam a atividade quando ambas estiverem autorizadas. Os dois componentes então procedem juntos para completar esta atividade compartilhada. A taxa da atividade compartilhada pode ser alterada para refletir o trabalho realizado

por ambos os componentes envolvidos com a finalidade de acabar a atividade. A capacidade total do componente C para realizar tarefas do tipo α é definida pela taxa aparente de α em P, denotada por $r_\alpha(P)$. PEPA assume *bounded capacity*, ou seja, a taxa de uma atividade compartilhada é igual ao valor mínimo entre as taxas das atividades dos componentes que estão cooperando.

Em alguns casos a taxa de uma atividade é deixada sem especificação (denotada por $\top$) e é determinada pela taxa da atividade do outro componente quando existe uma cooperação. Estas ações passivas devem ser sincronizadas no modelo final (forçando a ocorrência de uma taxa para a cooperação).

A sintaxe de PEPA pode ser formalmente introduzida a partir da seguinte gramática:

$$\begin{aligned} S &::= (\alpha, r).S \mid S + S \mid C_S \\ P &::= P \underset{L}{\bowtie} P \mid P/L \mid C \end{aligned}$$

onde S denota um componente sequencial e P denota um componente de modelo o qual é executado em paralelo. C é uma constante que denota tanto um componente sequencial quando um componente de modelo. C_S são constantes que denotam componentes sequenciais. O efeito desta separação sintática entre estes tipos de constantes serve para restringir componentes PEPA para configurarem cooperações de processos sequenciais, uma condição necessária para existência de processos Markovianos bem definidos[5].

O significado de uma expressão PEPA é dado pela semântica operacional estrutural que define as formas pelas quais ocorrem a evolução de um determinado modelo, podendo ser aplicado exaustivamente para formar um sistema de rótulos denominado Grafo de Derivação (GD) representando em última análise o espaço de estados do modelo (somente os atingíveis). Trata-se de um grafo o qual cada nodo é uma forma sintática distinta, definida como derivativo (*derivative*) e cada arco representa uma atividade possível causando uma mudança de estado. O estado muda em conjunção com a Matriz de Transição (MT), que guarda o nome do evento e a sua taxa de ocorrência para cada dois estados.

É importante notar que em PEPA a representação de estados é de fato um sistema de rótulos com multi-transições, pois guarda a multiplicidade de arcos, particularmente quando componentes repetidos estão envolvidos. O GD pode ser considerado como um diagrama de transição de estados de uma CTMC; $q(P,Q)$ então denota a taxa de transição entre o derivativo P e o derivativo Q. Análises de desempenho são feitas em termos da procura da distribuição estacionária de probabilidades entre os estados da PEPA. O GD também deve ser fortemente conectado para geração da CTMC correspondente.

A seguir, um exemplo de um único componente com três derivativos, $P1$, $P2$ e $P3$, cada um com uma ação distinta (*start* ou início, *run* ou execução e *stop* ou parada) e mesma taxa de ocorrência r para todas as ações. S denota a Equação do Sistema (*System Equation*), responsável pela instanciação dos processos e marcação do estado inicial, utilizado para percorrer os derivativos em

[5]Ou seja, a Cadeia de Markov deve ser ergódica [STE94].

busca dos estados possíveis do modelo. Este caso em específico não possui cooperação, tratando-se apenas de uma composição paralela simples ($\|$), detalhado a seguir.

$$\begin{aligned} P_1 &\stackrel{def}{=} (start, r).P_2 \\ P_2 &\stackrel{def}{=} (run, r).P_3 \\ P_3 &\stackrel{def}{=} (stop, r).P_1 \\ S &\stackrel{def}{=} (P_1 \parallel P_1) \end{aligned}$$

A MC gerada a partir deste modelo está mostrada na Figura 2.7. A combinação dos três componentes do modelo gerou nove estados e o sistema de transições de acordo com as regras predefinidas.

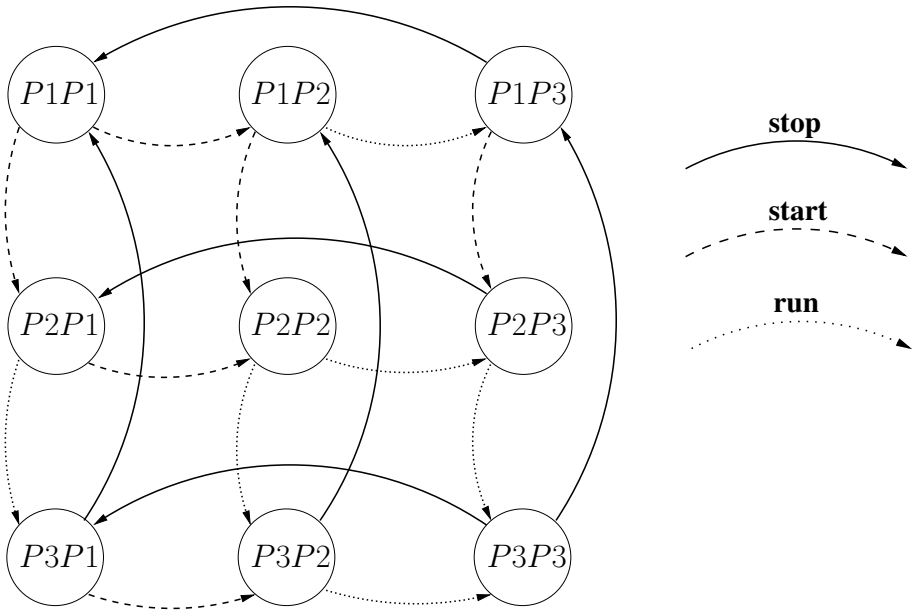


Figura 2.7: MC gerada a partir do exemplo modelado em PEPA.

Apesar do fato das álgebras de processos possuírem as vantagens expressas acima, sua modelagem é puramente textual e não fornece um atrativo visual de modelagem como verificado em outros formalismos como Redes de Petri Coloridas ou SAN, por exemplo. Esse fato, entre outros, instigou a definição do formalismo das PEPA nets, discutido a seguir.

2.5.2 PEPA nets

Com o objetivo de se criar um formalismo com representações gráficas e outros recursos, foram definidas as PEPA nets em 2001 [GIL01]. PEPA nets [GIL03] é uma combinação entre Redes de Petri Coloridas [JEN96] e PEPA. Os modelos são construídos como redes possuindo lugares que contém tanto componentes PEPA estáticos quanto móveis (melhor explicados a seguir). Este formalismo combina a composicionalidade de PEPA com a notação gráfica das Redes de Petri. PEPA nets

oferecem mais expressividade para os modeladores, mas deve ser vista como complementar à PEPA, pois alguns comportamentos são melhor definidos em um formalismo ou em outro [GIL03].

PEPA nets foi desenhada para capturar sistemas que possuam dois tipos distintos de mudança de estados, em particular, sistemas computacionais onde verifica-se a existência de mobilidade (sejam componentes que alteram sua posição em um plano ou entidades que deslocam-se de maneiras diferentes). Esse tipo de modelagem é a principal motivação para definição de PEPA nets. Para atingir esse objetivo, foi descrita uma combinação de formalismos com componentes PEPA representando as transições locais e os disparos das Redes de Petri capturando as reconfigurações globais do sistema (a mobilidade).

PEPA nets são Redes de Petri Coloridas onde as cores das marcas são componentes PEPA [GIL03]. Cada lugar da CPN é um contexto de PEPA o qual pode conter componentes estáticos e conjuntos de cooperação restringindo a movimentação das marcas dentro dos lugares. As marcas podem se mover para um determinado lugar apenas se existe espaço para um componente do seu próprio tipo, ou seja, o estado não possui nenhuma marca até o momento da movimentação (utiliza-se o símbolo [_] para esse propósito). PEPA nets são modelagens PEPA com maior dimensionalidade, ou seja, associam cores (na forma de componentes PEPA) a cada estado do sistema. As formas de realizar a conversão de uma PEPA net para PEPA foram pesquisadas em [GIL04a], o qual demonstra as regras a serem seguidas para efetuar a traduação de forma consistente.

Existem dois tipos de mudança de estados em uma PEPA net, que podem ser vistas como mudanças microscópicas ou macroscópicas. As mudanças microscópicas são locais, afetando componentes dentro dos seus contextos e são representadas usando-se transições governadas pela semântica usual de PEPA. Em contraste, as mudanças macroscópicas são globais e capturam o disparo de eventos no nível da rede. Estes disparos fazem com que as marcas sejam transferidas de um lugar para outro [GIL03]. Cada lugar, ou contexto de PEPA, contém células especificamente definidas, que só podem ser ocupadas pelo tipo apropriado. Esta definição é fortemente tipada, ou seja, uma célula ou nodo não pode ter uma marca que não seja a marca definida para esta célula.

Uma PEPA net distingue dois tipos de componentes: marcas e componentes estáticos. As marcas podem se movimentar entre os lugares (assim como ocorre em PN), enquanto que os componentes estáticos não permitem movimentação na rede com alterações globais de estado, representando o ambiente dentro daquele lugar. Estes componentes estáticos podem ser de dois tipos: *stateless* ou *stateful*. Um componente *stateless* não possui derivações e simplesmente oferece uma ou mais ações para interação. Os componentes *stateful* em contraste, possuem um conjunto mais rico de comportamentos incluindo a possibilidade de definir derivativos ou estados.

Um exemplo simples de PEPA net pode ser verificado através do modelo *MobileAgents*. Tratam-se de entidades, denominadas neste contexto de *agentes*, que visitam diferentes lugares de uma rede. A Figura 2.8 mostra uma visualização gráfica deste sistema. Existem três lugares na rede chamados P_1, P_2 e P_3, onde agentes podem se movimentar entre estes lugares de acordo com uma taxa. As transições locais acontecem dentro de cada lugar e os movimentos nas transições da rede causam mudanças globais ao sistema, na figura representadas pelos eventos *go* (com taxas λ_l e λ_r) e *return*

(com taxas μ_l e μ_r). A marcação inicial da rede encontra-se em P_2 em uma marca chamada A, representando um agente, chamado *Agent* conforme os derivativos do modelo descritos a seguir. Este modelo está explicado em detalhes em [GIL04a].

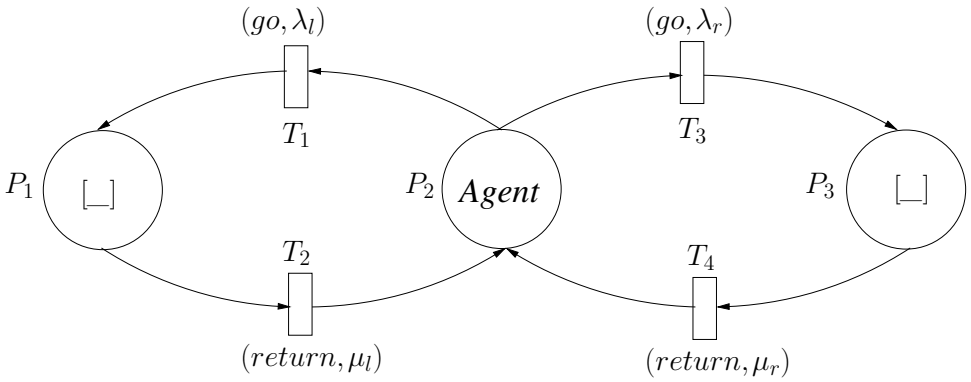


Figura 2.8: Um exemplo de uma PEPA net que descreve agentes móveis.

A ideia básica do modelo *MobileAgents* é considerar agentes que inspecionam diferentes lugares e retornam para um servidor chamado *Mestre* (*Master*) para depositar os resultados coletados. A classificação do lugar $Master$ é estático do tipo *stateful*, ou seja, pode alterar seu estado para $Master1$ com a atividade de depósito ($dump$). Este fato provocará a criação de um autômato adicional para relacionar esta problemática para uma tradução correspondente. A seguir, são definidos os componentes e lugares do sistema (em linguagem de PEPA nets):

$$
\begin{aligned}
Agent &\stackrel{def}{=} (go, \lambda).Agent1 \\
Agent1 &\stackrel{def}{=} (interrogate, r_i).Agent2 \\
Agent2 &\stackrel{def}{=} (return, \mu).Agent3 \\
Agent3 &\stackrel{def}{=} (dump, r_d).Agent \\
Master &\stackrel{def}{=} (dump, \top).Master1 \\
Master1 &\stackrel{def}{=} (analyse, \top).Master \\
Probe &\stackrel{def}{=} (monitor, r_m).Probe + (interrogate, \top).Probe \\
\mathbf{P1} &\stackrel{def}{=} Agent[\_] \underset{\mathcal{L}_1}{\bowtie} Probe \\
\mathbf{P2} &\stackrel{def}{=} Agent[Agent] \underset{\mathcal{L}_2}{\bowtie} Master \\
\mathbf{P3} &\stackrel{def}{=} Agent[\_] \underset{\mathcal{L}_1}{\bowtie} Probe
\end{aligned}
$$

onde $\mathcal{L}_1$ é o conjunto $\{ interrogate \}$ e $\mathcal{L}_2$ é o conjunto $\{ dump \}$.

PEPA nets podem ser naturalmente utilizadas no caso de existirem componentes que precisem se movimentar em uma rede ou alterar sua posição entre os lugares, comunicar com outros eventuais componentes que existam no lugar e retornar para seu lugar de origem. Este tipo de comportamento é mais facilmente perceptível na maneira gráfica que as PEPA nets utilizam para mostrar as interações envolvidas entre as diferentes entidades de um sistema e como estas cooperam para realizar atividades. Um outro exemplo válido são processos em uma rede de computadores, ou *grid*, que podem ser enviados para outros lugares de execução onde alteram seus estados locais, mas também

ocorre uma movimentação global na rede. Existem situações mais adaptadas para a utilização de PEPA nets, especialmente casos onde estejam presentes comportamentos locais e sincronizações globais. Como todo formalismo estruturado, uma PEPA net possui uma Cadeia de Markov correspondente, composta basicamente pela combinação das informações contidas na Matriz de Transição e no Grafo de Derivação, utilizando a Equação do Sistema para gerar os estados globais, assim como em PEPA.

Tanto modelos PEPA quanto modelos PEPA nets são analisados através da ferramenta computacional *PEPA Workbench* (pwb). Este ambiente completo de análise oferece ferramentas para lidar com a compilação de modelos que geram tanto o GD quanto a MT incluindo opções de agregação de estados (reduzindo o seu espaço de estados). Este ambiente também gera arquivos para entrada em outras ferramentas de uso matemático, como *Maple* ou *Mathematica* com vistas à solução via métodos diretos dos modelos criados.

PEPA nets já foi utilizada para modelar mobilidade [HIL04], modelagem de conceitos de Engenharia de *Software* [GIL04a], *Role-Playing Games* [GIL04b] e um ambiente para aplicações móveis usando *UML - Unified Modeling Language* [GIL06], apenas para citar algumas aplicações. A variedade de formas de modelagem evidenciam a versatilidade do formalismo e como este pode ser utilizado para descrever sistemas complexos.

Para uma completa definição de PEPA nets sugere-se a leitura de [GIL03, GIL04a], onde descrições detalhadas sobre o formalismo são definidas e conceitualizadas. A seguir é apresentada uma breve discussão entre os formalismos expostos neste trabalho e uma comparação sobre as vantagens e desvantagens dos seu usos.

2.6 Discussão

As Cadeias de Markov foram o primeiro formalismo diretamente aplicado à avaliação de desempenho de sistemas computacionais [STE07, STE09]. Seus conceitos podem ser aplicados a outras áreas tais como economia ou onde exista a necessidade de se descobrir probabilidades associadas a estados da realidade. Desde a sua adoção, o problema da explosão do espaço de estados emergiu como um grave impedimento ao seu uso, fato mitigado pelas definições dos formalismos estruturados para análise de sistemas complexos em termos de modelagem. As MC demonstram uma grande aplicabilidade devido às suas primitivas simples e facilidade do seu uso. Entretanto, dependendo do modelo criado constata-se que é fácil definir um espaço de estados massivo e, usualmente, intratável. Ao estruturar um determinado sistema é mais fácil entender o que foi modelado para então utilizar métodos específicos de solução que tratem de tais representações estruturadas. O objetivo é não apenas modelar sistemas mas inferir índices computacionais sobre sua operação, em um tempo razoável, facilitando a tomada de decisões *antes* do sistema estar fisicamente disponível.

Todos os formalismos possuem em comum comportamentos locais que importam a apenas cada entidade bem como comportamentos sincronizantes onde está presente a necessidade de trabalharem cooperativamente. Estes tipos de interações são adaptáveis para problemas distribuídos e paralelos

de diversas aplicações. A escolha de um formalismo para avaliar um sistema implica em decidir as vantagens e desvantagens observando-se a composicionalidade, a localidade, o sincronismo, a evolução, a interação, a abstração e a percepção visual (sistemas com representações gráficas). Cada formalismo propõe um conjunto de ferramentas e diferentes metodologias para tratar da modelagem e solução de sistemas.

Pesquisas atuais centram seus esforços no aumento do poder de representação dos sistemas para lidar com novas primitivas de modelagem. Outro importante foco de atenção é direcionado ao aperfeiçoamento dos mecanismos de análise e simulação oferecidos aos usuários. Estas análises permitem que sejam descobertos os fatores mais importantes que melhoram ou degradam o desempenho de sistemas, fornecendo informações adicionais para ajudar os modeladores a decidir sobre as melhores condições para uma operação otimizada com um uso racional dos recursos. Estas melhorias são direcionadas à concepção de novas primitivas de modelagem de sistemas, aumentando a oferta de diferentes possibilidades de mapear as realidades para modelos analíticos.

Mais especificamente, ao se comparar dois formalismos tais como SAN e PEPA nets, observa-se o compartilhamento de similaridades. Uma constatação inicial, por exemplo, é que ambos descrevem sistemas através de representações gráficas e, no caso de PEPA nets, são baseadas em formalismos clássicos (Redes de Petri Coloridas) facilitando o entendimento. O mesmo acontece para o caso das Redes de Petri e Redes de Petri Coloridas. Esses recursos visuais auxiliam e simplificam a análise e inspeção de modelos. Uma vantagem de PEPA e PEPA nets sobre SAN diz respeito ao produto do espaço de estados (a combinação de todos os estados). Em SAN, pode-se ter que trabalhar com um espaço de estados atingível muito menor que o espaço de estados produto necessário para a solução do modelo. Em contraposição, em PEPA, apenas os estados atingíveis são considerados. Entretanto, em termos de solução, SAN, por trabalhar com descritores Markovianos, emprega melhor os recursos disponíveis, principalmente os de memória, para calcular o vetor de probabilidades. Em PEPA, precisa-se incorporar a MT com as taxas dos eventos em uma ferramenta computacional com recursos matemáticos, como por exemplo, o *Maple* ou *Mathematica*. Dependendo do tamanho do modelo são necessários muitos recursos em termos de memória e processamento para que uma solução seja obtida.

Dadas as opções de formalismos Markovianos verifica-se que existem diversas formas de representar e descrever sistemas com diferentes primitivas e abstrações. Cada formalismo captura aspectos particulares de realidades incorporando variadas análises. Alguns formalismos oferecem opções sofisticadas para representar a interação entre os componentes enquanto que outros se preocupam com facilidades de prover as abstrações. É comum adaptar ideias de diferentes formalismos e combiná-las entre si, ampliando as definições com o intuito de aumentar as funcionalidades ofertadas.

O próximo capítulo trata sobre abstrações de modelagem que podem ser usadas para decompor sistemas e sobre correspondência entre formalismos estruturados de solução. Aborda-se também um estudo de caso de mapeamento entre formalismos para obtenção de um descritor Markoviano.

3. MODELO ABSTRATO DE FORMALISMOS ESTRUTURADOS

Este capítulo trata de formas mais gerais para mapeamento de realidades, regras de descrição, correspondência entre formalismos estruturados e as dificuldades existentes nestas conversões. A Seção 3.1 introduz o assunto para, na Seção 3.2, descrever as definições de base. A Seção 3.3 demonstra as correspondências entre alguns formalismos da literatura para na Seção 3.4, apresentar um estudo de caso para conversão de modelos PEPA nets em SAN. A Seção 3.5 descreve as primitivas de comunicação e interação entre os formalismos com um exemplo, conforme é mostrado na Seção 3.6. O capítulo é finalizado com uma discussão sobre as traduções entre as formas de representação de sistemas através de formalismos para análise de desempenho, na Seção 3.7.

3.1 Introdução

Os formalismos estruturados existentes para mapeamento de realidades definem seus próprios conjuntos de primitivas de modelagem. A principal característica que os distingue é o grau de abstração disponível e o funcionamento do seu mapeamento em alto nível para uma descrição mais formal e não ambígua.

Antes de prosseguir com a discussão de modelos mais abstratos para tratar com formalismos estruturados é necessário realizar a distinção entre uma *Cadeia de Markov* e um *Processo Markoviano*. A forma usual de separar estes conceitos é através da observação da escala de tempo adotada. Enquanto que a primeira é uma definição em escala de tempo discreto, os Processos Markovianos tratam apenas de escala de tempo contínuo [BUC08]. Um Processo Markoviano se refere a um modelo matemático de um sistema dotado da propriedade Markoviana, ou seja, um sistema onde a propensão para se estar em um estado futuro depende somente das informações contidas no estado atual, sem precisar nenhuma outra informação sobre o histórico do passado. Como mencionado na Seção 2.2, esta propriedade é central para o estudo deste formalismo e é referenciada na literatura através de *memoryless property* [STE94].

Entretanto, apesar da existência das duas escalas terem sido definidas originalmente, a maneira mais usual de se trabalhar com processos estocásticos para análise quantitativa de sistemas é definir um Processo Markoviano [BHA97], em alguns casos mais natural que trabalhar com escala discreta e consequentemente com Cadeias de Markov (este tópico será mais discutido a seguir, na Seção 3.2). Conforme mencionado no Capítulo 2, o objetivo é analisar a estacionariedade de modelos observando o comportamento dos sistemas em longas trajetórias, como se fosse 'simulado' até atingir um estágio onde não mais importa seu estado inicial. Ao atingir um regime estacionário, o sistema produz probabilidades no regime estacionário que atestam valores quantitativos para serem analisados pelos modeladores, os quais interpretam os dados para extração de índices de desempenho.

O que se observa mais frequentemente é o uso de ambas nomenclaturas, ou seja, tanto Processos Markovianos quanto Cadeias de Markov para querer referenciar sistemas com um total de estados

enumerável e finito possuidor de transições. No contexto desta tese, particularmente na Seção 2.2, escolheu-se descrever Cadeias de Markov e introduzir os conceitos relativos a Processos Markovianos neste capítulo, já que existe tal distinção entre ambos.

Uma forma mais abstrata de se estudar Processos Markovianos com estruturação está baseada no conceito de como as partes ou subsistemas se relacionam, ou seja, como sincronizam atividades e desempenham papéis independentes. Este conceito, ao considerar-se os módulos como um todo, também pode ser visto como se cada parte fosse um processo e estes interagem através de comunicações ou troca de mensagens de forma sincronizada. Tais processos foram denominados Processos Markovianos com Comunicação (PMC) [BUC08].

A definição de PMC para avaliação de desempenho não é nova, sendo já utilizada por outros autores em outros contextos [CAM99, BUC02b, BUC08], até mesmo em outros formalismos [DON94, HIL96, KEM96]. Esta classe de processos é estudada há bastante tempo mas só recentemente teve sua utilização realçada pelo problema da explosão do espaço de estados e complexidade das operações tensoriais envolvidas [FER98b, BUC00]. Outro importante fator que impulsionou a pesquisa em PMC foi a definição de múltiplos formalismos estruturados de modelagem e a percepção de diversas características em comum entre estes. Exemplos de formalismos com propriedades em comum, destacam-se as Redes de Filas de Espera, Redes de Autômatos Estocásticos, Redes de Petri Estocásticas e Álgebras de Processos, todos definindo e trabalhando com descritores Markovianos para representação implícita da matriz de transição da Cadeia de Markov correspondente [DON93, DON94, FER98b, HIL07].

3.2 Conceitos preliminares

A descrição de um sistema através de Processos Markovianos com Comunicação é uma maneira versátil de modelagem de realidades com transições complexas e com grande número de sincronizações entre suas entidades. A solução eficiente desta classe de processos aplicada às transições generalizadas (componentes que realizam transições em função dos estados de outros componentes), ainda é um problema em aberto, alvo de pesquisas que apontem maneiras de se tratar com tais estruturas de forma eficaz. Esta seção discutirá as principais formas de definição de PMC e calcular as probabilidades estacionárias ou transientes e suas implicações na memória necessária e seus efeitos no descritor Markoviano correspondente.

A ideia de se utilizar PMC para descrição de modelos é pensar que um sistema é composto por um conjunto de componentes com uma organização interna e verificar como estes componentes (ou módulos) interagem entre si, ou seja, onde existe pelo menos uma comunicação para sincronização de atividades. Este conceito de comunicação está presente em MC, pois pode-se pensar que os estados com transições definidas estão em processo de comunicação.

A motivação para o uso de PMC está centrada no fato de que é mais fácil entender um sistema abstraindo os detalhes menos importantes e pensar na sua semântica operacional como um todo. Depois desta descrição composicional (onde descrevem-se os componentes) do sistema, parte-se para

o próximo estágio, que é determinar computacionalmente a existência ou não de estacionariedade e estabelecer a fase de análise e refinamento do modelo para refletir a realidade [BUC08].

A própria ideia de se decompor um sistema em partes menores para serem melhor entendidas foi estudada no contexto da Engenharia de *Software* na década de 70 [PAR72], usando o conceito de 'módulos'. Dando seguimento a estes conceitos pode-se inferir sua utilização no contexto de avaliação quantitativa de desempenho de sistemas, tal como o formalismo de PEPA (maiores informações na Seção 2.5.1). A definição original de PEPA [HIL96] estabelece que cada componente sequencial do modelo (conforme Seção 2.5.1) é visto como um 'autômato', ou seja, estes relacionamentos estão presentes nas definições originais de alguns formalismos estruturados. Estes conceitos implicam na necessidade da área de avaliação de desempenho de definir formatos intercambiáveis pois, na verdade, a maioria dos formalismos possui primitivas em comum. É nesse cenário que são introduzidos os Processos Markovianos com Comunicação, responsáveis pela definição dos componentes existentes e de suas sincronizações de transições.

O uso de mecanismos estruturados potencializa a descoberta de propriedades da Cadeia de Markov correspondente. Ao compartimentalizar os elementos do sistema, abre-se a possibilidade de se descobrir partições mais facilmente, utilizadas para efeitos de agregação e redução do modelo [SCH83, BUC94, BEN04a, HIL05b]. Esta composição do sistema promove, também, um maior entendimento do sistema, pois divide o problema em partes mais gerenciáveis e menos suscetíveis a erros.

Outro importante fato que contribui para a disseminação de PMC é a aplicação de técnicas do tipo divisão-e-conquista para atacar um problema. Tais métodos são utilizados para a detecção de gargalos e características indesejadas em sistemas usualmente com grandes espaços de estados e possuidores de transições complexas.

A seguir, na próxima seção, serão mostrados exemplos de correspondência entre os formalismos estruturados e PMC bem como um estudo de caso compreendendo a transformação de uma PEPA net (explicado anteriormente na Seção 2.5.2) em uma SAN (conforme a Seção 2.4), mostrando a sua correspondência.

3.3 Correspondência entre formalismos

Estabelece-se que uma correspondência entre os formalismos existe ao ser obtida a mesma matriz de transição da Cadeia de Markov subjacente. O vetor solução calculado para um determinado modelo é idêntico ao da solução do mesmo modelo em um formalismo estruturado e na MC, dada a existência da correspondência. Como explicado anteriormente, uma das principais motivações para o uso de PMC (e as SAN são uma classe deste tipo de formalismo) deve-se ao fato de nunca ser necessário armazenar a matriz de transição da MC correspondente mas acessar a estrutura através da utilização de propriedades da Álgebra Tensorial, onde são compilados os descritores Markovianos. Toda a fundamentação teórica envolvida para PMC envolve a definição de tais descritores, bem como as formas de se multiplicar o vetor de probabilidades por essa estrutura, para então

usar um método iterativo de solução. A utilização de descritores tensoriais mitiga o problema de armazenamento relacionado à explosão do espaço de estados associado às Cadeias de Markov. Portanto, os formalismos estruturados de solução podem ser classificados como Processos Markovianos com Comunicação. No caso de SAN, a principal unidade criada é um *autômato*, em PEPA é um *processo* e nas SGSPN é um *módulo*. PMC é então utilizado para definir como estas unidades se comunicam, através da definição das transições existentes e suas taxas. Uma extensão das PMC são os chamados Processos Markovianos com Comunicação Generalizados (PMCG) que definem o funcionamento das transições no sistema através da utilização de taxas funcionais (usando preceitos da Álgebra Tensorial Generalizada).

A seguir, estuda-se um modelo baseado em uma Rede de Filas de Espera Aberta a ser transposto para uma Rede de Autômatos Estocásticos, uma Rede de Petri Estocástica e para um modelo em Álgebra de Processos (escrito no formalismo de PEPA), conforme a Figura 3.1. O objetivo é o de demonstrar a correspondência entre alguns exemplos de diferentes formalismos estruturados.

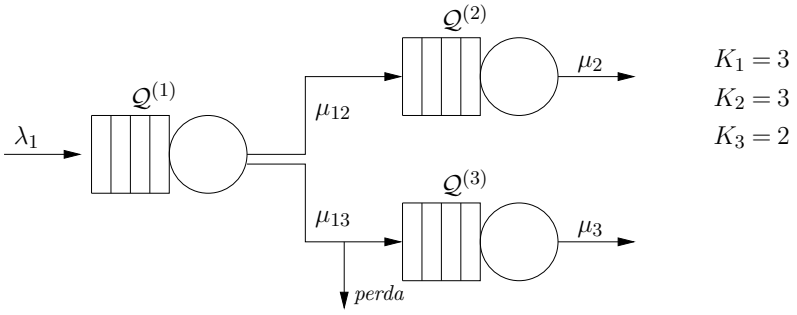


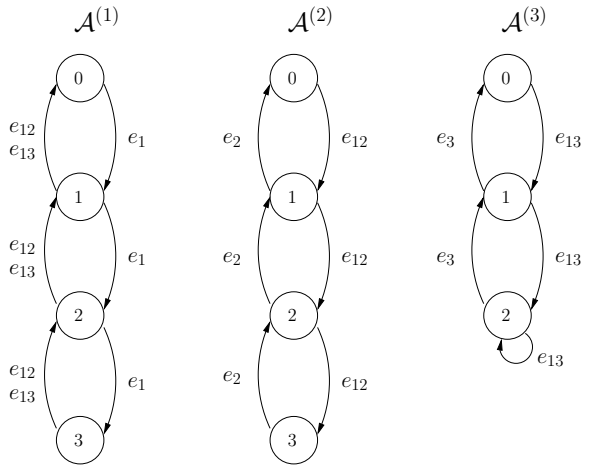
Figura 3.1: Rede de Filas de Espera Aberta com bloqueio e perda.

Neste exemplo, observa-se uma rede composta por três filas definidas por $\mathcal{Q}^{(1)}$, $\mathcal{Q}^{(2)}$ e $\mathcal{Q}^{(3)}$ com capacidades iguais a $K_1 = 3$, $K_2 = 3$ e $K_3 = 2$, respectivamente. As filas são caracterizadas pela existência de bloqueio e perda, ou seja, quando estão preenchidas ou cheias (ou seja, encontram-se na sua capacidade máxima), os clientes ou são bloqueados ou são perdidos, dependendo do caso.

O exemplo da Figura 3.1 apresenta graficamente as três filas, a taxa de chegada de clientes (λ_1), as taxas de saída da primeira fila para a segunda e para a terceira, definidas por μ_{12} e μ_{13} respectivamente e as de saída de clientes dadas por μ_2 e μ_3. A figura mostra uma fila com comportamento bloqueante (para os clientes da fila $\mathcal{Q}^{(1)}$ para a fila $\mathcal{Q}^{(2)}$) e outra fila com comportamento de perda (para os clientes da fila $\mathcal{Q}^{(1)}$ para a fila $\mathcal{Q}^{(3)}$).

Este modelo estruturado de filas pode ser visto, por exemplo, como uma Rede de Autômatos, uma Rede de Petri ou descrições em Álgebra de Processos. A seguir, discute-se este exemplo definido nestes formalismos citados, começando por Redes de Autômatos Estocásticos. A Figura 3.2 mostra uma tradução possível da Rede de Filas de Espera da Figura 3.1 para SAN.

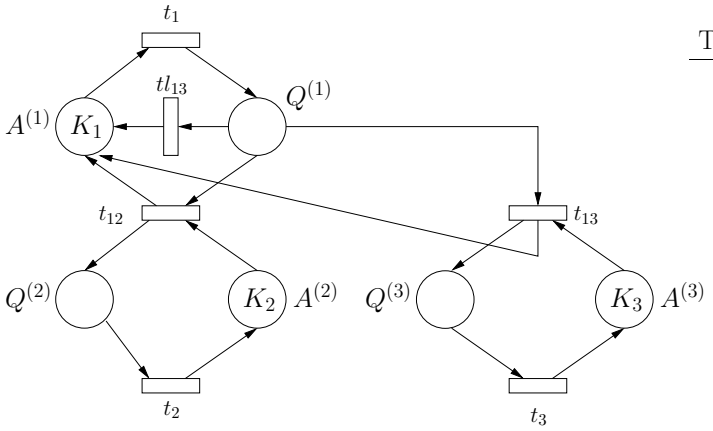
Cada fila é modelada por um autômato ($\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$, $\mathcal{A}^{(3)}$) e cada estado representa a quantidade de clientes na fila, ou seja, a sua capacidade. Por exemplo, as filas tem capacidades iguais a três para as filas $\mathcal{Q}^{(1)}$ e $\mathcal{Q}^{(2)}$ e capacidade dois para a fila $\mathcal{Q}^{(3)}$. Para a modelagem, serão criados dois



Tipo	Evento	Taxa
loc	e_1	λ_1
loc	e_2	μ_2
loc	e_3	μ_3
syn	e_{12}	μ_{12}
syn	e_{13}	μ_{13}

Figura 3.2: Tradução de Redes de Filas de Espera Aberta para SAN.

autômatos com quatro estados e um autômato com três estados, simbolizando nenhum, um, dois ou três clientes para as duas primeiras filas, e nenhum, um ou dois clientes para a terceira fila. O comportamento de perda da fila $\mathcal{Q}^{(3)}$ é simbolizado pelo autômato $\mathcal{A}^{(3)}$ com a auto-transição do último estado. O comportamento bloqueante está representado na inexistência de auto-transição no último estado do autômato $\mathcal{A}^{(2)}$ informando que o evento e_{12} só é possível quando a fila $\mathcal{Q}^{(2)}$ não está cheia, logo, o estado de $\mathcal{A}^{(1)}$ fica bloqueado até o evento e_{12} poder ser disparado.



Transição	Taxa
t_1	λ_1
t_2	μ_2
t_3	μ_3
t_{12}	μ_{12}
t_{13}	μ_{13}
tl_{13}	$\mu_{13} * (tk(Q^{(3)}) == K_3)$

Figura 3.3: Tradução de Redes de Filas de Espera Aberta para SPN.

A Figura 3.3 mostra a mesma conversão para uma Rede de Petri Estocástica (inspirada na existente em [BRE05a]). Cada fila i corresponde a um par de lugares $Q^{(i)}$ e $A^{(i)}$. O lugar $A^{(i)}$ representa a disponibilidade dos clientes na fila $Q^{(i)}$, ou seja, sua capacidade, e o lugar $Q^{(i)}$ representa os clientes correntes da fila i. A transição l_1 representa uma chegada de cliente na fila $Q^{(1)}$. Já as transições t_2 e t_3 representam, respectivamente, saídas de clientes das filas $Q^{(2)}$ e $Q^{(3)}$ com taxas

iguais a μ_2 e μ_3. A transição sincronizante t_{12} representa o roteamento de clientes da fila $Q^{(1)}$ para $Q^{(2)}$. De forma análoga, a transição t_{13} corresponde ao roteamento da fila $Q^{(1)}$ para $Q^{(3)}$. A transição tl_{13} simboliza a perda de clientes de $Q^{(1)}$ para $Q^{(3)}$ (representado na SAN da Figura 3.2 pelo disparo do evento e_{13} no autômato $A^{(1)}$ e auto-transição no último estado do autômato $A^{(3)}$). Esta transição só é disparada caso a fila $Q^{(3)}$ esteja cheia (conforme a condição de guarda que verifica se o total de marcas é igual à capacidade da fila).

Por fim, converte-se a Rede de Fila Aberta deste exemplo para o seu correspondente em PEPA. A Figura 3.4 mostra a tradução através da construção de três processos, simbolizando as três filas do exemplo. Os derivativos dos processos assumem a nomenclatura dada segundo a forma QNC onde N corresponde ao identificador da fila e C à quantidade de clientes na fila. Os processos ditam as transições e as atividades do sistema como um todo.

$$
\begin{aligned}
&Q10 \stackrel{def}{=} (e_1, \lambda_1).Q11 \\
&Q11 \stackrel{def}{=} (e_1, \lambda_1).Q12 + (e_{13}, \mu_{13}).Q10 + (e_{12}, \mu_{12}).Q10 \\
&Q12 \stackrel{def}{=} (e_1, \lambda_1).Q13 + (e_{13}, \mu_{13}).Q11 + (e_{12}, \mu_{12}).Q11 \\
&Q13 \stackrel{def}{=} (e_{13}, \mu_{13}).Q12 + (e_{12}, \mu_{12}).Q12
\end{aligned}
$$

$$
\begin{aligned}
&Q20 \stackrel{def}{=} (e_{12}, \mu_{12}).Q21 \\
&Q21 \stackrel{def}{=} (e_{12}, \mu_{12}).Q22 + (e_2, \mu_2).Q20 \\
&Q22 \stackrel{def}{=} (e_{12}, \mu_{12}).Q23 + (e_2, \mu_2).Q21 \\
&Q23 \stackrel{def}{=} (e_2, \mu_2).Q22
\end{aligned}
$$

$$
\begin{aligned}
&Q30 \stackrel{def}{=} (e_{13}, \mu_{13}).Q31 \\
&Q31 \stackrel{def}{=} (e_{13}, \mu_{13}).Q32 + (e_3, \mu_3).Q30 \\
&Q32 \stackrel{def}{=} (e_3, \mu_3).Q31 + (e_{13}, \mu_{13}).Q32
\end{aligned}
$$

$$
\mathbf{S} \stackrel{def}{=} Q20 \underset{e_{12}}{\bowtie} Q10 \underset{e_{13}}{\bowtie} Q30
$$

Figura 3.4: Tradução de Redes de Filas de Espera Aberta para PEPA.

Neste modelo PEPA a Equação do Sistema é definida por $\mathbf{S}$ e identifica as comunicações existentes entre os processos e o estado inicial (inicialmente, todas as filas encontram-se sem clientes, ou seja, em $Q10$, $Q20$ e $Q30$). Entre a Fila $Q2$ e a $Q1$ existe uma sincronização dada pela ação e_{12} enquanto que pela Fila $Q1$ e $Q3$, a sincronização é realizada pela atividade e_{13}. Cabe ressaltar que os modelos previamente traduzidos possuem solução correspondente, ou seja, produzem a mesma matriz para ser usada no cálculo dos índices através de métodos diretos ou iterativos de solução de sistemas, conforme explicado na Seção 2.2. Apesar de existirem diferentes formas de visualizar e definir o problema, modelos de diferentes formalismos podem vir a produzir a mesma matriz subjacente, ou seja, existem características em comum para as classes de problemas. Estas propriedades em comum são tratadas pelos PMC, que auxiliam os usuários na definição e solução deste formalismo. A seguir é mostrado um estudo de caso de transcrição do formalismo de PEPA nets para um descritor Markoviano baseado no formalismo de SAN.

3.4 Estudo de caso: tradução de PEPA nets para descritor

Mecanismos de tradução entre formalismos são alternativas válidas para permitir que modelos sejam visualizados e resolvidos de diferentes maneiras. Neste sentido, a dificuldade de se realizar um mapeamento consistente e representativo deve-se ao fato de que cada formalização apresenta um conjunto de primitivas específico, normalmente difícil de ser mapeado. A seguir será mostrada uma tradução entre dois formalismos bastante usuais de modelagem, que são as PEPA nets (maiores informações na Seção 2.5.2) e as SAN (conforme Seção 2.4), um formalismo que gera um formato tensorial através da criação do descritor Markoviano.

Dado que as PEPA nets podem ser vistas como uma rede onde as marcas são componentes PEPA, as traduções a seguir podem ser aplicadas diretamente ao mapeamento de modelos PEPA, sem perda de generalidade. A seguir será mostrado como realizar um mapeamento válido entre os formalismos indicados através do uso do Grafo de Derivação (GD) e da Matriz de Transição (MT) gerada pelo compilador de PEPA nets.

O objetivo deste estudo de caso é apresentar as regras de tradução que convertem um modelo descrito em PEPA nets para uma SAN de forma direta, sem precisar converte-lo para uma representação intermediária baseada em PEPA [GIL04a]. A partir das informações contidas tanto no GD quanto no MT será possível criar transições com eventos associados de acordo com seu tipo (local ou sincronizante). Os resultados apresentados demonstram um mapeamento válido, mostrando uma correspondência entre as matrizes de transição de ambos os formalismos.

3.4.1 Trabalhos relacionados e observações iniciais

Uma tradução válida entre PEPA nets (Seção 2.5.2) e PEPA foi proposta para facilitar e acessar o conjunto de ferramentas disponíveis para álgebras de processos [GIL04a]. Contudo, ao converter uma PEPA net para uma SAN observa-se que um dos objetivos é o mesmo, que é o de utilizar o *software* de solução e as técnicas (no caso as ferramentas já existentes para tratar com descritores deste tipo) disponíveis em ferramentas implementadas para este fim, tais como o PEPS ou o GTAexpress. Um outro objetivo é o de obter uma representação tensorial para PEPA, já que este formalismo trabalha apenas com a matriz de transição. No trabalho que mais se aproxima deste objetivo [HIL07], os autores realizaram um estudo sobre como relacionar PEPA e SAN de forma conjunta, analisando as vantagens e desvangens deste mapeamento inclusive quando taxas funcionais (e, consequentemente, ATG) são usadas.

Outras traduções entre formalismos foram observadas anteriormente, tais como representação de PEPA para SPN [RIB95] ou através do uso do Compilador Imperial para PEPA (*ipc*, ou *The Imperial PEPA Compiler*). Um outro trabalho importante que tratou sobre o mapeamento para descritor foi estudado em [HIL01] que estabeleceu os fundamentos de base necessários para representar modelos PEPA.

Ambos formalismos considerados nesta seção são visualmente atraentes e mostram as relações de causa e efeito de forma simples. Esta forma permite que sejam criados mecanismos que convertem

as regras semânticas e da álgebra de processo das PEPA nets para determinar como as entidades colaboram entre si em um formato tensorial similar ao das SAN. Ao utilizar as informações contidas na MT e no GD, percebe-se as ocorrências das taxas no nível da MC e como está ocorrendo a evolução de um estado para o próximo. Estes dados são compilados pela ferramenta PEPA Workbench (pwb) restando apenas discernir as informações mais relevantes contidas nos arquivos produzidos, desta forma simplificando o mapeamento. Devem ser guardados os rótulos das transições e os nomes e taxas dos eventos que foram previamente utilizados pelo modelador, para facilitar o entendimento do modelo convertido.

3.4.2 Método de tradução

A definição formal de PEPA nets [GIL03] considera a interação entre os processos existentes e como as entidades colaboram, entretanto, a *pwb* não faz distinção entre eventos locais ou sincronizantes e este dado é necessário para a construção de SAN. Ao inspecionar o GD e a MT de uma PEPA net, reconhece-se quatro classes distintas de eventos:

- **Ações Individuais (*Individual Actions, ou IA*):** corresponde a uma mudança dentro de um *lugar*, significando que um componente PEPA trocou de estados de forma individual e é traduzido sob a forma de descritor para um evento local simples (LOC). Ações individuais relacionam-se a marcas e a componentes estáticos do tipo *stateful* (definição na Seção 2.5.2);

- **Movimentação de Marca (*Token Movement, ou TK*):** duas marcas se movem em *lugares* diferentes, causando duas mudanças de estado em dois autômatos diferentes. Utiliza-se a notação presente em SAN para eventos sincronizante (SYN);

- **Ações de Cooperação (*Cooperation Action, ou CA*):** o mesmo que *TK*, exceto que uma mudança também promove uma outra mudança no estado de um componentes estático do tipo *stateful*;

- **Ações de Cooperação Múltiplas (*Multi-Cooperation Action, ou MCA*):** este tipo de evento é um caso particular no contexto de PEPA. Um único evento é responsável pela mudança de estado de diversos autômatos. A tradução é um evento sincronizante que opera sobre múltiplas entidades, alterando o estado global do sistema, igualmente chamado de sincronização do tipo *multi-way* na literatura de autômatos e álgebras de processos.

A Tabela 3.1 mostra as quatro classes de eventos observadas. A coluna *Evento* contém o rótulo entre duas transições. A tabela mostra um trecho de uma MT e DG, onde observa-se que ambos os componentes do tipo *Probe* dos lugares $Place_1$ e $Place_3$ são componentes estáticos do tipo *stateless*, enquanto que o componente *Master* é estático do tipo *stateful*, podendo se comportar eventualmente como *Master1*. No topo de cada coluna, os rótulos PN_i e CN_i indicam o nome do autômato correspondente que será criado de acordo com a Figura 3.5, onde o método é aplicado e

Tabela 3.1: Tradução do modelo *MobileAgents* para PEPA nets.

Cl.	*MT*	*GD*	$P1_1$		$P2_1$	$C2_1$	$P3_1$		*Trad.*
	Evento	Estado	$Place_1 - Probe$		$Place_2 - Master$		$Place_3 - Probe$		
...	...	...	...	...	...	...	...	...	...
IA	*interrogate*	*from* 2	Agent[**Agent1**]	Probe	Agent[_]	Master	Agent[_]	Probe	*LOC*
		to 4	Agent[**Agent2**]	Probe	Agent[_]	Master	Agent[_]	Probe	
IA	*analyse*	*from* 6	Agent[_]	Probe	Agent[Agent]	**Master1**	Agent[_]	Probe	*LOC*
		to 1	Agent[_]	Probe	Agent[Agent]	**Master**	Agent[_]	Probe	
TK	*go*	*from* 1	Agent[_]	Probe	Agent[**Agent**]	Master	Agent[_]	Probe	*SYN*
		to 2	Agent[**Agent1**]	Probe	Agent[_]	Master	Agent[_]	Probe	
CA	*dump*	*from* 5	Agent[_]	Probe	Agent[**Agent3**]	**Master**	Agent[_]	Probe	*SYN*
		to 6	Agent[_]	Probe	Agent[**Agent**]	**Master1**	Agent[_]	Probe	
...	...	...	...	...	...	...	...	...	...

gera quatro autômatos para este exemplo (melhor explicado na sequência desta seção). A classe é mostrada pela coluna *Cl.* e a tradução para SAN está na coluna *Trad*.

Ao conhecer as classes de eventos possíveis, as informações da MT e GD, calcula-se a quantidade de autômatos e o produto do espaço de estados necessário dada a existência de um modelo PEPA net com P lugares e C componentes. Cada lugar será transformado em um autômato, caso exista componentes estáticos do tipo *stateful*, será necessário um autômato para cada lugar definido, correspondendo a $A = P + C \times (1 - f)$ autômatos, onde f corresponde ao tipo de componente.

O conjunto de derivações possíveis com sincronizações nos lugares de cada componente ditarão o número de estados dos autômatos. Se o componente é estático e *stateless*, será necessário definir um estado adicional para representar uma célula vazia (e.g. [_] na PEPA net), diferentemente do procedimento a ser adotado para componentes estáticos do tipo *stateful*, onde não existem células vazias, apenas transições. O cálculo final do PSS é o produto dos estados de todos os autômatos que foram traduzidos do modelo PEPA ou PEPA net.

Cada classe de evento descrita na Tabela 3.1 irá afetar a maneira de se construir a especificação final dos autômatos do modelo. Para eventos locais é direto: o rótulo para o evento é salvo e a transição é criada. Para eventos sincronizantes é necessário descobrir todos os autômatos envolvidos no sincronismo e verificar se já não existe um evento definido previamente que realiza a mesma tarefa. Caso não exista, o evento sincronizante é criado, observando o rótulo definido pelo modelador.

O método de tradução é apresentado no Algoritmo 3.1, de acordo com três fases distintas: primeiramente monta as estruturas de dados necessárias observando as marcações iniciais passando para a fase de preenchimento de informações e criação de autômatos sendo finalizada com a fase de composição e formação de detalhes para criação de um modelo SAN com todas as suas características (funções de integração, seções de eventos e identificadores, entre outros). A primeira fase é responsável pela criação de um dicionário de rótulos contendo lugares, componentes, transições, marcas e taxas para serem manipuladas pelas demais fases.

Para a operação do algoritmo são necessárias as informações do GD, da MT e duas listas auxiliares, uma contendo os lugares para cada transição $g_i \in GD$ e outra para as mudanças entre os

Algoritmo 3.1: Tradução de PEPA nets para SAN.

1: $TM \leftarrow$ Matriz de Transição
2: $DG \leftarrow$ Grafo de Derivação
3: $P \leftarrow$ Lista de Lugares para todas as transições $t \in DG$
4: $C \leftarrow$ Lista de Componentes para todas as transições $t \in DG$
5: $Automata \leftarrow CreateEmptyAutomatonSet(P, C)$
6: $EL \leftarrow$ Lista de Eventos //inicialmente vazia
7: $INTF \leftarrow$ Lista de Funções de Integração //inicialmente vazia
8: **for all** $(q_{i,j} \in TM \;\; e \;\; q_{i,j} > 0)$ **do**
9: $evt \leftarrow GetEvent(q_{i,j})$
10: $(pc_1, pc_2) \leftarrow$ *descobre mudanças de lugares em* $P_i \rightarrow P_j$
11: $(cc_1, cc_2) \leftarrow$ *descobre mudanças de componentes em* $C_i \rightarrow C_j$
12: $class \leftarrow DiscoverClass(pc_{1,2},\ cc_{1,2})$
13: **if** $(class = IA)$ **then**
14: $(t_{from}, t_{to}) \leftarrow DiscoverTransition(pc_1, cc_1)$
15: $AddLocal(t_{from}, t_{to}, Automata[pc_1, cc_1], evt)$
16: **else if** $(class = TK)$ **then**
17: $(t_{from_1}, t_{to_1}) \leftarrow DiscoverTransition(pc_1)$
18: $(t_{from_2}, t_{to_2}) \leftarrow DiscoverTransition(pc_2)$
19: $AddSynch(t_{from_1}, t_{to_1}, Automata[pc_1], evt)$
20: $AddSynch(t_{from_2}, t_{to_2}, Automata[pc_2], evt)$
21: **else if** $(class = CA)$ **then**
22: $(t_{from_1}, t_{to_1}) \leftarrow DiscoverTransition(pc_1)$
23: $(t_{from_2}, t_{to_2}) \leftarrow DiscoverTransition(cc_1)$
24: $AddSynch(t_{from_1}, t_{to_1}, Automata[pc_1], evt)$
25: $AddSynch(t_{from_2}, t_{to_2}, Automata[cc_1], evt)$
26: **else if** $(class = MCA)$ **then**
27: $AddMultiWaySynch(AllTokenChanges, Automata, evt)$
28: **else**
29: *classe não antecipada* $\rightarrow$ $erro$
30: **end if**
31: $AddEvent(EL, evt, class)$
32: **end for**
33: **for all** $(g_i \in DG)$ **do**
34: $AddIntegrationFunction(INTF, Automata, g_i)$
35: **end for**
36: $Verify(Automata)$
37: usar EL para preencher seção SAN de *identifiers* e seção de *events*
38: usar marcação inicial PEPA na seção SAN de *partial reachability*
39: usar $Automata$ para criar seção SAN de *network*
40: usar $INTF$ para listar seção SAN de *results*
41: **return** $(Automata)$

componentes. Ainda no início, é chamada uma função denominada $CreateEmptyAutomatonSet$ que descobre as mudanças de componentes existentes e cria o conjunto de autômatos sem nenhuma transição ou evento. Após descobrir os autômatos necessários, passa pela MT descobrindo todas as transições que foram efetuadas, organizando-as conforme vai operando sobre o total de linhas existentes. Para descobrir os eventos é usada uma função chamada $GetEvent$ que retorna o nome do evento utilizado para o mapeamento dos rótulos.

As linhas 10 e 11 do algoritmo manipulam duas transições (de P_i para P_j e de C_i para C_j) e, para cada caso, são analisadas e retornam os índices das marcas que alteraram suas posições e para o caso de componentes, os índices dos componentes que mudaram. A função $DiscoverClass$ usa esta classe de índices descobertos e determina a classe dentro das possibilidades existentes ($\{IA$, TK, CA, $MCA\}$). Se o método retorna algo imprevisto ao tentar detectar a classe, uma condição de erro é instaurada. Esta ocorrência é plenamente possível pois o usuário da ferramenta *pwb* (*PEPA Workbench*) pode estar usando opções de agregação ($-aggregate$).

A seguir, em cada caso, são chamadas funções específicas, tais como $DiscoverTransition$, $AddLocal$ e $AddSynch$ que utilizam a informação do GD para retornar os rótulos para as transições para que os autômatos sejam criados com o mesmo nome utilizado na modelagem da PEPA net. As funções acrescentam eventos locais e sincronizantes entre os autômatos envolvidos, detectando se cada transição é nova (apenas acrescenta as transições que ainda não estão definidas). O final do algoritmo é marcado por funções que verificam a rede de autômatos criada, de acordo com a função $VerifyAutomata$. Ao término, caso não tenha ocorrido nenhuma indefinição, a representação correspondente entre os formalismos é criada e pode ser usada em ferramentas de compilação e solução de descritores Markovianos existentes. A seguir é mostrado um exemplo de conversão de uma PEPA net para uma SAN.

3.4.3 Exemplo de tradução

Esta seção apresenta dois modelos PEPA net: *MobileAgents* e *ClassifiedAgents*. Todos os eventos estão sendo criados de acordo com o nome utilizado pelo modelador sendo necessário, algumas vezes, alterar o nome internamente para refletir o fato que o um novo evento é necessário dentro do modelo SAN (melhor explicado a partir das figuras utilizadas nos exemplos de tradução). O modelo *MobileAgents* foi explicado juntamente com o formalismo das PEPA nets e está descrito na Seção 2.5.2.

A Figura 3.5 mostra a estrutura do modelo *MobileAgents* e seu correspondente modelo descrito em SAN. Ao comparar os modelos percebe-se que ambos fornecem auxílios visuais para percepção dos relacionamentos e as atividades desempenhadas por cada elemento do modelo. A única diferença é a forma com que ambos são resolvidos computacionalmente. Em SAN, o modelo gera um descritor Markoviano e é resolvido através das formas usuais de MVD existentes enquanto que o modelo PEPA net necessita executar uma ferramenta de solução matemática como *Matematica*, *Maple* ou *Octave*, que podem ser utilizadas para este propósito.

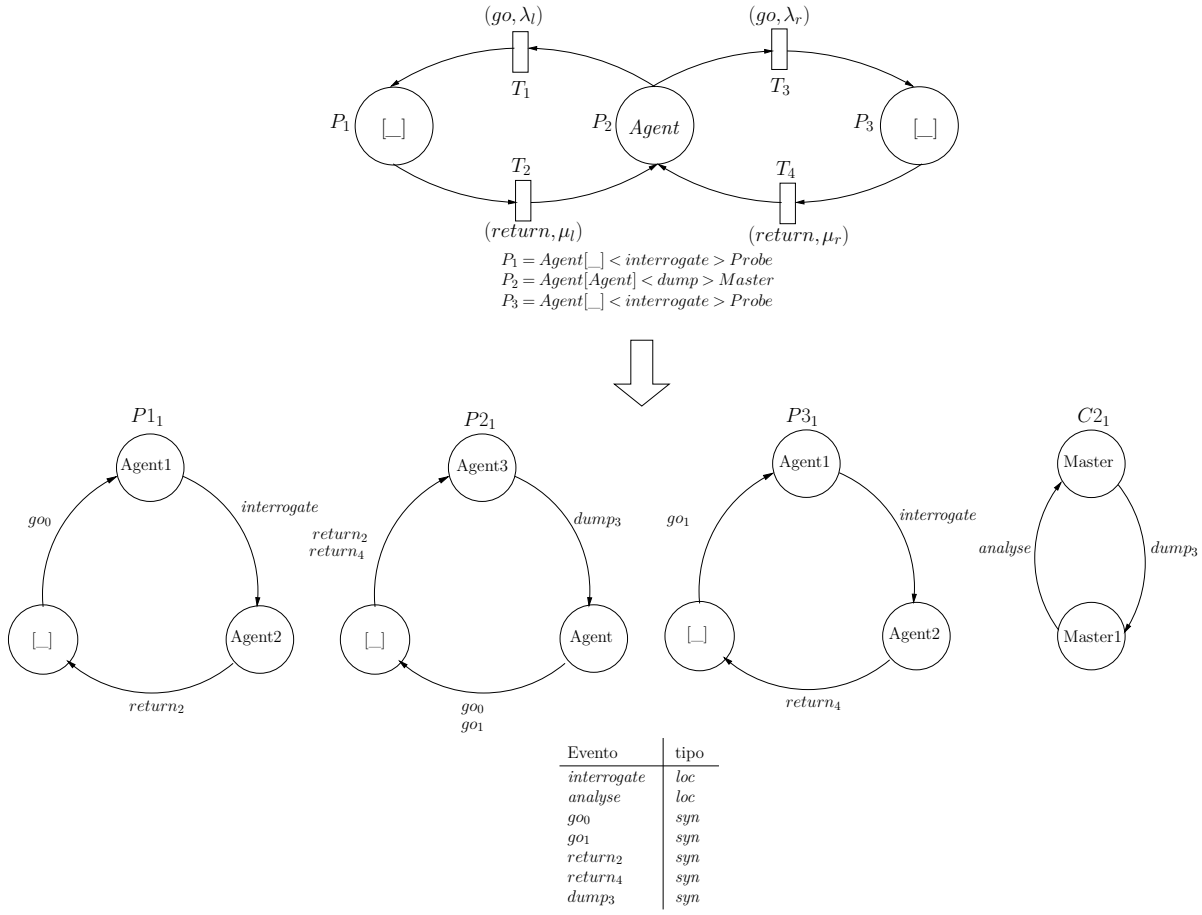


Figura 3.5: Modelo *MobileAgents* (ver Figura 2.8) em PEPA nets traduzido para SAN.

O próximo exemplo considerado é baseado no anterior e chama-se *ClassifiedAgents*. Este modelo possui três lugares e duas classificações para agentes: $AgentA7$ and $AgentA8$. O sistema representa agentes do tipo $AgentA7$ que podem baixar (*download*) ou salvar (*save*) informações e ainda se comportarem como $AgentB7$, conforme mostrado a seguir.

$$
\begin{aligned}
AgentA7 &\stackrel{def}{=} (download, r_1).AgentB7 + (save, r_2).AgentB7 \\
AgentB7 &\stackrel{def}{=} (move, s).AgentA7 \\
AgentA8 &\stackrel{def}{=} (select, r_3).AgentB8 + (process, r_4).AgentB8 \\
AgentB8 &\stackrel{def}{=} (move, s).AgentA8
\end{aligned}
$$

$$
\begin{aligned}
Remote &\stackrel{def}{=} (download, \top).Remote \\
Socket &\stackrel{def}{=} (save, \top).Socket + (select, \top).Socket \\
MI5 &\stackrel{def}{=} (process, \top).MI5
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{P1} &\stackrel{def}{=} Remote \underset{\mathcal{L}_1}{\bowtie} (AgentA7[AgentA7] \parallel AgentA7[AgentA7]) \\
\mathbf{P2} &\stackrel{def}{=} Socket \underset{\mathcal{L}_2}{\bowtie} (AgentA7[\_] \parallel AgentA7[\_] \parallel AgentA8[\_]) \\
\mathbf{P3} &\stackrel{def}{=} MI5 \underset{\mathcal{L}_3}{\bowtie} AgentA8[AgentA8]
\end{aligned}
$$

onde $\mathcal{L}_1$ é o conjunto $\{ download, save \}$, $\mathcal{L}_3$ é o conjunto $\{ select, process \}$ e $\mathcal{L}_2$ é $\mathcal{L}_1 \cup \mathcal{L}_3$.

Como mostrado pela Figura 3.6, a rede possui três lugares estáticos do tipo *stateless* chamados *Remote*, *Socket* e *MI5*. O lugar *Remote* permite que agentes troquem informações com agentes do lugar chamado *Socket*, que podem salvar ou selecionar (*select*) informações. O lugar chamado *MI5* é responsável por processar informações. A figura também mostra a tradução para SAN com todos estados, eventos e transições que foram criadas bem como seus tipos correspondentes. Os eventos sincronizantes chamados $move_{0..9}$ possuem uma propriedade particular: eles são disparados em diferentes autômatos e suas ocorrências fazem com que agregações semânticas sejam difíceis de serem aplicadas, pois sincronizam os autômatos sempre de dois em dois, simbolizando que o lugar ficou vago ou foi ocupado por outra marca (no caso, agente).

O nome escolhido para cada autômato reflete o número de componentes em cada lugar. O lugar $Remote$ possui dois componentes do tipo $AgentA7$ enquanto que $Socket$ possui três: dois do tipo $AgentA7$ e um do tipo $AgentA8$. O lugar $MI5$ possui apenas um agente do tipo $AgentA8$, fazendo com que cada autômato seja chamado de $P1_1$, $P2_1$, $P2_2$, $P2_3$ e $P3_1$, respectivamente.

A tradução realizada transforma cada lugar em um autômato representando os aspectos comportamentais dos componentes envolvidos bem como a movimentação das marcas dentro da rede. Os autômatos precisam guardar a informação que um lugar não possui nenhuma marca em um estado chamado [_]. As marcas trocam de lugares de forma sincronizada através do evento $move$, que sincroniza a saída de um lugar com a chegada em outro lugar. As transições locais são representadas no autômato de forma similar à modelada em PEPA nets, através de diferentes eventos ($save$, $select$, $download$ ou $process$).

Uma vantagem de se traduzir de PEPA nets para SAN, ou para qualquer formato que descreve um descritor Markoviano, é não precisar armazenar a matriz de transição. A tradução efetuada nesta seção gerou a mesma matriz correspondente sem perder informações tanto visuais quanto qualitativas (quanto aos nomes de eventos e lugares). Os exemplos mostrados permitiram constatar que alguns eventos encontram-se entrelaçados, ou seja, disparam a movimentação das marcas de forma sincronizada simbolizando a desocupação de um lugar e a movimentação para um outro lugar na rede.

Dando seguimento à tradução, uma tarefa que pode ser realizada é aplicar formas de reduzir os estados locais, transições ou eventos de forma que permaneçam representando o mesmo comportamento e possibilitando análises quantitativas em menos tempo. O objetivo neste contexto foi o de demonstrar através de exemplos a existência de representações correspondentes entre formalismos, apresentando um estudo de caso onde modelagens em PEPA nets são traduzidas para SAN sem perda de informação. Com estes conhecimentos é possível dar continuidade ao capítulo discutindo meios mais alto-nível (mais abstratos) para comunicação e interação de subsistemas.

3.5 Comunicação e interação

Conforme mencionado anteriormente, a forma de interação entre os componentes individuais de um sistema é através de comunicações ou sincronismos. Apesar de existirem trocas de informações

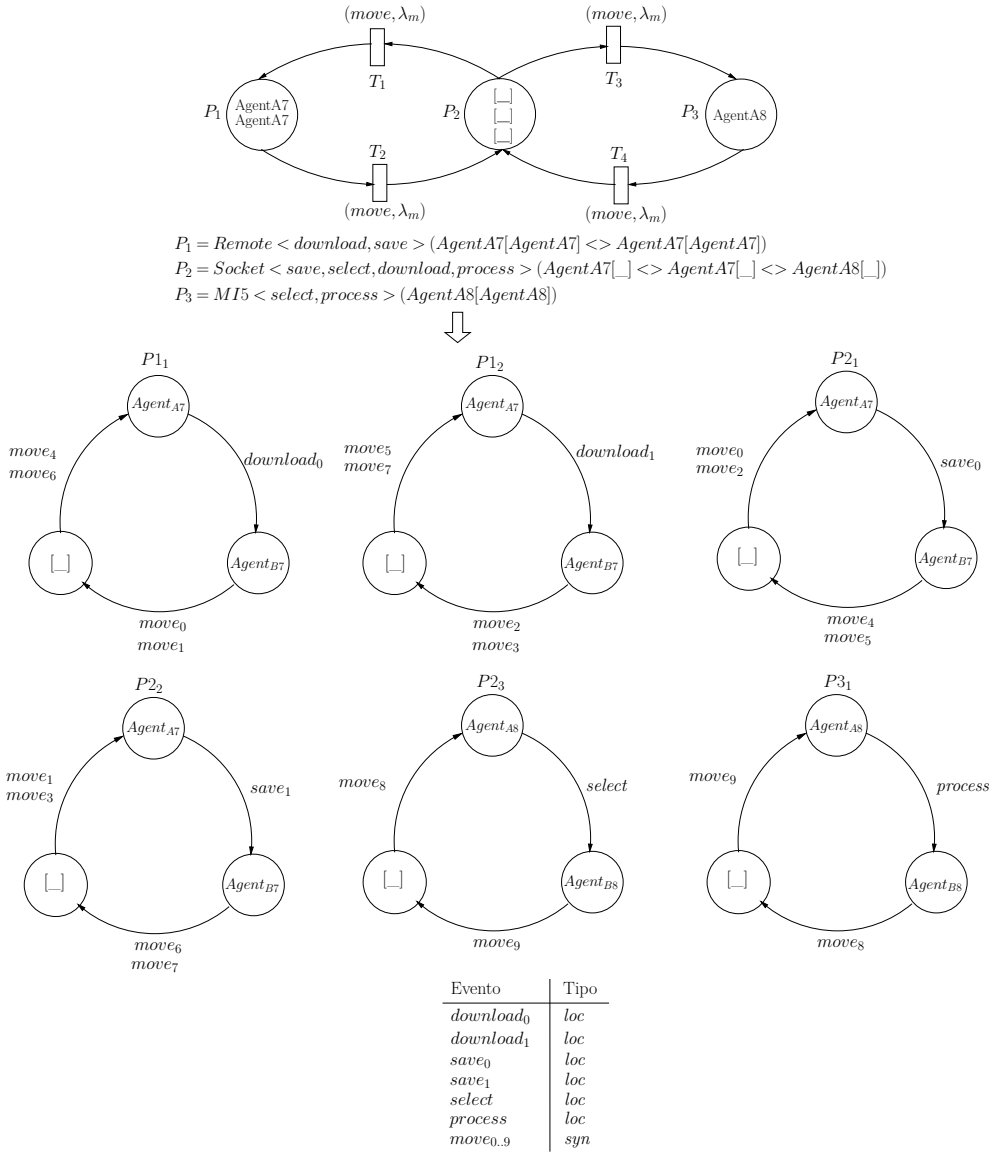


Figura 3.6: Modelo *ClassifiedAgents* em PEPA nets traduzido para SAN.

locais, os elementos do sistema cooperam de forma global, sincronizando atividades quando necessário. Cada componente do sistema pode ser visto como um processo, um autômato, um módulo, uma partição ou qualquer outra denominação. Este corresponde à menor entidade do sistema e constitui-se em um bloco onde escolhe-se que este não será mais dividido em partes menores. A seguir, explica-se como os componentes interagem e suas implicações no descritor Markoviano.

A Tabela 3.2 explica os principais elementos de decomposição dos formalismos de solução. Percebe-se, pela tabela, a existência de elementos em comum, que desempenham as mesmas funções quando um descritor Markoviano é definido. Se em SAN tem-se transições com eventos locais ou sincronizantes, em PEPA tem-se processos, ações locais e cooperação, e assim por diante. No nível da definição do descritor, todas as matrizes que representam estas comunicações são correspondentes e realizam as mesmas atividades neste contexto.

Tabela 3.2: Principais elementos existentes em alguns formalismos de solução.

Formalismo	Permite Composição	Menor Unidade	Comunicação Local	Comunicação Global
Cadeias de Markov	não	Estado Global	não	Transição Global
Redes de Petri	sim	Lugares/Estados	Transição	Transição
SAN	sim	Autômato	Transição Local	Transição Sincronizante
SGSPN	sim	Módulo/Partição	Transição Local	Transição Sincronizante
PEPA	sim	Processo	Ação Local	Cooperação entre Processos

Sem perda de generalidade, consideram-se apenas estados, transições e eventos em PMC. Um componente possui uma lista enumerável de estados, chamados também de estados locais. Quando observa-se a totalidade dos componentes em um sistema e combina-se todos os estados entre si, gera-se o Produto do Espaço de Estados, definindo seus estados globais. Supondo que todos, ou uma grande maioria, dos estados são atingíveis (sem considerar os estados inatingíveis que podem ser potencialmente produzidos no produto dos estados locais) e que a MC correspondente é ergódica (com estados finitos, aperiódica e irredutível) [STE94], uma transição pode ocorrer de acordo com um ou mais eventos que pode ter taxa constante ou apresentar uma taxa de ocorrência ou frequência. Esta classificação é importante para uma maior compreensão das características de PMC, onde observa-se as similaridades existentes com outros formalismos estruturados, em particular Redes de Autômatos Estocásticos, Álgebras de Processos e Redes de Petri Estocásticas.

A Figura 3.7 mostra uma representação gráfica de PMC. O objetivo é determinar o descritor Markoviano correspondente a ser multiplicado pelo vetor de probabilidades através das informações contidas nos componentes e transições do modelo. O que potencializa o uso de PMC para a descrição de sistemas é o fato de possuir simples primitivas de descrição e outros recursos de modelagem tais como taxas funcionais e sincronizações. Qualquer formalismo estruturado que gere um descritor Markoviano é suscetível de ser tratado com as opções de solução existentes de PMC, por exemplo, usando-se o Algoritmo *Shuffle* ou *Split* (maiores detalhes no Capítulo 4).

Esta seção apresentou um formalismo estruturado que agrega definições de outros formalismos de descrição presentes na literatura, auxiliando o processo de solução de modelagens de sistemas. Seu ponto forte reside na modularização dos componentes, mapeando os comportamentos independentes e de sincronização de atividades. A abstração do sistema em modelos deste tipo facilita a compreensão das operações que o sistema desempenha e permite análises de grande porte, uma vez que são utilizados mecanismos otimizados de MVD.

3.6 Exemplo de modelagem abstrata

O exemplo de tradução da Seção 3.4 mostrou como fazer para converter uma modelagem baseada em PEPA nets em um formato usado pelo formalismo de SAN que em última análise cria um descritor Markoviano. Entretanto, ao invés de utilizar um formalismo ou o outro, pode-se abstrair os problemas e pensar apenas nas entidades ou módulos envolvidos e como estes sincronizam

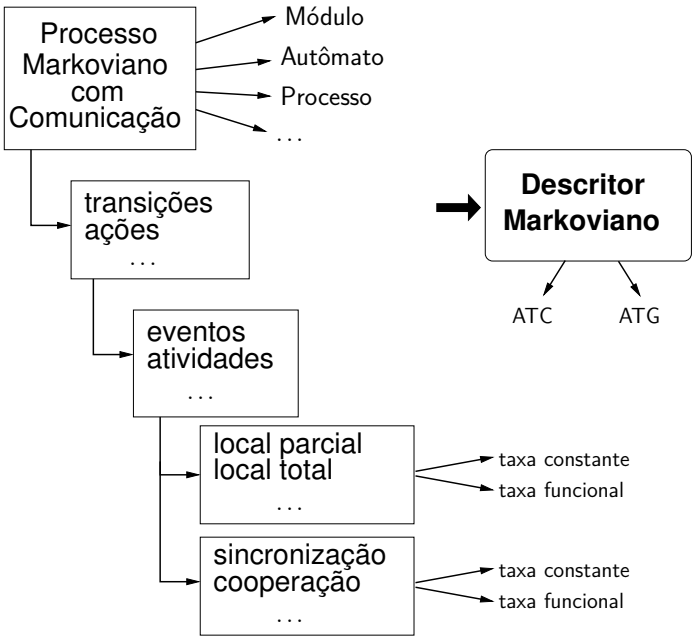


Figura 3.7: Relacionamentos de PMC e primitivas de modelagem de formalismos estruturados.

atividades e produzem comunicações entre si. Como exemplo de conversão, consideram-se *blocos* em um sistema qualquer de computação onde deseja-se inferir o grau de *acoplamento* [YOU79], ou seja, deseja-se quantificar a comunicação entre as partes do sistema. O ideal, ao construir um *software*, é construir blocos[1] contendo codificações de funcionalidades de um sistema, os quais desempenham tarefas locais de forma independente. Entretanto, algumas vezes, é necessário trocar mensagens com outros blocos ou esperar algum resultado produzido por outras partes do sistema.

Para o caso de uma modelagem abstrata deste problema, considera-se cada parte do sistema como sendo um módulo, em última análise possuidor de um conjunto de classes associadas e, consequentemente, métodos associados e potencialmente instâncias de outros objetos auxiliares. O fato importante é que existem comportamentos locais e globais neste sistema, sendo plenamente adaptável para a modelagem proposta por este capítulo. Considera-se que o sistema possui quatro módulos, denominados *Entrada*, *Calcula*, *Relatório* e *Publica*, melhor explicados a seguir:

1. *Entrada*: este bloco é responsável por todas as entradas de dados no *software*, guardando as estruturas de dados necessárias, ou seja, qualquer novo dado requisitado pelo sistema é imediatamente comunicado a esta parte, que os salva em memória, por exemplo;

2. *Calcula*: utiliza os dados existentes na entrada para calcular médias e outras estatísticas (os blocos definidos neste problema realizam operações estatísticas para efeitos de modelagem, entretanto, poderiam desempenhar quaisquer outras atividades onde existissem comunicações entre os blocos definidos);

[1] Usa-se *blocos* para distinguir a palavra *módulo*, utilizada neste contexto para descrever uma parte da modelagem abstrata que pode equivaler a um autômato, processo, partição, etc, como explicado anteriormente.

3. *Relatório*: a partir dos cálculos efetuados, utiliza os dados do sistema para construir relatórios;
4. *Publica*: este bloco utiliza o término da parte de cálculos e relatórios para publicar as informações previamente desenvolvidas.

Este problema abstrai a parametrização, ou seja, assume que as frequências de comunicação entre os blocos estão sendo capturadas em outra parte, mas serão eventualmente usadas para compor as taxas do sistema. Cada bloco deste *software* pode ser representado como um módulo, que desempenha atividades internas e que, às vezes, comunica requisições ou informa processamentos aos outros módulos. Nota-se que os módulos possuem uma interdependência, ou seja, necessitam de informações presentes em outros módulos. Este comportamento pode ser representado através da utilização de eventos sincronizantes e a parte que dita que o módulo desempenha atitudes independentes pode ser representada através de eventos locais simples, observando ou não o estado de outros módulos.

Este exemplo de sistema mostrou a utilização de abstração ao modelar o sistema em questão bem como foi usado para extração de índices que atestam a quantidade de comunicação existente entre os blocos do *software*. Nota-se a importância de se descrever um problema utilizando-se o mínimo de propriedades e características para permitir um maior controle quando este for representado através de uma descrição de alto nível. O sistema avaliado pode ser descrito através de uma linguagem que construirá um descritor Markoviano contendo o conjunto de informações quantitativas demonstrando o nível de comunicação entre os blocos. Esta informação pode ser utilizada para reduzir a quantidade de comunicação entre os blocos do *software* significando menos defeitos e maior produtividade por parte dos programadores do sistema (uma possível interpretação). Uma outra vantagem de se quantificar estas informações é permitir um maior reuso dos componentes de *software* que fazem parte de um sistema maior, promovendo um menor custo do projeto como um todo.

3.7 Discussão

O uso de Álgebra Tensorial Clássica (ATC) ou Generalizada (ATG) potencializou que outros formalismos estruturados fossem definidos com a vantagem de não ser preciso gerar a matriz de transições e sim acessá-la implicitamente através da utilização das propriedades definidas pelas álgebras. Observa-se, cada vez mais, a necessidade de se centralizar os conceitos que usam tais álgebras para extração de índices de desempenho. É através dos Processos Markovianos com Comunicação (PMC) que as principais definições serão concentradas e melhor formalizadas, especificando os próximos passos necessários para serem pesquisados. Tais processos auxiliam na abstração de realidades, onde os detalhes são mapeados para componentes indivisíveis que operam de forma independente ou comunicando suas trocas de estados com outros componentes do mesmo sistema. Trata-se de uma proposta mais geral de definição e concepção de sistemas Markovianos com vistas à análise de desempenho e descoberta de propriedades quantitativas.

O presente capítulo discutiu formas de maior abstração ao se compor sistemas de forma estruturada e utilizou os conceitos de Processos Markovianos com Comunicação para este propósito. Esta

visão composicional de descrição de sistemas com primitivas simples de comunicação é observada em múltiplos formalismos estruturados com representação correspondente de descritor Markoviano. Tais processos auxiliam na abstração para decomposição de sistemas para inferência de índices de desempenho. A sua contribuição está baseada no fato que possibilita que conceitos chave sejam observados e definidos em sistemas, agregados em uma mesma definição. Ao modelar os sistemas pensando nos seus módulos (ou partições, elementos, etc), é possível capturar sua operação principal, instanciando elementos que representam o comportamento individual e em conjunção com suas outras partes.

O objetivo da representação de sistemas através de PMC é o de permitir a construção de descritores Markovianos válidos para descreverem sistemas contendo múltiplos componentes com múltiplas interações e comunicações. O próximo capítulo trata da solução de tais descritores tanto para Álgebra Tensorial Clássica quanto Generalizada e descreve os principais algoritmos de MVD.

4. SOLUÇÃO DE DESCRITORES MARKOVIANOS

Este capítulo abordará a multiplicação de um vetor de probabilidade por uma estrutura mais complexa do que uma matriz esparsa, ou seja, por um descritor Markoviano. Quando esta estrutura é multiplicada por um vetor inicial de probabilidades inúmeras vezes, o método pode ou não convergir para um vetor solução que encontra-se na estacionariedade, ou seja, um vetor que atingiu o estado de equilíbrio. Em todos os formalismos estruturados com representação tensorial existem operações que são efetuadas para gerar implicitamente um Gerador Infinitesimal [STE94], ou matriz de transição correspondente à MC subjacente. A seguir serão definidos os principais conceitos de Álgebra Tensorial Clássica e Generalizada nas Seções 4.1 e 4.2, bem como suas propriedades. O capítulo continua com a definição de descritores tensoriais na Seção 4.3 e finaliza com os principais algoritmos existentes de MVD na Seção 4.4.

4.1 Álgebra Tensorial Clássica – ATC

Para resolver estruturas tensoriais é importante definir as principais operações existentes na Álgebra Tensorial. No contexto deste trabalho, as operações mais importantes são o produto tensorial e a soma tensorial, melhor explicadas a seguir.

4.1.1 Produto tensorial clássico

Por exemplo, sejam as matrizes A e B definidas como:

$$A = \begin{pmatrix} a_{11}\, a_{12} \\ a_{21}\, a_{22} \end{pmatrix} B = \begin{pmatrix} b_{11}\, b_{12}\, b_{13} \\ b_{21}\, b_{22}\, b_{23} \\ b_{31}\, b_{32}\, b_{33} \end{pmatrix}$$

O produto tensorial de $C = A \otimes B$ é igual a:

$$C = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} = \left(\begin{array}{ccc|ccc} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} & a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} & a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} \end{array}\right)$$

O produto tensorial $C = A \otimes B$ é definido algebricamente pela atribuição do valor $a_{i,j}b_{k,l}$ ao elemento dentro da posição (k,l) do bloco (i,j), *i.e.*[1]:

[1] Onde α_n e β_n correspondem à dimensão das matrizes.

$$c_{[i,k],[j,l]} = a_{i,j} b_{k,l} \quad \text{onde } i \in [1..\alpha_1],\ j \in [1..\alpha_2],\ k \in [1..\beta_1] \text{ e } l \in [1..\beta_2] \tag{4.1}$$

Esta representação de elementos de uma matriz corresponde ao produto tensorial realizado sobre os elementos $c_{[i,k],[j,l]}$ que estão em ordem lexicográfica de acordo com os seus índices (ou seja, na ordem que foram originalmente definidos).

4.1.2 Soma tensorial clássica

A soma tensorial utiliza um produto tensorial para ser calculada. Sejam duas matrizes quadradas A e B, sua soma tensorial é definida como a soma dos fatores normais de cada matriz, de acordo com a seguinte fórmula:

$$A \oplus B = (A \otimes I_{n_B}) + (I_{n_A} \otimes B) \tag{4.2}$$

onde I_{n_A} e I_{n_B} correspondem a matrizes identidade de dimensões iguais às das matrizes A e B respectivamente. Logo, um fator normal é dito um produto tensorial de uma matriz por uma ou mais identidades, dependendo do número de matrizes na soma tensorial. Por exemplo, sejam A e B matrizes definidas como:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

A soma tensorial definida por $C = A \oplus B$ é igual a:

$$C = \left(\begin{array}{ccc|ccc} a_{11} & 0 & 0 & a_{12} & 0 & 0 \\ 0 & a_{11} & 0 & 0 & a_{12} & 0 \\ 0 & 0 & a_{11} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22} & 0 & 0 \\ 0 & a_{21} & 0 & 0 & a_{22} & 0 \\ 0 & 0 & a_{21} & 0 & 0 & a_{22} \end{array}\right) + \left(\begin{array}{ccc|ccc} b_{11} & b_{12} & b_{13} & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 & 0 \\ b_{31} & b_{32} & b_{33} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & b_{11} & b_{12} & b_{13} \\ 0 & 0 & 0 & b_{21} & b_{22} & b_{23} \\ 0 & 0 & 0 & b_{31} & b_{32} & b_{33} \end{array}\right)$$

$$C = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & 0 & 0 \\ b_{21} & a_{11}+b_{22} & b_{23} & 0 & a_{12} & 0 \\ b_{31} & b_{32} & a_{11}+b_{33} & 0 & 0 & a_{12} \\ \hline a_{21} & 0 & 0 & a_{22}+b_{11} & b_{12} & b_{13} \\ 0 & a_{21} & 0 & b_{21} & a_{22}+b_{2,2} & b_{23} \\ 0 & 0 & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right)$$

Seja $\delta_{i,j}$ o elemento da linha i e coluna j de uma matriz identidade (ou seja, $\delta_{ij} = 1$, se $i = j$, senão $\delta_{ij} = 0$). A soma tensorial $C = A \oplus B$ é definida algebricamente pela atribuição do valor $a_{i,j}\delta_{k,l} + \delta_{i,j}b_{k,l}$ ao elemento da posição (k,l) do bloco (i,j), *i.e.*:

$$c_{[i,k],[j,l]} = a_{i,j}\delta_{k,l} + \delta_{i,j}b_{k,l} \quad \text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \tag{4.3}$$

O operador produto tensorial ($\otimes$) tem precedência sobre o operador da soma tensorial ($\oplus$) e estes dois operadores tensoriais, por sua vez, também têm maior precedência sobre os operadores tradicionais de multiplicação e soma ($\times$ e $+$).

4.1.3 Propriedades

As propriedades da ATC de interesse são listadas a seguir. Suas demonstrações estão descritas em [FER98b].

- Associatividade da soma e do produto tensorial:

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C \tag{4.4}$$

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{4.5}$$

- Distributividade com relação à soma convencional:

$$(A + B) \otimes (C + D) = (A \otimes C) + (B \otimes C) + (A \otimes D) + (B \otimes D) \tag{4.6}$$

- Compatibilidade com a multiplicação convencional:

$$(A \times B) \otimes (C \times D) = (A \otimes C) \times (B \otimes D) \tag{4.7}$$

- Compatibilidade com a transposição de matrizes:

$$(A \otimes B)^T = A^T \otimes B^T \tag{4.8}$$

- Compatibilidade com a inversão de matrizes (se A e B são matrizes inversíveis):

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{4.9}$$

- Decomposição em fatores normais:

$$A \otimes B = (A \otimes I_{n_B}) \times (I_{n_A} \otimes B) \tag{4.10}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$(A \times B) \otimes I_n = (A \otimes I_n) \times (B \otimes I_n) \tag{4.11}$$

$$I_n \otimes (A \times B) = (I_n \otimes A) \times (I_n \otimes B) \tag{4.12}$$

- Comutatividade dos fatores normais:

$$(A \otimes I_{n_B}) \times (I_{n_A} \otimes B) = (I_{n_A} \otimes B) \times (A \otimes I_{n_B}) \tag{4.13}$$

- Pseudo-comutatividade (onde σ é uma permutação sobre o intervalo $[1..N]$ e P_σ) [FER98b]:

$$\bigotimes_{k=1}^{N} A^{(k)} = P_\sigma \times \left[\bigotimes_{k=1}^{N} A^{(\sigma_k)} \right] \times {P_\sigma}^T \tag{4.14}$$

4.2 Álgebra Tensorial Generalizada – ATG

A Álgebra Tensorial Generalizada é uma extensão da Álgebra Tensorial Clássica, e tem como principal objetivo permitir a utilização de objetos que são funções discretas sobre linhas de uma matriz. Trabalha-se nas matrizes com elementos passíveis de avaliações diferentes, ou seja, tem-se uma matriz que pode ter diferentes instâncias de acordo com uma avaliação.

A diferença fundamental da ATG com relação à ATC é a introdução do conceito de *elementos funcionais*. Entretanto, uma matriz pode ser composta de elementos constantes (pertencentes a $\mathbb{R}$) ou de elementos funcionais. Um elemento funcional é uma função real dos índices de linha de uma ou mais matrizes, *i.e.*, o domínio dessa função é $\mathbb{R}^n$ e seu contra–domínio é $\mathbb{R}$.

Um elemento funcional $b(\mathcal{A})$ é dito *dependente* da matriz $\mathcal{A}$ se algum índice de linha da matriz $\mathcal{A}$ pertencer ao conjunto de parâmetros desse elemento funcional. Uma matriz que contém ao menos um elemento funcional dependente da matriz $\mathcal{A}$ é dita *dependente* da matriz $\mathcal{A}$.

Assim como a ATC, a ATG é definida por dois operadores matriciais:

- produto tensorial generalizado $\underset{g}{\otimes}$;
- soma tensorial generalizada $\underset{g}{\oplus}$.

A notação definida na Seção 4.1 continua sendo válida para as *matrizes constantes* (*i.e.*, matrizes sem elementos funcionais). As matrizes com elementos funcionais, denominadas *matrizes funcionais*, são descritas com o uso da notação seguinte:

Sejam:

a_k índice de linha k da matriz A;

$A(\mathcal{B},\mathcal{C})$ matriz funcional A que possui como parâmetros as matrizes B e C;

$a_{ij}(\mathcal{B},\mathcal{C})$ elemento funcional (i,j) da matriz $A(\mathcal{B},\mathcal{C})$;

$A(b_k,\mathcal{C})$ matriz funcional $A(\mathcal{B},\mathcal{C})$ na qual o índice de linha da matriz B já é conhecido e igual a k (essa matriz é considerada dependente da matriz C somente);

$a_{ij}(b_k,\mathcal{C})$ elemento funcional (i,j) da matriz $A(b_k,\mathcal{C})$;

$A(b_k,c_l)$ matriz funcional $A(\mathcal{B},\mathcal{C})$ na qual os índices de linha das matrizes B e C já são conhecidos e iguais a k e l respectivamente (uma vez que todos os parâmetros da matriz são conhecidos, ela é considerada uma matriz constante);

$a_{ij}(b_k,c_l)$ elemento constante (elemento funcional de valor determinado) (i,j) da matriz $A(b_k,c_l)$;

$\ell_k(A)$ matriz com todos os elementos iguais a zero, exceto aqueles pertencentes à linha k que é igual à linha k da matriz A ($A=\sum_{k=1}^{n_A}\ell_k(A)$);

$A(\mathcal{B})\underset{g}{\otimes}B(\mathcal{A})$ produto tensorial generalizado entre as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$;

$A(\mathcal{B})\underset{g}{\oplus}B(\mathcal{A})$ soma tensorial generalizada entre as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$.

4.2.1 Produto tensorial generalizado

Sejam por exemplo duas matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$ dadas por:

$$A(\mathcal{B})=\begin{pmatrix} a_{11}[\mathcal{B}] & a_{12}[\mathcal{B}] \\ a_{21}[\mathcal{B}] & a_{22}[\mathcal{B}] \end{pmatrix} \qquad B(\mathcal{A})=\begin{pmatrix} b_{11}[\mathcal{A}] & b_{12}[\mathcal{A}] & b_{13}[\mathcal{A}] \\ b_{21}[\mathcal{A}] & b_{22}[\mathcal{A}] & b_{23}[\mathcal{A}] \\ b_{31}[\mathcal{A}] & b_{32}[\mathcal{A}] & b_{33}[\mathcal{A}] \end{pmatrix}$$

O produto tensorial definido por $C=A(\mathcal{B})\underset{g}{\otimes}B(\mathcal{A})$ é igual a:

$$C=\left(\begin{array}{ccc|ccc} a_{11}(b_1)b_{11}(a_1) & a_{11}(b_1)b_{12}(a_1) & a_{11}(b_1)b_{13}(a_1) & a_{12}(b_1)b_{11}(a_1) & a_{12}(b_1)b_{12}(a_1) & a_{12}(b_1)b_{13}(a_1) \\ a_{11}(b_2)b_{21}(a_1) & a_{11}(b_2)b_{22}(a_1) & a_{11}(b_2)b_{23}(a_1) & a_{12}(b_2)b_{21}(a_1) & a_{12}(b_2)b_{22}(a_1) & a_{12}(b_2)b_{23}(a_1) \\ a_{11}(b_3)b_{31}(a_1) & a_{11}(b_3)b_{32}(a_1) & a_{11}(b_3)b_{33}(a_1) & a_{12}(b_3)b_{31}(a_1) & a_{12}(b_3)b_{32}(a_1) & a_{12}(b_3)b_{33}(a_1) \\ \hline a_{21}(b_1)b_{11}(a_2) & a_{21}(b_1)b_{12}(a_2) & a_{21}(b_1)b_{13}(a_2) & a_{22}(b_1)b_{11}(a_2) & a_{22}(b_1)b_{12}(a_2) & a_{22}(b_1)b_{13}(a_2) \\ a_{21}(b_2)b_{21}(a_2) & a_{21}(b_2)b_{22}(a_2) & a_{21}(b_2)b_{23}(a_2) & a_{22}(b_2)b_{21}(a_2) & a_{22}(b_2)b_{22}(a_2) & a_{22}(b_2)b_{23}(a_2) \\ a_{21}(b_3)b_{31}(a_2) & a_{21}(b_3)b_{32}(a_2) & a_{21}(b_3)b_{33}(a_2) & a_{22}(b_3)b_{31}(a_2) & a_{22}(b_3)b_{32}(a_2) & a_{22}(b_3)b_{33}(a_2) \end{array}\right)$$

Os elementos da matriz A variam em função dos elementos da matriz B por isso a denominação $A(\mathcal{B})$, ocorrendo o mesmo para a matriz B, onde seus elementos variam em função da matriz A, ou seja, $B(\mathcal{A})$. O produto tensorial generalizado $C = A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A})$ é definido algebricamente pela atribuição do valor $a_{ij}(b_k)b_{kl}(a_i)$ ao elemento $c_{[ik][jl]}$, *i.e.*:

$$\begin{aligned} c_{[ik][jl]} &= a_{ij}(b_k)b_{kl}(a_i) \\ &\text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \end{aligned} \tag{4.15}$$

4.2.2 Soma tensorial generalizada

A soma tensorial generalizada é definida utilizando-se o conceito de matriz identidade com o produto tensorial generalizado da Equação 4.16:

$$A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A}) = (A(\mathcal{B}) \underset{g}{\otimes} I_{n_B}) + (I_{n_A} \underset{g}{\otimes} B(\mathcal{A})) \tag{4.16}$$

Sejam as matrizes $A(\mathcal{B})$ e $B(\mathcal{A})$ utilizadas para descrever o produto tensorial generalizado. A soma tensorial definida por $C = A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ é igual a:

$$C = \left(\begin{array}{ccc|ccc} a_{11}(b_1)+b_{11}(a_1) & b_{12}(a_1) & b_{13}(a_1) & a_{12}(b_1) & 0 & 0 \\ b_{21}(a_1) & a_{11}(b_2)+b_{22}(a_1) & b_{23}(a_1) & 0 & a_{12}(b_2) & 0 \\ b_{31}(a_1) & b_{32}(a_1) & a_{11}(b_3)+b_{33}(a_1) & 0 & 0 & a_{12}(b_3) \\ \hline a_{21}(b_1) & 0 & 0 & a_{22}(b_1)+b_{11}(a_2) & b_{12}(a_2) & b_{13}(a_2) \\ 0 & a_{21}(b_2) & 0 & b_{21}(a_2) & a_{22}(b_2)+b_{22}(a_2) & b_{23}(a_2) \\ 0 & 0 & a_{21}(b_3) & b_{31}(a_2) & b_{32}(a_2) & a_{22}(b_3)+b_{33}(a_2) \end{array}\right)$$

A soma tensorial generalizada $C = A(\mathcal{B}) \underset{g}{\oplus} B(\mathcal{A})$ é definida algebricamente pela atribuição do valor $a_{ij}(b_k)\delta_{kl} + b_{kl}(a_i)\delta_{ij}$ ao elemento $c_{[ik][jl]}$, *i.e.*:

$$\begin{aligned} c_{[ik][jl]} &= a_{ij}(b_k)\delta_{kl} \; + \; b_{kl}(a_i)\delta_{ij} \\ &\text{onde } i,j \in [1..n_A] \text{ e } k,l \in [1..n_B] \end{aligned} \tag{4.17}$$

4.2.3 Propriedades

Em ATG, são definidas as seguintes propriedades fundamentais [FER98b]:

- Distributividade do produto tensorial generalizado e relação à soma convencional de matrizes:

$$\begin{aligned} &[A(\mathcal{C},\mathcal{D}) + B(\mathcal{C},\mathcal{D})] \underset{g}{\otimes} [C(\mathcal{A},\mathcal{B}) + D(\mathcal{A},\mathcal{B})] = \\ &A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \; + \; A(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) \; + B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \; + \; B(\mathcal{C},\mathcal{D}) \underset{g}{\otimes} D(\mathcal{A},\mathcal{B}) \end{aligned} \tag{4.18}$$

- Associatividade do produto tensorial generalizado e da soma tensorial generalizada:

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\otimes} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\otimes} C(\mathcal{A},\mathcal{B}) \,] \tag{4.19}$$

$$[\, A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} B(\mathcal{A},\mathcal{C}) \,] \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) = A(\mathcal{B},\mathcal{C}) \underset{g}{\oplus} [\, B(\mathcal{A},\mathcal{C}) \underset{g}{\oplus} C(\mathcal{A},\mathcal{B}) \,] \tag{4.20}$$

- Distributividade com relação à multiplicação pela matriz identidade:

$$[\, A(\mathcal{C}) \times B(\mathcal{C}) \,] \underset{g}{\otimes} I_{n_C} = A(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \times B(\mathcal{C}) \underset{g}{\otimes} I_{n_C} \tag{4.21}$$

$$[\, I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \,] \times B(\mathcal{C}) = I_{n_C} \underset{g}{\otimes} A(\mathcal{C}) \times I_{n_C} \underset{g}{\otimes} B(\mathcal{C}) \tag{4.22}$$

- Decomposição em fatores normais I:

$$A \underset{g}{\otimes} B(\mathcal{A}) = I_{n_A} \underset{g}{\otimes} B(\mathcal{A}) \times A \underset{g}{\otimes} I_{n_B} \tag{4.23}$$

- Decomposição em fatores normais II:

$$A(\mathcal{B}) \underset{g}{\otimes} B = A(\mathcal{B}) \underset{g}{\otimes} I_{n_B} \times I_{n_A} \underset{g}{\otimes} B \tag{4.24}$$

- Pseudo-comutatividade:

$$A(\mathcal{B}) \underset{g}{\otimes} B(\mathcal{A}) = P_\sigma \times B(\mathcal{A}) \underset{g}{\otimes} A(\mathcal{B}) \times {P_\sigma}^T \tag{4.25}$$

- Decomposição em produto tensorial clássico:

$$A \underset{g}{\otimes} B(\mathcal{A}) = \sum_{k=1}^{n_A} \ell_k(A) \underset{g}{\otimes} B(a_k) \tag{4.26}$$

Uma vez descritas as operações existentes em Álgebra Tensorial Clássica e Generalizada, segue-se o capítulo com uma discussão sobre os descritores Markovianos.

4.3 Descritores Markovianos

Sistemas representados por Redes de Autômatos Estocásticos ou outros formalismos estruturados, permitem uma visão modular onde as interações entre os diferentes subsistemas são modeladas através de transições e eventos. As transições conectam os estados e os eventos, por sua vez, são

locais ou sincronizantes e mapeiam taxas constantes ou funcionais. O descritor Markoviano é o Gerador Infinitesimal $\tilde{\mathcal{Q}}$ da Cadeia de Markov quando as matrizes que capturam as transições entre os estados são expressas no formato tensorial, de acordo com a Equação 4.27. As operações que podem ser efetuadas (por exemplo, soma tensorial ou produto tensorial) reproduzem as interações e igualmente como os diferentes autômatos sincronizam o disparo de eventos.

$$\tilde{\mathcal{Q}} = \bigoplus_{i=1}^{N} \mathcal{Q}_l^{(i)} + \sum_{e \in E} \left(\bigotimes_{i=1}^{N} \mathcal{Q}_{e^+}^{(i)} + \bigotimes_{i=1}^{N} \mathcal{Q}_{e^-}^{(i)} \right) \tag{4.27}$$

onde
- N número total de autômatos na rede;
- E número total de eventos sincronizantes definidos na rede;
- $\mathcal{Q}_l$ são matrizes que representam a ocorrência e o ajuste da diagonal de eventos locais;
- $\mathcal{Q}_{e^+}$ são matrizes que representam a ocorrência de eventos sincronizantes;
- $\mathcal{Q}_{e^-}$ são matrizes que representam o ajuste diagonal dos eventos sincronizantes.

Somas tensoriais são na verdade fatores normais (produtos tensoriais efetuados com matrizes do tipo *identidade*) conforme discutido nas propriedades da Álgebra Tensorial. Com isso, simplifica-se a Equação 4.27 em termos de notação do descritor Markoviano para:

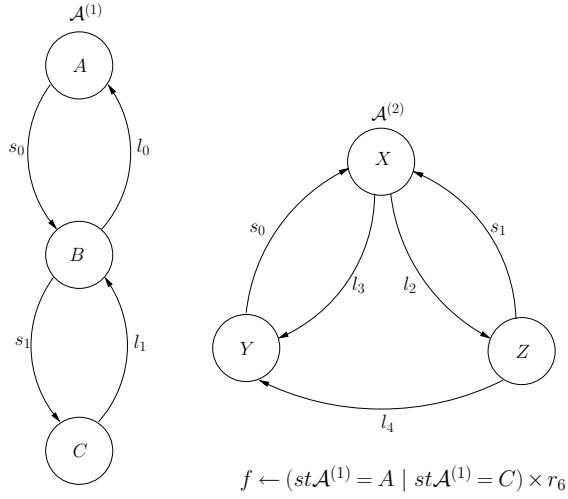
$$\tilde{\mathcal{Q}} = \sum_{j}^{|\mathcal{L}|} \bigotimes_{i=1}^{N} \mathcal{Q}_j^{(i)} \tag{4.28}$$

onde $\mathcal{L}$ é um conjunto de termos tensoriais do tipo $\{l, e^+, e^-\}$ com cardinalidade $|\mathcal{L}| = N + 2E$.

4.3.1 Descrição de eventos

A seguir, discute-se o efeito de cada tipo de evento em um descritor Markoviano e assinalam-se as implicações em se definir um modelo com eventos locais ou sincronizantes, com taxas constantes ou funcionais. A análise será iniciada pelos eventos locais (com taxas constantes e funcionais) e depois tratará dos eventos sincronizantes. O modelo de exemplo da Figura 4.1 possui cinco eventos locais, l_0, l_1, l_2, l_3 e l_4. No descritor Markoviano eles correspondem a duas matrizes (devido à existência de dois autômatos) de dimensão três (pois ambos autômatos possuem três estados).

Antes de começar a compor o descritor Markoviano com as taxas de ocorrência relacionadas aos eventos dos autômatos em questão são necessárias outras definições de base para o mapeamento para a MC correspondente. A rede da Figura 4.1 é composta por dois autômatos de três estados cada e possui eventos locais e sincronizantes com taxas constantes e funcionais. Para este caso, como existem dois autômatos de três estados cada, o seu PSS (seu Produto do Espaço de Estados, de acordo com a Seção 2.2.1) corresponde a $|\mathcal{X}| = 3 \times 3 = 9$, sendo este o produto cartesiano dos estados do autômato $\mathcal{A}^{(1)} = \{A, B, C\}$ pelos estados do autômato $\mathcal{A}^{(2)} = \{X, Y, Z\}$, ou seja, o conjunto $\mathcal{X} = \{AX, AY, AZ, BX, BY, BZ, CX, CY, CZ\}$.



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

Figura 4.1: Um exemplo de SAN para conversão na Cadeia de Markov correspondente.

		A			B			C		
		X	Y	Z	X	Y	Z	X	Y	Z
A	X									
	Y			$\ldots$						
	Z									
B	X									
	Y					$\ldots$				
	Z									
C	X									
	Y								$\ldots$	
	Z									

Figura 4.2: Matriz correspondente ao produto cartesiano dos estados dos autômatos $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$.

A Figura 4.2 mostra a matriz de transição resultante que corresponde ao Gerador Infinitesimal $\tilde{Q}$ da MC. Cabe ressaltar que esta matriz encontra-se vazia (sem indicação dos seus elementos) para efeitos de explicação do método. A forma correta de representá-la é preenchendo-se as células com as taxas que mostram a frequência com a qual troca-se de estado na MC. Cada linha e coluna da matriz corresponde a um estado global do modelo contido no conjunto $\mathcal{X}$.

A seguir, inicia-se a análise pelos eventos locais desta rede de autômatos estocásticos.

Parte local correspondente aos autômatos $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(2)}$:

$$\mathcal{Q}_l = \mathcal{Q}_l^{\mathcal{A}^{(1)}} \oplus \mathcal{Q}_l^{\mathcal{A}^{(2)}} = \begin{array}{c} \\ \text{A} \\ \text{B} \\ \text{C} \end{array} \begin{array}{c} \begin{array}{ccc} \text{A} & \text{B} & \text{C} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ r_2 & -r_2 & 0 \\ 0 & r_3 & -r_3 \end{pmatrix} \end{array} \oplus \begin{array}{c} \\ \text{X} \\ \text{Y} \\ \text{Z} \end{array} \begin{array}{c} \begin{array}{ccc} \text{X} & \text{Y} & \text{Z} \end{array} \\ \begin{pmatrix} -(r_5+r_4) & r_5 & r_4 \\ 0 & 0 & 0 \\ 0 & r_6 & -r_6 \end{pmatrix} \end{array}$$

Seja $A = \mathcal{Q}_l^{\mathcal{A}^{(1)}}$ e $B = \mathcal{Q}_l^{\mathcal{A}^{(2)}}$. Segundo a Equação 4.2, $A \oplus B = (A \otimes I_{n_B}) + (I_{n_A} \otimes B)$. É necessário calcular cada parte individualmente, começando por $A \otimes I_{n_B}$:

$$A \otimes I_{n_B} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 \end{array}\right)$$

Uma vez calculado $A \otimes I_{n_B}$, é necessário calcular $I_{n_A} \underset{g}{\otimes} B$ utilizando as propriedades da ATG para esse evento em particular, pois deve-se avaliar a função f:

$$I_{n_A} \underset{g}{\otimes} B = \left(\begin{array}{ccc|ccc|ccc} -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_6 & -r_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & -(r_5+r_4) & r_5 & r_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_6 & -r_6 \end{array}\right)$$

Cabe ressaltar que os elementos marcados em negrito, correspondem a avaliações de elementos funcionais que resultam no valor zero, pois essas linhas equivalem ao autômato $\mathcal{A}^{(1)}$ estar no estado B. Para o caso da função f definida no modelo da Figura 4.1, é retornada uma avaliação lógica para *falso* que é então convertida para o valor zero, e consequentemente, não definindo a taxa r_6 para essa linha (como seria a taxa de ocorrência normal para o caso da f retornar *verdadeiro*).

Somando-se os dois termos que foram previamente calculados, tem-se que:

$$A \oplus B = \left(\begin{array}{ccc|ccc|ccc} -(r_5+r_4) & r_5 & r_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_6 & -r_6 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline r_2 & 0 & 0 & -(r_5+r_4+r_2) & r_5 & r_4 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2 & 0 & 0 & -r_2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & r_3 & 0 & 0 & -(r_5+r_4+r_3) & r_5 & r_4 \\ 0 & 0 & 0 & 0 & r_3 & 0 & 0 & -r_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_3 & 0 & r_6 & -(r_3+r_6) \end{array}\right)$$

A seguir, calculam-se os termos tensoriais correspondentes aos eventos sincronizantes (também chamados de parte positiva) e ao ajuste da diagonal dos mesmos (ou parte negativa):

Parte sincronizante positiva da ocorrência do evento $\mathbf{s_0}$:

$$\mathcal{Q}_{s_{0+}} = \mathcal{Q}^{\mathcal{A}^{(1)}}_{s_{0+}} \otimes \mathcal{Q}^{\mathcal{A}^{(2)}}_{s_{0+}} = \begin{array}{c} \\ \mathsf{A} \\ \mathsf{B} \\ \mathsf{C} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \end{array} \\ \begin{pmatrix} 0 & r_0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \otimes \begin{array}{c} \\ \mathsf{X} \\ \mathsf{Y} \\ \mathsf{Z} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{X} & \mathsf{Y} & \mathsf{Z} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{s_0}$:

$$\mathcal{Q}_{s_{0-}} = \mathcal{Q}^{\mathcal{A}^{(1)}}_{s_{0-}} \otimes \mathcal{Q}^{\mathcal{A}^{(2)}}_{s_{0-}} = \begin{array}{c} \\ \mathsf{A} \\ \mathsf{B} \\ \mathsf{C} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \end{array} \\ \begin{pmatrix} -r_0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \otimes \begin{array}{c} \\ \mathsf{X} \\ \mathsf{Y} \\ \mathsf{Z} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{X} & \mathsf{Y} & \mathsf{Z} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -r_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{s_1}$:

$$\mathcal{Q}_{s_{1+}} = \mathcal{Q}^{\mathcal{A}^{(1)}}_{s_{1+}} \otimes \mathcal{Q}^{\mathcal{A}^{(2)}}_{s_{1+}} = \begin{array}{c} \\ \mathsf{A} \\ \mathsf{B} \\ \mathsf{C} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & r_1 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \otimes \begin{array}{c} \\ \mathsf{X} \\ \mathsf{Y} \\ \mathsf{Z} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{X} & \mathsf{Y} & \mathsf{Z} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r_1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{s_1}$:

$$\mathcal{Q}_{s_{1-}} = \mathcal{Q}^{\mathcal{A}^{(1)}}_{s_{1-}} \otimes \mathcal{Q}^{\mathcal{A}^{(2)}}_{s_{1-}} = \begin{array}{c} \\ \mathsf{A} \\ \mathsf{B} \\ \mathsf{C} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{A} & \mathsf{B} & \mathsf{C} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & -r_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \otimes \begin{array}{c} \\ \mathsf{X} \\ \mathsf{Y} \\ \mathsf{Z} \end{array} \begin{array}{c} \begin{array}{ccc} \mathsf{X} & \mathsf{Y} & \mathsf{Z} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -r_1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

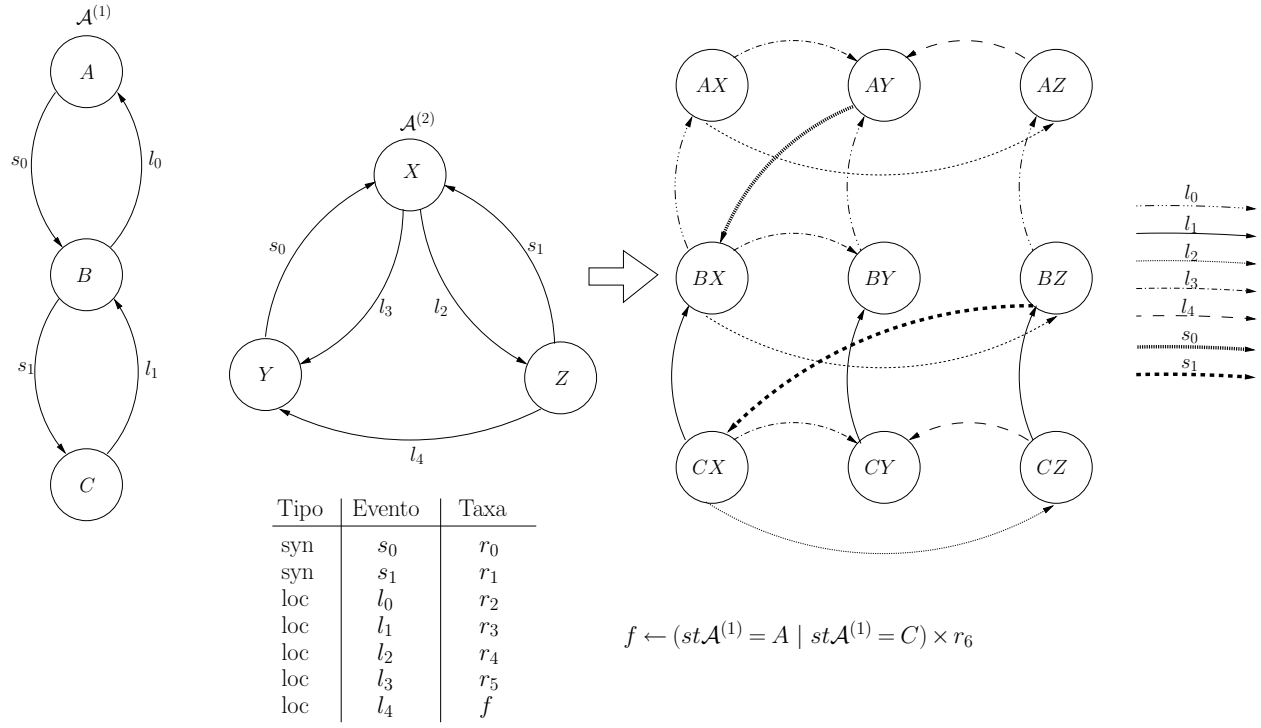
O Gerador Infinitesimal é dado por:

$$\tilde{\mathcal{Q}} = \mathcal{Q}_l + \left(\mathcal{Q}_{s_{0+}} + \mathcal{Q}_{s_{0-}}\right) + \left(\mathcal{Q}_{s_{1+}} + \mathcal{Q}_{s_{1-}}\right)$$

A Figura 4.3 mostra o Gerador Infinitesimal produzido a partir deste exemplo de conversão de uma SAN em uma Cadeia de Markov. Nota-se que esta matriz final já conta com os ajustes diagonais necessários (devido aos eventos sincronizantes negativos dos termos que foram criados) e a soma de cada linha resulta em zero.

		A			B			C		
		X	Y	Z	X	Y	Z	X	Y	Z
A	X	$-(r_5+r_4)$	r_5	r_4	0	0	0	0	0	0
	Y	0	$-r_0$	0	r_0	0	0	0	0	0
	Z	0	r_6	$-r_6$	0	0	0	0	0	0
B	X	r_2	0	0	$-(r_5+r_4+r_2)$	r_5	r_4	0	0	0
	Y	0	r_2	0	0	$-r_2$	0	0	0	0
	Z	0	0	r_2	0	0	$-(r_2+r_1)$	r_1	0	0
C	X	0	0	0	r_3	0	0	$-(r_5+r_4+r_3)$	r_5	r_4
	Y	0	0	0	0	r_3	0	0	$-r_3$	0
	Z	0	0	0	0	0	r_3	0	r_6	$-(r_3+r_6)$

Figura 4.3: Gerador Infinitesimal ($\tilde{\mathcal{Q}}$) do mapeamento de uma SAN para MC.



Tipo	Evento	Taxa
syn	s_0	r_0
syn	s_1	r_1
loc	l_0	r_2
loc	l_1	r_3
loc	l_2	r_4
loc	l_3	r_5
loc	l_4	f

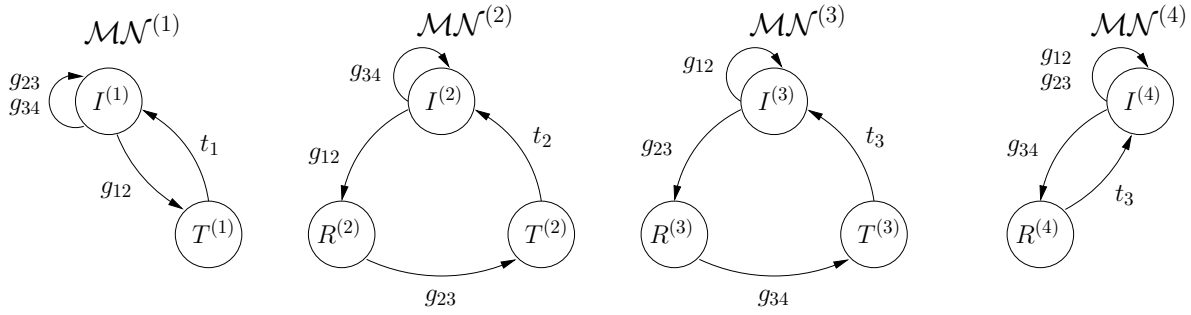
Figura 4.4: Mapeamento de uma SAN para sua Cadeia de Markov correspondente.

A Figura 4.4 mostra que para obter a MC correspondente a uma SAN é necessário iterar sobre o produto do espaço dos estados locais dos autômatos, produzindo todas as combinações possíveis. Isso evidencia o fato da modelagem por SAN ser mais compacta e compartimentalizada, como mencionado anteriormente. Entretanto, ao realizar essa combinação, pode-se potencialmente gerar estados que nunca serão atingidos. Esta figura mostra, na parte direita, diferentes tipos de linha (algumas pontilhadas, outras maiores, etc) para cada evento correspondente à SAN. Ressalta-se que em MC não existem diferenças quanto ao tipo de uma transição, *i.e.*, todas são indistinguíveis. Decidiu-se mostrar a cadeia desta forma para que fosse entendido o mapeamento de cada tipo de evento da SAN e sua transição correspondente na MC.

Nota-se que, para a solução de SAN, é suficiente guardar em memória apenas as pequenas matrizes que compõem o descritor, sendo desnecessário salvar o Gerador Infinitesimal $\tilde{\mathcal{Q}}$ correspondente à Cadeia de Markov. Essa é a principal vantagem de SAN sobre outros formalismos estruturados, pois baseia-se no uso da Álgebra Tensorial para armazenar implicitamente a matriz de transição e calcular o vetor de probabilidade estacionário ou transiente do qual são extraídos posteriormente os índices de desempenho. A obtenção deste vetor será melhor detalhada da Seção 4.4.

4.3.2 Exemplo

Este descritor está baseado no modelo *Redes Wireless ad hoc* (RW) para quatro nós (mostrado na Figura 4.5 e descrito na Seção 7.2.2). Este modelo possui quatro autômatos, sendo dois autômatos com dois estados e outros dois autômatos com três estados. O PSS deste modelo é igual a $|\mathcal{X}| = 2 \times 3 \times 3 \times 2 = 36$ estados com seis eventos onde $l = \{t_1, t_2\}$ corresponde aos eventos locais e o conjunto $e = \{t_3, g_{12}, g_{23}, g_{34}\}$ enumera os eventos sincronizantes.



Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	t_1	μ_1	syn	g_{12}	λ_{12}
loc	t_2	μ_2	syn	g_{23}	λ_{23}
syn	t_3	μ_3	syn	g_{34}	λ_{34}

Figura 4.5: Modelo *Redes Wireless ad hoc* com descritor Markoviano clássico para quatro nós.

Logo, considerando-se as matrizes que representam os eventos locais, os eventos sincronizantes positivos e os seus ajustes diagonais, é necessário um termo com somas tensoriais (correspondendo aos eventos locais) e oito termos com produtos tensoriais (por sua vez equivalendo-se aos eventos sincronizantes). Estes termos estão descritos a seguir:

Parte local[2] correspondente aos autômatos $\mathcal{MN}^{(1)}$ e $\mathcal{MN}^{(2)}$:

$$\mathcal{Q}_l = \mathcal{Q}_l^{\mathcal{MN}^{(1)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(2)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(3)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 0 & 0 \\ \mu_1 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_2 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{t_3}$:

$$\mathcal{Q}_{t_{3+}} = \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mu_3 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{t_3}$:

$$\mathcal{Q}_{t_{3-}} = \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\mu_3 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{12}}$:

$$\mathcal{Q}_{g_{12+}} = \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} 0 & \lambda_{12} \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{12}}$:

$$\mathcal{Q}_{g_{12-}} = \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{12} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

[2]Cabe ressaltar que os autômatos $\mathcal{MN}^{(3)}$ e $\mathcal{MN}^{(4)}$ não possuem eventos locais, logo estes não tem relevância no cálculo do Gerador Infinitesimal final. Por este motivo as matrizes correspondentes a estes autômatos encontram-se zeradas em $\mathcal{Q}_l$.

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{23}}$:

$$\mathcal{Q}_{g_{23+}} = \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} \lambda_{23} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{23}}$:

$$\mathcal{Q}_{g_{23-}} = \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{23} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante positiva da ocorrência do evento $\mathbf{g_{34}}$:

$$\mathcal{Q}_{g_{34+}} = \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} \lambda_{34} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Parte sincronizante negativa relativa ao ajuste diagonal do evento $\mathbf{g_{34}}$:

$$\mathcal{Q}_{g_{34-}} = \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(4)}} =$$

$$\begin{pmatrix} -\lambda_{34} & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

O Gerador Infinitesimal é dado por:

$$\tilde{\mathcal{Q}} = \mathcal{Q}_l + \left(\mathcal{Q}_{t_{3+}} + \mathcal{Q}_{t_{3-}}\right) + \left(\mathcal{Q}_{g_{12+}} + \mathcal{Q}_{g_{12-}}\right) + \left(\mathcal{Q}_{g_{23+}} + \mathcal{Q}_{g_{23-}}\right) + \left(\mathcal{Q}_{g_{34+}} + \mathcal{Q}_{g_{34-}}\right)$$

O Gerador Infinitesimal deste exemplo corresponde a uma matriz quadrada de dimensão 36 e está fora do escopo desta seção mostrá-lo por completo. No entanto, observa-se que o descritor Markoviano apresenta características marcantes tais como matrizes esparsas e diversas matrizes do tipo identidade, as quais marcam que um determinado evento *não* ocorre em um autômato.

A seguir, na próxima seção, descrevem-se os mecanismos de cálculo computacional do vetor de probabilidade estacionário ou transiente utilizando-se maneiras sofisticadas para multiplicação entre vetores e descritores Markovianos.

4.4 Multiplicação Vetor-Descritor

As Cadeias de Markov são resolvidas, em última análise, através de multiplicações entre um vetor e uma matriz, conforme explicado na Seção 2.2. Entretanto, formalismos estruturados com descritores Markovianos nunca geram a matriz de transição correspondente à MC subjacente e sim sua representação tensorial, ou seja, um descritor Markoviano. Para resolver um modelo descrito desta forma não é mais possível utilizar a mesma abordagem de MC, ou seja, uma multiplicação vetor-matriz, mas sim uma Multiplicação Vetor-Descritor (MVD). Graficamente, usando-se o descritor do exemplo anterior (Seção 4.3.2), a MVD pode ser explicada da seguinte forma (seja n o número total de estados dado por $|\mathcal{X}|$):

$$\overbrace{\left(\pi_0\ \pi_1\ \pi_2\ \ldots\ \pi_{n-2}\ \pi_{n-1}\right)}^{\text{vetor de probabilidades } \pi} \times \overbrace{\left[\begin{array}{c} \mathcal{Q}_l^{\mathcal{MN}^{(1)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(2)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(3)}} \oplus \mathcal{Q}_l^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3+}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{t_{3-}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12+}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{12-}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23+}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{23-}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34+}}^{\mathcal{MN}^{(4)}} \\ + \\ \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(1)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(2)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(3)}} \otimes \mathcal{Q}_{g_{34-}}^{\mathcal{MN}^{(4)}} \end{array}\right]}^{\text{Descritor Markoviano}} = \Big(0 \ldots 0\Big)$$

Esta seção tratará das formas de multiplicar um vetor de probabilidade por um descritor. Um dos ramos de pesquisa em SAN foca-se em formas de melhorar o desempenho desta forma não trivial de multiplicação. Existem atualmente três algoritmos distintos para operar com o produto de um vetor por um descritor: Algoritmo *Esparso* [STE94], Algoritmo *Shuffle* [FER98b] e uma combinação destas duas abordagens denominado Algoritmo *Split* [CZE07]. Estas três maneiras serão mais detalhadas a seguir. Definindo-se mais formalmente, a MVD pode ser vista como a multiplicação de um vetor de probabilidade π por $|\mathcal{L}| = N + 2E$ termos tensoriais compostos por N matrizes, onde $\mathcal{L}$ é um conjunto de termos tensoriais do tipo $\{l, e^+, e^-\}$, e dada pela expressão[3]:

$$\sum_{j=1}^{|\mathcal{L}|} \left(\pi \times \left[\bigotimes_{i=1}^{N} \mathcal{Q}_j^{(i)} \right] \right) \tag{4.29}$$

[3]Conforme Seção 4.1.2, uma soma tensorial é um produto tensorial com matrizes identidade.

Antes de explicar detalhadamente os algoritmos de MVD, são apresentadas as notações utilizadas.

Sejam:

n_k dimensão da matriz $\mathcal{Q}^{(k)}$;

nz_k total de elementos não-nulos da matriz $\mathcal{Q}^{(k)}$;

$\theta_{(1\ldots N)}$ o conjunto dos escalares gerados a partir das combinações de elementos não-nulos das matrizes de $\mathcal{Q}^{(1)}$ até $\mathcal{Q}^{(N)}$ de um termo tensorial;

$\prod_{i=1}^{N} nz_i$ a cardinalidade de $\theta_{(1\ldots N)}$, e consequentemente o número de elementos não-nulos em $\mathcal{Q}$;

$nright_k$ tamanho do espaço de estados correspondente a todas as matrizes *depois* da matriz k-ésima do termo tensorial numericamente definida como $\prod_{i=k+1}^{N} n_i$, com o caso especial $nright_N = 1$;

$nleft_k$ tamanho do espaço de estados correspondente a todas as matrizes *antes* da matriz k-ésima do termo tensorial numericamente definida como $\prod_{i=1}^{k-1} n_i$, com o caso especial $nleft_1 = 1$;

$njump_k$ produto do espaço de estados depois da k-ésima matriz do termo tensorial pela dimensão n_k desta mesma matriz;

Υ, υ vetores de probabilidades;

4.4.1 Algoritmo *Esparso*

O Algoritmo *Esparso* é o mais intuitivo dos métodos de MVD [STE94, STE09]. A ideia principal do algoritmo é considerar o termo tensorial como uma única e potencialmente grande matriz esparsa a ser multiplicada pelo vetor de probabilidade. Este algoritmo assemelha-se ao método utilizado para a multiplicação de um vetor pelo Gerador Infinitesimal, a única diferença é que se considera apenas cada termo tensorial como uma grande matriz.

Para um termo tensorial composto por N matrizes $\mathcal{Q}^{(i)}$, cada uma de dimensão n_i com nz_i elementos não-nulos, o Algoritmo *Esparso* gerará todos os elementos em uma matriz $\mathcal{Q}$ resultando em $\mathcal{Q} = \bigotimes_{i=1}^{N} \mathcal{Q}^{(i)}$, com dimensão $\prod_{i=1}^{N} n_i$.

Uma possível implementação do Algoritmo *Esparso* [STE91] é apresentada no Algoritmo 4.1. Cabe ressaltar que a linha 6 mostra que antes de realizar as multiplicações, dependendo do evento e das funções associadas no modelo, serão necessárias avaliações de função através da chamada **evaluate**. O número de multiplicações em ponto flutuante para essa implementação é dada pela Equação 4.30:

$$C = N \times \prod_{i=1}^{N} nz_i \tag{4.30}$$

Algoritmo 4.1: *Esparso* – $\Upsilon = \upsilon \times \otimes_{g\,i=1}^{N} \mathcal{Q}^{(i)}$.

1: $\Upsilon = 0$
2: **for all** $i_1, \ldots, i_N, j_1, \ldots, j_N \in \theta(1 \ldots N)$ **do**
3: $\quad s = 1$
4: $\quad base_{in} = base_{out} = 0$
5: $\quad$ **for all** $k = 1, 2, \ldots, N$ **do**
6: $\quad\quad$ **evaluate** $Q^{(k)}$
7: $\quad\quad s = s \times q^{(k)}_{(i_k, j_k)}$ {cálculo de um escalar}
8: $\quad\quad base_{in} = base_{in} + ((i_k - 1) \times nright_k)$
9: $\quad\quad base_{out} = base_{out} + ((j_k - 1) \times nright_k)$
10: $\quad$ **end for**
11: $\quad \Upsilon[base_{out}] = \Upsilon[base_{out}] + \upsilon[base_{in}] \times s$
12: **end for**

Entretanto, nesta versão, todos os elementos não-nulos de $\mathcal{Q}$ são, descrevendo-se o problema computacionalmente, gerados em tempo de execução do algoritmo. Tal geração representa $(N - 1) \times \prod_{i=1}^{N} nz_i$ multiplicações que poderiam ser evitadas se uma matriz esparsa (usualmente grande) fosse criada para guardar estes $\prod_{i=1}^{N} nz_i$ elementos não-nulos. Esse procedimento eliminaria as linhas 3 e 7 do Algoritmo 4.1 (o escalar s seria buscado de uma estrutura de dados auxiliar em memória) e reduziria o número de multiplicações em ponto flutuante como definido pela Equação 4.31:

$$C = \prod_{i=1}^{N} nz_i \tag{4.31}$$

Esta opção, ao utilizar mais memória, reduz o número de cálculo de índices (que são normalmente efetuadas em outros algoritmos, por exemplo, no *Shuffle*, visto a seguir) e permite que o Algoritmo *Esparso* seja eficiente em termos de tempo gasto de execução. Entretanto, a memória gasta para salvar essa estrutura de dados (além dos vetores de probabilidade necessários) torna alguns modelos intratáveis, fazendo com que essa opção seja descartada para modelos que contém um espaço produto de estados alto. O total de chamadas à diretiva de avaliação de funções varia de modelo para modelo e não faz parte da equação de complexidade apresentada.

4.4.2 Algoritmo *Shuffle*

A seguir, explica-se o Algoritmo *Shuffle* [DAV81], que é eficiente em termos de memória gasta para armazenar o descritor, pois nunca gera explicitamente a matriz $\tilde{\mathcal{Q}}$ ou mesmo uma parte dela, apenas as informações contidas nas matrizes de cada termo tensorial. A seguir, aplica eficazmente operações de Álgebra Tensorial para o cálculo das informações necessárias. O princípio básico deste algoritmo é a aplicação da decomposição de um termo tensorial na propriedade do produto ordinário de fatores normais [FER98b]:

$$
\begin{aligned}
Q^{(1)} \underset{g}{\otimes} Q^{(2)} \underset{g}{\otimes} \ldots \underset{g}{\otimes} Q^{(N-1)} \underset{g}{\otimes} Q^{(N)} = & (Q^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N}) \times \\
& (I_{n_1} \underset{g}{\otimes} Q^{(2)} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N}) \times \\
& \ldots \\
& (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} Q^{(N-1)} \underset{g}{\otimes} I_{n_N}) \times \\
& (I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \ldots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} Q^{(N)})
\end{aligned}
$$

O Algoritmo *Shuffle* consiste em multiplicar sucessivamente o vetor de probabilidade por cada fator normal. Mais precisamente, o vetor v é multiplicado pelo primeiro fator normal e o vetor resultante é multiplicado pelo próximo e assim por diante, até o último fator normal.

Internamente, para cada fator normal, as multiplicações são feitas usando-se pequenos vetores chamados z_{in} e z_{out} como descrito no Algoritmo 4.2. Estes vetores pequenos em termos de dimensão igual à ordem da matriz sendo multiplicada guardam os valores de v para multiplicação pela $i^{\text{ésima}}$ matriz do fator normal z_{in} e guardar o resultado em z_{out}.

Algoritmo 4.2: *Shuffle* – $\Upsilon = v \times \otimes_{g\, i=1}^{N} Q^{(i)}$.

```
1:  for all i = 1,2,...,N do
2:     base = 0
3:     for all m = 0,1,2,...,nleft_i - 1 do
4:        for all j = 0,1,2,...,nright_i - 1 do
5:           evaluate   Q^(i)
6:           index = base + j
7:           for all l = 0,1,2,...,n_i - 1 do
8:              z_in[l] = v[index]
9:              index = index + nright_i
10:          end for
11:          multiply  z_out = z_in × Q^(i)
12:          index = base + j
13:          for all l = 0,1,2,...,n_i - 1 do
14:             v[index] = z_out[l]
15:             index = index + nright_i
16:          end for
17:       end for
18:       base = base + (nright_i × n_i)
19:    end for
20: end for
21: Υ = v
```

A linha 5 do algoritmo corresponde a uma avaliação de função (caso seja necessária para o modelo e de acordo com a definição da função) através da diretiva **evaluate**. A multiplicação pelo

vetor v pelo $i^{\text{ésimo}}$ fator normal consiste, generalizadamente, em *embaralhar*[4] os elementos de v para montar $nleft_i \times nright_i$ vetores de tamanho n_i e multiplicá-los pela matriz $\mathcal{Q}^{(i)}$.

Assumindo-se que a matriz $\mathcal{Q}^{(i)}$ é guardada de forma esparsa, o número de operações necessárias para multiplicar um vetor pelo $i^{\text{ésimo}}$ fator normal é: $nleft_i \times nright_i \times nz_i$, onde nz_i corresponde ao número de elementos não-nulos da $i^{\text{ésima}}$ matriz do termo tensorial $\mathcal{Q}^{(i)}$.

Considerando-se o número de multiplicações por todos os fatores normais de um termo tensorial, o custo computacional do Algoritmo *Shuffle* para efetuar a operação de multiplicação de um vetor por um descritor é dada pela Equação 4.32 [FER98b]:

$$C = \sum_{i=1}^{N} nleft_i \times nright_i \times nz_i = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i} \tag{4.32}$$

Uma outra vantagem de uso do Algoritmo *Shuffle* consiste no fato de possuir otimizações para modelos com taxas funcionais, ou seja, modelos que utilizam propriedades da Álgebra Tensorial Generalizada e também no que diz respeito ao uso de reordenamentos da estrutura tensorial (também chamadas de permutações). Como no Algoritmo *Esparso* as chamadas para avaliações de funções variam em cada modelo e não estão sendo contabilizadas no cálculo da complexidade final.

4.4.3 Algoritmo *Split*

Eventos locais em SAN são normalmente esparsos, indicando um comportamento que independe de outros autômatos de um sistema. O mesmo não ocorre com eventos sincronizantes, pois podem potencialmente afetar todos os autômatos de um modelo. Entretanto, dependendo do modelo que está sendo analisado, essa condição é inexistente, *i.e.*, um número substancial de ocorrências dizem respeito à entre pelo menos dois autômatos e um número que raramente compreende o total de autômatos. Esse é um dos motivos pelos quais iniciou-se a pesquisa de algoritmos para execução do processo de MVD de forma híbrida tirando vantagem da propriedade da *Decomposição Aditiva* combinada à decomposição de produtos tensoriais clássicos em fatores normais.

A propriedade da Decomposição Aditiva dita que qualquer produto tensorial pode ser decomposto em uma soma ordinária de matrizes compostas por um único elemento não-nulo. Seja $\hat{q}_{(i_1,\ldots,i_{N-1},j_1,\ldots,j_N)}$ a matriz de dimensão $\prod_{i=1}^{N} n_i$ composta por apenas um elemento não-nulo o qual encontra-se na posição $i_1,\ldots,i_N,j_1,\ldots,j_N$ e igual a $\prod_{k=1}^{N} q^{(k)}_{i_k,j_k}$. A propriedade da Decomposição Aditiva pode ser definida como:

$$\begin{array}{c} Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-1)} \otimes Q^{(N)} = \sum_{i_1=1}^{n_1} \cdots \sum_{i_N=1}^{n_N} \sum_{j_1=1}^{n_1} \cdots \sum_{j_N=1}^{n_N} \\ \left(\hat{q}^{(1)}_{(i_1,j_1)} \otimes \ldots \otimes \hat{q}^{(N)}_{(i_N,j_N)} \right) \end{array} \tag{4.33}$$

onde $\hat{q}^{(k)}_{(i,j)}$ é uma matriz de dimensão n_k a qual o elemento na linha i e coluna j é $q^{(k)}_{i,j}$.

[4] Do inglês, *shuffling*.

Tabela 4.1: Possibilidades de divisão de termos tensoriais para o Algoritmo *Split*.

σ	Termo Tensorial
0	$\overset{\sigma}{\downarrow}$ $\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{\textit{shuffle}}$
1	$\underbrace{Q^{(1)}}_{\textit{esparso}} \overset{\sigma}{\downarrow} \otimes \underbrace{Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{\textit{shuffle}}$
2	$\underbrace{Q^{(1)} \otimes Q^{(2)}}_{\textit{esparso}} \overset{\sigma}{\downarrow} \otimes \underbrace{\ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{\textit{shuffle}}$
$\vdots$	$\vdots$
N-2	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)}}_{\textit{esparso}} \overset{\sigma}{\downarrow} \otimes \underbrace{Q^{(N-1)} \otimes Q^{(N)}}_{\textit{shuffle}}$
N-1	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)}}_{\textit{esparso}} \overset{\sigma}{\downarrow} \otimes \underbrace{Q^{(N)}}_{\textit{shuffle}}$
N	$\underbrace{Q^{(1)} \otimes Q^{(2)} \otimes \ldots \otimes Q^{(N-3)} \otimes Q^{(N-2)} \otimes Q^{(N-1)} \otimes Q^{(N)}}_{\textit{esparso}} \overset{\sigma}{\downarrow}$

O Algoritmo *Split* [CZE07] propõe uma solução combinada usando fundamentalmente a propriedade da Decomposição Aditiva para um dado conjunto de matrizes em um produto tensorial seguido da aplicação da operação de *embaralhamento* para as matrizes restantes do termo. Cada fator é chamado de Fator Normal Aditivo Unitário (*Additive Unitary Normal Factor* – AUNF). Esse conceito é central para o Algoritmo *Split* e internamente trata-se de uma estrutura de dados que armazena os índices dos vetores de entrada ($base_{in}$) e saída ($base_{out}$), um escalar s e um vetor de tamanho σ contendo os índices de linha e coluna de cada matriz que foram usados para calcular s (esse vetor será usado para avaliação de funções com parâmetros que estejam na parte esparsa posteriormente).

Agora tem-se então uma divisão do termo tensorial em dois grupos distintos: o primeiro contendo matrizes que serão unidas sob a forma de uma lista de fatores normais aditivos unitários e o segundo grupo, contendo o restante das matrizes. Em termos de solução, o primeiro grupo será manipulado com uma solução esparsa, multiplicando tensorialmente cada AUNF pelo segundo grupo usando a solução *Shuffle*. A ideia principal é dividir o termo tensorial em dois conjuntos e tratá-los de forma distinta em termos de multiplicação vetor-descritor.

A Tabela 4.1 apresenta estes conceitos do Algoritmo *Split* de forma gráfica. O índice da matriz escolhida para delimitar o fim do tratamento com a parte esparsa (e subsequentemente, o início da parte *Shuffle*) é denominado *parâmetro de corte* σ, ou simplesmente *corte* σ. Casos especiais do Algoritmo *Split* são os casos onde o corte possui valores extremos, *i.e.*, quando $\sigma = N$ significa que a abordagem *Esparsa* pura está sendo utilizada e quando $\sigma = 0$ não existem matrizes no lado esparso do termo tensorial e apenas o Algoritmo *Shuffle* será executado. Estes são casos particulares do Algoritmo *Split*.

Parte-se a seguir para a definição formal da operação do método em si, de acordo com o Algoritmo 4.3. A ideia principal do método consiste em computar o elemento escalar s de cada AUNF em $\theta(1 \ldots \sigma)$ multiplicando cada elemento não-nulo s de cada matriz do primeiro conjunto de matrizes (parte esparsa) de $\mathcal{Q}^{(1)}$ até $\mathcal{Q}^{(\sigma)}$ (linhas 5 a 9). De acordo com os elementos de linha que foram utilizados para gerar s, uma fatia contígua do vetor de entrada v, chamado v_{in}, e será utilizado em uma estrutura de dados. O vetor v_{in} de tamanho $nright_{\sigma}$ (correspondente ao produto das dimensões das matrizes após o *parâmetro de corte* σ do termo tensorial) é multiplicado pelo escalar s. Nas linhas 10 e 11 são realizadas as multiplicações de s para cada posição em v, finalizando a parte esparsa do algoritmo. O vetor resultante v_{in} também é utilizado como vetor de entrada para a parte Shuffle (linhas 13 a 32) pelo produto tensorial das matrizes do segundo conjunto (de $\mathcal{Q}^{(\sigma+1)}$ até $\mathcal{Q}^{(N)}$). Ao término da parte *Shuffle* o vetor v obtido é acumulado no vetor final Υ (linhas 33 a 35).

O algoritmo possui três blocos distintos, um bloco onde para cada AUNF calculado (de zero a σ), um vetor auxiliar de tamanho $nright_{\sigma}$ (*i.e.*, $nright$ de zero até o valor do corte) é multiplicado em posições chave originadas durante o cálculo do AUNF em questão. Um segundo bloco, onde o método *Shuffle* é chamado para um sub-espaço à esquerda que desconta as matrizes da parte esparsa (linha 15) e finalmente, uma seção onde este vetor auxiliar que foi modificado no bloco anterior é acumulado no vetor final Υ.

A linha 17 do algoritmo mostra uma diretiva chamada **evaluate**, utilizada para a avaliação de elementos funcionais. Cabe ressaltar que o Algoritmo *Split* original [CZE07] foi estendido para tratar de descritores generalizados, ou seja, modelos com taxas funcionais. Essa é basicamente a diferença fundamental entre as alternativas existentes até o momento de MVD que utilizem soluções híbridas de armazenamento. Os impactos desta diretiva serão melhor analisados nos Capítulos 5 e 7 referentes às estratégias de modificação dos descritores e aos resultados obtidos, respectivamente.

O custo computacional do Algoritmo *Split* em termos de multiplicações é descrito pela Equação 4.34. Em (a), encontra-se o total de AUNFs, (b) o custo para criar *um* AUNF, em (c) o subespaço à direita de σ a ser multiplicado por cada AUNF e, por fim, (d) a parte que adiciona a complexidade do Algoritmo *Shuffle*, o qual é operado de $(\sigma+1)$ a N.

$$C = \overbrace{\left(\prod_{i=1}^{\sigma} nz_i\right)}^{\text{(a)}} \left[\overbrace{(\sigma-1)}^{\text{(b)}} + \overbrace{\left(\prod_{i=\sigma+1}^{N} n_i\right)}^{\text{(c)}} + \overbrace{\left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{i=\sigma+1}^{N} \frac{nz_i}{n_i}\right)}^{\text{(d)}} \right] \tag{4.34}$$

Existem algumas otimizações que podem ser implementadas em algoritmos para MVD. Estas modificações em termos de implementação alteram significativamente o custo computacional teórico apresentado na Equação 4.34. Para o caso do Algoritmo *Shuffle*, pode-se otimizar a maneira pela qual o método lida com matrizes do tipo identidade.

Estas matrizes fazem com que seja desnecessário gerar fatores normais para o cálculo do vetor solução, pois, sendo identidades, as matrizes restantes também o são, logo todo o termo tensorial

Algoritmo 4.3: *Split* – $\Upsilon = v \times \otimes_{g\,i=1}^{N} \mathcal{Q}^{(i)}$ considerando ponto de corte σ.

1: $\Upsilon = 0$
2: **for all** $i_1, \ldots, i_\sigma, j_1, \ldots, j_\sigma \in \theta(1 \ldots \sigma)$ **do**
3: $s = 1$
4: $base_{in} = base_{out} = 0$
5: **for all** $k = 1, 2, \ldots, \sigma$ **do**
6: $s = s \times q^{(k)}_{(i_k, j_k)}$ {cálculo do escalar - **evaluate** $Q^{(i)}$ se necessário}
7: $base_{in} = base_{in} + ((i_k - 1) \times nright_k)$
8: $base_{out} = base_{out} + ((j_k - 1) \times nright_k)$
9: **end for**
10: **for all** $l = 0, 1, 2, \ldots, nright_\sigma - 1$ **do**
11: $v_{in}[l] = v[base_{in} + l] \times s$
12: **end for**
13: **for all** $i = \sigma + 1, \ldots, N$ **do**
14: $base = 0$
15: **for all** $m = 0, 1, 2, \ldots, \frac{nleft_i}{nleft_\sigma} - 1$ **do**
16: **for all** $j = 0, 1, 2, \ldots, nright_i$ **do**
17: **evaluate** $Q^{(i)}$ {se necessário}
18: $index = base + j$
19: **for all** $l = 0, 1, 2, \ldots, n_i - 1$ **do**
20: $z_{in}[l] = v_{in}[index]$
21: $index = index + nright_i$
22: **end for**
23: $multiply\ z_{out} = z_{in} \times \mathcal{Q}^{(i)}$
24: $index = base + j$
25: **for all** $l = 0, 1, 2, \ldots, n_i - 1$ **do**
26: $v_{in}[index] = z_{out}[l]$
27: $index = index + nright_i$
28: **end for**
29: **end for**
30: $base = base + (nright_i \times n_i)$
31: **end for**
32: **end for**
33: **for all** $l = 0, 1, 2, \ldots, nright_\sigma - 1$ **do**
34: $\Upsilon[base_{out} + l] = \Upsilon[base_{out} + l] + v_{in}[l]$
35: **end for**
36: **end for**

é uma identidade. É natural que, em se tratando deste caso particular, nenhum fator normal seja criado, reduzindo o custo computacional de execução. Este custo corresponde a transformar a Equação 4.32 na Equação 4.35:

$$C = \prod_{i=1}^{N} n_i \times \sum_{\text{iff} Q^{(i)} \neq Id}^{N} \frac{nz_i}{n_i} \tag{4.35}$$

Entretanto, dada a flexilidade do Algoritmo *Split* e dependendo do evento que esteja sendo considerado, o número de AUNFs gerados pode onerar a memória a ser gasta para armazenamento dos escalares necessários. Nesse caso, uma análise prévia minuciosa de cada termo tensorial deve ser realizada, detectando casos onde um grande bloco composto unicamente por identidades está definido (após o termo ser permutado), detectando onde o método esparso obterá um ganho significativo de desempenho frente as demais modalidades de MVD existentes.

Esta melhoria sugere que o mesmo pode ser aplicado na parte do método *Shuffle* no Algorimo 4.3 referente ao Algoritmo *Split* nas matrizes $\mathcal{Q}^{(\sigma+1)}$ a $\mathcal{Q}^{(N)}$. Caso a matriz indexada por i no algoritmo ($\mathcal{Q}^{(i)}$) não seja uma matriz identidade, o custo de $\frac{nz_i}{n_i}$ multiplicações é adicionado.

Analogamente ao Algoritmo *Shuffle*, a Equação 4.34 pode ser reescrita alterando-se o custo para o cálculo a partir de σ. O número de multiplicações resultantes para o Algoritmo *Split* seguirá a Equação 4.36:

$$C = \left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1) + \left(\prod_{i=\sigma+1}^{N} n_i\right) + \left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{\substack{i=\sigma+1 \\ \text{iff } Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}\right)\right] \tag{4.36}$$

Usualmente, em SAN, os termos tensoriais são esparsos, uma vez que indicam a ocorrência dos eventos em cada autômato (exceto em casos onde existam diversos eventos ocorrendo em diversos autômatos, indicando matrizes quase plenas no termo).

Uma outra otimização que pode ser realizada trata dos pré-cálculos dos elementos não-nulos, salvando-os em estruturas de dados que serão acessadas em todas as iterações, mas computados apenas na primeira vez que o método iterativo é chamado. Estas otimizações foram melhor estudadas por [MIN00] e causam uma redução significativa no custo computacional do Algoritmo *Split*, similar ao apresentado na Seção 4.4.1 (Equação 4.30) no que diz respeito ao cálculo dos elementos não-nulos. Logo, o valor final da definição do número de multiplicações em ponto flutuante necessárias para a execução do Algoritmo *Split* não é mais o valor definido pela Equação 4.36, e sim dado pela Equação 4.37:

$$C = \left(\prod_{i=1}^{\sigma} nz_i\right)\left[\left(\prod_{i=\sigma+1}^{N} n_i\right) + \left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{\substack{i=\sigma+1 \\ \text{iff } Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}\right)\right] \tag{4.37}$$

Mas, se, por exemplo, só existirem matrizes do tipo identidade na parte estruturada, toda a complexidade envolvida pelo Algoritmo Shuffle não é mais necessária, fazendo com que a Equação 4.37 seja convertida na Equação 4.38:

$$C = \prod_{i=1}^{\sigma} nz_i \prod_{i=\sigma+1}^{N} n_i \tag{4.38}$$

4.5 Permutações dos termos tensoriais

Uma das operações mais custosas efetuadas na MVD é a avaliação de funções. No contexto deste trabalho, estas avaliações estão presentes nos algoritmos que foram explicados nas Seções 4.4.1, 4.4.2 e 4.4.3 referentes respectivamente aos Algoritmos *Esparso*, *Shuffle* e *Split*. Estas chamadas de avaliações estão presentes nos subespaços à esquerda e à direita da matriz sendo multiplicada pelo vetor, ou seja, são executadas diversas vezes ao longo do método, dependendo do tamanho.

Para diminuir os efeitos destas execuções, são apresentadas técnicas de reordenamento ou permutação dos termos tensoriais. Estas técnicas servem para reorganizar as matrizes dos termos tensoriais detectando uma forma para avaliar menos funções, colocando a diretiva de avaliação dentro de apenas um laço e não em dois laços como é feito originalmente no Algoritmo *Shuffle* [BEN03b].

A definição inicial do Algoritmo *Split* não previa a realização de permutações. A inclusão de solução para tensores em ATG é uma contribuição que foi introduzida neste trabalho bem como o estudo e implementação adicional para contemplar tal necessidade. Como será melhor explicado nos próximos capítulos, o uso de permutações no Algoritmo *Split* será necessário para uma solução adequada. Ao observar os casos particulares de produtos tensoriais generalizados, pode-se compreender melhor o interesse em realizar estas otimizações. Por exemplo, no produto tensorial generalizado dado por

$$\bigotimes_{i=1}^{N}{}_{g} \mathcal{Q}^{(i)}(\mathcal{A}^{(i+1)},\ldots,\mathcal{A}^{(N)})$$

os fatores normais da decomposição devem ser tratados de acordo com a seguinte ordem [BEN03b]:

$$\begin{array}{lccc} & I_{nleft_1} \underset{g}{\otimes} \mathcal{Q}^{(1)}(\mathcal{A}^{(2)},\ldots,\mathcal{A}^{(N)}) \underset{g}{\otimes} I_{nright_1} \\ \times & I_{nleft_2} \underset{g}{\otimes} \mathcal{Q}^{(2)}(\mathcal{A}^{(3)},\ldots,\mathcal{A}^{(N)}) \underset{g}{\otimes} I_{nright_2} \\ \times & \ldots & \ldots & \ldots \\ \times & I_{nleft_N} \underset{g}{\otimes} & \mathcal{Q}^{(N)} & \underset{g}{\otimes} I_{nright_N} \end{array}$$

Observa-se que cada matriz $\mathcal{Q}^{(i)}(\ldots)$ só depende das matrizes à sua direita. De maneira análoga, para o seguinte produto tensorial generalizado:

$$\bigotimes_{i=1}^{N}{}_{g} \mathcal{Q}^{(i)}(\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(i-1)})$$

Os fatores normais da decomposição devem ser tratados de acordo com a seguinte ordem:

$$\begin{array}{lccc} & I_{nleft_N} \underset{g}{\otimes} & \mathcal{Q}^{(N)}(\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(N-1)}) & \underset{g}{\otimes} I_{nright_N} \\ \times & I_{nleft_{N-1}} \underset{g}{\otimes} & \mathcal{Q}^{(N-1)}(\mathcal{A}^{(1)},\ldots,\mathcal{A}^{(N-2)}) & \underset{g}{\otimes} I_{nright_{N-1}} \\ \times & \ldots & \ldots & \ldots \\ \times & I_{nleft_1} \underset{g}{\otimes} & \mathcal{Q}^{(1)} & \underset{g}{\otimes} I_{nright_1} \end{array}$$

Para este caso, cada matriz $\mathcal{Q}^{(i)}(\ldots)$ só depende das matrizes à sua esquerda. Devido a esse fato, como o posicionamento dos autômatos é conhecido, é possível modificar os algoritmos de MVD para realizarem menos chamadas à diretivas de avaliação de funções. No caso do Algoritmo 4.2, a linha 5 seria movida para dentro do laço de $nright$ (ou seja, logo abaixo da linha 3) e no caso do Algoritmo 4.3, a linha 17 seria colocada logo abaixo da linha 15, pelo mesmo motivo.

Ainda encontram-se em aberto estudos de como realizar esse reordenamento de autômatos para se encontrar os casos onde esse procedimento é passível de ser realizado visando a redução do número de avaliações de elementos funcionais. Para isso, informalmente, são poucos os casos onde as funções definidas dependem do estado de *todos* os outros autômatos. Normalmente, os parâmetros das funções (*i.e.*, estados de autômatos) possuem um tamanho restrito. Caso uma função dependesse do estado de todos os autômatos, este seria o pior caso para o processo de permutação, inclusive tornaria desnecessária a realização de quaisquer reordenamentos, pois não eliminariam o número de avaliações necessárias.

Não faz parte do escopo deste trabalho explicar detalhadamente as variadas técnicas de reordenamento de termos tensoriais, pois estas já foram exaustivamente feitas em [FER98a, FER98b, BEN03b]. Este trabalho parte do princípio que a realização de reordenamentos pode ser extremamente eficaz tanto para reduzir-se o número de avaliações como utilizar essas reordens para reduzir o tempo necessário para executar métodos híbridos de solução, neste caso, especificamente para o caso do Algoritmo *Split*.

Tabela 4.2: Reordenamentos da estrutura original dos índices dos autômatos e seus *ranks*.

$\mathcal{A}^{(1)},\mathcal{A}^{(2)},\mathcal{A}^{(3)}$				$\mathcal{A}^{(3)},\mathcal{A}^{(1)},\mathcal{A}^{(2)}$			
rank	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	*rank*	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0
2	0	1	0	2	0	2	0
3	0	1	1	3	1	0	0
4	0	2	0	4	1	1	0
5	0	2	1	5	1	2	0
6	1	0	0	6	0	0	1
7	1	0	1	7	0	1	1
8	1	1	0	8	0	2	1
9	1	1	1	9	1	0	1
10	1	2	0	10	1	1	1
11	1	2	1	11	1	2	1

A seguir, serão apresentados os princípios da permutação em termos tensoriais, seus objetivos e algumas definições de base. Dado que os autômatos podem trocar de posição no termo tensorial, os elementos do vetor de probabilidade final devem trocar de lugar pois a ordem lexicográfica original foi alterada. A Tabela 4.2 mostra os estados de um vetor pelos autômatos ordenados de duas maneiras distintas. Os elementos do vetor são organizados de acordo com uma ordem lexicográfica expressa pela lista de autômatos, igualmente respeitando uma ordem.

À esquerda da tabela é utilizada a ordem $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}$. Os autômatos deste exemplo possuem tamanho $n_1 = 2$, $n_2 = 3$ e $n_3 = 2$. À direita da tabela é realizada a seguinte reordem dos autômatos: $\mathcal{A}^{(3)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)}$. A coluna *rank* equivale ao índice no vetor de probabilidade. Como para esse exemplo o $|\mathcal{X}| = 2 \times 3 \times 2 = 12$, esta coluna varia de 0 a 11.

A ideia aqui é descobrir o próximo estado dado um estado corrente. Para simplificar as operações envolvidas, a melhor forma de se fazer esta descoberta é pensar em um algoritmo que retorne o próximo estado, dada uma ordem. Logo, observa-se os autômatos do último até o primeiro e tenta-se incrementar seu estado local. Um incremento é inválido quando o estado local do autômato é o último estado (não existem mais estados). Quando o incremento é inválido , o estado local do autômato é zerado (*reset* de estado) e se considera o próximo estado (de acordo com uma determinada ordem, podendo não ser a original e sim qualquer ordem). Este procedimento acaba quando um incremento válido foi realizado.

Para exemplificar esse processo, considere-se a ordem $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \mathcal{A}^{(3)}$ onde se deseja saber o próximo estado (um incremento de estado) dado que o estado atual é o $(0, 1, 1)$. O autômato $\mathcal{A}^{(3)}$ é o primeiro a ser considerado. Tenta-se incrementá-lo, mas, por estar já no último estado possível (1), coloca-se nessa posição, o valor 0. A seguir, tenta-se incrementar $\mathcal{A}^{(2)}$, e o incremento é válido, para 2. Logo, o próximo estado global do sistema seguido do estado $(0, 1, 1)$ é o $(0, 2, 0)$ (observe essa ocorrência na Tabela 4.2, entre os *ranks* 3 e 4 do lado esquerdo).

Entretanto, para o caso de uma permutação das posições no termo tensorial, os incrementos não são mais tão simples, e devem seguir um cálculo mais elaborado do *rank* final para descoberta do próximo estado. É necessário se calcular o novo *rank* a partir do *rank* corrente do vetor permutado. Seguindo-se o mesmo exemplo, o *rank* do estado $(0, 1, 1)$ pela nova ordem lexicográfica dada por $\mathcal{A}^{(3)}, \mathcal{A}^{(1)}, \mathcal{A}^{(2)}$ é 7. O *rank* após o incremento que corresponde ao estado $(0, 2, 0)$ deve ser 2.

Para calcular esse novo índice, realizam-se incrementos e decrementos, zerando alguns estados locais. Os valores destes incrementos e decrementos variam de acordo com a nova ordem lexicográfica. Um incremento válido para o próximo estado de um estado local do autômato $\mathcal{A}^{(i)}$ corresponde à adição de $nright_i$ no *rank*. No exemplo em questão foi feito um *reset* no estado do autômato $\mathcal{A}^{(3)}$, e sob a nova ordem lexicográfica, $nright_3 = 2 \times 3 = 6$. Com isso é obtido o estado de *rank* $7 - 6 = 1$. A seguir, como ocorrido no exemplo anterior, incrementa-se o estado do autômato $\mathcal{A}^{(2)}$, e este incremento é válido, ou seja, obtem-se o valor 2 para o *rank* final, como seria normalmente retornado mas, neste caso, com o uso de permutações.

Com isso, observa-se que são feitos os mesmos procedimentos para ambos os exemplos, apenas no caso onde houve a permutação das ordens originais é que, ao invés de se incrementar, foi descontado um valor que correspondia ao $nright_i$, equivalendo ao salto que deve ser feito na estrutura permutada. Trata-se de uma maneira transparente de se efetuar as permutações, incorrendo-se apenas no recálculo de índices para saltar na estrutura tensorial.

As permutações foram inicialmente utilizadas no Algoritmo *Shuffle* com o intuito principal de eliminar chamadas desnecessárias de avaliações de elementos funcionais. Com isso, para alguns modelos, observou-se um ganho de desempenho, pois menos operações são efetuadas se uma per-

mutação satisfatória for encontrada. Naturalmente, não é fácil descobrir como melhor permutar cada termo tensorial para obter o máximo de desempenho, sendo esta uma questão aberta de pesquisa.

No caso do Algoritmo *Split*, as permutações desempenharão um papel ainda mais importante que a verificada para o Algoritmo *Shuffle* e usada com êxito para alguns casos. No caso do *Split*, os reordenamentos serão cruciais para entender qual a melhor forma de dividir os termos tensoriais enviando matrizes ou para o lado estruturado (direito do parâmetro de corte σ) ou enviando as avaliações para a parte esparsa e vice-versa. Somente com as permutações é possível verificar onde o método é melhor executado, pois separa cada característica dos termos tensoriais.

4.6 Algoritmo *Split* com permutações e identidades à direita de σ

A Figura 4.6 mostra a operação do Algoritmo *Split* quando existem apenas matrizes do tipo identidade na parte à direita do parâmetro de corte σ (utilizada pelo Algoritmo *Shuffle*). Observa-se que os AUNFs (não-nulos de 0 a σ, *i.e.*, $\prod_{i=1}^{\sigma} nz_i$) possuem um papel central nessa representação, pois guardam, para cada escalar s do vetor de entrada apontado por Υ os índices de linha e coluna que foram utilizados em cada matriz ($n = |\mathcal{X}|$).

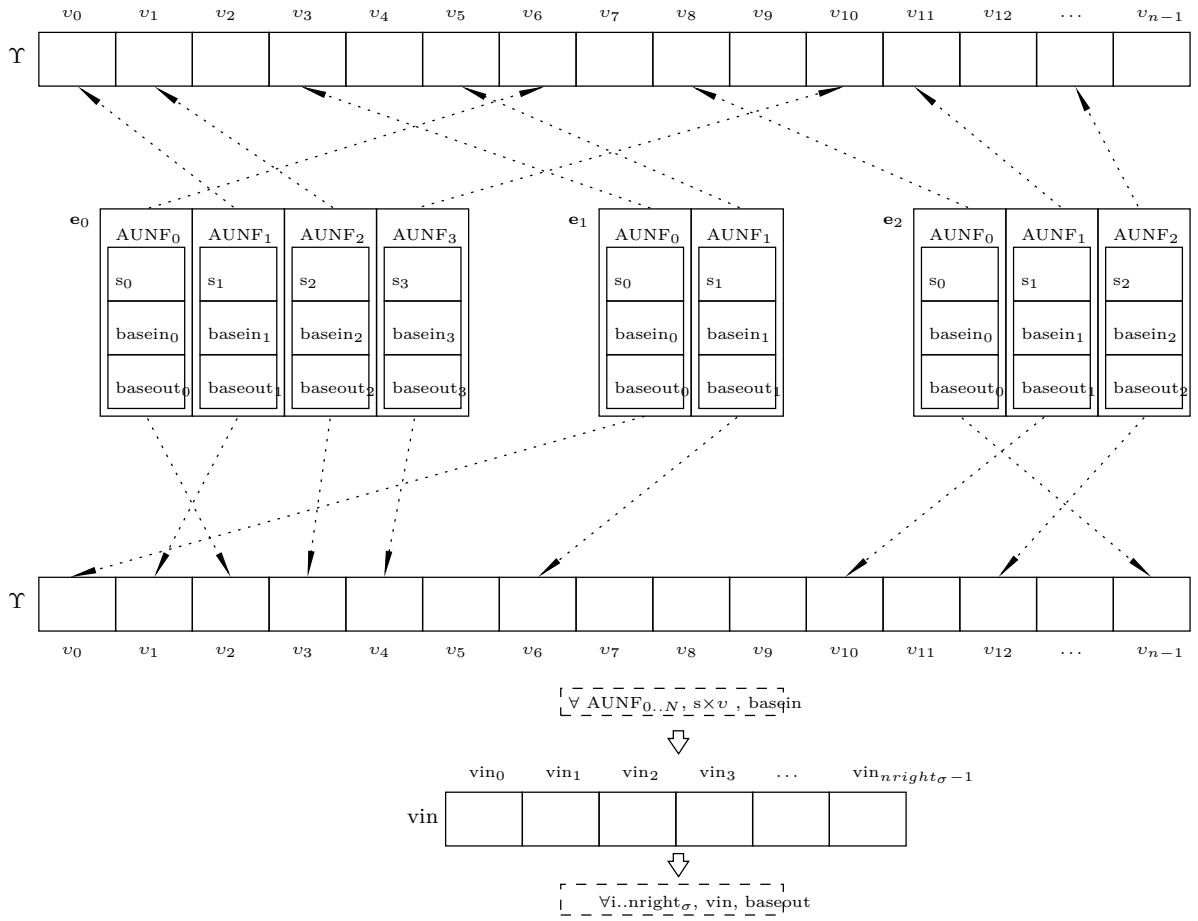


Figura 4.6: Operação do Algoritmo Split com permutações e identidades na parte estruturada.

Desta maneira, um AUNF conhece a posição de entrada a ser usada e também sabe onde colocar o valor multiplicado no vetor de saída. Essas são as únicas informações necessárias para que seja executado o método, dado que agora só matrizes do tipo identidade estão presentes na parte que compete ao Algoritmo *Shuffle*, não sendo necessária a geração de fatores normais, apenas a multiplicação do escalar pelo vetor em posições chave.

A Figura 4.6 explica a Equação 4.38 de forma gráfica. É necessário criar um vetor auxiliar de tamanho $nright_\sigma$ (equivalendo à seguinte parte da equação: $\prod_{i=\sigma+1}^{N} n_i$) e multiplicar o escalar s de acordo com o número de AUNFs existentes ($\prod_{i=1}^{\sigma} nz_i$).

Os acessos às posições dos vetores na implementação com permutações possuem custos computacionais associados pois, como explicado na Seção 4.5 precisam recalcular deslocamentos na estrutura tensorial de acordo com a nova ordem das matrizes nos termos. Como será visto no Capítulo 7 que trata sobre os resultados obtidos, estes custos não inviabilizam a solução, sendo indireções nos acessos aos índices dos vetores de acordo com o reordenamento escolhido.

Estas análises finalizam o capítulo sobre a solução de descritores Markovianos. Constatou-se que o Algoritmo *Split* possui restrições ao delimitar a divisão dos termos tensoriais, seja em função das identidades existentes ou das dependências funcionais. Atualmente o algoritmo é executado nas primeiras iterações para coleta de tempos de execução para então decidir quais cortes aplicará nas iterações seguintes. Esta escolha não segue um procedimento determinístico, baseado nas características das matrizes, trata-se de uma heurística baseada nas amostras das execuções. O próximo capítulo tratará das estratégias de reordenação do descritor para otimizar a solução de modelos complexos.

5. ESTRATÉGIAS DE RE-ESTRUTURAÇÃO

O objetivo deste capítulo é tratar das estratégias para escolha de um ponto de corte σ no qual re-estruturam-se termos tensoriais de descritores Markovianos através de permutações, objetivando a otimização da solução do Algoritmo *Split*. As estratégias que serão estudadas servirão de base para a discussão dos resultados apresentados no Capítulo 7. A Seção 5.1 apresenta análises preliminares para uma maior compreensão do problema da determinação de σ para, na Seção 5.2 discutir as re-estruturações que podem ser aplicadas para tratar descritores clássicos e generalizados. A Seção 5.3 mostra um estudo das características de termos tensoriais e, na Seção 5.4, apresenta uma previsão dos custos computacionais em termos de multiplicações de ponto-flutuante.

5.1 Considerações preliminares

Como explicado anteriormente, o Algoritmo *Split* multiplica os termos tensoriais dos descritores Markovianos de forma híbrida, basicamente de duas formas distintas. De um lado do termo (à esquerda do ponto de corte σ) aplica-se a solução esparsa e à direita de σ, efetua-se a multiplicação aplicando o Algoritmo *Shuffle*. Neste trabalho, esta divisão dos termos estabelecida para os termos tensoriais será definida como *parte esparsa* (esquerda de σ) e *parte estruturada* (direita de σ), respectivamente (maiores informações na Tabela 4.1).

Devido às características dos termos tensoriais, o desafio encontra-se na determinação do ponto de corte σ e divisão de cada termo tensorial tal que haja um equilíbrio entre o tempo gasto para execução da MVD e a memória extra necessária. Entre as principais características para a determinação do escopo destas tarefas, interessa-se pelas matrizes identidade dos termos tensoriais, pelas avaliações de elementos funcionais e pela quantidade de escalares não nulos em cada matriz. Neste sentido, nota-se a importância das sincronizações descritas pelos modelos, as quais encontram-se intrinsicamente relacionadas às esparsidades de cada matriz, como exemplificado nas Seções 4.3.1 e 4.3.2. Estas cooperações ou interações influenciam diretamente na quantidade de matrizes identidade na composição dos termos tensoriais bem como na memória adicional a ser gasta.

O ponto de corte σ escolhido para cada termo tensorial ditará a memória extra que será gasta na parte esparsa, logo, deseja-se evitar que um termo tensorial utilize uma quantidade massiva a ponto de inviabilizar a execução do Algoritmo *Split*. A melhor forma de se considerar esse aspecto é através da identificação da constituição de cada termo tensorial, verificando os tipos e esparsidades de cada matriz presente.

Cabe ressaltar que cada termo tensorial será tratado individualmente, ou seja, este será dividido de forma única com diferentes valores de σ para os diferentes termos. As análises estão focadas no formalismo das SAN mas, como mencionado anteriormente, podem ser aplicadas a quaisquer outros formalismos estruturados que utilizem descrições tensoriais para mapear a matriz de transição da MC correspondente.

Esta seção utiliza os conceitos explicados anteriormente no Capítulo 4, especificamente as definições referentes à Álgebra Tensorial Clássica e Generalizada descritas na Seções 4.1 e 4.2, descrição dos métodos de MVD da Seção 4.4 e o conceito de AUNF que é descrito na Seção 4.4.3. A re-estruturação dos termos tensoriais e a criação de escalares está fundamentada em duas propriedades da Álgebra Tensorial: propriedade da Pseudo-comutatividade (Equação 4.14 para ATC e Equação 4.25 para ATG) e propriedade da Decomposição Aditiva (Equação 4.33), ambas definidas no Capítulo 4.

Por definição, cada termo tensorial de um descritor Markoviano corresponde à ocorrência de um evento. Portanto, é necessário estudar quantos componentes este evento afeta, ou seja, deve ser determinada a esparsidade e o número de matrizes do termo tensorial que conterão informações sobre a ocorrência deste evento. Precisa-se descobrir o *grau de dependência* de cada termo para facilitar a análise das estratégias.

O *Grau de Dependência Constante* (GDC) e o *Grau de Dependência Generalizado* (GDG) estão relacionados, respectivamente, aos termos tensoriais clássicos e generalizados. Enquanto que o GDC informa a quantidade de autômatos que afetam a ocorrência de um evento, o GDG atesta o número de autômatos que são parâmetros de cada elemento funcional para a correta aplicação da função. Mais especificamente, tem-se a seguinte classificação para os termos tensoriais de uma SAN:

- Termo tensorial clássico (utiliza ATC): um termo clássico é um termo tensorial sem funções, contendo apenas matrizes com elementos constantes que indicam onde cada evento ocorre em cada autômato e com que taxa;
  - *GDC Alto*: um termo tensorial deste tipo envolve um valor superior à metade (seja t o total de matrizes do termo, a sua metade m corresponde a $m = \frac{t}{2}$) das matrizes que o compõem[1]. Ao envolver metade dos autômatos, implica na existência da outra metade ser composta por matrizes do tipo identidade;
  - *GDC Parcial*: este evento sincroniza suas atividades com menos da metade das matrizes existentes no termo tensorial;
- Termo tensorial generalizado (utiliza ATG): contém matrizes com elementos funcionais para serem avaliados. Termos deste tipo são classificados quanto ao grau de dependência generalizado, podendo ser:
  - *GDG Alto*: envolvem todas as matrizes do termo tensorial. Trata-se basicamente de uma função que consulta os estados de todos os autômatos, uma ou múltiplas vezes, para ser calculada;
  - *GDG Parcial*: envolvem apenas algumas matrizes do termo tensorial.

[1] Definiu-se dividir pela metade o termo tensorial apenas por questões de classificação, correspondendo a um compromisso razoável, dada a lista de matrizes dos termos tensoriais.

Termos tensoriais são compostos por matrizes de diferentes características. As matrizes de um termo podem ser classificadas da seguinte forma (seja nz o número de elementos não-nulos e n a dimensão de uma matriz):

- Identidade: a matriz é do tipo identidade, possuindo o valor 1 para a diagonal principal e o valor 0 para o restante das posições;
- Constante: trata-se de uma matriz com mais de um elemento não-nulo, com esparsidade variável (entre $\frac{2}{n}$ até $\frac{nz}{n}$);
- Elemento: trata-se de um caso especial de matriz constante, ou seja, é uma matriz esparsa com apenas um elemento não-nulo (esparsidade $\frac{1}{n}$);
- Funcional: indica que a matriz possui um elemento com uma função definida, e necessita de informações de outras matrizes do termo (para o escopo desta tese, de estados de outros autômatos) para sua correta avaliação.

Diferentes estratégias de corte devem ser tomadas para os diferentes tipos de termos tensoriais. Para os termos clássicos as características mais importantes são as dimensões das matrizes, a esparsidade e a ocorrência de matrizes identidade. Para os termos generalizados, além destas, consideram-se também as matrizes que são parâmetros da função e o lugar onde será feito o cálculo de avaliação, se na *parte esparsa* ou na *parte estruturada*.

A quantidade de avaliações de funções que são executadas no processo da MVD é um fator determinante para uma possível degradação de desempenho. Os custos computacionais envolvidos variam para cada modelagem, entretanto, dependendo do caso, contribuem em aumento significativo do número de operações que são efetuadas para a avaliação. Devido ao fato de que o método iterativo será potencialmente executado muitas vezes e para os mesmos parâmetros de função, constata-se que seria mais interessante avaliá-las uma vez salvando os valores calculados em estruturas de dados auxiliares. Contudo, é vantajoso utilizar as diretivas funcionais como primitivas de modelagem, pois aumentam-se as maneiras pelas quais é possível modelar sistemas contendo transições complexas. Neste sentido, uma prática é buscar mecanismos onde seja possível utilizar taxas funcionais ao mesmo tempo que na parte de solução seja usada a melhor maneira de se efetuar a MVD (estas discussões encontram-se mais adiante).

Quando as avaliações de funções são realizadas na parte esparsa, o descritor Markoviano generalizado é convertido em clássico, implicitamente. Intuitivamente, avaliar as funções na parte esparsa remove a sobrecarga de se ter que avaliá-la inúmeras vezes em potencialmente inúmeras iterações. Entretanto, uma análise mais abrangente se faz necessária para substanciar essas evidências.

Para começar esta análise mais aprofundada, é necessário definir precisamente as opções existentes ao dividir termos tensoriais. A seguir é feita uma análise sobre como proceder através do estudo de opções para os casos clássico e generalizado. Seja M o total de matrizes de um termo tensorial e seja V o número de matrizes envolvidas na sincronização dado que, por definição, $V = 1$

não acontece ($1 < V < M$), pois não é permitida a definição de um evento sincronizante envolvendo apenas um autômato:

- Para termos clássicos, reordenar o termo tensorial e reconfigurá-lo, privilegiando o deslocamento das matrizes do tipo identidade para o lado estruturado:
  - caso desvantajoso: $V = M$, ou seja, não existem identidades, *i.e.*, o evento sincronizante envolve todas as matrizes do termo tensorial;
  - caso médio: um valor V tal que $2 < V < M$. Esse caso significa que existem matrizes do tipo identidade no termo;
  - caso vantajoso: $V = 2$, ou seja, somente dois autômatos estão sincronizando atividades. Também indica que os AUNFs a serem criados são iguais ao produtório dos elementos não-nulos contabilizados até a primeira matriz identidade, o qual deve ser o valor escolhido para σ.
- Para termos generalizados, escolhe-se basicamente *onde* pode se avaliar as funções:
  - avaliar funções na parte esparsa: implica em converter implicitamente o termo tensorial generalizado para um termo tensorial clássico, uma vez que esse procedimento avalia diretamente os elementos funcionais transformando-os em elementos constantes;
  - avaliar funções na parte estruturada: aplicar as avaliações de funções como são feitas no Algoritmo *Shuffle*, ou seja, na parte estruturada, um número de vezes que depende dos valores dos deslocamentos *nright* e *nleft* calculados respectivamente para as operações desta parte.

O objetivo na escolha do corte e reordenamentos é o de minimizar o tempo gasto para realizar a MVD em cada termo. A escolha de qual matriz será posicionada em cada parte deve estar de acordo com a classificação do termo tensorial e as características envolvidas. Dessa forma, o uso de permutações das matrizes dos termos tensoriais é mandatório uma vez que, para os termos clássicos, é desvantajoso posicionar matrizes identidade na parte esparsa (pois criam AUNFs desnecessários), devendo estas matrizes serem deslocadas para a *parte estruturada*. Para os termos generalizados, precisa-se dos parâmetros das funções, ou seja, das informações contidas nas matrizes que fazem parte do conjunto de dependência da função a ser avaliada.

Se um dado termo tensorial possuir uma função em uma dada matriz e escolhe-se realizar as avaliações na parte esparsa. Todas as matrizes que a função depende, inclusive a matriz onde a função está definida, devem estar nesta mesma parte. O resto das matrizes que não dependem da função podem ser enviadas para qualquer um dos lados do corte, sendo apenas necessário estudar qual parte será mais vantajosa dependendo da otimização a ser feita, em termos de memória ou de tempo. Caso a matriz seja do tipo identidade que não é parâmetro para a função, esta é melhor tratada na parte estruturada, fato já constatado nos estudos realizados sobre o método *Shuffle* que descarta estas matrizes do cálculo.

Para o caso onde deseja-se aplicar as avaliações na parte estruturada, é necessário apenas que a função permaneça também nessa parte, sendo que as outras matrizes podem ser posicionadas na parte esparsa. Novamente, neste caso é interessante deslocar as matrizes identidade para a parte estruturada.

5.2 Re-estruturações de descritores Markovianos

A re-estruturação de descritores Markovianos (e seus termos tensoriais), clássicos ou generalizados, deve ser considerada de forma distinta, dada a natureza das matrizes que os compõem. No caso de descritores clássicos, apenas matrizes constantes, elementos e identidades são utilizadas, enquanto que para os descritores generalizados, matrizes funcionais estão presentes. A seguir, serão mostrados exemplos de um termo tensorial de cada tipo de descritor e a aplicação de re-estruturações das matrizes que os compõem.

5.2.1 Descritor Markoviano clássico

Para este exemplo, será utilizado um termo tensorial composto por diversos componentes, neste caso, por seis autômatos. Para o mapeamento do evento que corresponde a este termo, são necessárias seis matrizes que mapeiam as transições dos elementos. Seus respectivos valores são: $nz = \{3,3,2,2,3,2\}$, dimensões $= \{3,3,2,3,3,2\}$ e tipos $= \{$*identidade*, *identidade*, *constante*, *constante*, *identidade*, *identidade*$\}$.

Seja σ o valor do parâmetro de corte do termo tensorial, com PSS dado por $|\mathcal{X}| = 3 \times 3 \times 2 \times 2 \times 3 \times 2 = 216$ estados. As matrizes podem ser permutadas, alterando a ordem original definida no modelo e as divisões possíveis variam entre zero (σ_0) e seis (σ_6). Quando o corte é definido em σ_0 existe o máximo de estruturação (abordagem *Shuffle*) e para σ_6, nenhuma estruturação será feita, ou seja, o termo será tratado de forma puramente esparsa.

A seguir, é apresentado o termo tensorial exemplo:

$$\underset{\sigma_0}{\overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_1}{\overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_2}{\overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 4 \\ 3 & 0 \end{pmatrix}}} \otimes \underset{\sigma_3}{\overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}}} \otimes \underset{\sigma_4}{\overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_5}{\overset{\mathcal{A}^{(6)}}{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}} \underset{\sigma_6}{}$$

Para este exemplo, as possibilidades de re-estruturação são restritas: ou as matrizes identidade são deslocadas para a direita, ou para a esquerda do ponto de corte σ. Como discutido anteriormente, matrizes identidade na parte esparsa apenas ocupam memória extra desnecessariamente. Neste sentido, o mais interessante seria mover as matrizes contantes para a parte esparsa, fixando o ponto de divisão logo a seguir, deixando matrizes identidade na parte estruturada, tal como:

$$\underset{\sigma_0}{\overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 4 \\ 3 & 0 \end{pmatrix}}} \otimes \underset{\sigma_1}{\overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}}} \overset{\downarrow}{\otimes} \underset{\boldsymbol{\sigma_2}}{\overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_3}{\overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_4}{\overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \otimes \underset{\sigma_5}{\overset{\mathcal{A}^{(6)}}{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}} \underset{\sigma_6}{}$$

parte esparsa — *parte estruturada*

Neste exemplo, o ponto de corte seria fixado em σ_2 sendo necessários quatro AUNFs ($2 \times 2 = 4$), conforme a esparsidade das matrizes do lado esparso. As matrizes foram reordenadas para que na parte estruturada apenas matrizes do tipo identidade estejam definidas. O subespaço criado a partir do ponto de corte ($nright_\sigma$) seria igual a $3 \times 3 \times 3 \times 2 = 54$, onde o Algoritmo *Split* multiplicaria cada AUNF criado por este subespaço, para o evento correspondente a este termo tensorial, e não executaria o Algoritmo *Shuffle* na parte estruturada, aproveitando a otimização das matrizes identidade, discutida na Seção 4.4.2.

Na próxima seção será mostrado um exemplo de um termo tensorial de um descritor Markoviano generalizado, com pelo menos uma matriz contendo um elemento funcional.

5.2.2 Descritor Markoviano generalizado

Para o seguinte termo tensorial generalizado, composto por cinco autômatos (no caso, cinco matrizes), com respectivos valores para $nz = \{2,3,1,1,4\}$, dimensões $= \{2,3,2,3,4\}$, tipos $= \{$*constante*, *identidade*, *funcional*, *elemento*, *identidade*$\}$. Define-se a função f na matriz que representa o autômato $\mathcal{A}^{(3)}$ por[2]:

$$f \leftarrow \Big(st\,\mathcal{A}^{(1)} = 1 \;\&\; st\,\mathcal{A}^{(5)} = 0 \Big) \times r$$

onde $r \in \mathbb{R}^*_+$ e st uma função *booleana* que recebe dois parâmetros, um autômato e um estado, retornando uma avaliação baseada no estado corrente. Como no caso anterior, seja σ o valor do parâmetro de corte do termo tensorial, com PSS dado por $|\mathcal{X}| = 2 \times 3 \times 2 \times 3 \times 4 = 144$ estados:

$$\underset{\sigma_0}{\overset{\mathcal{A}^{(1)}}{\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}}} \underset{\sigma_1}{\overset{g}{\otimes}} \underset{\sigma_2}{\overset{\mathcal{A}^{(2)}}{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}} \underset{\sigma_3}{\overset{g}{\otimes}} \underset{}{\overset{\mathcal{A}^{(3)}}{\begin{pmatrix} 0 & 0 \\ f & 0 \end{pmatrix}}} \underset{\sigma_4}{\overset{g}{\otimes}} \underset{}{\overset{\mathcal{A}^{(4)}}{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 5 & 0 \end{pmatrix}}} \underset{\sigma_5}{\overset{g}{\otimes}} \underset{}{\overset{\mathcal{A}^{(5)}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}}$$

Os valores possíveis para dividir o termo tensorial variam de acordo com os pontos de corte, de σ_0 a σ_5. Nota-se que, ao aplicar o Algoritmo *Split* para σ_0, a abordagem escolhida será a equivalente ao Algoritmo *Shuffle* e, para σ_5, será a correspondente ao Algoritmo *Esparso*.

[2] A função verifica o estado dos autômatos $\mathcal{A}^{(1)}$ e $\mathcal{A}^{(5)}$ e define, caso verdadeiro, uma taxa genérica r para f.

Pode-se escolher avaliar a função f na parte estruturada ou na parte esparsa do termo tensorial. Caso fosse escolhido realizar as avaliações de funções na parte *estruturada*, o termo tensorial precisaria ser reordenado da seguinte maneira:

$$\underset{\sigma_0}{}\ \overset{\mathcal{A}^{(2)}}{\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}}\ \underset{\sigma_1}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(4)}}{\begin{pmatrix}0&0&0\\0&0&0\\0&5&0\end{pmatrix}}\ \underset{\boldsymbol{\sigma_2}}{\overset{\otimes}{\underset{g}{\downarrow}}}\ \overset{\mathcal{A}^{(1)}}{\begin{pmatrix}0&1\\2&0\end{pmatrix}}\ \underset{\sigma_3}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(5)}}{\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}\ \underset{\sigma_4}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(3)}}{\begin{pmatrix}0&0\\f&0\end{pmatrix}}\ \underset{\sigma_5}{}$$

parte esparsa | *parte estruturada*

Neste caso, o conjunto das matrizes que fazem parte da dependência funcional de f é dado por $Dep(f) = \{\mathcal{A}^{(1)}, \mathcal{A}^{(5)}\}$. O corte escolhido para esse caso seria σ_2, com uma matriz do tipo identidade e uma matriz constante para a parte estruturada e a outra matriz identidade, uma matriz constante e a matriz funcional para a parte esparsa, conservando as dependências necessárias para que o método *Shuffle* consiga avaliar apropriadamente f. Para este caso de corte igual a σ_2, teriam-se três AUNFs[3] $(3 \times 1 = 3)$.

No caso das avaliações de funções serem feitas na parte *esparsa*, a matriz onde a função está definida entra obrigatoriamente no conjunto de dependências funcionais, sendo igual a $Dep(f) = \{\mathcal{A}^{(1)}, \mathcal{A}^{(5)}, \mathcal{A}^{(\mathbf{3})}\}$. Para este caso, o termo tensorial seria reorganizado da seguinte forma:

$$\underset{\sigma_0}{}\ \overset{\mathcal{A}^{(1)}}{\begin{pmatrix}0&1\\2&0\end{pmatrix}}\ \underset{\sigma_1}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(5)}}{\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}\ \underset{\sigma_2}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(3)}}{\begin{pmatrix}0&0\\f&0\end{pmatrix}}\ \underset{\boldsymbol{\sigma_3}}{\overset{\otimes}{\underset{g}{\downarrow}}}\ \overset{\mathcal{A}^{(2)}}{\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}}\ \underset{\sigma_4}{\overset{\otimes}{\underset{g}{}}}\ \overset{\mathcal{A}^{(4)}}{\begin{pmatrix}0&0&0\\0&0&0\\0&5&0\end{pmatrix}}\ \underset{\sigma_5}{}$$

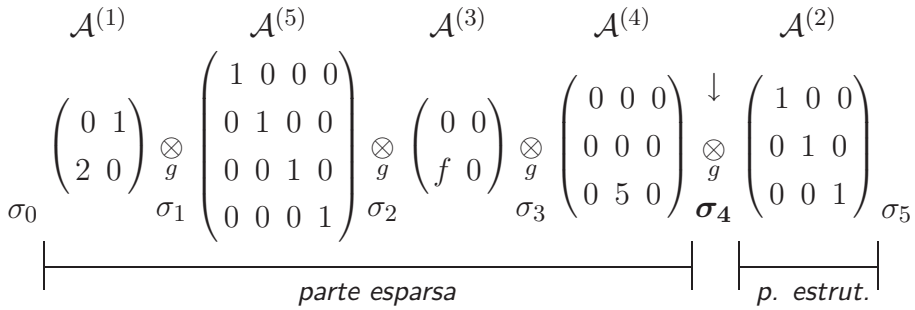
parte esparsa | *parte estruturada*

Para este caso, o ponto de corte σ_3 foi escolhido e tem-se oito AUNFs $(2 \times 4 \times 1 = 8)$. Para esta reconfiguração, pode-se escolher enviar a matriz *elemento* $(\mathcal{A}^{(4)})$ da parte estruturada para a parte esparsa, sem o aumento do número de AUNFs permanecendo apenas matrizes identidade na parte estruturada. Contudo, não seria possível deslocar a matriz identidade de $\mathcal{A}^{(5)}$ para a parte estruturada, já que a função não poderia ser avaliada dado que uma das matrizes que ela depende está posicionada em uma parte inacessível (não é possível avaliar a função com sucesso devido à falta de parâmetros).

Uma alternativa válida, entretanto, com a preocupação de não aumentar o número de AUNFs que são gerados, é enviar a matriz do tipo elemento definida em $\mathcal{A}^{(4)}$ para a parte esparsa, deixando apenas matrizes identidade na parte estruturada.

[3] Um AUNF é calculado como o produtório dos elementos não-nulos de uma matriz, de 0 a σ. Maiores informações na Seção 4.4.3.

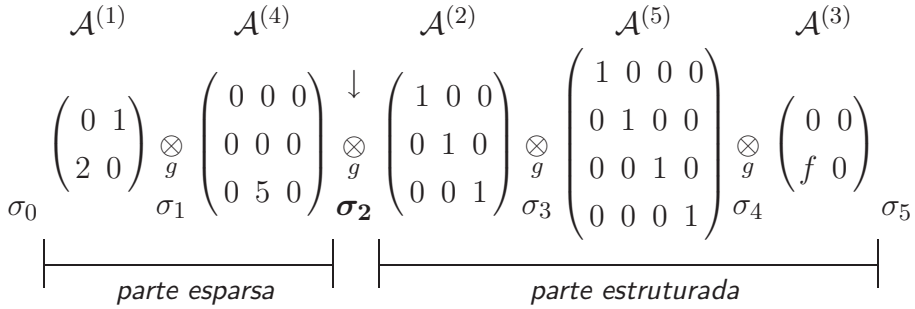
Nesta alternativa, o ponto de corte é igual a σ_4 e o termo tensorial seria assim reorganizado:



Neste exemplo nota-se que a parte estruturada não terá custo computacional associado, pois a matriz restante é do tipo identidade. Outra constatação é que matrizes do tipo elemento sempre podem ser deslocadas para o lado esparso sem um ônus relativo ao aumento do número de AUNFs necessários. Caso essa mesma matriz permanecesse na parte estruturada, ainda assim contribuiria para aumentar os cálculos de deslocamentos à esquerda e à direita do vetor na multiplicação, pois são mais relevantes para o Algoritmo *Shuffle*, em primeiro lugar, as dimensões das matrizes e, em segundo lugar, as quantidades de elementos não-nulos [FER98b].

Ao eliminar a restrição de permitir que funções na parte estruturada possam acessar elementos da parte esparsa, propõe-se mais uma alternativa de reordenamento, onde deixa-se apenas a matriz que contem o elemento funcional f, mais as identidades na parte estruturada e as demais matrizes na parte esparsa. Este caso é possível, apesar das dependências funcionais estarem em partes distintas, pois, ao se calcular os AUNFs, guardam-se os índices de linha das matrizes que o originaram.

Cabe ressaltar que o contrário seria impossível (*i.e.*, a matriz que contém a função f ser colocada na parte esparsa e uma ou as demais matrizes na parte estruturada) pois f necessita destes parâmetros que encontram-se na parte estruturada do termo tensorial. Por pertencerem ao subespaço à direita do corte onde o Algoritmo *Shuffle* será executado, os parâmetros da função à esquerda do corte, encontram-se indisponíveis. O próximo caso possui a vantagem de tornar desnecessário o armazenamento dos AUNFs relativos às matrizes identidade e trata-se de uma boa maneira de se reordenar os termos tensoriais.



Como as matrizes identidade serão desconsideradas pela parte estruturada e é possível avaliar as funções com sucesso (pois os parâmetros encontram-se obrigatoriamente à esquerda da definição da função), esta reordem oferece um baixo número de AUNFs necessários, pois a parte esparsa possui uma matriz do tipo elemento.

É importante salientar que não é necessário adotar uma solução puramente esparsa para esse termo tensorial (corte em σ_5), pois aumentaria o número de AUNFs e seria oneroso, pois o método *Shuffle*, por definição, não será executado e menos operações para cálculos de deslocamentos no vetor para a multiplicação serão feitos (estas análises serão feitas nas próximas seções). Igualmente desnecessário é se gastar espaço de memória armazenando-se AUNFs formados a partir da combinação de elementos de matrizes do tipo identidade. Também verifica-se que, na prática, o aumento da memória não necessariamente implica em um ganho de desempenho, ainda mais para esse caso, que armazenaria os AUNFs com mesmos valores de escalares e nunca aproveitaria os saltos na estrutura tensorial para melhorar a execução.

Os exemplos de re-estruturações aqui demonstrados evidenciaram a importância em se alterar a ordem natural das matrizes do termo tensorial para obter uma melhor organização e consequente execução do processo de MVD com o Algoritmo *Split*. O custo computacional de se usar reordenamentos é baixo e realizado de forma transparente para o usuário, pois trata-se fundamentalmente de cálculos dos deslocamentos[4], ou seja, novos valores para *nright*, *nleft* e *njump*, equivalendo respectivamente aos saltos do subespaço à direita, à esquerda e os saltos na estrutura tensorial de acordo com a dimensão da matriz não-identidade no fator normal. Estes exemplos mostraram que devem ser adotadas diferentes estratégias para tratamento dos diferentes termos tensoriais. Foi igualmente constatado que a flexibilidade do Algoritmo *Split* possibilita a definição de pontos de corte personalizados, de acordo com as características presentes nos termos tensoriais.

Uma vez que o problema foi investigado com as possibilidades de divisão de cada termo tensorial, nota-se que um conjunto finito de características afetam o desempenho da solução da MVD. Este conjunto é dado por: a) as dimensões das matrizes correspondente aos estados dos autômatos, b) o total de elementos não-nulos, c) o total de avaliações de funções que são feitas, e d) total de matrizes identidade nos termos tensoriais. Estas características afetam diretamente a complexidade dos métodos escolhidos para a MVD.

A principal ideia deste capítulo é definir as principais estratégias para dividir um termo tensorial de uma maneira que não onere a memória gasta e que otimize o tempo de solução. Como explicado anteriormente, esta divisão implicou na obrigatoriedade de reorganização do termo tensorial. Essa reconfiguração não é custosa em termos computacionais, pois trata-se de cálculos de deslocamentos que são computados na primeira vez e depois apenas consultados na execução dos métodos. O objetivo então é aplicar a melhor transformação ao termo tensorial e dividi-lo em partes que otimizem o tempo de resposta de modelos. Para tanto, é necessário estudar os efeitos de cada propriedade dos termos tensoriais para determinar a melhor maneira de se fazer essa operação. A seguir é realizado um estudo sobre estas propriedades visando a solução de descritores baseados em ATC e ATG. Este estudo permitirá antecipar os desempenhos esperados de cada experiência a ser realizada no Capítulo 7, referente aos resultados obtidos.

[4] Esses cálculos são melhor explicados na Seção 4.5.

5.3 Características de termos tensoriais

Uma vez definidos os compromissos de desempenho do Algoritmo *Split* e as formas de divisão dos termos tensoriais para otimizar o tempo gasto na solução de modelos baseados em descritores Markovianos, a próxima etapa é estudar as implicações nos custos de execução do algoritmo. Um termo tensorial é composto por uma lista de matrizes de diferentes dimensões, tipos e esparsidades. Cada matriz contém informação relevante sobre como as entidades sincronizam atividades, ou seja, como estes *interferem* uns nos outros. O grau desta interferência dita a formação das matrizes dos termos tensoriais, ou seja, serão criadas matrizes identidade para representar que um evento não ocorre em um dado elemento (no caso de SAN, em um autômato) e matrizes do tipo elemento, indicando a ocorrência de um evento de acordo com uma taxa de ocorrência. Cabe ressaltar que a terminologia é aqui usada sem perda de generalidade para as seguintes atividades: *envolvimento* (ou envolver), *interferência* (ou interferir), *afetação* (ou afetar), *i.e.*, no contexto deste trabalho, serão usadas de forma a representar a mesma ideia.

A seguir, um estudo das implicações de cada tipo de característica existente em termos tensoriais e o que correspondem ao nível do algoritmo que será executado:

1. dimensão de cada matriz: influenciam os subespaços à direita e à esquerda do corte σ dos termos tensoriais, ou seja, implicam no número de vezes que o algoritmo é executado por iteração;

2. tipo da matriz: uma matriz, como discutido anteriormente, pode ser constante, identidade, elemento ou funcional. Cada tipo pode otimizar um diferente aspecto, por exemplo, uma matriz funcional pode otimizar o número de avaliações que são feitas, enquanto que matrizes elemento e identidades dizem respeito principalmente à memória extra que será gasta bem como o número de operações de multiplicação;

3. permutação do termo: um termo tensorial pode sofrer uma transformação na ordem original das suas matrizes para execução do processo de MVD;

4. ponto de corte σ: o ponto de corte determina, em última análise, o quanto de memória extra precisará ser armazenada para efetuar o Algoritmo *Split*. Este valor deve ser escolhido de forma a maximizar também o tempo de solução levando-se em conta otimizações conhecidas tais como a) identidades deslocadas para a parte estruturada e b) matrizes elemento na parte esparsa, como mencionado anteriormente.

A seguir é feita uma análise mais aprofundada sobre estas características e seus efeitos na complexidade computacional envolvida, começando-se pelo estudo do efeito das identidades nos termos tensoriais.

5.3.1 Matrizes identidade na multiplicação de termos tensoriais

O efeito das matrizes do tipo identidade no termo tensorial é observado diferentemente em termos tensoriais clássicos e em termos generalizados. Em termos clássicos, as identidades podem fazer parte do lado esquerdo do corte, executando o Algoritmo *Esparso*, ou na parte estruturada, que executará o Algoritmo *Shuffle*. Como mencionado anteriormente, matrizes do tipo identidade na parte esparsa aumentam o número de AUNFs necessários sem guardar informação relevante. Se estas matrizes identidade fossem deslocadas para a parte estruturada, levariam à execução de um menor número de multiplicações em ponto flutuante para o cálculo dos deslocamentos que são necessários ao Algoritmo *Shuffle*.

Esta redução de operações também é verificada se a parte estruturada contiver *apenas* matrizes do tipo identidade. Nesse caso, somente é necessário o tempo inicial gasto para se descobrir os AUNFs de cada termo tensorial e depois multiplicá-los em índices pré-calculados para os vetores usados no método. Este caso, para os termos tensoriais generalizados, implica que os elementos funcionais foram avaliados na parte esparsa, *i.e.*, este termo tensorial não precisa mais ser considerado um termo generalizado, sendo a partir desse momento um termo clássico, sem haver a necessidade de outras avaliações de função em operações posteriores envolvendo elementos funcionais.

Ao não precisar mais avaliar funções na parte estruturada, descarta-se toda a complexidade envolvida para realizar esta operação. Com isso, o termo generalizado é convertido para um termo clássico, em tempo de execução, pois as estruturas de dados auxiliares para armazenamento dos AUNFs contém os valores das funções já avaliadas. Caso o número de AUNFs existentes não inviabilizem a execução do Algoritmo *Split* (passando-se dos limites disponíveis de memória, por exemplo), esta é tecnicamente uma boa relação custo/benefício em termos de número de operações de multiplicações.

O ganho relacionado ao efeito das identidades na multiplicação dos termos tensoriais pode ser avaliado de duas formas: com permutação das matrizes do termo ou sem permutação e utilizando algum critério de escolha do ponto de corte σ. Como estas permutações são, na verdade pré-cálculos de deslocamentos na estrutura tensorial, realizados apenas uma vez no início do método, seus custos são negligenciáveis do ponto de vista da execução do processo de MVD como um todo.

A existência de matrizes do tipo identidade está fortemente relacionada ao termo tensorial informando os autômatos que um dado evento interfere. Dado um evento sincronizante e, se este está sincronizando uma transição com todos os autômatos, este termo não será composto por nenhuma identidade e sim por matrizes do tipo elemento ou matrizes constantes. A abordagem flexível proposta pelo Algoritmo *Split* auxilia na escolha de opções balanceadas para escolha do ponto de corte σ, baseadas na memória disponível.

Para o caso onde as identidades serão tratadas na parte estruturada, todos os cálculos necessários são desconsiderados (ver Algoritmo 4.3, linhas de 13 a 32). O algoritmo somente calcula os AUNFs necessários, monta um vetor de tamanho $nright_\sigma$ com posições vindas dos índices de entrada de cada AUNF e após multiplicar o escalar |AUNF| vezes, salva no vetor v de acordo com os índices pre-

calculados e armazenados pelas posições de saída. Portanto, ao realizar somente estas operações, a complexidade em termos de multiplicações em ponto flutuante, caso garanta-se que somente existam identidades na parte estruturada, resulta em (assumindo que os AUNFs foram precalculados para cada termo tensorial):

$$C = \prod_{i=1}^{\sigma} nz_i \prod_{i=\sigma+1}^{N} n_i \tag{5.1}$$

Esta equação para complexidade é diferente da equação correspondente à versão original do *Split*, sem considerar que permutações possam ser usadas para o benefício crucial do método. A Equação 5.2 da Seção 4.4.3 é re-apresentada a seguir:

$$C = \left(\prod_{i=1}^{\sigma} nz_i\right)\left[\left(\prod_{i=\sigma+1}^{N} n_i\right) + \left(\prod_{i=\sigma+1}^{N} n_i \times \sum_{\substack{i=\sigma+1 \\ \text{iff } Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}\right)\right] \tag{5.2}$$

Esta equação mostra os cálculos necessários para a parte esparsa e a parte estruturada. A parte $\prod_{i=1}^{\sigma} nz_i$ corresponde ao cálculo dos AUNFs e a parte $\prod_{i=\sigma+1}^{N} n_i$ equivale ao tratamento da parte estruturada a partir de blocos de tamanho $nright_\sigma$, ou seja, o produtório das dimensões de todas as matrizes (n_i), computadas de $\sigma+1$ até N. A parte final corresponde à chamada do Algoritmo *Shuffle* tradicional. Estas equações de complexidade demonstram o ganho do *Split* sobre o *Shuffle* quando a parte estruturada possui apenas matrizes do tipo identidade.

5.3.2 Custo das avaliações dos elementos funcionais e permutações

A avaliação de funções é uma operação que depende de diferentes fatores em um modelo SAN. O custo envolvido depende fundamentalmente das seguintes características [FER98c]:

1. Os argumentos da função. Estes valores são pré-determinados, referentes aos estados dos autômatos;

2. A avaliação da função propriamente dita. Uma vez argumentos da função são conhecidos, esta pode ser avaliada, retornando o valor da avaliação;

3. O número de avaliações que são realizadas. Esse valor é dependente do bloco correspondente ao subespaço da esquerda do índice de onde a função está definida. O método para avaliação de funções pode ser chamado inúmeras vezes, de acordo com o Algoritmo 4.3. O método trata as matrizes de acordo com uma ordem de permutação pré-estabelecida e converte os seus elementos funcionais em elementos constantes.

Ao dividir o termo tensorial de forma que as funções das matrizes sejam avaliadas na parte estruturada, aumenta-se o número de operações que são feitas, uma vez que as avaliações necessárias são realizadas em cada iteração e sempre resultam no mesmo conjunto de valores. O oposto desta escolha é dividir o termo tensorial deslocando-se as matrizes dependentes e a matriz que contém o

elemento funcional para a parte esparsa e executando-se o método de avaliação de função apenas uma vez no início do processo de solução, armazenando os AUNFs convertidos para valores escalares. Ao se fazer isso, os termos tensoriais generalizados são convertidos para termos clássicos e não existem mais avaliações de função como existia anteriormente e o método deve rodar mais rapidamente.

Existem duas maneiras de se efetuar as avaliações de funções, na parte esparsa ou na parte estruturada. A seguir, analisam-se ambos os casos, iniciando pelo caso da avaliação na parte esparsa. O pior caso é definido quando a função depende de matrizes do tipo identidade. A única restrição do Algoritmo *Split* é ter que possuir todas as matrizes que a função depende na parte esparsa, pois o método não acessa a parte estruturada. Ter identidades na parte esparsa implica que muitos AUNFs serão potencialmente criados sem necessidade, pois os escalares que compõem o AUNF não são alterados quando multiplicados pelos elementos das matrizes identidade. Já na parte estruturada, duas opções são possíveis, manter a lista de matrizes dependentes nessa parte (essa lista é criada a partir dos parâmetros da função) ou então enviá-las para a parte esparsa que pre-computará os AUNFs necessários. Também pode-se deixar apenas a matriz onde a função está definida e deixar apenas as identidades na parte estruturada. Essa abordagem auxilia a execução do Algoritmo *Split*, sendo similar ao caso clássico onde as identidades estão na parte estruturada.

No caso das permutações, dada a natureza de cada termo tensorial, existem muitas formas de se reordenar suas matrizes internas, compostas por diversos tipos e esparsidades. A busca é orientada pela descoberta de uma maneira razoável de se reorganizar cada termo tensorial, minimizando o tempo a ser gasto para solução o modelo.

Para permutar um termo tensorial e obter uma ordem de posicionamento é necessário calcular novos índices referentes aos deslocamentos que serão utilizados no algoritmo de MVD para que ocorram os saltos apropriados na estrutura tensorial. Esse reposicionamento faz com que o método seja executado como anteriormente, mas acessando os elementos da estrutura de outra forma, no caso, permutada. Estes custos de recálculos de índices são negligenciáveis, pois são calculados apenas na primeira vez que o processo é executado e são utilizados sistematicamente para cada iteração. O problema da reordenação é descobrir um posicionamento ideal que, quando combinado com o ponto de corte, auxilia na diminuição do tempo requerido para solucionar um modelo e retornar o vetor de probabilidade estacionário ou transiente, dependendo da análise.

Cada termo tensorial pode ser reorganizado de uma maneira diferente, devido principalmente aos elementos que o compõem. Este reposicionamento pode levar em consideração quando será executado o Algoritmo *Split*, de acordo com a memória disponível do sistema, a esparsidade do termo tensorial, o número de identidades e o número de matrizes que serão vistas como os argumentos da função para termos generalizados.

5.4 Discussão

A Tabela 5.1 mostra um comparativo das complexidades de cada escolha do ponto de corte σ de termo tensoriais para o Algoritmo *Shuffle* e para o Algoritmo *Split*.

Tabela 5.1: Complexidades existentes para os Algoritmos *Shuffle* e *Split*.

Complexidade *Shuffle*		$\sum_{i=1}^{N} nleft_i \times nright_i \times nz_i = \prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}$
		Complexidade do método completo, sem otimizações
Complexidade *Shuffle* c/otimização		$\prod_{i=1}^{N} n_i \times \sum_{\substack{i=1 \\ \text{iff } Q^{(i)} \neq Id}}^{N} \frac{nz_i}{n_i}$
		Só multiplica a matriz se esta não é uma identidade
Complexidade *Split*	método normal	$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)+\left(\prod_{i=1}^{N} n_i \times \sum_{i=1}^{N} \frac{nz_i}{n_i}\right)\right]$
		Custo para calcular um AUNF e executar Algoritmo *Shuffle*
	identidades na parte estruturada	$\left(\prod_{i=1}^{\sigma} nz_i\right)\left[(\sigma-1)+\left(\prod_{i=\sigma+1}^{N} n_i\right)\right]$
		Trata-se do custo para se calcular e multiplicar os AUNFs
	identidades na parte estruturada	$\prod_{i=1}^{\sigma} nz_i \prod_{i=\sigma+1}^{N} n_i$
		Mesmo que o anterior, mas AUNFs são precalculados
	avaliações na parte esparsa	As avaliações de funções dependem da quantidade de parâmetros necessários, *i.e.*, o número de autômatos que são necessários para o cálculo da complexidade. Custo de avaliação negligenciável, dependendo da definição da função, mais custo do Algoritmo *Split*
	avaliações na parte estruturada	Avaliam-se as funções na parte estruturada, mais custo do cálculo dos AUNFs
	avaliações na parte esparsa, identidades na parte estruturada	Neste caso, todas as avaliações são feitas na parte esparsa e o Algoritmo *Shuffle* não é executado, pois só existem identidades na parte estruturada

No caso do *Shuffle* (σ_0), tem-se a complexidade do método e a otimização das matrizes identidade. Para a complexidade do *Split* ($> \sigma_0$), tem-se outras possibilidades, tais como execução do método avaliando-se funções na parte esparsa e na parte estruturada e, também, seu efeito direto no número de multiplicações. Nota-se que o melhor custo computacional teórico foi obtido quando, independente do termo ser classificado como clássico ou generalizado, as avaliações de funções

são feitas apenas uma vez na parte esparsa, permutando-se as matrizes identidade para a parte estruturada sempre que possível, caso a memória disponível permita. Nesse caso, as multiplicações necessárias são apenas as que dizem respeito ao cálculo dos AUNFs e sua multiplicação pelo subespaço à direita de σ no vetor, deixando a parte estruturada sem precisar ser executada, pois as matrizes restantes são identidades.

Observa-se ainda que o grau de dependência, ou seja, o GDC e o GDG (definidos na Seção 5.1) constituem-se em uma métrica relevante para inspeção das características dos termos tensoriais obser-vando-se apenas os eventos sincronizantes e as funções existentes. Estes graus afetam diretamente no tempo de execução, pois estão relacionados a questões númericas de eficiência, devido ao número de multiplicações exigido e aos saltos que são dados quando as matrizes são do tipo identidade.

Ao se avaliar as funções apenas uma vez, reordena-se o termo tensorial e deixa-se todas as identidades para a parte estruturada é que foi encontrada uma nova forma de perceber o termo tensorial e considerá-lo esparso, ou seja, de execução extremamente rápida, sem precisar chamar as operações da parte estruturada e gastando uma memória extra razoável. Para tanto, deve-se considerar o papel que os eventos possuem nos descritores Markovianos. São os eventos relacionados aos termos tensoriais que ditam como o Algoritmo *Split* irá se comportar na sua execução. Eventos que dependem de muitos autômatos para fazer uma transição ditarão o tamanho dessa nova matriz esparsa que é implicitamente construída. A seguir é apresentada a ferramenta GTAexpress e detalhes da sua operação.

6. GTAEXPRESS – FERRAMENTA PARA SOLUÇÃO DE DESCRITORES

Este capítulo descreve a ferramenta GTAexpress [CZE09] cujo objetivo é trabalhar com descritores Markovianos, fornecendo os mecanismos para sua descrição e solução. Uma das vantagens de se utilizar tais representações é a possibilidade de definir sistemas com uma descrição alto-nível. Os modelos mais adaptados são compostos por partições, componentes, módulos, processos, autômatos ou qualquer outra nomenclatura estruturada de definição de sistemas.

A GTAexpress implementa o Algoritmo *Split* [CZE07], adicionando modificações (de acordo com a Seção 4.4.3) para tratar com descritores generalizados. A ferramenta permite configurar as estratégias estudadas nesta tese para dividir termos tensoriais de descritores Markovianos em dois blocos distintos, esparso e estruturado. Também possibilita a escolha de métodos iterativos de solução como, por exemplo, o Método da Potência, para efetuar a MVD. A principal ideia da ferramenta é trabalhar genericamente com tais estruturas tensoriais, permitindo que outros formalismos estruturados, além de SAN, usem suas funcionalidades. Para um uso efetivo da ferramenta, assume-se que os modelos construídos possuam características de composição e interação entre suas subpartes.

A implementação da GTAexpress foi realizada no ambiente de desenvolvimento PEPS [PLA88, BEN03a, BRE07], trabalhando com análise, permutação e divisão dos termos tensorias em duas partes distintas, denominadas parte esparsa e estruturada. O Algoritmo *Split* foi estendido para tratamento de taxas funcionais e suporte a permutações de matrizes como uma estratégia para melhorar o desempenho do método de MVD.

Este Capítulo está organizado da seguinte forma: inicialmente, na Seção 6.1 será explicada a motivação e os detalhes pertinentes da ferramenta, seguido de explicações da formação de descritores Markovianos na Seção 6.2. Na Seção 6.3 serão explicados como modelos advindos de diferentes formalismos estruturados podem ser definidos através de descritores Markovianos para, na Seção 6.4, exemplificar como usar a GTAexpress, alterando as configurações propostas. O capítulo encerra-se na Seção 6.5 com uma discussão sobre a ferramenta.

6.1 Abordagem tensorial de armazenamento

A ferramenta GTAexpress [CZE09] foi construída para solucionar descritores generalizados (com ATG) usando o Algoritmo *Split*, que trabalha com a divisão de cada termo tensorial, correspondendo a um evento na realidade descrita, em duas partes distintas: uma parte esparsa e uma parte estruturada. Esta abordagem híbrida reune as vantagens dos algoritmos *Esparso* e *Shuffle* (de acordo com a Seção 4.4), em uma única proposição. Ao separar os termos tensoriais, aproveita-se o fato que os eventos que compõem um modelo normalmente não envolvem, ou não interferem, em todas, ou na maioria, das entidades, módulos ou componentes. Ao afetar, ou sincronizar, apenas alguns módulos, propicia-se a criação de matrizes do tipo identidade no termo tensorial para representar que

o evento *não* ocorre nos outros componentes (maiores informações na Seção 5.3.1). Esse aspecto é explorado pelo Algoritmo *Split* que utiliza a propriedade da pseudo-comutatividade da ÀlgebraTensorial re-estruturando os termos tensoriais, conforme explicado na Seção 5.2.

A principal vantagem de se trabalhar com descritores tensoriais é não precisar alocar a matriz de transição que descreve a evolução do sistema através das taxas de mudanças entre estados. Dependendo do modelo sob análise, esta matriz onera substancialmente a memória necessária para representar as transições do sistema. Um formato tensorial gera esta matriz de forma implícita usando as propriedades da ÀlgebraTensorial. A proposição de descritores tensoriais para avaliação de desempenho existe desde as definições iniciais de Redes de Autômatos Estocásticos (SAN) [PLA91, FER98b]. Entretanto, outros formalismos estruturados também usaram tais primitivas para descrever seus modelos. Exemplos de tais formalismos, além de SAN, são as Redes de Petri Estocásticas Generalizadas Superpostas [DON93, KEM96] (e *frameworks* para decomposição de Redes de Petri Generalizadas [AJM84, CIA91, DON93] para Redes de Autômatos Estocásticos [BRE05b]), PEPA [HIL96, HIL07] e Redes de Filas de Espera [GEL87, GEL91, BOL98] também são conhecidas na literatura em Avaliação de Desempenho. As ferramentas que tratam destes formatos são igualmente conhecidas e exemplos são PEPS (para SAN), SMART (para Redes de Petri) ou PEPA Eclipse Plug-in [TRI07, TRI09] (para PEPA).

Uma desvantagem das traduções entre formalismos é devida à necessidade de precisar trabalhar com formatos tensoriais específicos para cada descrição. A ideia básica da GTAexpress é preencher esta lacuna ao auxiliar os modeladores a definir seus sistemas de forma que possam ser transcritos sob formas tensoriais e, posteriormente, ajudando a analisar e extrair índices quantitativos de desempenho ou sobre a operação do sistema. Cabe ressaltar que o principal destaque da GTAexpress ao ser comparado com as outras ferramentas é tratar com descritores clássicos e generalizados de forma otimizada. O uso de taxas funcionais aumenta as possibilidades de descrição de um formalismo estruturado ao mesmo tempo que simplifica o número de termos tensoriais necessários (ao reduzir o número de eventos, por exemplo), implicando diretamente no tempo envolvido para a solução [BRE05a].

Por exemplo, sabe-se que a ferramenta SMART só trabalha com Álgebra Tensorial Clássica e que a ferramenta PEPA *Eclipse Plug-in* não realiza a multiplicação vetor-descritor, existindo uma definição teórica sobre uma forma de se traduzir modelos de PEPA para descritores Markovianos [HIL07]. A principal diferença entre a ferramenta PEPS e a GTAexpress é devida ao fato que a primeira não implementa o Algoritmo *Split* [CZE07].

A implementação da ferramenta GTAexpress é baseada em dois eixos: 1) na definição e construção de descritores tensoriais válidos para a extração de índices de desempenho de sistemas através da proposição de um formato padrão simples de usar e 2) na execução dos algoritmos de MVD (para ATC e ATG) com o Algoritmo *Split* e re-estruturação (transparente aos usuários) dos termos tensoriais. O formato de entrada proposto pela GTAexpress está baseado na linguagem *eXtensible Markup Language* (XML), já usada em Redes de Petri, através da *Petri Net Markup Language* [BIL03] (PNML). Os modeladores dos sistemas devem se preocupar apenas em como compor um sistema

(usando módulos, autômatos, ou qualquer outro nome para decomposição) e informar seus comportamentos, de forma individual ou sincronizada entre os demais módulos ou componentes. Estas primitivas simples serão usadas para a construção do descritor Markoviano.

Como o objetivo é generalizar ao máximo a descrição que corresponde ao funcionamento de um sistema, serão utilizados ao longo do capítulo uma nomenclatura padrão como, por exemplo, rótulos para decomposição de sistemas. Serão utilizados os seguintes nomes: *módulos*, *partições*, *componentes*, *autômatos* ou *processos* para corresponder as formas existentes de descrever um sistema através de partes distintas e indivisíveis, capturando os comportamentos independentes e dependentes entre si. A seguir é mostrado como se construir descritores Markovianos para posterior execução na ferramenta GTAexpress.

6.2 Descritores Markovianos

Esta seção mostrará uma proposta para a composição de descritores Markovianos e seus principais elementos dentro da Ferramenta GTAexpress. Inicialmente serão descritas as partes constituintes de um descritor, tais como:

- Todo modelo estruturado é composto por um número N de módulos, componentes, processos, autômatos, ou outra nomenclatura a ser utilizada para decompor um sistema em partes[1], similar à utilização de métodos do tipo *divisão-e-conquista* [AHO74];
- Genericamente, cada módulo $n^{(i)}$ possui estados locais $s_j^{(i)}$ onde $j = 1..S$;
- Cada estado $s_j^{(i)}$ apresenta uma ou mais transições para outros estados, de acordo com uma taxa de ocorrência associada;
- Uma transição possui uma lista de eventos que ocorrem de forma independente (L eventos locais) ou sincronizada com outros módulos (E eventos sincronizantes);
- Ao utilizar o mesmo nome de evento para transições pertencentes a módulos distintos, esta será internamente classificada como sendo do tipo *sincronizante*, caso contrário, será do tipo *local* (parcial ou total, explicado a seguir);
- O espaço de estados de um modelo estruturado é composto pelo produto cartesiano de todos os estados locais de todos os módulos, formando o Produto do Espaço de Estados (PSS);
- As transições entre os estados dos módulos ocorrem de três formas distintas:
  - *Transições completamente locais:* tratam-se de transições que ocorrem em um componente e não são afetadas nem afetam outros módulos;

[1] A partir desta seção, será utilizado *módulo* para chamar esta divisão do sistema.

- *Transições parcialmente locais:* são transições que ocorrem em um módulo de acordo com o estado de outros módulos, entretanto afeta a mudança de estado de apenas um único componente;
- *Transições sincronizantes:* este tipo de transição afeta o estado local de dois ou mais componentes, potencialmente todos os módulos de um determinado modelo (dependendo do que o evento está representando no sistema).

As transições dentro de cada módulo são representadas por matrizes cuja dimensão é o número de estados e seus elementos internos são as taxas de ocorrência, afetadas ou não através do uso de diretivas funcionais. Por exemplo, pode-se usar uma taxa funcional, que observa o estado de outro módulo, para definir (configurando um teste que retorna um valor lógico diferente de falso, ou seja, uma taxa maior que zero) ou não (valor lógico igual a falso, ou seja, zero) uma transição entre estados.

Um descritor Markoviano é composto por somas de produtos tensoriais composto por N matrizes cada. O descritor é formado por matrizes com as taxas dos eventos locais (completa ou parcialmente) e por matrizes para representar os eventos sincronizantes, sendo necessárias operações de somas tensoriais e produtos tensoriais entre estas matrizes de transição, respectivamente. Como a operação da soma tensorial é vista como produtos tensoriais com matrizes identidade (ou seja, são gerados fatores normais, de acordo com a Seção 4.1.2), um descritor é composto por somas de produtos tensoriais, e o total destas somas equivale a quantidade de eventos sincronizantes e matrizes locais de autômatos com comportamentos independentes em cada modelo. A Tabela 6.1 mostra as divisões existentes em um descritor, divididas entre locais e sincronizantes, com parte positiva e a adequação da sua parte negativa (os ajustes da diagonal, conforme explicado na Seção 4.3). A tabela mostra produtos e somas tensoriais generalizadas, ou seja, pressupõe que taxas funcionais estão sendo utilizadas.

A tabela mostra N matrizes $Q_l^{(i)}$ que representam as transições que afetam apenas um dos N elementos, *i.e.*, a parte local. Cada uma destas matrizes é composta por um conjunto de $N-1$ matrizes do tipo identidade que representam que outros componentes não são afetados pela ocorrência das transições locais. A Tabela 6.1 mostra $2E$ conjuntos de N matrizes $Q_{e_k}^{(i)}$ que representam E transições com sincronizações entre elementos, *i.e.*, a parte sincronizante do descritor. Cabe ressaltar que cada transição sincronizante e_k necessita dois conjuntos de matrizes: o primeiro contendo sua ocorrência nos módulos ($Q_{e_k^+}^{(i)}$) e o segundo representando o ajuste da sua diagonal ($Q_{e_k^-}^{(i)}$), como mencionado anteriormente. Dependendo do evento a ser observado, dependendo de quantos outros módulos este afeta, tem-se matrizes do tipo identidade para representar que o módulo não altera o seu estado local com este dado evento. Esta consideração é crucial para o desempenho de algoritmos como o *Split*, que se aproveita deste fato para acelerar o processo de MVD, conforme Seção 4.4.3. A seguir, são mostradas as regras para criação e definição de descritores válidos usados pela GTAexpress.

Tabela 6.1: Representação geral de descritores Markovianos.

<table>
<tr><td rowspan="4">Σ</td><td colspan="2" rowspan="2">N</td><td>$Q_l^{(1)} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \cdots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N}$</td></tr>
<tr><td>$I_{n_1} \underset{g}{\otimes} Q_l^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} I_{n_N}$
$\vdots$
$I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_l^{(N-1)} \underset{g}{\otimes} I_{n_N}$
$I_{n_1} \underset{g}{\otimes} I_{n_2} \underset{g}{\otimes} \cdots \underset{g}{\otimes} I_{n_{N-1}} \underset{g}{\otimes} Q_l^{(N)}$</td></tr>
<tr><td rowspan="2">$2E$</td><td>e^+</td><td>$Q_{e_1^+}^{(1)} \underset{g}{\otimes} Q_{e_1^+}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{e_1^+}^{(N-1)} \underset{g}{\otimes} Q_{e_1^+}^{(N)}$
$\vdots$
$Q_{e_E^+}^{(1)} \underset{g}{\otimes} Q_{e_E^+}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{e_E^+}^{(N-1)} \underset{g}{\otimes} Q_{e_E^+}^{(N)}$</td></tr>
<tr><td>e^-</td><td>$Q_{e_1^-}^{(1)} \underset{g}{\otimes} Q_{e_1^-}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{e_1^-}^{(N-1)} \underset{g}{\otimes} Q_{e_1^-}^{(N)}$
$\vdots$
$Q_{e_E^-}^{(1)} \underset{g}{\otimes} Q_{e_E^-}^{(2)} \underset{g}{\otimes} \cdots \underset{g}{\otimes} Q_{e_E^-}^{(N-1)} \underset{g}{\otimes} Q_{e_E^-}^{(N)}$</td></tr>
</table>

6.3 Etapas para construção de descritores

Como exemplificado ao longo deste capítulo, o uso de descritores Markovianos é incentivado devido principalmente aos ganhos de memória ao se armazenar pequenas matrizes com operações algébricas tensoriais entre si ao invés de salvar um Gerador Infinitesimal completo. Esta vantagem fez com que outros formalismos definissem seus próprios formatos. Entretanto, observando as etapas em comum entre as definições usadas para PN, PEPA, SAN e Redes de Filas de Espera, nota-se que existe um conjunto específico de generalidades, descrito a seguir.

1. abstrair e observar a operação de uma realidade ou sistema, descobrir o que se deseja representar, inferir ou descobrir sobre o seu funcionamento;

2. identificar como decompor o sistema em subpartes menores e mais tratáveis, em componentes ou módulos com comportamentos individuais e interações com outros módulos;

3. identificar especificamente suas operações internas e as interações observadas entre as partes que o compõe; determinar as interações que acontecem de forma independente e dependente;

4. descobrir e definir os estados internos, dentro de cada subparte observada;

5. identificar as transições entre os estados e classifica-las quanto à sua dependência (dependem que outras partes estejam em algum estado ou não); para parametrização do modelo, determinar a frequência de troca de estado do processo Markoviano, para definição de uma CTMC;

6. usar um formato descritivo para representar estas estruturas em um modelo válido;

7. usar uma ferramenta que transforme este formato descritivo baseado em módulos, elementos, processos, autômatos, etc, em um descritor válido que represente em última análise as matrizes de transições entre estados do sistema e suas taxas de ocorrência;

8. identificar os índices de desempenho que se deseja estudar e extrair do sistema; descobrir as análises quantitativas que se deseja inferir sobre o funcionamento ou comportamento do sistema em regime estacionário ou transiente;

9. usar a ferramenta GTAexpress para utilizar mecanismos especializados de MVD para resolver o sistema e determinar as probabilidades de permanência nos estados do mesmo;

10. de acordo com os resultados, retornar à fase de concepção e refinar o modelo, retirando estados, transições ou adicionando outros mecanismos de interação para capturar seu funcionamento de acordo com novas características.

Essas etapas podem ser utilizadas para definir descritores Markovianos em geral e também podem ser usadas, sem perda de generalidade, para descrição de diversas outras realidades ou traduções de outros formalismos estruturados que forneçam representação tensorial. Estas regras nada mais representam do que a construção de descritores generalizados para inferência de índices de desempenho, confiabilidade ou outras medidas quantitativas sobre o sistema modelado. Todos os modelos que forem compostos observando-se as etapas definidas acima serão passíveis de serem executados na ferramenta GTAexpress, ou seja, a ferramenta é especializada para tratar com descritores Markovianos. A próxima seção descreve a arquitetura da ferramenta e os formatos de entrada e saída propostos.

6.4 Arquitetura e detalhes da ferramenta

A ferramenta GTAexpress foi concebida no contexto desta tese de acordo com as seguintes premissas:

- Um modelo estruturado pode ser decomposto de forma genérica e transposto para um formato que representa um descritor Markoviano;

- Métodos específicos de MVD são usados para resolver computacionalmente um modelo, calculando os índices de desempenho a serem utilizados na fase de análise.

Diversos formalismos estruturados já propuseram representações tensoriais para seus modelos e possuem regras similares para o correto tratamento dos seus descritores Markovianos. Com o propósito de estudar as maneiras de se converter um formalismo para sua representação tensorial, tradutores especializados são construídos. Desta forma, modelagens descritas com primitivas de diferentes formalismos são traduzidos para para um formato que descreva um descritor Markoviano [DON93, FER98b, HIL01, BRE05b]. A falta de regras precisas para efetuar este mapeamento de forma automática e possibilitar a modelagem analítica de sistemas reais com álgebra tensorial é uma lacuna das pesquisas atuais e objeto de inúmeras pesquisas em avaliação de desempenho.

Antes de descrever a proposição para os formatos de entrada e os passos de execução da ferramenta, são necessárias explicações de detalhes de implementação no ambiente PEPS. Para o Algoritmo *Split* (com versão estendida para ATG) foi criado um novo método na classe que implementa a MVD clássica, ou seja, o Algoritmo *Shuffle*. Para guardar os AUNFs (definição na Seção 4.4.3) necessários para cada modelo é criada, antes da execução e compilação, uma lista de eventos que contém a lista de todos os AUNFs. O tempo gasto e a memória extra usada para efetuar essa tarefa é salvo internamente para efeitos de estatísticas sobre a execução do método como um todo. Foram implementados também novos métodos para permutar a ordem lexicogrática de cada termo tensorial, movendo os índices de matrizes que correspondem a identidades para a esquerda ou para a direita, de acordo com a estratégia utilizada, entre as demais estratégias. Por fim, foram alteradas as opções da interface para representar as opções de chamada do novo método de MVD e na opção que mostra as estatísticas da execução da ferramenta. A versão disponível foi implementada para ser executada na plataforma Linux, entretanto, pode ser compilada em um ambiente Windows sem ser necessário alterações no código-fonte. A seguir serão explicados o funcionamento do formato de entrada proposto, compilação e execução de modelos e formato de saída da GTAexpress.

6.4.1 Formato de entrada proposto

Como mencionado anteriormente, foi escolhido um formato padrão, já utilizado em outros formalismos para descrição dos seus elementos, baseado na linguagem XML. O arquivo definido em XML é composto pelas seguintes partes (para aumentar a adoção da descrição por outros pesquisadores, o formato encontra-se atualmente definido na língua inglesa):

- *properties*: são as propriedades do sistema e outras informações que o resolvedor irá utilizar para multiplicar o vetor pelo descritor ou testar a convergência. Exemplos desta parte do arquivo são: descrição do nome do modelo e sua operação, informações sobre o autor e quando foi concebido (para propósitos de histórico), tolerância do erro para teste de convergência, método iterativo a ser usado (Potência, GMRES, Arnoldi, etc), tipo de análise (transiente ou estacionária), escala de tempo (contínua ou discreta) ou modo de depuração de mensagens de texto (*verbose*);

- *modules*: define a forma que o sistema foi estruturado, ou seja, suas partes ou subsistemas. Nesta representação geral, chama-se de módulos mas correspondem a autômatos, processos,

partições ou outra nomenclatura composicional. No arquivo XML é possível definir uma lista de módulos, contendo diversos blocos do tipo estado (*states*);

- *states*: são os estados que o módulo define e possuem dois blocos internos: 1. bloco gráficos (*graphics*) podendo conter informações visuais (onde desenhar cada estado na tela) e 2. uma lista de transições (*transitions*) para outros estados de acordo com uma lista de taxas (*rates*);
- *graphics*: informações visuais contendo o tipo da fonte, a cor do estado e o seu posicionamento na tela, usado para representação visual do sistema, quando existir;
- *transitions*: compreende o conjunto de transições de um módulo;
- *rates*: correspondem as taxas observadas para cada transição em questão;
- *gcc-code*: esta parte corresponde a funções descritas na linguagem de programação C/C++ que serão adicionadas ao executável da ferramenta GTAexpress em tempo de execução. Este bloco é um mapeamento para uma taxa constante ou funcional definida no bloco das taxas (*rates*). A vantagem de se descrever funções desta maneira é o de permitir que sejam calculadas taxas de forma complexa, tendo a linguagem C/C++ como base;
- *results*: seção que realiza a extração dos índices de desempenho. Estas diretivas correspondem à *Integração de Funções* no PEPS e operam sobre o espaço de estados para retornar as probabilidades de permanência em cada estado.

A seguir, mostra-se um exemplo do arquivo XML para um sistema de teste com a definição de duas impressoras (cada módulo corresponde a uma impressora). Observa-se que para um modelo descrito em XML estar sintaticamente correto, existem bibliotecas que validam o arquivo a partir de um esquema (*XML Schema Definition*, ou XSD) predefinido. Uma vantagem de se usar XML para descrever um sistema é a possibilidade de validar arquivos sintaticamente. Entretando, a semântica dos modelos continua sendo de responsabilidade dos usuários da ferramenta e do esforço gasto para mapear o sistema sob estudo.

```
<?xml version="1.0" encoding="ISO_8859-1"?>
<model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="model.xsd">
  <properties>
    <model-name>two-printers</model-name>
    <description>This is a test model to describe two printers.</description>
    <author>John Smith</author>
    <creation-timestamp>2009-08-18 12:00</creation-timestamp>
    <time-scale>continuous</time-scale>
    <analysis-type>transient</analysis-type>
    <error-tolerance>0.0000000001</error-tolerance>
```

```
    <max-iterations>1000</max-iterations>
    <iterative-method>power</iterative-method>
    <verbose>true</verbose>
  </properties>
  <gcc-code>
#define MAX 10
int rate1(int *state) {
   return (state[printer0] * MAX);
}
 </gcc-code>
  <modules>
    <module name="printer0">
      <states>
        <state name="ON">
          <transitions>
            <transition>
              <to>OFF</to>
              <rates>
                <rate>rate1</rate>
                <rate>rate2</rate>
              </rates>
            </transition>
          </transitions>
        </state>
        <state name="OFF">
          <transitions>
            <transition>
              <to>IDLE</to>
              <rates>
                <rate>rate3</rate>
                <rate>rate4</rate>
              </rates>
            </transition>
          </transitions>
        </state>
        <state name="IDLE">
          <transitions>
            <transition>
              <to>ON</to>
              <rates>
                <rate>rate5</rate>
```

```
            </rates>
          </transition>
          <transition>
            <to>OFF</to>
            <rates>
              <rate>rate6</rate>
            </rates>
          </transition>
        </transitions>
      </state>
    </states>
  </module>
  <module name="printer1">
    <states>
      <state name="ON">
        <transitions>
          <transition>
            <to>OFF</to>
            <rates>
              <rate>rate4</rate>
            </rates>
          </transition>
        </transitions>
      </state>
      <state name="OFF">
        <transitions>
          <transition>
            <to>ON</to>
            <rates>
              <rate>rate4</rate>
            </rates>
          </transition>
        </transitions>
      </state>
    </states>
  </module>
 </modules>
 <results></results>
</model>
```

Com as informações dos módulos, transições, eventos e taxas contidas no arquivo é possível criar um procedimento que verifica se as transições de cada módulo são disparadas corretamente

(não existem transições para lugares indefinidos), entre outras. A estrutura em árvore do arquivo XML facilita a definição não-ambígua do sistema e fornece uma visão modular do sistema para os modeladores. A seguir são apresentadas as configurações de entrada, execução e saída da ferramenta.

6.4.2 Configurações de entrada, execução e saída

A ferramenta GTAexpress recebe um descritor (nesta etapa, aproveita-se o fato de que SAN trabalha com descritores Markovianos) e está implementada no ambiente PEPS. A GTAexpress permite solução estacionária para escala de tempo contínua (CTMC), com o método iterativo de solução conhecido por Método da Potência [STE94]. Para tanto, manteve-se as características originais já conhecidas e testadas do ambiente PEPS, acrescentando o Algoritmo *Split* para solução de descritores generalizados e possibilidades de escolhas das diferentes estratégias.

Para usar o Algoritmo *Split* [CZE07] é necessário informar previamente ao compilador da ferramenta, que tratará de criar as estruturas de dados necessárias para este fim (criação da lista de valores escalares, ou AUNFs, para cada termo tensorial do descritor do modelo), as permutações necessárias e os valores de corte personalizados (de acordo com a estratégia) para separar a parte esparsa da estruturada. Caso o modelo seja definido com primitivas de taxas funcionais, o compilador trata de deslocar os parâmetros do termo tensorial com taxa funcional para manter a integridade da função ao retornar a sua avaliação no método (explicado na Seção 5.2.2).

Demonstra-se a seguir cinco passos (de I a V) necessários para a configuração, compilação e execução da ferramenta GTAexpress:

```
I - Rodar a ferramenta e a partir da tela principal, escolher as opções:
[TELA INICIAL]
... [escolher o menu de preferências]
4) Preferences
... [escolher o método para multiplicação vetor-descritor]
8) Vector-Descriptor product method :
... [escolher usar o Algoritmo Split]
4) use a Split approach
II - Escolher uma das seguintes 'estratégias' para divisão dos termos tensoriais:
Choose a strategy to 'cut' tensor terms:
    1 move identities to shuffle [cnst]                                (strategy A)
    2 functional evaluations in shuffle [fnct]                         (strategy B)
    3 functional evaluations in split [fnct]                           (strategy C)
    4 functional evaluations in split, identities to shuffle [fnct] (strategy D)
    5 only functional definition and identities in shuffle [fnct]   (strategy E)
III - Escolher as opções usuais de compilação do modelo
IV - Escolher a opção de Solução do modelo em questão e executar
[TELA FINAL]
V - Inspecionar o arquivo de saída definido em <MODELO>.tim
```

As etapas III e IV correspondem às formas usuais de compilação e solução de SAN, a única diferença encontra-se nas etapas I e II onde escolhe-se rodar a proposta *Split* de acordo com uma determinada estratégia entre cinco possibilidades (chamadas de A a E). Uma vez que o modelo foi executado, os índices de desempenho encontram-se no arquivo com terminação .tim, no mesmo diretório com o executável da ferramenta GTAexpress. Para exemplificar esse arquivo de saída, mostra-se a seguir o Modelo *Wireless ad hoc* com 12 nós (*ad12c*). A letra *'i'* representa matrizes do tipo identidade e as esparsidades de cada matriz encontram-se à esquerda do símbolo '|':

```
================================================================
File ad12c.tim
================================================================
ad12c   -- A model with 12 modules and 12 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           236196 states
Reachable state space:         1 states
Modules sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 2 ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 118 Kbytes
================================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
The strategy to 'cut' the tensor terms:
   identities to shuffle [constant] (strategy A)
Extended (E) Probability Vectors
================================================================
Number of AUNFs: 12 (0 Kb)
Chosen cuts: [ 4 5 6 6 6 6 6 6 6 5 4 2 ]
SYN Term: evt. [ permutation orders and cut (|) ] [ idnt list ]

   0. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]    [ 1 1 1 1|i i i i i i i i ]  [cnst_term]
   1. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]    [ 1 1 1 1 1|i i i i i i i ]  [cnst_term]
   2. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   3. [  1  2  3  4  5  6| 0  7  8  9 10 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   4. [  2  3  4  5  6  7| 0  1  8  9 10 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   5. [  3  4  5  6  7  8| 0  1  2  9 10 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   6. [  4  5  6  7  8  9| 0  1  2  3 10 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   7. [  5  6  7  8  9 10| 0  1  2  3  4 11 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   8. [  6  7  8  9 10 11| 0  1  2  3  4  5 ]    [ 1 1 1 1 1 1|i i i i i i ]  [cnst_term]
   9. [  7  8  9 10 11| 0  1  2  3  4  5  6 ]    [ 1 1 1 1 1|i i i i i i i ]  [cnst_term]
  10. [  8  9 10 11| 0  1  2  3  4  5  6  7 ]    [ 1 1 1 1|i i i i i i i i ]  [cnst_term]
  11. [ 10 11| 0  1  2  3  4  5  6  7  8  9 ]    [ 1 1|i i i i i i i i i i ]  [cnst_term]

Time needed to generate the AUNFs list: 0 seconds
```

```
Number of iterations performed: 47887
CPU average time per iteration: 0.0557882097437718 seconds
Total CPU time:                 2671.53 seconds
----------------------
Vector characteristics
----------------------
Name:                       ad12c.vct
Sum of all elements:        1
Largest element:            0.05533790942991294 (in position 4752)
Smallest element:           4.344646345974808e-11 (in position 158978)
Residue after convergence: 9.995410057229764e-11
  (the vector ad12c.vct is the stationary solution of the model ad12c)
---------------------
Results
---------------------
t_MN_1_I: 0.831117360267249
t_MN_1_T: 0.2952493701621532
t_MN_2_I: 0.521105375648411
...
```

A parte de resultados mostra as permutações que foram realizadas para cada termo tensorial (*SYN Term:*), informações sobre o corte e sobre o número de AUNFs (memória extra gasta) para a execução do método, tempo para geração da lista de AUNFs, número de iterações executadas até a convergência. São igualmente mostradas as informações sobre a execução, tais como número de eventos, tipos de eventos, tempo médio por iteração e método iterativo utilizado. A seção de *Results* do arquivo de saída mostra as probabilidades de permanência nos estados, escolhidos pelo modelador.

6.5 Discussão

Os primórdios da definição de SAN trataram de descrever um sistema através de módulos, os quais na época foram chamados de autômatos. As interações entre os autômatos foram descritas através de primitivas de sincronização a partir da atribuição de taxas constantes ou funcionais às transições entre os estados. A proposta da ferramenta GTAexpress é a de definir uma nova abstração para a composição de sistemas para cálculo computacional de índices de desempenho. A ferramenta baseia-se fundamentalmente na execução do Algoritmo *Split*, que re-estrutura os termos tensoriais para tirar proveito da otimização das matrizes identidade (dependendo do caso) e executar cada passo do método iterativo de forma mais rápida.

Quanto à modelagem, a GTAexpress facilita a descrição de realidades, pois deixa aos modeladores a tarefa de abstrair o problema pensando apenas na estruturação do sistema e como cada parte se

comunica. Estas primitivas possibilitam a construção de descritores Markovianos para extração e análise dos índices de desempenho. Internamente, dadas as descrições realizadas pelos usuários, a ferramenta detecta o nível de interação entre os componentes e auxilia na escolha de uma estratégia para execução do Algoritmo *Split*. Este, por sua vez, divide cada termo tensorial e minimiza a memória extra gasta e o tempo por iteração necessário (conforme Capítulo 5) de acordo com a estratégia.

Cabe ressaltar que os diferentes formalismos estruturados possuem conjuntos próprios de definição e solução. O fato de não gerar a matriz de transição correspondente à MC subjacente atrai pesquisas para o mapeamento de formatos tensoriais para os diversos formalismos. Entretanto, somente SAN trabalha com primitivas avançadas de modelagem, tais como taxas funcionais. Com a ferramenta GTAexpress, espera-se que outros formalismos possam aproveitar as ideias aqui descritas e usar os métodos disponíveis para investigação de propriedades quantitativas de sistemas. O objetivo da proposta de uma descrição genérica ou formato de entrada para a ferramenta GTAexpress é o de prover mecanismos que permitam que modeladores abstraiam seus sistemas estruturadamente e que usem o *software* para efetuar a MVD.

O presente capítulo propôs um formato padrão descrito na linguagem XML, objetivando traduções e mapeamentos para outros formalismos estruturados. O próximo capítulo apresenta os resultados numéricos obtidos através da utilização da ferramenta GTAexpress.

7. RESULTADOS NUMÉRICOS

Este capítulo aborda os resultados numéricos obtidos para um conjunto de experimentos e modelos. Na Seção 7.1 são definidos os experimentos e as considerações iniciais. Cada modelo é apresentado seguido dos seus resultados, comparando-se os experimentos em termos dos ganhos obtidos na Seção 7.2. A Seção 7.3 mostra um estudo de caso para o modelo das *Redes Wireless ad hoc* para, na Seção 7.4 discutir os ganhos obtidos para os algoritmos de MVD executados. A Seção 7.5 mostra o algoritmo proposto para determinação de ponto de corte para descritores Markovianos. O capítulo é finalizado com análises e questionamentos sobre os experimentos na Seção 7.6.

7.1 Experimentos

O objetivo da definição dos experimentos deste capítulo é testar os efeitos das características dos termos tensoriais verificando o desempenho da solução numérica do Algoritmo *Split*. Deseja-se descobrir os experimentos com os melhores tempos para uma iteração com a execução dos métodos de MVD, comparando-os para a mesma classe de modelos com descritores clássicos (ou generalizados) e com (ou sem) permutações das ordens originais (lexicográficas) das matrizes.

Na Tabela 7.1 são apresentados *nove* experimentos mapeados para execução dos modelos. A base de comparação é o método usual de solução baseado no Algoritmo *Shuffle*, de acordo com os Experimentos $2, 4$ e 9. As experiências foram executadas variando-se as possibilidades de avaliação de funções e re-estruturação das matrizes identidade dentro dos termos tensoriais.

Para avaliar o custo da permutação das matrizes dos termos tensoriais foram criados os Experimentos 2 e 4 para o Algoritmo *Shuffle* e Experimentos 1 e 3 para o Algoritmo *Split*. Para o caso do Experimento 3, o ponto de corte escolhido é na última matriz constante do termo tensorial. O Experimento 8 calcula o custo das avaliações de funções, pois deixa apenas a matriz que contém o elemento funcional e as matrizes identidade na parte estruturada e somente as matrizes constantes na parte esparsa do termo. Os experimentos foram mapeados a partir da análise teórica conduzida na Seção 5.3. Os estudos efetuados indicaram a necessidade de se testar os diferentes métodos de MVD permutando-se o termo tensorial e dividindo-o de acordo com as suas dependências funcionais e outras características, tais como os tipos das matrizes e as suas esparsidades.

Os exemplos aqui descritos possuem eventos sincronizantes com taxas funcionais para o caso generalizado onde também foram devidamente convertidos para o seu modelo correspondente clássico (ATC). O objetivo desta conversão é o de verificar em quais casos é satisfatório traduzir descritores baseados em ATG para ATC, apesar do aumento do número de eventos no termo tensorial [BRE05a].

A execução dos experimentos foi realizada em uma máquina *Intel Pentium*® $3{,}2$ GHz com 4 Gb de memória RAM e 2 Mb de *cache*. A ferramenta PEPS [BRE07] foi utilizada como base onde implementou-se o Algoritmo *Split* para descritores clássicos e generalizados na ferramenta *GTAexpress* (Capítulo 6). Além disso, o armazenamento e geração dos AUNFs e outras estruturas de

Tabela 7.1: Detalhamento dos experimentos escolhidos para execução.

Experimento	Método	Tipo de Descritor	Uso de Permutação	Descrição
1	*Split*	clássico	$\surd$	identidades para parte estruturada
2	*Shuffle*	clássico	$\surd$	método usual para descritores clássicos
3	*Split*	clássico	$\times$	sem permutações, cortar na última matriz constante
4	*Shuffle*	clássico	$\times$	sem permutações, determinação do custo para efetuar permutação
5	*Split*	generalizado	$\surd$	avaliação de funções na parte estruturada
6	*Split*	generalizado	$\surd$	avaliação de funções na parte esparsa
7	*Split*	generalizado	$\surd$	avaliação de funções na parte esparsa, somente identidades para parte estruturada
8	*Split*	generalizado	$\surd$	avaliação de funções na parte estruturada, somente identidades para parte estruturada
9	*Shuffle*	generalizado	$\surd$	método usual para descritores generalizados

dados auxiliares foram codificadas e usou-se a GCC[1] (g++ versão $4.0.4$) com opções de otimização ($-O3$) e *linking* dinâmico para as funções dos descritores generalizados.

Estão sendo comparados o Algoritmo *Split* e o Algoritmo *Shuffle* com permutações para os Experimentos $1, 2, 5, 6, 7, 8$ e 9 e sem reordenamentos para os casos 3 e 4. Foram computados os tempos de execução com intervalos de confiança de 95% para 50 execuções sequenciais com 25 iterações cada (para o valor do tempo coletado foi calculado o tempo médio de execução do método considerando-se este número de iterações). Serão apresentadas tabelas contendo os resultados obtidos para os diversos modelos. Estas tabelas de resultados mostram o nome do modelo em questão (coluna *Modelo*), o seu PSS (espaço de estados produto) e uma divisão entre as informações relativas aos AUNFs, informações relativas ao tempo de execução (em *Tempos*) e ganhos comparando-se sempre dois experimentos (no caso, o ganho será calculado da seguinte maneira, a última coluna dividida pela anterior).

Na parte relativa aos AUNFs, tem-se a o total de escalares e índices para o experimento em questão (sempre será executado um experimento para o Algoritmo *Split* e outro experimento para o Algoritmo *Shuffle*), a sua memória extra (coluna *Mem.* – apenas para o Algoritmo *Split*), descrita em *Kilobytes* (Kb) e o tempo necessário para computar a tabela com os AUNFs, realizado apenas uma vez em todo o método de solução, na coluna *Cálculo*. O tempo dos experimentos são mostrados nas colunas correspondentes, medidos em *segundos* (s), mostrados na tabela com o seu intervalo de confiança de 95% correspondendo à média de *uma* iteração do *Método da Potência* fornecido

[1] *GNU Compiler Collection*.

pela ferramenta no arquivo de saída de dados ($.tim$). Como o tamanho do descritor é constante para todos os algoritmos de MVD, este não está sendo mostrado. O Apêndice A mostra as saídas de dados de alguns modelos utilizados bem como o tamanho do descritor.

Cabe ressaltar que os modelos variam o número de autômatos que os compõem, de forma incremental. Em casos específicos, como os modelos de *Redes Wireless ad hoc* e *Mestre-Escravo*, a definição original do descritor generalizado foi explicitamente mapeada para o seu correspondente clássico em termos de solução numérica [BRE05a].

7.2 Modelos

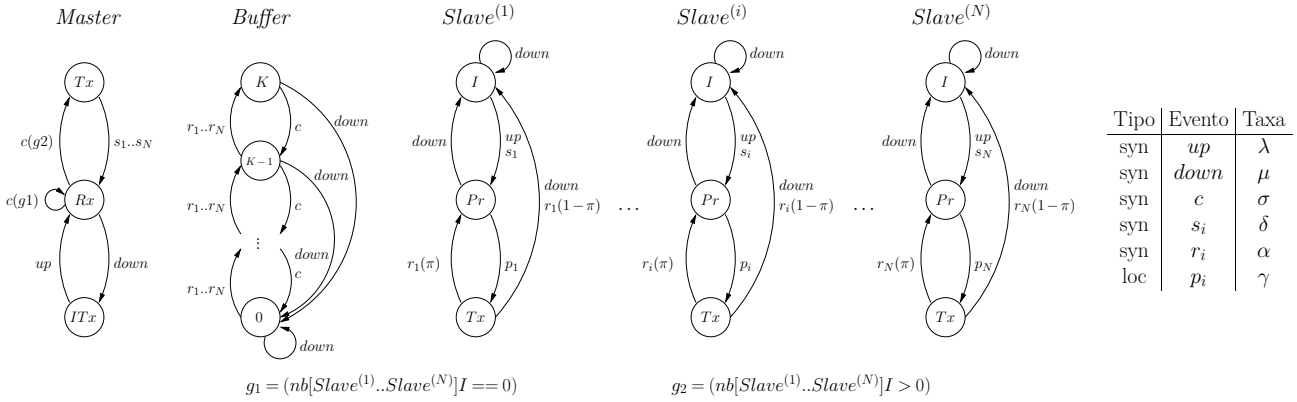
A existência de *benchmarks* para testes ainda é insuficiente em análise numérica de sistemas de avaliação de desempenho [BUC06]. No contexto deste trabalho, todos os testes foram executados para uma lista de modelos para SAN (alguns tradicionais, como *Compartilhamento de Recursos* (*Resource Sharing*) e outros provenientes de modelagens de uma arquitetura *Mestre-Escravo* (*Master-Slave*) e *Redes Wireless ad hoc*. As tabelas em cada subseção comparam as diferentes estratégias de divisão dos termos tensoriais, de acordo com os experimentos. Os demais modelos que compreendem essa seção definem o problema do *Jantar dos Filósofos*, *First Available Server*, *Alternate Service Patterns*, *Non-Uniform Memory Access*, *Open Queueing Network* (Rede de Filas de Espera Abertas) e duas realidades convertidas do formalismo PEPA: *Workcell* e *WebServer* (maiores informações na Seção 2.5). A seguir, uma breve explicação de cada modelo seguido dos resultados obtidos para todos os experimentos acima definidos.

7.2.1 Modelo *Mestre-Escravo*

O modelo SAN definido na Figura 7.1 refere-se à implementação paralela do Algoritmo *Propagation* com comunicação assíncrona [BAL05] através do paradigma Mestre-Escravo. Este modelo é responsável por indicar aos desenvolvedores de aplicações paralelas os gargalos e problemas de configurações existentes antes do início da fase de implementação. Este modelo contem três estruturas particulares: um autômato denominado *Mestre*, um autômato que representa um *Buffer* (uma região temporária de armazenamento) e S autômatos *Escravos*, aqui denominados por $Slave^{(i)}$, onde $i = 1 \ldots S$.

O número total de autômatos para este modelo é dado por $(S+2)$ (onde S é o número de *Escravos* no modelo), o qual possui S eventos locais e $(3S-3)$ eventos sincronizantes para o caso de descritores clássicos e o mesmo número de autômatos mas $(2S+3)$ eventos sincronizantes para o caso dos descritores generalizados. Isso faz com que descritores baseados em ATC possuam $(7S-8)$ termos tensoriais e descritores ATG $(4S+6)$ termos.

A Tabela 7.2 mostra os resultados para os Experimentos $1, 2, 3$ e 4. Cabe ressaltar que as informações relativas ao cabeçalho das tabelas estão também explicadas na Seção 7.1. Observa-se que dentre estes experimentos a melhor execução foi obtida na estratégia que coloca identidades na parte estruturada (Experimento 1). O uso de memória constatado foi pequeno (para 10 escra-



Tipo	Evento	Taxa
syn	*up*	λ
syn	*down*	μ
syn	c	σ
syn	s_i	δ
syn	r_i	α
loc	p_i	γ

Figura 7.1: Modelo SAN generalizado para *Mestre-Escravo*.

vos, aproximadamente 76 Mb), tendo-se em vista o ganho obtido, da ordem de $quatro$ vezes em comparação com o Experimento 2 que executa o método usual. Os Experimentos 3 e 4 mostram os tempos de estratégias quando não se permutam as matrizes dos termos, onde infere-se que o Algoritmo *Shuffle* executa mais rápido inclusive que sua versão com permutação, mostrando que o uso de reordenamentos está relacionado ao modelo e as suas características internas.

Tabela 7.2: Resultados para o modelo *Mestre-Escravo* em ATC e Experimentos {1,2 e 3,4}.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
slaves05c	29.889	11.849	370	0,0300 ± 0,0001	**0,0094 ± 0,0001**	0,0335 ± 0,0001	3,56
slaves06c	89.667	33.176	1.036	0,0918 ± 0,0011	**0,0331 ± 0,0001**	0,1206 ± 0,0001	3,64
slaves07c	269.001	95.635	2.988	0,3388 ± 0,0013	**0,1176 ± 0,0001**	0,4311 ± 0,0005	3,66
slaves08c	807.003	280.210	8.756	0,9426 ± 0,0029	**0,3915 ± 0,0004**	1,4742 ± 0,0012	3,76
slaves10c	7.263.027	2.463.180	76.974	10,2764 ± 0,0200	**4,1841 ± 0,0033**	16,1104 ± 0,0069	3,85
					Experimento 3	Experimento 4	
slaves05c	29.889	50.125	1.566	0,1634 ± 0,0026	0,0179 ± 0,0001	0,0199 ± 0,0001	1,11
slaves06c	89.667	149.334	4.666	0,5014 ± 0,0063	0,0604 ± 0,0001	0,0756 ± 0,0001	1,25
slaves07c	269.001	445.681	13.927	1,5754 ± 0,0156	0,1977 ± 0,0003	0,2970 ± 0,0004	1,50
slaves08c	807.003	1.332.162	41.630	4,7496 ± 0,0359	0,6333 ± 0,0009	1,0292 ± 0,0007	1,62
slaves10c	7.263.027	11.939.214	373.100	45,1638 ± 0,2744	6,6515 ± 0,0062	11,3418 ± 0,0385	1,70

A Tabela 7.3 mostra os resultados para os Experimentos $5, 6, 7, 8$ e 9. Esta classe de modelos mostra a escalabilidade dos algoritmos, pois a cada incremento de *dois* escravos, tem-se um aumento aproximadamente 10 vezes em relação ao tempo necessário e à memória adicional gasta usando o Algoritmo *Split*.

O melhor tempo foi verificado para o Experimento 6, que fornece uma boa relação custo/benefício em comparação com o Experimento 7, que gasta um pouco mais de memória e tempo. Para o caso de 10 escravos, onde o PSS é elevado, verifica-se que com 79 Mb de memória tem-se um ganho de aproximadamente *três* vezes em relação à base de comparação do Experimento 9. Constata-se que esta memória gasta é negligenciável pois a plataforma de execução tem 4 Gb de memória RAM.

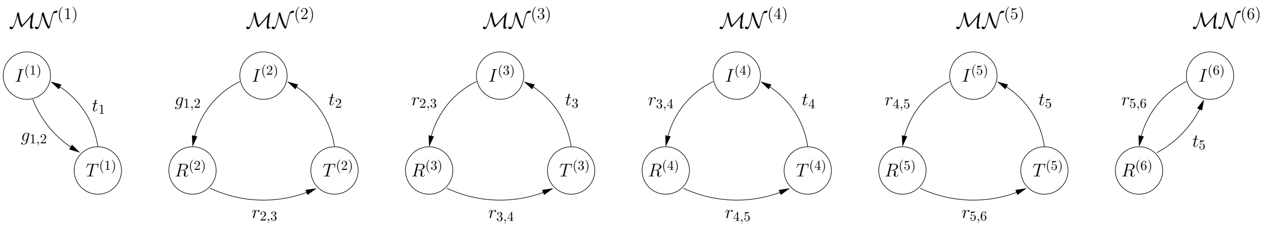
Tabela 7.3: Caso *Mestre-Escravo* em ATG e Experimentos {5,6,7,8,9}.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	Experimento 5	Experimento 9	
slaves05f	29.889	10.409	325	0,0200 ± 0,0001	0,0138 ± 0,0001	0,0274 ± 0,0001	1,98
slaves06f	89.667	30.416	950	0,0802 ± 0,0004	0,0480 ± 0,0001	0,0985 ± 0,0004	2,05
slaves07f	269.001	90.275	2.821	0,2872 ± 0,0015	0,1670 ± 0,0002	0,3476 ± 0,0003	2,08
slaves08f	807.003	269.690	8.427	0,9188 ± 0,0019	0,5539 ± 0,0004	1,2023 ± 0,0014	2,17
slaves10f	7.263.027	2.421.860	75.683	9,1362 ± 0,0168	5,9836 ± 0,0052	13,3783 ± 0,0049	2,23
					Experimento 6	Experimento 9	
slaves05f	29.889	10.855	339	0,0218 ± 0,0011	**0,0096 ± 0,0001**	0,0274 ± 0,0001	2,85
slaves06f	89.667	31.834	994	0,0842 ± 0,0014	**0,0346 ± 0,0001**	0,0985 ± 0,0004	2,85
slaves07f	269.001	94.609	2.956	0,2974 ± 0,0018	**0,1213 ± 0,0001**	0,3476 ± 0,0003	2,86
slaves08f	807.003	282.772	8.836	0,9608 ± 0,0034	**0,3985 ± 0,0003**	1,2023 ± 0,0014	3,02
slaves10f	7.263.027	2.539.918	79.372	9,4901 ± 0,0099	**4,2506 ± 0,0028**	13,3783 ± 0,0049	3,15
					Experimento 7	Experimento 9	
slaves05f	29.889	29.809	931	0,0676 ± 0,0012	0,0134 ± 0,0001	0,0274 ± 0,0001	2,05
slaves06f	89.667	88.696	2.771	0,2134 ± 0,0014	0,0474 ± 0,0001	0,0985 ± 0,0004	2,08
slaves07f	269.001	265.195	8.287	0,7150 ± 0,0026	0,1621 ± 0,0002	0,3476 ± 0,0003	2,14
slaves08f	807.003	794.530	24.829	2,4072 ± 0,0053	0,5215 ± 0,0004	1,2023 ± 0,0014	2,30
slaves10f	7.263.027	7.145.740	223.304	23,2650 ± 0,0356	5,4150 ± 0,0052	13,3783 ± 0,0049	2,47
					Experimento 8	Experimento 9	
slaves05f	29.889	10.409	325	0,0202 ± 0,0004	0,0138 ± 0,0001	0,0274 ± 0,0001	1,98
slaves06f	89.667	30.416	950	0,0808 ± 0,0008	0,0480 ± 0,0001	0,0985 ± 0,0004	2,05
slaves07f	269.001	90.275	2.821	0,2850 ± 0,0018	0,1664 ± 0,0001	0,3476 ± 0,0003	2,09
slaves08f	807.003	269.690	8.427	0,9184 ± 0,0035	0,5476 ± 0,0009	1,2023 ± 0,0014	2,19
slaves10f	7.263.027	2.421.860	75.683	9,1464 ± 0,0598	5,9853 ± 0,0039	13,3783 ± 0,0049	2,23

Pode-se comparar o mapeamento em tempo de execução do descritor baseado em ATG para descritor ATC, uma vez que as funções estão sendo avaliadas na parte esparsa para o experimento que executou mais rapidamente (Experimento 6). Para esse caso, a tradução de ATG para ATC e utilização do Algoritmo *Split* não implica necessariamente em um ganho substancial, pois, para o caso de 10 escravos, o tempo para o modelo clássico é de $4{,}18$ segundos e para o modelo generalizado é de $4{,}25$ segundos. Comparando-se o Experimento 1 e 9 percebe-se que essa conversão já é mais interessante, pois tem-se $16{,}11$ segundos para o caso clássico e $13{,}38$ segundos para o caso generalizado.

7.2.2 Modelo Redes *Wireless* ad hoc

O modelo SAN descrito na Figura 7.2 representa uma cadeia de nós móveis em uma Rede *Wireless* (de forma *ad hoc*) executada sobre o padrão 802.11 e definida com taxas funcionais. Este modelo [DOT05] é uma adaptação do experimento de uma rede *ad hoc* presente em [LI01]. A cadeia tem N nós que se movimentam, onde o primeiro é chamado $\mathcal{MN}^{(1)}$ (autômato *Source*) que gera pacotes de acordo com as definições de um padrão de comunicação. Os pacotes são enviados através desta cadeia por autômatos do tipo *Relay* chamados de $\mathcal{MN}^{(i)}$, onde i varia entre 2 e $(N-1)$, até chegar ao último nó que foi chamado de $\mathcal{MN}^{(N)}$ (ou autômato *Sink*).

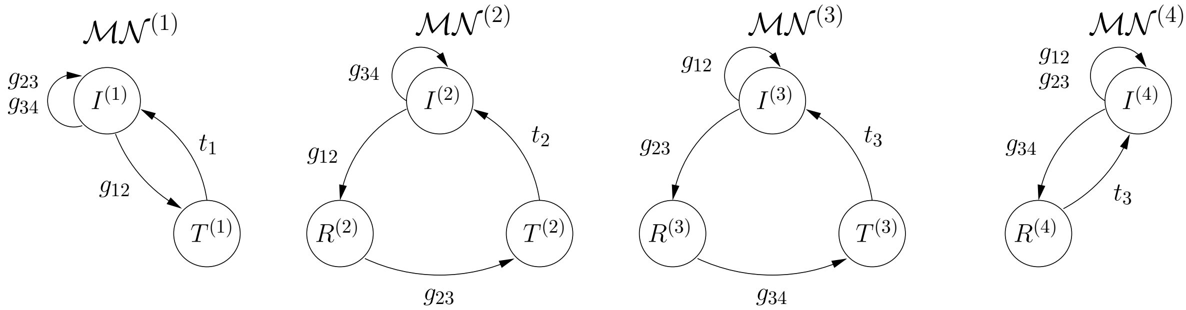


Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	$t_1, \ldots, t_4$	μ	syn	$r_{2,3}$	$f_{2,3}$
syn	t_5	μ	syn	$r_{3,4}$	$f_{3,4}$
syn	$g_{1,2}$	$f_{1,2}$	syn	$r_{4,5}$	$f_{4,5}$
			syn	$r_{5,6}$	$f_{5,6}$

$$
\begin{aligned}
f_{1,2} &= [((st\,\mathcal{MN}^{(3)} \,!= T^{(3)}) \;\&\&\; (st\,\mathcal{MN}^{(4)} \,!= T^{(4)}))] * \lambda \\
f_{2,3} &= [((st\,\mathcal{MN}^{(1)} \,!= T^{(1)}) \;\&\&\; (st\,\mathcal{MN}^{(4)} \,!= T^{(4)}) \;\&\&\; (st\,\mathcal{MN}^{(5)} \,!= T^{(5)}))] * \lambda \\
f_{3,4} &= [((st\,\mathcal{MN}^{(1)} \,!= T^{(1)}) \;\&\&\; (st\,\mathcal{MN}^{(2)} \,!= T^{(2)}) \;\&\&\; (st\,\mathcal{MN}^{(5)} \,!= T^{(5)}) \;\&\&\; (st\,\mathcal{MN}^{(6)} \,!= T^{(6)}))] * \lambda \\
f_{4,5} &= [((st\,\mathcal{MN}^{(2)} \,!= T^{(2)}) \;\&\&\; (st\,\mathcal{MN}^{(3)} \,!= T^{(3)}) \;\&\&\; (st\,\mathcal{MN}^{(6)} \,!= T^{(6)}))] * \lambda \\
f_{5,6} &= [((st\,\mathcal{MN}^{(3)} \,!= T^{(3)}) \;\&\&\; (st\,\mathcal{MN}^{(4)} \,!= T^{(4)}))] * \lambda
\end{aligned}
$$

Figura 7.2: Modelo de *Redes Wireless ad hoc* com seis nós definido em ATG.

Os modelos SAN aqui descritos possuem genericamente $(N-2)$ eventos locais e N eventos sincronizantes para descritores contantes. Nesse caso, existem $[(N-2)+2N]$ termos tensoriais. Para o caso de descritores generalizados, tem-se $(N-2)$ eventos locais e igualmente N eventos sincronizantes com N termos. Este modelo foi aumentado para refletir um maior número de nós em uma rede com $10, 12, 14$ e 16 nós.



Tipo	Evento	Taxa	Tipo	Evento	Taxa
loc	t_1	μ	syn	g_{12}	λ_{12}
loc	t_2	μ	syn	g_{23}	λ_{23}
syn	t_3	μ	syn	g_{34}	λ_{34}

Figura 7.3: Modelo de *Redes Wireless ad hoc* com quatro nós e taxas constantes.

A Figura 7.3 mostra um modelo com um descritor Markoviano baseado em ATC de uma Rede *Wireless* ad hoc com quatro nós que foi traduzida da correspondente rede original que utilizava definições de descritores baseados em ATG. Observa-se que para essa classe de modelo existem mais eventos sincronizantes no descritor clássico devido à conversão dos elementos funcionais em sincronizações [BRE05a].

A Tabela 7.4 informa os resultados obtidos para os Experimentos $1, 2, 3$ e 4. Observa-se que Experimento 1 obteve o melhor desempenho com um ganho de aproximadamente *sete* vezes em relação ao Experimento 2. A tabela também mostra que as permutações impactam de forma considerável a memória extra utilizada e, para este caso, no tempo. Nota-se que as matrizes dos termos tensoriais foram reordenadas de forma a otimizar por completo a geração de AUNFs (apenas

Tabela 7.4: Caso *Redes Wireless ad hoc*, modelos em ATC, comparando Experimentos $\{1,2,3,4\}$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
ad10c	26.244	10	≈0	≈0,0000 ± 0,0000	**0,0028 ± 0,0001**	0,0221 ± 0,0001	7,89
ad12c	236.196	12	≈0	≈0,0000 ± 0,0000	**0,0382 ± 0,0001**	0,2583 ± 0,0039	6,76
ad14c	2.125.764	14	≈0	≈0,0000 ± 0,0000	**0,3940 ± 0,0010**	2,7428 ± 0,0012	6,96
ad16c	19.131.876	16	≈0	≈0,0000 ± 0,0000	**4,0235 ± 0,0174**	28,1046 ± 0,0729	6,98
					Experimento 3	Experimento 4	
ad10c	26.244	5.105	159	0,0086 ± 0,0010	0,0038 ± 0,0001	0,0157 ± 0,0001	4,13
ad12c	236.196	45.929	1.435	0,1196 ± 0,0008	0,0508 ± 0,0008	0,2031 ± 0,0004	3,99
ad14c	2.125.764	413.345	12.917	1,1962 ± 0,0041	0,5024 ± 0,0041	2,1787 ± 0,0020	4,34
ad16c	19.131.876	3.720.089	116.252	11,3510 ± 0,0593	5,0693 ± 0,0593	22,6656 ± 0,0008	4,47

16 para o maior caso, com 16 nós), refletindo no melhor tempo obtido. Também evidenciou que não necessariamente o uso de mais memória implicará em uma melhor execução do Algoritmo *Split*. No caso do Experimento 3, foram gastos 116 Mb e ainda assim o tempo foi superior ao constatado no Experimento 1.

Tabela 7.5: Caso *Redes Wireless ad hoc* em ATG, comparando Experimentos $\{5,6,7,8,9\}$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	Experimento 5	Experimento 9	
ad10f	26.244	1.513	47	≈0,0000 ± 0,0000	0,0177 ± 0,0001	0,0438 ± 0,0002	2,47
ad12f	236.196	14.257	445	0,0196 ± 0,005	0,2262 ± 0,0004	0,5679 ± 0,0009	2,51
ad14f	2.125.764	134.137	4.191	0,2432 ± 0,013	2,6409 ± 0,0029	6,8353 ± 0,0076	2,59
ad16f	19.131.876	1.259.713	39.366	2,5310 ± 0,042	29,8566 ± 0,0445	79,3119 ± 0,2193	2,66
					Experimento 6	Experimento 9	
ad10f	26.244	406	12	≈0,0000 ± 0,0000	0,0067 ± 0,0001	0,0438 ± 0,0002	6,54
ad12f	236.196	568	17	≈0,0000 ± 0,0000	0,0937 ± 0,0001	0,5679 ± 0,0009	6,06
ad14f	2.125.764	730	22	≈0,0000 ± 0,0000	1,0963 ± 0,0020	6,8353 ± 0,0076	6,23
ad16f	19.131.876	892	27	≈0,0000 ± 0,0000	11,5802 ± 0,0101	79,3119 ± 0,2193	6,85
					Experimento 7	Experimento 9	
ad10f	26.244	406	12	≈0,0000 ± 0,0000	**0,0040 ± 0,0001**	0,0438 ± 0,0002	10,95
ad12f	236.196	568	17	≈0,0000 ± 0,0000	**0,0568 ± 0,0001**	0,5679 ± 0,0009	9,99
ad14f	2.125.764	730	22	≈0,0000 ± 0,0000	**0,6434 ± 0,0011**	6,8353 ± 0,0076	10,62
ad16f	19.131.876	892	27	≈0,0000 ± 0,0000	**6,6018 ± 0,0078**	79,3119 ± 0,2193	12,01
					Experimento 8	Experimento 9	
ad10f	26.244	10	≈0	0,0000 ± 0,0000	0,0165 ± 0,0001	0,0438 ± 0,0002	2,65
ad12f	236.196	12	≈0	0,0000 ± 0,0000	0,2137 ± 0,0013	0,5679 ± 0,0009	2,66
ad14f	2.125.764	14	≈0	0,0000 ± 0,0000	2,5059 ± 0,0027	6,8353 ± 0,0076	2,73
ad16f	19.131.876	16	≈0	0,0000 ± 0,0000	28,3623 ± 0,0241	79,3119 ± 0,2193	2,80

A Tabela 7.5 exemplifica os resultados para os Experimentos $5,6,7,8$ e 9 para o caso dos modelos com descritores generalizados. No Experimento 5, tem-se a avaliação das funções na parte estruturada. Nota-se, em relação ao Experimento 9, um ganho de aproximadamente $2{,}6$ vezes. Para este caso, foi utilizado por volta de 40 Mb de memória extra no Algoritmo *Split* para o caso dos 16 nós, um valor muito superior ao requerido pelos demais experimentos. No entanto, apesar de

ambos os Experimentos 6 e 7 utilizarem a mesma quantidade de memória, executam em tempos completamente diferentes (um em $11{,}58$ segundos e o outro em $6{,}60$ segundos, respectivamente, para o caso dos 16 nós na rede *wireless*). Esse fato pode ser explicado pela existência de matrizes do tipo elemento no descritor, que, como mencionado anteriormente, estas matrizes não aumentam o número de AUNFs necessários e mesmo assim, por deixar apenas identidades na parte estruturada, executam de forma otimizada. O ganho observado foi o maior constatado em todas as variações de modelos executados, da ordem de 12 vezes mais rápido, para um PSS de tamanho considerado elevado para este caso em questão, ou seja, aproximadamente 19 milhões.

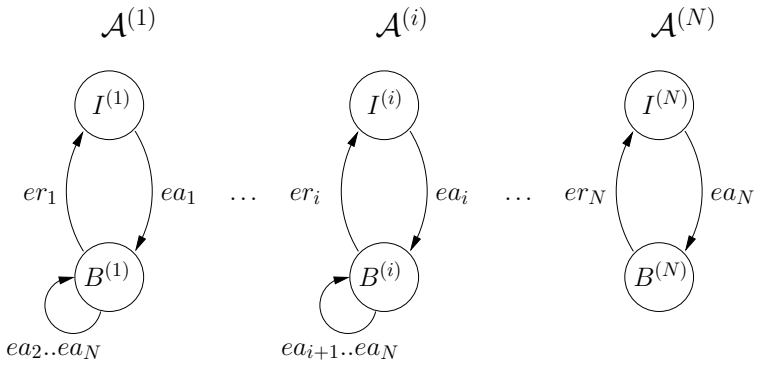
O Experimento 8 avalia as funções na parte estruturada e move as matrizes identidade da parte esparsa para a parte estruturada. Na parte estruturada só existem matrizes funcionais ou identidades. Esse experimento mostra que foram gerados poucos AUNFs, mas mesmo assim, os tempos obtidos foram parecidos aos verificados no Experimento 5. Esse fato corrobora que a avaliação de funções onera o desempenho da MVD, pois observa-se que, ao avaliar esses elementos funcionais e enviar as identidades para a parte estruturada os ganhos são superiores, como visto no Experimento 7. Este caso generalizado para 16 nós, precisou de $6{,}60$ segundos para uma iteração, valores ainda superiores aos $4{,}02$ segundos necessários no caso clássico. Para esses casos, observa-se também que é melhor converter os descritores de ATG para ATC. No entanto, o Experimento 8 usa uma memória extra baixa, pois precisa no maior caso (para 16 nodos) de apenas 16 AUNFs.

Constata-se que este caso específico das *Redes Wireless ad hoc* também possui uma outra característica marcante, todos os eventos possuem muitas identidades e todas as matrizes restantes possuem apenas um elemento não-nulo, fazendo com que cada termo produza apenas *um* AUNF. Esse é um dos melhores casos para o Algoritmo *Split*, pois a memória extra gasta é desprezível e o tempo de solução é mais rápido, conforme visto pela execução dos experimentos. Nota-se que os modelos que seguem essas características resultam em um tempo mais otimizado para realizar a MVD. Observa-se que o Algoritmo *Shuffle*, apesar de realizar importantes otimizações para as matrizes identidade, ainda é necessário tratar das matrizes esparsas, calculando deslocamentos na estrutura tensorial a cada iteração.

7.2.3 Modelo *First Available Server*, ou FAS

Esta seção apresenta o modelo SAN construído para analisar a disponibilidade de servidores, para N servidores (o modelo é denominado *First Available Server*). Cada servidor $A^{(i)}$ pode ser descrito como possuidor de dois estados distintos: *Idle* (em espera – $I^{(i)}$) e *Busy* (ocupado – $B^{(i)}$). Neste exemplo, pacotes que chegam no *terminal de servidores bloqueados* (ou *servers switch block*), partem através da primeira porta de saída (ou servidor) que não está ocupada, a única restrição é a de que pelo menos um servidor não esteja bloqueado.

O modelo está definido na Figura 7.4 e pode ser visto como uma ferramenta de análise de um tipo de sistema de filas (por exemplo, ocupação de linhas de *call centers*). Cada pacote que chega na fila pode avançar tão logo seja possível que um determinado servidor esteja livre, de acordo com uma prioridade pré-estabelecida, do primeiro (1) ao último servidor (N) e PSS $= 2^N$ estados.



Tipo	Evento	Taxa
loc	ea_1	λ
loc	er_1	μ
syn	ea_2	λ
loc	er_2	μ
⋮	⋮	⋮
syn	ea_N	λ
loc	er_N	μ

Figura 7.4: Modelo *First Available Server*.

Este modelo está definido apenas com taxas constantes (ATC). A Tabela 7.6 mostra os resultados para os Experimentos $1, 2, 3$ e 4. A partir da tabela observa-se que para esse modelo em particular, não importa reorganizar as matrizes dos termos tensoriais (Experimento 1) ou cortar na última matriz constante do termo (Experimento 3), pois o método realiza a iteração com uma pequena diferença em tempo. Outra característica deste modelo é precisar de um número reduzido de AUNFs para executar o método. Para 20 servidores são necessários 19 AUNFs. Isso faz com que a memória extra requerida seja negligenciável. O ganho verificado com o Algoritmo *Split* frente ao Algoritmo *Shuffle* foi da ordem de nove vezes em tempo, para a maioria dos casos e sete vezes para os casos onde os termos foram reordenados. Este modelo está definido apenas para descritores clássicos.

Tabela 7.6: Caso *FAS*, descritores clássicos, comparando Experimentos {1,2,3,4}.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
fas18c	262.144	17	≈0	≈0,0000 ± 0,0000	**0,0795 ± 0,0001**	0,7029 ± 0,0043	8,84
fas19c	524.288	18	≈0	≈0,0000 ± 0,0000	**0,1726 ± 0,0003**	1,5550 ± 0,0012	9,01
fas20c	1.048.576	19	≈0	≈0,0000 ± 0,0000	**0,3660 ± 0,0026**	3,4416 ± 0,0087	9,40
					Experimento 3	Experimento 4	
fas18c	262.144	17	≈0	≈0,0000 ± 0,0000	**0,0796 ± 0,0001**	0,5538 ± 0,0008	6,96
fas19c	524.288	18	≈0	≈0,0000 ± 0,0000	**0,1725 ± 0,0001**	1,2254 ± 0,0014	7,10
fas20c	1.048.576	19	≈0	≈0,0000 ± 0,0000	**0,3616 ± 0,0004**	2,8037 ± 0,0181	7,75

Os próximos modelos SAN foram traduzidos a partir do formalismo PEPA e modelam duas problemáticas distintas: células industriais de produção e um servidor para a Internet. Estes modelos foram traduzidos para redes de autômatos com o objetivo de estudar as características envolvidas na tradução entre estes dois formalismos e as propriedades a serem consideradas neste mapeamento. O método de tradução aplicado foi o mesmo utilizado na Seção 3.4, adaptado para tratar com modelos PEPA ao invés de PEPA nets.

7.2.4 Modelos do formalismo PEPA: *Workcell* e *WebServer*

Para demonstrar a execução de modelos definidos em outros formalismos, foram escolhidos dois modelos em particular, gerados a partir de PEPA. O primeiro, chamado *Workcell*, modela células industriais de produção e o segundo define um servidor para Internet de alta disponibilidade, chamado *WebServer*. Aqui serão feitas breves explicações sobre o funcionamento geral dos sistemas modelados, maiores informações podem ser encontradas em [HOL95, BRA03] para, respectivamente, *Workcell* e *WebServer*. Estes modelos foram traduzidos para SAN de forma direta, gerada a partir da composicionalidade de PEPA observando-se as movimentações nos estados da CTMC. Esta tradução não é a melhor tradução deste tipo de modelagem, mas mesmo assim, produz os resultados de saída correspondentes à solução PEPA.

O modelo *Workcell* modela um sistema complexo de uma célula industrial de produção. O objetivo desta célula é construir placas de metais em uma prensa, consistindo dos seguintes componentes principais: cintos de alimentação, tabelas rotatórias, robôs, prensas, cintos de depósitos e artefatos de movimentação, para citar alguns componentes do sistema. O principal objetivo desta modelagem é inferir índices que demonstrem como estes componentes trabalham em conjunto, com vistas à avaliação de desempenho destes componentes.

Já o modelo *WebServer* descreve um servidor de alta disponibilidade para Internet, composta principalmente por elementos de notícias. Este modelo prevê qualidade de serviço em disponibilidade e tempo de resposta. O objetivo principal deste tipo de modelagem é experimentar com a engenharia de desempenho de aplicações de alta demanda com vistas à verificação de eficiência de pico, entre outras análises. O modelo executado aqui, possui quatro parâmetros: S compreende o número de servidores no sistema, B, denota a capacidade do *buffer* de memória, R diz respeito ao número de leitores e W ao número de escritores existentes no sistema (como trata-se de um sistema de notícias, as atividades compreendem a leitura e a escrita de materiais). Para o caso deste exemplo em particular, foi escolhido trabalhar com um modelo com um PSS da ordem de ≈ 350.000 estados, com os seguintes parâmetros: $S = 4$, $B = 3$, $R = 3$ e $W = 3$.

Tabela 7.7: Casos *Workcell* e *WebServer*, descritores clássicos, comparando Experimentos $\{1,2,3,4\}$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
workcell	320.112	16	≈0	≈0,0000 ± 0,0000	**0,0357 ± 0,0001**	0,2410 ± 0,0001	6,75
webserver	349.920	63	≈1	≈0,0000 ± 0,0000	**0,1087 ± 0,0002**	1,0010 ± 0,0012	9,21
					Experimento 3	Experimento 4	
workcell	320.112	11.662	364	0,0162 ± 0,0014	0,0385 ± 0,0001	0,1707 ± 0,0001	4,43
webserver	349.920	574.453	17.951	1,4496 ± 0,0046	0,2442 ± 0,0006	0,7678 ± 0,0008	3,14

Estes modelos baseados em PEPA existem apenas sob a forma de um descritor clássico e os seus resultados estão sendo mostrados na Tabela 7.7. Esta tabela informa o tempo de execução para os Experimentos $1, 2, 3$ e 4. Para o modelo *Workcell*, o tempo verificado é bastante similar nos Experimentos 1 e 3 e, quando comparados ao Experimento 2 e 4 atestam um ganho de seis e

quatro vezes. A diferença entre as execuções está na memória extra gasta em cada experimento com o Algoritmo *Split*. Enquanto que no a memória é desprezível, no Experimento 3 foi necessário 364 Kb.

Para o caso do *WebServer*, a melhor execução foi a alcançada pelo Experimento 1, sendo nove vezes mais rápido quando comparado ao Algoritmo *Shuffle* realizando-se permutações. No caso de se escolher não fazer permutações (Experimentos 3 e 4), o ganho ainda assim é considerável (da ordem de três vezes), mas são necessários 18 Mb para salvar os AUNFs. Visto que é possível reordenar as matrizes dos termos, não gastar em memória extra e ainda assim ganhar em tempo de execução, esta é uma boa alternativa para a solução deste modelo.

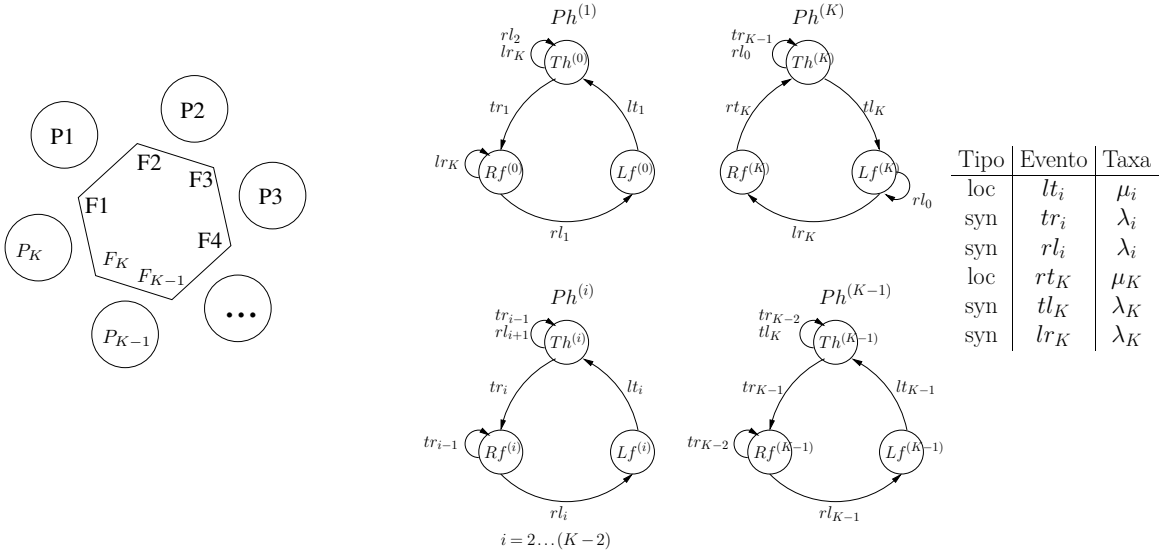
7.2.5 Modelo do *Jantar dos Filósofos*

Esta seção apresenta um modelo clássico para avaliação de desempenho para análise de exclusão mútua em compartilhamento de recursos. O modelo em questão é uma abstração chamada de *problema do Jantar dos Filósofos* e, resumidamente, pode ser descrita da seguinte forma: K filósofos estão sentados em uma mesa e a eles somente são permitidas duas ações, comer ou pensar. Os filósofos estão sentados em uma mesa circular com uma tijela de comida no centro desta (de acordo com a Figura 7.5a). Um garfo F_k está posicionado entre cada filósofo P_k, assim, cada um possui um garfo à sua esquerda e um garfo à sua direita. Para comer, um filósofo necessita de dois garfos nas suas mãos, simultaneamente (o problema é que não existem recursos suficientes para todos ao mesmo tempo).

O modelo SAN da Figura 7.5b possui K autômatos do tipo $Ph^{(k)}$ representando os filósofos, cada um com três estados: $Th^{(k)}$ (pensando), $Lf^{(k)}$ (pegando garfo da esquerda), $Rf^{(k)}$ (pegando o garfo situado à direita). O filósofo pode reservar o garfo à sua esquerda ou direita para comer utilizando necessariamente dois garfos disponíveis. Para evitar *deadlock* fica estabelecida uma ordem para pegar os garfos na mesa, para cada filósofo presente no modelo. O PSS deste modelo é dado por 3^K estados.

Os resultados para o modelo do *Jantar dos Filósofos* estão mostrados na Tabela 7.8. Os modelos foram definidos apenas para descritores clássicos, onde variou-se o número de filósofos entre 12 e 16. Observa-se que o PSS cresce de acordo com o número de filósofos, sendo que para 16 filósofos, ele é da ordem de 43 milhões de estados. Não foi possível computar para este modelo os tempos dos Experimentos 3 e 4 uma vez que foi preciso por volta de $1{,}2$ Gb de memória apenas para armazenar a tabela referente aos AUNFs nos cortes estabelecidos nos experimentos. Como este modelo é possuidor de muitas identidades, o Experimento 1 mostra claramente o efeito de se passar estes elementos para a parte estruturada do termo: apenas 48 AUNFs são necessários para o caso com 16 filósofos. E esse fato se reflete no tempo gasto para executar uma iteração, onde os ganhos médios calculados ficaram na ordem de quatro vezes, entre os Experimentos 1 e 2, que executaram permutações nos termos tensoriais.

A Tabela 7.8 mostra a influência dos eventos sincronizantes na geração dos termos tensoriais, pois, ao aumentar o número de filósofos em um determinado modelo, implica no aumento propor-



Tipo	Evento	Taxa
loc	lt_i	μ_i
syn	tr_i	λ_i
syn	rl_i	λ_i
loc	rt_K	μ_K
syn	tl_K	λ_K
syn	lr_K	λ_K

Figura 7.5: (a) configuração possível de filósofos em uma mesa. (b) modelo SAN para K filósofos.

Tabela 7.8: Caso *Jantar dos Filósofos*, modelos em ATC, comparando Experimentos {1,2,3,4}.

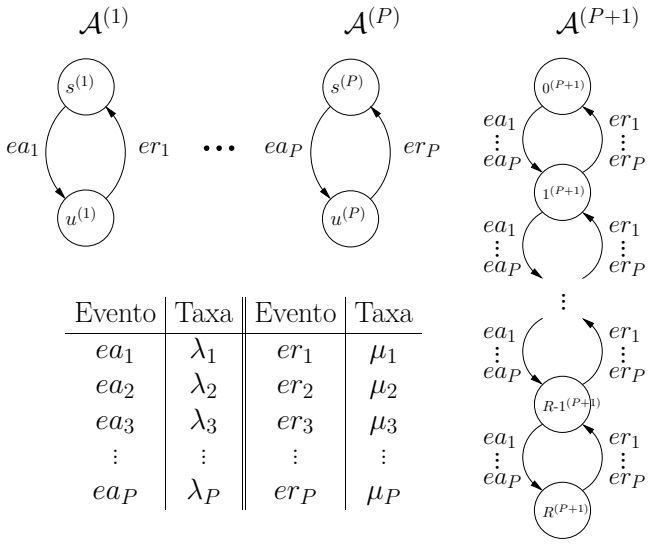
Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
phil12c	531.441	36	≈1	≈0,0000 ± 0,0000	**0,1995 ± 0,0007**	0,7319 ± 0,0019	3,67
phil13c	1.594.323	39	≈1	≈0,0000 ± 0,0000	**0,6270 ± 0,0008**	2,3617 ± 0,0009	3,77
phil14c	4.782.969	42	≈1	≈0,0000 ± 0,0000	**1,9989 ± 0,0016**	7,6590 ± 0,0018	3,83
phil15c	14.348.907	45	≈1	≈0,0000 ± 0,0000	**6,2543 ± 0,0318**	24,2281 ± 0,0607	3,87
phil16c	43.046.721	48	≈1	≈0,0000 ± 0,0000	**19,7111 ± 0,0119**	77,4296 ± 0,0279	3,93
					Experimento 3	Experimento 4	
phil12c	531.441	501.914	15.684	1,2862 ± 0,0032	0,3146 ± 0,0006	0,4960 ± 0,0005	1,58
phil13c	1.594.323	1.505.747	47.054	4,0062 ± 0,0119	0,9798 ± 0,0020	1,6074 ± 0,0016	1,64
phil14c	4.782.969	4.517.246	141.163	12,6720 ± 0,0311	3,0583 ± 0,0037	5,1728 ± 0,0047	1,70
phil15c	14.348.907	13.551.743	423.491	39,7944 ± 0,0946	9,8197 ± 0,0102	16,6351 ± 0,0203	1,70
phil16c	43.046.721	40.655.234	1.270.476	≈136,1200	–	–	–

cional de comunicações que ocorrem e, consequentemente, de termos tensoriais. As variações do número de filósofos do modelo foram chamadas de *philK*, onde K representa o número de filósofos existentes. Para o caso de 15 filósofos, por exemplo, foi necessário 423 Mb de memória extra para o Algoritmo *Split* e o método executou em $9{,}8$ segundos, enquanto que gastando-se por volta de 1 Kb, foi executado em $6{,}2$ segundos. O modelo do *Jantar dos Filósofos*, assim como o modelo das *Redes Wireless ad hoc* evidenciam quando a abordagem utilizada pelo Algoritmo *Split* é mais eficaz que o Algoritmo *Shuffle* em termos de tempo de execução.

7.2.6 Modelo *Compartilhamento de Recursos*

O modelo de *Compartilhamento de Recursos* (*Resource Sharing*) trata os casos onde N processos compartilham R recursos. A Figura 7.6 mostra um sistema de compartilhamento de recursos onde

cada processo é representado por um autômato $A^{(i)}$ composto por dois estados: $S^{(i)}$ (dormindo, ou *sleeping*) e $U^{(i)}$ (usando ou *using*). Um conjunto de recursos é simbolizado pelo autômato $A^{(N+1)}$ o qual possui $R+1$ estados indicando o número de recursos que estão sendo utilizados.



Evento	Taxa	Evento	Taxa
ea_1	λ_1	er_1	μ_1
ea_2	λ_2	er_2	μ_2
ea_3	λ_3	er_3	μ_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ea_P	λ_P	er_P	μ_P

Figura 7.6: Modelo de *Compartilhamento de Recursos* modelado em SAN.

Este modelo apresenta um conjunto de autômatos no qual os eventos sincronizantes estão relacionados a apenas dois autômatos cada um. Isto significa que os termos tensoriais serão compostos por muitas matrizes de baixa esparsidade e igualmente muitas identidades. Os principais resultados para o modelo do *Compartilhamento de Recursos* está mostrado na Tabela 7.9. A tabela mostra as diferentes configurações de redes de autômatos, denominadas *rsP_R* que indicam no nome do modelo o número de processos P existentes para R recursos compartilhados. Verifica-se as comparações entre os tempos obtidos de *uma* iteração entre os Experimentos $1,2$ e $3,4$.

Tabela 7.9: Caso *Compartilhamento de Recursos* com ATC comparando Experimentos $\{1,2,3,4\}$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 2	
rs14_10c	180.224	280	8	≈0,0000 ± 0,0000	**0,1067 ± 0,0001**	0,3127 ± 0,0005	2,93
rs14_11c	196.608	308	9	≈0,0000 ± 0,0000	**0,1173 ± 0,0001**	0,3409 ± 0,0008	2,90
rs15_15c	524.288	450	14	≈0,0000 ± 0,0000	**0,3448 ± 0,0007**	1,0169 ± 0,0012	2,95
rs20_25c	27.262.976	1000	31	≈0,0000 ± 0,0000	**24,7120 ± 0,0106**	88,0908 ± 0,0383	3,56
					Experimento 3	Experimento 4	
rs14_10c	180.224	327.660	10.239	0,9914 ± 0,0026	0,2019 ± 0,0003	0,2093 ± 0,0002	1,04
rs14_11c	196.608	360.426	11.263	1,0720 ± 0,0028	0,2226 ± 0,0004	0,2306 ± 0,0003	1,03
rs15_15c	524.288	983.010	30.719	2,9890 ± 0,0263	0,6448 ± 0,0140	0,6812 ± 0,0008	1,06
rs20_25c	27.262.976	52.428.750	1.638.398	≈234,9400	–	–	–

A tabela mostra que para o caso onde existem 20 processos para 25 recursos (o nome do modelo é $rs20\_25c$), não foi possível computar os tempos devido ao alto custo em memória para salvar os AUNFs, da ordem de $1{,}6$ Gb. Esta classe de modelos também possui muitas identidades e o

experimento que obteve os maiores ganhos foi o Experimento 1, melhor em média três vezes que o Experimento 2, onde foram necessários 31 Kb de memória para guardar os AUNFs.

7.2.7 Modelo *Alternate Service Patterns*, ou ASP

Este exemplo descreve um sistema de rede aberta composto por quatro filas (F_1, F_2, F_3 e F_4), representadas pelos autômatos $\mathcal{A}^{(1)}$,$\mathcal{A}^{(2)}$,$\mathcal{A}^{(3)}$,$\mathcal{A}^{(4)}$, com capacidades finitas (K_1, K_2, K_3 e K_4) respectivamente. O padrão de roteamento de consumidores para esse caso, chegam em F_1 e F_2 com taxa λ_1 e λ_2, podendo sair de F_1 para F_3 se existir espaço (comportamento bloqueante) e também sair de F_2 para F_3 se existir espaço ou sair do modelo em caso contrário (comportamento de perda). Os consumidores também podem sair de F_3 para F_4 com comportamento bloqueante.

Enquanto F_1, F_2 e F_4 possuem comportamento de serviço padrão (no caso, *single*), *i.e.*, uma mesma taxa média de serviço para todos os consumidores (μ_1, μ_2 e μ_4 respectivamente), a fila F_3 possui um comportamento alternado de padrão de serviço (*Alternate Service Patterns*, ou ASP). A taxa de serviço para esta fila varia de acordo com P diferentes taxas de padrão de serviço (μ_{31}, ..., μ_{3P}). A fila F_3 pode trocar seu padrão simultaneamente de acordo com o final do serviço de um consumidor. Isso faz com que quando um consumidor é servido pelo padrão P_i, F_3 pode permanecer servindo o próximo consumidor no mesmo padrão com probabilidade π_{ii}, ou pode alternar para um padrão de serviço diferente P_j com probabilidade π_{ij} (para todos os padrões de serviço $P_i : \sum_{j=1}^{P} \pi_{ij} = 1$).

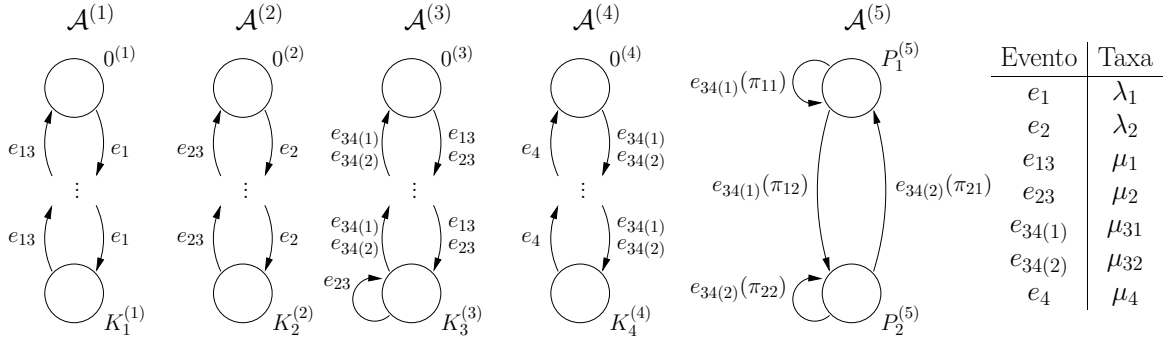


Figura 7.7: Modelo *Alternate Service Patterns* com descritor clássico.

O modelo SAN descrito na Figura 7.7 é composto por um autômato usado para cada fila com serviço simples ($\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$ e $\mathcal{A}^{(4)}$) e outros dois autômatos para a fila de padrão de serviço alternado ($\mathcal{A}^{(3)}$ e $\mathcal{A}^{(5)}$). O modelo ASP mostra que o melhor tempo encontrado foi correspondente ao Experimento 4, conforme a Tabela 7.10. Este experimento obteve os melhores índices para uma iteração, quando não são executadas permutações nos termos tensoriais. A tabela mostra os casos com descritores clássico onde os tempos, apesar de serem parecidos entre os Experimentos 1 e 4, mostram um ganho relativo implicando que a melhor abordagem a ser escolhida nestes casos, é provavelmente a adotada pelo Algoritmo *Shuffle*.

Tabela 7.10: Caso *ASP*, modelos em ATC, comparando Experimentos {1,2,3,4}.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	Experimento 1	Experimento 2	
asp10c	252.890	10.298	321	0,0142 ± 0,0014	0,1506 ± 0,0005	0,1911 ± 0,0007	1,27
asp12c	342.732	21.036	657	0,0398 ± 0,0004	0,2478 ± 0,0001	0,2991 ± 0,0006	1,21
asp14c	399.854	28.524	891	0,0518 ± 0,0011	0,3309 ± 0,0032	0,4013 ± 0,0013	1,21
					Experimento 3	**Experimento 4**	
asp10c	252.890	2.094.078	65.439	5,1660 ± 0,0670	0,5146 ± 0,0015	**0,1490 ± 0,0006**	0,29
asp12c	342.732	3.508.284	109.633	9,0288 ± 0,0488	0,8481 ± 0,0029	**0,2331 ± 0,0008**	0,27
asp14c	399.854	4.473.756	149.179	12,6500 ± 0,0829	1,1451 ± 0,0049	**0,3130 ± 0,0013**	0,23

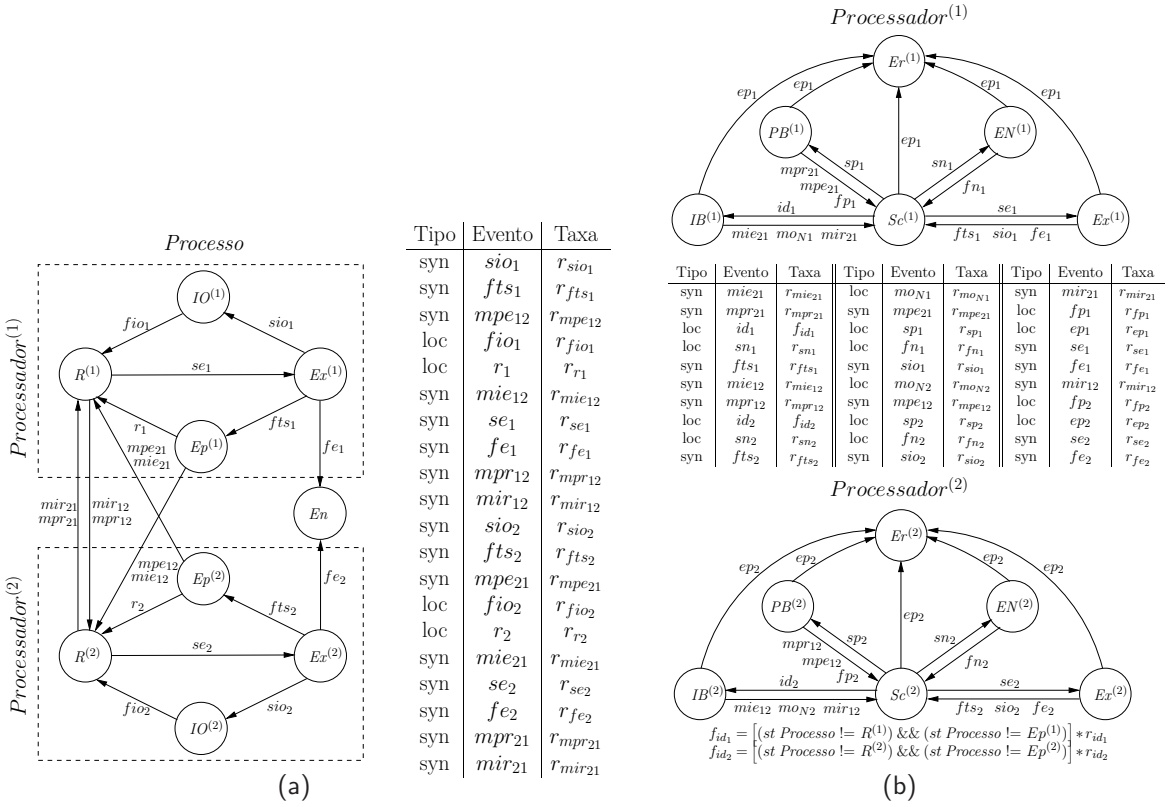
7.2.8 Modelo *Non-Uniform Memory Access*, ou NUMA

Máquinas NUMA são arquiteturas utilizadas em conjunção com diversos sistemas operacionais. O próximo exemplo descreve uma modelagem de partes do sistema operacional Linux para ser usado em máquinas NUMA. O modelo permite uma análise compreensiva de diferentes tamanhos de processos para um número variado de processadores de uma máquina NUMA, generalizando o conceito sobre o comportamento dos processos sob a ótica de um único processo em um ambiente multiprocessado. Esta representação genérica permite que sejam analisadas as chances de um processador falhar, analisa o processo de migração de processos entre os processadores e o funcionamento do escalonador do sistema operacional Linux [CHA06].

Os eventos internos do modelo NUMA envolvem sempre dois autômatos e cada termo tensorial possui duas matrizes com apenas um elemento não nulo e o resto, matrizes do tipo identidade. Este fato é verificado pelo número de AUNFs que são necessários para armazenamento deste modelo que é igual ao número de eventos existentes. Observando-se os resultados nota-se que este modelo obteve os melhores ganhos em tempo frente ao Algoritmo *Shuffle*, da ordem de 17 e 28 vezes para seis e oito processadores. Foram necessários entre cinco e nove *Kilobytes* de memória para armazenamento dos AUNFs. Este adequa-se ao Algoritmo *Split* pois gasta pouca memória extra com ganhos expressivos.

O autômato *Processo* da Figura 7.8a representa os processos escalonados para R processadores. Contém quatro estados: $IO^{(i)}$ (processo está esperando uma operação de entrada/saída), $R^{(i)}$ (processo encontra-se na fila de "pronto", ou seja, esperando para ser escalonado no $i^{\text{ésimo}}$ processador), $Ex^{(i)}$ (representando que o processo está sendo executado no processador correspondente) e $Ep^{(i)}$ (processo está na fila de processos "expirados", ou seja, terminou sua fatia de tempo e está esperando para ser movido para a fila de processos "prontos"). Para cada autômato *Processador* $Proc^{(i)}$, onde $i = 1 \ldots R$, existe mais um estado chamado En, simbolizando que o processo terminou sua execução e não faz mais parte do sistema.

Já o autômato *Processador* da Figura 7.8b contém seis estados: $Er^{(i)}$ (algum erro aconteceu e o processador encontra-se inoperante), $Pb^{(i)}$ (o processador está executando seu algoritmo periódico de balanceamento de carga), $En^{(1)}$ (o processador está ocupado, executando qualquer outro



Tipo	Evento	Taxa
syn	sio_1	r_{sio_1}
syn	fts_1	r_{fts_1}
syn	mpe_{12}	$r_{mpe_{12}}$
loc	fio_1	r_{fio_1}
loc	r_1	r_{r_1}
syn	mie_{12}	$r_{mie_{12}}$
syn	se_1	r_{se_1}
syn	fe_1	r_{fe_1}
syn	mpr_{12}	$r_{mpr_{12}}$
syn	mir_{12}	$r_{mir_{12}}$
syn	sio_2	r_{sio_2}
syn	fts_2	r_{fts_2}
syn	mpe_{21}	$r_{mpe_{21}}$
loc	fio_2	r_{fio_2}
loc	r_2	r_{r_2}
syn	mie_{21}	$r_{mie_{21}}$
syn	se_2	r_{se_2}
syn	fe_2	r_{fe_2}
syn	mpr_{21}	$r_{mpr_{21}}$
syn	mir_{21}	$r_{mir_{21}}$

Tipo	Evento	Taxa	Tipo	Evento	Taxa	Tipo	Evento	Taxa
syn	mie_{21}	$r_{mie_{21}}$	loc	mo_{N1}	$r_{mo_{N1}}$	syn	mir_{21}	$r_{mir_{21}}$
syn	mpr_{21}	$r_{mpr_{21}}$	syn	mpe_{21}	$r_{mpe_{21}}$	loc	fp_1	r_{fp_1}
loc	id_1	f_{id_1}	loc	sp_1	r_{sp_1}	loc	ep_1	r_{ep_1}
loc	sn_1	r_{sn_1}	loc	fn_1	r_{fn_1}	syn	se_1	r_{se_1}
syn	fts_1	r_{fts_1}	syn	sio_1	r_{sio_1}	syn	fe_1	r_{fe_1}
syn	mie_{12}	$r_{mie_{12}}$	loc	mo_{N2}	$r_{mo_{N2}}$	syn	mir_{12}	$r_{mir_{12}}$
syn	mpr_{12}	$r_{mpr_{12}}$	syn	mpe_{12}	$r_{mpe_{12}}$	loc	fp_2	r_{fp_2}
loc	id_2	f_{id_2}	loc	sp_2	r_{sp_2}	loc	ep_2	r_{ep_2}
loc	sn_2	r_{sn_2}	loc	fn_2	r_{fn_2}	syn	se_2	r_{se_2}
syn	fts_2	r_{fts_2}	syn	sio_2	r_{sio_2}	syn	fe_2	r_{fe_2}

Figura 7.8: Eventos para (a) autômato do Processo *NUMA*. (b) autômatos de dois processadores.

processo), $Sc^{(i)}$ (o processador está executando seu algoritmo de escalonamento), $IB^{(i)}$ (em modo de espera mas acionando seu algoritmo de balanceamento de carga) e, por fim, $Ex^{(i)}$ (processador executando processo do autômato *Process*).

Sendo R o número de processadores de um dado modelo, o seu PSS é dado por $[(4^R+1)\times 6^R]$. O número total de autômatos é genericamente dado por $(R+1)$, contendo 18 eventos locais e 32 eventos sincronizantes. O descritor, neste caso, é formado por somas tensoriais com $R+1$ matrizes locais e 64 termos tensoriais produto com $R+1$ matrizes. Para as execuções sequenciais deste modelo considerado foram utilizados $R=6$ e $R=8$.

Tabela 7.11: Caso *NUMA*, modelos em ATC, comparando Experimentos {1,4}.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 4	
NUMA6c	1.166.400	174	5	0,0000 ± 0,0000	**0,3793 ± 0,0002**	6,5213 ± 0,0039	17,19
NUMA8c	55.427.328	312	9	0,0000 ± 0,0000	**25,7019 ± 0,0236**	730,1900 ± 1,7020	28,41

A Tabela 7.11 mostra os resultados obtidos para dois tipos de modelos: uma máquina NUMA que descreve seis processadores e outra, oito processadores. A variação do PSS é de ≈1 milhão de estados para o caso de seis processadores até ≈55 milhões para oito processadores. Este modelo alinha-se ao modelo das *Redes Wireless ad hoc* quanto à afetação dos eventos sincronizantes dos descritores clássicos. Para seis processadores foram necessários sete autômatos e 174 eventos e para

o caso de oito processadores, nove autômatos e 312 eventos. Em termos de memória auxiliar para o Algoritmo *Split*, gasta respectivamente 5 Kb e 9 Kb, demonstrando que os eventos existentes no modelo afetam poucos autômatos. A solução usando o Algoritmo *Split* é favorecida com o uso de permutações e cortes na última matriz constante.

7.2.9 Modelo *Open Queueing Network*, ou OQN

Redes de Filas de Espera Abertas representam filas com comportamentos independentes (do ponto de vista da chegada e saída de clientes de cada fila presente no sistema) e interações com sincronização (considerando-se as trocas de clientes entre as diferentes filas). Um exemplo de um modelo de filas deste tipo é apresentado na Figura 7.9, contendo duas filas de capacidade três e uma fila com capacidade dois. O modelo possui uma fila com comportamento bloqueante (para os clientes da Fila $\mathcal{Q}^{(1)}$ para a Fila $\mathcal{Q}^{(2)}$) e outra fila com comportamento de perda (para os clientes da Fila $\mathcal{Q}^{(1)}$ para a Fila $\mathcal{Q}^{(3)}$).

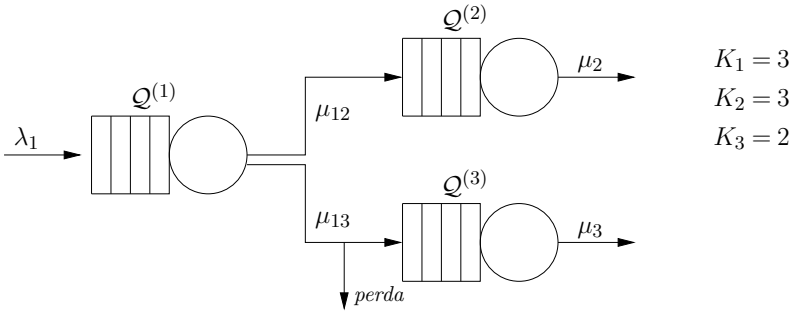


Figura 7.9: Modelo *OQN* com bloqueio e perda.

Tabela 7.12: Caso *OQN*, descritor clássico, comparando Experimentos $\{1,4\}$.

Modelo	PSS	AUNF			Tempos (s) – 1 iteração		Ganho
		Total	Mem. (Kb)	Cálculo (s)	**Experimento 1**	Experimento 4	
oqn1	2.560.451	50.250	1.570	0,2094 ± 0,0009	**0,4292 ± 0,0028**	0,7307 ± 0,0126	1,70
oqn2	5.723.051	75.250	2.351	0,3320 ± 0,0035	**1,0455 ± 0,0033**	1,7159 ± 0,0305	1,64
oqn3	10.140.651	100.250	3.132	0,4578 ± 0,0017	**1,8723 ± 0,0084**	3,1923 ± 0,0606	1,71
oqn4	15.813.251	125.250	3.914	0,5934 ± 0,0022	**2,9891 ± 0,0136**	5,1428 ± 0,0928	1,72

A Tabela 7.12 mostra os resultados obtidos para OQN contendo três filas com capacidade variável[2]. O ganho em relação ao *Shuffle* manteve-se em $1{,}7$ vezes, considerando-se que o descritor possui termos tensoriais de tamanho três e termos com pelo menos uma matriz identidade, o que demonstrou ser adequado na execução do método *Split*. O total de AUNFs necessários manteve-se igualmente constante, e o maior exemplo, com filas de capacidade 250 foram necessários 4 Mb para salvar a lista de AUNFs. O tempo de criação desta lista variou entre $0{,}2$ segundos e $0{,}6$ segundos.

[2]Capacidade de cada modelo: **oqn1** $= K_1 = 250, K_2 = 100, K_3 = 100$, **oqn2** $= K_1 = 250, K_2 = 150, K_3 = 150$, **oqn3** $= K_1 = 250, K_2 = 200, K_3 = 200$ e **oqn4** $= K_1 = 250, K_2 = 250, K_3 = 250$.

7.3 Estudo de caso: solução do Modelo *Redes Wireless ad hoc*

Tendo-se em vista os resultados expressivos obtidos no modelo de *Redes Wireless ad hoc* conforme explicado na Seção 7.2.2, conduz-se a seguir uma análise mais abrangente das explicações em torno da eficácia da sua execução. O objetivo é explicar porque essa classe de modelos executou mais rapidamente que a abordagem *Shuffle* ao mesmo tempo que utilizou inexpressivas quantidades de memória adicional. A seguir, um detalhamento do modelo e a estrutura do seu descritor Markoviano clássico e generalizado.

7.3.1 Modelo

A explicação deste modelo foi iniciada na Seção 7.2.2. Na presente seção, será dada uma maior ênfase aos detalhes internos do modelo e as escolhas feitas em termos de modelagem tanto para descritores clássicos como para generalizados. Este modelo descreve uma rede *wireless* ad hoc com N autômatos, sendo dois autômatos de dois estados situados nos extremos (um chamado *Source* e o outro *Sink*) e $N-2$ autômatos intermediários de três estados situados entre estes (chamados de *Relay*).

Os estados de cada autômato representam as possibilidades possíveis de troca de estado interno. O autômato *Source* somente pode estar transmitindo pacotes ou em modo de espera (*idle*), sendo representados pelos estados T (para *Transmitting*) e I (para *Idle*), respectivamente. O autômato *Sink*, pode estar em espera (I) ou recebendo (R). Os demais autômatos possuem três estados, podendo estar em espera (I), recebendo (R) ou transmitindo (T) pacotes.

O objetivo dos autômatos extremos é modelar um envio de um pacote de *Source* até *Sink*. Para que a comunicação funcione entre as entidades que compõem o modelo, os eventos devem causar a mudança de estado dos autômatos de uma maneira coordenada e sincronizada. Ao ocorrer um evento que deseja enviar um pacote para o autômato *Sink*, este altera seu estado interno de I para T ao mesmo tempo que o seu autômato "vizinho" do tipo *Relay* altera seu estado de I para R (em modo de espera para recebendo), através do evento $g_{1,2}$ (que observa os estados dos próximos dois autômatos, neste caso, $\mathcal{MN}^3$ e $\mathcal{MN}^4$, e aplica a taxa λ se ambos são diferentes do estado de transmitindo). O processo se repete até o autômato *Sink* mudar seu estado interno de 'em espera' para 'recebendo'. Para escalar o modelo, criam-se mais autômatos do tipo *Relay* e atualizam-se os eventos para descrever as mudanças de estados coerentemente entre os autômatos.

A Tabela 7.13 mostra as características principais da classe das redes *wireless*, informando, para cada caso, o espaço de estados (coluna PSS), o número total de autômatos (em #automatos), os totais de eventos locais e sincronizantes para descritores clássicos (ATC loc e syn) e totais para descritores generalizados (ATG loc e syn). A seguir, a coluna (#funções) mostra o total de funções para o modelo, finalizando pela última coluna (#termos) que indica o total de termos para cada modelo (somando o número de eventos locais com os eventos sincronizantes positivos e negativos). Este modelo possui a particularidade de possuir um número de eventos sincronizantes equivalentes ao número de autômatos, conforme mostra a tabela.

Tabela 7.13: Detalhamento das características dos modelos de *Redes Wireless ad hoc*.

	Modelo	PSS - $\lvert\mathcal{X}\rvert$	#autômatos	eventos - ATC		eventos - ATG		#funções - ATG		#termos ATG/ATC
				loc	syn	loc	syn	total	param.	
1	ad10	26.244	10	8	10	8	10	9	30	28
2	ad12	236.196	12	10	12	10	12	11	38	34
3	ad14	2.125.764	14	12	14	12	14	13	46	40
4	ad16	19.131.876	16	14	16	14	16	15	54	46
5	ad17	57.395.628	17	15	17	15	17	16	58	49
6	ad18	172.186.884	18	16	18	16	18	17	62	52
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i	ad$n$	$3^{n-2} \times 2^2$	n	$n-2$	n	$n-2$	n	$n-1$	$4n-10$	$3n-2$

Note que o último modelo tratável com a ferramenta de solução implementada (GTAexpress) *sem* o uso de memórias auxiliares em disco (*i.e.swapping*) é o caso para 17 nós, atingindo um PSS de ≈57 milhões de estados (o limite atual da ferramenta é de ≈65 milhões de estados, para uma máquina com 4 Gb).

Como o modelo foi originalmente definido usando-se funções para o seu sistema de transição, a análise começará pelo descritor generalizado e seguirá para a conversão das funções em eventos sincronizantes na seção posterior.

7.3.2 Descritor Markoviano generalizado

As taxas funcionais relativas às transições entre os autômatos foram definidas para reforçar, via eventos sincronizantes, a existência de uma única transferência de dados (ou seja, todos os outros autômatos não estão no estado "transmitindo"). Essas definições impactam tanto a solução do modelo por MVD quanto na formação do descritor Markoviano correspondente.

A Tabela 7.14 refere-se aos Experimentos 5 a 8 (a definição completa está descrita na Seção 7.1), equivalentes à solução de descritores generalizados. Apenas relembrando, o Experimento 5 realiza as avaliações de funções na parte estruturada e o Experimento 6, na parte esparsa. O Experimento 7 também avalia na parte esparsa mas, usando o Algoritmo *Split*, à direita do ponto de corte (parte estruturada) existem somente matrizes do tipo identidade. O Experimento 8 possui apenas a matriz onde a definição da função propriamente dita está presente e matrizes identidade na parte estruturada.

Na parte esquerda da Tabela 7.14, mostram-se as permutações de matrizes no descritor, variando de 0 até N, sendo N o número total de matrizes presentes em cada termo. Na parte direita, descreve-se quantidade total de elementos não-nulos (exclusivamente para esse caso, este valor equivale-se à um) e a ocorrência de matrizes identidade (através da letra 'i'). Para cada experimento, mostra-se a ordem escolhida das matrizes e o ponto de corte σ (aqui exemplificado pelo caractere '|') refletido em ambos os lados da tabela. No fim de cada experimento, informa-se o total de AUNFs necessários para o cálculo efetivo da MVD para as diferentes estratégias de corte.

Tabela 7.14: Permutações escolhidas e esparsidades dos termos para o caso *Redes Wireless ad hoc* com 12 nós nos Experimentos 5,6,7 e 8 – *Split* ATG.

	evt	Ordem dos autômatos e ponto de corte σ (|)												Esparsidade das matrizes e ponto de corte σ (|)										
Experimento 5	1.	10	11|	0	1	2	3	4	5	6	7	8	9	1	1|	i	i	i	i	i	i	i	i	i	i
	2.	1	4	5	6	7	8	9	10	11|	2	3	0	1	i	i	i	i	i	i	i	i|	i	i	1
	3.	2	5	6	7	8	9	10	11|	0	3	4	1	1	i	i	i	i	i	i	i|	i	i	i	1
	4.	3	6	7	8	9	10	11|	0	1	4	5	2	1	i	i	i	i	i	i	i|	i	i	i	1
	5.	0	4	7	8	9	10	11|	1	2	5	6	3	i	1	i	i	i	i	i|	i	i	i	i	1
	6.	0	1	5	8	9	10	11|	2	3	6	7	4	i	i	1	i	i	i	i|	i	i	i	i	1
	7.	0	1	2	6	9	10	11|	3	4	7	8	5	i	i	i	1	i	i	i|	i	i	i	i	1
	8.	0	1	2	3	7	10	11|	4	5	8	9	6	i	i	i	i	1	i	i|	i	i	i	i	1
	9.	0	1	2	3	4	8	11|	5	6	9	10	7	i	i	i	i	i	1	i|	i	i	i	i	1
	10.	0	1	2	3	4	5	9|	11	6	7	10	8	i	i	i	i	i	i	1|	i	i	i	i	1
	11.	0	1	2	3	4	5	6	10|	11	7	8	9	i	i	i	i	i	i	i	1|	i	i	i	1
	12.	0	1	2	3	4	5	6	7	11|	8	9	10	i	i	i	i	i	i	i	i	1|	i	i	1
Total de AUNFs necessários: 14.257 (445 Kb)																								
Experimento 6	1.	10	11|	0	1	2	3	4	5	6	7	8	9	1	1|	i	i	i	i	i	i	i	i	i	i
	2.	2	3	0|	1	4	5	6	7	8	9	10	11	i	i	1|	1	i	i	i	i	i	i	i	i
	3.	0	3	4	1|	2	5	6	7	8	9	10	11	i	i	i	1|	1	i	i	i	i	i	i	i
	4.	0	1	4	5	2|	3	6	7	8	9	10	11	i	i	i	i	1|	1	i	i	i	i	i	i
	5.	1	2	5	6	3|	0	4	7	8	9	10	11	i	i	i	i	1|	i	1	i	i	i	i	i
	6.	2	3	6	7	4|	0	1	5	8	9	10	11	i	i	i	i	1|	i	i	1	i	i	i	i
	7.	3	4	7	8	5|	0	1	2	6	9	10	11	i	i	i	i	1|	i	i	i	1	i	i	i
	8.	4	5	8	9	6|	0	1	2	3	7	10	11	i	i	i	i	1|	i	i	i	i	1	i	i
	9.	5	6	9	10	7|	0	1	2	3	4	8	11	i	i	i	i	1|	i	i	i	i	i	1	i
	10.	11	6	7	10	8|	0	1	2	3	4	5	9	i	i	i	i	1|	i	i	i	i	i	i	1
	11.	11	7	8	9|	0	1	2	3	4	5	6	10	i	i	i	1|	i	i	i	i	i	i	i	1
	12.	8	9	10|	0	1	2	3	4	5	6	7	11	i	i	1|	i	i	i	i	i	i	i	i	1
Total de AUNFs necessários: 568 (17 Kb)																								
Experimento 7	1.	10	11|	0	1	2	3	4	5	6	7	8	9	1	1|	i	i	i	i	i	i	i	i	i	i
	2.	2	3	0	1|	4	5	6	7	8	9	10	11	i	i	1	1|	i	i	i	i	i	i	i	i
	3.	0	3	4	1	2|	5	6	7	8	9	10	11	i	i	i	1	1|	i	i	i	i	i	i	i
	4.	0	1	4	5	2	3|	6	7	8	9	10	11	i	i	i	i	1	1|	i	i	i	i	i	i
	5.	1	2	5	6	3	4|	0	7	8	9	10	11	i	i	i	i	1	1|	i	i	i	i	i	i
	6.	2	3	6	7	4	5|	0	1	8	9	10	11	i	i	i	i	1	1|	i	i	i	i	i	i
	7.	3	4	7	8	5	6|	0	1	2	9	10	11	i	i	i	i	1	1|	i	i	i	i	i	i
	8.	4	5	8	9	6	7|	0	1	2	3	10	11	i	i	i	i	1	1|	i	i	i	i	i	i
	9.	5	6	9	10	7	8|	0	1	2	3	4	11	i	i	i	i	1	1|	i	i	i	i	i	i
	10.	11	6	7	10	8	9|	0	1	2	3	4	5	i	i	i	i	1	1|	i	i	i	i	i	i
	11.	11	7	8	9	10|	0	1	2	3	4	5	6	i	i	i	1	1|	i	i	i	i	i	i	i
	12.	8	9	10	11|	0	1	2	3	4	5	6	7	i	i	1	1|	i	i	i	i	i	i	i	i
Total de AUNFs necessários: 568 (17 Kb)																								
Experimento 8	1.	10	11|	0	1	2	3	4	5	6	7	8	9	1	1|	i	i	i	i	i	i	i	i	i	i
	2.	1|	4	5	6	7	8	9	10	11	2	3	0	1|	i	i	i	i	i	i	i	i	i	i	1
	3.	2|	5	6	7	8	9	10	11	0	3	4	1	1|	i	i	i	i	i	i	i	i	i	i	1
	4.	3|	6	7	8	9	10	11	0	1	4	5	2	1|	i	i	i	i	i	i	i	i	i	i	1
	5.	4|	0	7	8	9	10	11	1	2	5	6	3	1|	i	i	i	i	i	i	i	i	i	i	1
	6.	5|	0	1	8	9	10	11	2	3	6	7	4	1|	i	i	i	i	i	i	i	i	i	i	1
	7.	6|	0	1	2	9	10	11	3	4	7	8	5	1|	i	i	i	i	i	i	i	i	i	i	1
	8.	7|	0	1	2	3	10	11	4	5	8	9	6	1|	i	i	i	i	i	i	i	i	i	i	1
	9.	8|	0	1	2	3	4	11	5	6	9	10	7	1|	i	i	i	i	i	i	i	i	i	i	1
	10.	9|	0	1	2	3	4	5	11	6	7	10	8	1|	i	i	i	i	i	i	i	i	i	i	1
	11.	10|	0	1	2	3	4	5	6	11	7	8	9	1|	i	i	i	i	i	i	i	i	i	i	1
	12.	11|	0	1	2	3	4	5	6	7	8	9	10	1|	i	i	i	i	i	i	i	i	i	i	1
Total de AUNFs necessários: 12 (≈ 0 Kb)																								

A tabela mostra que, para o caso das *Redes Wireless ad hoc*, ao utilizar descritores generalizados, matrizes do tipo identidade encontram-se na parte esparsa. Teoricamente, o melhor jeito de se tratar cada termo tensorial é permitir apenas identidades e a matriz que contém a definição do elemento funcional na parte estruturada, gastando apenas 12 AUNFs em memória. Em outros experimentos, o número de AUNFs necessários é maior, tais como 568 para os Experimentos 7 e 8 e 14.257 para o Experimento 5.

Descritores generalizados não privilegiam a avaliação de funções na parte esparsa, pois forçam muitas vezes que matrizes do tipo identidade permaneçam também neste lado, dado que podem ser parâmetros da função, o que onera a memória e não altera o valor dos escalares que correspondem aos AUNFs. Ao converter o descritor generalizado para clássico, os eventos com taxas funcionais viram eventos sincronizantes. Neste caso, pode-se escolher usar a estratégia de re-estruturar o termo tensorial de forma que apenas matrizes identidade permaneçam na parte estruturada e compor AUNFs com os elementos não-nulos na parte esparsa. Esta estratégia não prevê a existência de matrizes identidade na parte esparsa, o que não acontecia no caso do descritor generalizado. Logo, pode-se verificar o custo computacional necessário para avaliação de funções dentro do modelo das *Redes Wireless ad hoc* e comparar com o mapeamento para descritor clássico. A seguir, mostra-se como as funções existentes no modelo foram convertidas para eventos sincronizantes.

7.3.3 Descritor Markoviano clássico

A definição do modelo de *Redes Wireless ad hoc* com descritores generalizados mostrou-se oneroso para a utilização do Algoritmo *Split*, principalmente quando se verificam as tabelas de resultados para ambos os casos descritas na seção anterior (Tabela 7.4 e Tabela 7.5, correspondendo respectivamente aos resultados clássicos e generalizados). Deseja-se descobrir porque houve uma melhora considerável nos tempos de execução entre ambos os casos. A detecção das propriedades dos descritores clássicos será fundamental para entender como o Algoritmo *Split* pode ser usado de forma mais adequada para as classes de modelos que seguirem este tipo de descritores.

A Tabela 7.15 mostra as re-estruturações efetuadas no descritor para os Experimentos 1 e 3 bem como o total de elementos não-nulos nas matrizes de cada termo tensorial, numerado aqui de 1 até 12. No Experimento 1, as matrizes identidade foram tratadas na parte estruturada e o Experimento 3 não alterou a ordem natural das matrizes (ou seja, não permutou as matrizes), escolhendo o ponto de corte σ como sendo a última matriz constante (no caso a última não-identidade no termo tensorial). Este último experimento precisou de um número elevado de AUNFs (45.929) enquanto que o Experimento 1 precisou de um AUNF por termo tensorial (apenas 12), sendo o caso que apresentou melhor desempenho em tempo para o Algoritmo *Split* mesmo quando este é comparado ao caso do descritor original generalizado. A característica mais importante no descritor clássico é que todas as matrizes que não são identidades, são matrizes do tipo elemento, sendo extremamente esparsas. E ao serem enviadas para a parte esparsa (à esquerda do ponto de corte), produzem um número ínfimo de AUNFs, evitando, na totalidade, os cálculos necessários pela parte estruturada (índices e deslocamentos na estrutura tensorial).

Tabela 7.15: Permutações escolhidas e esparsidades dos termos para o caso *Redes Wireless ad hoc* com 12 nós nos Experimentos 1 e 3 – *Split* ATC.

	evt	Ordem dos autômatos e ponto de corte σ (|)	Esparsidade das matrizes e ponto de corte σ (|)
Experimento 1	1.	10 11| 0 1 2 3 4 5 6 7 8 9	1 1| i i i i i i i i i i
	2.	0 1 2 3| 4 5 6 7 8 9 10 11	1 1 1 1| i i i i i i i i
	3.	0 1 2 3 4| 5 6 7 8 9 10 11	1 1 1 1 1| i i i i i i i
	4.	0 1 2 3 4 5| 6 7 8 9 10 11	1 1 1 1 1 1| i i i i i i
	5.	1 2 3 4 5 6| 0 7 8 9 10 11	1 1 1 1 1 1| i i i i i i
	6.	2 3 4 5 6 7| 0 1 8 9 10 11	1 1 1 1 1 1| i i i i i i
	7.	3 4 5 6 7 8| 0 1 2 9 10 11	1 1 1 1 1 1| i i i i i i
	8.	4 5 6 7 8 9| 0 1 2 3 10 11	1 1 1 1 1 1| i i i i i i
	9.	5 6 7 8 9 10| 0 1 2 3 4 11	1 1 1 1 1 1| i i i i i i
	10.	6 7 8 9 10 11| 0 1 2 3 4 5	1 1 1 1 1 1| i i i i i i
	11.	7 8 9 10 11| 0 1 2 3 4 5 6	1 1 1 1 1| i i i i i i i
	12.	8 9 10 11| 0 1 2 3 4 5 6 7	1 1 1 1| i i i i i i i i
Total de AUNFs necessários: 12 (≈ 0 Kb)			
Experimento 3	1.	0 1 2 3 4 5 6 7 8 9 10 11|	i i i i i i i i i 1 1|
	2.	0 1 2 3| 4 5 6 7 8 9 10 11	1 1 1 1| i i i i i i i i
	3.	0 1 2 3 4| 5 6 7 8 9 10 11	1 1 1 1 1| i i i i i i i
	4.	0 1 2 3 4 5| 6 7 8 9 10 11	1 1 1 1 1 1| i i i i i i
	5.	0 1 2 3 4 5 6| 7 8 9 10 11	i 1 1 1 1 1 1| i i i i i
	6.	0 1 2 3 4 5 6 7| 8 9 10 11	i i 1 1 1 1 1 1| i i i i
	7.	0 1 2 3 4 5 6 7 8| 9 10 11	i i i 1 1 1 1 1 1| i i i
	8.	0 1 2 3 4 5 6 7 8 9| 10 11	i i i i 1 1 1 1 1 1| i i
	9.	0 1 2 3 4 5 6 7 8 9 10| 11	i i i i i 1 1 1 1 1 1| i
	10.	0 1 2 3 4 5 6 7 8 9 10 11|	i i i i i i 1 1 1 1 1 1|
	11.	0 1 2 3 4 5 6 7 8 9 10 11|	i i i i i i i 1 1 1 1 1|
	12.	0 1 2 3 4 5 6 7 8 9 10 11|	i i i i i i i i 1 1 1 1|
Total de AUNFs necessários: 45.929 (1.435 Kb)			

Para este caso específico e, consequentemente, classes de modelos que seguirem estas características, é adequado converter as taxas funcionais para elementos constantes nas matrizes, pois não implica em um aumento do número de eventos sincronizantes para estes casos (como é esperado ao efetuar-se uma tradução ATG para ATC). Esse modelo utiliza pouca quantidade de memória adicional em relação ao Algoritmo *Shuffle* (um total de 12 AUNFs, ou seja, $\approx$0 Kb) e executa o processo da MVD $\approx$6 vezes mais rápido que o modelo equivalente com mesmo tipo de descritor e $\approx$1,5 vezes melhor ao ser comparado com o modelo que tem um descritor generalizado (comparando apenas o Algoritmo *Split*).

7.3.4 Cálculo do número de multiplicações e avaliações de funções

O processo da MVD pode ser realizado de diferentes formas, como explicado ao longo do trabalho. A escolha de cada algoritmo, *Esparso*, *Shuffle* ou *Split* está relacionado com a memória extra a ser gasta na execução. Cada diferente maneira implica em diferentes cálculos para determinar o número de multiplicações em ponto flutuante. Esta seção discute estes cálculos e a as influências dos descritores funcionais, dada as restrições existentes quanto aos parâmetros das funções eas relações com a memória.

O Algoritmo *Shuffle* é eficiente em memória e trata com descritores clássicos e generalizados de forma otimizada. Entretanto, para o Algoritmo *Split*, as quantidades de memória extra necessária para guardar partes do descritor estão diretamente relacionadas à escolha do ponto de corte σ para cada termo tensorial. Para o *Split*, a descrição de cada taxa funcional é crucial no processo de MVD pois, dependendo do número de parâmetros e da maneira pela qual foi construída, impactará em um maior ou menor nível de memória extra a ser armazenada em estruturas de dados internas[3].

Começando a análise pelas avaliações de funções, mostradas pelos Experimentos $5, 6, 7$ e 8. Ao avaliar elementos na parte esparsa, está-se, na verdade, convertendo-se um descritor generalizado para clássico em tempo de execução do método (no caso, apenas na primeira execução do método iterativo, pois existe a fase de pré-cálculo dos AUNFs necessários). Entretanto, ao escolher avaliar funções na parte esparsa, dependendo da função, faz com que matrizes identidade sejam movidas para esta parte. Como é necessário multiplicar toda a listagem dos AUNFs para cada evento por um bloco potencialmente grande ($nright_\sigma$, dependendo do ponto de corte), gasta-se um tempo adicional considerável multiplicando-se escalares desnecessariamente pois, no Algoritmo *Split*, matrizes identidade na parte esparsa oneram a memória (e, em alguns casos, podem inviabilizar a execução do Algoritmo *Split*).

O problema de se avaliar funções na parte estruturada reside no fato que estas chamadas internas são realizadas diversas vezes, dependendo do número de iterações que são realizadas. Dependendo da função, estas avaliações são custosas do ponto de vista computacional. Observa-se então a existência de um *tradeoff* em descritores generalizados quanto ao local onde serão avaliadas as funções e a memória a ser gasta a partir desta escolha.

7.4 Considerações sobre os resultados

A partir dos resultados obtidos percebeu-se as vantagens e desvantagens de cada estratégia utilizada para cada tipo de descritor, clássico ou generalizado. Para alguns modelos, converter descritores ATC em ATG corresponde a ganhos significativos de desempenho, dependendo do caso. As tabelas de resultados apresentadas nesta tese mostraram as execuções dos Algoritmos *Shuffle* e *Split* para uma iteração do *Método da Potência*.

A Tabela 7.16 mostra os ganhos observados ao se acelerar o processo de MVD. A tabela exibe, para os modelos, o número de iterações necessárias para que o método iterativo atinja a convergência, para um conjunto de parâmetros (taxas) pré-estabelecidos. A partir do tempo para uma iteração obtido nas tabelas de resultados anteriores (apresentadas na coluna T_1), multiplicou-se este valor pelo total de iterações (coluna *#iter.*), obtendo-se o tempo total necessário para que se tenham índices confiáveis de desempenho.

A coluna *Tempo para Solução* mostra o tempo com escala variável, ou seja, em minutos (min), horas (h) ou dias) que cada método gastou para a convergência, com a escolha da estratégia que

[3]Como discutido ao longo do trabalho, um aumento de memória significativo será observado se uma função for definida com diversos parâmetros que traduzem-se em matrizes identidade no termo tensorial correspondente ao evento, ao mesmo tempo que escolhe-se a estratégia de avaliar funções na parte esparsa, por exemplo.

privilegia as avaliações de funções na parte esparsa e mantém apenas as matrizes identidade na parte estruturada (à direita do corte) de cada termo tensorial.

A coluna *Memória* mostra, em Kb, a memória adicional necessária para o *Split*. Cabe ressaltar que este possui a mesma limitação da ferramenta em termos de memória para armazenar o PSS existente, sendo o valor equivalente a ≈ 65 milhões de estados. Este é o limite existente para ambos os métodos em máquinas com 4 Gb de memória principal. Neste caso, observa-se que para o caso de 10 escravos (caso *slaves10*), o *Split* alocou por volta de 79 Mb adicionais. Entretanto, para 13 escravos, o modelo seria composto por outros três autômatos de três estados, elevando o PSS para $|\mathcal{X}| = 7.263.027 \times 3^3 \approx 180$ milhões de estados. O caso dos 13 escravos não pode ser executado por nenhum dos algoritmos nesta versão da ferramenta, entretanto, existem técnicas tais como simulação [FER08] para trabalhar com espaços de estados que ultrapassam a memória principal, e abordá-las está fora do escopo deste trabalho.

Tabela 7.16: Comparação entre os tempos obtidos até convergência (ou análise transiente) do método.

Modelo	PSS	#iter.	Tempo (s)				Mem. (Kb)	Tempo para Solução	
			Shuffle		*Split*			*Shuffle*	*Split*
			T_1	T_t	T_1	T_t			
ad14c	2.125.764	78.029	$2,1787$	$1,70E5$	$0,3940$	$3,07E4$	≈0	≈1,96 dias	≈8,54 h
ad14f	2.125.764	78.029	$6,8353$	$5,33E5$	$0,6434$	$5,02E4$	22	≈6,17 dias	≈13,94 h
ad16c	19.131.876	93.126	$22,6656$	$2,11E6$	$4,0235$	$3,75E5$	≈0	≈24,43 dias	≈4,33 dias
ad16f	19.131.876	93.126	$79,3119$	$7,39E6$	$6,6018$	$6,15E5$	27	≈85,48 dias	≈7,11 dias
asp12c	342.732	766	$0,2331$	$1,79E2$	$0,2478$	$1,90E2$	657	≈2,97 min	≈3,16 min
asp14c	399.854	750	$0,3130$	$2,35E2$	$0,3309$	$2,48E2$	891	≈3,91 min	≈4,14 min
fas19c	524.288	396	$1,2254$	$4,85E2$	$0,1725$	$6,83E1$	≈0	≈8,08 min	≈1,13 min
fas20c	1.048.576	416	$2,8037$	$1,17E3$	$0,3660$	$1,52E2$	≈0	≈19,44 min	≈2,53 min
phil15c	14.348.907	1.002	$16,6351$	$1,67E4$	$6,2543$	$6,27E3$	≈1	≈4,63 h	≈1,74 h
phil16c	43.046.721	1.071	$77,4296$	$8,29E4$	$19,7111$	$2,11E4$	≈1	≈23,04 h	≈5,86 h
NUMA6c	1.166.400	10.000	$6,5213$	$6,52E4$	$0,3793$	$3,79E3$	5	≈18,11 h	≈1,05 h
NUMA8c	55.427.328	10.000	$730,1900$	$7,30E6$	$25,7019$	$2,57E5$	9	≈84,51 dias	≈2,97 dias
oqn3	10.140.651	10.000	$3,1923$	$3,19E4$	$1,8723$	$1,87E4$	3.132	≈8,87 h	≈5,20 h
oqn4	15.813.251	10.000	$5,1428$	$5,14E4$	$2,9891$	$2,99E4$	3.914	≈14,29 h	≈8,30 h
rs15_15c	524.288	153	$0,6812$	$1,04E2$	$0,3448$	$5,28E1$	14	≈1,73 min	≈0,88 min
rs20_25c	27.262.976	180	$88,0908$	$1,59E4$	$24,7120$	$4,45E3$	31	≈4,40 h	≈1,23 h
slaves08c	807.003	3.094	$1,0292$	$3,18E3$	$0,3915$	$1,21E3$	8.756	≈53,07 min	≈20,18 min
slaves08f	807.003	3.094	$1,2023$	$3,72E3$	$0,3985$	$1,23E3$	8.836	≈61,90 min	≈20,55 min
slaves10c	7.263.027	2.696	$11,3418$	$3,06E4$	$4,1841$	$1,13E4$	76.974	≈8,49 h	≈3,13 h
slaves10f	7.263.027	2.696	$13,3783$	$3,61E4$	$4,2506$	$1,15E4$	79.372	≈10,01 h	≈3,18 h
webserver	349.920	599	$0,7678$	$4,60E2$	$0,1087$	$6,51E1$	≈1	≈7,66 min	≈1,08 min
workcell	320.112	2.162	$0,1707$	$3,69E2$	$0,0357$	$7,72E1$	≈0	≈6,15 min	≈1,28 min
							Total	**≈206,50 dias**	**≈16,62 dias**

O número de iterações necessárias para cada modelo é dependente das taxas que são utilizadas. A Tabela 7.16 mostra numericamente a vantagem em se adotar a execução do Algoritmo *Split* em comparação com a abordagem anterior existente descrito pelo Algoritmo *Shuffle*. Ao se acelerar a solução da MVD a cada iteração esse ganho é propagado por todas as iterações necessárias, resultando em um ganho em tempo ao se avaliar a solução como um todo. O Algoritmo *Split*

permite obter resultados numéricos de forma mais rápida e, consequentemente, antecipando a fase de análise dos modelos estudados.

Excetuando-se o caso *Mestre-Escravo*, que precisou ≈80 Mb para 10 escravos, o restante dos modelos obteve um ganho expressivo em termos de tempo para solução dos exemplos onde a memória a ser potencialmente gasta restringiu-se a limites aceitáveis, mostrando que o Algoritmo *Split* é uma alternativa eficaz de MVD. Observa-se que diferentes modelos, o método ainda permanece sendo eficaz no armazenamento de dados extra em memória, como mostrado pelos dados da Tabela 7.16. O ganho médio observado ao utilizar o Algoritmo *Split* foi de $12{,}8$ vezes para os modelos escolhidos. Para os modelos NUMA e OQN, escolheu-se um total de 10.000 iterações arbitrariamente para efeitos de cálculos, pois os modelos possuem apenas solução transiente.

Entretanto, os modelos executados dificultam a captura de resultados para determinar o custo real de se realizar as permutações. Permutar as matrizes dos termos tensoriais é indissociável ao Algoritmo *Split*, mais evidente para o caso de descritores clássicos. Entretanto, em pelo menos um caso de modelo as permutações não são necessárias e mesmo assim garante-se que apenas matrizes do tipo identidade estejam na parte estruturada, condição necessária para compor uma base de comparação entre uma versão com e sem re-estruturações. O modelo em questão é o *First Available Server* (FAS), que contém eventos que não alteram a ordem original ao mesmo tempo que possibilitam cortar o termo tensorial na última matriz constante, deixando apenas matrizes identidade na outra parte.

Para calcular o custo de permutação, executou-se o modelo FAS, com a estratégia onde todas as matrizes identidade encontram-se na parte estruturada. Os testes foram realizados com implementações com e sem permutação calculando-se o custo envolvido em termos de tempo. Escolheu-se rodar um descritor clássico pois, para descritores generalizados, as taxas funcionais contém parâmetros que eventualmente forçam permutações, para que os elementos sejam avaliados coerentemente. A Tabela 7.17 mostra o modelo para $18, 19, 20, 21, 22$ e 23 servidores, onde cada coluna mostra a versão do Algoritmo *Split* executada, com ou sem reordem. Como nos experimentos anteriores, os métodos foram executados 50 vezes onde capturou-se a média para 25 iterações e computou-se o intervalo de confiança de 95%.

Tabela 7.17: Avaliação do custo de efetuar permutações nos termos tensoriais.

		Tempos (s) - 1 iteração			
Modelo	PSS	*Split* **sem** (S) permutação	*Split* **com** (C) permutação	Diferença C – S (s)	Aumento S → C
fas18c	262.144	0,0758 ± 0,0001	0,0814 ± 0,0007	0,0056	7,39%
fas19c	524.288	0,1652 ± 0,0001	0,1744 ± 0,0001	0,0092	5,57%
fas20c	1.048.576	0,3509 ± 0,0004	0,3667 ± 0,0004	0,0158	4,50%
fas21c	2.097.152	0,7336 ± 0,0006	0,7707 ± 0,0013	0,0371	5,06%
fas22c	4.194.304	1,5411 ± 0,0012	1,6071 ± 0,0075	0,0660	4,28%
fas23c	8.388.608	3,2490 ± 0,0020	3,3525 ± 0,0030	0,1035	3,19%

Apesar do aumento constatado da versão sem permutação para a com permutação, para este modelo, o impacto das chamadas internas diminui à medida que o modelo aumenta o número de servidores. O modelo FAS é o único caso onde é possível realizar estes cálculos de custos computacionais. Maiores informações sobre os cortes e a ordem de tratamento das matrizes podem ser vistas no Apêndice A.1.3 e seu experimento equivalente (corte na última constante) no Apêndice A.2.3. Qualquer outro modelo, mesmo que não realize reordens nas matrizes dos termos tensoriais mas, caso a parte estruturada tenha uma matriz constante ou elemento, o Algoritmo *Shuffle* será executado e influenciará negativamente nos resultados. Essas chamadas influenciarão o custo computacional envolvido na MVD, não podendo ser comparado com uma versão do Algoritmo *Split* sem permutações.

7.5 Determinação do ponto de corte σ para o Algoritmo *Split*

Os exemplos avaliados neste capítulo mostraram as vantagens e as características dos termos tensoriais que devem ser observadas para determinar onde dividi-los e reposicionar as matrizes para obtenção de um melhor tempo de execução. A seguir, é discutida uma proposta para a escolha deste ponto de corte, observando a memória extra que será potencialmente gasta, refletindo no tempo de solução.

As equações para o custo computacional de multiplicação dos termos tensoriais, aliado aos resultados obtidos na Seção 7.1 permitem que seja proposto um algoritmo para utilização do Algoritmo *Split* em descritores clássicos e generalizados. Este algoritmo leva em consideração o custo em termos de operações de ponto flutuante requeridas pelo algoritmo, bem como a memória adicional necessária para armazenamento dos AUNFs da parte esparsa.

Os resultados obtidos permitiram análises sobre o processo da MVD para descritores clássicos e generalizados como, por exemplo:

1. avaliar elementos funcionais apenas uma vez na primeira iteração é melhor do que avaliá-las muitas vezes, por muitas iterações;

2. matrizes constantes e matrizes elemento são melhor tratadas se estiverem posicionadas na parte esparsa, dadas as restrições para quando taxas funcionais estão presentes (os parâmetros devem estar à esquerda da matriz que contém a função);

3. uma esparsidade baixa do termo tensorial (o número de elementos não-nulos existentes) implica em um número de AUNFs igualmente baixo;

4. matrizes identidade são melhor tratadas se pertencerem à parte estruturada (respeitadas as restrições existentes), pois na parte esparsa, geram um número de AUNFs maior que antes desta reorganização e não alteram o valor do escalar;

5. termos altamentes dependentes já são adequadamente tratados pelo Algoritmo *Shuffle*. Para estes casos, talvez a memória extra necessária inviabilize o método;

6. para o Algoritmo *Split* ser mais eficaz, os modelos devem possuir autômatos que sincronizam suas atividades com poucos outros autômatos, forçando a existência de muitas matrizes identidade em cada termo tensorial.

O Algoritmo 7.1 recebe como entrada um termo tensorial e decide o ponto de corte a ser utilizado a partir das características das matrizes (dimensões, identidades, quantidade de elementos não-nulos) que o compõe cada termo tensorial. O algoritmo funciona da seguinte forma: a partir da descoberta do tipo do termo tensorial, que pode ser uma de três possibilidades, constante, parcialmente dependente ou totalmente dependente, é realizado o processo de descoberta do ponto de corte σ.

Algoritmo 7.1: Determinação do ponto de corte σ do termo tensorial τ.

1: $M \leftarrow$ {memória disponível}
2: $A \leftarrow$ {memória da tabela de AUNFs, assume-se que já foram calculados previamente}
3: $tipo \leftarrow T(\tau)$ {descobre o tipo do termo tensorial τ}
4: **if** $tipo =$ constante **then**
5: $i \leftarrow transforma(\tau)$ {identidades para o fim, salva índice da última matriz constante}
6: $A \leftarrow \mathcal{E}(i,\tau)$ {a função $\mathcal{E}$ descobre o total de AUNFs para i e τ}
7: **if** $A < M$ **then** {se existe memória suficiente para a operação}
8: $\sigma \leftarrow i$ {o ponto de corte σ recebe o índice i}
9: **else**
10: {encontra um outro ponto de corte σ utilizando menos memória ($+$ estruturação)}
11: **end if**
12: **else if** $tipo =$parcialmente dependente **then**
13: $c \leftarrow constante(\tau)$ {verifica a existência de matrizes constantes}
14: $i \leftarrow transforma(\tau, c)$ {identidades para o fim, a partir da última matriz constante}
15: $A \leftarrow \mathcal{E}(i,\tau)$ {a função $\mathcal{E}$ descobre o total de AUNFs para i e τ}
16: **if** $A < M$ **then**
17: $j \leftarrow depend(\tau)$ {descobre o índice da última matriz dependente}
18: $\sigma \leftarrow c + j$ {determina σ da última matriz constante $+$ última dependente}
19: **else**
20: {encontra um outro ponto de corte σ utilizando menos memória ($+$ estruturação)}
21: **end if**
22: **else if** $tipo =$ totalmente dependente **then**
23: $u \leftarrow ultima(\tau)$ {a função *ultima* descobre o índice da última matriz do termo}
24: $A \leftarrow \mathcal{E}(u,\tau)$ {descobre o total de AUNFs para i e τ}
25: **if** $A < M$ **then** {considerar a chamada de método esparso puro}
26: $\sigma \leftarrow u$
27: **else** {utiliza a abordagem puramente estruturada}
28: $\sigma \leftarrow 0$
29: **end if**
30: **end if**
31: $retorna\ \sigma$ {retorna o valor de σ para τ calculado}

Caso o tipo de do termo τ for clássico, ocorre uma transformação no termo tensorial enviando as matrizes identidade para o final e armazenando-se esse índice na variável i. Caso haja memória

suficiente para essa operação, a transformação é validada e σ recebe o valor do índice i. Já no caso do termo ser parcialmente dependente, tem-se a verificação da ocorrência de matrizes elemento (ou constantes) e o envio de matrizes para a parte esparsa do termo tensorial, exceto identidades.

O valor de σ para esse caso é o índice da última matriz dependente depois desta transformação. Mas se o tipo for altamente dependente seja em sincronizações ou quantidade de parâmetros de elementos funcionais, a melhor possibilidade para lidar com esse termo é adotar a abordagem totalmente estruturada, pois inclusive pode-se ter dependências cíclicas e funções especificamente que serão tratadas de forma adequada pelo Algoritmo *Shuffle*.

Termos tensoriais com matrizes mais densas farão com que seja calculado um número de AUNFs que talvez inviabilize o método (dependendo do corte), e a melhor abordagem a seguir é dividir o termo em um ponto onde a memória a ser gasta é suficientemente calculada para abrigar um número considerável de elementos. Em casos altamente dependentes em termos de funções, a melhor alternativa é cortar em σ_0, ou seja, trata o termo de forma estruturada. A experiência em modelagem SAN pressupõe poucos casos com matrizes plenas ou altamente densas em um termo tensorial. Estas análises finalizam a parte de proposição de um algoritmo para dividir um termo tensorial. A seguir, é feita uma discussão sobre os experimentos e os resultados obtidos.

7.6 Discussão

Os resultados obtidos mostraram os casos em que o Algoritmo *Split* é mais eficaz: (i) termos tensoriais esparsos estão presentes, no melhor caso, com apenas *um* ou poucos elementos não-nulos, (ii) termos com alta incidência de matrizes identidade simbolizando que o evento sincroniza poucas atividades, (iii) efetuam-se permutações na ordem original e, (iv) para descritores generalizados, as avaliações de elementos funcionais são efetuadas quando os AUNFs são calculados apenas uma vez no início do método. O melhor caso pode ser comprovado através do modelo das *Redes Wireless ad hoc*, onde foi preciso armazenar N escalares na tabela de AUNFs, correspondendo ao número de termos existentes, ou seja, um AUNF por termo tensorial. Os ganhos em tempo para o caso clássico foram da ordem de 12 vezes em relação à abordagem do Algoritmo *Shuffle*.

O uso de primitivas que contemplam transições funcionais é importante em modelagem estocástica de sistemas. Entretanto, nas soluções por métodos iterativos não se justifica a avaliação frequente de elementos funcionais a cada execução do método de MVD. Estes elementos funcionais serão avaliados segundo uma ordem específica. Isso vai ao encontro da proposta sugerida nesta tese de utilizar estruturas de dados auxiliares para guardar as avaliação das funções apenas uma vez no início do método iterativo, convertendo os descritores generalizados em clássicos em tempo de execução da ferramenta. No entanto, para entender o processo interno do algoritmo na sua essência, é necessário observar os detalhes do descritor Markoviano em si. Uma vez que os eventos locais são somas simples de matrizes, são os eventos sincronizantes, com taxas constantes ou funcionais, que determinam o comportamento do Algoritmo *Split* em termos de custos de memória e tempo de solução.

Cabe ressaltar ainda que o Algoritmo *Split*, quando permuta as identidades para a parte estruturada e trabalha apenas com os AUNFs que correspondem às taxas dos eventos sincronizantes, não são gerados fatores normais na parte estruturada. Apenas é necessário multiplicar o escalar (s) do AUNF em posições chave ($base_{in}$ e $base_{out}$) por um vetor de tamanho $nright_{\sigma}$ (conforme Algoritmo 4.3.). O custo gasto para reordenar o termo tensorial, calcular índices auxiliares para saltar na estrutura permutada e utilizar esses índices ao longo do método de MVD é negligenciável ao utilizar o Algoritmo *Split*, como constatado em todos os experimentos da Seção 7.2. Verificou-se que a maioria dos modelos considerados executaram mais rápido que o Algoritmo *Shuffle* para ATC e ATG, onde a tabela com os AUNFs ficou dentro de limites aceitáveis. O único limitador para solução de modelos estruturados em SAN continua sendo o tamanho do vetor estacionário de probabilidade que corresponde ao PSS. Dada a relativa memória utilizada para salvar os AUNFs requerida inclusive quando o PSS é elevado, pode-se afirmar que o método somente não executará para os mesmos limites que são impostos ao *Shuffle*, estimado atualmente em 65 milhões de estados.

Um aspecto que não foi estudado até o presente momento é o efeito das dimensões das matrizes (principalmente quando não correspondem à identidades) na solução do modelo. Esta variável pode ser melhor estudada no modelo *Mestre-Escravo*, que possui um autômato modelado como um *Buffer*, ou seja, uma região temporária de memória em uma determinada problemática (este tipo de modelagem é bastante utilizada em sistemas paralelos e distribuídos, sendo útil para verificar quando estas estruturas chegam aos seus limites ou quando estão subutilizadas). Esses estudos podem alterar significativamente o algoritmo de corte de termo tensorial pois, na parte estruturada, na parte da geração dos fatores normais, um grande bloco será descartado quando a matriz em questão possuir uma dimensão elevada. Maiores estudos nesse sentido devem ser conduzidos para um melhor entendimento sobre o comportamento do método quando matrizes deste porte estão presentes.

O tamanho da lista de AUNFs necessárias para cada modelo, levando-se em consideração todos os eventos, dita tanto a memória extra a ser gasta quanto a complexidade em termos de multiplicações em ponto flutuante requeridas (seu tamanho indica os produtos necessários no vetor $nright_{\sigma}$). Um modelo que possua eventos sincronizantes que não afetem muitos outros autômatos possuirão uma lista menor de escalares para serem multiplicados no vetor. Estas discussões finalizam o capítulo de resultados. A seguir, as considerações finais e as perspectivas futuras.

8. CONSIDERAÇÕES FINAIS E PERSPECTIVAS

Esta tese descreveu os principais formalismos estruturados de solução utilizados no contexto de avaliação de desempenho de sistemas, mostrando suas principais vantagens e desvantagens ao serem adotados para diferentes problemáticas da realidade. A partir destas descrições, foram estudados mecanismos para composição de modelos mais abstratos, capturando as características fundamentais do funcionamento dos sistemas. Estas descrições são utilizadas para a construção de descritores Markovianos (ou Kronecker) que ao serem avaliados os operadores da Álgebra Tensorial, representam o Gerador Infinitesimal da Cadeia de Markov correspondente ao sistema em questão.

Uma vez que o modelo foi transposto para um formato tensorial e, por conseguinte, associado a um descritor Markoviano, existem diferentes algoritmos para multiplicar um vetor de probabilidade por tais estruturas, sendo os mais importantes o Algoritmo *Esparso*, o Algoritmo *Shuffle* e o Algoritmo *Split*. Tais algoritmos fornecem formas únicas de tratamento para a MVD e são documentados e utilizados para o cálculo computacional do vetor de probabilidades estacionário ou transiente que representa a probabilidade de permanência nos estados de um sistema. O cálculo de índices de desempenho sobre este vetor de probabilidades é utilizado para analisar o sistema, prever seus comportamentos ou antecipar problemas antes que estes sejam fisicamente instalados. Esta tese mostrou que, dados descritores decompostos em operações tensoriais que representam os eventos de um sistema, é possível alterar a ordem das matrizes e aproveitar as suas características internas para realizar menos operações de multiplicações, reduzindo o tempo gasto por iteração.

Estas escolhas foram implementadas em uma ferramenta chamada de GTAexpress. Este *software* foi executado para uma série de modelos de diferentes realidades, com taxas constantes ou funcionais, com o propósito de descobrir e validar maneiras pelas quais fosse possível organizar as matrizes dos termos tensoriais de descritores e dividi-las de uma forma razoável, objetivando a aceleração do processo de MVD. Na seção de resultados foram estudados diversos modelos através de experimentos com o Algoritmo *Split* onde o corte foi modificado para verificar os ganhos em termos de produtos efetuados entre os elementos das matrizes. Este capítulo discute os principais algoritmos de MVD na Seção 8.1. A seguir a Seção 8.2 lista as contribuições efetuadas nesta tese. O capítulo finaliza com uma discussão sobre os trabalhos futuros na Seção 8.3 e o epílogo na Seção 8.4.

8.1 Resumo

A seguir apresenta-se um detalhamento das principais características dos algoritmos de MVD, os desafios da utilização da abordagem *Split* e um resumo dos objetivos da tese:

1. Algoritmos de MVD:
    - **Esparso**: esta abordagem é custosa em termos de memória ao precisar armazenar a matriz de transição e, dependendo do modelo e da estrutura de acesso à matriz, é uma

alternativa *eficaz quanto ao tempo necessário* para efetuar o processo de multiplicação. Pode se tornar ineficaz em tempo e memória caso matrizes com elevadas dimensões possuam uma grande quantidade de elementos não-nulos;

- **Shuffle**: este algoritmo é considerado *eficiente em memória* ao armazenar o descritor e usar as propriedades da álgebra tensorial para efetuar a MVD, entretanto, devido aos inúmeros cálculos de índices que são necessários para acessar essa estrutura não trivial pode, dependendo do modelo, ser ineficaz em tempo para processar o descritor na sua totalidade;
- **Split**: esta abordagem é uma combinação das anteriores e se destaca por proporcionar uma maior *flexibilidade* ao considerar o processo de MVD, reordenando e dividindo cada termo tensorial de descritores clássicos e generalizados de acordo com as suas propriedades, otimizando a manipulação das matrizes para a MVD.

2. Desafios da abordagem do Algoritmo *Split*:
   - *matrizes do tipo identidade*: modelagens estruturadas baseada em SAN são naturalmente esparsas e, dependendo do caso, os descritores possuem diversas matrizes identidade;
   - *permutação do posicionamento original das matrizes que compõem o descritor*: salvo o caso generalizado, onde existe a limitação dos parâmetros das funções estarem situados à esquerda da definição da função, qualquer ordenação das matrizes internas é permitida;
   - *flexibilidade no gasto de memória*: a flexibilidade do Split permite que as matrizes sejam permutadas em uma ordem que seja escolhida de forma a refletir a memória a ser gasta pelo método. Esse fator pode ser importante para a aplicação em abordagens paralelas, ao permitir o balanceamento de carga de trabalhos entre os diferentes processadores de sistemas paralelos ou distribuídos;
   - *operação interna em descritores generalizados*: dadas as restrições das avaliações dos elementos funcionais deseja-se mapear as consequências de se tratar matrizes deste tipo nos descritores e o seu impacto na memória extra necessária para operação do método.

3. Resumo da tese:
   - mapeamento da problemática para determinar pontos de corte no Algoritmo *Split* objetivando a melhoria do tempo de execução do processo de MVD. Ao reduzir o tempo necessário para uma única iteração do método de convergência utilizado, estes ganhos são replicados de acordo com o total de iterações necessárias;
   - análise das características de descritores clássicos e generalizados, observando o tipo de cada matriz e sua influência no método, apresentando a complexidade para cada re-estruturação;
   - enumeração de experimentos relevantes para descoberta das propriedades existentes utilizando os modelos disponíveis na literatura (e alguns clássicos tais como o problema

do *Compartilhamento de Recursos* ou *Jantar dos Filósofos*) bem como as estratégias passíveis de execução dada a flexibilidade do Algoritmo Split;

- implementação de uma ferramenta contendo o Algoritmo Split para descritores clássicos e generalizados, adaptável ao modelo estudado (o *software* permite a configuração da estratégia para execução);
- discussão dos principais formalismos para avaliação de desempenho e proposição de formas de tradução entre formalismos estruturados utilizando premissas para abstração e composição de sistemas.

8.2 Contribuições

Um dos desafios na modelagem estocástica de sistemas é capturar as principais características e semântica operacional de problemáticas, descrevendo-as através de um modelo em um formalismo. Entretanto, apenas representá-lo através de primitivas de mapeamento que demonstrem seu sistema de transições é insuficiente para uma análise completa. As diferentes modelagens de sistemas devem ser usadas em conjunção com a sua *solução*, ou seja, através da descoberta dos índices calculados a partir do sistema de equações que é produzido. Estas informações, quando submetidas a rigorosas análises por parte dos modeladores, fornecem índices computacionais de desempenho que atestam a operação do sistema para um determinado conjunto de taxas escolhidas.

O uso de descritores baseados em tensores é justificado pelo fato de não ser necessário o armazenamento da matriz de transição que contém as taxas entre os estados de um modelo. Através de propriedades da Álgebra Tensorial, opera-se implicitamente sobre esta matriz realizando-se a MVD sem precisar gerar esta matriz de tamanho potencialmente grande. Os algoritmos *Esparso* e *Shuffle* presentes na literatura abordam a técnica de maneira distinta: enquanto o primeiro necessita muita memória e poucos cálculos para encontrar os elementos da matriz, o segundo é eficiente em memória mas são necessários muitos cálculos de índices para acesso aos elementos.

Portanto, a definição do Algoritmo *Split*, ao operar sobre as vantagens e desvantagens conhecidas dos algoritmos *Esparso* e *Shuffle*, apresenta um compromisso razoável para tratamento dos termos tensoriais de forma individualizada controlando a memória a ser gasta. Dadas as características das modelagens, observou-se também que, dependendo do descritor, o *Split* não necessita de quantidades elevadas de memória adicional, o que amplia os casos em que acelera o processo da MVD.

Ao utilizar diferentes estratégias de corte para termos tensoriais, constatou-se que algumas classes de modelos não gastavam quantidades excessivas de memória e aceleravam o processo da MVD. Ao estudar tais classes mais minuciosamente, observou-se que estas apresentavam características únicas tais como baixa densidade (poucos elementos não nulos dentro de cada matriz do termo tensorial) e fraco envolvimento entre as demais entidades do mesmo termo (o que foi definido como possuidor de baixo Grau de Dependência Constante ou Generalizado, conforme Seção 5.1). Esta métrica mostrou ser crucial para inferir onde será fixado o corte do termo tensorial e onde as funções (caso presentes) serão avaliadas, convertendo ou não os descritores ATG em ATC em tempo de execução.

A Tabela 8.1 mostra as possibilidades de re-estruturações para descritores clássicos e generalizados. A opção a ser escolhida depende do descritor, caso este seja clássico, a única variável a observar é o grau de interação ou interferência dos eventos do modelo. Já descritores generalizados dependem da quantidade de parâmetros e da complexidade de cada uma das funções criadas pelos usuários. Este fator impactará na quantidade de vezes que avaliações funcionais são realizadas, tanto no Algoritmo *Shuffle* como no Algoritmo *Split*. Como mencionado anteriormente, ao avaliar as funções na parte esparsa, estas chamadas são realizadas apenas uma vez no início do método iterativo enquanto que, na abordagem *Shuffle*, múltiplas avaliações são feitas (em função do número de iterações).

Tabela 8.1: Possibilidades de re-estruturações para descritores clássicos e generalizados.

Descritor	Grau de *interação* ou *interferência*	Recomendações para re-estruturações
ATC	muitos eventos envolvendo muitos autômatos	*balancear a memória adicional a ser gasta na parte esparsa, dependendo do tamanho do modelo*
	poucos eventos envolvendo muitos autômatos	*priorizar o envio das matrizes identidade para a parte estruturada, balanceando a memória extra a ser gasta de acordo com o grau de interferência existente*
	entre três e todos os autômatos	*observar a memória disponível e analisar cada evento para descobrir quantos e quais autômatos estes interferem*
	todos os eventos são interações entre apenas dois autômatos	*permutar matrizes identidade para a parte estruturada e dividir na última matriz constante*
	Detalhes quanto as funções	
ATG	muitos ou todos elementos são parâmetros e complexidade média	*dependendo da complexidade das funções, usar o Algoritmo Split, senão utilizar abordagem Shuffle*
	poucos elementos são parâmetros e complexidade baixa	*permutar matrizes funcionais e suas dependências ou traduzir descritor para clássico*
	parâmetros cíclicos	*encapsular os parâmetros cíclicos na parte esparsa e usar o Split ou usar diretamente o Algoritmo Shuffle*
	funções simples	*verificar impacto no número de eventos e esparsidade de matrizes ao traduzir descritor para clássico*
	funções complexas	*usar o Algoritmo Shuffle com otimizações funcionais*

Para o caso onde não existam taxas funcionais, os modelos com as características presentes acima, além de possuírem descritores com termos tensoriais esparsos, dependendo do modelo, possuem matrizes do tipo identidade que, ao serem tratadas pelo algoritmo *Shuffle* após a fixação

do ponto de corte e a descoberta dos índices de permutação, serão desconsideradas. Para o caso funcional, é necessário observar os parâmetros das funções ou então converter o modelo baseado em ATG para ATC explicitamente[1] [BRE05a]. Ao realizar esta tarefa de conversão, na maioria dos casos aumenta-se o número de eventos nos modelos, ou seja, criam-se eventos sincronizantes que forçam a verificação de estados em diferentes autômatos de acordo com os parâmetros utilizados.

A Tabela 8.2 mostra dois exemplos com descritores clássicos e generalizados baseados nos modelos de Redes *Wireless ad hoc* (*'ad'* com 14 e 16 nós) e modelo Mestre-Escravo (*'slaves'*, com 08 e 10 escravos). A tabela está separada em duas partes, sendo que a parte de cima mostra os tempos obtidos para uma iteração, para todos os experimentos (coluna *Exp.*) conduzidos no Capítulo 7 (Tabela 7.1) e, a parte de baixo mostra a memória adicional a ser gasta para execução de cada algoritmo. A partir da tabela verifica-se a necessidade (ou não) de conversão de modelos ATG para ATC, pois tem-se o tempo necessário para cada execução. Em termos de tempo, e comparando-se apenas a abordagem *Shuffle*, observando-se apenas o caso de 16 nós *wireless*, constata-se que é interessante converter o descritor para ATC, pois o tempo por iteração é reduzido de $79{,}31$ segundos (Exp. 9) para $22{,}67$ segundos (Exp. 4). O mesmo não ocorre em outros modelos, como para 10 escravos, que obteve tempos similares, ou seja, $13{,}38$ segundos e $11{,}34$ segundos para os experimentos 4 e 9.

Tabela 8.2: Comparação entre modelos *Redes Wireless ad hoc* e *Mestre-Escravo* para ATC e ATG.

Modelo	PSS	Tempos (s) – 1 iteração								
		ATC				*ATG*				
		Exp. 1 *Split*	Exp. 2 *Shuffle*	Exp. 3 *Split*	Exp. 4 *Shuffle*	Exp. 5 *Split*	Exp. 6 *Split*	Exp. 7 *Split*	Exp. 8 *Split*	Exp. 9 *Shuffle*
ad14	2.125.764	**0,39**	2,74	0,50	2,18	2,65	1,10	**0,64**	2,51	6,83
ad16	19.131.876	**4,02**	28,10	5,07	22,67	29,86	11,58	**6,60**	28,36	79,31
slaves08	807.003	**0,39**	1,47	0,63	1,03	0,55	**0,40**	0,52	0,55	1,20
slaves10	7.263.027	**4,18**	16,11	6,65	11,34	5,98	**4,25**	5,41	5,98	13,38
		Memória – AUNFs (Mb)								
ad14	2.125.764	≈**0**	–	12,61	–	4,09	0,02	0,02	≈**0**	–
ad16	19.131.876	≈**0**	–	113,52	–	38,44	0,03	0,03	≈**0**	–
slaves08	807.003	**8,55**	–	40,65	–	**8,23**	8,63	24,25	**8,23**	–
slaves10	7.263.027	**75,17**	–	364,35	–	**73,91**	77,51	218,07	**73,91**	–

Entretanto, ao se comparar a abordagem *Shuffle* com a *Split* e mesmo tipo de descritor, tem-se que para ATG (16 nós) são necessários $79{,}31$ segundos para rodar o método *Shuffle* (Exp. 9) e $6{,}60$ segundos para o *Split* (Exp. 7), um ganho em tempo superior a 10 vezes. Ao se converter este caso para descritores clássicos, o ganho verificado é comparativamente menor, de $22{,}67$ segundos (Exp. 4) para $4{,}02$ segundos (Exp. 1), ou seja, cinco vezes mais rápido. Para o caso *Mestre-Escravo* (10 escravos), tanto faz converter ou não o descritor, pois os tempos medidos foram $4{,}25$ (Exp. 6) e $4{,}18$ (Exp. 1), em contraste com o *Shuffle*, com $13{,}38$ (Exp. 9) e $11{,}34$ (Exp. 4).

[1] Este processo é dependente da complexidade da definição da função mapeada para o problema, como mencionado anteriormente.

A memória extra gasta com o Algoritmo *Split* para o modelo *Redes Wireless ad hoc* pode ser considerado desprezível (por exemplo, para uma máquina com 4Gb de memória RAM) para os menores tempos. Já para o modelo *Mestre-Escravo*, o gasto em memória também pode ser considerado baixo, da ordem de $\approx$74 Mb.

Cabe ressaltar que estes ganhos obtidos para uma única iteração são multiplicados pelo número total de iterações, ou seja, mesmo pequenos valores influenciam o tempo total necessário para finalização do processo completo de MVD. Estes exemplos evidenciam a flexibilidade presente na abordagem adotada pelo Algoritmo *Split*, permitindo diferentes formas de se re-estruturar os termos tensoriais para aceleração do processo de MVD. Esta característica pode também ser importante quando o algoritmo for proposto em versões paralelas, as quais tornarão necessária a descoberta de heurísticas de corte para balanceamento das tarefas atribuídas a cada unidade de processamento.

Compor um grande espaço de estados, descobrir seus estados atingíveis e armazená-lo não é mais um problema, visto as técnicas existentes para este propósito tais como gerações simbólicas ou baseadas em diagramas de matrizes [SAL09]. O problema agora é encontrar o vetor estacionário de probabilidade utilizando métodos numéricos de última geração que possibilitem e acelerem o processo de convergência a partir de diferentes estratégias, tais como aproximação da solução [BUC06] ou simulação perfeita [FER08, WEB09].

O presente trabalho alinha-se com este propósito, ao definir formas de se reduzir as operações necessárias ao tratar cada termo tensorial propondo novas formas de estruturá-lo com permutações e divisões e tratando cada parte de forma diferenciada. Este trabalho definiu as estratégias para a solução de descritores clássicos e generalizados ao mesmo tempo que comparou estas diferentes possibilidades para serem utilizadas pelo Algoritmo *Split*.

Com os resultados obtidos, descreveu-se um algoritmo que calcula um ponto de corte para termos tensoriais, observando a memória gasta e o tempo ganho, reorganizando-os para efetuar menos operações numéricas. A melhor classe de modelos possui alta esparsidade (baixa densidade) e altas quantidades de matrizes do tipo identidade.

A abordagem usada pelo Algoritmo *Shuffle* é eficiente em memória enquanto que a do Algoritmo *Esparso*, dependendo do caso, necessita de um espaço de armazenamento alto. Os dois casos tratam a problemática da memória de maneira distinta ao efetuar a MVD. Esta tese combinou as vantagens destas duas abordagens em uma alternativa híbrida, onde os gastos em memória são flexíveis.

O Algoritmo *Split* foi refinado para aumentar a variedade de modelos que resolve ao permitir que sejam construídos modelos com taxas constantes e generalizadas (ATC ou ATG). As pesquisas anteriores [CZE07, WEB09] desconheciam as implicações do algoritmo ao lidar com elementos funcionais. Foram elencadas as principais problemáticas ao se lidar com termos tensoriais generalizados, enumerando as formas de separar conjuntos de matrizes de acordo com características que reduzissem a complexidade computacional envolvida.

8.3 Perspectivas

A seguir serão detalhadas as perspectivas, classificadas da seguinte forma: conversão de descritores ATG em ATC, classes de modelos para que o Algoritmo *Split* seja melhor executado, métodos de paralelização e distribuição de tarefas, avanços em novas primitivas de modelagem e aplicações, criação de AUNFs com informações adicionais e aproximações da solução, melhor descritas a seguir.

a) conversão de descritores ATG em ATC

Como mencionado anteriormente, os melhores resultados do Algoritmo *Split* para descritores baseados em ATC são verificados quando em cada módulo de um sistema as interações e sincronismos são baixos, privilegiando a formação de um descritor com termos tensoriais esparsos e com muitas identidades. Entretanto, uma importante primitiva de modelagem é a utilização de taxas funcionais, ocorrendo em transições que verificam o estado de outros módulos de forma dinâmica. Dada a possibilidade de se converter um modelo definido em ATG para ATC, pouco se sabe sobre os casos em que esta tradução implicará em ganhos substanciais para a execução do Algoritmo *Split* pois esta característica é sensível ao modelo sob análise.

A conversão de descritores ATG em ATC em tempo de execução é válida e eficaz, como constatado ao longo do trabalho. Entretanto, outros mecanismos de adequação podem ser construídos para auxiliar nesta tarefa. Atualmente, o uso de taxas funcionais fica sob responsabilidade do modelador que utiliza sua experiência para definir os relacionamentos entre os componentes. Eventualmente, a tarefa de detectar se o modelo será resolvido mais eficientemente usando-se ATG ou ATC pode ser implementado por um compilador mais 'inteligente', já que este pode construir e determinar as matrizes dos termos tensoriais a partir dos parâmetros das funções do modelo, bem como regras para mapear a taxa funcional. Esta tese evidenciou que, dependendo do caso, as funções podem ser convertidas para um formato clássico correspondente (de ATG para ATC) sem perda de informação objetivando a aceleração da convergência ou transiência. A técnica de conversão entre tais descritores é conhecida no contexto de SAN [BRE05a], falta determinar as classes mais adaptadas.

b) novas classes de problemas para utilização do Algoritmo *Split*

Um outro eixo alvo de pesquisas é o de determinar outras classes de modelos que garantirão um desempenho mais eficaz do Algoritmo *Split*, detectando as principais características envolvidas e a composição dos elementos descritos. Este trabalho evidenciou bons resultados quando somente às vezes as entidades envolvidas sincronizam suas atividades com outros elementos. Neste sentido, modelos para aplicações paralelas apresentam estas características, sendo desejável que os nodos ou núcleos de processamento desempenhem atividades internas mas comuniquem o término entre os demais componentes do modelo, sincronizando suas atividades.

Outros problemas também podem ser atacados, tais como a quantificação de comunicação em equipes distribuídas globalmente, sincronizando, por exemplo, a finalização de um módulo de um sistema, para a área da Engenharia de *Software*. Quaisquer problemas onde exista esta qualidade de independência e sincronização global pode ser alvo de descrições modulares tais como as apresentadas nesta tese. Ao permitir que os modeladores pensem apenas nos problemas e informem como

o sistema opera de forma local e global, torna-se transparente a forma de solução que será adotada para cálculo dos índices quantitativos de desempenho a partir de modelos analíticos.

c) métodos para paralelização e distribuição

Métodos de paralelização desta técnica podem ser desenvolvidos mais facilmente do que na abordagem tradicional presente no Algoritmo *Shuffle*. Ao flexibilizar a MVD, reduz-se significativamente a dependência e usa-se a propriedade da Decomposição Aditiva para multiplicar os escalares em posições chave do vetor de probabilidades (dependendo da estratégia adotada). Os resultados apresentados mostraram os pontos de corte para uma execução sequencial do algoritmo. Entretanto, estes mesmos cortes podem ou não apresentar os mesmos desempenhos em soluções paralelas sendo necessária a realização de estudos mais aprofundados para delimitar os cortes a ser adotados para cada caso, garantindo o balanceamento de carga, por exemplo.

Outras premissas devem ser levadas em consideração tais como a quantidade de tarefas a serem realizadas por cada nodo de uma rede, se é melhor enviar cada termo tensorial para um nodo diferente e as implicações existentes ao sincronizar os resultados intermediários no vetor final de probabilidades. O objetivo é explorar a flexibilidade do método para adequar e balancear os gastos de memória (e eventualmente de comunicação) em virtude das arquiteturas heterogêneas em que estarão executando.

d) avanços em primitivas de modelagem e novas aplicações

Sobre avanços em termos de modelagem, podem ser considerados sistemas baseados no conceito de *Multi-layers* e sua formalização. Este tipo de modelo é usado para representar internamente sub-sistemas com alta independência local mas que eventualmente sincronizam atividades globalmente, em conjunção com outros sub-sistemas. Tais modelagens podem ser usadas para descrever comportamentos de redes distribuídas, como por exemplo, redes baseadas em comunicação entre pares (*Peer-to-Peer*) ou até mesmo para testes estocásticos de sistemas. Para estas modelagens será necessário ampliar a descrição dos modelos para contemplar estas novas primitivas bem como fornecer ferramentas especializadas para solução. Para tanto, podem ser usadas as ideias apresentadas como matrizes *Quase Completamente Decomponíveis* (*Nearly Completely Decomposable*), pesquisadas anteriormente no contexto de Cadeias de Markov [KOU84, STE94]. Neste sentido, a adição de outras primitivas de modelagem pode vir a ser proposta, ampliando as possibilidades de descrição e mapeamento de realidades.

A modelagem e a solução devem ser o mais transparente possíveis para os modeladores. Enquanto que o primeiro grupo conhece o problema a ser descrito, o segundo processo explora a formação do descritor e, de acordo com a complexidade dos métodos existentes, decide a forma de dividir os termos tensoriais efetuando a MVD. O usuário, em última análise, deseja apenas que o seu modelo seja convertido em números ou índices que possam ser interpretados para inferir as principais características dos problemas mapeados. A solução do sistema linear que descreve as transições do sistema deve utilizar técnicas eficientes para minimização do tempo necessário descoberta ou não da estacionariedade. Caso fosse preciso uma elevada quantidade de estados onde soluções analíticas não

estivessem habilitadas[2] o ideal seria que, internamente e transparentemente, primitivas de simulação fossem executadas [WEB09]. Neste caso, seriam gerados índices computacionais baseados em aproximações em relação à solução analítica.

e) AUNFs contendo os ajustes diagonais

Ainda acrescenta-se a concepção de uma versão do GTAexpress que consiga criar os AUNFs necessários que guarde em memória os ajustes diagonais. Esta funcionalidade tornará desnecessária a criação do vetor de correção da diagonal, atualmente uma otimização do PEPS mas, ao mesmo tempo, um limitador de memória, determinando que 65 milhões de estados possam ser definidos. Caso esta implementação seja efetuada, o vetor diagonal não precisará ser mais armazenado e o limite do número de estados da ferramenta aumentará consideravelmente. Também podem ser investigadas as relações existentes entre a criação dos AUNFs e a geração do espaço de estados atingível visando uma representação para ser usada na solução.

f) otimizações computacionais

Dadas as características do Algoritmo *Split* e as suas estratégias, a utilização de diferentes tamanhos e hierarquias de memória pode acelerar ainda mais o processo de MVD. Cabe ressaltar que esperam-se resultados melhores em função do tamanho da lista de AUNFs (para modelos que utilizem muita memória para salvar a estrutura, os benefícios não serão tão aparentes). Outras otimizações computacionais podem ser realizadas no nível do projeto da ferramenta para avaliar os ganhos das diferentes implementações. Um exemplo são a utilização de *threads* na multiplicação dos AUNFs pelo vetor de probabilidades. Estes mecanismos podem ser usados para paralelizar a solução e aproveitar a característica independente do Algoritmo *Split* ao tratar cada termos tensorial dos descritores Markovianos.

g) aproximações da solução

Por fim, outros eixos de pesquisa podem ser direcionados para o estudo de aproximações da solução, descoberta de agregações entre os termos tensoriais (e as implicações envolvidas, por exemplo, mesmos cortes e mesmas transformações em termos de permutações) e mecanismos para operar apenas com os estados atingíveis com o *Split*, para citar algumas possibilidades. Esta tese desenvolveu uma nova forma de se multiplicar um vetor por um descritor Markoviano e encontrou uma forma mais otimizada de realizar esta tarefa com base em permutações das ordens originais dos termos tensoriais e em cortes definidos para criar uma matriz agregada que encontre grandes blocos com matrizes identidade. Como observado, outras técnicas podem ser adaptadas ou utilizar o *Split* para acelerar ainda mais a convergência do processo, seja através da descoberta de novas classes de modelos ou através de mecanismos que aproximem os resultados. Para todos os casos, o principal objetivo é aumentar o poder de representação, descrição e solução de sistemas fornecendo os índices de desempenho em menos tempo.

[2]Para o caso do GTAexpress e do PEPS, o número permitido de estados permitidos corresponde à 65 milhões de estados em uma máquina com 4Gb de memória RAM.

8.4 Epílogo

O processo de se buscar a aceleração da MVD é relevante no contexto da avaliação de desempenho de sistemas através de representações tensoriais da matriz de transição que corresponde à Cadeia de Markov. Ao longo da tese foram explorados diferentes formalismos estruturados que podem possuir ou não tais representações eficientes em memória em seus modelos. Em um substancial número de casos, os resultados obtidos demonstraram ganhos significativos para a solução de modelos criados a partir de formalismos estruturados que possuam um descritor Markoviano. Neste sentido, constata-se que o trabalho aqui descrito contribuiu na direção de ampliar o entendimento sobre esta importante problemática dentro da área de Avaliação de Desempenho e Modelagem Estocástica de Sistemas. A tese enunciou as principais formas de se re-estruturar termos tensoriais de descritores Markovianos e descreveu os casos onde a flexibilidade do Algoritmo *Split* mostrou a existência de um equilíbrio entre o tempo e a memória extra a ser gasta. Desta forma, aumentou-se o conhecimento prévio sobre os detalhes de execução do *Split* quando primitivas funcionais de modelagem estão presentes nas abstrações das diferentes realidades e os impactos do seu uso no processo de solução de modelos.

Estas considerações finais e perspectivas encerram a tese. Este capítulo apresentou um resumo do trabalho realizado, descreveu as contribuições e discutiu os avanços em termos de pesquisa em solução de modelos Markovianos para fins de avaliação de desempenho.

REFERÊNCIAS BIBLIOGRÁFICAS

[AHO74] A. V. Aho e J. E. Hopcroft. "The Design and Analysis of Computer Algorithms". Addison-Wesley, 1974, 470p.

[AJM84] M. Ajmone-Marsan, G. Conte e G. Balbo. "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems", *ACM Transactions on Computer Systems*, vol. 2-2, Maio 1984, pp. 93–122.

[AJM95] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli e G. Franceschinis. "Modelling with Generalized Stochastic Petri Nets". John Wiley & Sons, 1995, 324p.

[BAE05] J. C. M. Baeten. "A brief history of process algebra", *Theoretical Computer Science*, vol. 335-(2-3), Maio 2005, pp. 131–146.

[BAI03] C. Baier, B. Haverkort, H. Hermanns e J.-P. Katoen. "Model-checking algorithms for continuous-time markov chains", *IEEE Transactions on Software Engineering*, vol. 29-6, Junho 2003, pp. 524–541.

[BAL04] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho e T. Webber. "Parallel PEPS Tool Performance Analysis using Stochastic Automata Networks", *Lecture Notes in Computer Science*, vol. 3149, Agosto/Setembro 2004, pp. 214–219.

[BAL05] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes e A. Sales. "Performance Models for Master/Slave Parallel Programs", *Electronic Notes In Theoretical Computer Science*, vol. 128-4, Abril 2005, pp. 101–121.

[BAL07] G. Balbo. "Introduction to Generalized Stochastic Petri Nets", *Lecture Notes in Computer Science*, vol. 4486, Maio/Junho 2007, pp. 83–131.

[BEN03a] A. Benoit, L. Brenner, P. Fernandes, B. Plateau e W. J. Stewart. "The PEPS Software Tool", *Lecture Notes in Computer Science*, vol. 2794, Setembro 2003, pp. 98–115.

[BEN03b] A. Benoit. "Méthodes et algorithmes pour l'évaluation des performances des systèmes informatiques à grand espace d'états", Tese de Doutorado, Institut National Polytechnique de Grenoble, France, 2003, 170p.

[BEN04a] A. Benoit, L. Brenner, P. Fernandes e B. Plateau. "Aggregation of stochastic automata networks with replicas", *Linear Algebra and its Applications*, vol. 386, Julho 2004, pp. 111–136.

[BEN04b] A. Benoit, P. Fernandes, B. Plateau e W. J. Stewart. "On the benefits of using functional transitions and Kronecker algebra", *Performance Evaluation*, vol. 58, Dezembro 2004, pp. 367–390.

[BEN06] A. Benoit, B. Plateau e W. J. Stewart. "Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems", *Future Generation Computer Systems*, vol. 22-7, Agosto 2006, pp. 838–847.

[BER98] M. Bernardo e R. Gorrieri. "A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time", *Theoretical Computer Science*, vol. 202-(1-2), Julho 1998, pp. 1–54.

[BIL03] J. Billington, S. Christensen, K. Van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno e M. Weber. "The Petri net markup language: Concepts, technology e tools", *Lecture Notes in Computer Science*, vol. 2679, Fevereiro 2003, pp. 483–505.

[BHA97] A. T. Bharucha-Reid. "Elements of the Theory of Markov Processes and their Applications". Courier Dover Publications, 1997, 480p.

[BOL98] G. Bolch, S. Greiner, H. de Meer e K. S. Trivedi. "Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications". John Wiley & Sons, 1998, 726p.

[BRA03] J. T. Bradley, N. J. Dingle, S. Gilmore e W. J. Knottenbelt. "Derivation of passage-time densities in PEPA models using ipc: the imperial PEPA compiler". In: 11^{th} IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS), 2003, pp. 344–351.

[BRE05a] L. Brenner, P. Fernandes e A. Sales. "The Need for and the Advantages of Generalized Tensor Algebra for Kronecker Structured Representations", *International Journal of Simulation: Systems, Science & Technology*, vol. 6-(3-4), Fevereiro 2005, pp. 52–60.

[BRE05b] L. Brenner, P. Fernandes, A. Sales e T. Webber. "A Framework to Decompose GSPN models". In: 26^{th} International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (ATPN), 2005, pp. 1–20.

[BRE07] L. Brenner, P. Fernandes, B. Plateau e I. Sbeity. "PEPS2007 – Stochastic Automata Networks Software Tool". In: 4^{th} International Conference on the Quantitative Evaluation of Systems (QEST), 2007, pp. 163–164.

[BUC94] P. Buchholz. "Exact and Ordinary Lumpability in Finite Markov Chains", *Journal of Applied Probability*, vol. 31-1, Março 1994, pp. 59–75.

[BUC98] P. Buchholz. "A new approach combining simulation and randomization for the analysis of large continuous time Markov Chains", *ACM Transactions on Modeling and Computer Simulation*, vol. 8-2, Abril 1998, pp. 194–222.

[BUC00] P. Buchholz, G. Ciardo, S. Donatelli e P. Kemper. "Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models", *INFORMS Journal on Computing*, vol. 13-3, Julho 2000, pp. 203–222.

[BUC02a] P. Buchholz e P. Kemper. "Hierarchical reachability graph generation for Petri nets", *Formal Methods in Systems Design*, vol. 21-3, Novembro 2002, pp. 281–315.

[BUC02b] P. Buchholz e P. Kemper. "Efficient Computation and Representation of Large Reachability Sets for Composed Automata", *Discrete Event Dynamic Systems*, vol. 12-3, Julho 2002, pp. 265–286.

[BUC06] P. Buchholz. "Structured analysis techniques for large markov chains". In: 1^{st} Workshop on Tools for Solving Structured Markov Chains (SMCTools), 2006, 9p.

[BUC08] P. Buchholz. "Product form approximations for communicating markov processes". In: 5^{th} International Conference on the Quantitative Evaluation of Systems (QEST), 2008, pp. 135–144.

[CAM99] J. Campos, S. Donatelli e M. Silva. "Structured Solution of Asynchronously Communicating Stochastic Modules", *IEEE Transactions on Software Engineering*, vol. 25-2, Março/Abril 1999, pp. 147–165.

[CHA06] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer e A. F. Zorzo. "Analytical modeling for operating system schedulers on NUMA systems", *Electronic Notes in Theoretical Computer Science*, vol. 151-3, Junho 2006, pp. 131–149.

[CHI90] G. Chiola, C. Dutheillet, G. Franceschinis e S. Haddad. "On well-formed coloured nets and their symbolic reachability graph". In: 11^{th} International Conference on Application and Theory of Petri Nets (ATPN), 1990, pp. 387–410.

[CHI93a] G. Chiola, M. Ajmone-Marsan, G. Balbo e G. Conte. "Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications", *IEEE Transactions on Software Engineering*, vol. 19-2, Fevereiro 1993, pp. 89–107.

[CHI93b] G. Chiola, C. Dutheillet, G. Franceschinis e S. Haddad. "Stochastic well-formed coloured nets for symmetric modelling applications", *IEEE Transaction on Computers*, vol. 42-11, Novembro 1993, pp. 1343–1360.

[CHI97] G. Chiola, C. Dutheillet, G. Franceschinis e S. Haddad. "A symbolic reachability graph for coloured petri nets", *Theoretical Computer Science*, vol. 176-(1-2), Abril 1997, pp. 39–65.

[CHU04] M.-Y. Chung, G. Ciardo, S. Donatelli, N. He, B. Plateau, W. J. Stewart, E. Sulaiman e J. Yu. "A Comparison of Structural Formalisms for Modeling Large Markov Models". In: 18^{th} International Parallel and Distributed Processing Symposium (IPDPS), 2004, 8p.

[CIA91] G. Ciardo e K. S. Trivedi. "A Decomposition Approach for Stochastic Petri Nets Models". In: 4^{th} International Workshop on Petri Nets and Performance Models (PNPM), 1991, pp. 74–83.

[CIA97] G. Ciardo e A. S. Miner. "Storage Alternatives for Large Structured State Spaces", *Lecture Notes in Computer Science*, vol. 1245, Junho 1997, pp. 44–57.

[CIA01] G. Ciardo, R. L. Jones, A. S. Miner e R. Siminiceanu. "SMART: Stochastic Model Analyzer for Reliability and Timing". In: Aachen International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems, 2001, pp. 29–34.

[CZE07] R. M. Czekster, P. Fernandes, J. M. Vincent e T. Webber. "Split: a flexible and efficient algorithm to vector-descriptor product". In: 2^{nd} International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), 2007, 8p.

[CZE09] R. M. Czekster, P. Fernandes e T. Webber. "GTAexpress: a Software Package to Handle Kronecker Descriptors". In: 6^{th} International Conference on Quantitative Evaluation of Systems (QEST), 2009, pp. 281–282.

[DAV81] M. Davio. "Kronecker Products and Shuffle Algebra", *IEEE Transactions on Computers*, vol. C-30-(2), Fevereiro 1981, pp. 116–125.

[DEA98a] D. D. Deavours e W. H. Sanders. "An efficient disk-based tool for solving large Markov models", *Performance Evaluation*, vol. 33-1, Junho 1998, pp. 67–84.

[DEA98b] D. D. Deavours e W. H. Sanders. "'On-the-fly' Solution Techniques for Stochastic Petri Nets and Extensions", *IEEE Transactions on Software Engineering*, vol. 24-10, Outubro 1998, pp. 889–902.

[DON93] S. Donatelli. "Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space", *Performance Evaluation*, vol. 18, Julho 1993, pp. 21–36.

[DON94] S. Donatelli. "Superposed generalized stochastic Petri nets: definition and efficient solution". In: 15^{th} International Conference on Applications and Theory of Petri Nets (ATPN), 1994, pp. 258–277.

[DOT05] F. L. Dotti, P. Fernandes, A. Sales e O. M. Santos. "Modular Analytical Performance Models for Ad Hoc Wireless Networks". In: 3^{rd} International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2005, pp. 164–173.

[ERL09] A. K. Erlang. "The Theory of Probabilities and Telephone Conversations", *Nyt Tidsskrift for Matematik*, vol. 20-B, Julho 1909, pp. 33–39.

[FER98a] P. Fernandes. "Méthodes numériques pour la solution de systèmes Markoviens à grand espace d'états", Tese de Doutorado, Institut National Polytechnique de Grenoble, France, 1998, 262p.

[FER98b] P. Fernandes, B. Plateau e W. J. Stewart. "Efficient descriptor - Vector multiplication in Stochastic Automata Networks", *Journal of the ACM*, vol. 45-3, Maio 1998, pp. 381–414.

[FER98c] P. Fernandes, B. Plateau e W. J. Stewart. "Optimizing tensor product computations in stochastic automata networks", *RAIRO – Recherche opérationnelle*, vol. 32-3, Novembro 1998, pp. 325–351.

[FER08] P. Fernandes, J. M. Vincent e T. Webber. "Perfect Simulation of Stochastic Automata Networks", *Lecture Notes in Computer Science*, vol. 5055, Junho 2008, pp. 249–263.

[FLO85] G. Florin e S. Natkin. "Les réseaux de Petri stochastiques", *Tecniques et Sciences Informatiques*, vol. 4-1, Fevereiro 1985, pp. 143–160.

[GEL87] E. Gelenbe e G. Pujolle. "Introduction to Queueing Networks". John Wiley & Sons, 1987, 192p.

[GEL91] E. Gelenbe. "Product form queueing networks with negative and positive customers", *Journal of Applied Probability*, vol. 28, Setembro 1991, pp. 656–663.

[GIL01] S. Gilmore, J. Hillston e M. Ribaudo. "PEPA-coloured stochastic Petri nets". In: 17^{th} UK Performance Engineering Workshop (UKPEW), 2001, pp. 155–166.

[GIL03] S. Gilmore, J. Hillston, L. Kloul e M. Ribaudo. "PEPA nets: a structured performance modelling formalism", *Performance Evaluation*, vol. 54-2, Outubro 2003, pp.79–104.

[GIL04a] S. Gilmore, J. Hillston, L. Kloul e M. Ribaudo. "Software performance modelling using PEPA nets". In: 4^{th} International Workshop on Software and Performance (WOSP), 2004, pp. 13–24.

[GIL04b] S. Gilmore, L. Kloul e D. Piazza. "Modelling Role-Playing Games Using PEPA nets", *Lecture Notes in Computer Science*, vol. 3280, Outubro 2004, pp. 523–532.

[GIL06] S. Gilmore, V. Haenel, J. Hillston e J. Tenzer. "A design environment for mobile applications". In: 20^{th} International Parallel and Distributed Processing Symposium (IPDPS), 2006, 10p.

[GOT95] N. Götz, H. Hermanns, U. Herzog, V. Mertsiotakis e M. Rettelbach. "Quantitative Methods in Parallel Systems: Stochastic process algebras". Springer, 1995, 312p.

[HAG02] O. Häggström. "Finite Markov Chains and Algorithmic Applications". Cambridge University Press, 2002, 124p.

[HIL96] J. Hillston. "A compositional approach to performance modelling". Cambridge University Press, 1996, 168p.

[HIL01] J. Hillston e L. Kloul. "An Efficient Kronecker Representation for PEPA models". In: 1^{st} Joint International Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM-PROBMIV), 2001, pp. 120–135.

[HIL04] J. Hillston e M. Ribaudo. "Modelling mobility with PEPA nets", *Lecture Notes in Computer Science*, vol. 3280, Outubro 2004, pp. 513–522.

[HIL05a] J. Hillston. "Process algebras for quantitative analysis". In: 20^{th} Annual IEEE Symposium on Logic in Computer Science (LICS), 2005, pp. 239–248.

[HIL05b] J. Hillston. "Tuning Systems: From Composition to Performance", *The Computer Journal*, vol. 48-4, Julho 2005, pp. 385–400.

[HIL07] J. Hillston e L. Kloul. "Formal techniques for performance analysis: blending SAN and PEPA", *Formal Aspects of Computing*, vol. 19-1, Março 2007, pp. 3–33.

[HOL95] D. R. W. Holton. "A PEPA specification of an industrial production cell", *The Computer Journal*, vol. 38-7, Dezembro 1995, pp. 542–551.

[HOW07] R. A. Howard. "Dynamic Probabilistic Systems, Volume I: Markov Models". Dover Publications, 2007, 608p.

[JAI91] R. Jain. "The art of computer systems performance analysis". John Wiley & Sons, 1991, 720p.

[JEN81] K. Jensen. "Coloured Petri Nets and the Invariant Method", *Theoretical Computer Science*, vol. 14-3, Março 1981, pp. 317–336.

[JEN96] K. Jensen. "Coloured Petri nets: basic concepts, analysis methods and practical use". Springer, 1996, 84p.

[JEN07] K. Jensen, L. M. Kristensen e L. Wells. "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems", *International Journal on Software Tools for Technology Transfer*, vol. 9-(3-4), Junho 2007, pp. 213–254.

[KEM96] P. Kemper. "Numerical Analysis of Superposed GSPNs", *IEEE Transactions on Software Engineering*, vol. 22-9, Setembro 1996, pp. 615–628.

[KLE75] L. Kleinrock. "Queueing Systems. Volume 1: Theory". Wiley-Interscience, 1975, 448p.

[KOU84] J. R. Koury, D. F. McAllister e W. J. Stewart. "Iterative Methods for Computing Stationary Distributions of Nearly Completely Decomposable Markov Chains", *SIAM Journal on Algebraic and Discrete Methods*, Junho 1984, vol. 5-2, pp. 164–186.

[LAN06] A. N. Langville e C. D. Meyer. "Google's PageRank and Beyond: The Science of Search Engine Rankings". Princeton University Press, 2006, 234p.

[LAV83] S. S. Lavenberg. "Computer Performance Modeling Handbook". Academic Press, 1983, 399p.

[LAZ84] E. D. Lazowska, J. Zahorjan, G. S. Graham e K. C. Sevcik. "Quantitative System Performance: Computer System Analysis Using Queueing Network Models". Prentice Hall, 1984, 417p.

[LI01] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee e R. Morris. "Capacity of Ad Hoc Wireless Networks". In: 7^{th} Annual International Conference on Mobile Computing and Networking (MOBICOM), 2001, pp. 61–69.

[MIN99a] A. S. Miner e G. Ciardo. "A data structure for the efficient Kronecker solution of GSPNs". In: 8^{th} International Workshop on Petri Nets and Performance Models (PNPM), 1999, pp. 22–31.

[MIN99b] A. S. Miner e G. Ciardo. "Efficient Reachability Set Generation and Storage Using Decision Diagrams", *Lecture Notes in Computer Science*, vol. 1639, Junho 1999, pp. 6–25.

[MIN00] A. S. Miner. "Data Structures for the Analysis of Large Structured Markov Models", Tese de Doutorado, The College of William and Mary, Williamsburg, VA, EUA, 2000, 315p.

[MOL82] M. K. Molloy. "Performance analysis using stochastic Petri nets", *IEEE Transactions on Computer*s, vol. C-31-(9), Setembro 1982, pp. 913–917.

[MUR89] T. Murata. "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE*, vol. 77-4, Abril 1989, pp. 541–580.

[NOR98] J. R. Norris. "Markov Chains". Cambridge University Press, 1998, 256p.

[PAR72] D. L. Parnas. "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, vol. 15-12, Dezembro 1972, pp. 1053–1058.

[PLA85] B. Plateau. "On the stochastic structure of parallelism and synchronization models for distributed algorithms". In: International Conference on Measurements and Modeling of Computer Systems (ACM SIGMETRICS 1985), 1985, pp. 147–154.

[PLA88] B. Plateau, J.M. Fourneau e K.H. Lee. "PEPS: A package for solving complex Markov model of parallel systems". In: 1^{st} Modeling Techniques and Tools for Computer Performance Evaluation (TOOLS). 1988, pp. 341–360.

[PLA91] B. Plateau e K. Atif. "Stochastic Automata Networks for modeling parallel systems", *IEEE Transactions on Software Engineering*, vol. 17-10, Outubro 1991, pp. 1093–1108.

[PET66] C. A. Petri. "Communication with automata". DTIC Research Report AD0630125, 1966, vol. 1-1, 97p.

[PET77] J. L. Peterson. "Petri Nets", *ACM Computing Surveys*, vol. 9-3, Setembro 1977, pp. 223–252.

[PRO96] J. G. Propp e D. B. Wilson. "Exact Sampling with Coupled Markov Chains and Applications to Statistical Mechanics", *Random Structures and Algorithms*, vol. 9-(1-2), Agosto/Setembro 1996, pp. 223–252.

[RIB95] M. Ribaudo. "Stochastic Petri net semantics for stochastic process algebras". In: 6^{th} International Workshop on Petri Nets and Performance Models (PNPM), 1995, pp. 148–157.

[SAA95] Y. Saad. "Iterative Methods for Sparse Linear Systems". PWS Publishing Company, 1995, 2^{nd} edition, 528p.

[SAL09] A. Sales. "Réseaux d'Automates Stochastiques: Génération de l'espace d'états atteignables et Multiplication vecteur-descripteur pour une sémantique en temps discret", Tese de Doutorado, Institut Polytechnique de Grenoble, France, 2009, 268p.

[SAN01] W. H. Sanders e J. F. Meyer. "Stochastic Activity Networks: Formal Definitions and Concepts", *Lecture Notes in Computer Science*, vol. 2090, Julho 2001, pp. 315–343.

[SAV81] C. H. Saver e K. M. Chandy. "Computer Systems Performance Modeling". Prentice Hall, 1981, 352p.

[SCH83] P. J. Schweitzer. "Aggregation Methods for Large Markov Chains". In: International Workshop on Computer Performance and Reliability, 1983, pp. 275–286.

[STE91] W. J. Stewart. "MARCA: Markov chain analyzer, a software package for Markov modeling". In: 1^{st} International Workshop on the Numerical Solution of Markov Chains (NSMC), 1991, pp. 37–62.

[STE94] W. J. Stewart. "Introduction to the numerical solution of Markov chains". Princeton University Press, 1994, 539p.

[STE01] Ö. Stenflo. "Ergodic Theorems for Markov chains represented by Iterated Function Systems", *Bulletin of the Polish Academy of Sciences*, Julho 2001, vol. 49-1, pp. 27–43.

[STE07] W. J. Stewart. "Performance Modelling and Markov Chains", *Lecture Notes in Computer Science*, vol. 4486, Maio/Junho 2007, pp. 1–34.

[STE09] W. J. Stewart. "Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling". Princeton University Press, 2009, 758p.

[TAD99] C. Tadonki e B. Philippe. "Parallel Multiplication of a Vector by a Kronecker Tensor Product of matrices", *Parallel and Distributed Computing Practices*, vol. 2-4, Junho 1999, pp. 53–67.

[TRI07] M. Tribastone. "The PEPA Plug-in Project". In: 4^{th} International Conference on Quantitative Evaluation of Systems (QEST), 2007, pp. 53–54.

[TRI09] M. Tribastone, A. Duguid e S. Gilmore. "The PEPA Eclipse Plug-in", *ACM SIGMETRICS Performance Evaluation Review*, vol. 36-4, Março 2009, pp. 28–33.

[WEB09] T. Webber. "Reducing the Impact of State Space Explosion in Stochastic Automata Networks", Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul, Brasil, 2009, 136p.

[WEL06] L. Wells. "Performance analysis using CPN tools". In: 1^{st} International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), 2006, 10p.

[YOU79] E. Yourdon e L. L. Constantine. "Structured design: fundamentals of a discipline of computer program and systems design". Prentice-Hall, 1979, 473p.

A. EXPERIMENTOS

Este apêndice mostrará experimentos conduzidos ao longo do trabalho. Serão apresentados os dados obtidos dos seguintes casos (apresenta-se também a seção onde cada resultado encontra-se melhor explicado):

Experimentos para descritores clássicos (1 e 3):

- *slaves_10c* – Seção 7.2.1;
- *ad14c* – Seção 7.2.2;
- *fas20c* – Seção 7.2.3;
- *phil14c* – Seção 7.2.5;
- *rs15_15c* – Seção 7.2.6;

Experimentos para descritores generalizados (5, 6, 7 e 8):

- *slaves_10f* – Seção 7.2.1;
- *ad14f* – Seção 7.2.2;

Cabe ressaltar que o modelo (definido com descritor clássico) para o caso do *First Available Server* (FAS) – Seção 7.2.3 – contêm os cortes e reordens para verificação do custo das permutações.

A seguir serão mostradas as esparsidades dos termos tensoriais positivos, o ponto de corte (σ) escolhido em cada experimento, a memória, o número de AUNFs necessários e as re-estruturações escolhidas (permutações). Serão mostrados também trechos do arquivo de resultados da ferramenta GTAexpress (baseadas em um arquivo com terminação *.tim*).

A.1 Experimento 1

A.1.1 Caso slaves_10c - Experimento 1

```
============================================================
File slaves10c.tim
============================================================
slaves10c -- A model with 12 modules and 33 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 3553 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2463180 (76974 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 2 12 12 3 3 3 3 3 3 3 3 3 3 11 2 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
    0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
    9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]    [   1   1|  I   I   I   I   I   I   I   I   I   I   ]
   10. [  0  1  2  3  4  5  6  7  8  9 10 11|]    [   1  41   3   3   3   3   3   3   3   3   3|  ]
   11. [  0  1  2  3  4  5  6  7  8  9 10 11|]    [   1  40   2   2   2   2   2   2   2   2   2|  ]
   12. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   13. [  0  1  3| 2  4  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   14. [  0  1  4| 2  3  5  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   15. [  0  1  5| 2  3  4  6  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   16. [  0  1  6| 2  3  4  5  7  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   17. [  0  1  7| 2  3  4  5  6  8  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   18. [  0  1  8| 2  3  4  5  6  7  9 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   19. [  0  1  9| 2  3  4  5  6  7  8 10 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   20. [  0  1 10| 2  3  4  5  6  7  8  9 11 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   21. [  0  1 11| 2  3  4  5  6  7  8  9 10 ]    [   1  40   1|  I   I   I   I   I   I   I   I   I   ]
   22. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]    [   1   1   1   1   1   1   1   1   1   1|  I   ]
   23. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   24. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   25. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   26. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   27. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   28. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   29. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   30. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   31. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
   32. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]    [  40   2|  I   I   I   I   I   I   I   I   I   I   ]
```

A.1.2 Caso ad14c - Experimento 1

```
============================================================
File ad14c.tim
============================================================
ad14c -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
```

```
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 14 (0 Kb)
Chosen cuts: [ 2 4 5 6 6 6 6 6 6 6 6 6 5 4 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1| I I I I I I I I I I I I ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1| I I I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1| I I I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  4. [  1  2  3  4  5  6| 0  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  5. [  2  3  4  5  6  7| 0  1  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  6. [  3  4  5  6  7  8| 0  1  2  9 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  7. [  4  5  6  7  8  9| 0  1  2  3 10 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  8. [  5  6  7  8  9 10| 0  1  2  3  4 11 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
  9. [  6  7  8  9 10 11| 0  1  2  3  4  5 12 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 10. [  7  8  9 10 11 12| 0  1  2  3  4  5  6 13 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 11. [  8  9 10 11 12 13| 0  1  2  3  4  5  6  7 ]  [ 1 1 1 1 1 1| I I I I I I I I ]
 12. [  9 10 11 12 13| 0  1  2  3  4  5  6  7  8 ]  [ 1 1 1 1 1| I I I I I I I I I ]
 13. [ 10 11 12 13| 0  1  2  3  4  5  6  7  8  9 ]  [ 1 1 1 1| I I I I I I I I I ]
```

A.1.3 Caso fas20c - Experimento 1

```
============================================================
File fas20c.tim
============================================================
fas20c -- A model with 20 modules and 19 events
User name: 'FAS20c'
------------
Problem Size
------------
Product state space:           1048576 states
Reachable state space:         1048576 states
Modules Sizes:                   [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 522 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 19 (0 Kb)
Chosen cuts: [ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1|I I I I I I I I I I I I I I I I I I ]
  1. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1|I I I I I I I I I I I I I I I I I ]
  2. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1|I I I I I I I I I I I I I I I ]
  3. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1|I I I I I I I I I I I I I I ]
  4. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1|I I I I I I I I I I I I I ]
  5. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1|I I I I I I I I I I I I ]
  6. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1|I I I I I I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1|I I I I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I ]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I ]
 14. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16|17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I ]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17|18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18|19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19|]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|]
```

A.1.4 Caso phil14c - Experimento 1

```
============================================================
File phil14c.tim
============================================================
phil14c -- A model with 14 modules and 28 events
User name: 'PHILOSOPHERS'
------------
Problem Size
------------
```

```
Product state space:           4782969 states
Reachable state space:          195025 states
Modules Sizes:                   [ 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 2342 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 42 (1 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 2 1| I I I I I I I I I I I I ]
  1. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 ]  [ 2 1| I I I I I I I I I I I I ]
  2. [ 11 12| 0  1  2  3  4  5  6  7  8  9 10 13 ]  [ 2 1| I I I I I I I I I I I I ]
  3. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1| I I I I I I I I I I I I ]
  4. [ 10 11| 0  1  2  3  4  5  6  7  8  9 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  5. [ 11 12| 0  1  2  3  4  5  6  7  8  9 10 13 ]  [ 1 1| I I I I I I I I I I I I ]
  6. [  9 10| 0  1  2  3  4  5  6  7  8 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  7. [ 10 11| 0  1  2  3  4  5  6  7  8  9 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
  8. [  8  9| 0  1  2  3  4  5  6  7 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
  9. [  9 10| 0  1  2  3  4  5  6  7  8 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 10. [  7  8| 0  1  2  3  4  5  6  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 11. [  8  9| 0  1  2  3  4  5  6  7 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 12. [  6  7| 0  1  2  3  4  5  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 13. [  7  8| 0  1  2  3  4  5  6  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 14. [  5  6| 0  1  2  3  4  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 15. [  6  7| 0  1  2  3  4  5  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 16. [  4  5| 0  1  2  3  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 17. [  5  6| 0  1  2  3  4  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 18. [  3  4| 0  1  2  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 19. [  4  5| 0  1  2  3  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 20. [  2  3| 0  1  4  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 21. [  3  4| 0  1  2  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 22. [  1  2| 0  3  4  5  6  7  8  9 10 11 12 13 ]  [ 2 1| I I I I I I I I I I I I ]
 23. [  2  3| 0  1  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 24. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 25. [  1  2| 0  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 26. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1| I I I I I I I I I I I I ]
 27. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 ]  [ 1 2| I I I I I I I I I I I I ]
```

A.1.5 Caso rs15_15c - Experimento 1

```
============================================================
File rs15_15c.tim
============================================================
rs15_15c -- A model with 16 modules and 30 events
User name: 'RS15_15'
------------
Problem Size
------------
Product state space:           524288 states
Reachable state space:         32768 states
Modules Sizes:                   [ 16 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 266 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 450 (14 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0 15| 1  2  3  4  5  6  7  8  9 10 11 12 13 14 ]  [  15 1| I I I I I I I I I I I I ]
  1. [  0 14| 1  2  3  4  5  6  7  8  9 10 11 12 13 15 ]  [  15 1| I I I I I I I I I I I I ]
  2. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  3. [  0 12| 1  2  3  4  5  6  7  8  9 10 11 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  4. [  0 11| 1  2  3  4  5  6  7  8  9 10 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  5. [  0 10| 1  2  3  4  5  6  7  8  9 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  6. [  0  9| 1  2  3  4  5  6  7  8 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  7. [  0  8| 1  2  3  4  5  6  7  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  8. [  0  7| 1  2  3  4  5  6  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
  9. [  0  6| 1  2  3  4  5  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
 10. [  0  5| 1  2  3  4  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I ]
```

```
 11. [  0  4| 1  2  3  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 12. [  0  3| 1  2  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 13. [  0  2| 1  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 14. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 15. [  0 15| 1  2  3  4  5  6  7  8  9 10 11 12 13 14 ]  [  15 1| I I I I I I I I I I I I I I ]
 16. [  0 14| 1  2  3  4  5  6  7  8  9 10 11 12 13 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 17. [  0 13| 1  2  3  4  5  6  7  8  9 10 11 12 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 18. [  0 12| 1  2  3  4  5  6  7  8  9 10 11 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 19. [  0 11| 1  2  3  4  5  6  7  8  9 10 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 20. [  0 10| 1  2  3  4  5  6  7  8  9 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 21. [  0  9| 1  2  3  4  5  6  7  8 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 22. [  0  8| 1  2  3  4  5  6  7  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 23. [  0  7| 1  2  3  4  5  6  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 24. [  0  6| 1  2  3  4  5  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 25. [  0  5| 1  2  3  4  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 26. [  0  4| 1  2  3  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 27. [  0  3| 1  2  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 28. [  0  2| 1  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
 29. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1| I I I I I I I I I I I I I I ]
```

A.2 Experimento 3

A.2.1 Caso slaves_10c - Experimento 3

```
============================================================
File slaves10c.tim
============================================================
slaves10c -- A model with 12 modules and 33 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 3553 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 11939214 (373100 Kb)
Chosen cuts: [ 3 4 5 6 7 8 9 10 11 12 12 12 3 4 5 6 7 8 9 10 11 12 12 3 4 5 6 7 8 9 10 11 12 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ 1  I 1|I I I I I I I I I ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ 1  I I 1|I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ 1  I I I 1|I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ 1  I I I I 1|I I I I I I ]
  4. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ 1  I I I I I 1|I I I I I ]
  5. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ 1  I I I I I I 1|I I I I ]
  6. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ 1  I I I I I I I 1|I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ 1  I I I I I I I I 1|I I ]
  8. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ 1  I I I I I I I I I 1|I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1  I I I I I I I I I I 1|]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 40 2 2 2 2 2 2 2 2 2 2|]
 12. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ 1 40 1|I I I I I I I I I ]
 13. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ 1 40 I 1|I I I I I I I I ]
 14. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ 1 40 I I 1|I I I I I I I ]
 15. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ 1 40 I I I 1|I I I I I I ]
 16. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ 1 40 I I I I 1|I I I I I ]
 17. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ 1 40 I I I I I 1|I I I I ]
 18. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ 1 40 I I I I I I 1|I I I ]
 19. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ 1 40 I I I I I I I 1|I I ]
 20. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ 1 40 I I I I I I I I 1|I ]
 21. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 40 I I I I I I I I I 1|]
 22. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1  I 1 1 1 1 1 1 1 1 1 1|]
 23. [  0  1  2| 3  4  5  6  7  8  9 10 11 ]  [ I 40 2|I I I I I I I I I ]
 24. [  0  1  2  3| 4  5  6  7  8  9 10 11 ]  [ I 40 I 2|I I I I I I I I ]
 25. [  0  1  2  3  4| 5  6  7  8  9 10 11 ]  [ I 40 I I 2|I I I I I I I ]
 26. [  0  1  2  3  4  5| 6  7  8  9 10 11 ]  [ I 40 I I I 2|I I I I I I ]
 27. [  0  1  2  3  4  5  6| 7  8  9 10 11 ]  [ I 40 I I I I 2|I I I I I ]
 28. [  0  1  2  3  4  5  6  7| 8  9 10 11 ]  [ I 40 I I I I I 2|I I I I ]
 29. [  0  1  2  3  4  5  6  7  8| 9 10 11 ]  [ I 40 I I I I I I 2|I I I ]
```

```
 30. [  0  1  2  3  4  5  6  7  8  9|10 11 ]  [ I 40 I I I I I I I 2|I I ]
 31. [  0  1  2  3  4  5  6  7  8  9 10|11 ]  [ I 40 I I I I I I I I 2|I ]
 32. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ I 40 I I I I I I I I I 2|]
```

A.2.2 Caso ad14c - Experimento 3

```
============================================================
File ad14c.tim
============================================================
ad14c -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 413345 (12917 Kb)
Chosen cuts: [ 14 4 5 6 7 8 9 10 11 12 13 14 14 14 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I I 1 1|  ]
  1. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1|I I I I I I I I I I ]
  2. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1|I I I I I I I I I ]
  3. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ 1 1 1 1 1 1|I I I I I I I I ]
  4. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I 1 1 1 1 1 1 1|I I I I I I I ]
  5. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I 1 1 1 1 1 1 1|I I I I I I ]
  6. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I 1 1 1 1 1 1 1|I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I 1 1 1 1 1 1 1|I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I 1 1 1 1 1 1|I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I 1 1 1 1 1 1|I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I 1 1 1 1 1 1|I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I 1 1 1 1 1 1|]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I 1 1 1 1 1|]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I 1 1 1 1|]
```

A.2.3 Caso fas20c - Experimento 3

```
============================================================
File fas20c.tim
============================================================
fas20c -- A model with 20 modules and 19 events
User name: 'FAS20c'
------------
Problem Size
------------
Product state space:           1048576 states
Reachable state space:         1048576 states
Modules Sizes:                   [ 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 522 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 19 (0 Kb)
Chosen cuts: [ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1|I I I I I I I I I I I I I I I I I I ]
  1. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1|I I I I I I I I I I I I I I I I ]
  2. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1|I I I I I I I I I I I I I I ]
  3. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1|I I I I I I I I I I I I I ]
  4. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1|I I I I I I I I I I I I ]
  5. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1|I I I I I I I I I I I ]
  6. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1|I I I I I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1|I I I I I I I I ]
```

```
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1|I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I I ]
 12. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I I ]
 14. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|16 17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16|17 18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I I ]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17|18 19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18|19 ]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19|]  [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|]
```

A.2.4 Caso phil14c - Experimento 3

```
============================================================
File phil14c.tim
============================================================
phil14c -- A model with 14 modules and 28 events
User name: 'PHILOSOPHERS'
------------
Problem Size
------------
Product state space:           4782969 states
Reachable state space:          195025 states
Modules Sizes:                   [ 3 3 3 3 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 2342 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 4517246 (141163 Kb)
Chosen cuts: [ 14 14 13 14 12 13 11 12 10 11 9 10 8 9 7 8 6 7 5 6 4 5 3 4 2 3 2 14 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I I 2 1|]
  1. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ 2 I I I I I I I I I I I I 1|]
  2. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I I I I I 2 1|I ]
  3. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ I I I I I I I I I I I I 1 1|]
  4. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I I I I I I 2 1|I I ]
  5. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 ]  [ I I I I I I I I I I I 1 1|I I ]
  6. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I I I I I I 2 1|I I I ]
  7. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 ]  [ I I I I I I I I I I 1 1|I I I ]
  8. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I I I I I I 2 1|I I I I ]
  9. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 ]  [ I I I I I I I I I 1 1|I I I I ]
 10. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I I I I I I 2 1|I I I I I ]
 11. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 ]  [ I I I I I I I I 1 1|I I I I I ]
 12. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I I I I I I 2 1|I I I I I I ]
 13. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 ]  [ I I I I I I I 1 1|I I I I I I ]
 14. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I I I I I I 2 1|I I I I I I I ]
 15. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 ]  [ I I I I I I 1 1|I I I I I I I ]
 16. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ I I I I 2 1|I I I I I I I I ]
 17. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 ]  [ I I I I I 1 1|I I I I I I I ]
 18. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ I I I 2 1|I I I I I I I I I ]
 19. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
 20. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 2 1|I I I I I I I I I ]
 21. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 ]  [ I I I 1 1|I I I I I I I I ]
 22. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 ]  [ I 2 1|I I I I I I I I I ]
 23. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 1 1|I I I I I I I I ]
 24. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1|I I I I I I I I I I ]
 25. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 ]  [ I 1 1|I I I I I I I I I ]
 26. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 ]  [ 1 1|I I I I I I I I I I ]
 27. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|]  [ 1 I I I I I I I I I I I 2|]
```

A.2.5 Caso rs15_15c - Experimento 3

```
============================================================
File rs15_15c.tim
============================================================
rs15_15c -- A model with 16 modules and 30 events
User name: 'RS15_15'
------------
Problem Size
------------
Product state space:           524288 states
Reachable state space:          32768 states
```

```
Modules Sizes:                  [ 16 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 0
Size of the Normalized Descriptor: 266 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 983010 (30719 Kb)
Chosen cuts: [ 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|]  [  15 I I I I I I I I I I I I I I 1|]
  1. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 ]  [  15 I I I I I I I I I I I I I 1|I ]
  2. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 ]  [  15 I I I I I I I I I I I I 1|I I ]
  3. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 ]  [  15 I I I I I I I I I I I 1|I I I ]
  4. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 ]  [  15 I I I I I I I I I I 1|I I I I ]
  5. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 ]  [  15 I I I I I I I I I 1|I I I I I ]
  6. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 ]  [  15 I I I I I I I I 1|I I I I I I ]
  7. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 ]  [  15 I I I I I I I 1|I I I I I I I ]
  8. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 ]  [  15 I I I I I I 1|I I I I I I I I ]
  9. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 ]  [  15 I I I I I 1|I I I I I I I I I ]
 10. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 ]  [  15 I I I I 1|I I I I I I I I I I ]
 11. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I I I 1|I I I I I I I I I I I ]
 12. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I I 1|I I I I I I I I I I I I ]
 13. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I 1|I I I I I I I I I I I I I ]
 14. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I I I ]
 15. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15|]  [  15 I I I I I I I I I I I I I 1|]
 16. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14|15 ]  [  15 I I I I I I I I I I I I I 1|I ]
 17. [  0  1  2  3  4  5  6  7  8  9 10 11 12 13|14 15 ]  [  15 I I I I I I I I I I I I 1|I I ]
 18. [  0  1  2  3  4  5  6  7  8  9 10 11 12|13 14 15 ]  [  15 I I I I I I I I I I I 1|I I I ]
 19. [  0  1  2  3  4  5  6  7  8  9 10 11|12 13 14 15 ]  [  15 I I I I I I I I I I 1|I I I I ]
 20. [  0  1  2  3  4  5  6  7  8  9 10|11 12 13 14 15 ]  [  15 I I I I I I I I I 1|I I I I I ]
 21. [  0  1  2  3  4  5  6  7  8  9|10 11 12 13 14 15 ]  [  15 I I I I I I I I 1|I I I I I I ]
 22. [  0  1  2  3  4  5  6  7  8| 9 10 11 12 13 14 15 ]  [  15 I I I I I I I 1|I I I I I I I ]
 23. [  0  1  2  3  4  5  6  7| 8  9 10 11 12 13 14 15 ]  [  15 I I I I I I 1|I I I I I I I I ]
 24. [  0  1  2  3  4  5  6| 7  8  9 10 11 12 13 14 15 ]  [  15 I I I I I 1|I I I I I I I I I ]
 25. [  0  1  2  3  4  5| 6  7  8  9 10 11 12 13 14 15 ]  [  15 I I I I 1|I I I I I I I I I I ]
 26. [  0  1  2  3  4| 5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I I I 1|I I I I I I I I I I I ]
 27. [  0  1  2  3| 4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I I 1|I I I I I I I I I I I I ]
 28. [  0  1  2| 3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 I 1|I I I I I I I I I I I I I ]
 29. [  0  1| 2  3  4  5  6  7  8  9 10 11 12 13 14 15 ]  [  15 1|I I I I I I I I I I I I I I ]
```

A.3 Experimento 5

A.3.1 Caso slaves_10f - Experimento 5

```
============================================================
File slaves10f.tim
============================================================
slaves10f -- A model with 12 modules and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2421860 (75683 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 1 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
```

```
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  1| 2  3  4  5  6  7  8  9 10 11  0 ]  [ 40|I I I I I I I I I I 2 ]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [  1 1 1 1 1 1 1 1 1 1 1|I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.3.2 Caso ad14f - Experimento 5

```
============================================================
File ad14f.tim
============================================================
ad14f -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 134137 (4191 Kb)
Chosen cuts: [ 2 11 10 9 9 9 9 9 9 9 9 10 11 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I I ]
  1. [  1  4  5  6  7  8  9 10 11 12 13| 2  3  0 ]  [ 1 I I I I I I I I I I|I I 1 ]
  2. [  2  5  6  7  8  9 10 11 12 13| 0  3  4  1 ]  [ 1 I I I I I I I I I|I I I 1 ]
  3. [  3  6  7  8  9 10 11 12 13| 0  1  4  5  2 ]  [ 1 I I I I I I I I|I I I I 1 ]
  4. [  0  4  7  8  9 10 11 12 13| 1  2  5  6  3 ]  [ I 1 I I I I I I I|I I I I 1 ]
  5. [  0  1  5  8  9 10 11 12 13| 2  3  6  7  4 ]  [ I I 1 I I I I I I|I I I I 1 ]
  6. [  0  1  2  6  9 10 11 12 13| 3  4  7  8  5 ]  [ I I I 1 I I I I I|I I I I 1 ]
  7. [  0  1  2  3  7 10 11 12 13| 4  5  8  9  6 ]  [ I I I I 1 I I I I|I I I I 1 ]
  8. [  0  1  2  3  4  8 11 12 13| 5  6  9 10  7 ]  [ I I I I I 1 I I I|I I I I 1 ]
  9. [  0  1  2  3  4  5  9 12 13| 6  7 10 11  8 ]  [ I I I I I I 1 I I|I I I I 1 ]
 10. [  0  1  2  3  4  5  6 10 13| 7  8 11 12  9 ]  [ I I I I I I I 1 I|I I I I 1 ]
 11. [  0  1  2  3  4  5  6  7 11|13  8  9 12 10 ]  [ I I I I I I I I 1|I I I I 1 ]
 12. [  0  1  2  3  4  5  6  7  8 12|13  9 10 11 ]  [ I I I I I I I I I 1|I I I 1 ]
 13. [  0  1  2  3  4  5  6  7  8  9 13|10 11 12 ]  [ I I I I I I I I I I 1|I I 1 ]
```

A.4 Experimento 6

A.4.1 Caso slaves_10f - Experimento 6

```
============================================================
File slaves10f.tim
============================================================
slaves10f -- A model with 12 modules and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
```

```
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2539918 (79372 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 11 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  2  3  4  5  6  7  8  9 10 11  0| 1 ]  [ I  I I I I I I I I I 2|40]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [ 1  1 1 1 1 1 1 1 1 1 1| I]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.4.2 Caso ad14f - Experimento 6

```
============================================================
File ad14f.tim
============================================================
ad14f -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 730 (22 Kb)
Chosen cuts: [ 2 3 4 5 5 5 5 5 5 5 5 5 4 3 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I I ]
  1. [  2  3  0| 1  4  5  6  7  8  9 10 11 12 13 ]  [ I I 1|1 I I I I I I I I I ]
  2. [  0  3  4  1| 2  5  6  7  8  9 10 11 12 13 ]  [ I I I 1|1 I I I I I I I I ]
  3. [  0  1  4  5  2| 3  6  7  8  9 10 11 12 13 ]  [ I I I I 1|1 I I I I I I I ]
  4. [  1  2  5  6  3| 0  4  7  8  9 10 11 12 13 ]  [ I I I I 1|I 1 I I I I I I ]
  5. [  2  3  6  7  4| 0  1  5  8  9 10 11 12 13 ]  [ I I I I 1|I I 1 I I I I I ]
  6. [  3  4  7  8  5| 0  1  2  6  9 10 11 12 13 ]  [ I I I I 1|I I I 1 I I I I ]
  7. [  4  5  8  9  6| 0  1  2  3  7 10 11 12 13 ]  [ I I I I 1|I I I I 1 I I I ]
  8. [  5  6  9 10  7| 0  1  2  3  4  8 11 12 13 ]  [ I I I I 1|I I I I I 1 I I ]
  9. [  6  7 10 11  8| 0  1  2  3  4  5  9 12 13 ]  [ I I I I 1|I I I I I I 1 I ]
 10. [  7  8 11 12  9| 0  1  2  3  4  5  6 10 13 ]  [ I I I I 1|I I I I I I I 1 ]
 11. [ 13  8  9 12 10| 0  1  2  3  4  5  6  7 11 ]  [ I I I I 1|I I I I I I I I 1 ]
 12. [ 13  9 10 11| 0  1  2  3  4  5  6  7  8 12 ]  [ I I I 1|I I I I I I I I I 1 ]
 13. [ 10 11 12| 0  1  2  3  4  5  6  7  8  9 13 ]  [ I I 1|I I I I I I I I I I 1 ]
```

A.5 Experimento 7

A.5.1 Caso slaves_10f - Experimento 7

```
============================================================
```

```
File slaves10f.tim
============================================================
slaves10f -- A model with 12 modules and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 7145740 (223304 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 12 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I  I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I  I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I  I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3  3|]
 11. [  2  3  4  5  6  7  8  9 10 11  0  1|]  [ I  I I I I I I I I I 2 40|]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [ 1  1 1 1 1 1 1 1 1 1 1| I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I  I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I  I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I  I ]
```

A.5.2 Caso ad14f - Experimento 7

```
============================================================
File ad14f.tim
============================================================
ad14f -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 730 (22 Kb)
Chosen cuts: [ 2 4 5 6 6 6 6 6 6 6 6 6 5 4 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
  0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]  [ 1 1|I I I I I I I I I I I I ]
  1. [  2  3  0  1| 4  5  6  7  8  9 10 11 12 13 ]  [ I I 1 1|I I I I I I I I I I ]
  2. [  0  3  4  1  2| 5  6  7  8  9 10 11 12 13 ]  [ I I I 1 1|I I I I I I I I I ]
  3. [  0  1  4  5  2  3| 6  7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
  4. [  1  2  5  6  3  4| 0  7  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
  5. [  2  3  6  7  4  5| 0  1  8  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
  6. [  3  4  7  8  5  6| 0  1  2  9 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
  7. [  4  5  8  9  6  7| 0  1  2  3 10 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
  8. [  5  6  9 10  7  8| 0  1  2  3  4 11 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
```

```
  9. [  6  7 10 11  8  9| 0  1  2  3  4  5 12 13 ]  [ I I I I 1 1|I I I I I I I I ]
 10. [  7  8 11 12  9 10| 0  1  2  3  4  5  6 13 ]  [ I I I I 1 1|I I I I I I I I ]
 11. [ 13  8  9 12 10 11| 0  1  2  3  4  5  6  7 ]  [ I I I I 1 1|I I I I I I I I ]
 12. [ 13  9 10 11 12| 0  1  2  3  4  5  6  7  8 ]  [ I I I 1 1|I I I I I I I I ]
 13. [ 10 11 12 13| 0  1  2  3  4  5  6  7  8  9 ]  [ I I 1 1|I I I I I I I I ]
```

A.6 Experimento 8

A.6.1 Caso slaves_10f - Experimento 8

```
============================================================
File slaves10f.tim
============================================================
slaves10f -- A model with 12 modules and 23 events
User name: 'propagation'
------------
Problem Size
------------
Product state space:           7263027 states
Reachable state space:         4842017 states
Modules Sizes:                   [ 3 41 3 3 3 3 3 3 3 3 3 3 ]
Modules Sizes after aggregation: [ ]
Current Number of Functions: 2
Size of the Normalized Descriptor: 3551 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 2421860 (75683 Kb)
Chosen cuts: [ 2 2 2 2 2 2 2 2 2 2 12 1 11 2 2 2 2 2 2 2 2 2 2 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) informa
  0. [  0  2| 1  3  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  1. [  0  3| 1  2  4  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  2. [  0  4| 1  2  3  5  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  3. [  0  5| 1  2  3  4  6  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  4. [  0  6| 1  2  3  4  5  7  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  5. [  0  7| 1  2  3  4  5  6  8  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  6. [  0  8| 1  2  3  4  5  6  7  9 10 11 ]  [ 1  1|I I I I I I I I I I ]
  7. [  0  9| 1  2  3  4  5  6  7  8 10 11 ]  [ 1  1|I I I I I I I I I I ]
  8. [  0 10| 1  2  3  4  5  6  7  8  9 11 ]  [ 1  1|I I I I I I I I I I ]
  9. [  0 11| 1  2  3  4  5  6  7  8  9 10 ]  [ 1  1|I I I I I I I I I I ]
 10. [  0  1  2  3  4  5  6  7  8  9 10 11|]  [ 1 41 3 3 3 3 3 3 3 3 3 3|]
 11. [  1| 2  3  4  5  6  7  8  9 10 11  0 ]  [ 40|I I I I I I I I I I 2 ]
 12. [  0  2  3  4  5  6  7  8  9 10 11| 1 ]  [  1 1 1 1 1 1 1 1 1 1 1|I ]
 13. [  1  2| 0  3  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 14. [  1  3| 0  2  4  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 15. [  1  4| 0  2  3  5  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 16. [  1  5| 0  2  3  4  6  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 17. [  1  6| 0  2  3  4  5  7  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 18. [  1  7| 0  2  3  4  5  6  8  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 19. [  1  8| 0  2  3  4  5  6  7  9 10 11 ]  [ 40 2|I I I I I I I I I I ]
 20. [  1  9| 0  2  3  4  5  6  7  8 10 11 ]  [ 40 2|I I I I I I I I I I ]
 21. [  1 10| 0  2  3  4  5  6  7  8  9 11 ]  [ 40 2|I I I I I I I I I I ]
 22. [  1 11| 0  2  3  4  5  6  7  8  9 10 ]  [ 40 2|I I I I I I I I I I ]
```

A.6.2 Caso ad14f - Experimento 8

```
============================================================
File ad14f.tim
============================================================
ad14f -- A model with 14 modules and 14 events
User name: 'Ad'
------------
Problem Size
------------
Product state space:           2125764 states
Reachable state space:             674 states
Modules Sizes:                   [ 2 3 3 3 3 3 3 3 3 3 3 3 3 2 ]
Modules Sizes after aggregation: [ ]
```

```
Current Number of Functions: 13
Size of the Normalized Descriptor: 1041 Kbytes
============================================================
Solution performed: power method with no preconditionning
The multiplication method  use a Split approach (method S)
Extended (E) Probability Vectors
============================================================
Number of AUNFs: 14 (0 Kb)
Chosen cuts: [ 2 1 1 1 1 1 1 1 1 1 1 1 1 1 ]
Synchr. Term Information: evt. [ modules permutation orders and cut (|) information ] [ identities list]
    0. [ 12 13| 0  1  2  3  4  5  6  7  8  9 10 11 ]    [ 1 1|I I I I I I I I I I I I ]
    1. [  1| 4  5  6  7  8  9 10 11 12 13  2  3  0 ]    [ 1|I I I I I I I I I I I I 1 ]
    2. [  2| 5  6  7  8  9 10 11 12 13  0  3  4  1 ]    [ 1|I I I I I I I I I I I I 1 ]
    3. [  3| 6  7  8  9 10 11 12 13  0  1  4  5  2 ]    [ 1|I I I I I I I I I I I I 1 ]
    4. [  4| 0  7  8  9 10 11 12 13  1  2  5  6  3 ]    [ 1|I I I I I I I I I I I I 1 ]
    5. [  5| 0  1  8  9 10 11 12 13  2  3  6  7  4 ]    [ 1|I I I I I I I I I I I I 1 ]
    6. [  6| 0  1  2  9 10 11 12 13  3  4  7  8  5 ]    [ 1|I I I I I I I I I I I I 1 ]
    7. [  7| 0  1  2  3 10 11 12 13  4  5  8  9  6 ]    [ 1|I I I I I I I I I I I I 1 ]
    8. [  8| 0  1  2  3  4 11 12 13  5  6  9 10  7 ]    [ 1|I I I I I I I I I I I I 1 ]
    9. [  9| 0  1  2  3  4  5 12 13  6  7 10 11  8 ]    [ 1|I I I I I I I I I I I I 1 ]
   10. [ 10| 0  1  2  3  4  5  6 13  7  8 11 12  9 ]    [ 1|I I I I I I I I I I I I 1 ]
   11. [ 11| 0  1  2  3  4  5  6  7 13  8  9 12 10 ]    [ 1|I I I I I I I I I I I I 1 ]
   12. [ 12| 0  1  2  3  4  5  6  7  8 13  9 10 11 ]    [ 1|I I I I I I I I I I I I 1 ]
   13. [ 13| 0  1  2  3  4  5  6  7  8  9 10 11 12 ]    [ 1|I I I I I I I I I I I I 1 ]
```

Estes dados encerram as saídas dos experimentos conduzidos, mostrando os trechos dos arquivos de resultado da ferramenta GTAexpress (conforme arquivo de saída *.tim*).