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

ABORDAGENS PARALELAS PARA *MODEL CHECKING* DE REDES DE AUTÔMATOS ESTOCÁSTICOS

LUCAS GIARETTA OLEKSINSKI

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

Orientador: Prof. Dr. Fernando Luís Dotti

Porto Alegre
2013

O45a Oleksinski, Lucas Giaretta
Abordagens paralelas para Model Checking de redes de autômatos estocáticos / Lucas Giaretta Oleksinski. – Porto Alegre, 2013.
123 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Fernando Luís Dotti.

1. Informática. 2. Redes de Autômatos Estocásticos. 3. Lógica Temporal (Computação). 4. Simulação e Modelagem em Computadores. I. Dotti, Fernando Luís. 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 DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Abordagens Paralelas para *Model Checking* de Redes de Autômatos Estocásticos" apresentada por Lucas Giaretta Oleksinski como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 25/03/2013 pela Comissão Examinadora:

Prof. Dr. Fernando Luís Dotti – Orientador — PPGCC/PUCRS

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

Prof. Dr. Afonso Henrique Correa de Sales – FACIN/PUCRS

Prof. Dr. Osmar Marchi dos Santos – UFSM

Homologada em 09/07/2013, conforme Ata No. 012 pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

DEDICATÓRIA

Dedico este trabalho à meus pais por todo apoio prestado em todos dias da minha vida.

“Os desafios não são difíceis porque tentamos, é por não tentarmos que são difíceis.”
(Séneca)

AGRADECIMENTOS

Chega ao fim mais uma etapa, meus caros amigos! O tempo voa, não poupa nada, não poupa ninguém. Simples palavras de agradecimento não conseguem expressar o sentimento, a convivência com pessoas que nos fazem crescer, mirar o futuro, instigar e ao mesmo tempo desafiar o conhecimento.

Agradeço as pessoas abaixo listadas:

Primeiramente, aos professores Fernando Dotti, Paulo Fernandes e Afonso Sales, pela grande oportunidade, por todos os conselhos dados no decorrer do curso, por tornarem as reuniões de grupo agradáveis e, principalmente, pela amizade.

Em especial ao professor Fernando Dotti (orientador), por estar sempre disposto a debater novas ideias e ter sido um orientador muito presente.

Aos colegas de curso: Bernardo Estácio, Claiton Correa, Eli Maruani, Eduardo Spies, Guilherme Madalozzo, Lucas Hilgert, Miguel Xavier, Timóteo Lange, Tiago Paes, Silvana Teodoro e Viviane Lara por todos os momentos sejam de diversão ou estudo, dicas, e principalmente pela parceria.

Ao pessoal do projeto Paleoprospec: Joaquim Assunção, Daiane Hemerich, Gabriel Couto, Luciana Espindola, Leonardo Peres e Maria Pível por todos momentos agradáveis e ensinamentos recebidos.

Em especial a todos os familiares e a Flavia Betoni (te amo :D) pela força recebida em todos os momentos sejam de alegria ou preocupação.

Vamos tricolor! Dá-lhe tricolor!

Fica aqui um registro sincero de agradecimento.

ABORDAGENS PARALELAS PARA *MODEL CHECKING* DE REDES DE AUTÔMATOS ESTOCÁSTICOS

RESUMO

O emprego de sistemas complexos e críticos para automação de tarefas do cotidiano faz crescer a dependência das pessoas, gerando desconforto em relação à segurança de tais sistemas. Nos últimos anos algumas técnicas têm sido desenvolvidas visando facilitar as atividades relacionadas à validação de projetos nos estágios iniciais do ciclo de desenvolvimento. Verificação de modelos é uma técnica formal automática que permite a verificação de sistemas concorrentes de estados finitos sob propriedades descritas em lógicas temporais através do emprego de algoritmos que avaliam exaustivamente o sistema sob consideração. Entretanto, esta técnica é custosa no que tange ao armazenamento em memória e processamento, justificando o desenvolvimento de algoritmos paralelos e distribuídos para poderosos agregados computacionais. Esta dissertação relata o estudo e desenvolvimento de algoritmos de verificação de modelos descritos em Redes de Autômatos Estocásticos e propriedades descritas na lógica temporal *Computation Tree Logic* para ambientes que endereçam espaços de memória de maneira distribuída.

Palavras Chave: Redes de Autômatos Estocásticos, Lógica Temporal Ramificada, Verificação Formal, Verificação de Modelos Paralela e Distribuída.

Trabalho parcialmente suportado pelos projetos FAPERGS PqG 1014867 e CNPq 560036/2010-8 "Verificação de Modelos Descritos em Redes de Autômatos Estocásticos".

PARALLEL APPROACHES FOR MODEL CHECKING STOCHASTIC AUTOMATA NETWORKS

ABSTRACT

The use of critical and complex systems at automation of daily tasks increases the people's dependence, generating unease about the safety of such systems. In the last years several techniques have been developed to facilitate activities related to design validation in the early stages of the development cycle. Model Checking is an automatic formal technique that allows verification of finite-state concurrent systems under properties described in temporal logics by employing verification algorithms that exhaustively assess the correctness of the system under consideration. Indeed, this technique is costly with respect to storage and processing, justifying the development of parallel and distributed algorithms for powerful computing clusters. This dissertation reports the study and development of verification algorithms for models described in Stochastic Automata Networks and properties written in Computation Tree Logic temporal logic for environments that address memory spaces in a distributed way.

Keywords: Stochastic Automata Networks, Computation Tree Logic, Formal Verification, Parallel and Distributed Model Checking.

This work is partially sponsored by research projects FAPERGS PqG 1014867 and CNPq 560036/2010-8 "Model Checking Stochastic Automata Networks".

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

GB – Gigabyte

MB – Megabyte

KB – Kilobyte

SAN – *Stochastic Automata Networks*

PSS – *Product State Space*

RSS – *Reachable State Space*

BDD – *Binary Decision Diagrams*

MDD – *Multivalued Decision Diagrams*

LTL – *Linear Temporal Logic*

CTL – *Computation Tree Logic*

CTL* – *Extended Computation Tree Logic*

AP – *Atomic Proposition*

TS – *Transition System*

MPI – Message Passing Interface

LAD – Laboratório de Alto Desempenho

PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul

SMBTL – *Shared-Memory Byte Transfer Layer*

SUMÁRIO

1. INTRODUÇÃO

O emprego de sistemas complexos e críticos para automatização de tarefas traz grandes benefícios para a sociedade. Entretanto, a dependência das pessoas nestes sistemas gera desconforto em relação à corretude e segurança dos mesmos. Erros em sistemas de *hardware* e *software* críticos são conhecidos e têm causado danos tanto de cunho financeiro como à integridada humana. Um exemplo clássico de erro de projeto de tais sistemas é o erro de software no foguete $Ariane\ 5$ o qual causou sua destruição logo após a decolagem gerando um prejuízo em torno de US$ 500 milhões [2].

No decorrer dos anos, algumas técnicas têm sido desenvolvidas visando amenizar as dificuldades relacionadas à validação de complexos sistemas. *Model Checking* (Verificação de Modelos) é uma técnica formal que permite a verificação de sistemas concorrentes de estados finitos através de exaustiva exploração do conjunto de estados que representa uma realidade, tal como o funcionamento de um algoritmo ou protocolo de comunicação [18]. Um atrativo interessante que a técnica apresenta é permitir a condução dos passos relacionados à verificação de forma automática sem necessidade de intervenção do usuário. Devido aos interessantes aspectos apresentados, alguns formalismos de modelagem têm recebido ferramental de suporte, como Cadeias de Markov (verificador PRISM [24]), Redes de Petri Estocásticas (verificador SMART [16]) e Álgebra de Processos (verificador FDR [11]).

Redes de Autômatos Estocásticos (*Stochastic Automata Networks* - SAN) é um formalismo Markoviano estruturado focado na modelagem de sistemas concorrentes com grandes espaços de estados [41] primeiramente voltado à avaliação de desempenho. Em SAN, complexas realidades podem ser modeladas através de um conjunto de subsistemas que ocasionalmente interagem. Cada subsistema é definido através de autômatos os quais são compostos por estados e as respectivas transições entre estes estados que podem ser guiadas por eventos locais ou sincronizantes.

Como principal diferencial em relação a outros formalismos de modelagem, SAN oferece a vantagem de permitir o uso de funções para avaliação de estados globais de um modelo, habilitando a definição de dependências entre autômatos e a descrição de comportamentos complexos de maneira compacta [10]. O formalismo é empregado para modelagem de diferentes realidades, como escalonamento de processos em arquiteturas NUMA [14], programas paralelos mestre/escravo [4], redes *wireless* Ad Hoc [21], modelos de linhas de produção [23], dentre outros[1].

Apesar dos interessantes aspectos de modelagem fornecidos e a existência de ferramental especializado para avaliação de desempenho, a ausência de ferramentas de verificação dificulta a validação de modelos SAN. Visando suprir tal carência, esforços foram investidos através de um trabalho cooperativo na construção de um verificador de propriedades descritas na lógica temporal *Computation Tree Logic* (CTL), tendo os primeiros resultados gerados discutidos em [19].

[1] Mais exemplos podem ser encontrados em: http://www.inf.pucrs.br/peg

Embora algumas técnicas tenham sido desenvolvidas ao longo dos anos permitindo a verificação de complexos sistemas, tal como uso de simetrias [47], redução de ordem parcial [36] e representação e manipulação simbólica de espaços de estados [37], um fator limitante à técnica de verificação de modelos é a conhecida explosão do espaço de estados. Este problema se deve ao crescimento exponencial na estrutura que representa o sistema sob consideração e a falta de recursos computacionais para manipular grandes quantidades de informação [5].

Devido aos constantes avanços pela indústria, o acesso à agregados computacionais têm permitido o desenvolvimento de algoritmos paralelos para ambientes que endereçam espaços de memória de forma distribuída ou compartilhada [29, 5, 6, 48, 28]. Desta maneira, construir algoritmos que permitam a verificação de complexas realidades motiva o estudo e desenvolvimento de novas pesquisas na área.

1.1 Motivação e Objetivos

Grande parte das abordagens de verificação paralela foca em ambientes com processamento e memória distribuídos onde atribui-se pequenos conjuntos disjuntos de estados para diferentes máquinas para computação em paralelo. Entretanto, a exploração de espaços de estados distribuídos pode sofrer de sobrecarga de comunicação, uma vez que estados que sejam relacionados através de transições podem estar localizados em diferentes partições [44]. Desta maneira, o cálculo de estados sucessores ou antecessores de um dado estado pode ser um gargalo, uma vez que sejam geradas partições do espaço de estados sem considerar as relações entre os estados, configurando uma distribuição de pouca qualidade.

Embora o emprego de ambientes com memória distribuída permita o armazenamento de abstrações relativas à complexas realidades, representar o espaço de estados de um modelo em sua íntegra possibilita conduzir localmente o cálculo de estados antecessores/sucessores de um dado estado. Ao se considerar que a verificação de determinados operadores que compõem a lógica temporal CTL possuem algoritmos de verificação extremamente custosos devido à necessidade de cálculo de ponto fixo, o qual é obtido através de exaustiva avaliação do conjunto de estados que representa uma realidade, estando o espaço de estados integralmente disponível em um conjunto de máquinas habilita-se as mesmas a explorar conjuntos de estados distintos de maneira quase independente, de modo que comunicação e sincronismo são necessários somente para obtenção de ponto fixo.

De outro lado, ferramentas de verificação de modelos que aceitam como entrada propriedades da lógica temporal *Computation Tree Logic* normalmente implementam algoritmos de verificação para fórmulas representadas em um formato mínimo, o qual permite derivação de toda e qualquer fórmula CTL válida. Apesar desta vantagem simplificar a implementação de ferramentas de verificação, a transcrição de fórmulas para este formato aumenta consideravelmente o tamanho da fórmula em questão [2], consequentemente aumentando o tempo de verificação.

A adoção de ambientes distribuídos aliado às características da lógica temporal adotada motiva a pesquisa e desenvolvimento de novas técnicas de verificação. Neste contexto, esta dissertação de mestrado objetiva a proposta, implementação, validação e experimentação de duas abordagens para estes ambientes, sendo:

- Distribuição da verificação dos operadores de maior custo computacional pertencentes a um formato mínimo de escrita de fórmulas CTL válidas. O ganho de desempenho é focado na condução distribuída de cada iteração necessária à obtenção de ponto fixo, onde cada máquina empregada é responsável pela computação de um subconjunto de estados do conjunto em consideração;
- Desenvolvimento de um algoritmo de análise e escolha de particionamento da árvore sintática que representa propriedades CTL escritas no formato aceito pelo algoritmo de verificação adotado e posterior distribuição das partições para computação em paralelo. Para cada partição identificada, um grupo de máquinas conduz a verificação distribuída dos operadores de maior custo computacional presentes na partição através do emprego da primeira abordagem paralela.

Para ambas as abordagens, réplicas da estrutura simbólica que representa o espaço de estados gerado pelo formalismo SAN e respectiva função de transição são feitas, possibilitando a condução de computações distribuídas de maneira a realizar troca de mensagens somente para obtenção de ponto fixo. Embora esta abordagem pareça pouco eficiente quanto à consumo de memória, ganhos são discutidos no decorrer do trabalho em razão da distribuição das tarefas relacionadas à computação dos operadores de lógica temporal, estando estas tarefas diretamente relacionadas ao consumo.

1.2 Organização do Trabalho

Após a introdução do trabalho, o Capítulo 2 apresenta o embasamento teórico necessário ao entendimento dos conceitos que compreendem a pesquisa em si, detalhando Redes de Autômatos Estocásticos, Lógicas Temporais, Diagramas de Decisão, Verificação de Modelos e Verificação de Modelos Paralela e Distribuída. O Capítulo 3 se limita a descrever a arquitetura da ferramenta de verificação para modelos SAN inicialmente desenvolvida, bem como, detalhar o algoritmo sequencial de verificação adotado. Na sequência, o Capítulo 4 descreve detalhadamente as abordagens paralelas propostas neste trabalho. O Capítulo 5 descreve os experimentos conduzidos sob diferentes modelos SAN e discute os ganhos de desempenho obtidos. Por fim, o Capítulo 6 apresenta as considerações finais acerca do trabalho desenvolvido.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica necessária ao entendimento dos conceitos que compreendem este trabalho. Sendo assim, Redes de Autômatos Estocásticos, Diagramas de Decisão, Lógicas Temporais, Verificação de Modelos e Verificação de Modelos Paralela e Distribuída são descritos.

2.1 Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos - *Stochastic Automata Networks* (SAN) - é um formalismo Markoviano estruturado para modelagem de sistemas com grandes espaços de estados [41]. A idéia central deste formalismo é permitir a modelagem de sistemas como um conjunto de subsistemas que possuem comportamento independente e ocasional interação. Possui equivalência de descrição com Cadeias de Markov [50] de forma que é possível representar toda Cadeia de Markov através de um modelo SAN.

Em SAN, modelos são descritos através de um conjunto de autômatos os quais possuem um número finito de estados, chamados estados locais, e as respectivas transições entre estes estados. As transições entre estados possuem uma lista de eventos que podem disparar estas transições, podendo ser eventos locais ou sincronizantes.

Os eventos que ocorrem em um único autômato, ou seja, eventos que alterem somente o estado de um determinado autômato não interferindo nos demais autômatos da rede, são chamados eventos locais. De outro lado, eventos sincronizantes alteram simultaneamente os estados locais dos autômatos envolvidos [22] permitindo a construção de dependência entre estes.

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 [20]. Uma taxa constante é uma frequência média com a qual a transição ocorre. O uso de taxas funcionais também permite a criação de dependência entre autômatos sendo definidas por funções que são avaliadas de acordo com os estados dos autômatos envolvidos. O uso de funções é a principal característica que difere SAN em relação a outros formalismos de modelagem, tais como Redes de Petri [3] e Cadeias de Markov [49].

O estado global de um modelo SAN é dado pela concatenação dos estados locais dos autômatos da rede. Conforme descrito anteriormente, devido à presença de dependência entre autômatos gerada pela existência de eventos sincronizantes e taxas funcionais, há a possibilidade de haver estados globais que não possam ser atingidos dentro do espaço produto (*Product State Space* (PSS) - Espaço de Estados Produto). Dessa forma, é necessário definir uma função *booleana* parcial de atingibilidade ou *partial reachability* que é usada para definição dos estados iniciais do modelo os quais são utilizados como ponto de partida para geração do Espaço de Estados Atingíveis ou *Reachable State Space* (RSS).

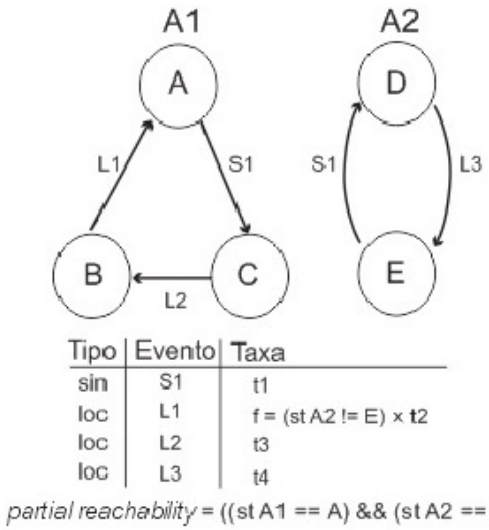


Figura 2.1 – SAN com eventos locais, sincronizantes e taxas funcionais.

Para ilustrar o formalismo SAN, considere a Figura 2.1 a qual apresenta um modelo simples com dois autômatos, $A1$ e $A2$. As transições $L2$ e $L3$ são guiadas por eventos locais alterando somente os estados locais destes autômatos de maneira independente, possuindo taxas constantes $t3$ e $t4$, respectivamente. A transição $S1$ é guiada por um evento sincronizante e altera simultaneamente o estado dos dois autômatos, com taxa $t1$. Por fim, a transição identificada por $L1$ é guiada por um evento local possuindo taxa funcional sendo disparada com taxa $t2$ caso o autômato $A2$ esteja em um estado diferente de E. Caso contrário, a transição não ocorrerá. Exemplos de modelos SAN os quais serão utilizados na condução dos experimentos (Capítulo 5) podem ser encontrados no Apêndice A.

2.2 Diagramas de Decisão

O uso de diagramas de decisão tem ganhado bastante atenção no contexto de modelagem e verificação de circuitos digitais e *software*. A representação e manipulação de espaços de estados utilizando estas estruturas de dados, dentre outras técnicas de redução do problema da explosão do espaço de estados, levou ao surgimento do conceito de Verificação de Modelos Simbólica [18]. Dentre os tipos de diagramas de decisão pode-se citar os *Binary Decision Diagrams* (BDD) ou Diagramas de Decisão Binária, *Multivalued Decision Diagrams* (MDD) ou Diagramas de Decisão Multi-valorada, entre outros.

Diagramas de Decisão Binária (BDD) são representações compactas de funções *booleanas* em grafos acíclicos e direcionados onde a representação de informações redundantes é completamente evitada. Esta estrutura de dados possui dois tipos de vértices, sendo um terminal e o outro não-terminal. Cada vértice não-terminal é rotulado por uma variável v e, como o diagrama é binário, pode possuir até 2 sucessores, $low(v)$ onde v tem valor 0 e $high(v)$, onde v tem valor 1 [12, 18]. A Figura

2.2 apresenta um diagrama de decisão binária inicialmente com redundância e a sua representação final após remoção dos nodos duplicados.

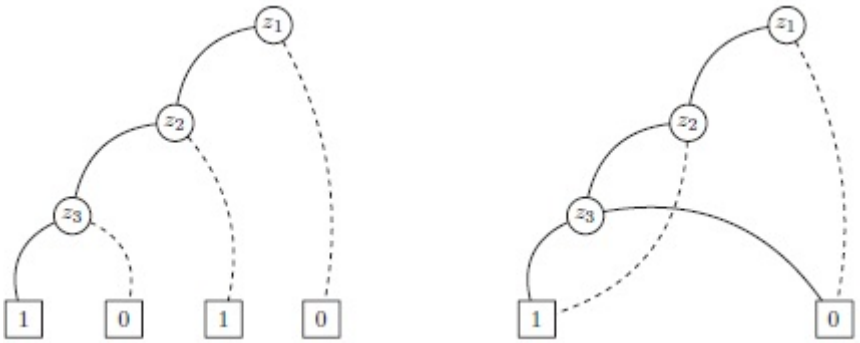


Figura 2.2 – Exemplo de Diagramas de Decisão Binária [2].

De outro lado, Diagramas de Decisão Multi-valorada (MDD) são multi-grafos acíclicos dirigidos com arestas rotuladas as quais aceitam múltiplos valores para ligação entre os nodos. A diferença básica comparada aos diagramas de decisão binária é permitir a representação de um conjunto mais abrangente de informações, possuindo as seguintes propriedades [17]:

- Nodos são organizados em $N + 1$ níveis, onde N é o número de componentes do sistema podendo ser o número de autômatos de um modelo SAN, por exemplo;
- Nível N possui apenas um nodo não-terminal r (nodo *root*), enquanto do nível $N - 1$ até o nível 1 há um ou mais nodos não-terminais;
- Nível 0 tem dois nodos terminais: 0 e 1;
- Um nodo não-terminal p em um nível l contém n_l arcos apontando para nodos no nível *l* - 1;
- Não há duplicação de nodos com características semelhantes.

As propriedades acima descritas podem ser visualizadas tomando a Figura 2.3 como base, a qual representa a codificação do espaço de estados atingível de um modelo SAN do problema do Jantar dos Filósofos para três filósofos, sendo este último detalhado no Apêndice A.1. Entretanto, como os estados são armazenados de maneira simbólica, é necessário conhecer o significado dos dados que os compõe: o valor 0 indica que o filósofo i está no estado $Pensando$, 1 indica que o filósofo está no estado $QuerComer$ e 2 indica que o filósofo está no estado $Comendo$, respectivamente.

Conforme descrito na Seção 2.1, um estado global de um modelo SAN é dado pela concatenação dos estados locais dos autômatos do modelo. Dessa maneira, para representação de um estado global para este modelo, como por exemplo, todos autômatos estarem no estado $Pensando$, o respectivo estado estaria codificado na estrutura MDD da seguinte maneira: (000), onde cada valor de índice i corresponde ao estado local de um autômato que representa um filósofo j.

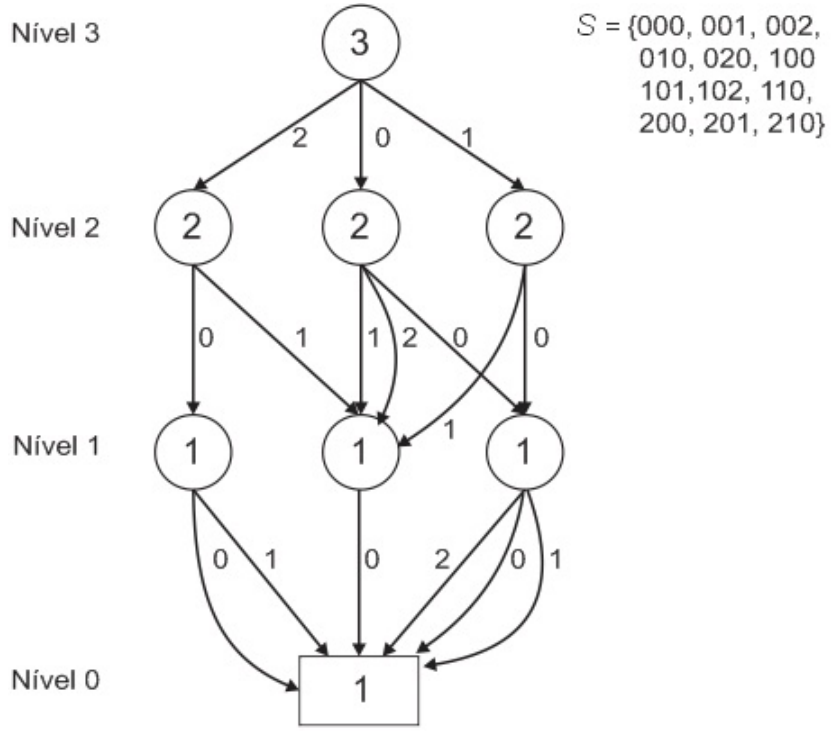


Figura 2.3 – Exemplo de Diagrama de Decisão Multi-valorada que codifica o RSS de um modelo SAN do Jantar dos Filósofos para três filósofos.

2.3 Lógicas Temporais

Lógica temporal é um formalismo matemático que pode ser utilizado para descrição de sequências de transições entre estados em sistemas reativos sem fazer menção explícita ao tempo. Basicamente, uma fórmula da lógica temporal define uma relação de ordem entre estados globais de um sistema que obedecem certas propriedades. O seu emprego para expressão de propriedades sobre modelos de estados finitos tem ganhado importância no decorrer dos anos. Inúmeras ferramentas de verificação de modelos têm sido desenvolvidas trazendo o benefício do uso de lógicas temporais para eficiente verificação de sistemas e um conjunto muito importante de propriedades [18].

Dentre as lógicas temporais pode-se citar a *Linear Temporal Logic* (LTL), *Computation Tree Logic* (CTL) e *Extended Computation Tree Logic* (CTL*), as quais diferem basicamente pela noção de tempo que abrangem. Em resumo, a LTL trabalha sobre uma perspectiva de tempo linear apresentando a vantagem de expressar propriedades sobre um único caminho de computação. A CTL, por sua vez, apresenta a vantagem de permitir a expressão de propriedades sobre um ou todos os caminhos de uma estrutura de estados ramificada. Por fim, a CTL* possui comportamento similar à CTL, porém é mais abrangente no que tange a expressão de propriedades permitindo a combinação de características de ambas as lógicas, LTL e CTL [2]. Este trabalho adota a lógica temporal *Computation Tree Logic* para expressão de propriedades sobre modelos SAN, sendo detalhada a seguir.

A *Computation Tree Logic* é uma lógica temporal baseada na lógica proposicional que abrange uma noção ramificada de tempo e tem por objetivo permitir a expressão de propriedades sobre o comportamento de sistemas. Quanto à escrita de propriedades, as fórmulas CTL permitem que se expresse propriedades de algumas ou todas as computações que iniciem em um determinado estado de um sistema, sendo compostas por quantificadores de caminho, operadores temporais e proposições atômicas.

Quantificadores de caminho são utilizados para determinar o ramo da árvore de estados que será abrangido. Há dois quantificadores de caminho distintos: $\forall$ (pronunciado "para todos os caminhos") também representado por A, indicando que todos os caminhos de computação possíveis devem respeitar determinada condição, e $\exists$ (pronunciado "existe um caminho") também representado por E, que indica que determinada condição será atendida em pelo menos um ramo da estrutura ramificada de estados.

Os operadores temporais, por sua vez, têm por objetivo indicar a ordenação temporal de ocorrência de eventos sobre um ramo da árvore de estados. Basicamente, a CTL é composta por quatro operadores temporais, sendo:

- $\bigcirc$ - Chamado *Next*, simbolizando "no próximo estado", também representado por X. Este operador indica a ocorrência de determinada condição em um estado sucessor ao estado em que se encontra o sistema em determinado momento;
- $\Diamond$ - Chamado *Eventually*, simbolizando "em um estado futuro", também representado por F. Este operador indica a ocorrência de determinada condição em um estado futuro ao estado em que se encontra o sistema em dado momento;
- $\Box$ - Chamado *Globally*, simbolizando "globalmente", também representado por G. Este operador indica a ocorrência de determinada condição globalmente, ou seja, em todos os estados sucessores do estado atual do sistema;
- $\cup$ - Chamado *Until*, simbolizando "até que". Este operador indica a precedência entre eventos em um dado momento, indicando que determinada condição deve ser verdade até o momento em que outra condição venha a ser satisfeita.

Por fim, as proposições atômicas - *Atomic Propositions* (AP) - são usadas para formalizar condições de interesse nos estados do sistema sob consideração podendo ser definidas como expressões as quais tem o objetivo de descrever um determinado comportamento sobre um modelo. Além dos quantificadores de caminho, operadores temporais e proposições atômicas, a composição de fórmulas pode ser feita utilizando conectores booleanos da lógica proposicional [27], tais como: a conjunção ($\wedge$), negação ($\neg$) e a implicação ($\rightarrow$).

Uma fórmula CTL válida possui sintaxe mínima definida:

$$\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

A gramática acima define o que se chama de fórmulas estado (*state formulae*) que são afirmações sobre as proposições atômicas nos estados e suas estruturas ramificadas, onde:

- a pertence ao conjunto das proposições atômicas AP.
- φ simboliza fórmulas caminho (*path formulae*) as quais expressam propriedades temporais de caminhos, sendo formadas de acordo com a seguinte gramática mínima:

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \cup \Phi_2$$

Onde:

- Φ, Φ_1 e Φ_2 são fórmulas estado.

O operador $\cup$ em conjunto com conectores booleanos ainda permite a derivação dos operadores $\Diamond$ e $\Box$, da sequinte maneira:

$$\Diamond\varphi \equiv true \cup \varphi \qquad\qquad \Box\varphi \equiv \neg\Diamond\neg\varphi$$

Intuitivamente, fórmulas estado expressam propriedades de um único estado, enquanto fórmulas caminho expressam propriedades de um caminho o qual pode ser uma sequência infinita de estados [2]. Exemplos de fórmulas CTL e seus comportamentos temporais sobre estruturas ramificadas de tempo podem ser visualizados na Figura 2.4. Mais detalhes sobre a sintaxe e semântica da lógica temporal *Computation Tree Logic* são encontrados em [27].

De acordo com Baier e Katoen [2] e Clarke, Grumberg e Peled [18], fórmulas CTL são interpretadas sobre estados e caminhos de um sistema de transição TS. A relação de satisfação para CTL é definida da seguinte maneira: sendo TS = (S, Act, $\rightarrow$, I, AP, L) um sistema de transição de estados sem estados terminais tal que:

S é um conjunto de estados;

Act é um conjunto de ações;

$\rightarrow \subseteq S$ x Act x S é a relação de transição;

$I \subseteq S$ é um conjunto de estados iniciais;

AP é o conjunto de proposições atômicas;

$L : S \rightarrow 2^{AP}$ é a função de rotulação.

e tendo-se uma proposição atômica a tal que $a \in AP$, um estado $s \in S$, Φ e Ψ sendo fórmulas CTL válidas, e φ sendo uma fórmula CTL caminho, a relação de satisfação $\models$ para fórmulas estado é definida por:

$s \models a$ if a $\in L(s)$

$s \models \neg\Phi$ if $not\ s \models \Phi$

$s \models \Phi \wedge \Psi$ if $(s \models \Phi)$ e $(s \models \Psi)$

$s \models \exists\varphi$ if $\pi \models \varphi$ para algum $\pi \in Paths(s)$

$s \models \forall\varphi$ if $\pi \models \varphi$ para todos $\pi \in Paths(s)$

Para um caminho π, a relação de satisfação $\models$ para fórmulas caminho é definida por:

$\pi \models \bigcirc\Phi$ if $\pi[1] \models \Phi$

$\pi \models \Phi \cup \Psi$ if $\exists j \geq 0.\ (\pi[j] \models \Psi \wedge (\forall 0 \leq k < j.\pi[k] \models \Phi))$

onde para um caminho $\pi = s_0 s_1 s_2 ...$ e um inteiro $i \geq 0$, $\pi[i]$ denota o $(i+1)$-ésimo estado de π, isto é, $\pi[i] = s_i$ [2].

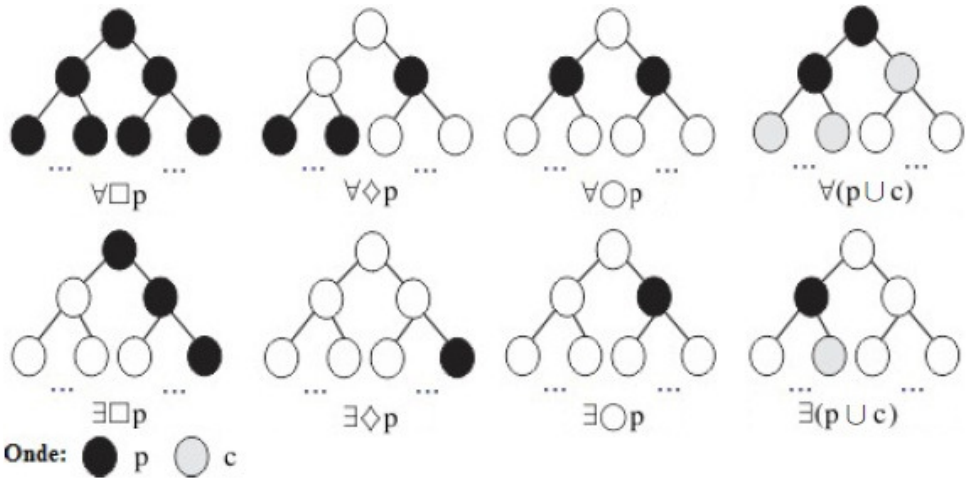


Figura 2.4 – Visualização da semântica de algumas fórmulas CTL básicas.

A Figura 2.4 apresenta algumas fórmulas CTL simples, sendo semanticamente assim descritas:

- $\forall\Box(p)$ - Para todos os caminhos, em todos os estados, p deve ser satisfeito. Esta propriedade denota um comportamento global em todos caminhos e estados possíveis da computação.
- $\forall\Diamond(p)$ - Para todos os caminhos, no futuro, p deve ser satisfeito. A propriedade denota a existência de um ou mais estados que satisfaçam p em dado momento, para todos caminhos de computação.
- $\forall\bigcirc(p)$ - Para todos os caminhos, no próximo estado, p deve ser satisfeito. A propriedade descreve um comportamento em um estado sucessor de um dado estado para todos caminhos de computação.
- $\forall(p \cup c)$ - Para todos os caminhos, p é verdade até que c seja satisfeito. A propriedade demonstra a precedência na ocorrência de um evento p para que o evento c ocorra, sendo c um estado sucessor de p. A sintaxe deste operador indica ainda que, sendo c válido em todos

os caminhos, não é necessária a ocorrência de p nestes mesmos caminhos para que a fórmula seja válida.

- $\exists\Box(p)$ - Existe um caminho na árvore de estados em que em todos estados p é satisfeito. A propriedade demonstra a ocorrência de pelo menos um caminho de computação onde p é sempre verdade.
- $\exists\Diamond(p)$ - Existe um caminho onde, no futuro, em pelo menos um estado p é verdade. Para a propriedade ser válida, deve haver pelo menos um estado que satisfaz p na árvore de estados, não importando o ramo.
- $\exists\bigcirc(p)$ - Existe um caminho onde, no próximo estado, p deve ser satisfeito. Propriedade válida somente para os estados que possuam como sucessores estados que satisfazem p.
- $\exists(p \cup c)$ - Existe um caminho onde p é satisfeito até que c venha a ser válido. Fórmula válida em pelo menos um caminho da árvore onde, para c ser satisfeito, p deve ser satisfeito em pelo menos um estado antecessor direto de c.

A CTL possui formatos de equivalência para escrita de propriedades. A *Existential Normal Form* (ENF) ou *Forma Normal Existencial*, a qual é um formato comumente aceito por ferramentas de verificação, permite a escrita de toda fórmula CTL válida através do emprego do quantificador de caminho existencial $\exists$ e leis de equivalência e operadores da lógica proposicional.

Segundo Baier e Katoen [2], os operadores básicos $\exists\bigcirc$, $\exists\Box$ e $\exists\cup$ são suficientes para definir toda a sintaxe desta lógica temporal. Tendo a pertencendo ao conjunto das proposições atômicas AP, o conjunto de fórmulas CTL em Forma Normal Existencial pode ser definido através da seguinte sintaxe:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists \bigcirc \Phi \mid \exists(\Phi_1 \cup \Phi_2) \mid \exists\Box\Phi$$

Para cada fórmula CTL, existe uma fórmula equivalente em ENF. Para prova deste teorema é necessário levar em consideração algumas leis de dualidade que permitem a eliminação do quantificador universal [2]:

$$\forall \bigcirc \Phi \equiv \neg\exists \bigcirc \neg\Phi \tag{2.1}$$

$$\forall(\Phi \cup \Psi) \equiv \neg\exists(\neg\Psi \cup (\neg\Phi \wedge \neg\Psi)) \wedge \neg\exists\Box\neg\Psi \tag{2.2}$$

Quando se implementa a tradução de uma fórmula CTL para ENF também pode-se usar regras análogas para os operadores derivados, tais como:

$$\forall\Diamond\Phi \equiv \neg\exists\Box\neg\Phi \tag{2.3}$$

$$\forall\Box\Phi \equiv \neg\exists\Diamond\neg\Phi = \neg\exists(\mathit{true} \cup \Phi) \tag{2.4}$$

Quando da transcrição de uma fórmula CTL para Forma Normal Existencial através das regras acima, a fórmula resultante apresenta crescimento quase que exponencial aumentando consideravelmente o tempo de verificação de um modelo, sendo este linear ao tamanho da fórmula [2]. Para exemplificação deste comportamento, basta considerar a tradução da fórmula $\forall(\Phi \cup \Psi)$ a qual triplica a ocorrência da fórmula à direita (Ψ), conforme visto na equação 2.2.

2.4 Verificação de Modelos

Nas últimas décadas algumas técnicas tem sido empregadas para minimizar o trabalho relacionado a garantir correta especificação de sistemas nos estágios iniciais do ciclo de desenvolvimento, tais como: simulação, testes e verificação formal de *hardware* e *software*. Enquanto simulação e testes exploram alguns comportamentos possíveis e cenários desejados de um sistema, porém não provando a ausência de erros, técnicas de verificação formal como *Model Checking* conduzem exaustiva exploração de todos os comportamentos possíveis de um sistema [18].

Comparada à outras abordagens, esta técnica apresenta a vantagem de ser totalmente automática não requerendo supervisão do usuário nem conhecimentos matemáticos avançados. Ainda, a possibilidade de geração de contra-exemplos e testemunhas é outra característica interessante da mesma, as quais descrevem importantes informações para depuração do modelo sob verificação.

Neste contexto, falhando a verificação de uma propriedade, um contra-exemplo é gerado para o usuário demonstrando um comportamento inadequado (*error-trace*) presente no modelo. Segundo Clarke *et al.* [18], utilizando-se a lógica temporal CTL para descrição de propriedades e sendo a fórmula composta por um quantificador de caminho universal, a ferramenta deve gerar um caminho de computação em que a negação da fórmula é verdade. Em outras palavras, por exemplo, se a verificação da fórmula $\forall\Box f$ retornar falso, a ferramenta deve produzir um caminho onde $\exists\Diamond\neg f$ é satisfeito.

Caso a verificação retorne um resultado positivo, uma testemunha é gerada demonstrando um caminho de computação que ratifica a propriedade. Similarmente, sendo verdadeira a verificação de uma fórmula quantificada existencialmente como, por exemplo, $\exists\Diamond f$, a ferramenta deve produzir um caminho para um estado em que f seja satisfeito.

O trabalho do usuário/especificador de uma ferramenta de verificação pode ser descrito em um conjunto bem definido de passos: *modelagem*, *especificação* e *verificação*, conforme demonstrado na Figura 2.5.

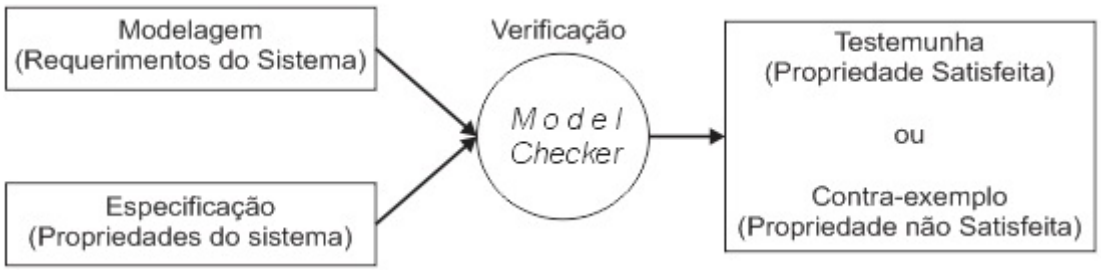


Figura 2.5 – A abordagem de Verificação de Modelos.

Como primeira tarefa, a modelagem tem o objetivo de que se converta uma determinada realidade, tal como o funcionamento de um algoritmo, para um formalismo que seja aceito por uma ferramenta de verificação, podendo esta etapa ainda ser conhecida como uma tarefa de compilação.

Após a modelagem, é necessário que sejam elencadas propriedades as quais se deseja verificar. Esta especificação normalmente é feita em algum formalismo lógico. Para sistemas de *hardware* e *software*, comumente usa-se lógicas temporais, já abordadas na Seção 2.3, que permitem a descrição de afirmativas sobre a evolução do comportamento de um sistema no decorrer do tempo, sem a necessidade de introduzir tempo explicitamente [2].

Por fim, algoritmos de verificação avaliam exaustivamente os estados do modelo de entrada em busca de estados que satisfaçam as propriedades elencadas na fase de especificação. A verificação, apesar de ser automática, envolve assistência humana para análise dos resultados gerados, sendo esta etapa a interpretação das testemunhas e contra-exemplos gerados pela ferramenta adotada. Para melhor entendimento desta fase, a descrição de um algoritmo de verificação de propriedades CTL é feita no Capítulo 3.

Model Checking é uma técnica que pode ainda ser descrita de maneira formal. Dada uma estrutura de *Kripke* $M = (S, S_0, R, L)$ onde:

1. S é um conjunto finito de estados atingíveis de um modelo;

2. $S_0 \subseteq S$ é o conjunto de estados iniciais;

3. $R \subseteq S \times S$ é a relação de transição a qual deve ser total, ou seja, para todo estado $s \in S$ deve haver um estado $s' \in S$ tal que $R(s, s')$;

4. $L : S \to 2^{AP}$ é a função que rotula os estados com o conjunto de proposições atômicas satisfeitas em cada estado.

encontrar todos os estados de S que sejam satisfeitos em uma propriedade f, tal que:

$$\{s \in S \mid M, s \models f\}$$

2.5 Verificação de Modelos Paralela e Distribuída

À medida que se aumenta a complexidade do *design* de um modelo a sua verificação se torna uma tarefa custosa tanto em termos de tempo quanto espaço. Nos últimos anos, grandes esforços têm sido empregados no que tange ao desenvolvimento de técnicas que venham a lidar com o problema da explosão do espaço de estados, possibilitando a verificação de modelos com grandes quantidades de atributos. Dentre estas técnicas é possível citar o uso de verificação de modelos simbólica utilizando-se diagramas de decisão [37], métodos que explorem simetria [47], redução de ordem parcial [36] e técnicas de verificação paralelas e distribuídas, sendo estas últimas o foco deste trabalho.

Abordagens de verificação de modelos paralelas e distribuídas têm sido utilizadas com o intuito de lidar com o problema da explosão do espaço de estados empregando máquinas poderosas e consideráveis quantidades de memória. Devido ao crescente avanço na concepção de máquinas com essas características, o surgimento de algoritmos que explorem ambientes computacionais com memória e processamento distribuídos além de ambientes que usufruam de memória compartilhada são alternativas viáveis aos tradicionais algoritmos sequenciais de verificação.

O uso de máquinas com processamento e espaços de memória distribuídos é relacionado a lidar com o problema da explosão do espaço de estados utilizando-se abordagens baseadas em particionamento de espaços de estados [29]. Entretanto, o principal desafio da construção de ferramentas de verificação que explorem estes ambientes é criar conjuntos disjuntos de estados com certo grau de qualidade os quais serão atribuídos a diferentes processadores para conduzir computações em paralelo.

Mecanismos propostos para ambas técnicas existentes de representação e manipulação de espaços de estados (explícitas e simbólicas) dependem da função de particionamento adotada. Estas funções tem como objetivo gerar conjuntos de estados balanceados reduzindo a presença de transições que atravessam as partições geradas, também conhecidas como *cross-partition transitions*. Uma transição com estas características pode ser descrita como uma transição entre estados localizados em partições distintas [44]. Como consequência, o cálculo de estados sucessores ou antecessores de um dado estado pode requerer comunicação, uma vez que o espaço de estados é representado em pequenos conjuntos distribuídos [29].

Técnicas de particionamento basicamente diferem pela natureza da função de particionamento adotada visando prover balanceamento e localidade. Balanceamento pode ser classificado em:

- Espacial - Cada processador na rede receberá porções de estados com o mesmo número de estados;
- Temporal - Cada processador estará ocupado a maior parte do tempo, isto é, realizando a verificação de estados, passando pouco tempo bloqueado e/ou trocando mensagens.

Localidade pode ser definida como a atribuição de estados que sejam relacionados durante uma verificação ao mesmo processador [43]. Tipicamente, estados sucessores de um dado estado devem ser manipulados pelo mesmo processador visando diminuir a troca de mensagens [15]. Dentre as técnicas de particionamento de espaços de estados pode-se citar a existência de funções de particionamento estáticas, dinâmicas e guiadas pelo usuário, as quais serão brevemente relatadas a seguir.

Funções estáticas são baseadas em uma função matemática a qual tem o objetivo de particionar o espaço de estados sem levar em consideração informações estruturais acerca do modelo, somente entregando partições balanceadas espacialmente. Dessa maneira, as partições geradas podem possuir um número elevado de transições entre as mesmas sofrendo de sobrecarga de comunicação entre as diferentes máquinas empregadas durante a fase de verificação. Funções de particionamento estáticas são as mais utilizadas podendo se confirmar pela grande quantidade de ferramentas disponíveis (Divine [6], Murφ [48] e UPPAAL [7]), entre outras.

De outro lado, funções dinâmicas de particionamento de espaços de estados tentam distribuir porções de estados para diferentes processadores sem a necessidade de conhecimento prévio do sistema. Geralmente, uma função dinâmica de particionamento inicialmente atribui um subconjunto de estados do espaço para cada nodo e, quando certas condições se tornam verdade como, por exemplo, por falta de memória principal, o algoritmo reatribui alguns estados para nodos diferentes [44]. Em contraste, para Lerda *et al.* [35], a função de particionamento é adaptada em tempo de verificação usando informações sobre as relações entre os estados colhidas durante execução para melhor atender o sistema que está sendo verificado.

Ainda, esta técnica de particionamento possui como benefício ser eficiente no que tange a balanceamento de memória, entretanto requerendo altos níves de comunicação e complexidade de implementação, embora permitindo balanceamento de carga e versatilidade. Por outro lado, minimizar o número de transições entre as partições geradas também é importante. Alguns trabalhos relacionados [29, 40, 31] computam uma pequena aproximação do espaço de estados possibilitando a extração de informações acerca das relações entre os estados, facilitando assim a escolha dos conjuntos de estados a serem distribuídos.

Por fim, funções de particionamento de espaços de estados guiadas pelo usuário dependem da experiência do mesmo sobre o sistema. Em [15], Ciardo *et al.* propõe uma função de particionamento baseada na estrutura de Redes de Petri visando prover balanceamento e localidade levando em consideração apenas um conjunto de estados, chamado pelos autores de conjunto controle P. Em geral, qualquer transição que seja disparada e não involva um dado em P corresponde a uma transição entre estados que estão atribuídos ao mesmo processador, caso contrário, a respectiva transição corresponde a estados atribuídos à diferentes partições.

Para Lerda e Sisto [34], quando um nodo computa um novo estado, primeiramente verifica se este estado pertence ao seu subconjunto de estados ou ao subconjunto de outro nodo. Sendo local, o nodo prossegue a verificação normalmente, caso contrário uma mensagem contendo o estado é enviada ao respectivo "dono" do estado. A desvantagem desta abordagem é a não-existência de

uma forma automática de seleção do conjunto de controle, dependendo fortemente da intuição do usuário em selecioná-lo. Algumas abordagens similares são apresentadas em [43] e [13].

De outro lado, o desenvolvimento de processadores *multi-core*, ou seja, que possuam mais de um núcleo de processamento encapsulado no mesmo *chip*, tem recebido amplos esforços nos últimos anos, uma vez que o seu emprego é uma alternativa viável devido à barreiras atingidas pela indústria de semicondutores, tais como: aumento de frequência dos processadores, dificuldade para eficiente dissipação de calor e diminuição do tamanho de determinados componentes. Seguindo esta evolução do *hardware*, ferramentas de verificação as quais usufruam de ambientes *multi-core* e memória compartilhada têm sido amplamente estudadas buscando alternativas a ambientes que endereçam espaços de memória distribuídos.

A principal vantagem que ambientes com memória compartilhada oferecem sobre memória distribuída é justamente prover estruturas de dados compartilhadas para manipulação concorrente, sem a necessidade de troca de mensagens e evitando o desenvolvimento de algoritmos complexos de particionamento de espaços de estados. A diferença entre os algoritmos paralelos para memória compartilhada reside na estrutura de dados adotada e a estratégia de balanceamento de trabalho escolhida para distribuir trabalho entre os processadores [44]. Consequentemente, técnicas de sincronismo se fazem necessárias de serem implementadas, uma vez que garantir acesso concorrente de maneira mutuamente exclusiva é, obviamente, de fundamental importância. Mesmo o foco deste trabalho visando ambientes com memória distribuída, uma breve descrição acerca de técnicas de verificação para ambientes que compartilham memória é feita a seguir.

Dentre os trabalhos que exploram ambientes com memória compartilhada pode-se citar, inicialmente, o trabalho de Allmaier *et al.* [1]. Este trabalho foi um dos pioneiros a empregar algoritmos de geração de espaços de estados e verificação para ambientes com memória compartilhada. Diferentemente de outras abordagens, os autores utilizaram árvores balanceadas para armazenamento de dados compartilhados entre os processos participantes ao invés de *hash tables*, as quais são a estrutura comumente utilizada na maioria das abordagens de verificação para ambientes com estas características.

Existem três abordagens para distribuição de trabalho que são adotadas quando do uso de soluções compartilhadas baseadas em *hash tables*: *static-slicing*, *work stealing* e *stack-slicing*. Conforme anteriormente mencionado, a ferramenta DiVine é baseada em uma estratégia de particionamento estático (*static-slicing*), possuindo ainda uma versão para máquinas com memória compartilhada, a qual segue as mesmas linhas da implementação distribuída. Esta estratégia distribui os estados para cada processador participante sem levar em consideração informações sobre as relações entre os mesmos apresentando ganhos de tempo satisfatórios sendo vantajosa no que tange à simplicidade de implementação [44].

Inggs *et al.* [28] propõe um algoritmo paralelo para exploração de estados baseado em um paradigma de escalonamento *work-stealing* o qual visa garantir balanceamento dinâmico de trabalho sem fases de bloqueio. O conceito básico deste algoritmo permite que processadores sub-utilizados possam "*roubar*" trabalho de outros processadores, ou seja, a estratégia permite um processador

ocioso realizar algumas das tarefas que estejam na fila de trabalho de um processador sobrecarregado. Esta implementação utiliza duas estruturas de filas em cada processo participante, sendo uma fila privada e outra compartilhada que são utilizadas para armazenar estados não-expandidos (*"unexpanded"* segundo o autor, ou não visitados).

Toda vez que um processo ou *thread* não mais possuir estados a serem explorados em sua fila privada, tem de adquirir acesso mutuamente exclusivo a sua fila compartilhada e verificar a existência de estados disponíveis. Não encontrando estados, o processo inicia a busca em todas as filas compartilhadas dos outros processos envolvidos até achar uma fila não-vazia ou verificar que todas as filas estão vazias, o que indica o final da computação.

Por fim, Holzmann *et al.* [26, 25] e Laarman *et al.* [32] empregaram o paradigma *stack-slicing* para distribuição de trabalho entre processadores. Nestes trabalhos, os autores utilizam filas compartilhadas para "*conectar*" os processadores de maneira que se crie uma estrutura em forma de anel onde cada processador tem a capacidade de entregar trabalho ao seu vizinho à direita. Como vantagem, os autores citam a possibilidade de que cada fila de trabalho possua apenas um processo realizando leitura e um realizando escrita ao mesmo tempo, facilitando assim a implementação de mecanismos de bloqueio e atingindo bons resultados de tempo.

3. *STOCHASTIC AUTOMATA NETWORKS MODEL CHECKER*

Conforme anteriormente descrito, SAN conta com interessantes aspectos em relação a outros formalismos como, por exemplo, possibilidade de criação de dependência entre autômatos através de eventos sincronizantes e taxas funcionais. Entretanto, a inexistência de ferramental de verificação para tal formalismo acaba dificultando a verificação de propriedades sobre modelos SAN.

Desta maneira, através de um trabalho cooperativo, esforços foram investidos na construção de um verificador de propriedades escritas na lógica temporal *Computation Tree Logic*, tendo os primeiros resultados discutidos em [19] e [38]. Este capítulo foca na descrição da arquitetura da ferramenta de verificação inicialmente desenvolvida, a qual é base para discussão das abordagens paralelas neste trabalho propostas, sendo estas a contribuição do autor no escopo do projeto de pesquisa.

Para armazenamento e manipulação do espaço de estados gerado por modelos descritos em SAN empregou-se diagramas de decisão multi-valorada, os quais são abordados na Seção 2.2, tendo sua implementação detalhada em [45] e [46] . A Figura 3.1 apresenta os processos (representados por retângulos) e dados (representados por paralelogramos) necessários à verificação de propriedades com a ferramenta, os quais são descritos da seguinte maneira:

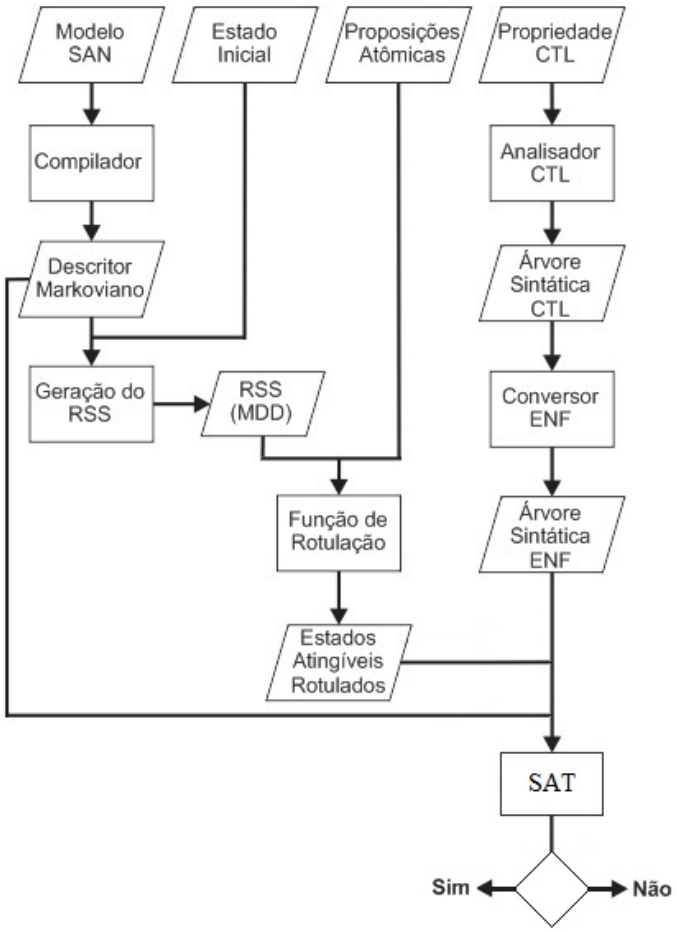


Figura 3.1 – Arquitetura do verificador [19, 38].

- Modelo SAN - o modelo SAN a ser verificado descrito em formato textual;
- Estado Inicial - um ou mais estados iniciais devem ser definidos na função *partial reachability*. Esta função é o ponto de partida para a geração do espaço de estados atingível (RSS);
- Proposições atômicas - conforme descrito na Seção 2.3, proposições atômicas são asserções sobre o modelo que avaliam verdadeiro ou falso sob dado estado global do modelo. Na arquitetura da ferramenta, uma proposição atômica é uma expressão de SAN a qual não possui operadores da lógica temporal;
- Propriedades CTL - propriedades descritas em CTL [27] em que as proposições atômicas sejam definidas como expressões SAN. Exemplos são apresentados no Capítulo 5;
- Compilador - o compilador é responsável por gerar o Descritor Markoviano o qual representa implicitamente uma Cadeia de Markov. Um Descritor Markoviano é composto por um conjunto de tensores que, ao ser operado por álgebra tensorial permite que sejam alcançados todos os elementos da Cadeia de Markov de forma implícita, sendo extremamente eficiente no que tange à armazenamento em memória. As operações (soma tensorial ou produto tensorial) reproduzem as interações e como os autômatos sincronizam o disparo de eventos de um modelo SAN [20];
- Geração do Espaço de Estados atingível - uma vez que o Descritor Markoviano é usado como a função de transição do sistema, dado um estado inicial do modelo, o algoritmo responsável pela geração do RSS verifica nos tensores do descritor se há algum evento habilitado a ser disparado. Havendo evento e possuindo taxa constante, o sucessor do respectivo estado pode ser alcançado e este estado é computado no RSS o qual é codificado como um MDD. Como a técnica de geração do RSS é baseada em saturação, a política de disparo de eventos é guiada pelo algoritmo de saturação [46] o qual dispara eventos por nodo, de maneira *bottom-up* (de baixo para cima), exaustivamente, em vez de disparar por níveis um por iteração. Um nodo do MDD é dito saturado quando codifica um conjunto de estados que caracterizam um ponto fixo com respeito ao disparo de qualquer evento neste nível ou nos níveis abaixo;
- Função de rotulação - esta função rotula os estados atingíveis com o conjunto de proposições atômicas que são verdade em cada estado. Dado o conjunto de proposições atômicas e o RSS, a função cria um MDD para cada proposição atômica codificando os estados que são verdade na referida proposição;
- Analisador CTL - este módulo é responsável por manipular e armazenar a fórmula CTL de entrada em uma árvore sintática;
- Conversor ENF - este módulo é responsável por manipular a árvore sintática e transcrever a fórmula CTL para Forma Normal Existencial. Para tanto, utilizou-se as regras de equivalência de fórmulas descritas em [2] (página 332), as quais foram apresentadas na Seção 2.3;

- Algoritmo de Satisfação de Fórmulas CTL (SAT) - este algoritmo executa a verificação da fórmula de entrada sobre o modelo. A sua implementação é baseada no algoritmo de satisfação de fórmulas CTL em ENF apresentado em [2] (página 348), o qual é detalhado a seguir;

1: ***Sat***(Φ):

2: *Entrada:* sistema de transição finito TS com conjunto de estados S (RSS) rotulado para $L(s)$ e fórmula Φ em ENF
3: *Saída*: $Sat(\Phi) = \{\ s \in S \mid s \models \Phi\ \}$

(* computação recursiva dos conjuntos $Sat(\Psi)$ para todas sub-fórmulas Ψ de Φ *)
4: **switch**(Φ):
5: $true$: **return** S;
6: a: **return** $\{\ s \in S \mid a \in L(s)\ \}$;
7: $(\Phi_1 \wedge \Phi_2)$: **return** $Sat(\Phi_1) \cap Sat(\Phi_2)$;
8: $\neg\Psi$: **return** $S \setminus Sat(\Psi)$;
9: $\exists\bigcirc\Psi$: **return** $\{\ s \in S \mid Post(s) \cap Sat(\Psi) \neq \varnothing\ \}$;
10: $\exists(\Phi_1 \cup \Phi_2)$: $T := Sat(\Phi_2)$; (* Computa o menor ponto fixo *)
11: **while** $\{\ s \in Sat(\Phi_1) \setminus T \mid Post(s) \cap T \neq \varnothing\ \} \neq \varnothing$ **do**
12: **let** $s \in \{\ s \in Sat(\Phi_1) \setminus T \mid Post(s) \cap T \neq \varnothing\ \}$;
13: $T := T \cup \{\ s\ \}$;
14: **od**;
15: **return** T;
16: $\exists\Box\Psi$: $T := Sat(\Psi)$; (* Computa o maior ponto fixo *)
17: **while** $\{\ s \in T \mid Post(s) \cap T = \varnothing\ \} \neq \varnothing$ **do**
18: **let** $s \in \{\ s \in T \mid Post(s) \cap T = \varnothing\ \}$;
19: $T := T \setminus \{\ s\ \}$;
20: **od**;
21: **return** T;
22: **end switch**

$Post(s)$: Conjunto de estados posteriores (sucessores) de um estado s
$L(s)$: Conjunto de rótulos ($labels$) de um estado s, ou seja, proposições atômicas que são satisfeitas em s

---

Figura 3.2 – Algoritmo de Satisfação de fórmulas CTL em ENF [2].

Para melhor entendimento do algoritmo de verificação adotado, considere a Figura 3.2 que o descreve em alto nível. Conforme mencionado anteriormente, o algoritmo somente aceita como entrada fórmulas escritas em ENF, sendo um formato suficiente para representar toda e qualquer fórmula CTL válida. O algoritmo Sat é executado recursivamente através da árvore sintática que representa a fórmula e, para cada chamada $Sat(\Phi)$ onde Φ é uma fórmula ENF válida, os seguintes passos são executados:

- Caso $true$: caso a chamada possua o *booleano true*, retorna S, ou seja, o conjunto de estados que pertence ao espaço de estados atingível.

- Caso a: no caso de proposições atômicas, o algoritmo retorna o conjunto de estados que satisfazem a proposição a.

- Caso $\Phi_1 \wedge \Phi_2$: se for uma conjunção ($\wedge$), o algoritmo retorna o conjunto de estados resultante da intersecção das execuções de $Sat(\Phi_1)$ e $Sat(\Phi_2)$.

- Caso $\neg$: caso negação ($\neg$), o algoritmo realiza a diferença entre o conjunto de estados S, ou seja, o espaço de estados atingível, e os estados resultantes da execução de $Sat(\Psi)$, retornando os estados que não satisfazem Ψ.

- Caso $\exists\bigcirc\Psi$: se a fórmula for $\exists\bigcirc$, o algoritmo avalia os estados pertencentes ao RSS retornando como resultado o conjunto de estados tal que a intersecção dos sucessores destes estados com o conjunto de estados que satisfaz a fórmula Ψ seja diferente de vazio. Caso contrário, o estado em avaliação não satisfaz $\exists\bigcirc\Psi$.

- Caso $\exists(\Phi_1 \cup \Phi_2)$: partindo da semântica deste operador é necessário que Φ_2 seja verdade para que o operador $\exists\cup$ seja satisfeito sob um conjunto de estados. Essa definição estabelece o menor ponto fixo para este operador. Após as execuções de $Sat(\Phi_1)$ e $Sat(\Phi_2)$, a diferença entre os conjuntos de estados resultantes é realizada tendo o resultado armazenado em T. Para cada estado pertencente a este conjunto, o algoritmo realiza a intersecção dos estados sucessores com o conjunto de estados que satisfaz Φ_2 e, caso não-vazia, adiciona o estado em avaliação no conjunto Φ_2. Estes passos são executados repetidas vezes até que o conjunto T atinja um ponto fixo, ou seja, o momento em que não sejam adicionados mais estados ao conjunto Φ_2.

- Caso $\exists\Box\Psi$: caso a fórmula for $\exists\Box$, o algoritmo executa $Sat(\Psi)$ onde o conjunto de estados que satisfaz Ψ será o maior ponto fixo para este operador, sendo armazenado em T. Para cada estado pertencente à T, caso a intersecção de seus sucessores com o conjunto T seja vazia, o estado em avaliação é removido de T. Uma vez que estados são removidos, o novo conjunto T é avaliado executando-se estes mesmos passos repetidas vezes até atingir um ponto fixo, ou seja, um conjunto de estados em que não mais se removam estado. O retorno do algoritmo será o conjunto de estados restantes em T.

4. ABORDAGENS PARALELAS

Considerando o problema da verificação de modelos o qual visa o desenvolvimento de ferramentas eficientes no que tange a uso de memória e tempo para verificação de propriedades sobre complexas realidades, o emprego de máquinas poderosas e algoritmos paralelos e distribuídos é de fundamental importância. Devido à redução do custo de produção dessas máquinas o acesso a tais recursos tem sido facilitado à comunidade científica, impulsionando o desenvolvimento de algoritmos paralelos e distribuídos para ambas técnicas de representação de espaços de estados, explícitas e simbólicas.

Apesar do ganho em memória provido pelo emprego de estruturas de dados simbólicas, a medida que se aumenta a complexidade da realidade modelada é natural que recursos computacionais de máquinas simples venham a se esgotar. Embora estas estruturas permitam representar eficientemente grandes conjuntos de estados, as tarefas relacionadas à computação dos operadores de lógica temporal necessitam de estruturas auxiliares para obtenção do resultado da verificação, o qual é consideravelmente inferior em consumo de memória em relação à estas estruturas.

Para exemplificação deste comportamento pode-se considerar a Tabela 4.1 a qual demonstra o consumo de memória necessário (pela implementação de diagramas de decisão multi-valorada adotada) para representar o espaço de estados atingível de dois modelos, Jantar dos Filósofos e Redes *Wireless Ad Hoc*. Considerando o modelo do Jantar dos Filósofos com 15 filósofos, o qual possui um número considerável de estados pertencentes ao RSS ($|S| = 470.832$), armazená-los consome apenas 10,51 KB. De outro lado, representar o conjunto de estados resultante da verificação de uma fórmula sob o modelo como, por exemplo, $\exists\Box(Phil_i \neq Left)$, consome apenas 8,88 KB, enquanto a estrutura de dados intermediária necessária para atingí-lo chega a 1.611.467,45 KB. A semântica das propriedades e respectivas proposições atômicas descritas na Tabela 4.1 é apresentada no Capítulo 5.

Tabela 4.1 – Consumo de memória da estrutura MDD para verificação de propriedades CTL sob os modelos do Jantar dos Filósofos e Redes *Wireless Ad Hoc*.

Jantar dos Filósofos $\exists\Box(Phil_i \neq Left)$				
Tamanho do Modelo	$\|S\|$	**RSS (KB)**	**Pico (KB)**	**Resultado (KB)**
13	80.782	9,09	240.952,47	7,05
14	195.025	9,80	624.614,66	8,07
15	470.832	10,51	1.611.467,45	8,88
Redes *Wireless Ad Hoc* $\forall\Box((st\ MN\_1 == T) \rightarrow \forall\Diamond(st\ MN\_N == R))$				
20	12.102	20,51	301.759,91	13,29
22	31.682	22,85	968.927,77	14,71
24	82.946	25,19	3.071.106,46	16,12

Conforme descrito na Seção 2.5, ferramentas de verificação de modelos que se utilizam de ambientes providos com espaços de memória distribuídos são projetadas a lidar com o problema da explosão do espaço de estados distribuindo conjuntos disjuntos de estados para diferentes máquinas empregando-se considerável poder computacional e espaços de memória. Embora o emprego destes ambientes permita a representação de grandes espaços de estados, algoritmos que dependam de técnicas de particionamento podem sofrer sobrecarga de comunicação durante a fase de verificação. Sendo o espaço de estados representado de maneira distribuída, estados relacionados através de transições podem estar localizados em diferentes processadores requerendo comunicação durante a computação, possivelmente implicando em perda de desempenho [44, 29].

Conforme anteriormente mencionado, representar consideráveis conjuntos de estados de forma simbólica requer pequenos espaços de memória, de modo que a representação do RSS em sua íntegra em máquinas empregadas para verificação de propriedades habilita as mesmas a calcular estados sucessores de um dado estado localmente. Desta maneira, comunicação acaba sendo necessária somente para cálculo de ponto fixo no âmbito da verificação.

Considerando estas características, duas abordagens paralelas são propostas visando ambientes com processamento e memória distribuídos, sendo:

- Primeira abordagem: possuindo cada máquina uma réplica do espaço de estados e respectiva função de transição, a verificação dos operadores de maior custo computacional, isto é, aqueles em que seja necessária avaliação iterativa de um conjunto de estados até obtenção de ponto fixo, pode ser conduzida de maneira distribuída, de forma que as máquinas empregadas estejam habilitadas a explorar frações distintas do conjunto de estados em consideração. Desta maneira, comunicação é necessária somente para cálculo de ponto fixo. A Seção 4.1 detalha os algoritmos desenvolvidos para esta abordagem;

- Segunda abordagem: conforme relatado na Seção 2.3, a transcrição de uma fórmula CTL para Forma Normal Existencial pode aumentar exponencialmente seu tamanho, implicando no aumento do tempo de verificação, o qual é linear ao tamanho da fórmula [2]. Desta maneira, considerou-se como segunda abordagem, além da distribuição da verificação de conjuntos de estados para os operadores mais custosos da lógica temporal adotada, a distribuição de determinadas partições da árvore sintática que representa a fórmula computacionalmente. A tarefa de verificar uma propriedade consideravelmente grande é realizada através de um algoritmo que analisa e escolhe as partições da árvore sintática ENF equivalente e as distribui para computação em paralelo. Para cada partição, grupos de máquinas são empregadas para computação das tarefas relacionadas à verificação dos operadores mais custosos presentes, através da primeira abordagem descrita. Para esta abordagem, replicação do espaço de estados e respectiva função de transição também se faz necessária. A Seção 4.2 detalha os algoritmos desenvolvidos.

Além da redução no tempo de execução de uma verificação, reduzir o consumo de memória é esperado em algoritmos distribuídos. Embora as abordagens propostas por este trabalho pareçam

pouco eficientes uma vez que réplicas do espaço de estados são feitas, ganhos em consumo são possíveis devido à distribuição das operações relacionadas à verificação. Conforme anteriormente mencionado, estando o aumento do consumo de memória pela estrutura simbólica adotada unicamente ligado à computação dos operadores de lógica temporal, a distribuição das tarefas relacionadas à computação dos mesmos permite considerável diminuição no consumo de memória de cada máquina empregada. Exemplos deste comportamento serão discutidos no Capítulo 5.

4.1 Distribuindo a computação de operadores CTL

Nesta seção são descritos os algoritmos desenvolvidos para distribuição das tarefas relacionadas à computação de operadores da lógica temporal *Computation Tree Logic* utilizando-se a abordagem de programação mestre/escravo. Vale destacar que o algoritmo de verificação adotado somente aceita como entrada fórmulas escritas em Forma Normal Existencial. Sendo assim, foram desenvolvidos algoritmos paralelos para os operadores $\exists\bigcirc$, $\exists\Box$ e $\exists\cup$, os quais em conjunto com operadores da lógica proposicional ($\neg$ e $\wedge$) formam a base para derivação de qualquer operador da CTL. Não foram desenvolvidos algoritmos distribuídos para verificação dos operadores da lógica proposicional, uma vez estas operações são mapeadas para simples operações sobre conjuntos de estados na estrutura MDD - diferença ($\setminus$) e intersecção ($\cup$) - não justificando o desenvolvimento de algoritmos em que troca de mensagens se faz necessária.

4.1.1 Tarefas do Processo Mestre

Antes de introduzir os algoritmos para cada operador, é importante que fiquem claras as tarefas delegadas ao processo mestre, sendo:

- Compilar o modelo SAN gerando o Descritor Markoviano e calcular o espaço de estados atingível. Este último é armazenado na estrutura MDD a qual possui as propriedades descritas na Seção 2.2;
- Transcrever a fórmula CTL de entrada para Forma Normal Existencial e amazená-la em uma árvore sintática;
- Executar a função de rotulação (*Labelling Function*);
- Replicar o Descritor Markoviano, RSS e proposições atômicas rotuladas para os processos escravos.

O algoritmo trabalha de maneira *bottom-up* e divide o trabalho de verificar os operadores $\exists\bigcirc\Psi$, $\exists\Box\Psi$ e $\exists(\Phi_1 \cup \Phi_2)$ entre máquinas escravas. Conforme mencionado no Capítulo 3, o algoritmo de verificação adotado é recursivo. Deste modo, o processo mestre é também responsável

por executar a recursão através da árvore sintática computando as operações $\neg$ e $\wedge$ sequencialmente. A tarefa de unir os conjuntos de estados resultantes das computações das máquinas escravas é também por este executada.

A verificação de determinados operadores requer que se atinjam conjuntos de estados que respeitem um ponto fixo, ou seja, conjuntos de estados que não são alterados pela execução de um algoritmo. Desta maneira, quando se verificam os operadores $\exists\Box$ ou $\exists(\Phi_1 \cup \Phi_2)$, um ponto fixo é atingido em um dado momento em que nenhum processo escravo detecte a remoção ou adição de um estado no conjunto de estados que está sendo verificado. Mais detalhes sobre os algoritmos para estes operadores são encontrados nas Seções 4.1.4 e 4.1.3.

Para troca de estados, estruturas simbólicas são utilizadas a fim de codificar conjuntos que possuam consideráveis números de estados, de maneira compacta. Estas estruturas não são trafegadas através de mensagens, sendo armazenadas em arquivos em disco os quais sofrem operações de leitura e escrita durante a verificação.

Nas seções seguintes são descritos os algoritmos paralelos para os operadores anteriormente mencionados, bem como, as tarefas executas por ambos processos, mestre e escravos. A Seção 4.1.5 apresenta o algoritmo de manipulação de mensagens utilizado para controle da comunicação e execução de operações.

4.1.2 Computação Paralela de $\exists\bigcirc\Psi$

A característica principal do algoritmo para o operador $\exists\bigcirc$ é avaliar o conjunto de estados pertencente ao RSS buscando por estados em que a intersecção de seus posteriores com o conjunto de estados que satisfaz a sub-fórmula Ψ seja não-vazia. Para a computação paralela deste operador, a qual é descrita em alto nível no Algoritmo 1, a tarefa de avaliar os estados pertencentes ao RSS é particionada em $N-1$ processos escravos, os quais executam os seguintes passos:

- Recebem o conjunto de estados que satisfaz Ψ do processo mestre;
- Avaliam conjuntos de estados do RSS de igual tamanho respeitando o algoritmo para $\exists\bigcirc\Psi$, conforme descrito no algoritmo nas linhas 10 à 16;
- Retornam o conjunto de estados resultante (T_2) ao processo mestre.

A avaliação deste operador não necessita o alcance de um ponto fixo o que permite a adoção de um número escalável de processos os quais podem trabalhar separadamente sem necessidade de rodadas de sincronia. Desta maneira, a troca de mensagens fica limitada à ditar o início e fim da computação, compreendendo este último o retorno dos estados resultantes ao processo mestre, o qual é responsável por uní-los.

Algoritmo 1: Algoritmo $Sat(\Phi)$ com implementação paralela para $\exists\bigcirc\Psi$

Entrada: $P/\ Mestre$ - sistema de transição finito TS com conjunto de estados S (RSS) rotulado para $L(s)$ e fórmula Φ em ENF
Entrada: $P/\ Escravos$ - sistema de transição finito TS com conjunto de estados S (RSS) rotulado para $L(s)$ e conjunto de estados que satisfazem Ψ tal que $\Psi \subseteq$ RSS e fórmula Φ em ENF
Saída: $P/\ Mestre$ - conjunto de estados que satisfazem $\exists\bigcirc\Psi$
Saída: $P/\ Escravos$ - conjunto de estados avaliados pelo processo j na partição k de S que satisfazem $\exists\bigcirc\Psi$
Dados: **MDD** T, T_2;
Dados: **Estado** s;

```
1  início
2      caso Φ
3          selecione ∃○Ψ faça
4              se mestre então
5                  T := Sat(Ψ);
                   /* Início de computação distribuída para ∃○          */
6                  DisparaComputacaoParalela(T);
7                  Sincroniza(T);
8                  return T;
9              senão
                   /* Processo escravo j avalia partição k do RSS       */
10                 para i := inicial até i < final faça
11                     seja s ∈ k
12                     se Post(s) ∩ T != ∅ então
13                         T_2 := T_2 ∪ s;
14                     fim se
15                 fim para
                   /* Processos escravos retornam T_2 ao mestre         */
16                 return T_2;
17             fim se
18             break;
19         fim selec
20     fim selec
21 fim
22 ---------------------------------------------------------------------
```

23 $DisparaComputacaoParalela(T)$: Envio de mensagem aos escravos indicando início da verificação do operador $\exists\bigcirc$ para conjunto T
24 $Sincroniza(T)$: Recebimento e união dos estados resultantes vindos dos escravos
25 $Post(s)$: Recupera estados posteriores (sucessores) de um estado s
26 $L(s)$: Conjunto de rótulos ($labels$) de um estado s, ou seja, proposições atômicas que são satisfeitas em s

4.1.3 Computação Paralela de $\exists(\Phi_1 \cup \Phi_2)$

Tendo-se o RSS e o conjunto de estados resultante da diferença entre os conjuntos Φ_1 e Φ_2, o algoritmo para o operador $\exists\cup$ realiza exaustiva busca por estados que possuam sucessores pertencendo ao conjunto Φ_2 e, satisfeita esta regra, adiciona-os a este conjunto. Estes passos são executados repetidas vezes até que se atinja um ponto fixo e o resultado para este operador é o conjunto de estados armazenado em Φ_2. O algoritmo executado por ambos processos mestre e escravos é descrito no Algoritmo 2. Para obtenção de paralelismo na avaliação do operador $\exists\cup$, os processos escravos executam os seguintes passos:

- Recebem do processo mestre os conjuntos de estados que satisfazem Φ_1 e Φ_2. Cada conjunto é carregado em memória uma única vez;
- Calculam a diferença entre Φ_1 e Φ_2 ($R \setminus T$), conforme descrito na linha 18 do algoritmo;
- Avaliam uma partição de estados que pertence a diferença acima mencionada respeitando o algoritmo para o operador $\exists\cup$ (linhas 19 à 26);
- Enviam ao processo mestre o conjunto de estados locais resultantes, ou seja, estados adicionados ao conjunto Φ_2. Neste caso, retorna-se o conjunto saída local T_2 (linha 26);
- Aguardam o momento em que o processo mestre executa uma rodada de sincronia enviando o conjunto global de estados adicionados;
- Adicionam os estados em Φ_2 vindos do processo mestre pela respectiva rodada de sincronia. Este passo é executado juntamente com o algoritmo de tratamento de mensagens (*Message Handler*), detalhado na Seção 4.1.5;
- Repetem do passo dois em diante a avaliação de estados em seus conjuntos locais de estados até o processo mestre detectar a obtenção de um ponto fixo, ou seja, não mais se adicionam estados em Φ_2.

Algoritmo 2: Algoritmo $Sat(\Phi)$ com implementação paralela para $\exists(\Phi_1 \cup \Phi_2)$

Entrada: $P/\ Mestre$ - sistema de transição finito TS com conjunto de estados S (RSS) rotulado para $L(s)$ e fórmula Φ em ENF
Entrada: $P/\ Escravos$ - sistema de transição finito TS com conjuntos de estados que satisfazem Φ_1 e Φ_2 tal que $\Phi_1 \subseteq$ RSS e $\Phi_2 \subseteq$ RSS e fórmula Φ em ENF
Saída: $P/\ Mestre$ - conjunto de estados que satisfazem $\exists(\Phi_1 \cup \Phi_2)$
Saída: $P/\ Escravos$ - conjunto de estados adicionados em T pelo processo de índice j
Dados: **MDD** R, T, T_2, D;
Dados: **Estado** s;

```
1  início
2      caso Φ
3          selecione ∃(Φ1 ∪ Φ2) faça
4              se mestre então
5                  R := Sat(Φ1);
6                  T := Sat(Φ2);
7                  T2 := T;
8                  enquanto true faça
                       /* Envia mensagem aos escravos indicando início de computação  */
9                      DisparaComputacaoParalela(R, T2);
                       /* Executa rodada de sincronia                                 */
10                     se Sincroniza(T2) == true então
11                         break;
12                     senão
                           /* Caso contrário, adiciona estados oriundos da sincronia em T  */
13                         T := T ∪ T2;
14                     fim se
15                 fim enqto
16                 return T;
17             senão
18                 D := Φ1 \ Φ2;
                   /* Cada escravo j verifica uma partição k da diferença D       */
19                 para i := inicial até i < final faça
20                     seja s ∈ k
21                     se Post(s) ∩ T != ∅ então
                           /* Adiciona o estado corrente s no conjunto T          */
22                         T := T ∪ s;
                           /* Adiciona o estado corrente s no conjunto saída local T2  */
23                         T2 := T2 ∪ s;
24                     fim se
25                 fim para
                   /* Processos escravos retornam T2 ao processo mestre o qual executa rodada de
                   sincronia e verifica se um ponto fixo foi atingido          */
26                 return T2;
27             fim se
28             break;
29         fim selec
30     fim selec
31 fim
32 ---------------------------------------------------------------------------
```

33 $DisparaComputacaoParalela(R, T_2)$: Envio de mensagem aos escravos indicando início da verificação do operador $\exists\cup$ para conjuntos R e T_2
34 $Sincroniza(T_2)$: Recebimento e união dos estados resultantes vindos dos escravos e detecção de ponto fixo
35 $Post(s)$: Recupera estados posteriores (sucessores) de um estado s
36 $L(s)$: Conjunto de rótulos ($labels$) de um estado s, ou seja, proposições atômicas que são satisfeitas em s

4.1.4 Computação Paralela de $\exists\Box\Psi$

Como característica do algoritmo de verificação adotado para este operador, é necessário avaliar o conjunto de estados em que a sub-fórmula Ψ é satisfeita buscando por estados que possuam sucessores pertencendo a este conjunto. Os estados que não respeitarem esta regra são removidos do conjunto sendo necessária sua reavaliação até a obtenção de um ponto fixo. O Algoritmo 3 descreve em alto nível os passos necessários para obtenção de paralelismo para o operador. Os processos escravos são responsáveis pela execução de um conjunto de passos, sendo:

- Inicialmente recebem o conjunto de estados que satisfaz Ψ. Segundo a descrição do algoritmo deste operador realizada no Capítulo 3, o conjunto que inicialmente satisfaz Ψ é considerado o maior ponto fixo para o operador;
- Avaliam um intervalo específico de estados respeitando o algoritmo para $\exists\Box$ até que este conjunto atinja um ponto fixo intermediário, ou seja, significa que existe um ponto fixo local para uma partição de estados k do conjunto Ψ sendo avaliado pelo escravo j, entretanto não há um ponto fixo global no que diz respeito a computação como um todo do conjunto Ψ, incluindo todas as partições distribuídas em $N-1$ processos. Este comportamento pode ser visualizado no algoritmo nas linhas 17 à 29;
- Retornam o conjunto de estados removidos do conjunto Ψ para o processo mestre (conjunto T_2);
- Aguardam até o momento da execução de uma rodada de sincronia;
- Recebem o conjunto global de estados removidos os quais foram unidos pelo mestre e removem estes estados de seu conjunto local Ψ. Este passo e o anterior são executados juntamente com o algoritmo de manipulação de mensagens, descrito na Seção 4.1.5;
- Repetem do passo dois em diante a avaliação de estados em seus conjuntos locais de estados;
- Executam estes passos até o momento que o processo mestre detecte um ponto fixo global, isto é, uma rodada em que os escravos não mais removam estados de Ψ (conjunto T).

Ao final da computação dos operadores $\exists\Box$ e $\exists\cup$, é percebível que os processadores envolvidos terão atingido um conjunto igual de estados. Em um dado momento, disparando o processo mestre a computação de um dos operadores $\exists\bigcirc$, $\exists\Box$ ou $\exists\cup$, permanece bloqueado aguardando o retorno da computação dos escravos para execução de uma rodada de sincronia, no caso dos operadores $\exists\Box$ ou $\exists\cup$, ou a união das computações para o operador $\exists\bigcirc$.

Algoritmo 3: Algoritmo $Sat(\Phi)$ com implementação paralela para $\exists\Box\Psi$

Entrada: $P/\ Mestre$ - sistema de transição finito TS com conjunto de estados S (RSS) rotulado para $L(s)$ e fórmula Φ em ENF
Entrada: $P/\ Escravos$ - sistema de transição finito TS com conjunto de estados que satisfazem Ψ tal que $\Psi \subseteq$ RSS e fórmula Φ em ENF
Saída: $P/\ Mestre$ - conjunto de estados que satisfazem $\exists\Box\Psi$
Saída: $P/\ Escravos$ - conjunto de estados removidos da partição k de T sendo avaliada pelo processo j
Dados: **MDD** T, T_2;
Dados: **Estado** s;
Dados: **bool** flagPontoFixo;

```
1  início
2      caso Φ
3          selecione ∃□Ψ faça
4              se mestre então
5                  T := Sat(Ψ);
6                  T_2 := T;
7                  enquanto true faça
                       /* Mensagem aos escravos indicando início de computação para ∃□ */
8                      DisparaComputacaoParalela(T_2);
9                      se (Sincroniza(T_2) == true) então
10                         break;
11                     senão
                           /* Caso contrário, remove estados oriundos da sincronia */
12                         T := T \ T_2;
13                     fim se
14                 fim enqto
15                 return T;
16             senão
                   /* Cada processo escravo j verifica uma partição k de T */
17                 flagPontoFixo := true;
18                 enquanto flagPontoFixo == true faça
19                     flagPontoFixo := false;
20                     para i := inicial até i < final faça
21                         seja s ∈ k
22                         se Post(s) ∩ T == ∅ então
                               /* Remove o estado corrente s do conjunto T */
23                             T := T \ s;
                               /* Adiciona o estado corrente s no conjunto saída T_2 */
24                             T_2 := T_2 ∪ s;
25                             final := final − 1;
26                             flagPontoFixo := true;
27                         fim se
28                     fim para
29                 fim enqto
                   /* Processos escravos retornam T_2 ao processo mestre que executa rodada de
                   sincronia e verifica se um ponto fixo foi atingido */
30                 return T_2;
31             fim se
32             break;
33         fim selec
34     fim selec
35 fim
36
```

37 $DisparaComputacaoParalela(T_2)$: Envio de mensagem aos escravos indicando início da verificação do operador $\exists\Box$ para conjunto T_2
38 $Sincroniza(T_2)$: Recebimento e união dos estados resultantes vindos dos escravos e detecção de ponto fixo
39 $Post(s)$: Recupera estados posteriores (sucessores) de um estado s
40 $L(s)$: Conjunto de rótulos ($labels$) de um estado s, ou seja, proposições atômicas que são satisfeitas em s

4.1.5 Algoritmo para Manipulação de Mensagens (*Message Handler*)

Devido à adoção de ambientes que usufruam de espaços de memória distribuídos o uso de mecanismos de troca de mensagens se fez necessária para desenvolvimento dos algoritmos neste capítulo relatados. Para tanto, adotou-se uma implementação *open-source* do ambiente MPI (*Message Passing Interface*) - *OpenMPI* versão 1.4.5 - a qual provê funções síncronas e assíncronas de comunicação entre máquinas.

Funções síncronas de comunicação foram utilizadas uma vez que os algoritmos possuem grande dependência de dados. Cada máquina escrava, ao receber uma mensagem, executa uma rotina de tratamento descrita em alto nível pelo Algoritmo 4 a qual indica as operações a serem executadas, sendo:

- $DADOS$: este identificador indica aos processos escravos que estão aptos à carregar em memória o Descritor Markoviano e RSS, possuindo este último os conjuntos de estados que são válidos para cada proposição atômica definida.
- EX: este identificador indica início de computação distribuída do operador $\exists\bigcirc$ (EX). Neste momento os processos escravos possuem em memória o conjunto de estados que satisfaz Ψ, vide algoritmo descrito na Seção 4.1.2.
- EU: este identificador indica início da computação distribuída do operador $\exists(\Phi_1 \cup \Phi_2)$ (EU). Neste momento cada processo escravo possuirá em memória os conjuntos de estados que satisfazem Φ_1 e Φ_2, vide algoritmo descrito na Seção 4.1.3.
- EG: este identificador indica o início de computação distribuída para o operador $\exists\Box$ (EG). Após o recebimento desta mensagem, processos escravos possuirão em memória o conjunto de estados que satisfaz Ψ, conforme algoritmo descrito na Seção 4.1.4.
- EU_sinc: este identificador indica a execução de uma rodada de sincronia para o operador $\exists\cup$. Processos escravos carregam em memória e adicionam o conjunto global de estados vindo do processo mestre ao conjunto de estados que satisfaz Φ_2 (conjunto T).
- EG_sinc: este identificador indica a execução de uma rodada de sincronia para o operador $\exists\Box$. Processos escravos carregam em memória o conjunto de estados vindo do mestre e o removem de seu conjunto de estados local Ψ (conjunto T).

Toda vez que a função $CarregaDados$ é executada, uma leitura em disco é feita carregando-se em memória a estrutura simbólica que representa a referida computação (conjunto de estados) no conjunto dado como parâmetro na função. Não serão detalhadas as operações executadas na estrutura MDD para codificação e carregamento de conjuntos de estados de forma simbólica.

Algoritmo 4: Método de tratamento de mensagens

Entrada: Mensagem MPI
Dados: **MDD** RSS;

```
1  início
2      caso mensagem.tipo
3          selecione DADOS faça
4              CarregaDescritor();
               /* Carrega em memória RSS rotulado para AP                    */
5              CarregaDados(RSS);
6              break;
7          fim selec
8          selecione EX faça
9              CarregaDados(Ψ);
10             Sat(∃◯Ψ);
11             break;
12         fim selec
13         selecione EU faça
14             CarregaDados(Φ1);
15             CarregaDados(Φ2);
16             Sat(∃(Φ1 ∪ Φ2));
17             break;
18         fim selec
19         selecione EG faça
20             CarregaDados(Ψ);
21             Sat(∃□Ψ);
22             break;
23         fim selec
24         selecione EU_sinc faça
25             CarregaDados(T2);
26             T := T ∪ T2;
27             Sat(∃(Φ1 ∪ Φ2));
28             break;
29         fim selec
30         selecione EG_sinc faça
31             CarregaDados(T2);
32             T := T \ T2;
33             Sat(∃□Ψ);
34             break;
35         fim selec
36     fim selec
37 fim
38 ─────────────────────────────────────────────────────────────────
39 CarregaDescritor(): Carrega em memória o arquivo que representa o Descritor
   Markoviano o qual é utilizado como função de transição do sistema e foi gerado no
   momento da compilação do modelo SAN
40 CarregaDados(): Função que carrega na estrutura MDD o conjunto de estados
   simbolicamente representado em um arquivo
41 ─────────────────────────────────────────────────────────────────
```

4.2 Distribuindo a computação de fórmulas CTL

Conforme descrito no início deste capítulo, além da distribuição da computação de operadores CTL, considerou-se o desenvolvimento de um algoritmo de distribuição de determinadas partes da árvore sintática que representa a propriedade computacionalmente, uma vez que a transcrição da mesma para o formato aceito pelo algoritmo de verificação pode aumentar consideravelmente seu tamanho. Sendo o tempo de verificação do algoritmo adotado linear ao tamanho da fórmula [2] dada como entrada, a adoção de distribuição de tarefas (partições da árvore sintática) para computação em paralelo permite ganhos de desempenho.

Anteriormente à descrição do algoritmo, alguns pontos devem ser considerados. A verificação de determinados operadores CTL é mais custosa e requer um conjunto maior de passos a serem executados que a de outros. Operadores como a negação ($\neg$) e a conjunção ($\wedge$) são mapeados para simples operações sob conjuntos, sendo eficientemente executados como simples operações de intersecção ($\cap$) e diferença ($\setminus$) sobre a estrutura de diagramas de decisão adotada. Partindo destas características, empregar algoritmos paralelos para computação destes operadores acaba tornando-se inviável pois o tempo necessário à computação sequencial dos mesmos é baixa, não justificando o desenvolvimento de algoritmos distribuídos onde troca de mensagens entre máquinas se faz necessária.

Partindo destas regras, particionar uma fórmula ENF fica condicionado à encontrar ramos da árvore que possuam operadores em que sua verificação seja computacionalmente custosa, tais como: $\exists\bigcirc$, $\exists\cup$ e $\exists\Box$. Para a verificação destes operadores foram empregados os algoritmos descritos na Seção 4.1. Desta forma, a verificação de uma fórmula CTL é conduzida distribuindo-se partições da árvore sintática que representa a fórmula em ENF para diferentes máquinas.

Cada máquina dona de uma partição da árvore, por sua vez, executa a recursão através da partição e computa sequencialmente as operações da lógica proposicional presentes ($\neg$ e $\wedge$), disparando a verificação dos operadores mais custosos para um grupo de máquinas, participando conjuntamente na verificação destes operadores. Para construção deste algoritmo, também adotou-se o padrão mestre/escravo. É importante descrever os passos executados por ambos processos mestre e escravos antes da introdução do algoritmo de particionamento de fórmulas propriamente dito.

4.2.1 Tarefas do Processo Mestre

O processo mestre é responsável pela execução de um conjunto de passos antes e durante a verificação distribuída de uma propriedade CTL. Os passos que dizem respeito à compilação do modelo SAN (geração do Descritor Markoviano), geração do espaço de estados atingível, execução da função de rotulação, transcrição da fórmula de entrada para ENF e replicação destes dados para todos processos escravos envolvidos são os mesmos descritos no algoritmo da Seção 4.1.

Para o algoritmo em questão, o processo mestre é responsável pela execução de mais algumas tarefas, sendo:

- Execução do algoritmo de escolha de particionamento de fórmulas ENF. Este algoritmo, o qual é detalhado na Seção 4.2.3, tem como objetivo estudar e escolher quais partes da fórmula são suscetíveis à verificação em paralelo. Como saída o algoritmo gera uma pilha de trabalho contendo índices que são utilizados para identificação das partições das fórmulas a serem distribuídas;
- Distribuição das partições para os processos escravos e controle da computação;
- Recepção e armazenamento dos estados resultantes das verificações. Conforme as computações vão sendo finalizadas, o processo mestre recebe os estados resultantes e, por fim, manipula a sua árvore sintática local removendo as partições já verificadas. Ao se remover ramos da árvore sintática, eventuais ramos restantes podem estar suscetíveis à verificação em paralelo, sendo necessária nova execução do algoritmo de escolha de particionamento pelo processo mestre;
- Execução da verificação das operações restantes na árvore sintática. O processo mestre, ao receber todas as computações oriundas das verificações das partições da árvore sintática, possui em memória apenas algumas operações restantes na sua árvore local, as quais não configuram duas ou mais partições. Neste caso, a verificação dos operadores restantes na árvore sintática é conduzida exatamente de acordo com os algoritmos descritos na Seção 4.1, onde distribui-se a verificação dos operadores $\exists\bigcirc$, $\exists(\Phi_1 \cup \Phi_2)$ e $\exists\Box$, na presença destes.

4.2.2 Tarefas dos Processos Escravos

Uma vez que o processo mestre seja responsável por entregar trabalho aos processos escravos, a tarefa de conduzir rodadas de sincronia para cálculo de ponto fixo, no caso dos operadores $\exists\cup$ e $\exists\Box$, ou união das computações resultantes no caso do operador $\exists\bigcirc$, fica delegada aos próprios processos escravos. Desta forma, dois grupos de escravos foram definidos, processos escravos coordenadores e processos escravos que obedecem aos coordenadores, os quais diferem pelas tarefas atribuídas. Para a computação de cada partição da fórmula são definidos grupos de trabalho compostos por um processo coordenador e N processos escravos, onde o tamanho de cada grupo é fixo e definido pelo usuário da ferramenta e vai do processo i até $N-1$, onde i é o identificador do processo coordenador do grupo em questão.

Os processos escravos coordenadores são responsáveis pela execução de rodadas de sincronia e detecção de ponto fixo. De outro lado, processos escravos que obedecem aos coordenadores são responsáveis somente pela computação distribuída das operações recebidas do processo coordenador do grupo de trabalho corrente, e executam as mesmas tarefas já descritas nos algoritmos da Seção 4.1. Abaixo são detalhadas as tarefas delegadas aos processos escravos coordenadores:

- Após recebimento do identificador da partição da árvore a ser verificada, o coordenador corrente busca na sua árvore local a referência à respectiva partição. Após este passo, invoca o algoritmo recursivo $Sat(\Phi)$ executando as operações de negação ($\neg$) e conjunção lógica ($\wedge$) presentes na partição;

- Ao final da verificação das sub-fórmulas (Ψ, Φ_1 e Φ_2) dos operadores $\exists\bigcirc\Psi$, $\exists(\Phi_1 \cup \Phi_2)$ ou $\exists\Box\Psi$, o processo coordenador dispara a verificação do operador ativo para o conjunto de processos pertencentes ao seu grupo de trabalho, participando conjuntamente com estes na exploração de estados, isto é, assumindo neste momento o papel de escravo;

- A execução das rodadas de sincronia necessárias pelos algoritmos para cálculo de ponto fixo ficam delegadas aos processos coordenadores, sendo executados os mesmos passos descritos para os algoritmos da Seção 4.1;

- Ao final da computação da partição recebida, cada coordenador retorna o conjunto de estados resultante da verificação da partição da árvore sintática em questão ao processo mestre e aguarda a atribuição de novo trabalho.

Processos escravos que obedecem os coordenadores, por sua vez, executam as seguintes tarefas:

- Aguardam a atribuição de trabalho do processo coordenador.

- Recebendo mensagem a qual contém a operação a ser executada, carregam em memória os dados relativos ao conjunto de estados à ser verificado. Após este passo, executam uma chamada ao algoritmo $Sat(\Phi)$ e procedem os mesmos passos para exploração distribuída de estados conforme descrito na Seção 4.1.

- Após execução do algoritmo de verificação, retornam os estados resultantes simbolicamente codificados ao processo coordenador, o qual executa as respectivas rodadas de sincronia e detecta ou não a obtenção de um ponto fixo para a operação que está sendo verificada.

- Aguardam novamente a atribuição de trabalho.

Sem fazer distinção, processos escravos estão suscetíveis ainda a receber trabalho vindo diretamente do processo mestre. Conforme mencionado no início desta Seção, em um dado momento, após a verificação de todas as partições identificadas da árvore sintática ter sido concluída, cabe ao processo mestre conduzir a verificação das operações que restarem na árvore sintática, a qual pode ser guiada de maneira paralela, na presença de operadores que possuam algoritmo distribuído ($\exists\bigcirc\Psi$, $\exists(\Phi_1 \cup \Phi_2)$ ou $\exists\Box\Psi$), ou sequencial, no caso de operações de negação e conjunção lógica.

4.2.3 Algoritmo de Escolha de Particionamento de Fórmulas ENF

Conforme anteriormente mencionado, somente serão consideradas partições de uma fórmula ENF aptas à verificação em paralelo as quais possuem pelo menos um operador onde sua verificação é computacionalmente custosa. Estes operadores conduzem iterativamente exploração exaustiva de conjuntos de estados, sendo: $\exists\bigcirc$, $\exists\cup$ e $\exists\Box$.

O algoritmo desenvolvido percorre a árvore de maneira $top\text{-}down$ realizando a busca e contagem das operações que podem configurar uma sub-árvore como uma partição à ser distribuída. Desta maneira, todos os ramos da árvore sintática ENF necessitam ser avaliados. Para busca e contagem dos operadores acima listados, foi desenvolvido um algoritmo de busca em profundidade o qual progride partindo do nodo dado como referência e é executado de maneira não-recursiva, utilizando-se uma pilha para controle de nós não-visitados.

Em conjunto com o algoritmo de busca das operações de interesse, um algoritmo que avalia a disposição dos operadores na árvore sintática foi desenvolvido. Para cada sub-árvore, o algoritmo considera um conjunto de regras e executa os seguintes passos:

- Realiza a busca e contagem das operações $\exists\bigcirc$, $\exists\cup$ e $\exists\Box$;
- Havendo somente uma operação de interesse, o algoritmo adiciona o identificador do nodo pai da sub-árvore que está analisando em uma lista de partições a serem verificadas em paralelo;
- Havendo a presença de mais de uma operação de interesse, o algoritmo consulta todos os ramos presentes na sub-árvore em questão, em virtude de analisar a disposição dos operadores nos ramos. Para cada ramo as mesmas regras são consideradas. Um ramo é considerado pelo algoritmo como uma sequência de operadores em que não existam operações binárias;
- Havendo mais de um operador de interesse, entretanto estando dispostos em ramos que possuam dependência, ou seja, a verificação de um operador em um nível superior depende do resultado da verificação de um operador em um nível inferior, o ramo é assinalado como uma partição apta à verificação em paralelo. Um exemplo deste comportamento pode ser visualizado na partição de índice 2 da Figura 4.4. O algoritmo desenvolvido também identifica este comportamento em sub-árvores inteiras.

O algoritmo não permite que se assinale como uma partição apta à verificação em paralelo uma sub-árvore em que existam operações de interesse dispostas em ramos ou lados distintos, estes não possuindo dependência na verificação. O particionamento é feito de modo a criar o maior número possível de partições distintas para verificação em paralelo. Pilhas são utilizadas para controle das sub-árvores e ramos a serem visitados pelo algoritmo, bem como, para armazenamento temporário dos nodos pais de cada sub-árvore e/ou ramo em análise.

Para exemplificação do algoritmo de escolha de particionamento pode-se considerar as seguintes fórmulas CTL[1]:

$$\forall(p \cup q) \tag{4.1}$$

$$\forall\Diamond(q \wedge \neg r \rightarrow \forall(\neg r \cup (p \wedge \neg r))) \tag{4.2}$$

$$\forall\Box[\bigwedge_{\forall 1 \leq i < N} (p \rightarrow \forall\Diamond(\neg p))] \tag{4.3}$$

$$\exists\Box(\neg\exists\Box(\neg\exists \bigcirc p) \wedge \neg\exists(p \cup \neg q)) \tag{4.4}$$

As árvores sintáticas em Forma Normal Existencial que representam as fórmulas acima podem ser visualizadas nas Figuras 4.1, 4.2, 4.3 e 4.4, respectivamente. Cada partição escolhida pelo algoritmo é representada em um tom de cinza. Para facilitar a identificação, além da operação a ser executada, cada nodo presente na árvore sintática possui um valor inteiro. O restante deste capítulo se limita a demonstrar o particionamento das fórmulas acima listadas e os passos executados por ambos processos mestre e escravos durante a verificação.

Desta maneira, as propriedades acima são particionadas e verificadas da seguinte forma:

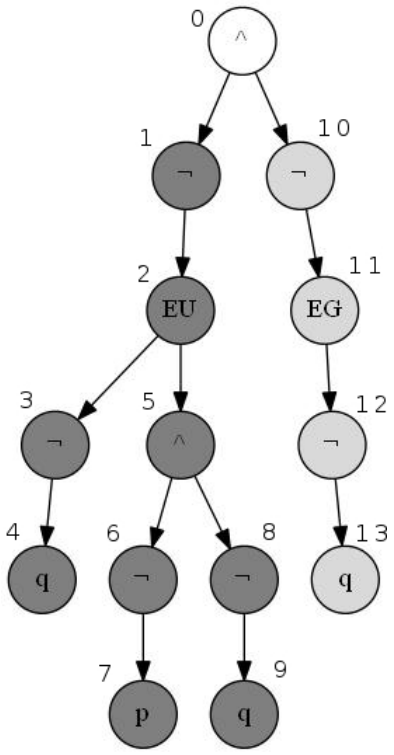


Figura 4.1 – Árvore sintática que representa a Propriedade 4.1 transcrita para ENF.

[1]Mais propriedades CTL podem ser encontradas em: http://patterns.projects.cis.ksu.edu

1. A Propriedade 4.1, transcrita para ENF na Figura 4.1, é particionada em duas sub-árvores para verificação em paralelo. Através da busca de operações de interesse pelo algoritmo, é possível visualizar que há a presença de apenas 1 destes operadores em cada lado da árvore sintática, partindo-se da raiz ($\wedge$). Desta maneira, como a presença de operações de negação e conjunção lógica são descartadas, a árvore permite sua quebra nos índices 1 (operação $\neg$) e 10 (operação $\neg$), sendo este comportamento demonstrado na respectiva figura. A verificação desta fórmula procede através da distribuição destes índices para dois grupos de processos, os quais executam os seguintes passos:

   - No caso da partição identificada pelo índice 1, o processo coordenador do grupo, vide algoritmo descrito na Seção 4.2.2, busca a referência ao operador correspondente na sua árvore sintática ENF local e invoca o algoritmo Sat sendo responsável pela execução sequencial das operações de índice 3 à 9. Finalizado este passo, o coordenador dispara a computação do operador $\exists\cup$ para os processos que fazem parte de seu grupo de trabalho, participando este conjuntamente da computação, sendo responsável pela detecção de ponto fixo para o operador. Ao final, o processo coordenador executa a operação de índice 1 e retorna o conjunto de estados resultante ao processo mestre, ficando disponível novamente à receber trabalho.
   - Para a partição identificada pelo índice 10, o coordenador que a recebe executa os seguintes passos: após buscar a referência ao respectivo nodo na árvore sintática, invoca o algoritmo de verificação executando as tarefas relacionadas às operações de índices 12 e 13. Como próximo passo, o coordenador dispara e participa da computação do operador $\exists\Box$ juntamente com os processos escravos pertencentes ao seu grupo de trabalho. Ao final, executa as tarefas relacionadas à operação de índice 10 e retorna os estados resultantes da computação desta partição ao processo mestre.
   - O mestre, por sua vez, após receber os respectivos conjuntos de estados resultantes da verificação das duas partições, manipula a árvore sintática removendo as partições e invoca o algoritmo de verificação sendo responsável pela execução da operação de índice 0 ($\wedge$), finalizando a fase de verificação.

2. A Propriedade 4.2, representada em ENF na Figura 4.2, possui a presença de três operadores considerados computacionalmente custosos nos nós de índices 1, 10 e 24. Entretanto, conforme a descrição do algoritmo, avaliando-se a disposição destes operadores na fórmula pode-se assinalar apenas 2 partições a serem verificadas em paralelo. Os nodos de índice 9 e 23 representam os nodos mais altos (nodos pai) para as respectivas partições, as quais somente possuem a presença de 1 operador de interesse cada, sendo $\exists\cup$ e $\exists\Box$, respectivamente. A verificação das partições desta fórmula procede de maneira semelhante à fórmula anterior ($\forall(p$ U $q)$). Entretanto, ao final da etapa de computação realizada para cada partição, o processo mestre executa a verificação do restante da fórmula de maneira igualmente descrita no algoritmo da Seção 4.1, executando-se os seguintes passos:

- O mestre, após recebimento das computações relativas às partições identificadas pelos índices 9 e 23 e manipulação da sua árvore sintática local, invoca o algoritmo de verificação para o restante da fórmula. A verificação das operações de índices 2 à 8 é executada sequencialmente pelo próprio processo mestre, disparando a computação do operador $\exists\Box$ de índice 1 para os processos escravos que, por sua vez, executam de maneira distribuída as tarefas relacionadas à verificação deste operador. As rodadas de sincronia necessárias para atingir um conjunto de estados que caracterizam um ponto fixo para o operador são executadas pelo processo mestre, vide algoritmo descrito na Seção 4.1.3. Ao final da computação deste operador, o processo mestre executa a operação $\neg$ de índice 0, finalizando a verificação da fórmula.

3. A Propriedade 4.3, representada em ENF pela Figura 4.3, apresenta um número considerável de sub-árvores as quais respeitam as regras para particionamento exploradas pelo algoritmo. Para facilitar o entendimento dos passos executados durante a verificação, um exemplo utilizando a partição identificada pelo índice 10 ($\neg$) é feito:

   - Após recebimento do identificador de partição e busca da respectiva referência na árvore sintática, o processo coordenador corrente, invocando a rotina de verificação $Sat(\Phi)$, executa as operações de índice 12 e 14. Finalizada esta etapa, a computação do operador 13 ($\exists\Box$) é distribuída entre o coordenador e o conjunto de processos escravos que fazem parte do respectivo grupo de trabalho. Finalizada a verificação de $\exists\Box$, as operações 10 e 11 são executadas pelo respectivo coordenador, finalizando a verificação desta partição ao retornar os estados resultantes para o processo mestre.
   - Para as demais partições demarcadas na fórmula, passos semelhantes são executados pelos respectivos coordenadores e grupos de trabalho. O número de partições que podem ser avaliadas em paralelo são dependentes, obviamente, da quantidade de processos empregados para computação e o tamanho escolhido pelo usuário para cada grupo de trabalho.
   - Ao final da computação de todas as partições da fórmula o processo mestre computa as operações de negação e conjunção lógica. A computação do operador de índice 1 ($\exists\cup$) é distribuída para todos os processos escravos, vide algoritmo descrito na Seção 4.1.

4. Para a Propriedade 4.4, duas partições são identificadas pelo algoritmo. A primeira, de índice 2, possui duas operações de interesse, entretanto, a disposição das mesmas permite a verificação desta árvore como uma partição independente. A segunda partição, identificada pelo nodo de índice 7, apresenta somente um operador de interesse ($\exists\cup$), podendo assim ser verificada em paralelo à primeira, sendo:

   - A verificação da partição identificada pelo índice 2 é realizada através da execução das operações de índices 3 ($\exists\Box$) e 5 ($\exists\bigcirc$) distribuindo-se a computação destes operadores entre coordenador e respectivo grupo de trabalho. A computação das operações 2, 4 e 6 é realizada pelo próprio coordenador, conforme já descrito para as propriedades anteriores.

- Para verificação da partição identificada pelo índice de número 7, o algoritmo procede de maneira semelhante onde o coordenador executa as operações de negação e conjunção lógica, distribuindo a computação da operação de índice 8 ($\exists\cup$), participando desta na exploração dos estados que satisfazem as sub-fórmulas Φ_1 e Φ_2.

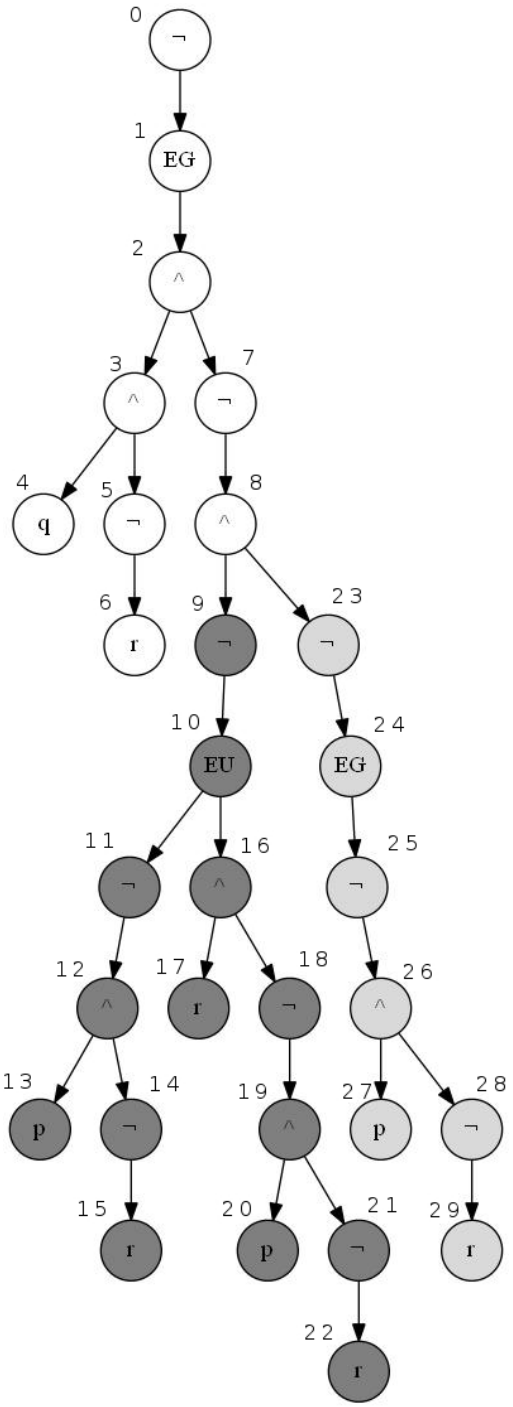


Figura 4.2 – Árvore sintática que representa a Propriedade 4.2 transcrita para ENF.

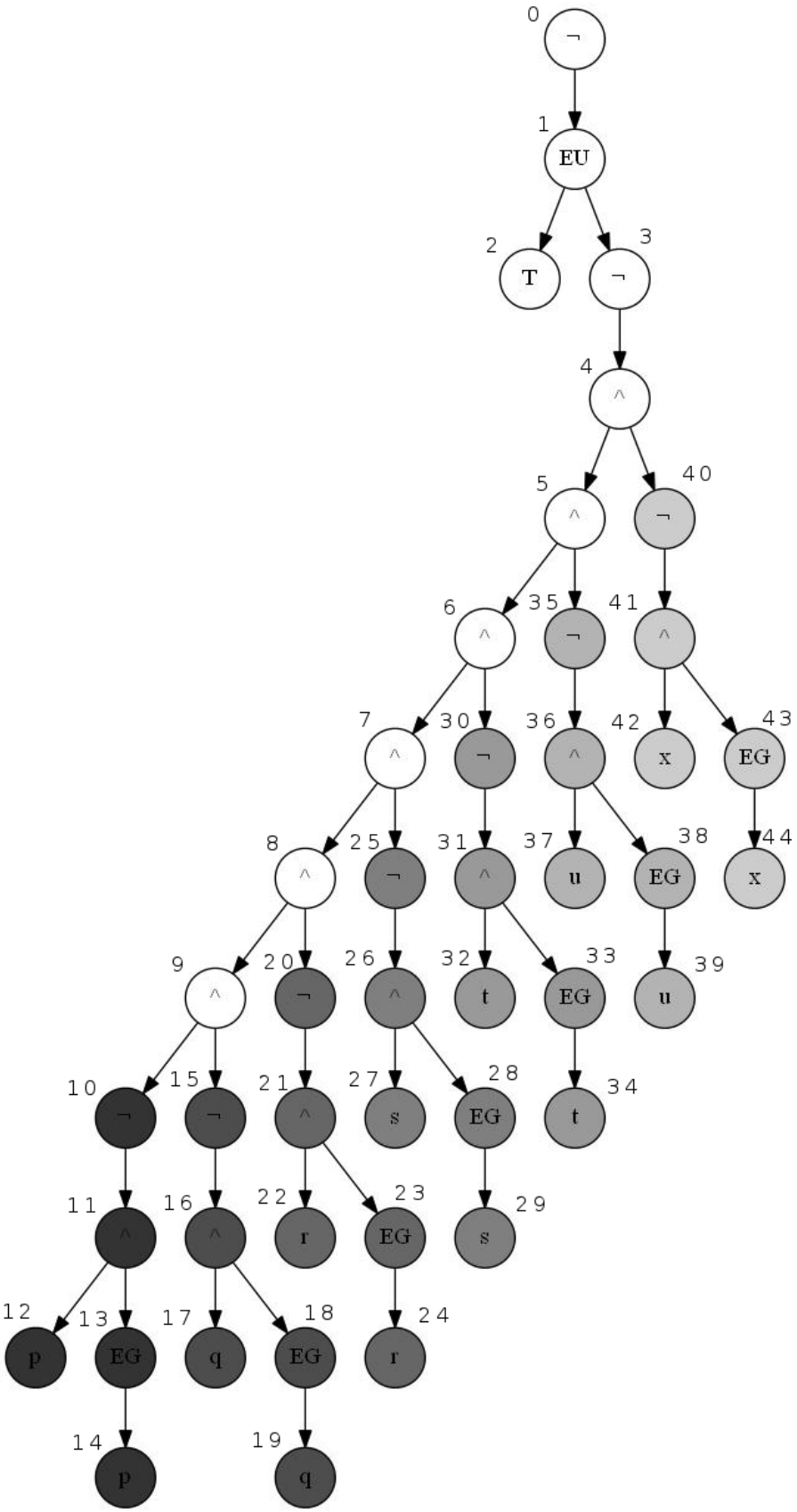


Figura 4.3 – Árvore sintática que representa a Propriedade 4.3 transcrita para ENF, onde as letras de p a x representam proposições atômicas. $N = 7$.

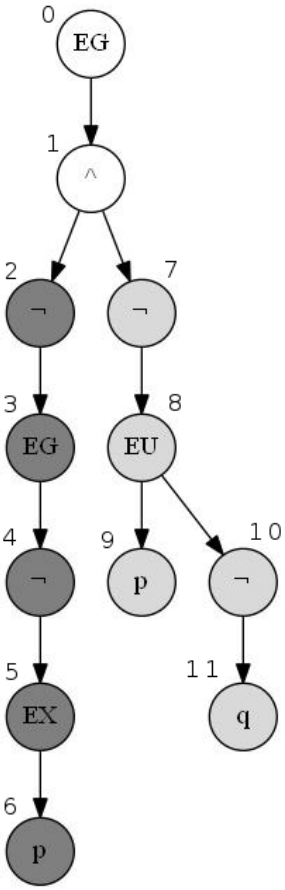


Figura 4.4 – Árvore sintática que representa a Propriedade 4.4 transcrita para ENF.

5. EXPERIMENTOS

Os experimentos para medição de desempenho com as abordagens paralelas neste trabalho descritas foram planejados utilizando uma abordagem estatística para validação dos resultados obtidos. Neste sentido, este capítulo apresenta a descrição do ambiente utilizado para condução dos experimentos (Seção 5.4), a abordagem de intervalos de confiança para estimar uma média do tempo de execução (Seção 5.1), determinação do tamanho amostral de dados a serem coletados (Seção 5.2), medidas para avaliação de desempenho (Seção 5.3), as propriedades CTL utilizadas nos experimentos (Seção 5.5) e, por fim, a discussão acerca dos resultados obtidos (Seção 5.6).

5.1 Intervalos de Confiança

Uma estimativa pontual, ou seja, baseada em um único valor tal como a média de n execuções de um algoritmo, não fornece um dado confiável acerca da exatidão deste dado para aproximação de um parâmetro populacional. Uma vez que o tempo de execução de programas paralelos tende a sofrer variações as quais podem ser causadas por diversos fatores tais como latência de rede, interferência por execução de rotinas padrão do sistema operacional, dentre outros, é importante a adoção de estimativas intervalares que garantam a qualidade dos dados coletados. Para Triola e Flores [51], adotando-se um nível de confiança é possível identificar uma taxa de sucesso para um procedimento experimental a fim de calcular um intervalo de confiança. Para este procedimento adotou-se um nível de confiança de 95%, para o qual o valor crítico ($z_{\alpha/2}$) é 1.96. Um valor crítico é um número na fronteira que separa estatísticas amostrais que têm chance de ocorrer daquelas que não têm.

Para cálculo do intervalo de confiança é necessário calcular anteriormente o desvio padrão (σ), através da seguinte fórmula:

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}} \tag{5.1}$$

Conhecido o desvio padrão, é possível calcular a margem de erro (E) para a média populacional (μ), utilizando-se a seguinte fórmula:

$$E = z_{\alpha/2} . \frac{\sigma}{\sqrt{n}} \tag{5.2}$$

Conhecida a margem de erro, o intervalo de confiança pode ser estimado através da seguinte fórmula:

$$\bar{x} - E < \mu < \bar{x} + E \tag{5.3}$$

onde $\bar{x}$ equivale a média de execuções do algoritmo, ou seja, a média amostral. Desta maneira, é possível estar 95% confiante que o verdadeiro valor de μ esteja entre $\bar{x}$ - E e $\bar{x} + E$, significando

que, selecionando-se várias amostras aleatórias de mesmo tamanho e construindo-se os respectivos intervalos de confiança, certamente 95% casos conterão o valor de μ.

5.2 Tamanho Amostral

O tamanho de uma amostra é relacionado com a quantidade necessária de dados a serem coletados para determinar uma média. Seu emprego visa garantir maior confiabilidade da média e reduzir o tempo gasto coletando dados desnecessários (neste caso realizando execuções desnecessárias do algoritmo). Uma vez que, para construção dos intervalos de confiança lidou-se com σ (desvio padrão populacional) desconhecido, uma das alternativas é calculá-lo através de um estudo piloto, com base nos 31 primeiros valores amostrais selecionados aleatoriamente [51]. Assim, para cada experimento 40 execuções foram conduzidas para determinar o desvio padrão (σ). O tamanho amostral (n) é calculado utilizando-se a seguinte fórmula:

$$n = \left[\frac{z_{\alpha/2}\sigma}{E}\right]^2 \tag{5.4}$$

onde E é a margem de erro calculada para 40 execuções e $z_{\alpha/2}$ é 1.96 para um nível de confiança de 95%, conforme relatado anteriormente. O resultado do cálculo pode ser um número não inteiro, devendo ser arredondado para o número inteiro mais próximo [51]. Desta maneira, para as várias execuções possíveis do algoritmo em questão, é necessário coletar somente uma amostra aleatória de tamanho n, tendo-se 95% de confiança que a média de execuções $\bar{x}$ estará abaixo do desvio padrão (σ) da verdadeira média de execuções (μ). Maiores detalhes acerca de estimativas intervalares e cálculo de tamanhos amostrais podem ser encontrados em [51] e [33].

5.3 Medidas para Avaliação de Desempenho

Para análise de programas paralelos, uma comparação com o tempo de execução da implementação sequencial é especialmente importante para ver o benefício obtido através do paralelismo [42]. Tal comparação é muitas vezes baseada na redução relativa do tempo de execução sendo expressada pela noção de *Speed-up* (fator de aceleração). O cálculo do fator de aceleração (Sp) de um programa paralelo é dado pela seguinte fórmula:

$$S_p = \frac{T_1}{T_p} \tag{5.5}$$

onde p é o número de processadores utilizados para solução do problema, T_1 é o tempo gasto pelo algoritmo sequencial e T_p é o tempo gasto pelo algoritmo paralelo para p processadores, sendo este último normalmente construído a partir da implementação sequencial. O conceito de *Speed-up* é

utilizado tanto para uma análise teórica de algoritmos quanto para a avaliação prática de programas paralelos [42].

5.4 Instrumentação e Ambiente Utilizado

Para condução dos experimentos adotou-se uma máquina integrada ao Laboratório de Alto Desempenho (LAD[1]) da PUCRS. A máquina utilizada (atlantica) é um *cluster* composto por 16 máquinas *Dell PowerEdge R610* possuindo dois processadores *Intel Xeon Quad-Core E5520 de 2.27 GHZ* com tecnologia *Hyper-Threading* cada. Conta com sistema operacional *Ubuntu Linux 10.04 server 64 bits* e interface de troca de mensagens (*Message Passing Interface*) - *OpenMPI* versão 1.4.5. Cada máquina possui 16 GB de memória e são interligadas por redes *Gigabit-Ethernet*.

Conforme descrito no Capítulo 4, a troca de estados entre as máquinas empregadas é feita através de acesso à disco onde armazena-se a estrutura simbólica que representa o conjunto de estados a ser trafegado. Por questões de disponibilidade, utilizou-se um disco compartilhado para armazenamento de tais dados. Embora o uso de discos com estas características possa apresentar interferências afetando operações de entrada e saída, variações significativas foram detectadas somente quando da execução de grandes quantidades destas operações. Resultados que comprovam este comportamento serão discutidas na Seção 5.6.

Apesar de ser possível o tráfego de tais estruturas simbólicas através de mensagens evitando-se operações de acesso à disco, operações intermediárias de manipulação e armazenamento em estruturas que possam ser trafegadas pelo ambiente MPI necessitariam ser executadas. Além deste fator, à medida que se aumenta a complexidade do modelo a ser verificado, é natural que as estruturas utilizadas para armazenamento dos estados venham a crescer, possivelmente pesando o tráfego de mensagens contendo tais informações.

Por razões de disponibilidade e concorrência com os demais usuários do laboratório, apenas dois nodos da máquina foram utilizados durante os experimentos. Desta maneira, devido à presença de processadores que encapsulam mais de um núcleo de processamento no mesmo *chip* em tais nodos, processos MPI acabam sendo alocados dentro de um mesmo processador, entretanto em núcleos distintos. Na condução dos experimentos foram gerados intervalos de confiança com nível de confiança de 95% para 40 execuções para ambos os algoritmos, paralelos e sequencial. Foram utilizados 1, 3, 5, 7, 9, 11, 13, e 15 processos relativos à 1 processo mestre e $N - 1$ escravos.

5.5 Propriedades CTL para Modelos SAN

Para demonstrar o comportamento dos algoritmos neste trabalho propostos, propriedades CTL foram escritas para os modelos apresentados no Apêndice A. Para definição do conjunto de proposições atômicas que compõe uma propriedade, expressões SAN são utilizadas.

[1] Acesso em: http://www.lad.pucrs.br

5.5.1 Propriedades CTL para o modelo do Jantar dos Filósofos

Para o modelo do clássico problema do Jantar dos Filósofos (Apêndice A.1), cinco propriedades foram escritas. Para expressar o comportamento desejado, as seguites proposições atômicas foram definidas:

$$Phil_i N\tilde{a}oCome = (st\ Phil_i\ != Left);$$

$$Phil_i Comendo = (st\ Phil_i == Left);$$

$$VizinhosComendo = ((st\ Phil_i == Left)\ \&\&\ (st\ Phil_{i+1} == Left));$$

A proposição $Phil_i N\tilde{a}oCome$ denota que o autômato que representa a situação do filósofo i está em um estado diferente de $Left/Right$, representando que não está comendo, de acordo com sua orientação. A proposição $Phil_i Comendo$ denota a situação contrária. A proposição $VizinhosComendo$ denota que os autômatos que representam a situação dos filósofos vizinhos i e $i+1$ estão no estado $Left$, representando que estão comendo ao mesmo tempo.

Com a definição e descrição das proposições atômicas acima, pode-se descrever algumas propriedades a serem verificadas sobre este modelo:

$$\exists\Box(Phil_i N\tilde{a}oCome) \tag{5.6}$$

Existe um caminho possível onde, em todos os estados, o filósofo $Phil_i$ não come. A propriedade CTL representada pela Equação 5.6 verifica a possibilidade do filósofo $Phil_i$ nunca atingir o estado $Left$, ou seja, nunca comer, caracterizando Postergação Indefinida (*Starvation*). Esta propriedade respeita naturalmente a Forma Normal Existencial.

$$\exists\Diamond(VizinhosComendo) \tag{5.7}$$

Existe um caminho onde, futuramente, filósofos vizinhos comem ao mesmo tempo. A Propriedade 5.7 testa se o modelo em questão é mutuamente exclusivo. Sendo falsa, garante este comportamento. Esta fórmula é representada em ENF da seguinte maneira: $\exists(true \cup VizinhosComendo)$.

$$\exists \bigcirc (Phil_i N\tilde{a}oCome) \tag{5.8}$$

Existe um caminho possível onde, no próximo estado, o filósofo $Phil_i$ está em um estado diferente de $Left$. A Propriedade 5.8 não possui sentido semântico e não visa a verificação de comportamentos sobre o modelo. Entretanto, foi definida visando cobrir a implementação paralela para o operador $\exists\bigcirc$. Esta propriedade respeita naturalmente a Forma Normal Existencial.

$$\forall(Phil_iComendo \cup (Phil_{i-1}NaoCome \wedge Phil_{i+1}N\tilde{a}oCome)) \quad (5.9)$$

Para todos os caminhos, o filósofo $Phil_i$ come até que os filósofos vizinhos ($i-1$ e $i+1$) estejam em um estado diferente de $Left$. A propriedade 5.9 visa verificar se em todos os caminhos possíveis um filósofo i come até que, ou melhor, enquanto que, seus vizinhos $i-1$ e $i+1$ não atinjam o estado que indica que estão comendo. Ainda, a propriedade foi definida visando medição de desempenho através do algoritmo de distribuição de partições da árvore sintática, conforme descrito na Seção 4.2. Sua fórmula equivalente em ENF é representada da seguinte forma: $\neg\exists(\neg(Phil_{i-1}N\tilde{a}oCome \wedge Phil_{i+1}N\tilde{a}oCome) \cup (\neg(Phil_iComendo) \wedge \neg(Phil_{i-1}NaoCome \wedge Phil_{i+1}N\tilde{a}oCome))) \wedge \neg\exists\Box\neg(Phil_{i-1}NaoCome \wedge Phil_{i+1}N\tilde{a}oCome)$. A Figura 4.1 representa a árvore sintática equivalente.

$$\forall\Box(Phil_iComendo \rightarrow \forall\Diamond(Phil_{i+1}N\tilde{a}oCome)) \quad (5.10)$$

Para todos os caminhos, em todos os estados, caso o filósofo i esteja no estado $Left$, isto implica em, para todos os caminhos, no futuro, o filósofo vizinho $i+1$ não atingir o estado que lhe permite comer. Esta propriedade testa a possibilidade do filósofo $i+1$ ficar bloqueado indefinidamente não sendo possível comer.

Equivalência em ENF para a fórmula: $\neg\exists(true \cup (Phil_iComendo \wedge \exists\Box\neg Phil_{i+1}N\tilde{a}oCome)$.

5.5.2 Propriedades CTL para o modelo *Ad Hoc Wireless Networks*

Propriedades foram escritas para o modelo de uma rede de nodos *wireless Ad Hoc*, apresentado no Apêndice A.2. Para tanto, as seguintes proposições atômicas foram definidas:

$$Nodo_iTransmite = (st\ MN_i == T);$$

$$Nodo_iRecebe = (st\ MN_i == R);$$

$$IntervaloTransmiss\tilde{a}o = (nb\ [MN_{i-2}\ ...\ MN_{i+3}]\ T == 1);$$

A proposição atômica $Nodo_iTransmite$ denota que o autômato que representa a situação do nodo i na cadeia de nodos está no estado T, o qual simboliza a transmissão de pacotes. A proposição atômica $Nodo_iRecebe$ denota que o autômato i está no estado R, indicando a recepção de pacotes pelo nodo que está sendo representado. A proposição atômica $IntervaloTransmiss\tilde{a}o$ denota que o número de autômatos que estão no estado T (transmitindo) no intervalo que vai de $i-2$ até $i+3$ é igual a 1.

Com as proposições acima definidas, as seguintes propriedades CTL foram escritas:

$$\forall\Box(Nodo_1Transmite \rightarrow \forall\Diamond(Nodo_NRecebe)) \tag{5.11}$$

Para todos os caminhos, em todos os estados, caso o nodo de índice 1 transmitir pacotes, isto implica em, para todos os caminhos, no futuro, o nodo N (último nodo da cadeia) recebê-los. Esta propriedade (5.11) visa verificar se, caso o primeiro nodo da cadeia transmitir pacotes, os mesmos serão recebidos pelo último nodo da cadeia. Vale destacar que a propriedade não garante recepção total dos pacotes enviados uma vez que não há uma forma de controlar a quantidade de pacotes gerados pelo primeiro nodo e recebidos pelo último nodo da cadeia. Esta fórmula é representada em ENF pela seguinte fórmula: $\neg\exists(true \cup (Nodo_1Transmite \wedge \exists\Box\neg Nodo_NRecebe)$.

$$\forall(\neg Nodo_NRecebe \cup Nodo_1Transmite) \tag{5.12}$$

Para todos os caminhos, o último nodo da cadeia não atinge o estado R, ou seja, não recebe pacotes, até que pelo menos uma transmissão seja realizada pelo primeiro nodo. Esta propriedade (5.12) testa a precedência na recepção de pacotes onde, o último nodo da cadeia pode atingir o estado R (recebendo) se e somente se o primeiro nodo da cadeia ter anteriormente realizado transmissões. Sua fórmula equivalente em ENF é representada da seguinte forma: $\neg\exists(\neg Nodo_1Transmite \cup (Nodo_NRecebe \wedge \neg Nodo_1Transmite)) \wedge \neg\exists\Box\neg Nodo_1Transmite$.

$$\forall\Box(Nodo_iTransmite \rightarrow IntervaloTransmissão) \tag{5.13}$$

Para todos os caminhos, em todos os estados, caso um nodo i esteja no estado transmitindo (T), então o número de nodos que estão transmitindo no intervalo entre $i-2$ e $i+3$ é igual a um. Esta propriedade (5.13) testa se a transmissão de pacotes respeita as regras de interferência do protocolo de comunicação adotado. Caso haver mais de um nodo realizando transmissões no intervalo acima citado, isto indica a possibilidade de perda de pacotes, a qual é ocasionada pela interferência *MAC*. Para maiores detalhes acerca do protocolo e do modelo adotado, recomenda-se a leitura de [21]. Equivalência em ENF para a propriedade: $\neg\exists(true \cup (Nodo_iTransmite \wedge \neg IntervaloTransmissão))$.

5.5.3 Propriedades CTL para o modelo Linha de Produção

Para o modelo que representa uma linha de produção, apresentado no Apêndice A.3, duas propriedades foram definidas: uma para verificar a vivacidade parcial do modelo, e outra a vivacidade do modelo em sua totalidade. Para expressar os comportamentos desejados, algumas proposições atômicas foram definidas:

$$Estação_iBloqueada = (st\ M_i == Bloqueada);$$

$$Estação_i Desbloqueada = (st\ M_i\ != Bloqueada);$$

onde $(st\ M_i == Bloqueada)$ representa o estado $1, 2$ e $(st\ M_i\ != Bloqueada)$ representa que o respectivo autômato está em um estado diferente de $1, 2$. Mais detalhes são encontrados no Apêndice A.3.

A proposição atômica $Estação_i Bloqueada$ denota que o autômato que representa o comportamento da estação i está no estado $1, 2$. A proposição $Estação_i Desbloqueada$, por sua vez, denota o contrário, o que indica que a estação não está bloqueada. Tendo as proposições atômicas acima definidas, as seguintes propriedades CTL podem ser escritas:

$$\forall\Box(Estação_i Bloqueada \rightarrow \forall\Diamond(Estação_i Desbloqueada)) \tag{5.14}$$

Para todos os caminhos, em todos os estados, caso a estação M_i bloquear isto implica em, para todos os caminhos, no futuro, a estação desbloquear. Como descrito no Apêndice A.3, uma estação bloqueada não permite que uma estação precedente prossiga. Desta forma, a Propriedade 5.14 verifica a possibilidade de uma estação bloquear indefinidamente a estação anterior. Equivalência em ENF para a fórmula CTL: $\neg\exists(true \cup (Estação_i Bloqueada \wedge \exists\Box\neg Estação_i Desbloqueada)$.

$$\forall\Box\Big[\bigwedge_{\forall_2 \leq i \leq N}(Estação_i Bloqueada \rightarrow \forall\Diamond(Estação_i Desbloqueada))\Big] \tag{5.15}$$

Para todos os caminhos, em todos os estados, caso a estação M_i bloquear, isto implica em, para todos os caminhos, no futuro, a estação M_i desbloquear. Para testar a vivacidade no modelo em sua totalidade, é necessário verificar para cada estação i, partindo de $i = 2$, se a referida estação não está bloqueando a estação precedente.

Para exemplificação, pode-se considerar um modelo para três estações. A propriedade representada pela Equação 5.15 verificará se, para todos os caminhos, em todos os estados, a conjunção de $(Estação_i Bloqueada \rightarrow \forall\Diamond(Estação_i Desbloqueada))$ para todo $2 \leq i \leq N$ é verdade. Conforme informado na descrição deste modelo (Apêndice A.3), o índice i inicia em 2 uma vez que a estação M_1 não é modelada. A fórmula ENF equivalente para esta propriedade pode ser visualizada na Figura 4.3, para N $= 8$. Para condução dos experimentos com este modelo, as estações foram modeladas com *buffer* com capacidade 1. Para maiores detalhes sobre o problema e respectivo modelo SAN recomenda-se a leitura de [23].

5.6 Resultados Obtidos

Nesta seção os resultados obtidos para as propriedades definidas na Seção 5.5 são tabelados e posteriormente discutidos. Os resultados estão dispostos nas tabelas da seguinte forma: a coluna Processadores apresenta a quantidade de processadores utilizados para verificação da propriedade em questão. As colunas μ*Tempo*, σ e *Confiança de 95%* apresentam a média de tempo, desvio padrão

e intervalo de confiança ambos calculados para as amostras de tempo coletadas. A coluna *speed-up* apresenta o valor do fator aceleração obtido para p processadores. Por fim, as colunas *Pico Memória* e μ*Memória* apresentam o valor de memória relativo à máquina (processo) que mais consumiu e a média de consumo, respectivamente. A Seção 5.7 apresenta possíveis causas identificadas para os ganhos de desempenho elevados obtidos.

Os tempos relativos à compilação do modelo SAN, geração do espaço de estados atingível e execução da função de rotulação (*Labelling Function*) estão incluídos nos valores de tempo apresentados, portanto não serão mostrados separadamente, pois não representam dados significativos em relação ao tempo de verificação.

5.6.1 Resultados para o modelo do Jantar dos Filósofos

Os resultados obtidos para as propriedades definidas para o modelo do Jantar dos Filósofos (Seção 5.5.1) são descritos abaixo.

Tabela 5.1 – Resultados para a Propriedade 5.6 (*Starvation*) para o modelo do Jantar dos Filósofos.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	24,90	0,39	$24{,}77 < \mu < 25{,}02$	1,00	235,31	-
3	14,99	0,29	$14{,}90 < \mu < 15{,}09$	1,66	162,18	101,49
5	6,86	0,37	$6{,}74 < \mu < 6{,}97$	3,62	108,93	77,62
7	4,75	1,14	$4{,}40 < \mu < 5{,}11$	5,23	81,75	62,22
9	3,64	0,51	$3{,}48 < \mu < 3{,}80$	6,82	66,83	51,94
11	3,45	0,42	$3{,}32 < \mu < 3{,}59$	7,20	56,40	45,27
13	3,10	0,41	$2{,}97 < \mu < 3{,}22$	8,03	48,74	36,36
15	2,91	0,55	$2{,}73 < \mu < 3{,}08$	8,55	44,09	33,03
14 Filósofos ($\|S\| = 195.025$)						
1	147,00	6,67	$144{,}93 < \mu < 149{,}07$	1,00	609,98	-
3	78,78	2,42	$78{,}03 < \mu < 79{,}53$	1,86	370,28	262,96
5	28,65	0,99	$28{,}34 < \mu < 28{,}96$	5,13	280,25	201,24
7	15,75	0,54	$15{,}58 < \mu < 15{,}92$	9,32	210,19	161,27
9	10,69	0,41	$10{,}56 < \mu < 10{,}82$	13,74	171,75	134,68
11	9,05	0,26	$8{,}97 < \mu < 9{,}13$	16,23	145,10	117,39
13	7,21	0,19	$7{,}15 < \mu < 7{,}27$	20,37	124,17	103,32
15	6,24	0,29	$6{,}15 < \mu < 6{,}33$	23,54	113,31	92,38
15 Filósofos ($\|S\| = 470.832$)						
1	$1{,}04 \times 10^3$	69,24	$1{,}02 \times 10^3 < \mu < 1{,}06 \times 10^3$	1,00	1.573,70	-
3	505,83	15,99	$500{,}88 < \mu < 510{,}79$	2,05	1.075,74	678,17
5	171,97	11,23	$168{,}49 < \mu < 175{,}45$	6,04	718,22	519,26
7	90,16	7,88	$87{,}71 < \mu < 92{,}60$	11,53	538,41	416,03
9	56,70	7,66	$54{,}33 < \mu < 59{,}08$	18,34	439,72	347,50
11	46,39	7,40	$44{,}09 < \mu < 48{,}68$	22,41	371,89	302,93
13	35,16	7,90	$32{,}71 < \mu < 37{,}61$	29,57	318,57	266,65
15	28,82	7,34	$26{,}54 < \mu < 31{,}10$	36,08	290,12	238,44

Os resultados obtidos para a Propriedade 5.6 ($\exists\Box(st\ Phil_i\ != Comendo)$ - *starvation*) são apresentados pela Tabela 5.1. Ganhos extremamente elevados para esta propriedade foram obtidos com a implementação paralela.

As causas para este comportamento derivam da natureza da propriedade e modelo aliados à aspectos de implementação. Uma vez que foram identificadas as execuções de apenas duas iterações para obtenção de ponto fixo para o operador em questão, habilita-se o algoritmo paralelo à coordenar

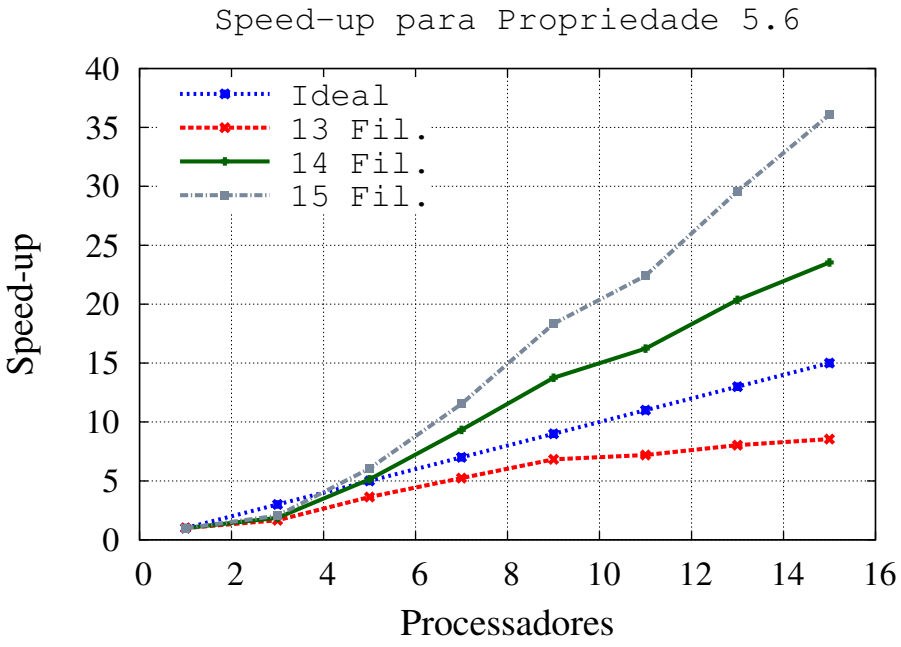


Figura 5.1 – Gráfico de *speed-up* para a Propriedade 5.6 (*starvation*).

a verificação de conjuntos extremamente pequenos de estados em cada máquina empregada, de maneira quase independente.

É possível conduzir testes para a propriedade em questão com tamanhos maiores de modelo tal como 16 filósofos, o qual possui espaço de estados atingível de 1.136.689 estados. Devido à grande quantidade de estados a ser verificada foram utilizados 30 processos (distribuídos homogeneamente em quatro nodos da máquina) na condução deste experimento, tendo tempo médio de verificação (cinco amostras) em torno de 61,14 segundos, e pico e média de consumo de memória de 415,40 MB e 355,24 MB, respectivamente. Não foram geradas estimativas intervalares e cálculo de fatores aceleração para tais testes devido ao tempo necessário para execução pela ferramenta sequencial.

A Tabela 5.2 apresenta os resultados obtidos para a Propriedade 5.7 (exclusão mútua). Sua equivalência em ENF é representada através da seguinte fórmula: $\exists(true \cup \Phi)$. Como descrito na Seção 4.1, o algoritmo para o operador $\exists\cup$ tem como característica avaliar o conjunto de estados pertencente à diferença entre Φ_1 e Φ_2 buscando por estados em que a intersecção de seus sucessores com Φ_2 seja diferente de vazio. Sendo esta condição verdadeira, os respectivos estados são adicionados ao conjunto que satisfaz Φ_2. Estes passos são executados repetidas vezes até que se obtenha um ponto fixo.

Como a proposição atômica $(st\ Phil_i == Left)$ && $(st\ Phil_{i+1} == Left)$ não é satisfeita em estado algum do modelo, a diferença entre $true$ (RSS) e Φ_2 $((st\ Phil_i == Left)$ && $(st\ Phil_{i+1} == Left))$ acaba sendo o próprio RSS. Desta maneira, cada processo escravo fica responsável pela avaliação de um conjunto pequeno de estados pertencente ao RSS e, como não há estados à serem adicionados ao conjunto Φ_2, o algoritmo executa apenas uma iteração de avaliação do conjunto.

Tabela 5.2 – Resultados para a Propriedade 5.7 (*Exclusão Mútua*) para o modelo do Jantar dos Filósofos.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	19,80	0,04	$19{,}78 < \mu < 19{,}81$	1,00	262,86	-
3	10,18	2,37	$9{,}44 < \mu < 10{,}91$	1,94	178,57	113,64
5	6,88	3,58	$5{,}77 < \mu < 7{,}99$	2,87	120,73	85,59
7	4,69	2,94	$3{,}78 < \mu < 5{,}61$	4,21	91,57	67,85
9	4,86	3,97	$3{,}63 < \mu < 6{,}09$	4,07	74,84	57,75
11	4,39	3,54	$3{,}29 < \mu < 5{,}49$	4,50	57,75	50,09
13	4,76	4,19	$3{,}46 < \mu < 6{,}06$	4,15	54,31	43,91
15	4,58	4,20	$3{,}28 < \mu < 5{,}88$	4,32	49,01	39,23
14 Filósofos ($\|S\| = 195.025$)						
1	117,00	4,76	$115{,}52 < \mu < 118{,}48$	1,00	690,21	-
3	49,91	4,74	$48{,}44 < \mu < 51{,}38$	2,34	467,43	298,52
5	23,71	3,82	$22{,}52 < \mu < 24{,}89$	4,93	314,92	224,88
7	18,65	5,23	$17{,}03 < \mu < 20{,}28$	6,27	238,28	178,24
9	13,39	4,29	$12{,}06 < \mu < 14{,}72$	8,73	194,69	151,79
11	11,73	4,39	$10{,}37 < \mu < 13{,}09$	9,97	163,90	131,70
13	10,96	4,91	$9{,}44 < \mu < 12{,}49$	10,66	141,49	115,45
15	11,70	4,87	$10{,}19 < \mu < 13{,}21$	9,99	127,45	103,13
15 Filósofos ($\|S\| = 470.832$)						
1	837,00	58,28	$818{,}94 < \mu < 855{,}06$	1,00	1.800,59	-
3	289,49	12,81	$285{,}52 < \mu < 293{,}46$	2,89	1.120,86	1.034,43
5	109,12	5,61	$107{,}38 < \mu < 110{,}86$	7,67	816,91	586,96
7	67,11	6,68	$65{,}04 < \mu < 69{,}19$	12,47	616,81	465,17
9	44,76	5,49	$43{,}06 < \mu < 46{,}46$	18,69	503,84	396,31
11	32,34	5,03	$30{,}78 < \mu < 33{,}90$	25,87	424,82	343,97
13	26,54	5,08	$24{,}96 < \mu < 28{,}11$	31,53	366,58	301,55
15	25,13	5,56	$23{,}40 < \mu < 26{,}85$	33,30	329,73	269,35

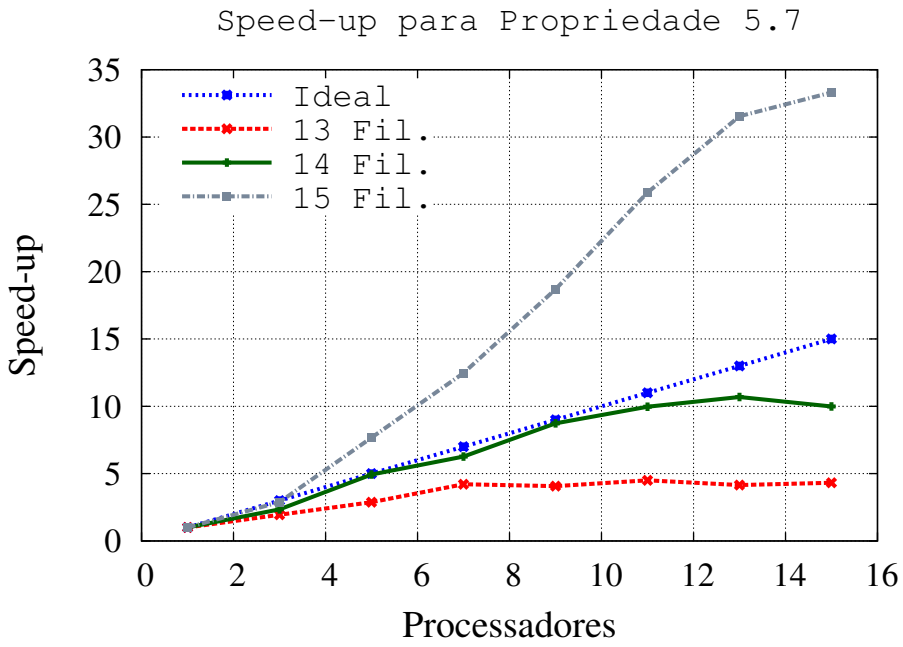


Figura 5.2 – Gráfico de *speed-up* para a Propriedade 5.7 (*Exclusão Mútua*).

Comunicação para verificação em paralelo é necessária somente para ditar o início e término do paralelismo, de forma que os ganhos sejam elevados. Os ganhos se mostraram mais favoráveis para 9, 11 e 13 processos.

Tabela 5.3 – Resultados para a Propriedade 5.8 para o modelo do Jantar dos Filósofos.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	26,20	0,06	$26{,}17 < \mu < 26{,}22$	1,00	336,21	-
3	11,41	0,05	$11{,}39 < \mu < 11{,}43$	2,29	225,09	145,98
5	5,66	0,11	$5{,}62 < \mu < 5{,}69$	4,62	151,11	109,75
7	4,16	0,17	$4{,}10 < \mu < 4{,}21$	6,29	113,91	86,79
9	3,39	0,05	$3{,}38 < \mu < 3{,}41$	7,70	88,49	73,90
11	3,02	0,15	$2{,}97 < \mu < 3{,}07$	8,66	78,66	64,00
13	2,87	0,18	$2{,}81 < \mu < 2{,}93$	9,12	68,91	56,23
15	2,90	0,22	$2{,}83 < \mu < 2{,}97$	9,01	60,94	50,28
14 Filósofos ($\|S\| = 195.025$)						
1	155,00	0,22	$154{,}93 < \mu < 155{,}07$	1,00	867,39	-
3	58,31	0,16	$58{,}26 < \mu < 58{,}36$	2,65	579,82	376,64
5	23,36	0,08	$23{,}33 < \mu < 23{,}39$	6,63	388,31	283,24
7	14,83	0,75	$14{,}60 < \mu < 15{,}07$	10,44	292,26	223,99
9	10,53	0,06	$10{,}51 < \mu < 10{,}55$	14,71	238,86	190,83
11	8,27	0,26	$8{,}18 < \mu < 8{,}35$	18,74	201,87	165,31
13	7,13	0,36	$7{,}02 < \mu < 7{,}25$	21,71	176,79	145,21
15	6,39	0,36	$6{,}27 < \mu < 6{,}50$	24,25	156,16	129,85
15 Filósofos ($\|S\| = 470.832$)						
1	$1{,}20 \times 10^3$	35,96	$1{,}19 \times 10^3 < \mu < 1{,}21 \times 10^3$	1,00	2.228,47	-
3	390,12	9,44	$387{,}20 < \mu < 393{,}05$	3,08	1.487,59	967,70
5	133,11	2,39	$132{,}37 < \mu < 133{,}85$	9,02	994,16	727,91
7	80,64	6,33	$78{,}68 < \mu < 82{,}60$	14,88	747,17	575,68
9	51,60	1,75	$51{,}06 < \mu < 52{,}15$	23,25	610,52	490,61
11	37,37	3,48	$36{,}29 < \mu < 38{,}45$	32,11	516,55	425,19
13	29,50	2,80	$28{,}63 < \mu < 30{,}37$	40,67	451,93	373,45
15	24,81	2,39	$24{,}07 < \mu < 25{,}55$	48,36	399,54	333,89

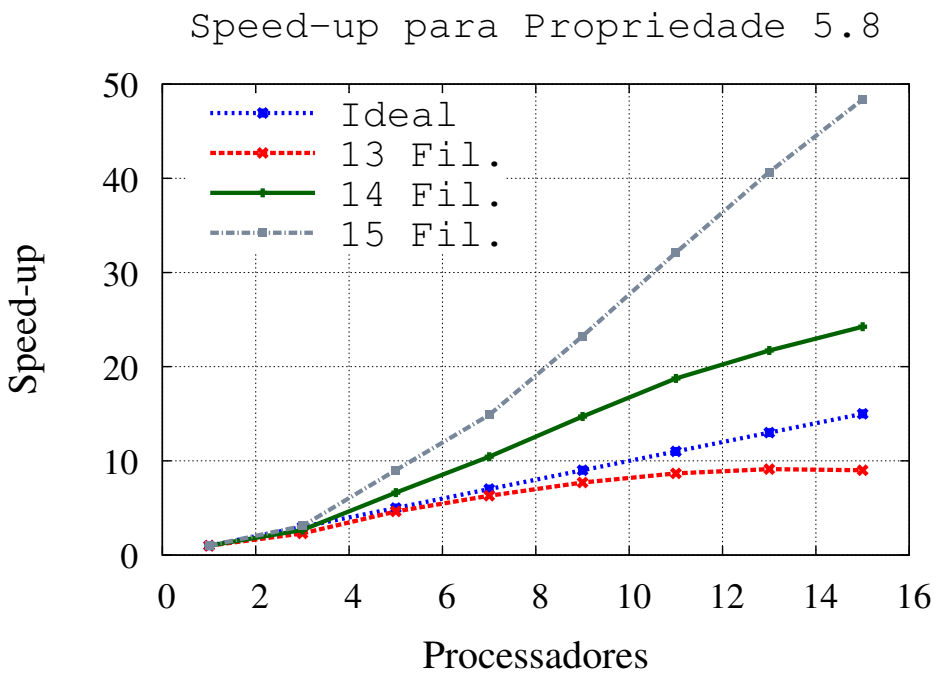


Figura 5.3 – Gráfico de *speed-up* para a Propriedade 5.8.

Os resultados para a Propriedade 5.8 ($\exists\bigcirc(st\ Phil_i\ != Left)$) são apresentados pela Tabela 5.3. Uma vez que o algoritmo paralelo para o operador $\exists\bigcirc$ particiona a computação do conjunto de estados pertencente ao RSS e, neste caso, não há necessidade de execução de rodadas

de sincronia entre processos escravos e mestre para cálculo de ponto fixo, a verificação de pequenos conjuntos de estados é feita de maneira independente pelos processos escravos.

Estando o ganho de aceleração posicionado acima do ideal, ou seja, apresentando *speed-ups* superlineares [42], um comparativo pode ser feito visando melhor analisar este comportamento para a referida propriedade. Tomando como referência o modelo com 13 filósofos, a verificação sequencial da propriedade para $|S| = 80.782$ estados apresenta média de tempo posicionada entre 26,17 e 26,22 segundos. Ao se considerar o modelo com 15 filósofos o qual possui $|S| = 470.832$ estados, proceder a verificação com 15 processos leva entre 24,07 e 25,55 segundos, para uma amostra de 40 execuções do algoritmo, uma vez que cada processo escravo conduz a verificação de uma porção em torno de 33.000 estados. Havendo comunicação somente para início e término da verificação para este operador, ganhos elevados podem ser esperados.

Testes foram conduzidos para o modelo com 16 filósofos ($|S| = 1.136.689$ estados). O tempo obtido para verificação com a implementação paralela leva em torno de 64,96 segundos empregando-se 30 processos distribuídos em quatro nodos da máquina, tendo pico e consumo médio de memória de 578,96 MB e 514,13 MB, respectivamente. O comportamento da implementação paralela deste operador apresentado pela Tabela 5.3 sugere que o aumento do grão para verificação implica em obtenção de elevados fatores aceleração.

Tabela 5.4 – Resultados para a Propriedade 5.10 para o modelo do Jantar dos Filósofos.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	26,26	0,11	$26{,}14 < \mu < 26{,}38$	1,00	284,94	-
3	14,50	0,57	$14{,}33 < \mu < 14{,}68$	1,81	192,15	124,40
5	7,57	0,36	$7{,}45 < \mu < 7{,}68$	3,47	130,37	97,37
7	5,63	0,43	$5{,}50 < \mu < 5{,}77$	4,65	97,03	80,00
9	4,88	0,36	$4{,}77 < \mu < 5{,}00$	5,37	80,80	59,80
11	4,62	0,39	$4{,}50 < \mu < 4{,}75$	5,67	69,40	60,18
13	4,62	0,58	$4{,}44 < \mu < 4{,}80$	5,67	61,35	53,33
15	4,69	0,38	$4{,}58 < \mu < 4{,}81$	5,59	55,29	44,28
14 Filósofos ($\|S\| = 195.025$)						
1	151,49	1,95	$149{,}53 < \mu < 153{,}45$	1,00	743,57	-
3	68,31	1,64	$67{,}80 < \mu < 68{,}82$	2,21	500,22	324,93
5	28,57	0,97	$28{,}27 < \mu < 28{,}88$	5,30	338,20	254,43
7	17,97	0,84	$17{,}71 < \mu < 18{,}23$	8,42	255,82	209,07
9	13,30	1,12	$12{,}96 < \mu < 13{,}65$	11,38	212,24	179,81
11	11,09	0,74	$10{,}86 < \mu < 11{,}32$	13,65	181,95	157,36
13	10,05	0,82	$9{,}79 < \mu < 10{,}30$	15,07	158,19	127,30
15	9,24	0,68	$9{,}03 < \mu < 9{,}46$	16,37	143,95	115,79
15 Filósofos ($\|S\| = 470.832$)						
1	$1{,}02 \times 10^3$	51,55	$9{,}65 \times 10^2 < \mu < 1{,}06 \times 10^3$	1,00	1.929,48	-
3	799,14	16,02	$794{,}18 < \mu < 804{,}11$	1,27	1.295,34	843,87
5	479,19	10,00	$449{,}73 < \mu < 508{,}65$	2,12	1.053,24	412,09
7	242,87	2,15	$203{,}68 < \mu < 282{,}05$	4,20	873,12	661,00
9	146,90	2,45	$146{,}24 < \mu < 147{,}57$	6,94	659,15	543,24
11	106,23	2,08	$91{,}84 < \mu < 120{,}63$	9,60	553,87	467,39
13	97,92	2,20	$97{,}17 < \mu < 98{,}68$	10,41	455,95	409,12
15	69,20	1,48	$68{,}56 < \mu < 69{,}85$	14,73	372,87	325,93

Os resultados obtidos para a Propriedade 5.10 são apresentados pela Tabela 5.4 e os respectivos fatores aceleração na Figura 5.4. Apesar do ganho obtido ser relativamente alto, é perceptível que, em comparação com outras propriedades para o mesmo modelo, o fator aceleração reduziu consideravelmente. Apesar do algoritmo executar um número relativamente baixo de rodadas

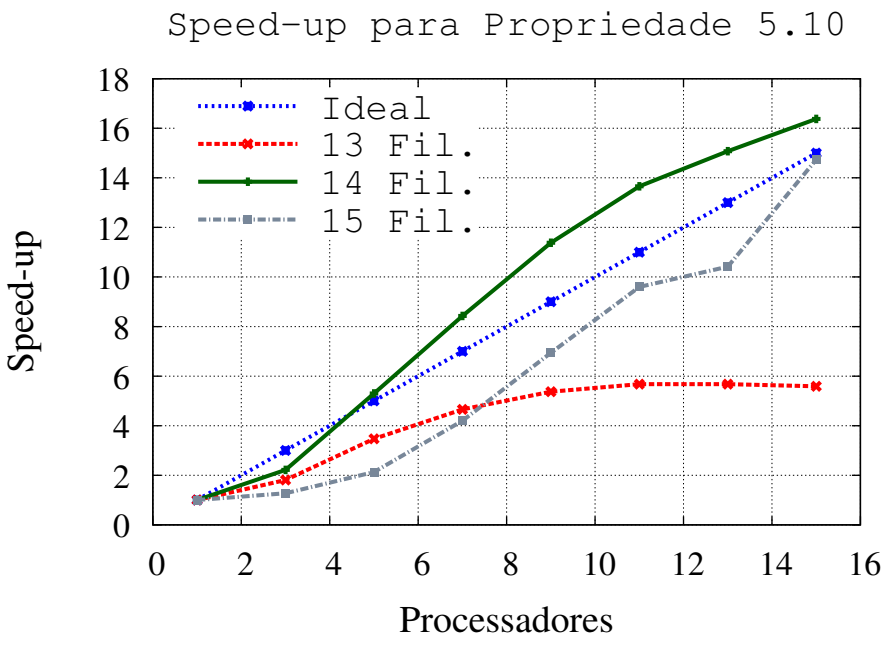


Figura 5.4 – Gráfico de *speed-up* para a Propriedade 5.10.

de sincronia (3 para o operador $\exists\Box$ e 1 para $\exists\cup$ para ambos os tamanhos de modelo), o ganho é reduzido devido às características da implementação, tais como: necessidade de troca de mensagens e acesso à disco para troca de estados e, ainda, a adoção de rotinas de comunicação síncrona.

A Tabela 5.5 apresenta os resultados de tempo e memória obtidos para a propriedade 5.9 para 15, 16 e 17 filósofos. A árvore sintática que representa esta fórmula transcrita para ENF (Figura 4.1) permite ser quebrada em duas partições para verificação em paralelo através do algoritmo descrito na Seção 4.2. Para um comparativo justo, os respectivos experimentos foram realizados também utilizando-se o algoritmo de distribuição da computação relativa aos operadores de lógica temporal (Seção 4.1), tendo os resultados apresentados na referida Tabela pelas informações contidas na parte superior (Sem quebra da árvore sintática). Vale destacar que o número de processos empregados na segunda abordagem (Com quebra da árvore sintática) representa a existência de 1 processo mestre e $N-1$ processos escravos, sendo os últimos dispostos em 2 grupos para computação das partições da árvore.

Pelos resultados apresentados pela tabela e respectivo Gráfico de *speed-up* (5.5), a abordagem de verificação sem particionamento da árvore sintática se mostrou mais favorável para esta propriedade. Este comportamento é obtido devido a maior quantidade de processos empregada na verificação dos operadores de maior custo computacional aliado à necessidade de execução de poucas rodadas de sincronia para obtenção de ponto fixo para tais operadores.

Para a segunda abordagem desenvolvida, ao se empregar apenas um processo na computação de cada partição da árvore sintática (resultados representados nas linhas que indicam as execuções com três processos) tempos levemente maiores que a implementação sequencial foram

Tabela 5.5 – Resultados para a Propriedade 5.9 para o modelo do Jantar dos Filósofos.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
Sem quebra da árvore sintática						
15 filósofos ($\|S\| = 470.832$)						
1	30,71	0,07	$30{,}69 < \mu < 30{,}74$	1,00	255,67	-
3	18,47	0,76	$18{,}23 < \mu < 18{,}71$	1,66	184,08	86,19
5	13,56	0,43	$13{,}42 < \mu < 13{,}69$	2,26	133,96	103,90
7	8,34	0,24	$8{,}27 < \mu < 8{,}42$	3,68	100,27	83,69
9	7,23	0,42	$7{,}10 < \mu < 7{,}36$	4,24	81,86	69,93
11	6,94	0,45	$6{,}79 < \mu < 7{,}08$	4,42	69,34	61,01
13	6,62	0,37	$6{,}50 < \mu < 6{,}73$	4,63	61,21	53,68
15	6,12	0,26	$6{,}04 < \mu < 6{,}20$	5,01	54,34	48,12
16 filósofos ($\|S\| = 1.136.689$)						
1	177,75	0,97	$177{,}45 < \mu < 178{,}05$	1,00	654,94	-
3	96,94	1,74	$96{,}40 < \mu < 97{,}49$	1,83	471,40	284,97
5	39,16	3,41	$38{,}10 < \mu < 40{,}22$	4,53	281,70	230,86
7	19,05	1,12	$18{,}70 < \mu < 19{,}40$	9,33	235,06	185,12
9	14,38	0,77	$14{,}14 < \mu < 14{,}62$	12,35	191,96	155,25
11	12,20	0,76	$11{,}96 < \mu < 12{,}43$	14,57	162,52	135,26
13	9,63	0,62	$9{,}44 < \mu < 9{,}82$	18,44	140,49	119,11
15	8,61	0,52	$8{,}45 < \mu < 8{,}77$	20,63	126,75	106,65
17 filósofos ($\|S\| = 2.744.210$)						
1	$1{,}27 \times 10^3$	120,96	$1{,}23 \times 10^3 < \mu < 1{,}30 \times 10^3$	1,000	1.672,34	-
3	684,76	26,78	$676{,}46 < \mu < 693{,}06$	1,85	1.203,24	727,97
5	209,40	11,70	$205{,}77 < \mu < 213{,}02$	6,06	799,60	590,31
7	109,10	4,76	$107{,}62 < \mu < 110{,}58$	11,64	598,42	473,25
9	64,56	3,25	$63{,}55 < \mu < 65{,}56$	19,68	488,49	396,97
11	50,59	2,34	$49{,}86 < \mu < 51{,}32$	25,12	413,90	345,90
13	37,56	2,23	$36{,}87 < \mu < 38{,}25$	33,83	357,88	304,62
15	29,96	1,93	$29{,}36 < \mu < 30{,}56$	42,40	322,54	272,79
Com quebra da árvore sintática						
15 filósofos ($\|S\| = 470.832$)						
1	30,71	0,07	$30{,}69 < \mu < 30{,}74$	1,00	255,67	-
3	33,67	1,49	$33{,}21 < \mu < 34{,}14$	0,91	255,59	85,25
5	23,16	1,76	$22{,}61 < \mu < 23{,}71$	1,32	184,11	66,73
7	14,07	1,54	$13{,}59 < \mu < 14{,}55$	2,18	145,91	56,77
9	11,33	1,16	$10{,}97 < \mu < 11{,}70$	2,70	122,94	50,00
11	9,50	1,03	$9{,}18 < \mu < 9{,}82$	3,23	105,15	43,92
13	7,94	0,51	$7{,}78 < \mu < 8{,}10$	3,86	92,08	38,87
15	8,14	1,26	$7{,}75 < \mu < 8{,}53$	3,77	83,12	35,15
16 filósofos ($\|S\| = 1.136.689$)						
1	177,75	0,97	$177{,}45 < \mu < 178{,}05$	1,00	654,94	-
3	191,38	8,31	$188{,}81 < \mu < 193{,}96$	0,93	654,85	218,34
5	98,72	2,04	$98{,}08 < \mu < 99{,}35$	1,80	471,43	170,99
7	48,80	0,38	$48{,}68 < \mu < 48{,}91$	3,64	372,71	145,53
9	34,29	0,61	$34{,}09 < \mu < 34{,}48$	5,18	314,00	128,26
11	26,15	0,90	$25{,}87 < \mu < 26{,}43$	6,79	268,56	112,66
13	19,19	0,61	$19{,}00 < \mu < 19{,}38$	9,26	235,09	99,69
15	16,52	0,81	$16{,}27 < \mu < 16{,}78$	10,75	206,24	90,16
17 filósofos ($\|S\| = 2.744.210$)						
1	$1{,}27 \times 10^3$	120,96	$1{,}23 \times 10^3 < \mu < 1{,}30 \times 10^3$	1,00	1.672,34	-
3	$1{,}33 \times 10^3$	65,77	$1{,}31 \times 10^3 < \mu < 1{,}35 \times 10^3$	0,95	1,650.25	557,47
5	722,96	118,62	$686{,}19 < \mu < 759{,}72$	1,75	1.203,27	436,80
7	335,03	22,26	$328{,}13 < \mu < 341{,}93$	3,79	949,21	371,88
9	234,71	18,44	$228{,}99 < \mu < 240{,}42$	5,41	799,63	327,96
11	165,80	14,91	$161{,}17 < \mu < 170{,}42$	7,66	683,97	288,01
13	123,87	21,43	$117{,}23 < \mu < 130{,}51$	10,25	598,45	254,84
15	106,45	13,48	$102{,}27 < \mu < 110{,}63$	11,93	525,62	230,51

obtidos. Estes resultados são obtidos devido a baixa quantidade de processos empregados em cada partição para computação e a necessidade de troca de mensagens entre as máquinas.

A Figura 5.6 apresenta um gráfico com o pico de memória para condução dos experimentos de forma sequencial e paralela para as Propriedades 5.6 (*starvation*), 5.7 (exclusão mútua), 5.8 e 5.10 para 15 filósofos e a propriedade 5.9 para 17 filósofos. À medida que se aumenta a quantidade de processos para computação das referidas propriedades é perceptível considerável redução do pico

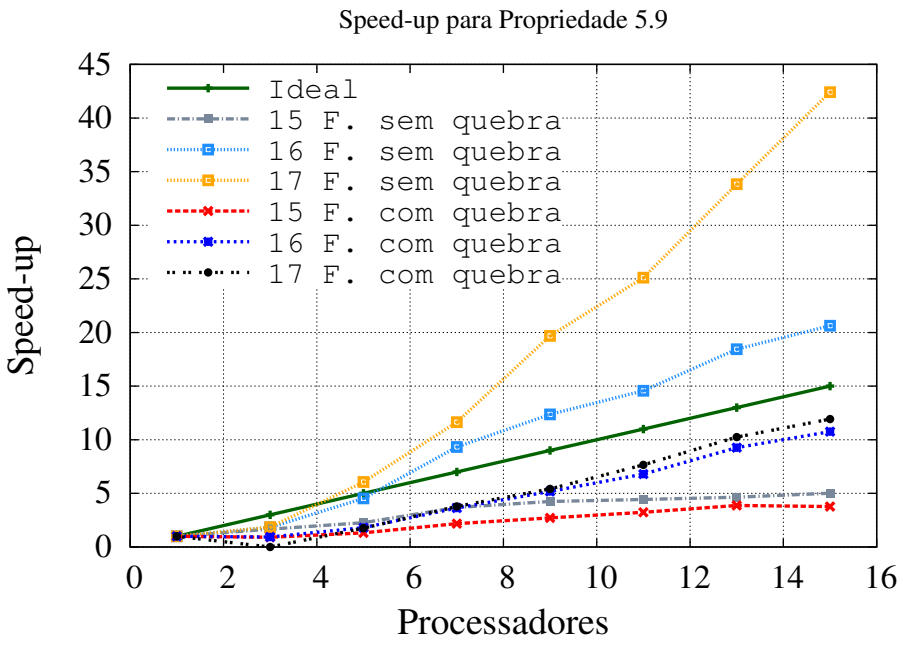


Figura 5.5 – Gráfico de *speed-up* para a Propriedade 5.9 com e sem particionamento da árvore sintática ENF.

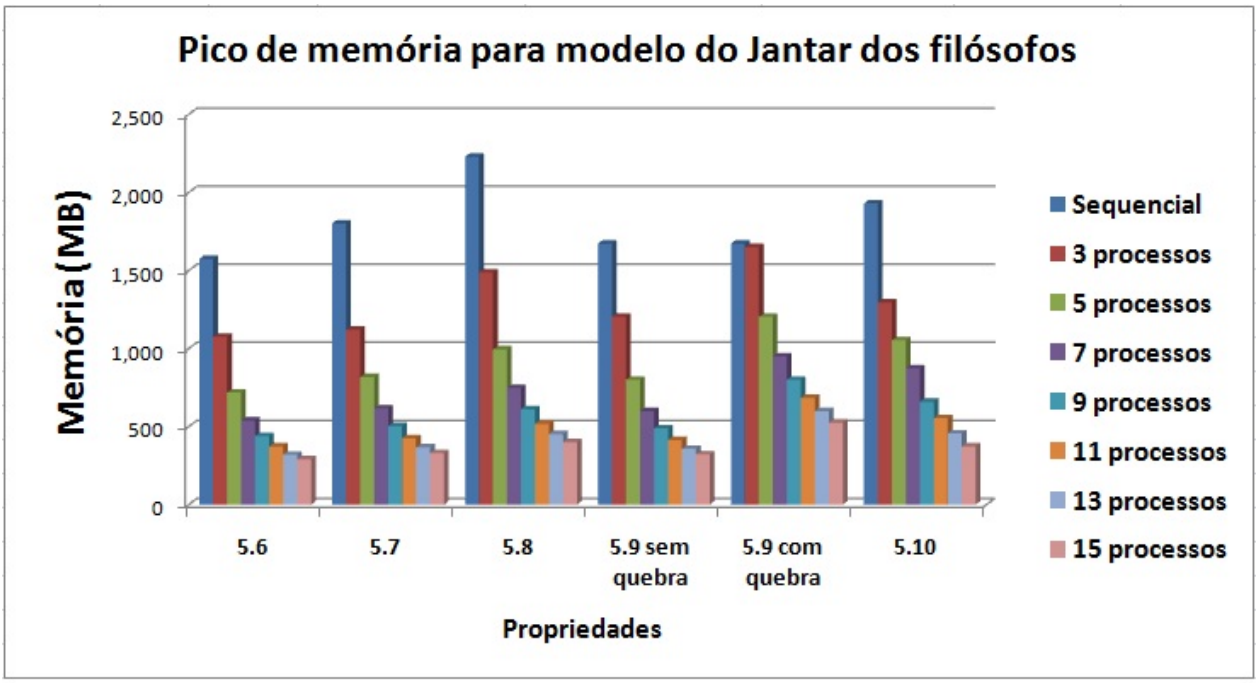


Figura 5.6 – Consumo de memória para as Propriedades 5.6, 5.7, 5.8 e 5.10 verificadas sob o modelo do Jantar dos Filósofos para 15 filósofos e Propriedade 5.9 para 17 filósofos com e sem particionamento (quebra e distribuição de ramos) da árvore sintática.

e média (vide tabelas) de memória, os quais tendem a estabilizar, indicando que estruturas básicas necessitam ser criadas na estrutura MDD para obtenção do resultado da verificação.

A média de memória apresenta valores muito inferiores aos valores de pico de memória. Este comportamento se justifica uma vez que o processo mestre realiza poucas operações sobre a estrutura MDD durante a verificação de propriedades, de forma que seu consumo de memória seja relativamente baixo em comparação aos processos escravos. Embora o consumo total de memória englobando-se todas as máquinas utilizadas seja relativamente alto em relação à implementação sequencial, as abordagens paralelas permitem distribuir o consumo, de maneira que os recursos computacionais em cada máquina possam ser melhor aproveitados.

Vale destacar que devido ao emprego de quantidades maiores de processos para computação dos operadores de maior custo computacional para a Propriedade 5.9 quando do emprego da implementação sem quebra da árvore sintática o consumo de memória pela primeira abordagem naturalmente tende a ser inferior em relação à segunda implementação.

5.6.2 Resultados para o modelo *Ad Hoc Wireless Networks*

Nesta seção são apresentados e discutidos os resultados obtidos para as propriedades definidas para o modelo *Ad Hoc Wireless Networks* (Seção 5.5.2).

Tabela 5.6 – Resultados para a Propriedade 5.11 para o modelo *Ad Hoc Wireless Networks*.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
20 nodos ($\|S\| = 12.102$)						
1	24,57	0,17	$24{,}51 < \mu < 24{,}62$	1,00	294,69	-
3	29,45	0,55	$29{,}28 < \mu < 29{,}62$	0,83	338,23	170,08
5	25,42	0,38	$25{,}30 < \mu < 25{,}53$	0,96	320,09	135,56
7	24,87	1,69	$24{,}34 < \mu < 25{,}39$	0,98	269,88	102,96
9	17,11	0,17	$17{,}06 < \mu < 17{,}17$	1,43	194,87	62,02
11	17,19	0,20	$17{,}12 < \mu < 17{,}25$	1,43	173,12	61,56
13	16,59	0,42	$16{,}46 < \mu < 16{,}72$	1,48	153,37	56,35
15	13,99	0,35	$13{,}88 < \mu < 14{,}10$	1,75	130,34	41,64
22 nodos ($\|S\| = 31.682$)						
1	131,77	0,47	$131{,}22 < \mu < 131{,}52$	1,00	946,22	-
3	167,18	6,24	$165{,}25 < \mu < 169{,}12$	0,79	1.142,69	404,14
5	136,24	3,36	$135{,}20 < \mu < 137{,}28$	0,96	1.122,87	341,10
7	141,25	1,98	$140{,}64 < \mu < 141{,}87$	0,93	1.114,06	409,03
9	74,16	2,10	$73{,}51 < \mu < 74{,}81$	1,78	679,99	210,83
11	78,25	2,35	$77{,}52 < \mu < 78{,}98$	1,68	628,72	219,21
13	58,45	1,22	$58{,}07 < \mu < 58{,}83$	2,25	505,03	180,06
15	42,27	5,84	$40{,}46 < \mu < 44{,}08$	3,12	443,6	134,65
24 nodos ($\|S\| = 82.946$)						
1	951,97	2,315	$951{,}26 < \mu < 951{,}61$	1,00	2.999,13	-
3	$1{,}35 \times 10^3$	74,58	$1{,}32 \times 10^3 < \mu < 1{,}37 \times 10^3$	0,707	4.061,97	1.455,08
5	$1{,}25 \times 10^3$	93,17	$1{,}22 \times 10^3 < \mu < 1{,}28 \times 10^3$	0,872	3.818,37	1.286,64
7	$1{,}09 \times 10^3$	68,90	$1{,}07 \times 10^3 < \mu < 1{,}11 \times 10^3$	0,762	3.671,23	1.140,34
9	613,85	52,71	$597{,}52 < \mu < 630{,}19$	1,55	3.214,64	766,67
11	462,00	35,25	$451{,}08 < \mu < 472{,}93$	2,06	2.578,49	734,63
13	457,89	23,32	$450{,}66 < \mu < 465{,}12$	2,07	2.172,86	644,92
15	339,98	22,04	$333{,}15 < \mu < 346{,}82$	2,80	2.162,01	516,03

A Tabela 5.6 apresenta os resultados obtidos para a Propriedade 5.11 ($\forall\Box(Nodo_1 Transmite \rightarrow \forall\Diamond(Nodo_N Recebe))$) para o modelo Ad Hoc. A abordagem utilizada se mostrou mais favorável com 11, 13 e 15 processadores, entretanto, obteve-se um fator aceleração relativamente baixo, próximo a 3 para 15 processadores. Apesar de o número de estados pertencente ao RSS ser relativamente pequeno, ao se aumentar o tamanho do modelo em questão o qual disponibilizaria maior grão para

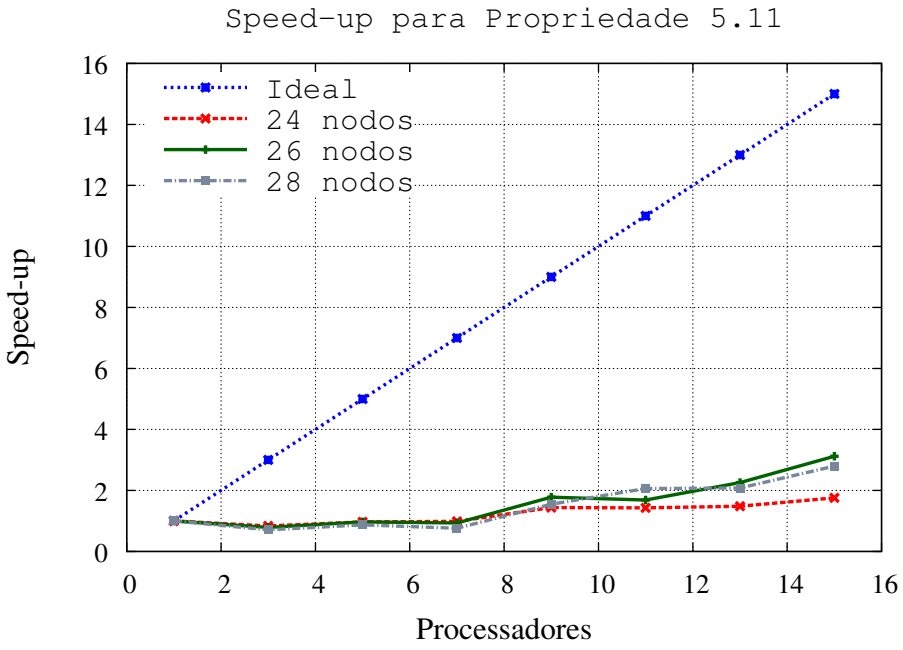


Figura 5.7 – Gráfico de *speed-up* para a Propriedade 5.11.

computação em cada iteração do algoritmo, o número de rodadas de sincronia apresentou tendências de crescimento, não permitindo ganhos consideráveis. As elevadas variações de tempo (desvio padrão elevado para as amostras) se dá devido à necessidade de constante acesso à disco. Conforme descrito na Seção 5.4, sendo não exclusivo o disco utilizado para troca de informações, a execução de muitas operações de leitura e escrita pode sofrer interferências causadas por comportamentos do ambiente, sendo paralelos ao algoritmo.

A Tabela 5.7 apresenta os resultados obtidos através do experimento para a Propriedade $\forall\Box(Nodo_i Transmite \rightarrow IntervaloTransmissão)$ (5.13). Os ganhos se mostraram mais favoráveis para 9, 11 e 13 processadores. Este comportamento é justificado uma vez que foram identificadas apenas duas rodadas de troca de mensagens entre processos mestre e escravos durante a verificação do operador $\exists(\Phi \cup \Psi)$, sendo uma para início e outra para término da computação. Isto indica que o conjunto de estados que provém da diferença entre Φ e Ψ a ser avaliado já representa um ponto fixo. Desta maneira, apenas uma iteração é realizada pelo algoritmo de mananeira distribuída para o operador em questão, não necessitando sincronismo entre os processos envolvidos e constante acesso a disco para troca de estados.

Tabela 5.7 – Resultados para a Propriedade 5.13 para o modelo *Ad Hoc Wireless Networks*.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
24 nodos ($\|S\| = 82.946$)						
1	10,77	0,07	$10{,}69 < \mu < 10{,}85$	1,00	202,03	-
3	5,61	0,04	$5{,}59 < \mu < 5{,}62$	1,92	135,53	69,63
5	3,83	0,50	$3{,}67 < \mu < 3{,}98$	2,81	85,25	63,57
7	3,32	0,34	$3{,}21 < \mu < 3{,}42$	3,25	65,51	52,37
9	3,13	0,42	$3{,}00 < \mu < 3{,}26$	3,44	52,30	44,60
11	3,08	0,35	$2{,}97 < \mu < 3{,}19$	3,50	46,40	34,57
13	2,59	0,29	$2{,}50 < \mu < 2{,}68$	4,16	40,68	34,64
15	2,15	0,14	$2{,}11 < \mu < 2{,}19$	5,01	35,56	30,48
26 nodos ($\|S\| = 217.154$)						
1	57,18	0,14	$57{,}03 < \mu < 57{,}32$	1,00	572,34	-
3	22,39	0,86	$22{,}12 < \mu < 22{,}66$	2,55	388,54	239,51
5	10,84	0,73	$10{,}61 < \mu < 11{,}06$	5,27	243,66	191,20
7	8,04	0,53	$7{,}87 < \mu < 8{,}20$	7,11	183,79	154,76
9	7,14	0,96	$6{,}84 < \mu < 7{,}44$	8,00	149,94	129,71
11	6,64	0,81	$6{,}39 < \mu < 6{,}89$	8,61	132,77	112,80
13	6,22	0,69	$6{,}00 < \mu < 6{,}43$	9,19	115,87	99,56
15	6,18	0,25	$6{,}10 < \mu < 6{,}26$	9,25	101,72	87,74
28 nodos ($\|S\| = 568.518$)						
1	392,56	2,25	$390{,}30 < \mu < 394{,}81$	1,00	1.620,53	-
3	136,83	7,80	$134{,}42 < \mu < 139{,}25$	2,87	1.105,55	678,97
5	52,68	3,33	$51{,}65 < \mu < 53{,}71$	7,46	691,63	542,64
7	34,18	3,51	$33{,}09 < \mu < 35{,}26$	11,49	531,41	439,76
9	24,61	3,76	$23{,}44 < \mu < 25{,}77$	15,96	426,96	368,67
11	20,46	2,75	$19{,}61 < \mu < 21{,}32$	19,20	377,21	320,79
13	17,82	3,10	$16{,}86 < \mu < 18{,}78$	22,04	328,21	283,15
15	16,94	3,06	$15{,}99 < \mu < 17{,}88$	23,20	289,08	249,74

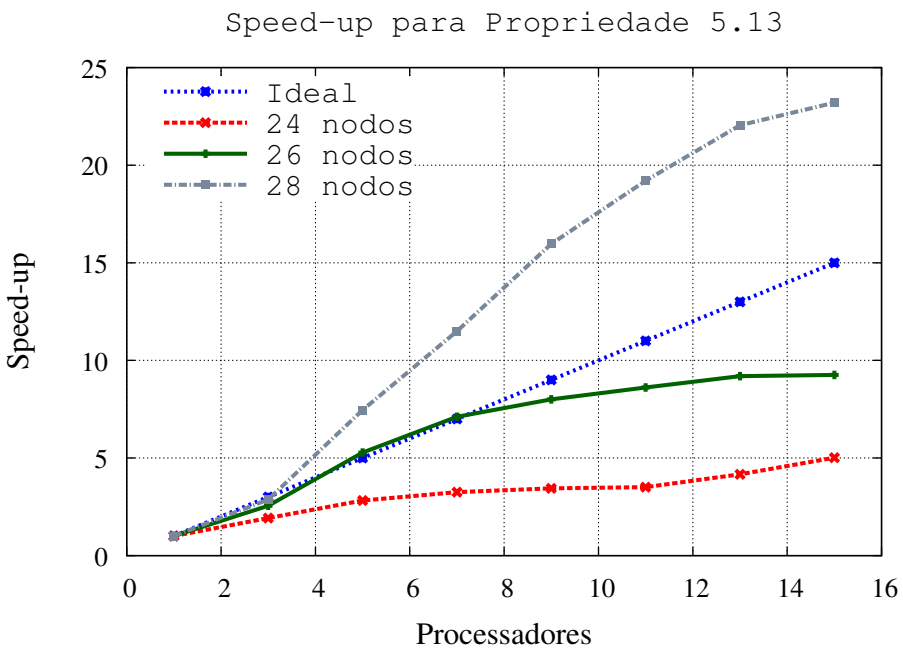


Figura 5.8 – Gráfico de *speed-up* para a Propriedade 5.13.

A Tabela 5.8 apresenta os resultados relativos aos experimentos realizados para a Propriedade 5.12 ($\forall(\neg Nodo_N Recebe \cup Nodo_1 Transmite)$). Os testes foram conduzidos através das duas abordagens paralelas propostas neste trabalho. Para esta propriedade foram identificadas as execuções de um número alto de iterações para obtenção de ponto fixo para os operadores $\exists\Box$ e $\exists\cup$ para ambos os modelos com 20 e 22 nodos, respectivamente. Embora o número de estados a ser

Tabela 5.8 – Resultados para a Propriedade 5.12 para o modelo *Ad Hoc Wireless Networks*.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
Sem Particionamento da Árvore Sintática						
20 nodos ($\|S\| = 12.102$)						
1	24,87	2,48	$22{,}38 < \mu < 27{,}36$	1,00	283,02	-
3	60,56	1,13	$60{,}21 < \mu < 60{,}91$	0,41	417,91	273,02
5	64,74	0,91	$64{,}46 < \mu < 65{,}03$	0,38	251,97	189,93
7	49,46	0,54	$49{,}29 < \mu < 49{,}63$	0,50	172,49	168,21
9	41,03	0,57	$40{,}85 < \mu < 41{,}21$	0,60	177,15	120,92
11	36,40	0,66	$36{,}20 < \mu < 36{,}61$	0,68	132,40	110,78
13	32,45	0,75	$32{,}22 < \mu < 32{,}69$	0,76	129,29	103,32
15	27,53	0,91	$27{,}25 < \mu < 27{,}82$	0,90	103,52	81,79
22 nodos ($\|S\| = 31.682$)						
1	126,99	1,38	$125{,}61 < \mu < 128{,}37$	1,00	895,04	-
3	432,37	8,93	$429{,}60 < \mu < 435{,}13$	0,29	1.429,08	952,68
5	391,63	18,70	$385{,}83 < \mu < 397{,}42$	0,32	1.106,92	744,30
7	272,00	5,24	$270{,}37 < \mu < 273{,}62$	0,46	902,05	675,79
9	211,61	3,47	$210{,}54 < \mu < 212{,}69$	0,60	741,87	544,29
11	162,39	1,90	$161{,}80 < \mu < 162{,}98$	0,78	541,79	407,04
13	143,23	1,69	$142{,}70 < \mu < 143{,}75$	0,88	482,74	401,29
15	107,70	2,07	$107{,}06 < \mu < 108{,}35$	1,17	415,56	317,13
Com Particionamento da Árvore Sintática						
20 nodos ($\|S\| = 12.102$)						
1	24,87	2,48	$22{,}38 < \mu < 27{,}36$	1,00	283,02	-
3	23,35	0,66	$23{,}15 < \mu < 23{,}56$	1,07	267,30	100,99
5	55,36	5,59	$53{,}63 < \mu < 57{,}10$	0,44	397,31	168,37
7	66,82	1,32	$66{,}41 < \mu < 67{,}23$	0,37	302,67	130,15
9	57,80	1,83	$57{,}23 < \mu < 58{,}37$	0,43	254,74	108,81
11	49,12	7,51	$46{,}80 < \mu < 51{,}45$	0,50	211,26	98,09
13	45,37	6,89	$43{,}23 < \mu < 47{,}51$	0,54	235,92	93,06
15	37,70	1,81	$37{,}14 < \mu < 38{,}26$	0,66	189,72	85,55
22 nodos ($\|S\| = 31.682$)						
1	126,99	1,38	$125{,}61 < \mu < 128{,}37$	1,00	895,04	-
3	118,38	5,48	$116{,}68 < \mu < 120{,}08$	1,07	853,60	317,41
5	337,18	10,98	$333{,}77 < \mu < 340{,}58$	0,37	1.420,93	585,10
7	455,93	15,98	$450{,}98 < \mu < 460{,}89$	0,27	1.312,60	545,68
9	366,01	14,47	$361{,}52 < \mu < 370{,}49$	0,34	1.088,45	423,16
11	281,36	13,22	$277{,}26 < \mu < 285{,}46$	0,45	849,30	364,86
13	246,23	16,62	$241{,}08 < \mu < 251{,}38$	0,51	882,18	362,73
15	204,50	19,54	$198{,}45 < \mu < 210{,}56$	0,62	754,70	333,66

verificado é pequeno, o grande número de iterações realizada pelo algoritmo não permitiu ganhos expressivos de desempenho, devido à contante comunicação.

Apesar de o aumento do tamanho do modelo fornecer maiores conjuntos de estados a serem avaliados, isto implica no aumento do número de iterações de avaliação necessárias para obtenção de ponto fixo. Este comportamento foi identificado para propriedades em que as proposições atômicas definidas fazem referência à comportamentos de autômatos que não possuam relação de dependência direta e, também, autômatos que possuem baixa influência sobre o modelo. Este comportamento também pode ser visualizado para a propriedade representada pela Equação 5.11 (Tabela 5.6).

Entretanto, quando do emprego de apenas um processo na computação de cada partição da árvore sintática pela segunda abordagem paralela (resultados apresentados nas linhas correspondentes a três processos) pequenos ganhos de desempenho são obtidos. Este comportamento é justificado devido a baixa comunicação necessária durante a computação uma vez que não são executadas rodadas de sincronia entre as máquinas empregadas.

O gráfico apresentado pela Figura 5.9 demonstra o pico de consumo de memória para as propriedades verificadas sobre o modelo *Ad Hoc Wireless Networks*. O consumo de memória para as

propriedades 5.11 e 5.12 tende a aumentar consideravelmente em relação à implementação sequencial quando do emprego de um número baixo de processadores. Devido à grande quantidade de operações de sincronia e iterações executadas pelo algoritmo, o crescimento das estruturas intermediárias necessárias à obtenção do resultado na estrutura MDD acaba sendo elevado.

No caso da Propriedade 5.12, para ambas as implementações quedas semelhantes são obtidas. A abordagem de distribuição de partições da árvore sintática apresenta maior consumo devido ao emprego de grupos menores de processos para verificação de cada partição, consequentemente distribuindo grupos maiores de estados para verificação para cada máquina empregada. Entretanto ao se considerar as execuções com três processos, quedas significativas são obtidas devido à concentração da computação uma vez que se emprega apenas um processo na verificação de cada partição da árvore sintática, o que possivelmente permite maior reuso de estruturas no MDD presente em cada processo.

Por fim, o consumo de memória para a Propriedade 5.13 apresenta comportamento semelhante ao obtido para as propriedades verificadas sobre o modelo do Jantar dos Filósofos, tendo este último consumo detalhado pela Figura 5.6, devido à execução de um número baixo de iterações e comunicação para obtenção de ponto fixo.

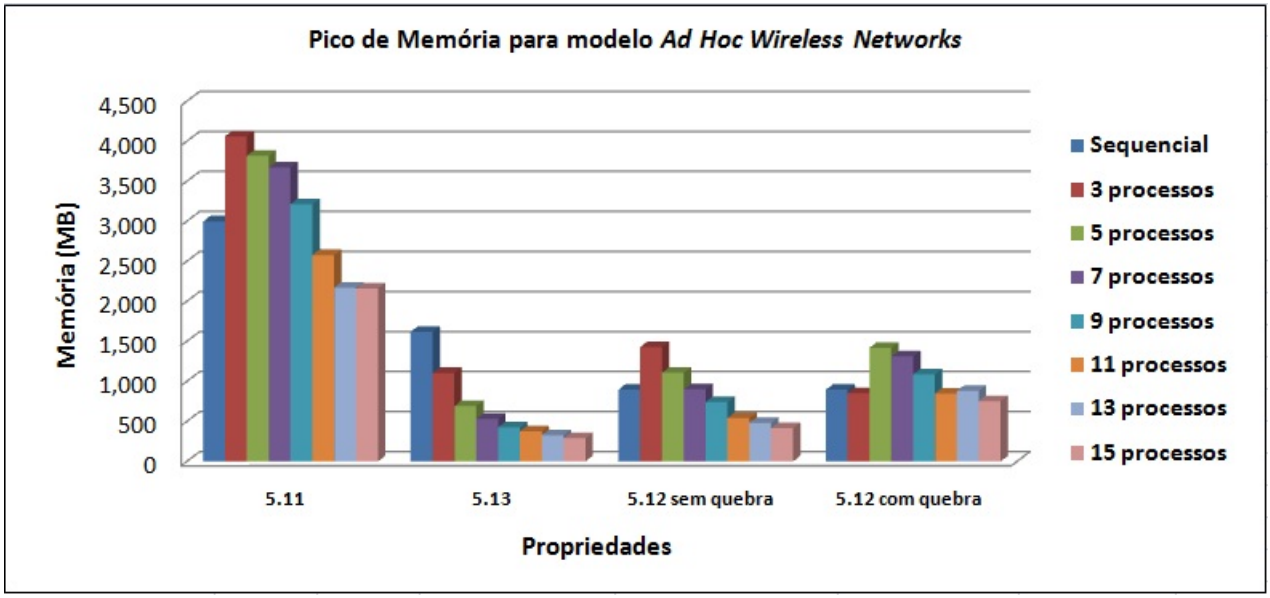


Figura 5.9 – Gráfico de pico de memória para as Propriedades 5.11 para 24 nodos, 5.12 com e sem particionamento da árvore sintática para o modelo com 22 nodos, e 5.13 para o modelo com 28 nodos.

5.6.3 Resultados para o modelo *Production Line*

Nesta seção são apresentados e discutidos os resultados obtidos para as propriedades definidas para o modelo *Production Line* (Seção 5.5.3).

Tabela 5.9 – Resultados para a Propriedade 5.14 para o modelo Linha de Produção.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
8 estações ($\|S\| = 10.864$)						
1	9,36	0,14	$9{,}21 < \mu < 9{,}50$	1,00	27,25	-
3	33,97	0,28	$33{,}89 < \mu < 34{,}06$	0,27	40,25	26,18
5	18,44	0,16	$18{,}39 < \mu < 18{,}49$	0,50	21,88	16,40
7	13,88	0,16	$13{,}83 < \mu < 13{,}93$	0,67	17,82	14,04
9	10,88	0,21	$10{,}81 < \mu < 10{,}94$	0,86	12,46	10,05
11	9,24	0,21	$9{,}17 < \mu < 9{,}31$	1,01	9,94	8,23
13	7,46	0,29	$7{,}36 < \mu < 7{,}55$	1,25	8,33	7,11
15	7,34	0,47	$7{,}19 < \mu < 7{,}49$	1,27	7,78	6,53
9 estações ($\|S\| = 40.545$)						
1	134,25	1,86	$132{,}38 < \mu < 136{,}12$	1,00	126,36	-
3	624,80	6,98	$622{,}63 < \mu < 626{,}96$	0,21	230,61	153,12
5	356,66	5,42	$354{,}98 < \mu < 358{,}34$	0,37	136,18	101,42
7	230,46	4,50	$229{,}06 < \mu < 231{,}85$	0,58	96,29	78,34
9	196,65	3,45	$195{,}58 < \mu < 197{,}73$	0,68	76,17	63,35
11	140,75	5,91	$138{,}91 < \mu < 142{,}58$	0,95	58,72	49,11
13	120,70	5,72	$118{,}93 < \mu < 122{,}48$	1,11	54,05	44,82
15	113,77	8,78	$111{,}05 < \mu < 116{,}49$	1,17	48,82	41,76

A Tabela 5.9 apresenta os resultados obtidos com a implementação descrita na Seção 4.1 para a propriedade representada pela Equação 5.14. Foram identificadas as execuções de um número considerável de iterações para verificação dos operadores que dependem de cálculo de ponto fixo para a propriedade em questão. A abordagem se mostrou mais favorável para 11, 13 e 15 processos, embora apresentando ganhos pouco significativos.

Embora seja esperado que o aumento do número de processadores para verificação permita obter maior desempenho, o pequeno número de estados atingíveis dos modelos aliado à grande quantidade de comunicação e acesso à disco necessários não permite ganhos expressivos. Experimentos foram conduzidos para o modelo com 10 nodos, entretanto ganhos expressivos não foram obtidos. O emprego de rotinas assíncronas de comunicação e otimizações relacionadas ao tráfego de estados habilita os processos escravos à evitar bloqueios, de forma que ganhos maiores podem ser obtidos mesmo quando se verificam conjuntos pequenos de estados.

A Tabela 5.10 apresenta os resultados obtidos para ambas as abordagens de verificação para a Propriedade 5.15 para 8 e 9 estações, tendo uma fórmula ENF semelhante apresentada na Figura 4.3. Esta propriedade testa a vivacidade do modelo em sua totalidade através da conjunção da Propriedade 5.14 para todas as estações do modelo.

Os ganhos obtidos não apresentam grande expressividade para ambas as abordagens devido à grande quantidade de operadores que necessitam de cálculo de ponto fixo aliado à grande quantidade de sincronia para troca de dados necessária para cada um destes operadores. Entretanto, ao se distribuir partições da árvore sintática empregando-se apenas um processo escravo na computação de cada partição (resultados representados pelas linhas que dizem respeito à três processos) consideráveis ganhos de desempenho são obtidos. Este comportamento ocorre devido a grande quantidade de partições geradas da árvore sintática e devido a baixa troca de mensagens causada pela inexistência de execução de rodadas de sincronia, de forma que o algoritmo se mostre mais eficiente em relação a primeira abordagem. Variações consideráveis de tempo podem ser visualizadas

Tabela 5.10 – Resultados para a Propriedade 5.15 para o modelo Linha de Produção.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
Sem Particionamento da Árvore Sintática						
8 estações ($\|S\| = 10.864$)						
1	37,23	0,71	$36{,}52 < \mu < 37{,}94$	1,00	80,18	-
3	123,53	13,59	$119{,}32 < \mu < 127{,}75$	0,30	138,18	90,85
5	87,92	39,24	$75{,}76 < \mu < 100{,}09$	0,42	75,78	58,52
7	67,73	34,34	$57{,}09 < \mu < 78{,}37$	0,55	59,95	44,84
9	56,75	19,47	$50{,}71 < \mu < 62{,}78$	0,65	43,85	34,37
11	51,82	17,05	$46{,}54 < \mu < 57{,}11$	0,71	33,80	27,67
13	51,44	17,95	$45{,}87 < \mu < 57{,}00$	0,72	29,15	24,60
15	53,68	32,54	$43{,}59 < \mu < 63{,}76$	0,69	28,49	21,37
9 estações ($\|S\| = 40.545$)						
1	606,97	6,54	$600{,}42 < \mu < 613{,}52$	1,00	419,73	-
3	$2{,}34 \times 10^3$	23,23	$2{,}34 \times 10^3 < \mu < 2{,}35 \times 10^3$	0,25	966,77	619,72
5	$1{,}33 \times 10^3$	14,39	$1{,}33 \times 10^3 < \mu < 1{,}34 \times 10^3$	0,45	548,99	401,59
7	956,50	17,67	$951{,}02 < \mu < 961{,}97$	0,63	412,41	308,82
9	745,06	49,91	$729{,}59 < \mu < 760{,}53$	0,81	293,12	237,52
11	632,98	393,21	$511{,}12 < \mu < 754{,}84$	0,95	247,52	192,85
13	484,19	44,35	$470{,}45 < \mu < 497{,}94$	1,25	214,92	165,26
15	414,59	57,79	$396{,}68 < \mu < 432{,}50$	1,46	183,58	148,86
Com Particionamento da Árvore Sintática						
8 estações ($\|S\| = 10.864$)						
1	37,23	0,71	$36{,}52 < \mu < 37{,}94$	1,00	80,18	-
3	26,30	1,11	$25{,}96 < \mu < 26{,}65$	1,42	47,69	33,22
5	77,04	19,68	$70{,}94 < \mu < 83{,}14$	0,48	87,18	57,92
7	69,42	23,45	$45{,}70 < \mu < 93{,}15$	0,56	64,82	46,35
9	65,66	25,66	$57{,}70 < \mu < 73{,}61$	0,63	55,68	34,78
11	61,85	49,09	$46{,}63 < \mu < 77{,}06$	0,53	43,75	29,61
13	58,40	45,01	$44{,}45 < \mu < 72{,}35$	0,60	41,07	25,99
15	36,79	11,62	$33{,}19 < \mu < 40{,}39$	1,01	32,91	21,47
9 estações ($\|S\| = 40.545$)						
1	606,97	6,54	$600{,}42 < \mu < 613{,}52$	1,00	419,73	-
3	361,74	7,52	$359{,}41 < \mu < 364{,}07$	1,68	255,02	170,00
5	$1{,}37 \times 10^3$	62,684	$1{,}35 \times 10^3 < \mu < 1{,}39 \times 10^3$	0,44	568,87	389,26
7	942,30	76,38	$918{,}63 < \mu < 965{,}97$	0,64	467,98	314,70
9	834,10	154,97	$786{,}08 < \mu < 882{,}13$	0,72	328,01	244,71
11	736,56	126,90	$697{,}24 < \mu < 775{,}89$	0,82	308,23	207,95
13	590,52	120,29	$553{,}24 < \mu < 627{,}80$	1,02	255,31	175,73
15	511,29	138,70	$468{,}30 < \mu < 554{,}27$	1,18	201,22	154,14

nas amostras colhidas para ambas as abordagens causadas pelo acesso a disco para troca estados durante sincronia.

A Figura 5.10 apresenta o consumo de memória (pico) para as Propriedades 5.14 e 5.15, ambas verificadas sobre o modelo *Linha de Produção* para 9 estações. É notório um aumento de consumo quando do emprego de poucos processos para verificação de ambas as propriedades, de forma que a execução de grande quantidade de rodadas de sincronia influi para este comportamento e, principalmente, devido à grande quantidade de iterações de avaliação dos conjunto de estados em consideração.

À medida que se aumenta o número de processos para verificação, para ambas as propriedades e abordagens utilizadas, a distribuição das tarefas relacionadas à computação dos operadores de lógica temporal permite que as estruturas simbólicas intermediárias necessárias à obtenção do resultado sejam melhor distribuídas, de maneira que o pico de consumo de memória em cada caso seja muito inferior ao valor atingido pela implementação sequencial.

Ao se empregar apenas um processo escravo na computação de cada partição da árvore sintática que representa a respectiva propriedade transcrita para ENF, consideráveis ganhos em

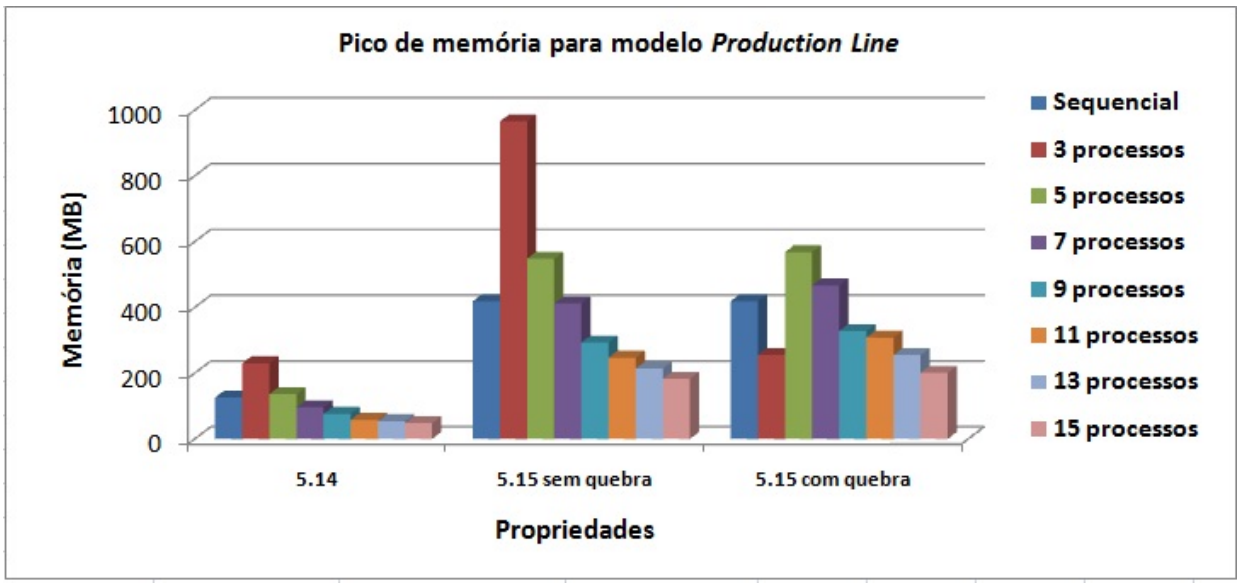


Figura 5.10 – Gráfico de pico de memória para as Propriedades 5.14 e 5.15, esta última verificada com e sem particionamento da árvore sintática. $N = 9$ estações.

questões de memória são obtidos. Este comportamento é justificado devido ao reuso de estruturas simbólicas pela estrutura MDD em cada processo empregado aliado a inexistência de execuções de rodadas de sincronia. Conforme anteriormente relatado, a execução de rodadas de sincronia necessitam de tráfego de estados entre as máquinas e execução de operações de união e intersecção, as quais necessitam da criação de estruturas intermediárias na estrutura MDD para obtenção do resultado, influenciando no aumento do consumo de memória.

5.7 Fatores Aceleração Elevados

Embora a ocorrência de fatores aceleração acima do ideal em ambientes distribuídos não seja frequente, comportamentos com estas características em abordagens paralelas de verificação são encontrados em alguns trabalhos.

Bell e Haverkort [8] apresentam algoritmos explícitos para geração do espaço de estados e verificação de propriedades CTL para modelos descritos em Redes de Petri [3]. Os resultados gerados apresentam fatores aceleração acima do esperado para os operadores $\exists\cup$ e $\exists\bigcirc$ para o problema do Jantar dos Filósofos. Como causa para este comportamento, os autores citam o uso de múltiplos processadores o que disponibiliza maiores espaços de *cache*, permitindo a obtenção de melhor performance e, também, a função de particionamento adotada a qual reduz a presença de arcos de transição entre as partições, consequentemente reduzindo a comunicação entre as máquinas.

Behrmann, Hune e Vaandrager [7] apresentam uma abordagem distribuída de verificação simbólica do formalismo *Timed Automata*. Cada nodo computacional recebe porções do espaço de estados após a sua geração de acordo com uma função de mapeamento e distribuição. Os

resultados obtidos mostram ganhos de desempenho próximos ao dobro do ideal para alguns casos, sendo justificados pelos autores devido ao acesso à grandes quantidades de memória *cache* local pelas máquinas empregadas. Este trabalho é baseado na implementação apresentada por Stern e Dill [48], onde utiliza-se uma tabela de estados para armazenar os estados atingíveis do modelo, sendo particionada entre os nodos empregados os quais possuem uma fila de trabalho para armazenamento dos estados ainda não-explorados.

Boukala e Petrucci [9] implementaram um algoritmo de geração de espaços de estados explícitos e verificação de propriedades CTL, ambos distribuídos, para o formalismo Redes de Petri. Fatores aceleração acima do esperado foram obtidos para o operador $\exists\cup$ para o modelo do Jantar dos Filósofos. Entretanto, os autores não argumentam sobre possíveis causas para este comportamento. Os resultados mostram números consideravelmente altos de troca de mensagens durante a geração do RSS, verificação e geração de contra-exemplos. Ambos os trabalhos acima descritos adotam o modelo de programação mestre/escravo e comunicação assíncrona para tráfego de estados entre as máquinas.

Para os autores Rauber e Runger [42], fatores aceleração superlineares ($S_p > p$) podem ser observados em alguns casos. A razão para este comportamento frequentemente encontra-se em efeitos de *cache*: um programa paralelo típico atribui somente uma fração do conjunto de dados para cada processador. A fração é selecionada de forma que o processador realize as suas computações no seu conjunto de dados. Nesta situação, pode ocorrer de o conjunto de dados como um todo ser maior que a área de *cache* disponível de um único processador executando o programa sequencialmente, assim levando à *cache misses* durante a computação.

Entretanto, quando muitos processadores executam um programa com a mesma quantidade de dados em paralelo, pode ocorrer de a fração dos dados atribuídos à cada processador ser inferior à área local de *cache* do mesmo, evitando-se *cache misses*, ou seja, não sendo necessário acesso à memória principal, a qual é mais lenta. Fatores aceleração superlineares não ocorrem com frequência, sendo uma situação típica que se alcance aceleração inferior ou próxima ao linear ($S_p \leq p$). Este comportamento é causado devido às operações adicionais necessárias para gerência do paralelismo e troca de dados entre processadores e, também, devido aos longos tempos de espera causados por balanceamento desigual de trabalho entre os processadores [42].

Conforme descrito na Seção 5.6, os experimentos que apresentaram ganhos acima do ideal são aqueles em que poucas iterações são necessárias para obtenção de ponto fixo, implicando na execução de poucas rodadas de comunicação, habilitando os processadores a seguir a verificação de estados de maneira quase independente. Aliado à esses fatores, cada máquina empregada na verificação está habilitada a calcular estados sucessores de um dado estado localmente, devido à replicação da estrutura MDD que representa o RSS rotulado para as proposições atômicas definidas, evitando-se troca de mensagens. Apesar da obtenção destes comportamentos estar ligada à estes fatores somado à possíveis efeitos de *cache*, algumas características do ambiente adotado foram consideradas para condução de experimentos visando avaliar o comportamento do algoritmo sob diferentes circunstâncias.

Devido à presença de otimizações para troca de mensagens para uso em máquinas *multi-core* pelo ambiente MPI adotado, processos que estejam dispostos em um mesmo nodo computacional terão benefícios na transferência de dados. Através do uso da tecnologia *Shared-Memory Byte Transfer Layer* (SMBTL) [39], a qual fornece um mecanismo de troca de mensagens de baixa latência e alta largura de banda, mensagens MPI em ambientes com estas características podem ser transferidas através de espaços de memória compartilhada entre os núcleos de processamento de um mesmo nodo. Visando melhor avaliar os resultados que apresentam fatores aceleração superlineares, as seguintes abordagens para execução dos experimentos foram consideradas:

- **Experimento 1:** Inicialmente, testes foram conduzidos utilizando-se apenas um nodo da máquina atlantica, de modo que não sejam trafegadas mensagens através da rede da máquina. Neste caso, foram obtidos consideráveis ganhos de desempenho quando do emprego de até 8 processos, o que de fato caracteriza o não uso da tecnologia *Hyper-Threading*, uma vez que se instanciam quantidades de processos iguais ao número de núcleos físicos disponíveis no nodo. Entretanto, não há controle sobre a alocação interna de processos nos núcleos físicos dos processadores, sendo esta tarefa delegada aos algoritmos de escalonamento do sistema operacional da máquina. À medida que se aumenta o número de processos, considerável redução do ganho de desempenho pôde ser visualizada. Não serão tabelados os resultados obtidos através do Experimento 1 uma vez que não há troca de mensagens pela rede da máquina.

- **Experimento 2:** Em uma tentativa de medir o impacto causado pelo acesso à rede, testes foram conduzidos em uma segunda máquina. A máquina adotada (*cluster* gates) é composta por 16 máquinas *Rackable Systems*. Cada nodo possui 2 processadores *AMD Opteron 246 de 2 GHz* e 8 GB de memória e estão interligados por redes *Gigabit-Ethernet* chaveadas.

  Ao conduzir os experimentos relativos às implementações sequencial e paralelas, posterior cálculo de *speed-up* demonstrou comportamentos semelhantes ao Experimento 1. Entretanto, à medida que se aumenta o número de processadores, o ganho de desempenho se mantém crescente, de modo que comunicação entre os nodos não afete o desempenho da aplicação para tais casos. Este comportamento reforça as razões dadas para os ganhos demonstrados na Seção 5.6. O Apêndice B apresenta os resultados de tempo obtidos para o Experimento 2.

- **Experimento 3:** Por fim, utilizando-se a máquina atlantica, experimentos foram conduzidos distribuindo-se processos MPI entre dois nodos da máquina. Os resultados apresentados na Seção 5.6 dizem respeito a este experimento, detalhado anteriormente na Seção 5.4. Para os três experimentos relatados, estimativas intervalares foram geradas para amostras de 40 execuções.

Realizando-se um comparativo entre os resultados colhidos através dos Experimentos 2 e 3, é perceptível que ambas as abordagens de execução apresentam comportamentos similares no que diz respeito ao cálculo de fatores aceleração. Este comportamento indica e reforça os argumentos

para tais ganhos, de forma que devido à baixa quantidade de comunicação necessária para cálculo de ponto fixo, latência de rede não é fator impactante para tais casos.

Conforme relatado na Seção 4, os dados inicialmente presentes na estrutura MDD, os quais dizem respeito ao espaço de estados atingível rotulado para as proposições atômicas definidas, apresentam consumo de memória extremamente baixo. Embora este seja um requisito para possível ocorrência de efeitos de *cache* em processadores, o crescimento da estrutura MDD durante a verificação de propriedades tende a ser extremamente elevado em ambas as máquinas empregadas, conforme detalhado na Seção 5.6. Apesar de ganhos de desempenho relacionados à ocorrência de efeitos de *cache* ser uma interferência positiva, sua ocorrência é de difícil detecção.

De outro lado, possíveis efeitos relacionados à estrutura MDD adotada podem afetar o desempenho do algoritmo sequencial. Tais efeitos podem ser provenientes das implicações causadas pelo aumento do número de informações intermediárias criadas na estrutura MDD para obtenção do resultado da verificação. Embora não se tenha feito estudos acerca de possíveis ocorrências destes efeitos na estrutura MDD em questão, alguns autores detectaram comportamentos de tal natureza.

Um exemplo deste comportamento é demonstrado por Joubert and Mateescu [30] em um algoritmo de verificação simbólico distribuído para μ-calculus. Os autores citam como causa da ocorrência de *speed-ups* superlineares o uso de *hash tables* para armazenamento de conjuntos de variáveis booleanas. Segundo os autores, sendo o balanceamento destas tabelas não-perfeito, colisões tendem a ocorrer, sendo um fenômeno frequente para a versão sequencial da ferramenta desenvolvida, a qual usa uma única *hash table* para armazenamento, ao contrário da implementação paralela, a qual endereça p tabelas menores, de modo que o balanceamento de variáveis em cada nodo empregado não apresente tal impacto.

Considerando este cenário e respectivos resultados elevados obtidos, uma abordagem que permita avaliar uma possível inteferência conforme acima relatada se fez necessária. Para tal feito, algoritmos que visam a coleção de estruturas desnecessárias presentes na estrutura MDD foram empregados para ambas as abordagens, paralelas e sequencial. Desta maneira, tais algoritmos realizam a liberação da memória alocada não mais necessária para a computação e posterior reordenação das *hash tables* utilizadas para armazenamento de variáveis da estrutura MDD. Considerável redução do pico de consumo de memória e tempo de verificação são esperados.

Para exemplificação do impacto causado pela adoção de tais algoritmos, experimentos foram conduzidos para as Propriedades 5.6, 5.7 e 5.13, tendo os resultados apresentados nas Tabelas 5.11, 5.12 e 5.13, respectivamente. Comparando-se estes resultados com os resultados obtidos através das implementações inicialmente desenvolvidas (Tabelas 5.1, 5.2 e 5.7) é perceptível que a adoção dos algoritmos acima descritos interfere consideravelmente tanto na redução de tempo como consumo de memória para a implementação sequencial. Este comportamento reforça a hipótese inicial de que o crescimento de estruturas intermediárias na estrutura MDD tende a interferir no desempenho da execução de operações necessárias a verificação de propriedades CTL.

De outro lado, analisando-se os tempos obtidos com a implementação paralela, um pequeno aumento no tempo de verificação é visualizado. De fato, tal comportamento é causado pela execução

das rotinas de liberação de memória anteriormente descritas, as quais apresentam uma pequena interferência durante a verificação.

Embora os resultados apresentem um pequeno aumento de tempo em relação as implementações inicialmente desenvolvidas, os ganhos de desempenho obtidos continuam sendo interessantes. Entretanto, não mais se atingem *speed-ups* superlineares, podendo estes comportamentos ser visualizados nos gráficos de *speed-up* apresentados para as propriedades em questão (Figuras 5.11, 5.12 e 5.13). Apesar da queda de tempo de verificação sequencial o qual foi causado pela adoção dos algoritmos acima descritos, é interessante destacar que os ganhos de desempenho obtidos ainda são consideráveis, o que demonstra que os algoritmos paralelos desenvolvidos são eficientes. Através da execução de tais experimentos, uma avaliação concreta dos ganhos de desempenho pôde ser obtida.

Embora ganhos em questão de tempo sejam de extrema importância, redução do consumo de memória também é esperado. Conforme anteriormente descrito, a adoção dos algoritmos de coleção acima descritos permitem que sejam liberados espaços de memória alocados pela estrutura MDD os quais não sejam necessários em um dado momento da computação. A Figura 5.14 apresenta um gráfico com o pico de consumo de memória e consumo médio entre os processadores empregados para as propriedades em consideração.

Tomando como base os resultados apresentados na Seção 5.6, é percebido considerável redução do consumo de memória pela aplicação sequencial. De outro lado, considerando-se os resultados obtidos com a implementação paralela, é notável que houve pequena redução no consumo de memória se comparado aos experimentos inicialmente conduzidos, demonstrando que os algoritmos de coleção pouco influenciam a implementação paralela. Este comportamento pode ser explicado levando em consideração a distribuição da computação entre as máquinas, de maneira que haja distribuição das estruturas simbólicas intermediárias necessárias à obtenção do resultado da verificação e, desta maneira, a execução do algoritmo de liberação de tais espaços de memória tende a apresentar pouca influência.

Tabela 5.11 – Resultados obtidos para a Propriedade 5.6 (*Starvation*) com uso de algoritmos de coleção.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	22,29	0,07	$22{,}27 < \mu < 22{,}31$	1,00	119,45	-
3	19,92	2,04	$19{,}29 < \mu < 20{,}55$	1,12	157,85	98,60
5	9,89	0,76	$9{,}66 < \mu < 10{,}13$	2,25	106,14	73,01
7	7,92	0,60	$7{,}74 < \mu < 8{,}11$	2,81	79,88	60,84
9	7,55	0,85	$7{,}29 < \mu < 7{,}82$	2,95	65,43	50,17
11	8,15	1,08	$7{,}81 < \mu < 8{,}49$	2,73	55,28	43,94
13	8,33	1,17	$7{,}97 < \mu < 8{,}69$	2,68	47,81	38,26
15	8,79	1,51	$8{,}32 < \mu < 9{,}26$	2,54	43,30	35,16
14 Filósofos ($\|S\| = 195.025$)						
1	83,99	0,45	$83{,}85 < \mu < 84{,}13$	1,00	160,72	-
3	97,81	10,20	$94{,}65 < \mu < 100{,}97$	0,86	408,11	255,99
5	37,04	2,36	$36{,}31 < \mu < 37{,}78$	2,27	273,49	185,64
7	23,67	2,07	$23{,}03 < \mu < 24{,}31$	3,55	198,41	157,94
9	19,87	1,43	$19{,}42 < \mu < 20{,}31$	4,23	161,52	132,24
11	18,81	2,10	$18{,}15 < \mu < 19{,}46$	4,47	142,40	114,29
13	18,34	3,00	$17{,}41 < \mu < 19{,}27$	4,58	121,92	101,92
15	17,69	3,09	$16{,}73 < \mu < 18{,}65$	4,75	111,39	89,99
15 Filósofos ($\|S\| = 470.832$)						
1	418,41	10,18	$415{,}25 < \mu < 421{,}56$	1,00	970,35	-
3	574,82	43,32	$561{,}40 < \mu < 588{,}25$	0,73	1.050,49	661,34
5	201,82	17,51	$196{,}39 < \mu < 207{,}24$	2,07	701,91	487,64
7	111,96	10,75	$108{,}63 < \mu < 115{,}29$	3,74	510,10	407,98
9	84,06	7,21	$81{,}83 < \mu < 86{,}30$	4,98	431,57	331,93
11	74,63	11,13	$71{,}19 < \mu < 78{,}08$	5,61	365,36	297,99
13	60,13	5,22	$58{,}51 < \mu < 61{,}75$	6,96	313,14	263,12
15	52,95	6,47	$50{,}94 < \mu < 54{,}95$	7,90	280,30	232,32

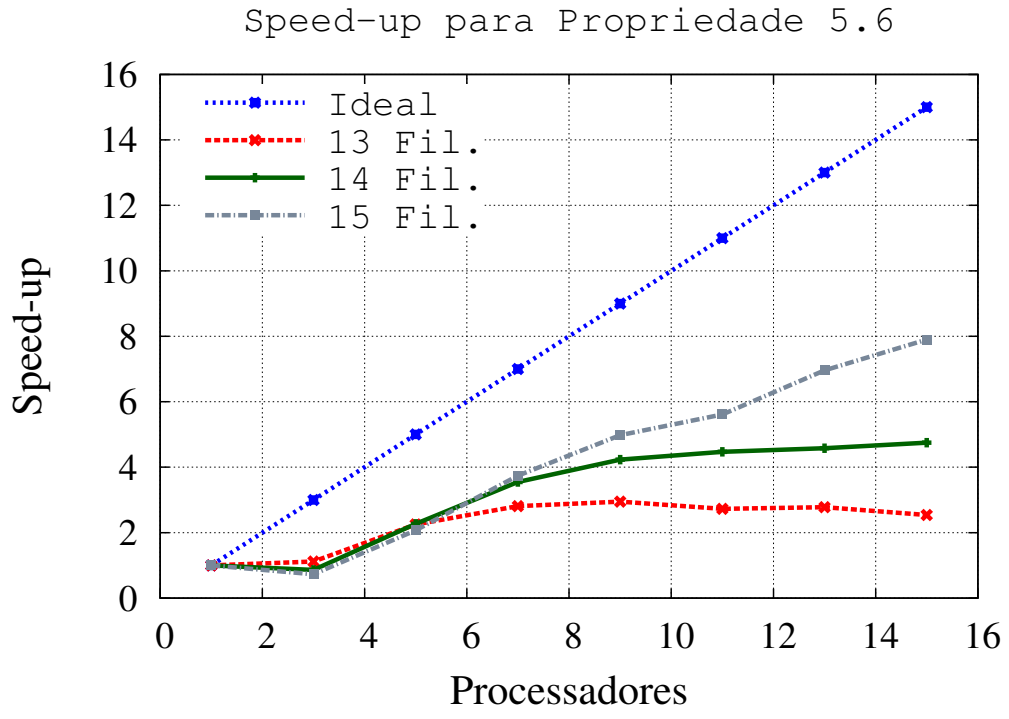


Figura 5.11 – *Speed-ups* obtidos para a Propriedade 5.6 com o uso de algoritmos de coleção.

Tabela 5.12 – Resultados obtidos para a Propriedade 5.7 (*Exclusão Mútua*) com uso de algoritmos de coleção.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
13 Filósofos ($\|S\| = 80.782$)						
1	13,49	0,04	$13{,}48 < \mu < 13{,}51$	1,00	71,50	-
3	9,54	0,44	$9{,}40 < \mu < 9{,}67$	1,41	178,58	113,64
5	5,73	0,49	$5{,}58 < \mu < 5{,}89$	2,35	120,73	85,59
7	4,45	0,45	$4{,}31 < \mu < 4{,}59$	3,03	91,57	67,85
9	3,84	0,30	$3{,}75 < \mu < 3{,}93$	3,51	70,01	57,75
11	3,74	0,31	$3{,}64 < \mu < 3{,}83$	3,61	62,90	50,09
13	3,66	0,34	$3{,}55 < \mu < 3{,}77$	3,69	54,32	43,91
15	3,55	0,21	$3{,}49 < \mu < 3{,}62$	3,80	49,01	39,23
14 Filósofos ($\|S\| = 195.025$)						
1	45,48	0,12	$45{,}44 < \mu < 45{,}52$	1,00	189,19	-
3	44,74	0,24	$44{,}67 < \mu < 44{,}82$	1,02	467,44	298,52
5	20,34	1,46	$19{,}88 < \mu < 20{,}79$	2,24	314,92	224,88
7	13,11	0,84	$12{,}85 < \mu < 13{,}37$	3,47	238,29	178,25
9	9,80	0,67	$9{,}59 < \mu < 10{,}01$	4,64	194,70	151,79
11	8,07	0,80	$7{,}82 < \mu < 8{,}32$	5,64	154,02	131,70
13	7,45	0,84	$7{,}19 < \mu < 7{,}71$	6,11	141,49	115,46
15	6,76	0,51	$6{,}61 < \mu < 6{,}92$	6,72	127,45	103,14
15 Filósofos ($\|S\| = 470.832$)						
1	194,71	0,41	$194{,}58 < \mu < 194{,}83$	1,00	496,64	-
3	279,36	6,33	$277{,}40 < \mu < 281{,}32$	0,69	1.216,19	779,05
5	103,63	2,05	$102{,}99 < \mu < 104{,}26$	1,87	816,92	586,97
7	62,68	3,59	$61{,}57 < \mu < 63{,}79$	3,10	616,82	465,18
9	40,52	1,04	$40{,}19 < \mu < 40{,}84$	4,80	503,85	396,31
11	28,76	1,01	$28{,}45 < \mu < 29{,}08$	6,76	424,83	343,98
13	23,38	0,86	$23{,}11 < \mu < 23{,}65$	8,32	366,58	301,55
15	20,05	0,80	$19{,}80 < \mu < 20{,}30$	9,71	329,73	269,35

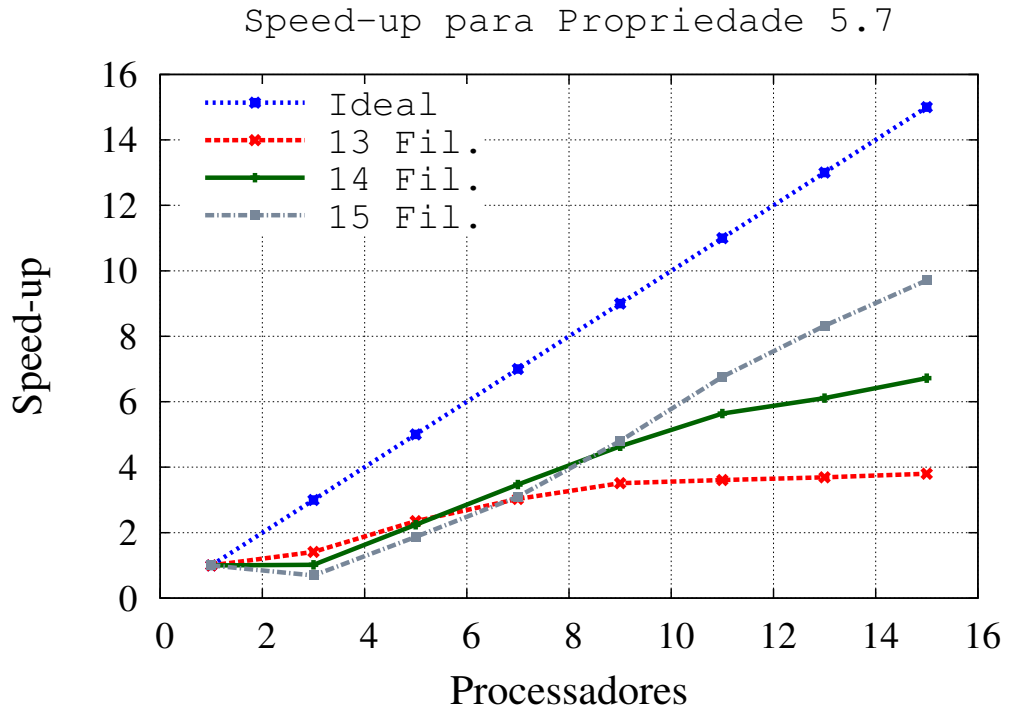


Figura 5.12 – *Speed-ups* obtidos para a Propriedade 5.7 com o uso de algoritmos de coleção.

Tabela 5.13 – Resultados obtidos para a Propriedade 5.13 para o modelo *Ad Hoc Wireless Networks* com uso de algoritmos de coleção.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*	Pico Memória (MB)	μMemória (MB)
24 nodos ($\|S\| = 82.946$)						
1	10,60	0,19	$10{,}41 < \mu < 10{,}80$	1,00	45,60	-
3	7,03	0,85	$6{,}76 < \mu < 7{,}29$	1,51	135,53	84,34
5	5,31	0,72	$5{,}09 < \mu < 5{,}53$	2,00	85,25	67,16
7	4,53	0,49	$4{,}38 < \mu < 4{,}68$	2,34	65,51	54,25
9	4,27	0,34	$4{,}16 < \mu < 4{,}37$	2,49	52,27	45,45
11	3,59	0,32	$3{,}49 < \mu < 3{,}69$	2,95	46,40	39,49
13	2,93	0,34	$2{,}83 < \mu < 3{,}04$	3,61	40,68	34,85
15	2,16	0,11	$2{,}12 < \mu < 2{,}19$	4,91	35,56	30,69
26 nodos ($\|S\| = 217.254$)						
1	33,12	0,08	$33{,}05 < \mu < 33{,}20$	1,00	88,61	-
3	24,01	1,94	$23{,}40 < \mu < 24{,}61$	1,38	388,54	239,52
5	13,55	1,81	$12{,}99 < \mu < 14{,}11$	2,44	243,67	191,21
7	10,18	1,05	$9{,}86 < \mu < 10{,}51$	3,25	187,21	154,76
9	8,95	1,63	$8{,}44 < \mu < 9{,}45$	3,70	149,94	129,71
11	8,02	1,54	$7{,}54 < \mu < 8{,}50$	4,13	132,77	112,80
13	7,81	1,24	$7{,}43 < \mu < 8{,}19$	4,24	115,87	99,57
15	7,53	1,02	$7{,}22 < \mu < 7{,}85$	4,40	101,73	87,74
28 nodos ($\|S\| = 568.518$)						
1	130,85	2,35	$128{,}50 < \mu < 133{,}20$	1,00	361,46	-
3	136,38	10,84	$133{,}03 < \mu < 139{,}74$	0,96	1.105,55	678,98
5	59,63	8,51	$56{,}99 < \mu < 62{,}27$	2,19	691,64	542,64
7	36,51	4,02	$35{,}26 < \mu < 37{,}76$	3,58	531,42	439,77
9	26,87	3,55	$25{,}77 < \mu < 27{,}97$	4,87	426,96	368,67
11	22,37	2,65	$21{,}54 < \mu < 23{,}19$	5,85	377,21	320,79
13	19,27	2,14	$18{,}61 < \mu < 19{,}94$	6,79	328,22	283,16
15	17,68	1,47	$17{,}23 < \mu < 18{,}14$	7,40	289,08	249,74

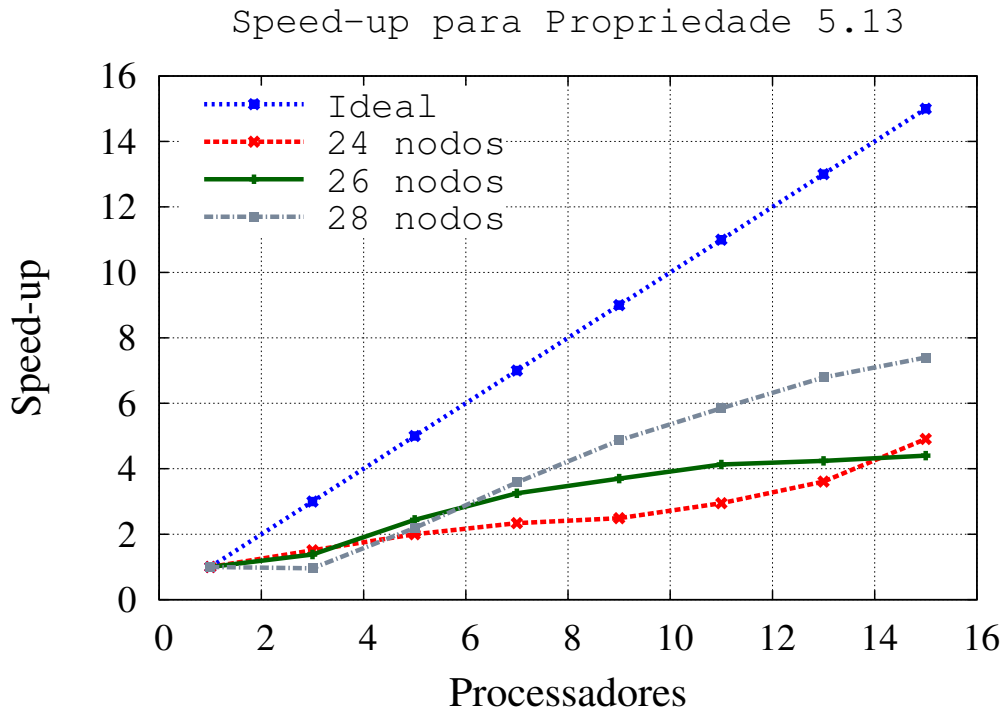


Figura 5.13 – *Speed-ups* obtidos para a Propriedade 5.13 com o uso de algoritmos de coleção.

Figura 5.14 – Gráfico de memória para as Propriedades 5.6 e 5.7 para o jantar dos filósofos com 15 filósofos e propriedade 5.13 para o modelo de uma cadeia de nodos *Ad Hoc Wireless* com 28 nodos.

6. CONSIDERAÇÕES FINAIS

Este trabalho integra um projeto de pesquisa que objetiva o desenvolvimento de um verificador de modelos para o formalismo Redes de Autômatos Estocásticos e propriedades descritas através da lógica temporal *Computation Tree Logic* (CTL). Estão também em andamento o desenvolvimento de algoritmos de geração de contra-exemplos e testemunhas, algoritmos otimizados de verificação e tradução de modelos SAN para a sintaxe aceita por outras ferramentas de verificação. O objetivo da pesquisa desenvolvida nesta dissertação é discutir, implementar e validar abordagens de verificação de propriedades para ambientes que usufruem de espaços de memória distribuídos.

Este trabalho apresentou duas propostas de algoritmos paralelos de verificação. O primeiro passo da pesquisa compreendeu o desenvolvimento de um algoritmo que distribui a computação relacionada à verificação dos operadores mais custosos da lógica temporal adotada. Após este passo, um algoritmo de análise e escolha de particionamento foi desenvolvido em virtude de permitir a verificação em paralelo de ramos independentes da árvore sintática que representa uma propriedade CTL escrita em Forma Normal Existencial. Aplicou-se então a primeira abordagem paralela para distribuição das tarefas relacionadas à verificação dos operadores mais complexos presentes em cada partição identificada. Para desenvolvimento das abordagens paralelas e posterior validação das mesmas, foi utilizada como base a implementação sequencial inicialmente desenvolvida [19] a qual implementa os algoritmos descritos em [2]. Uma metodologia para condução dos experimentos e coleta de dados foi utilizada em conjunto com diferentes abordagens de execução visando avaliar os ganhos de desempenho obtidos.

Mesmo que os experimentos conduzidos tenham mostrado que ambas as implementações paralelas são favoráveis à verificação de determinadas propriedades, os objetivos da pesquisa foram alcançados. O trabalho demonstrou que é possível reduzir o tempo de verificação de propriedades CTL através do algoritmo básico de satisfação descrito por Baier e Katoen [2], sem necessidade de emprego de técnicas complexas de particionamento de espaço de estados simbolicamente representados. Para ambas as abordagens, é viabilizada boa escalabilidade de máquinas permitindo a verificação de complexas realidades modeladas em SAN, formalismo que inicialmente não contava com ferramental de verificação.

Embora a replicação da função de transição e da estrutura simbólica que representa o espaço de estados referente à modelos SAN parecer pouco eficiente no que tange à consumo de memória, o trabalho demonstrou significativos ganhos em relação à implementação sequencial. A distribuição da computação relacionada à verificação de operadores da lógica temporal CTL permite que sejam distribuídas as estruturas simbólicas intermediárias relacionadas à obtenção do resultado da verificação, de forma que o consumo médio e pico de memória das máquinas empregadas seja consideravelmente inferior à implementação sequencial.

Apesar da redução do consumo em cada máquina ser um ponto positivo, o uso de ambientes que endereçam espaços de memória distribuídos aliado à replicação de estruturas simbólicas gera redundância, sendo este um fator limitante à adoção destes ambientes. Entretanto, esta técnica

habilita as máquinas empregadas a realizarem o cálculo de estados sucessores/antecessores de um dado estado localmente, permitindo exploração de conjuntos distintos de estados em cada máquina, de modo que comunicação é necessária somente para obtenção de ponto fixo.

Embora ganhos de desempenho relativamente baixos tenham sido obtidos para propriedades que exigem várias iterações sobre conjuntos de estados para obtenção de ponto fixo, o emprego de rotinas assíncronas de comunicação e otimizações na transferência de estados entre as máquinas empregadas tende a amenizar a sobrecarga de comunicação nestes casos. Em casos que apresentam o comportamento inverso, ganhos de desempenho extremamente elevados são obtidos, de modo que ambas as abordagens possam contribuir na verificação de complexas realidades.

6.1 Trabalhos Futuros

Considerando os benefícios e/ou possibilidades de aprimoramento apresentados, são identificados os seguintes trabalhos futuros:

- Otimizar a execução de rodadas de sincronia através do emprego de rotinas assíncronas de comunicação e desenvolver algoritmos de detecção de ponto fixo e término de computação necessários a esta abordagem;
- Otimizar a troca de estados codificando-se estruturas simbólicas em formatos que permitam ser trafegados através de mensagens;
- Empregar algoritmos de coleção de estruturas intermediárias desnecessárias presentes na estrutura de diagramas de decisão adotada; ✓
- Integrar uma abordagem de geração de contraexemplos e testemunhas;
- Comparar o desempenho dos algoritmos descritos no trabalho com outras abordagens de verificação para sistemas distribuídos;
- Otimizar a exploração do paralelismo através do emprego de opções de balanceamento de carga e escalonamento;
- Implementar algoritmos de verificação e técnicas já conhecidas de agrupamento e particionamento de espaços de estados;
- Implementar algoritmo paralelo de rotulação (*Labelling Function*) do espaço de estados.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Allmaier, S. C.; Kowarschik, M.; Horton, G. "State space construction and steady–state solution of gspns on a shared–memory multiprocessor". In: Proceedings of the 6th International Workshop on Petri Nets and Performance Models, 1997, pp. 112–121.

[2] Baier, C.; Katoen, J.-P. "Principles of Model Checking". Cambridge, Massachusetts: MIT Press, 2008, 975p.

[3] Balbo, G. "Introduction to stochastic Petri nets", *Lecture Notes in Computer Science*, vol. 2090, 2001, pp. 84–155.

[4] Baldo, L.; Brenner, L.; Fernandes, L. G.; Fernandes, P.; Sales, A. "Performance Models for Master/Slave Parallel Programs", *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 128, 2005, pp. 101–121.

[5] Barbosa, P. E. S. "Verificação distribuída de modelos: Investigando o uso de grades computacionais", Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, Campina Grande, PB, 2007, 117p.

[6] Barnat, J.; Brim, L.; Cerna, I.; Moravec, P.; Rockai, P.; Simecek, P. "Divine: A tool for distributed verification (tool paper)". In: *Computer Aided Verification*, Ball, T.; Jones, R. (Editores), Springer Berlin / Heidelberg, 2006, *Lecture Notes in Computer Science*, vol. 4144, pp. 278–281.

[7] Behrmann, G.; Hune, T.; Vaandrager, F. "Distributing timed model checking – how the search order matters". In: Proceedings of 12^{th} International Conference on Computer Aided Verification, 2000, pp. 216–231.

[8] Bell, A.; Haverkort, B. R. "Sequential and distributed model checking of petri nets", *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 7, 2005.

[9] Boukala, C.; Petrucci, L. "Distributed CTL model checking and counterexample search". In: Proceedings of 3^{rd} International Workshop on Verification and Evaluation of Computer and Communication Systems, 2009, pp. 1–12.

[10] Brenner, L.; Fernandes, P.; Sales, A. "The Need for and the Advantages of Generalized Tensor Algebra for Structured Kronecker Representations", *International Journal of Simulation: Systems, Science & Technology (IJSIM)*, vol. 6, 2005, pp. 52–60.

[11] Broadfoot, P. J.; Roscoe, A. W. "Tutorial on fdr and its applications". In: Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification, 2000, pp. 322.

[12] Bryant, R. E. "Symbolic boolean manipulation with ordered binary-decision diagrams", *ACM Computing Surveys*, vol. 24, Set 1992, pp. 293–318.

[13] Caselli, S.; Conte, G.; Marenzoni, P. "A distributed algorithm for gspn reachability graph generation", *Journal of Parallel and Distributed Computing*, vol. 61, Jan 2001, pp. 79–95.

[14] Chanin, R.; Corrêa, M.; Fernandes, P.; Sales, A.; Scheer, R.; Zorzo, A. "Analytical Modeling for Operating System Schedulers on NUMA Systems", *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 151, 2006, pp. 131–149.

[15] Ciardo, G.; Gluckman, J.; Nicol, D. "Distributed state-space generation of discrete-state stochastic models", *INFORMS Journal of Computing*, vol. 10, 1998, pp. 82–93.

[16] Ciardo, G.; Jones, R.L., I.; Marmorstein, R.; Miner, A.; Siminiceanu, R. "Smart: stochastic model-checking analyzer for reliability and timing". In: Proceedings of 1st International Conference on Dependable Systems and Networks, 2004, pp. 338–339.

[17] Ciardo, G.; Lüttgen, G.; Miner, A. S. "Exploiting interleaving semantics in symbolic state-space generation", *Form. Methods Syst. Des.*, vol. 31, Ago 2007, pp. 63–100.

[18] Clarke, E. M.; Grumberg, O.; Peled, A. D. "Model Checking". MIT Press, Cambridge, Massachusetts, 1999, 314p.

[19] Correa, C. M.; Dotti, F.; Fernandes, P.; Maruani, E.; Oleksinski, L.; Sales, A. "Um Verificador de Modelos Descritos em Redes de Autômatos Estocásticos". In: XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012): XIII Workshop de Testes e Tolerância a Falhas (WTF 2012), 2012, pp. 115–128.

[20] Czekster, R. "Solução numérica de descritores Markovianos a partir de re-estruturações de termos tensoriais", Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul – Faculdade de Informática – Programa de Pós-graduação em Ciência da Computação, Brasil, 2010, 195p.

[21] Dotti, F.; Fernandes, P.; Sales, A.; Santos, O. "Modular Analytical Performance Models for Ad Hoc Wireless Networks". In: 3rd International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt'05), 2005, pp. 164–173.

[22] Fernandes, P. "Méthodes numériques pour la solution de systèmes markoviens à grand espace d'états", Tese de Doutorado, Institut National Polytechnique de Grenoble - INPG, 1998, 262p.

[23] Fernandes, P.; O'Kelly, M.; Papadopoulos, C.; Sales, A. "Analysis of exponential reliable production lines using kronecker descriptors", *International Journal of Production Research*, 2013, pp. 1–18.

[24] Hinton, A.; Kwiatkowska, M.; Norman, G.; Parker, D. "PRISM: A tool for automatic verification of probabilistic systems". In: Proceedings of 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2006, pp. 441–444.

[25] Holzmann, G. J. "A stack-slicing algorithm for multi-core model checking", *Electronic Notes in Theorical Computer Science*, vol. 198, 2008, pp. 3–16.

[26] Holzmann, G. J.; Bosnacki, D. "The design of a multicore extension of the spin model checker", *IEEE Transactions on Software Engineering*, vol. 33, 2007, pp. 659–674.

[27] Huth, M.; Ryan, M. "Logic in Computer Science: Modelling and Reasoning about Systems". New York, NY, USA: Cambridge University Press, 2004, 440p.

[28] Inggs, C. P.; Barringer, H. "Effective state exploration for model checking on a shared memory architecture", *Electronic Notes in Theoretical Computer Science*, vol. 68, 2002, pp. 605–620.

[29] Iyer, S.; Sahoo, D.; Emerson, E. A.; Jain, J. "On partitioning and symbolic model checking". In: Proceedings of the international conference on Formal Methods, 2005, pp. 497–511.

[30] Joubert, C.; Mateescu, R. (Grid'5000). "Distributed On-the-Fly Model Checking and Test Case Generation", Relatório Técnico RR-5880, INRIA, 2006.

[31] Kumar, R.; Mercer, E. G. "Load balancing parallel explicit state model checking", *Electronic Notes on Theorical Computer Science*, vol. 128, 2005, pp. 19–34.

[32] Laarman, A.; van de, J. P.; Weber, M. "Boosting multi-core reachability performance with shared hash tables". In: 10th International Conference on Formal Methods in Computer-Aided Design, 2010, pp. 247–255.

[33] Law, A. M.; Kelton, W. D. "Simulation Modeling and Analysis". New York City, U.S.: McGraw–Hill Higher Education, 1997, 760p.

[34] Lerda, F.; Sisto, R. "Distributed-memory model checking with spin". In: Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking, 1999, pp. 22–39.

[35] Lerda, F.; Visser, W. "Addressing dynamic issues of program model checking". In: Proceedings of the 8th international SPIN workshop on Model checking of software, 2001, pp. 80–102.

[36] Lluch-lafuente, A.; Edelkamp, S.; Leue, S. "Partial order reduction in directed model checking". In: 9th International SPIN Workshop on Model Checking Software, 2002, pp. 112–127.

[37] McMillan, K. L. "Symbolic model checking: an approach to the state explosion problem", Tese de Doutorado, Carnegie Mellon University, Pittsburgh, PA, USA, 1992, 212p.

[38] Oleksinski, L.; Correa, C.; Dotti, F. L.; Sales, A. "A CTL Model Checker for Stochastic Automata Networks". In: 10th International Conference on the Quantitative Evaluation of Systems (QEST), 2013, pp. 1–4.

[39] OPENMPI. "Openmpi message passing interface". Disponível em: <http://www.open-mpi.org>, Acesso em: Junho, 2012.

[40] Orzan, S. M.; van de Pol, J. C.; Espada, M. V. "A State Space Distribution Policy Based On Abstract Interpretation", *Electronic Notes in Theoretical Computer science*, vol. 128, 2005, pp. 35 – 45.

[41] Plateau, B. "On the stochastic structure of parallelism and synchronization models for distributed algorithms", *SIGMETRICS Performance Evaluation Review*, vol. 13, August 1985, pp. 147–154.

[42] Rauber, T.; Runger, G. "Parallel Programming for Multicore and Cluster Systems". New York, NY: Springer, 2010, 466p.

[43] Rodrigues, C.; Barbosa, P.; Cabral, J.; de Figueiredo, J.; Guerrero, D. "A bag-of-tasks approach for state space exploration using computational grids". In: Fourth IEEE International Conference on Software Engineering and Formal Methods, 2006, pp. 226 –235.

[44] Saad, R. T. "Parallel Model Checking for Multiprocessor Architecture", Tese de Doutorado, L'Institut national des Sciences appliquées de Toulouse, France, 2011, 207p.

[45] Sales, A. "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, 266p.

[46] Sales, A.; Plateau, B. "Reachable state space generation for structured models which use functional transitions". In: 6th International Conference on the Quantitative Evaluation of Systems (QEST'09), 2009, pp. 269–278.

[47] Sistla, A. P.; Godefroid, P. "Symmetry and reduced symmetry in model checking", *ACM Transactions On Programming Languages And Systems*, vol. 26, Jul 2004, pp. 702–734.

[48] Stern, U.; Dill, D. L. "Parallelizing the murφ verifier", *Formal Methods in System Design*, vol. 18, 2001, pp. 117–129.

[49] Stewart, W. J. "Introduction to the numerical solution of Markov Chains". Princeton, New Jersey, USA: Princeton University Press, 1994, 539p.

[50] Stewart, W. J. "Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling". Princeton, New Jersey, USA: Princeton University Press, 2009, 776p.

[51] Triola, M. F.; de Farias e Flores, V. “Introdução à estatística”. Rio de Janeiro - RJ - Brasil: LTC, 2005, 656p.

APÊNDICE A – MODELOS SAN

Este apêndice apresenta a descrição de modelos SAN os quais foram utilizados para geração de resultados com os algoritmos descritos neste trabalho.

- *Dining Philosophers* - Jantar dos Filósofos (Seção A.1).
- *Ad Hoc Wireless Networks* - Cadeia de nodos *Wireless Ad Hoc* (Seção A.2).
- *Production Line* - Linha de Produção (Seção A.3).

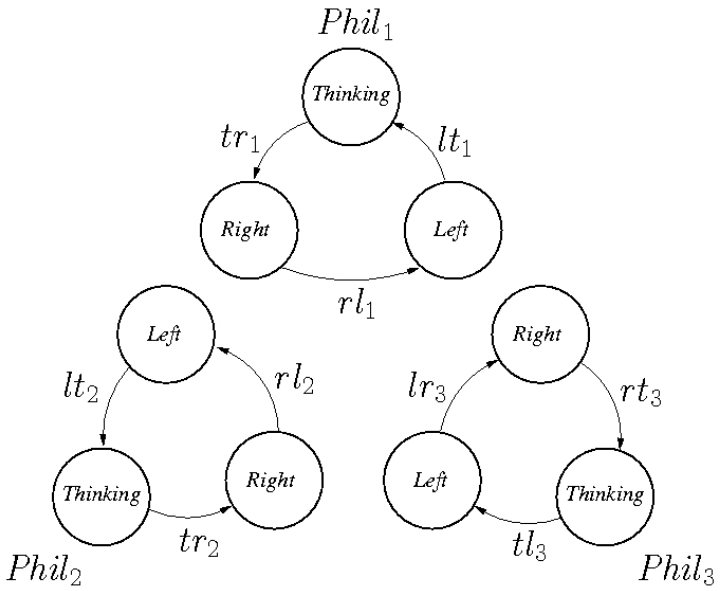
As Figuras A.2, A.4 e A.6 apresentam os modelos SAN descritos no formato textual aceito pela ferramenta relatada no Capítulo 3.

A.1 Jantar dos Filósofos

A Figura A.1 ilustra um modelo SAN para o clássico problema do Jantar dos Filósofos para três filósofos. Este problema pode ser assim descrito: filósofos estão sentados em uma mesa circular e possuem a sua frente um prato com espaguete. Entre cada filósofo participante há um garfo, sendo assim, cada um possui um garfo à sua esquerda e à sua direita. Aos filósofos somente 2 ações são permitidas, pensar ou comer. Para comer, um filósofo necessita de dois garfos nas suas mãos, simultaneamente, caracterizando concorrência entre os filósofos no modelo e problema.

Para melhor entendimento do modelo é necessário conhecer o significado dos estados de cada autômato. Em um dado momento, querendo um filósofo comer, este pega o garfo à sua direita, caso destro, ou o garfo à esquerda, caso canhoto, caracterizando a transição do estado $Thinking$ para $Left$ ou $Right$, dependendo de sua orientação. Sendo possível pegar o segundo garfo, passa para o estado $Left$ ou $Right$, indicando que o respectivo filósofo está comendo. Neste exemplo somente o último filósofo $Phil_3$ é canhoto.

Os eventos locais lt_1, lt_2 e rt_3 não caracterizam dependência, ao contrário dos eventos tr_1, rl_1, tr_2, rl_2, tl_3 e lr_3, os quais possuem taxas funcionais. As funções st e nb são usadas para consultar o estado de um autômato e o número de autômatos que está em um determinado estado, respectivamente. A Figura A.2 apresenta a descrição em formato textual deste modelo.



Type	Event	Rate
loc	tr_1	fr_1
loc	rl_1	fl_1
loc	lt_1	α
loc	tr_2	fr_2
loc	rl_2	fl_2
loc	lt_2	α
loc	tl_3	fl_3
loc	lr_3	fr_3
loc	rt_3	α

$fr_1 = (\text{st } Phil_2 \mathrel{!}= Left) * \lambda$
$fl_1 = (\text{st } Phil_3 \mathrel{!}= Right) * \lambda$
$fr_2 = (\text{st } Phil_3 == Thinking) * \lambda$
$fl_2 = (\text{st } Phil_1 == Thinking) * \lambda$
$fl_3 = (\text{st } Phil_2 == Thinking) * \lambda$
$fr_3 = (\text{st } Phil_1 \mathrel{!}= Left) * \lambda$

$$partial\ reachability = (\text{st } Phil_1 == \text{Thinking}) \;\&\&\; (\text{st } Phil_2 == \text{Thinking}) \;\&\&\; (\text{st } Phil_3 == \text{Thinking})$$

Figura A.1 – Modelo SAN do problema do Jantar dos Filósofos para 3 filósofos.

identifiers

$P = 3$; //Número de filósofos
$lambda = 1.000000$; //Taxa para aquisição de garfos
$alpha = 2.000000$; //Taxa de liberação de garfos

$fr_1 = (st\ Phil_2\ != Left)$ * $lambda$;
$fl_1 = (st\ Phil_3\ != Right)$ * $lambda$;

$fr_2 = (st\ Phil_3 == Thinking)$ * $lambda$;
$fl_2 = (st\ Phil_1 == Thinking)$ * $lambda$;

$fl_3 = (st\ Phil_2 == Thinking)$ * $lambda$;
$fr_3 = (st\ Phil_1\ != Left)$ * $lambda$;

events
$loc\ tr_1\ (fr_1)$;
$loc\ rl_1\ (fl_1)$;
$loc\ lt_1\ (alpha)$;

$loc\ tr_2\ (fr_2)$;
$loc\ rl_2\ (fl_2)$;
$loc\ lt_2\ (alpha)$;

$loc\ tl_3\ (fl_3)$;
$loc\ lr_3\ (fr_3)$;
$loc\ rt_3\ (alpha)$;

partial reachability $= ((nb\ Thinking) == P)$;

network $Philosophers$ (*continuous*)
$aut\ Phil_3$
$stt\ Thinking\ to\ (Left)\ tl_3$
$stt\ Left\ to\ (Right)\ lr_3$
$stt\ Right\ to\ (Thinking)\ rt_3$

$aut\ Phil_2$
$stt\ Thinking\ to\ (Right)\ tr_2$
$stt\ Right\ to\ (Left)\ rl_2$
$stt\ Left\ to\ (Thinking)\ lt_2$

$aut\ Phil_1$
$stt\ Thinking\ to\ (Right)\ tr_1$
$stt\ Right\ to\ (Left)\ rl_1$
$stt\ Left\ to\ (Thinking)\ lt_1$

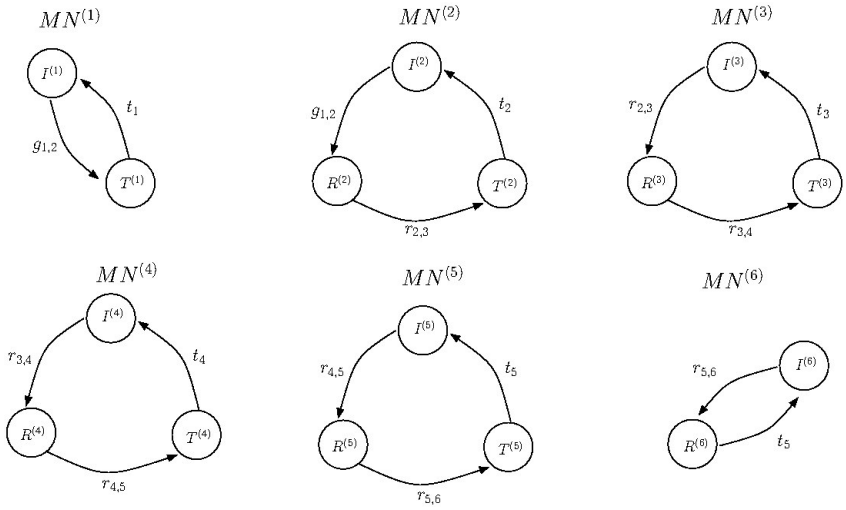
results
$Phil_1Thinking = (st\ Phil_1 == Thinking)$;

Figura A.2 – Modelo SAN do Jantar dos Filósofos para três filósofos apresentado em formato textual.

A.2 *Ad Hoc Wireless Networks*

A Figura A.3 ilustra o modelo SAN referente a uma rede *Wireless* de nodos *Ad Hoc* definida com taxas funcionais, modelada para seis nodos. Neste modelo, três autômatos são definidos: *source*, *relay*, e *sink*. O autômato *source* é responsável por gerar pacotes de acordo com um padrão de comunicação os quais serão trafegados pela cadeia e possui dois estados: $I^{(1)}$ (*idle* - ocioso) e $T^{(1)}$ (*transmitting* - transmitindo). Os pacotes gerados são retransmitidos por autômatos do tipo *relay* ($\mathcal{MN}^{(i)}$, $i = 2..N-1$) os quais possuem três estados, $I^{(i)}$ (*idle*), $R^{(i)}$ (*receiving*) and $T^{(i)}$ (*transmitting*). O autômato *sink* é o último nodo da cadeia e possui dois estados: $I^{(N)}$ (*idle*) e $R^{(N)}$ (*receiving*).

Em um dado momento, caso um nodo i esteja transmitindo um pacote, uma interferência *MAC* afeta os nodos no intervalo $i-2$ até $i+3$. Neste intervalo, somente um nodo pode estar realizando transmissões. Caso contrário, as transmissões são afetadas pela interferência do protocolo adotado, ocasionando perda de pacotes. Mais detalhes sobre este modelo SAN podem ser encontrados em [21]. A Figura A.4 apresenta a descrição em formato textual deste modelo.



$$f_{1,2} = [((stMN^{(3)}! = T^{(3)})\&\&(stMN^{(4)}! = T^{(4)}))] * \lambda$$
$$f_{2,3} = [((stMN^{(1)}! = T^{(1)})\&\&(stMN^{(4)}! = T^{(4)})\&\&(stMN^{(5)}! = T^{(5)})] * \lambda$$
$$f_{3,4} = [((stMN^{(1)}! = T^{(1)})\&\&(stMN^{(2)}! = T^{(2)})\&\&(stMN^{(5)}! = T^{(5)})\&\&(stMN^{(6)}! = T^{(6)}))] * \lambda$$
$$f_{4,5} = [((stMN^{(2)}! = T^{(2)})\&\&(stMN^{(3)}! = T^{(3)})\&\&(stMN^{(6)}! = T^{(6)})] * \lambda$$
$$f_{5,6} = [((stMN^{(3)}! = T^{(3)})\&\&(stMN^{(4)}! = T^{(4)}))] * \lambda$$

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

$$partial\ reachability = (\text{nb I} == 6)$$

Figura A.3 – Modelo SAN representando uma cadeia de nodos *Wireless Ad Hoc* para 6 nodos.

identifiers
$bandwidth\_b$ = 2000000; // largura de banda em bits/segundos
$bandwidth\_B = bandwidth\_b/8$ // largura de banda em bytes/segundos
$bandwidth\_Mbps = bandwidth\_b/1000000$; // largura de banda em megabytes/segundos
$size\_pack$ = 1500; // tamanho do pacote em bytes
RTS = 40; // requisição para envio de cabeçalho em bytes
CTS_ACK = 39; // *ack headers* em bytes
MAC = 47; // *Medium Access Control* header em bytes
$IFT = 50 + (3 * 10) + 280$; // InterFrame Time (DIFS + 3*SIFS + Tempo médio de *Backoff*) em microsegundos (aproximado)
IFS = (($bandwidth\_B$/1000000)*IFT); // ITF - InterFrame Size cost em bytes (aproximado)
$overhead = RTS + CTS\_ACK + MAC + IFS$; // $headers$
mu = ($bandwidth\_B$)/($size\_pack + overhead$); // taxa máxima de *throughput*
$lambda$ = 50000; // taxa de geração de pacotes

$f_{1,2}$ = $lambda$ * (($st\ MN\_3$!= T) && ($st\ MN\_4$!= T));
$f_{2,3}$ = $lambda$ * (($st\ MN\_1$!= T) && ($st\ MN\_4$!= T) && ($st\ MN\_5$!= T));
$f_{3,4}$ = $lambda$ * (($st\ MN\_1$!= T) && ($st\ MN\_2$!= T) && ($st\ MN\_5$!= T) && ($st\ MN\_6$!= T));
$f_{4,5}$ = $lambda$ * (($st\ MN\_2$!= T) && ($st\ MN\_3$!= T) && ($st\ MN\_6$!= T));
$f_{5,6}$ = $lambda$ * (($st\ MN\_3$!= T) && ($st\ MN\_4$!= T));

events
$loc\ t1\ mu$; // transmissão de um pacote
$loc\ t2\ mu$; // transmissão de um pacote
$loc\ t3\ mu$; // transmissão de um pacote
$loc\ t4\ mu$; // transmissão de um pacote
$syn\ t5\ mu$; // transmissão de um pacote
$syn\ g_{1,2}\ f_{1,2}$; // pacote gerado de 1 para 2
$syn\ r_{2,3}\ f_{2,3}$; // pacote encaminhado de 2 para 3
$syn\ r_{3,4}\ f_{3,4}$; // pacote encaminhado de 3 para 4
$syn\ r_{4,5}\ f_{4,5}$; // pacote encaminhado de 4 para 5
$syn\ r_{5,6}\ f_{5,6}$; // pacote encaminhado de 5 para 6

partial reachability = (nb $I == 6$); // estado inicial

network $Adhoc$ (*continuous*)
$aut\ MN\_1$
$stt\ I\ to\ (T)\ g_{1,2}$
$stt\ T\ to\ (I)\ t1$

$aut\ MN\_2$
$stt\ I\ to\ (R)\ g_{1,2}$
$stt\ R\ to\ (T)\ r_{2,3}$
$stt\ T\ to\ (I)\ t2$

$aut\ MN\_3$
$stt\ I\ to\ (R)\ r_{2,3}$
$stt\ R\ to\ (T)\ r_{3,4}$
$stt\ T\ to\ (I)\ t3$

$aut\ MN\_4$
$stt\ I\ to\ (R)\ r_{3,4}$
$stt\ R\ to\ (T)\ r_{4,5}$
$stt\ T\ to\ (I)\ t4$

$aut\ MN\_5$
$stt\ I\ to\ (R)\ r_{4,5}$
$stt\ R\ to\ (T)\ r_{5,6}$
$stt\ T\ to\ (I)\ t5$

$aut\ MN\_6$
$stt\ I\ to\ (R)\ r_{5,6}$
$stt\ R\ to\ (I)\ t5$

results
$nodo1transmitindo = (st\ MN\_1 == T)$;
$nodoNrecebendo = (st\ MN\_N == R)$;

Figura A.4 – Modelo SAN de uma cadeia de nodos *wireless Ad Hoc* para 6 nodos descrita em formato textual.

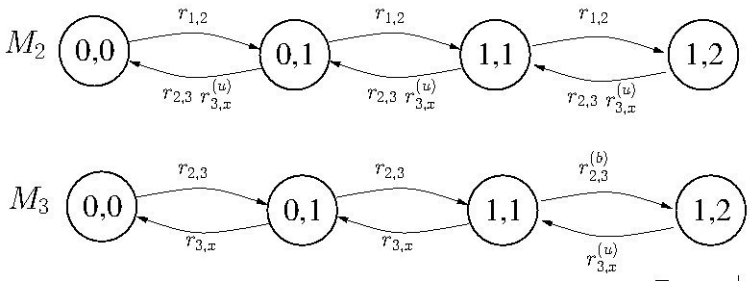
A.3 *Production Line*

A Figura A.5 apresenta um modelo SAN equivalente a uma rede de filas de uma linha de produção com três estações. Neste modelo, cada estação M_i juntamente com seu buffer B_i são modelados como um autômato. A primeira estação M_1 não é modelada uma vez que há uma suposição considerando que esta estação segue enviando trabalho à segunda estação sem interrupções e com taxa μ_1.

Os estados de cada autômato são dados pela combinação da ocupação do buffer n_i da estação correspondente e o estado s_i da estação i (n_i, s_i). Dessa maneira, considerando que as estações da Figura A.5 foram modeladas com buffer possuindo capacidade um, os estados de cada autômato são, respectivamente: (0,0; 0,1; 1,1; 1,2) e (0,0; 0,1; 1,1; 1,2).

Para melhor entendimento os estados são assim descritos: se $s_i = 0$, a estação i está vazia e ociosa. Se $s_i = 1$, a estação i está ocupada e pode estar ou não trabalhando dependendo se está bloqueada ou não, respectivamente. Se $s_i = 2$, a estação i está ocupada e está bloqueando a estação $i - 1$ e pode ou não estar trabalhando dependendo se está bloqueada ou não.

O evento local $r_{1,2}$ possui taxa μ_1 e representa a passagem de trabalho da estação M_1 para a estação M_2. O evento local $r_{3,x}$ representa a saída da estação M_3 enquanto o evento sincronizante $r_{2,3}$ representa a passagem de trabalho da estação M_2 para M_3. Os eventos $r_{2,3}^{(b)}$ e $r_{3,x}^{(u)}$ representam as relações entre as estações M_2 e M_3, isto é, a estação M_3 está ocupada e bloqueando a estação M_2 $r_{2,3}^{(b)}$ ou a mesma completa um trabalho e desbloqueia M_2 $r_{3,x}^{(u)}$. Mais detalhes sobre o modelo podem ser encontrados em [23]. A Figura A.6 apresenta a descrição em formato textual deste modelo para três estações com *buffer* de tamanho 1.



Tipo	Evento	Taxa
loc	$r_{1,2}$	μ_1
sin	$r_{2,3}$	μ_2
loc	$r_{2,3}^{(b)}$	$f_{2,3}^{(b)}$
loc	$r_{3,x}$	μ_3
sin	$r_{3,x}^{(u)}$	μ_3

Função	Definição
$f_{2,3}^{(b)}$	(estado $M_2 \neq 0,0$) * μ_2

$$partial\ reachability = !((\text{st } M_2 == 0,0) \;\&\&\; (\text{st } M_3 == 1,2))$$

Figura A.5 – Modelo SAN representando uma Linha de Produção com 3 estações.

identifiers

$mu_1 = 1.000000$; // taxa de serviço
$mu_2 = 1.000000$; // taxa de serviço
$mu_3 = 1.000000$; // taxa de serviço

// Função de bloqueio
$f2\_3\_b = (st\ M2$!= $st\_0\_0)$ * mu_2;

events

loc $r1\_2$ (mu_1);
syn $r2\_3$ (mu_2);
loc $r2\_3\_b$ $(f2\_3\_b)$;
loc $r3\_x$ (mu_3);
syn $r3\_x\_u$ (mu_3);

partial reachability = !$((st\ M2 == st\_0\_0)$ && $(st\ M3 == st\_1\_2))$;

network P_LINE (*continuous*)

$aut\ M2$
$stt\ st\_0\_0\ to\ (st\_0\_1)\ r1\_2$
$stt\ st\_0\_1\ to\ (st\_1\_1)\ r1\_2$
$to\ (st\_0\_0)\ r2\_3\ r3\_x\_u$
$stt\ st\_1\_1\ to\ (st\_1\_2)\ r1\_2$
$to\ (st\_0\_1)\ r2\_3\ r3\_x\_u$
$stt\ st\_1\_2\ to\ (st\_1\_1)\ r2\_3\ r3\_x\_u$

$aut\ M3$
$stt\ st\_0\_0\ to\ (st\_0\_1)\ r2\_3$
$stt\ st\_0\_1\ to\ (st\_1\_1)\ r2\_3$
$to\ (st\_0\_0)\ r3\_x$
$stt\ st\_1\_1\ to\ (st\_1\_2)\ r2\_3\_b$
$to\ (st\_0\_1)\ r3\_x$
$stt\ st\_1\_2\ to\ (st\_1\_1)\ r3\_x\_u$

results

$estacao\_3\_bloqueada = (st\ M3 == st\_1\_2)$;

Figura A.6 – Modelo SAN de Linha de Produção com 3 estações descrito em formato textual.

APÊNDICE B – RESULTADOS OBTIDOS PARA O EXPERIMENTO 2

Neste Apêndice os resultados de tempo obtidos para a segunda abordagem de execução são apresentados. Esta abordagem visa demonstrar o comportamento do algoritmo em relação ao tráfego de mensagens através da rede da máquina. A máquina adotada (*cluster* gates) é composta por 16 máquinas *Rackable Systems*. Cada nodo possui 2 processadores *AMD Opteron 246 de 2 GHz* e 8 GB de memória e estão interligados por duas redes *Gigabit-Ethernet* chaveadas.

Calculou-se estimativas intervalares para 40 amostras e nível de confiança de 95%. Somente foram realizados experimentos com a respectiva máquina para as propriedades que apresentaram fatores aceleração acima do ideal (*speed-up* superlineares) para o Experimento 3, ou seja, a abordagem de execução descrita no trabalho. Dados de memória não foram incluídos pois estas informações são detalhadas na Seção 5.6.

Tabela B.1 – Resultados de tempo para a Propriedade 5.6 (*starvation*) para o Experimento 2.

Processadores	**μTempo (s)**	σ	**Confiança de 95% (s)**	***Speed-up***
13 Filósofos ($\|S\| = 80.782$)				
1	51,44	3,17	$50{,}46 < \mu < 52{,}42$	1,00
3	33,44	1,71	$32{,}91 < \mu < 33{,}97$	1,53
5	14,09	0,58	$13{,}91 < \mu < 14{,}27$	3,65
7	9,53	0,31	$9{,}44 < \mu < 9{,}63$	5,39
9	6,89	0,15	$6{,}84 < \mu < 6{,}94$	7,45
11	6,48	0,10	$6{,}44 < \mu < 6{,}51$	7,93
13	5,46	0,09	$5{,}43 < \mu < 5{,}49$	9,41
15	4,84	0,09	$4{,}81 < \mu < 4{,}87$	10,62
14 Filósofos ($\|S\| = 195.025$)				
1	352,06	33,42	$341{,}70 < \mu < 362{,}42$	1,00
3	197,60	18,31	$191{,}92 < \mu < 203{,}27$	1,78
5	69,14	20,55	$62{,}77 < \mu < 75{,}51$	5,09
7	38,79	8,50	$36{,}15 < \mu < 41{,}43$	9,07
9	27,50	10,13	$24{,}36 < \mu < 30{,}65$	12,79
11	23,79	11,64	$20{,}19 < \mu < 27{,}40$	14,79
13	17,79	8,36	$15{,}20 < \mu < 20{,}39$	19,78
15	15,08	6,89	$12{,}94 < \mu < 17{,}22$	23,34
15 Filósofos ($\|S\| = 470.832$)				
1	$2{,}21 \times 10^3$	133,02	$2{,}16 \times 10^3 < \mu < 2{,}25 \times 10^3$	1,00
3	$1{,}04 \times 10^3$	42,98	$1{,}03 \times 10^3 < \mu < 1{,}06 \times 10^3$	2,10
5	328,58	7,09	$326{,}38 < \mu < 330{,}78$	6,72
7	221,08	19,55	$215{,}02 < \mu < 227{,}14$	9,99
9	170,57	12,92	$166{,}57 < \mu < 174{,}58$	12,95
11	134,03	10,79	$130{,}68 < \mu < 137{,}37$	16,48
13	98,77	7,46	$96{,}45 < \mu < 101{,}08$	22,37
15	78,08	6,04	$76{,}20 < \mu < 79{,}95$	28,30

Tabela B.2 – Resultados de tempo para a Propriedade 5.7 (exclusão mútua) para o Experimento 2.

Processadores	μ**Tempo (s)**	σ	**Confiança de 95% (s)**	***Speed-up***
13 Filósofos ($\|S\| = 80.782$)				
1	40,60	2,22	$39{,}91 < \mu < 41{,}29$	1,00
3	19,38	0,35	$19{,}27 < \mu < 19{,}49$	2,09
5	9,89	0,10	$9{,}86 < \mu < 9{,}92$	4,10
7	7,11	0,08	$7{,}08 < \mu < 7{,}13$	5,70
9	5,81	0,05	$5{,}79 < \mu < 5{,}83$	6,98
11	5,00	0,05	$4{,}98 < \mu < 5{,}01$	8,11
13	4,68	0,03	$4{,}67 < \mu < 4{,}69$	8,66
15	4,29	0,03	$4{,}28 < \mu < 4{,}30$	9,44
14 Filósofos ($\|S\| = 195.025$)				
1	270,42	24,20	$262{,}92 < \mu < 277{,}92$	1,00
3	102,79	5,76	$101{,}00 < \mu < 104{,}57$	2,63
5	41,30	1,18	$40{,}93 < \mu < 41{,}67$	6,54
7	25,02	0,64	$24{,}82 < \mu < 25{,}22$	10,80
9	18,06	0,70	$17{,}84 < \mu < 18{,}27$	14,97
11	13,33	0,56	$13{,}15 < \mu < 13{,}50$	20,28
13	11,18	0,61	$10{,}99 < \mu < 11{,}37$	24,18
15	9,89	0,21	$9{,}82 < \mu < 9{,}95$	27,34
15 Filósofos ($\|S\| = 470.832$)				
1	$1{,}57 \times 10^3$	89,38	$1{,}54 \times 10^3 < \mu < 1{,}60 \times 10^3$	1,00
3	563,26	31,19	$553{,}59 < \mu < 572{,}93$	2,78
5	241,58	9,21	$238{,}73 < \mu < 244{,}44$	6,49
7	172,27	15,42	$167{,}49 < \mu < 177{,}04$	9,11
9	117,01	9,77	$113{,}98 < \mu < 120{,}04$	13,41
11	86,84	6,86	$84{,}71 < \mu < 88{,}97$	18,07
13	70,49	5,27	$68{,}86 < \mu < 72{,}12$	22,27
15	56,49	4,05	$55{,}23 < \mu < 57{,}74$	27,79

Tabela B.3 – Resultados de tempo para a Propriedade 5.8 para o Experimento 2.

Processadores	μ**Tempo (s)**	σ	**Confiança de 95% (s)**	***Speed-up***
13 Filósofos ($\|S\| = 80.782$)				
1	57,54	3,41	$56{,}48 < \mu < 58{,}60$	1,00
3	24,73	0,60	$24{,}54 < \mu < 24{,}92$	2,32
5	12,16	0,18	$12{,}10 < \mu < 12{,}22$	4,73
7	8,54	0,11	$8{,}50 < \mu < 8{,}57$	6,73
9	6,94	0,06	$6{,}92 < \mu < 6{,}96$	8,28
11	5,89	0,05	$5{,}87 < \mu < 5{,}90$	9,77
13	5,30	0,02	$5{,}29 < \mu < 5{,}31$	10,84
15	4,98	0,05	$4{,}96 < \mu < 5{,}00$	11,54
14 Filósofos ($\|S\| = 195.025$)				
1	311,87	19,10	$305{,}95 < \mu < 317{,}79$	1,00
3	144,17	11,68	$140{,}55 < \mu < 147{,}79$	2,16
5	59,09	9,06	$56{,}28 < \mu < 61{,}90$	5,27
7	41,64	11,72	$38{,}01 < \mu < 45{,}27$	7,48
9	33,49	13,87	$29{,}19 < \mu < 37{,}79$	9,31
11	29,34	16,73	$24{,}15 < \mu < 34{,}52$	10,62
13	29,98	20,69	$23{,}57 < \mu < 36{,}40$	10,40
15	27,31	19,55	$21{,}25 < \mu < 33{,}36$	11,42
15 Filósofos ($\|S\| = 470.832$)				
1	-	-	$- < \mu < -$	-
3	925,58	12,32	$921{,}77 < \mu < 929{,}40$	-
5	326,92	18,92	$321{,}06 < \mu < 332{,}79$	-
7	168,03	9,81	$164{,}98 < \mu < 171{,}07$	-
9	111,06	8,06	$108{,}56 < \mu < 113{,}56$	-
11	70,85	1,66	$70{,}33 < \mu < 71{,}36$	-
13	54,71	1,85	$54{,}13 < \mu < 55{,}28$	-
15	45,18	1,29	$44{,}78 < \mu < 45{,}58$	-

Tabela B.4 – Resultados de tempo para a Propriedade 5.10 para o Experimento 2.

Processadores	μ**Tempo (s)**	σ	**Confiança de 95% (s)**	***Speed-up***
13 Filósofos ($\|S\| = 80.782$)				
1	58,18	2,93	$55{,}25 < \mu < 61{,}12$	1,00
3	34,84	4,68	$33{,}38 < \mu < 36{,}29$	1,67
5	18,83	5,73	$17{,}05 < \mu < 20{,}60$	3,09
7	14,57	6,98	$12{,}40 < \mu < 16{,}73$	3,99
9	13,38	8,32	$10{,}80 < \mu < 15{,}96$	4,34
11	11,82	6,95	$9{,}67 < \mu < 13{,}98$	4,92
13	10,87	5,73	$9{,}10 < \mu < 12{,}65$	5,35
15	10,25	4,50	$8{,}86 < \mu < 11{,}65$	5,67
14 Filósofos ($\|S\| = 195.025$)				
1	313,00	22,14	$290{,}85 < \mu < 335{,}15$	1,00
3	177,17	19,09	$171{,}26 < \mu < 183{,}09$	1,76
5	81,45	17,15	$76{,}13 < \mu < 86{,}76$	3,84
7	53,90	19,69	$47{,}80 < \mu < 60{,}01$	5,80
9	41,01	18,06	$35{,}41 < \mu < 46{,}61$	7,63
11	34,01	17,32	$28{,}64 < \mu < 39{,}38$	9,20
13	32,12	17,90	$26{,}57 < \mu < 37{,}67$	9,74
15	28,57	16,89	$23{,}34 < \mu < 33{,}81$	10,95
15 Filósofos ($\|S\| = 470.832$)				
1	-	-	$- < \mu < -$	-
3	799,14	16,02	$794{,}18 < \mu < 804{,}11$	-
5	284,22	10,00	$281{,}12 < \mu < 287{,}32$	-
7	146,90	2,15	$146{,}24 < \mu < 147{,}57$	-
9	97,92	2,45	$97{,}17 < \mu < 98{,}68$	-
11	69,20	2,08	$68{,}56 < \mu < 69{,}85$	-
13	56,97	2,20	$56{,}29 < \mu < 57{,}66$	-
15	45,43	1,48	$44{,}97 < \mu < 45{,}89$	-

Tabela B.5 – Resultados de tempo para a Propriedade 5.9 para o Experimento 2.

Processadores	μTempo (s)	σ	Confiança de 95% (s)	*Speed-up*
Sem quebra da árvore sintática				
15 filósofos ($\|S\| = 470.832$)				
1	63,82	3,57	$62{,}71 < \mu < 64{,}93$	1,00
3	36,93	1,79	$36{,}37 < \mu < 37{,}48$	1,72
5	16,08	0,57	$15{,}90 < \mu < 16{,}26$	3,96
7	10,46	0,34	$10{,}35 < \mu < 10{,}57$	6,09
9	7,62	0,28	$7{,}53 < \mu < 7{,}71$	8,37
11	7,04	0,45	$6{,}90 < \mu < 7{,}18$	9,05
13	5,79	0,14	$5{,}75 < \mu < 5{,}84$	11,00
15	5,36	0,12	$5{,}33 < \mu < 5{,}40$	11,89
16 filósofos ($\|S\| = 1.136.689$)				
1	387,26	41,95	$374{,}26 < \mu < 400{,}26$	1,00
3	206,51	16,38	$201{,}43 < \mu < 211{,}58$	1,87
5	74,85	3,56	$73{,}75 < \mu < 75{,}96$	5,17
7	36,98	2,22	$36{,}29 < \mu < 37{,}67$	10,47
9	24,42	1,09	$24{,}08 < \mu < 24{,}75$	15,85
11	20,02	0,75	$19{,}79 < \mu < 20{,}26$	19,33
13	15,60	0,69	$15{,}38 < \mu < 15{,}82$	24,81
15	12,97	0,50	$12{,}82 < \mu < 13{,}13$	29,84
17 filósofos ($\|S\| = 2.744.210$)				
1	$2{,}89 \times 10^3$	92,39	$2{,}86 \times 10^3 < \mu < 2{,}92 \times 10^3$	1,00
3	$1{,}47 \times 10^3$	91,60	$1{,}44 \times 10^3 < \mu < 1{,}50 \times 10^3$	1,95
5	466,95	26,09	$458{,}86 < \mu < 475{,}03$	6,19
7	204,83	13,79	$200{,}56 < \mu < 209{,}10$	14,11
9	121,36	8,23	$118{,}81 < \mu < 123{,}91$	23,83
11	92,25	6,70	$90{,}17 < \mu < 94{,}33$	31,34
13	65,72	3,73	$64{,}56 < \mu < 66{,}88$	44,00
15	51,04	3,12	$50{,}07 < \mu < 52{,}01$	56,65
Com quebra da árvore sintática				
15 filósofos ($\|S\| = 470.832$)				
1	63,82	3,57	$62{,}71 < \mu < 64{,}93$	1,00
3	67,20	4,76	$65{,}72 < \mu < 68{,}67$	0,02
5	41,55	2,32	$40{,}83 < \mu < 42{,}27$	1,53
7	23,01	1,29	$22{,}60 < \mu < 23{,}41$	2,77
9	17,29	0,89	$17{,}01 < \mu < 17{,}56$	3,69
11	13,22	0,11	$13{,}18 < \mu < 13{,}25$	4,82
13	10,06	0,04	$10{,}05 < \mu < 10{,}07$	6,34
15	9,25	0,06	$9{,}23 < \mu < 9{,}27$	6,89
16 filósofos ($\|S\| = 1.136.689$)				
1	387,26	41,95	$374{,}26 < \mu < 400{,}26$	1,00
3	410,23	36,94	$398{,}78 < \mu < 421{,}68$	0,01
5	239,25	18,04	$233{,}65 < \mu < 244{,}84$	1,61
7	119,39	10,98	$115{,}98 < \mu < 122{,}80$	3,24
9	87,39	9,57	$84{,}42 < \mu < 90{,}36$	4,43
11	62,27	10,47	$59{,}02 < \mu < 65{,}51$	6,21
13	44,11	9,57	$41{,}15 < \mu < 47{,}08$	8,77
15	38,02	9,45	$35{,}09 < \mu < 40{,}95$	10,18
17 filósofos ($\|S\| = 2.744.210$)				
1	$2{,}89 \times 10^3$	92,39	$2{,}86 \times 10^3 < \mu < 2{,}92 \times 10^3$	1,00
3	$2{,}67 \times 10^3$	106,96	$2{,}63 \times 10^3 < \mu < 2{,}70 \times 10^3$	0,92
5	$1{,}62 \times 10^3$	61,63	$1{,}60 \times 10^3 < \mu < 1{,}64 \times 10^3$	1,77
7	733,52	42,59	$720{,}32 < \mu < 746{,}71$	3,94
9	508,49	44,71	$494{,}64 < \mu < 522{,}35$	5,68
11	325,90	9,96	$322{,}81 < \mu < 328{,}99$	8,87
13	212,73	4,21	$211{,}42 < \mu < 214{,}03$	13,59
15	174,91	2,06	$174{,}27 < \mu < 175{,}55$	16,53

Tabela B.6 – Resultados de tempo para a Propriedade 5.13 para o Experimento 2.

Processadores	μ**Tempo (s)**	σ	**Confiança de 95% (s)**	***Speed-up***
24 nodos ($\|S\| = 82.946$)				
1	21,03	0,43	$20{,}90 < \mu < 21{,}17$	1,00
3	17,24	7,16	$15{,}02 < \mu < 19{,}46$	1,22
5	13,44	7,48	$11{,}12 < \mu < 15{,}76$	1,56
7	11,12	6,54	$9{,}09 < \mu < 13{,}14$	1,89
9	12,51	7,64	$10{,}14 < \mu < 14{,}88$	1,68
11	12,31	7,82	$9{,}88 < \mu < 14{,}73$	1,70
13	10,71	7,49	$8{,}38 < \mu < 13{,}03$	1,96
15	8,85	7,25	$6{,}60 < \mu < 11{,}10$	2,37
26 nodos ($\|S\| = 217.254$)				
1	113,05	7,37	$110{,}76 < \mu < 115{,}33$	1,00
3	49,07	4,36	$47{,}72 < \mu < 50{,}42$	2,30
5	25,66	4,77	$24{,}18 < \mu < 27{,}14$	4,40
7	18,67	4,68	$17{,}22 < \mu < 20{,}12$	6,05
9	15,44	4,22	$14{,}13 < \mu < 16{,}75$	7,32
11	13,09	5,96	$11{,}25 < \mu < 14{,}94$	8,63
13	13,18	7,00	$11{,}00 < \mu < 15{,}35$	8,57
15	10,52	4,24	$9{,}21 < \mu < 11{,}84$	10,73
28 nodos ($\|S\| = 568.518$)				
1	852,25	11,96	$848{,}54 < \mu < 855{,}95$	1,00
3	322,82	13,31	$318{,}69 < \mu < 326{,}95$	2,64
5	127,79	10,26	$124{,}61 < \mu < 130{,}97$	6,66
7	75,84	8,78	$73{,}12 < \mu < 78{,}56$	11,23
9	54,88	9,38	$51{,}98 < \mu < 57{,}79$	15,52
11	43,35	8,95	$40{,}57 < \mu < 46{,}12$	19,65
13	36,75	8,10	$34{,}23 < \mu < 39{,}26$	23,19
15	34,02	9,91	$30{,}94 < \mu < 37{,}09$	25,05