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

TRADUÇÃO DE MODELOS DE REDES DE AUTOMATOS ESTOCÁSTICOS PARA A LINGUAGEM DO NUSMV

ALBERTO DO CARMO SULZBACHER WONDRACEK

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. Fernando Luís Dotti

Porto Alegre
2013

Dados Internacionais de Catalogação na Publicação (CIP)

W872t Wondracek, Alberto do Carmo Sulzbacher
Tradução de modelos de redes de automatos estocásticos para a linguagem do NuSMV / Alberto do Carmo Sulzbacher Wondracek. - Porto Alegre, 2014.
267 p.

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

1. Informática. 2. Redes de Autômatos Estocásticos. 3. 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 "Tradução de Modelos de Redes de Automatos Estocásticos para a Linguagem do NuSMV" apresentada por Alberto do Carmo Sulzbacher Wondracek 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 05/06/2014, conforme Ata No. 010 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

TRADUÇÃO DE MODELOS DE REDES DE AUTOMATOS ESTOCÁSTICOS PARA A LINGUAGEM DO NUSMV

RESUMO

Redes de autômatos estocásticos (SAN) é um formalismo que permite a descrição de sistemas a fim de realizar avaliações quantitativas. O objetivo deste trabalho é possibilitar avaliações qualitativas de modelos SAN através de sua tradução para a linguagem de um verificador existente. O trabalho propõe, detalha e exemplifica o mapeamento de um subconjunto de modelos SAN para a linguagem de entrada do NuSMV. Conforme o resultado observado, os modelos para o NuSMV gerados pelo tradutor preservam a semântica dos respectivos modelos SAN originais pois apresentam sistemas de transição de estados isomórficos. A verificação de propriedades em CTL (*Computation Tree Logic*) sobre os modelos SAN é exemplificada.

Palavras Chave: Verificação de Modelos, SAN, NuSMV, Tradução de Modelos

TRANSLATION OF STOCHASTIC AUTOMATA NETWORK MODELS TO NUSMV INPUT LANGUAGE

ABSTRACT

Stochastic Automata Network (SAN) is a formalism that allows the description of systems in order to evaluate them quantitatively. The aim of this work is to enable the qualitative evaluation on SAN models through its translation to the language of an existent model checker. This work proposes, details and exemplifies the mapping of a subset of SAN models to the NuSMV input language. As observed, the NuSMV models generated by the translator preserve the semantic of its originals SAN models because they have an isomorphic transition state system. The model checking through CTL (Computation Tree Logic) on SAN models is exemplified as well.

Keywords: Model Checking, SAN, NuSMV, Model Translation

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos tem crescido o desenvolvimento de sistemas digitais na maior parte do mundo, sendo cada vez mais importante a validação da corretude deles.

Cerca de 30% a 50% do custo de um projeto de software é destinado para a fase de qualificação, sendo que as duas técnicas de verificação mais utilizadas são a revisão em pares e a execução de testes. Apesar da revisão em pares identificar em média 60% dos defeitos, erros sutis de concorrência e em algoritmos dificilmente são identificados por meio desta técnica. Diferentemente da primeira técnica, que analisa o código de forma estática, através da segunda é possível executar o código e verificar se o resultado é igual ao esperado. Como é praticamente inviável verificar todas as possíveis computações de um software, usualmente se utiliza apenas um subconjunto a fim de criar os testes. Portanto, a execução de testes possibilita identificar a presença de erros, mas não garante a sua ausência.

O custo para correção de uma falha identificada através destas técnicas pode ser até 500 vezes mais alto em relação a uma correção efetuada na fase conceitual do projeto. Portanto, quanto mais complexo for o projeto e maior o impacto do risco de insucesso, maior será a exigência de investimentos em análises preliminares. Como foi possível verificar, o foguete Ariane 5, lançado pela *European Space Agency* (ESA) em junho de 1996, devido a validação incorreta do sitema resultou num prejuízo de 7,5 bilhões de dólares. Há também o caso do Therac 25, uma máquina de radioterapia controlada pelo computador, que foi responsável por mortes e danos a saúde dos pacientes, entre os anos de 1985 e 1987, devido à radiação em excesso. Ambas as falhas poderiam ter sido identificadas através de uma técnica de verificação formal.

Métodos formais são muito eficientes na realização de verificações durante a fase conceitual possibilitando a redução do tempo total de verificação do sistema. Em uma abstração de alto nível, métodos formais podem ser considerados a aplicação matemática para modelagem e análise de sistemas digitais que tem por finalidade obter a corretude de sistemas com o rigor matemático. Além disso, os métodos formais são técnicas de verificação altamente recomendadas para o

desenvolvimento de software crítico de segurança de acordo com o padrão de melhores práticas do *International Electrotechnical Commision* (IEC) e dos padrões da ESA.

Uma técnica de verificação formal é a verificação de modelos, conhecida como *Model Checking*. Esta técnica permite realizar análises qualitativas a respeito de um modelo de estados finitos. Através destas análises, é possível verificar se um conjunto de propriedades é respeitado pelo modelo. As propriedades são tipicamente especificadas em lógica temporal, como por exemplo, *Computation Tree Logic* (CTL). Um grande atrativo desta técnica é o fato de que ela é completamente automática, mostrando contra exemplos caso o modelo não satisfaça alguma propriedade. Portanto, diferentemente das técnicas mais utilizadas (revisão de pares e execução de testes) para validação de sistemas, *Model Checking* valida a ausência de defeitos para as propriedades especificadas. Sua aplicação não se restringe apenas a projetos de software, sendo aplicado também em projetos de hardware.

É possível realizar *Model Checking* para diversos formalismos existentes, inclusive para formalismos voltados a avaliação quantitativa. O formalistmo Redes de Autômatos Estocásticos (*Stochastic Automata Network* – SAN) é um dentre os diversos formalismos que objetivam a avaliação quantitativa de sistemas. SAN é um formalismo estruturado, com conceitos simples – permitindo a descrição de modelos de forma fácil e clara – e focado em sistemas concorrentes com sincronização e dependências. Além disso, SAN conta com um conjunto de técnicas e ferramentas que possibilita a avaliação quantitativa de modelos descritos em SAN.

A motivação desta dissertação é decorrente da ausência da possibilidade de realizar análises qualitativas, através de *Model Cheking*, para modelos SAN a fim de obter a certeza da corretude dos modelos SAN existentes. Consequentemente, o objetivo deste trabalho é possibilitar a análise qualitativa, realizada através de *Model Checking*, de modelos descritos em SAN. Este objetivo pode ser alcançado a partir de duas abordagens: (i) desenvolvendo um verificador de modelo SAN e (ii) mapeando os conceitos SAN para linguagem de entrada de uma ferramenta existente de verificação. Neste trabalho é utilizada a segunda abordagem, e pela facilidade de modelar os conceitos de eventos sincronizantes para a linguagem da ferramenta NuSMV, comparando com outras linguagens de ambientes de *Model Checking*, optou-se por esta ferramenta.

Portanto, esta dissertação propõe um mapeamento de parte dos conceitos de SAN para a linguagem do NuSMV, apresenta os modelos SAN traduzidos bem como realiza análises qualitativas, por meio de propriedades CTL, através do NuSMV com as traduções dos modelos SAN.

Este trabalho está estruturado da seguinte forma: são apresentados os conceitos da linguagem SAN no Capítulo 2; a ferramenta NuSMV e alguns conceitos de sua linguagem são abordados no Capítulo 3; em seguida, no Capítulo 4 é explicado o conceito de CTL, os seus operadores, sua sintaxe e a aplicação de propriedades CTL sobre modelos SAN; no Capítulo 5 é apresentado os trabalhos relacionados; no Capítulo 6 é explicado o mapeamento de alguns dos conceitos SAN para a linguagem do NuSMV, a estrutura geral do código gerado para o NuSMV, o detalhamento da tradução apresentando pequenos exemplos, a tradução das operações SAN, o algoritmo de tradução de expressões SAN e o processo que foi utilizado para o desenvolvimento do tradutor bem como suas limitações; no Capítulo 7 é apresentada uma discussão sobre a corretude do tradutor SAN assim como o algoritmo de caminhamento desenvolvido; no Capítulo 8 é apresentado exemplos completos da tradução de modelos SAN para a linguagem do NuSMV assim como avaliações qualitativas realizadas com modelos traduzidos; e por fim a conclusão é apresentada no Capítulo 9.

2 REDES DE AUTÔMATOS ESTOCÁSTICOS (SAN)

De acordo com Fernandes et. al. [14], SAN possibilita a descrição de modelos de sistemas com o objetivo de avaliar seu desempenho. Ao representar um sistema, SAN permite a descrição deste sistema através de um conjunto de subsistemas, onde cada subsistema é modelado por um autômato estocástico, e pela interação deles entre si. O autômato estocástico é constituído por estados e transições, que são as mudanças do estado atual, ocasionadas por eventos locais ou sincronizantes.

Os eventos modelam fenômenos aleatórios da realidade em questão, por exemplo, o tempo de serviço de um servidor, o intervalo de tempo entre as chegadas à fila 1. Eles disparam transições, que mudam o estado de um ou mais autômatos. Os eventos locais disparam uma transição em apenas um autômato, já os sincronizantes são responsáveis por transições simultâneas em todos os autômatos onde constam mais de um autômato, representando uma importante forma de comunicação entre autômatos. Cada evento possui uma taxa de ocorrência, sendo um valor numérico ou o retorno de uma função. Funções avaliam sobre o estado global da rede, retornando um valor. Assim, um evento de um autômato pode ter uma taxa descrita por uma função cuja avaliação depende de outros autômatos, representando outra forma de comunicação ou dependência entre autômatos.

SAN possui operadores próprios de sua linguagem, tais como: *st, nb e rw*. A lista completa pode ser consultada em [3]. O operador *st* é aplicado sobre um autômato retornando o estado em que tal autômato se encontra. A fim de saber quantos autômatos no modelo se encontram em determinado estado, é utilizado o operador *nb* informando o estado desejado. O operador *rw* é aplicado sobre um autômato retornando o valor de *reward* que o estado atual daquele autômato possui.

As funções em SAN possuem grande capacidade de representação, sendo possível juntar operações de diferentes tipos na mesma expressão, como por exemplo, efetuar uma operação matemática com o resultado de uma operação lógica ao resultado de uma expressão SAN. Elas podem avaliar o estado do modelo SAN, ou seja, expressões que dado um estado global avaliam e retornam um valor.

Um autômato pode possuir em sua especificação um número de replicações do autômato ou de um determinado estado, provendo à modelagem uma capacidade de representação poderosa, pois não será necessário reescrever todo o autômato ou um determinado estado diversas vezes, mas apenas indicar o número de vezes a ser replicado. Ao estado de um autômato é possível atribuir um valor numérico de *reward*, que é utilizado pelo operador *rw*. Caso não seja atribuído um valor de *reward* a algum estado, este assume um valor sequencial, de acordo com a ordem da listagem deste estado dentro do autômato.

Ao modelar uma realidade em SAN é necessário informar os estados atingíveis, sendo possível determinar todos os estados atingíveis ou apenas parte deles através da função de alcançabilidade *reachability* ou alcançabilidade parcial *partial reachability*, respectivamente. Dado um estado do espaço de estados produto, a função de alcançabilidade retorna verdadeiro ou falso caso o estado seja respectivamente alcançável ou não. A função de alcançabilidade parcial retorna verdadeiro somente para um subconjunto dos estados alcançáveis pelo modelo. Ao utilizar a seção *partial reachability,* o PEPS (*Performance Evaluation Of Parallel Systems*), ferramenta que avalia os modelos SAN, descobre os outros estados atingíveis a partir do conjunto de estados definidos nesta seção [3].

2.1 Modelo SAN – Jantar dos Filósofos

O Modelo SAN apresentado nesta seção representa o problema do jantar dos filósofos, modelado com quatro filósofos. Este modelo foi retirado de [5] [24]. Este modelo apresenta quatro autômatos, respectivos a cada filósofo: P0, P1, P2 e P3. Todos os autômatos deste modelo possuem os mesmos estados: *Thinking, Right* e *Left*. Entretanto alguns possuem transições diferentes e todas as transições são originadas por eventos diferentes. Abaixo é apresentada a descrição textual deste modelo.

```
//=== Dining philosophers problem (P=4) ===

identifiers

  // Number of philosophers
  P = 4;
  // Acquisition rate
  lambda = 1.000000;
  // Release rate
  mu = 2.000000;
```

```
events

  syn t_r_0 (lambda);
  syn r_l_0 (lambda);
  loc l_t_0 (mu);

  syn t_r_1 (lambda);
  syn r_l_1 (lambda);
  loc l_t_1 (mu);

  syn t_r_2 (lambda);
  syn r_l_2 (lambda);
  loc l_t_2 (mu);

  syn t_l_3 (lambda);
  syn l_r_3 (lambda);
  loc r_t_3 (mu);

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

network PHILOSOPHERS (continuous)

  aut P3
    stt Thinking to (Left)     t_l_3
                to (Thinking) r_l_0 t_r_2
    stt Left     to (Right)    l_r_3
                to (Left)     r_l_0
    stt Right    to (Thinking) r_t_3

  aut P2
    stt Thinking to (Right)    t_r_2
                to (Thinking) t_r_1 t_l_3
    stt Right    to (Left)     r_l_2
                to (Right)    t_r_1
    stt Left     to (Thinking) l_t_2

  aut P1
    stt Thinking to (Right)    t_r_1
                to (Thinking) t_r_0 r_l_2
    stt Right    to (Left)     r_l_1
                to (Right)    t_r_0
    stt Left     to (Thinking) l_t_1

  aut P0
    stt Thinking to (Right)    t_r_0
                to (Thinking) r_l_1 l_r_3
    stt Right    to (Left)     r_l_0
                to (Right)    l_r_3
          stt Left     to (Thinking) l_t_0
```

Na imagem e tabela abaixo é possível verificar a estrutura de cada autômato e os eventos do modelo, respectivamente. A função de alcançabilidade parcial deste modelo, como é possível verificar acima, é *((nb Thinking) == P)*.

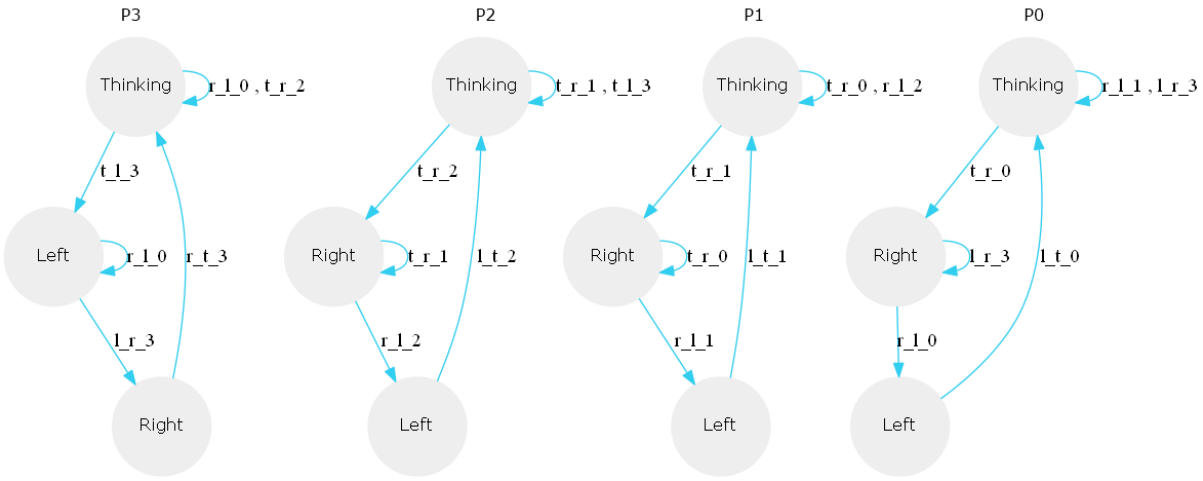


Figura 1: Representação gráfica de phil04c.san

Tabela 1: Eventos de phil04c.san

Evento	Tipo	Taxa
t_r_0	sincronizante	1
r_l_0	sincronizante	1
l_t_0	Local	2
t_r_1	sincronizante	1
r_l_1	sincronizante	1
l_t_1	Local	2
t_r_2	sincronizante	1
r_l_2	sincronizante	1
l_t_2	Local	2
t_l_3	sincronizante	1
l_r_3	sincronizante	1
r_t_3	Local	2

3 NUSMV

O NuSMV é um verificador simbólico de modelos (*Symbolic Model Checker*) desenvolvido a partir da reestruturação da ferramenta SMV (*Symbolic Model Verifier*), o verificador original que utiliza o conceito de BDD (*Binary Decision Diagrams*) para efetuar a verificação de propriedades de lógica temporal CTL [19].

O NuSMV foi desenvolvido em parceria pelas seguintes instituições: Carnegie Mellon University (CMU), Instituto per la Ricerca Scientifica e Tecnologica (IRST), University of Genova (UG) e University of Trento (UT) [21].

3.1 Linguagem do NuSMV

Nesta seção são apresentados os principais conceitos da linguagem do NuSMV, com base no “NuSMV 2.5 User Manual” [7] e no “NuSMV 2.5 Tutorial” [8].

Um modelo na linguagem do NuSMV possui um ou mais módulos, sendo o módulo *main* obrigatório. Cada módulo possui algumas seções especiais, tais como: *VAR, INIT, TRANS, DEFINE, CTLSPEC*, entre outras que não estão no escopo deste trabalho.

A declaração de variáveis dentro de um módulo é realizada dentro da seção *VAR*. Cada variável possui um nome e um tipo, seguindo a seguinte estrutura “*Nome : Tipo*”. Os tipos que a linguagens do NuSMV dispõem são: booleanos, inteiros, enumeradores, *array*, módulos, entre outros. Uma variável do tipo enumerador é composta por valores numéricos inteiros ou simbólicos, ou por ambos. Estes valores {1, 200, 10, -5, 4}, {OK, NotOK, 33, -2}, {a, B, c, X, Y, z} são alguns exemplos de valores possíveis do tipo enumerador.

A seção *INIT* possui uma expressão booleana a qual determina um estado inicial ou um conjunto de estados iniciais para o modelo. A expressão booleana “*var1 = valor1 & var2 = valor2 & var3 = valor3*” é um exemplo que pode ser utilizado na seção *INIT* para determinar os valores iniciais de três das variáveis de um determinado modelo.

As transições de um modelo são descritas na seção *TRANS* através de expressões booleanas que avaliam o estado atual e o próximo estado das variáveis do modelo. A fim de verificar o próximo estado de uma determinada variável é

utilizado o operador *next* da seguinte forma *next(variavel) = valor*. Desta forma a expressão "*var1 = valor1 & next(var1) = valor2*" descreve a transição do valor *valor1* da variável *var1* para o *valor2*.

A fim de utilizar a mesma expressão em diversos lugares do modelo atribui-se esta expressão a um identificador, que representa a expressão associada a ele, podendo ser utilizado quantas vezes forem necessárias descrendo a expressão somente uma vez. Esta atribuição de expressão a um identificador é o conceito de um *define*, os quais são descritos na seção *DEFINE* do módulo. Os *defines* são avaliados em tempo de execução, portanto, eles não possuem um tipo de retorno fixo como as variáveis.

Assim como diversas linguagens de programação, a linguagem do NuSMV possui a expressão chamada *If-Then-Else* ou operador ternário. Sua estrutura é representada da seguinte forma "*ExpressãoDeCondição ? Valor1 : Valor2*". Após avaliar o valor de *ExpressãoDeCondição*, esta expressão ternária retorna o *Valor1* caso *ExpressãoDeCondição* for verdadeiro e em caso contrário retorna o *Valor2*.

A seção *CTLSPEC* serve para especificar as propriedades em lógica temporal CTL que realizam uma análise qualitativa do modelo. Ao verificar as propriedades CTL do modelo, caso alguma delas não for verdadeira é apresentado pelo NuSMV um contraexemplo.

Além do módulo *main*, é possível criar outros módulos que possuam as mesmas seções anteriormente explicadas. Ao criar uma instância de um módulo é encapsulada toda a descrição do módulo – variável, *defines,* transições e outras características – e é criado um novo contexto para esta instância, não existindo conflito na estrutura de dados de duas instâncias diferentes. Para instanciar um módulo, basta criar uma variável do tipo daquele módulo, conforme anteriormente explicado. É possível acessar as variáveis e *defines* que uma instância de módulo possui através do operador "." (ponto) para entrar no contexto daquela instância. Por exemplo, dado um módulo X que possui a variável V e a instância A deste módulo, é possível acessar a variável V da seguinte forma "A.V".

No exemplo abaixo, retirado de [8], é possível verificar o uso de alguns dos conceitos acima explicados. De acordo com a seção *TRANS* do módulo *inverter* o valor da variável *output* será a negação de *input* ou permanecerá o mesmo, sendo esta escolha não determinística.

```
MODULE main
VAR
  gate1 : inverter(gate3.output);
  gate2 : inverter(gate1.output);
  gate3 : inverter(gate2.output);

MODULE inverter(input)
VAR
  output : boolean;
INIT
  output = FALSE
TRANS
  next(output) = !input | next(output) = output
```

O módulo *inverter* poderia ser escrito da forma apresentada abaixo, criando dois *defines* que possuem as expressões que estavam na seção *TRANS*. Estes dois novos *defines* – *changeOutputValue* e *unchanged* – são utilizados na seção *TRANS*. Desta forma o sentido semântico das transições deste módulo continua o mesmo, pois ao executar tais *defines* as expressões que eles contêm serão avaliadas.

```
MODULE inverter(input)
VAR
  output : boolean;
INIT
  output = FALSE
TRANS
  changeOutputValue | unchanged
DEFINE
  changeOutputValue := (next(output) = !input);
  unchanged := (next(output) = output);
```

3.2 Ferramenta NuSMV

A primeira versão do NuSMV foi atualizado em três dimensões, em relação à ferramenta SMV. Da perspectiva de funcionalidades do sistema, o NuSMV possui algumas funcionalidades a mais que aumentam a habilidade do usuário interagir com a ferramenta, além de prover heurísticas adicionais, como por exemplo, para alcançar a eficiência ou parcialmente controlar a explosão de estados. A arquitetura do NuSMV é altamente modular e aberta, permitindo não somente a adição, substituição e edição de módulos como também a ordenação de execução de alguns dos módulos do sistema. A terceira dimensão das atualizações é a qualidade do código, que além de ter aumentado existe uma documentação bem robusta, sendo considerado, por seus criadores, relativamente fácil de modificar [10].

A segunda versão do NuSMV contém adicionalmente a integração de técnicas proposicionais de satisfatibilidade (SAT) [9]. BDD e SAT geralmente solucionam diferentes categorias de problemas de verificação de modelos, tornando assim o NuSMV uma ferramenta eficiente, pois possuem técnicas complementares. O NuSMV permite a representação de modelos síncronos e assíncronos através de um conjunto de estados finitos, bem como suporta análise de propriedades expressas em CTL e LTL. Além disto, a licença da ferramenta foi modificada para ser OpenSource [7], permitindo que qualquer indivíduo possa utilizar livremente a ferramenta e participar no desenvolvido dela [20].

4 COMPUTATION TREE LOGIC (CTL)

CTL é uma lógica temporal que considera o conjunto de computações como uma estrutura ramificada, sendo o futuro não determinístico, a qual é suficientemente expressiva para formular um conjunto importante de propriedades de um sistema. Cada uma destas ramificações representa uma das diversas possibilidades de estados futuros que existem a partir de um determinado estado do sistema [2]. Esta lógica foi originalmente definida por Clarke e Emerson [11] e por Queille e Sifakis [25] para realizar verificação de modelos.

Uma fórmula CTL é composta por proposições atômicas, operadores booleanos e operadores específicos de CTL. As proposições atômicas são utilizadas para formalizar características do modelo e intuitivamente expressam fatos simples sobre o estado do sistema avaliado. Há dois tipos de operadores CTL: os que quantificam sobre os as ramificações da estrutura da computação e os que realizam avaliações temporais. Abaixo é possível verificar todos os operadores CTL:

- Operadores quantificadores:
    - O operador ∀ significa “para todos os caminhos” (representado por “A”);
    - O operador ∃ significa “existe pelo menos um caminho” (representado por “E”);
- Operadores Temporais:
    - O operador □ significa “globalmente” (representado por “G”);
    - O operador U significa “até que” (representado por “U”);
    - O operador ◊ significa “em um estado futuro” (representado por “F”);
    - O operador ◯ significa “no próximo estado” (representado por “X”);

A sintaxe de uma fórmula CTL é apresentada abaixo. Para mais informações sobre a sintaxe e sua semântica é possível verificar em [2].

$$\phi ::= \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \mid \exists \varphi \mid \forall \varphi$$
$$\varphi ::= \bigcirc \phi \mid \phi_1 \, U \, \phi_2$$

4.1 Propriedades CTL sobre modelos SAN e suas traduções

As propriedades CTL sobre um modelo SAN seguem a estrutura de uma propriedade CTL explicada anteriormente, com a única ressalva de que as proposições atômicas são descritas em SAN. Tais proposições atômicas descritas em SAN seguem as regras de [3] anteriormente apresentadas. O mesmo ocorre para propriedades CTL sobre um modelo aceito pelo NuSMV, conforme é possível verificar em [7], onde as proposições atômicas são, naturalmente, descritas na linguagem aceita pelo NuSMV.

A fim de verificar uma propriedade CTL com o NuSMV em um modelo traduzido é necessário seguir os seguintes passos:

- Criar a propriedade CTL escrevendo as proposições atômicas em SAN;
- Traduzir manualmente as proposições atômicas utilizando o mapeamento descrito na Seção 6;
- Adicionar uma seção *CTLSPEC* – para cada propriedade – no final do modelo traduzido juntamente com a propriedade CTL traduzida;

Executar o comando *check_ctlspec* do NuSMV, após carregar modelo no NuSMV;

5 TRABALHOS RELACIONADOS

Diversos formalismos para avaliação de desempenho contam com a possibilidade de *Model Checking*, como é o caso de Redes de Petri (RDP), Álgebra de Processos, e Cadeias de Markov. Até o momento, além deste trabalho, apenas uma abordagem para verificação de Redes de Autômatos Estocásticos foi proposta [12], apresentando a primeira versão de um verificador de modelos construído para SAN que faz uso de técnicas de verificação simbólica para propriedades escritas em CTL. Contudo, trata-se da primeira versão de um verificador, faltando a geração de contraexemplos.

A vantagem de utilizar o tradutor de SAN para a linguagem de entrada do NuSMV proposto por este trabalho, frente a um novo verificador próprio para SAN [12], é que parte-se da qualidade e do desempenho de duas décadas de experiências com este ambiente [19] mantido por um grupo de universidades [21].

Diversos trabalhos realizaram o mapeamento e tradução de uma determinada linguagem para a linguagem do NuSMV, dentre eles estão [1, 4, 16, 18].

5.1 Redes de Petri e Diagramas de Atividades em uma especificação de controle lógico

Em [16], Grobelna et. al. propõem um novo método de transformação para sistemas baseados em eventos. Nele é apresentando um mapeamento de Diagramas de Atividades (DA) da UML 2.0 para modelos de RDP, bem como um mapeamento das RDP para a linguagem do NuSMV a fim de realizar análises qualitativas.

O mapeamento entre DA e RDP foi realizado da seguinte forma: nodos de ação dos DA são transições das RDP; os nodos de bifurcações e junções dos DA são transições de bifurcações e junções das RDP; condições de entrada para ações dos DA são transições das RDP com a condição de disparo apropriada. Além disto, nodos de sincronização são necessários nas RDP visto que os DA forçam a sincronização através do nodo de junção.

Depois de aplicar o mapeamento anteriormente descrito, a RDP equivalente ao DA é criada. De acorco com os autores, a RDP resultante possui nodos desnecessários os quais podem ser eliminados aplicando um método para redução

de nodos dispensáveis. No exemplo apresentado em [16], ao utilizar esta técnica de redução foi possível reduzir em aproximadamente 40% do tamanho da RDP resultante. A RDP resultante é apresentada através da abstração RTL (*Register Transfer Level*), descrita em termos do fluxo de sinais, de tal forma que é fácil sintetizar, de acordo com os autores, como um controlador lógico reconfigurável sem modificações adicionais.

Além dos mapeamentos previamente citados, o mapeamento da RDP resultate para a linguagem do NuSMV é brevemente descrito fazendo referência a partes de um modelo do NuSMV equivalente a RDP resultante.

Todos estes mapeamentos estão explicados em nível conceitual em [16], pois ainda não desenvolveram um tradutor que faça tais mapeamentos automaticamente, sendo isto considerado como um trabalho futuro.

O modelo descrito na linguagem do NuSMV, resultante do mapeamento das RDP apresentado em [16], possui uma diferença notória com relação a esta dissertação que é a descrição de todo o espaço de estados atingíveis em apenas uma única variável do tipo *enum*, utilizando apenas o módulo *MAIN*. Isto ocasiona na falta de escalabilidade desta abordagem, tornando-se cada vez mais inviável a descrição do modelo ao passo que o modelo aumenta. Conforme é explicado no Capítulo 6, esta dissertação encapsula um conjunto de estados dentro de outros módulos, não especificando explicitamente todo o espaço de estados atingíveis que o modelo possui em uma variável, deixando este cálculo para o NuSMV realizar em tempo de execução. A abordagem utilizada neste trabalho, portanto, possui uma melhor escalabilidade.

Além desta diferença, há diferenças sintáticas tais como: a utilização da seção *ASSIGN* em [16] – em vez de usar as seções *INIT* e *TRANS* – para inicializar as variáveis e determinar as transições do modelo; a falta da utilização de seções *DEFINE*, contribuindo com o crescimento exponencial do tamanho da seção *ASSIGN*, de acordo com o tamanho do modelo, corroborando com o aumento da dificuldade de compreensão e descrição do modelo – isto está intimamente ligado à decisão de utilizar uma única variável para determinar todos os estados atingíveis do modelo; a utilização da seçao *LTLSPEC* no em vez da CTLSPEC – tal diferença apenas diferencia o tipo de lógica temporal utilizada a fim de realizar as análises qualitativas.

Enfim, através dos conceitos de mapeamento apresentados em [16] é possível modelar um determinado sistema em mais alto nível, através de DA, ou especificá-lo diretamente em RDP, a fim de realizar análises qualitativas sobre o modelo mapeado para a linguagem do NuSMV.

5.2 Model Checking em Redes de Petri Temporais utilizando o NuSMV

Em [4], Bobbio et. al. apresentam uma técnica para verificar se as Redes de Petri Temporais (RDPT) satisfazem propriedades especificadas em *Real Time Computational Tree Logic* (RTCTL), através de um mapeamento para a linguagem do NuSMV.

As RDPT são utilizadas para especificar sistemas onde o passar do tempo é um parâmetro crítico que pode afetar o comportamento do sistema. O grafo de transições das RDPT é construído de forma composicional baseado na discretização dos intervalos de disparo através da álgebra de Kronecker. Através desta descrição composicional, o modelo na linguagem do NuSMV pode ser automaticamente traduzido. Em [4] não apresentam nenhum tradutor que realize esta tradução de forma automática, apenas descrevem o mapeamento enfatizando que ele pode ser aplicado de forma a obter a tradução automaticamente.

O mapeamento entre RDPT e a linguagem do NuSMV foi descrito da seguinte forma: cada transição da RDTP é uma variável possuindo um valor numérico, que é mapeado para uma informação da transição (habilitada, desabilitada, tem memória e não é o próximo candidato, etc); uma variável numérica *em* é criada para indicar em qual nodo o marcador está.

O modelo descrito na linguagem do NuSMV, resultante do mapeamento das RDPT apresentado em [4], é similar ao apresentado neste trabalho com apenas duas pequenas diferenças de sintaxe: todo o modelo é descrito em um módulo secundário, permanecendo no módulo *MAIN* apenas a criação de uma instância do módulo secundário; a outra diferença é a não utilização de *DEFINE*, sendo especificado todas as transições diretamente na seção *TRANS*.

Em [4], da mesma forma que esta dissertação, não descreve todo o espaço de estados explicitamente da forma como em [16] (anteriormente explicado) realiza. Consequentemente, e diferentemente do apresentado por Grobelna et. al. [16], a seção que descreve as transições do modelo (em [4] é realizado na *TRANS*) não

cresce de forma exponencial. Isso traz benefícios a escalabilidade para modelagem de modelos maiores. Outro aspecto em comum em [4] com o esta dissertação é a utilização da seção *VAR* e *INIT* para criação e inicialização das variáveis do modelo, respectivamente.

5.3 AsmetaSMV

Diferentemente dos trabalhos anteriormente citados – que abordam formalismos de avaliação de desempenho de sistemas, tais como RDP e RDPT – em [1], Arcaini et. al. tratam de um método formal (*Abstract State Machine - ASM*) que é utilizado no desenvolvimento de sistemas complexos sem falhas, sendo aplicado desde a fase de criação dos requisitos até o seu desenvolvimento.

Neste trabalho é apresentada uma nova ferramenta, o AsmetaSMV, que agrega no conjunto de ferrametas ASMETA (*ASM mETAmodeling*). O AsmetaSMV apresenta um novo mapeamento de modelos ASM para a linguagem do NuSMV.

Conforme Arcaini et. al., diversas tentativas foram realizadas a fim de traduzir modelos ASM para a linguagem de um *model checker* existente, como SPIN e NuSMV. Outras abordagens para realizar a verificação de modelos ASM foram desenvolvidas como a criação do algoritmo AsmL – que realiza a verificação dos modelos ASM nativamente sem a necessidade de tradução para outra linguagem, mas de forma ineficiente e incapaz de lidar com modelos complexos – bem como a utilização mista do *model checker* [mc] square com o simulador CoreASM (pertencente ao conjunto ASMETA) utilizado para criar o espaço de estados dos modelos ASM.

A principal motivação da criação do AsmetaSMV é originado pela impossibilidade de utilização das abordagens de tradução de ASM para a linguagem do NuSMV bem como pelo término de suporte para tal ferramenta.

O modelo criado pelo AsmetaSMV é similiar ao apresentado neste trabalho, com as seguintes diferenças sintáticas: todo o modelo é descrito no módulo *MAIN*, não havendo módulos secundários; a inicialização de variáveis e as transições do modelo são especificadas dentro da seção *ASSIGN* em vez de utilizar a seção *INIT* e *TRANS*, respectivamente. Entretanto o modelo gerado pelo AsmetaSMV também possui seções *DEFINE*, bem como não especifica todo o espaço de estados atingíveis explicitamente.

O AsmetaSMV é uma ferramenta que prove uma camada de abstração ao utilizar o NuSMV de tal forma que o usuário não necessita ter conhecimento da sintaxe NuSMV e do modelo resultante do AsmetaSMV. Desta forma o usuário cria um modelo em ASM, especifica suas propriedades em CTL e manda o AsmetaSMV realizar a verificação de modelos. Ele, por sua vez, traduz o modelo ASM e interage com o NuSMV realizando a verificação do modelo. Caso algum contra exemplo seja gerado pelo NuSMV, o AsmetaSMV traduz o contra exemplo para utilizar as variáveis e nomenclaturas do modelo ASM e então informa o usuário o contra exemplo.

Esta camada de abstração não foi desenvolvida neste trabalho, sendo de responsabilidade do usuário do tradutor de SAN conhecer a sintaxe do NuSMV a fim de especificar as propriedades em CTL e então utilizar o NuSMV a fim de realizar a verificação de modelo. Esta camada pode ser desenvolvida como um trabalho futuro.

Portanto, Arcaini et. al. [1] criam o AsmetaSMV, uma nova ferramenta pertencente ao conjunto ASMETA a fim de prover uma forma de realizar verificação de modelos ASM, através do NuSMV.

5.4 S2N

Em [18], Jiang et. al. apresentam a ferramenta S2N que realiza a tradução de modelos descritos em Promela, linguagem de entrada do *model checker* SPIN, para a linguagem do NuSMV. Segundo os autores, o SPIN e o NuSMV, são os *model checker* mais convencionais existentes. Entretanto o SPIN suporta apenas verificação de modelos através de LTL, já o NuSMV suporta verificação de propriedades descritas em CTL como também em LTL.

O mapeamento abrange a maior parte das funcionalidades de Promela, entretanto é traduzido um *proctotype* por vez, portanto a complexidade é linear ao número de *proctotypes* que o modelo possuir. O modelo resultando na linguagem do NuSMV é composto de módulos secundários onde cada um possui as seções *VAR, ASSIGN* e *TRANS*. No módulo *MAIN* são criadas as variáveis globais e as locais ficam dentro dos módulos secundários. A seção *ASSIGN* contém a inicialização das variáveis bem como a suas transições. Já a seção *TRANS* possui regras de restrições sobre o agendamento dos processos e com seus contadores. A seção

DEFINE não é utilizada no mapeamento de modelos Promela. Da mesma forma que esta dissertação, em [18] não é especificado todo o espaço de estados atingíveis explicitamente, deixando este cálculo para o NuSMV.

É possível, portanto, verificar com propriedades CTL um modelo descrito em Promela, após traduzí-lo para a linguagem do NuSMV através do S2N. Além disto, segundo Jiang et. al., por Promela ser uma linguagem de mais alto nível em relação à linguagem do NuSMV, pode ser mais difícil especificar um modelo diretamente na linguagem do NuSMV. Neste caso a utilização do S2N pode ser interessante ao modelador, pois se pode especificar primeiramente o modelo em Promela e então traduzí-lo.

6 TRADUÇÃO DE MODELOS SAN PARA MODELOS DO NUSMV

Na primeira parte desta seção são comparados conceitos análogos de SAN e NuSMV, indicando a estratégia da tradução, na segunda parte é apresentada e discutida a estrutura geral de um modelo traduzido, na terceira parte é detalhado e exemplificado cada uma das equivalências, na quarta parte é apresentado a tradução das operações SAN, na quinta parte é explicado o algoritmo de tradução de expressões SAN e por último é apresentado o processo que foi utilizado para o desenvolvimento do tradutor bem como suas limitações.

6.1 Conceitos Análogos

Antes de detalhar o mapeamento feito entre SAN e a linguagem do NuSMV é necessário comparar os principais conceitos das linguagens. A Tabela 2 apresenta as principais equivalências que as linguagens possuem.

Tabela 2: As principais equivalências entre SAN e a linguagem do NuSMV

<table>
<tr><th>Modelos</th><th>SAN</th><th>NuSMV</th></tr>
<tr><td>São compostos de</td><td>Autômatos</td><td>Módulos</td></tr>
<tr><td>São representados por</td><td colspan="2">Finite State Machine (FSM)</td></tr>
<tr><td rowspan="9">Conceitos equivalentes</td><td>função de atingibilidade</td><td>estados iniciais</td></tr>
<tr><td>estados dos autômatos</td><td>variável state dos módulos</td></tr>
<tr><td>transições nas definições de autômatos</td><td>seção TRANS</td></tr>
<tr><td>Eventos</td><td rowspan="2">macro em seção DEFINE</td></tr>
<tr><td>Identificadores</td></tr>
<tr><td colspan="2">operadores matemáticos e lógicos são os mesmos
(exceto a precedência dos operadores)</td></tr>
<tr><td>operador ST</td><td>acesso da variável state do módulo</td></tr>
<tr><td>operador NB</td><td>expressão booleana que verifica a quantidade de módulos em determinado estado</td></tr>
<tr><td>operador RW</td><td>expressão booleana que verifica o valor de reward associado com o estado atual do módulo</td></tr>
</table>

Ambas as linguagens servem para a construção de modelos com um conjunto finito de estados. Também ambas linguagens possuem abstrações para

estruturar modelos, seja como autômato em SAN ou módulo na linguagem do NuSMV. Outro aspecto que vale ressaltar é a possibilidade de representar explicitamente transições de estado. Em SAN cada autômato tem transições e em alguns casos uma transição do modelo é dada pela composição de transições de autômatos que sincronizam no mesmo evento. No NuSMV transições entre estados anterior e posterior podem ser explicitamente declaradas considerando estados de diferentes módulos. O conceito de estados atingíveis é bastante peculiar de SAN, não se encontrando análogo em diversas outras linguagens. SAN também permite a definição de um subconjunto de estados atingíveis (atingibilidade parcial) [3]. O conceito mais próximo no NuSMV é o de estados iniciais. Considerando-se estados iniciais como atingíveis, se uma declaração de estados iniciais do NuSMV especificar o mesmo conjunto de estados da declaração de atingibilidade parcial de SAN, então este conceito pode ser mapeado. Como será visto em mais detalhe, é possível mapear uma declaração de atingibilidade em uma declaração de estados iniciais no NuSMV.

SAN possui o conceito de autômatos e cada um deles tem um conjunto de estados. A estrutura considerada análoga em NuSMV é a do módulo. Assim, cada autômato é representado em um módulo. Cada módulo tem uma variável de estado. O intervalo de valores possível desta variável de estado cobre exatamente os estados possíveis do autômato.

A mudança de estados em SAN pode ser local, afetando apenas um autômato, ou sincronizante, afetando mais de um autômato. Para representar estas possibilidades, especialmente a mudança simultânea de estado em mais de um autômato (módulo), optou-se pelo uso da seção *TRANS*. Na seção *TRANS* especifica-se logicamente estados atuais e próximos estados possíveis, levando em consideração os eventos, suas probabilidades e taxas, sejam estas funcionais ou não.

Para eventos locais deriva-se as possíveis mudanças de estado de um autômato de forma direta a partir da estrutura do autômato. Dado um estado atual (verdade) e um evento habilitado (verdade), existe um próximo estado definido. O não determinismo do NuSMV auxilia no caso onde um evento pode levar a mais de um estado, assim como no caso de mais de um evento habilitado.

Para eventos sincronizantes cuida-se para que todos autômatos que tenham um dado evento sincronizante habilitado em sua estrutura, que o evento

sincronizante realize uma mudança de estado em cada um destes autômatos. Em um dado autômato, um evento pode gerar diferentes mudanças de estado. Assim, um evento sincronizante no NuSMV, utilizando a seção *TRANS*, se traduz em uma conjunção entre as possibilidades de mudanças de estado de cada autômato que usa o evento considerado, onde cada possibilidade em cada autômato é dada por uma disjunção de condições que representam ocorrência de um evento em um autômato.

Existem três tipos de operadores, que compõem expressões, em SAN: lógicos, matemáticos e operadores da linguagem SAN. Os mesmos operadores lógicos e matemáticos existem na linguagem do NuSMV. Já os operadores específicos da linguagem SAN, apesar de não existirem na linguagem do NuSMV, podem ser mapeados facilmente através da construção de macros que avaliam o estado da rede. Os operadores SAN mapeados são: *st*, *nb* e *rw*. O operador *st* é diretamente representado pelo acesso do valor da variável de estado de uma instancia de um módulo.

O operador *nb* é representado por uma expressão que verifica o estado atual de todos os módulos do modelo, somando em uma unidade no valor que irá retornar para cada módulo em que se encontrar no estado determinado, ou seja, é uma soma de condições, havendo uma condição para cada módulo do modelo, onde cada condição retorna o valor numérico “1” quando o valor do estado atual do módulo for o valor verificado.

O operador *rw* é representado por uma expressão que verifica o estado atual do módulo e retorna o valor de *reward* associado ao estado atual. O mapeamento do valor de *reward* é retirado da estrutura do autômato e caso não exista um valor associado é atribuído um valor numérico sequencial para cada estado que não possua o *reward* conforme a definição de SAN segundo a página 6 do manual do PEPS 2003 [3].

Em SAN, diferentemente da linguagem do NuSMV, os operadores lógicos e matemáticos aceitam tanto valores numéricos quanto valores booleanos como argumentos. Além disto, SAN possui uma peculiaridade: a precedência implícita de uma expressão é sempre da esquerda para a direita, diferentemente da precedência comum matemática e lógica. Para especificar em SAN qualquer outra precedência diferente da esquerda para a direita é necessário o uso dos parênteses. Desta

forma, foi necessário realizar um algoritmo de tradução de expressões SAN para a linguagem do NuSMV, o qual será explicado posteriormente.

6.2 Estrutura geral do código gerado pelo tradutor para o NuSMV

O código gerado pelo tradutor para o NuSMV tem a seguinte estrutura:

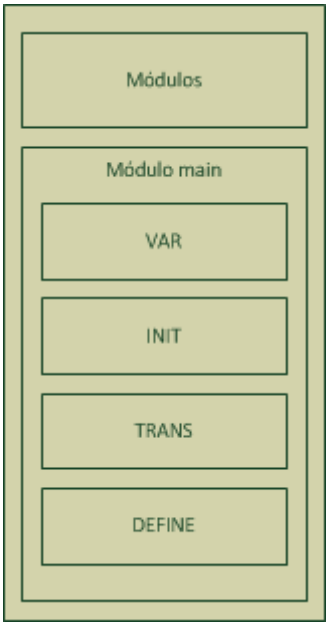


Figura 2: Estrutura geral do código gerado pelo tradutor para o NuSMV

A primeira parte do código é constituída por módulos, respectivos a cada autômato. Na segunda parte do código, o módulo *main* é definido, sendo composto por quatro seções: *VAR, INIT, TRANS* e *DEFINE*.

A primeira seção do módulo *main*, a *VAR*, é constituída por uma instância para cada módulo existente diferente do módulo *main.* Cada uma destas instâncias é guardada por uma variável, sendo que os nomes destas variáveis seguem uma regra de formação que utiliza o do autômato correspondente.

Na seção *INIT* é descrita uma expressão booleana que determina um estado inicial ou um conjunto de estados iniciais, mapeados diretamente da tradução da expressão do *partial reachability* ou do *reachability*.

A terceira seção do módulo *main*, a *TRANS*, é composta por uma disjunção de *defines* (entradas na seção *define*) que determinam as transições locais e sincronizantes do modelo. Desta forma, a cada mudança de estado acontece uma

transição local em um autômato ou uma composição de transições locais de autômatos, rotuladas com mesmo evento sincronizante, que representam apenas uma transição no estado global do modelo.

A última seção, a *DEFINE*, é composta por diversos *defines*, sendo que cada um deles possui um nome e uma expressão. Um *define* serve como uma macro de programação funcionando pela simples busca e substituição textual do seu conteúdo, que no caso do *define* é uma expressão. Portanto, se uma determinada expressão é utilizada mais de uma vez no modelo, pode ser descrita em um *define* a fim de ser reutilizada quantas vezes forem necessárias, bastando utilizar o nome do *define*. Visto que identificadores e eventos são conceitos de SAN que também são utilizados em mais de um lugar no modelo, tais conceitos são mapeados para o conceito de *define*. A explicação e exemplificação do mapeamento de identificadores e eventos para o conceito de *define* são apresentadas posteriormente, bem como a estrutura dos *defines* que descrevem as transições locais e sincronizantes.

6.3 Detalhamento da tradução e exemplificação

O conjunto de modelos SAN possíveis é determinado pelas regras de sintaxe em [3] (páginas 5 a 14). Este trabalho não mapeou os seguintes conceitos de SAN, significando que um subconjunto de modelos SAN, os quais não utilizam tais conceitos, são aceitos para o processo de tradução:

- números reais;
- os operadores *min* e *max*;
- replicação e, consequentemente, os operadores SAN que utilizam o conceito de replicação – *nb [aut ... aut] state*, *sum_rw [aut ... aut]*, *sum_rw [aut ... aut] state*, *prod_rw [aut ... aut]* e *prod_rw [aut ... aut] state*.

O único tipo numérico que o NuSMV contempla é o inteiro, por isso os números reais não estão mapeados. Portanto é necessário ajustar os modelos SAN a fim de remover quaisquer valores numéricos que não sejam inteiros antes de utilizar o tradutor. Os demais conceitos não foram mapeados por questões temporais. Deve-se notar que esta limitação não invalida a demonstração de possibilidade de mapear modelos SAN para o NuSMV uma vez que todo modelo SAN com replicação pode ser convertido para um modelo SAN sem utilizar

estruturas de replicação. A gramática aceita pelo tradutor SAN para a linguagem do NuSMV encontra-se no Apêndice A.

A seguir é detalhada a tradução de um modelo SAN para outro aceito pelo NuSMV, seguindo a estrutura de um código gerado para o NuSMV apresentada na Figura 2. Todos os trechos de código SAN utilizados para a exemplificação abaixo são baseadas na tradução do modelo do jantar dos filósofos, retirado de [5] [24]. A maior parte dos trechos são retirados do modelo *phil04c.san* e alguns do modelo *phil04f.san*. Ambos os modelos descrevem o jantar dos filósofos – modelados com quatro filósofos, entretanto o primeiro utiliza eventos sincronizantes e o segundo possui eventos com taxas funcionais. Utilizando estes dois modelos como base para detalhamento da tradução e exemplificação, é possível cobrir todos os conceitos mapeados de SAN para a linguagem do NuSMV.

Cada autômato é representado por um módulo. Este módulo possui a variável de estado *state* do tipo enumerador, onde são especificados todos os estados que o autômato que originou o módulo pode assumir. Cada valor do enumerador *state* – que representa o nome de um estado – é constituído do prefixo "*s_*" seguido do nome do estado. O nome do módulo possui o prefixo "*ad_*" – *automata definition* – seguido do nome do autômato. Além desta variável, o módulo é composto por um *define* chamado *unchanged* que retorna um valor booleano indicando se o estado atual do módulo é igual ao próximo estado do módulo na próxima transição de estado global do modelo. Este *define* é amplamente utilizado nos *defines* do módulo *main* para determinar as transições locais e sincronizantes, explicadas nesta seção posteriormente.

Por exemplo, tome-se a definição SAN abaixo:

```
aut P0
    stt Thinking to (Right)    t_r_0
                to (Thinking) r_l_1 l_r_3
    stt Right    to (Left)     r_l_0
                to (Right)    l_r_3
    stt Left     to (Thinking) l_t_0
```

O autômato acima P0 resulta no módulo P0 abaixo:

```
MODULE ad_P0()
VAR state : {s_Thinking, s_Right, s_Left};
DEFINE
unchanged := state = next(state);
```

A seção *VAR* do módulo *main*, possui uma instância para cada módulo anteriormente definido.

Desta forma, considerando existentes os módulos ad_P0, ad_P1, ad_P2 e ad_P3, esta seção resulta no exemplo abaixo, onde as instâncias destes módulos são a_P0, a_P1, a_P2 e a_P3 respectivamente:

```
MODULE main
VAR
  a_P0 : ad_P0();
  a_P1 : ad_P1();
  a_P2 : ad_P2();
  a_P3 : ad_P3();
```

A segunda seção do módulo *main*, a *INIT*, possui uma expressão booleana que determina um estado inicial ou um conjunto de estados iniciais, mapeados diretamente da tradução da expressão do *partial reachability* ou do *reachability*. Caso haja mais de um estado inicial válido o NuSMV escolherá, não deterministicamente um destes estados iniciais.

```
partial reachability = ((nb Thinking) == P);
```

Considerando a expressão *reachability* acima, onde *P* é um identificador com um valor numérico, a seção *INIT* resultante se encontra abaixo. Neste exemplo, esta expressão na seção *INIT* significa que qualquer estado global inicial do modelo que possuir o número *P* de autômatos no estado *Thinking* é um estado inicial válido.

```
INIT ( nb_Thinking = P )
```

A tradução e exemplificação dos operadores SAN que este trabalho aborda – *st*, *nb* e *rw* – serão detalhados posteriormente.

A seção *TRANS* é descrita pela composição da disjunção de *defines* que determinam as transições locais – um *define* para cada módulo que possuir pelo menos uma transição local – e sincronizantes – caso o modelo tenha pelo menos uma transição sincronizante.

```
TRANS
  next_local_step_ad_P0
  |  next_local_step_ad_P1
  |  next_local_step_ad_P2
  |  next_local_step_ad_P3
  |  synchronized_steps
```

Neste exemplo a seção *TRANS* utiliza cinco *defines*: quatro referentes aos módulos determinando transições locais e um *define* referente a todas as transições sincronizantes do modelo. Como cada um destes *defines* são mutuamente exclusivos, como veremos adiante, e estão ligados por uma disjunção, consequentemente apenas uma destas transições ocorrerá em cada mudança de estados do modelo.

Note-se que na linguagem do NuSMV as possíveis transições de estado são definidas como relações entre estado atual e próximo estado. Assim, como em um dado momento somente uma transição pode acontecer, segundo a semântica de SAN, então esta relação que leva em conta estado atual e estado posterior tem somente uma transição aplicável. Isto não significa que o não determinismo dos modelos SAN não está representado, pois tomando em consideração somente o estado atual, sem considerar o próximo estado, a tradução para a linguagem do NuSMV cobre todas possíveis transições para próximos estados desdobrando cada um dos casos possíveis.

A explicação da tradução das operações SAN e do algoritmo de tradução de expressões SAN serão explicados nas Seções 4.4 e 4.5 respectivamente.

Na seção *DEFINE* são representados os conceitos de: identificadores, eventos, transições locais e sincronizantes.

Cada identificador do modelo SAN possui um nome e uma expressão associada. Cada identificador é mapeado em um *define*, o qual também possui o mesmo nome, acompanhado do prefixo "*i_*", e a expressão traduzida para a linguagem do NuSMV. As expressões dos identificadores são sempre traduzidas para retornarem um valor numérico, a fim de manterem um padrão entre os *defines* que mapeiam os identificadores.

```
lambda = 1;
Thinking_to_Right_0 = (st P1 != Left) * lambda;
```

Os dois identificadores acima são traduzidos nos seguintes *defines*, ambos retornando valores numéricos:

```
i_lambda := ( 1 );
i_Thinking_to_Right_0 := ( ( ( ( v_P1.state != Left ) ) ? 1 : 0 ) * lambda );
```

Os eventos do modelo SAN possuem um nome, uma taxa, e um tipo – local ou sincronizante – e são representados por *defines*. O nome do *define* é o mesmo utilizado pelo evento acompanhado do prefixo "*e_*" e a expressão deste *define* é sempre estruturada de uma verificação – se a taxa do evento é superior a zero, indiferentemente se a sua taxa é uma valor numérico ou um identificador – ou seja, sempre irá retornar um valor booleano. Caso a taxa for um identificador, ele será, como explicado anteriormente, traduzido em um *define* que sempre retorna um valor numérico, desta forma é possível estruturar desta maneira a tradução de todos os eventos, não existindo construção de expressões inválidas na linguagem do NuSMV.

```
loc l_t_0 (mu);
loc t_r_0 (Thinking_to_Right_0);
syn some_event_with_numeric_rate (35);
```

Os eventos acima são traduzidos nos seguintes *defines*:

```
e_l_t_0 := ( mu > 0 );
e_t_r_0 := ( Thinking_to_Right_0 > 0 );
e_some_event_with_numeric_rate := ( 35 > 0 );
```

Para cada módulo que possuir pelo menos uma transição local existe um *define* que descreve todas as possíveis transições locais deste módulo. Este *define* possui o seu nome composto do prefixo "*local_step_*" concatenado ao nome do módulo no qual as transições locais estão sendo descritas.

No exemplo abaixo, é possível verificar apenas uma transição local do módulo *ad_P0* descrita no *define* chamado *local_step_ad_P0*, pois as outras transições deste módulo são sincronizantes. Caso houvesse mais de uma transição local para este módulo, esta conjunção apresentada abaixo, que descreve apenas uma transição local, estaria ligada através de uma disjunção a outras conjunções, que descreveriam as outras transições deste módulo.

```
local_step_ad_P0 := (
   -- connection s_Left - s_Thinking
   ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
);
```

Esta transição local é referente à mudança do estado s*_Left* para o s*_Thinking*. Esta transição está sendo ocasionada pelo evento local *l_t_0*. Conforme

explicado anteriormente, o *define* e*_l_t_0* que representa o evento *l_t_0* sempre retorna um valor booleano.

Este *define* chamado *local_step_ad_P0* determina apenas as transições locais do módulo *ad_P0* e não representa uma mudança de estados global do modelo. Por isso é criado outro *define*, para cada *define* que determina todos as transições locais de um determinado módulo. Este outro *define* é composto por uma conjunção entre os *defines unchanged* de todos os outros módulos do modelo e o *define* que determina todas as transições locais de um determinado módulo. Desta forma é possível descrever mudanças de estado globais e, consequentemente, utilizar tal *define* na seção *TRANS*.

O exemplo abaixo mostra o conteúdo do *define* chamado *next_local_step_ad_P0*, o qual é utilizado na seção *TRANS*, conforme exemplificado anteriormente. Ele consiste na conjunção do *define* chamado *local_step_ad_P0* com os *defines unchanged* de cada um dos outros módulos do modelo.

```
next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
);
```

Para todo modelo que possuir pelo menos um evento sincronizante, existe o *define* chamado *synchronized_steps* que possui a descrição de todas as transições ocasionadas por eventos sincronizantes do modelo. Tais transições são descritas através de uma conjunção entre as possibilidades de mudanças de estado de cada autômato usando o evento considerado, sendo estas uma disjunção de condições das aplicações do evento no autômato.

Considerando as transições do modelo *phil04c.san*, o exemplo abaixo mostra apenas uma porção do *define synchronized_steps* apresentando as possíveis transições resultantes obtidas somente a partir dos eventos sincronizantes *t_r_0* e *r_l_0*. Vale ressaltar que uma transição em SAN pode especificar o próximo estado igual ao estado anterior em um determinado autômato.

```
synchronized_steps := (
    (
        e_t_r_0
        &
        (
        -- connection s_Thinking - s_Thinking
```

```
        ( a_P1.state = s_Thinking & next(a_P1.state) = s_Thinking )
       -- connection s_Right - s_Right
       | ( a_P1.state = s_Right & next(a_P1.state) = s_Right )
       )
       &
       (
       -- connection s_Thinking - s_Right
        ( a_P0.state = s_Thinking & next(a_P0.state) = s_Right )
       )
       & a_P3.unchanged
       & a_P2.unchanged
   )
   |
   (
       e_r_l_0
       &
       (
       -- connection s_Thinking - s_Thinking
        ( a_P3.state = s_Thinking & next(a_P3.state) = s_Thinking )
       -- connection s_Left - s_Left
       | ( a_P3.state = s_Left & next(a_P3.state) = s_Left )
       )
       &
       (
       -- connection s_Right - s_Left
        ( a_P0.state = s_Right & next(a_P0.state) = s_Left )
       )
       & a_P2.unchanged
       & a_P1.unchanged
   )
   |

   ...

  );
```

No exemplo acima são apresentadas quatro possíveis mudanças de estados globais, sendo duas através do evento *e_t_r_0* e as outras pelo evento *e_r_l_0*. A primeira, ocasionada pelo evento *e_t_r_0*, a instância *a_P0* do módulo *ad_P0* transiciona do estado *s_Thinking* para *s_Right*, a instância *a_P1* do módulo *ad_P1* transiciona do estado *s_Thinking* para *s_Thinking* e as instâncias *a_P2* e *a_P3* permanecem no mesmo estado (utilização do *define unchanged*). A segunda, ocasionada pelo evento *e_t_r_0*, a instância *a_P0* transiciona do estado *s_Thinking* para *s_Right*, a instância *a_P1* transiciona do estado *s_Right* para *s_Right* e os as instâncias *a_P2* e *a_P3* permanecem no mesmo estado. A terceira, ocasionada pelo evento *s_r_l_0*, a instância *a_P0* transiciona do estado *s_Right* para *s_Left*, a instância *a_P3* transiciona do estado *s_Thinking* para *s_Thinking* e as instâncias *a_P1* e *a_P2* permanecem no mesmo estado. A quarta, ocasionada pelo evento *e_r_l_0*, a instância *a_P0* transiciona do estado *s_Right* para *s_Left*, a instância

a_P3 transiciona do estado *s_Left* para *s_Left* e as instâncias *a_P1* e *a_P2* permanecem no mesmo estado.

6.4 Tradução das operações SAN

As operações SAN que foram mapeadas e traduzidas à linguagem do NuSMV são *st*, *nb* e *rw*.

O operador *st* é diretamente representado pelo acesso do valor da variável de estado de uma instancia de um módulo.

```
st P1 != Left
```

Considerando a expressão SAN acima que verifica se o estado do autômato *P1* é diferente do estado *Left*, a expressão traduzida resultante é:

```
a_P1.state != s_Left
```

Esta expressão na linguagem do NuSMV acessa o valor da variável *state* da instância *a_P1*, que representa a instância do módulo *ad_P1*, e verifica se o seu valor é diferente de s_*Left*.

O operador *nb* é representado por uma expressão que verifica o estado atual de todos os módulos do modelo, somando em uma unidade no valor que irá retornar para cada módulo em que se encontrar no estado determinado.

```
nb Thinking
```

A expressão *nb* acima, que retorna um valor numérico referente à quantidade de autômatos que se encontram no estado *Thinking*, é traduzida na seguinte expressão, considerando que o modelo possui as seguintes instâncias *a_P0*, *a_P1*, *a_P2* e *a_P3*:

```
( a_P0.state = s_Thinking ? 1 : 0 ) + ( a_P1.state = s_Thinking ? 1 : 0 ) +
( a_P2.state = s_Thinking ? 1 : 0 ) + ( a_P3.state = s_Thinking ? 1 : 0 )
```

Esta representação é composta de verificações ternárias unidas pelo operador da soma. Desta forma é possível representar de forma correta o comportamento do operador *nb* de SAN.

O operador *rw* é representado por uma expressão que verifica o estado atual do módulo e retorna o valor de *reward* associado ao estado atual. O mapeamento do valor de *reward* é retirado da estrutura do autômato e caso não exista um valor associado é atribuído um valor numérico sequencial para cada estado que não possua o *reward*, conforme a definição de SAN segundo o manual do PEPS 2003 [3] apresentada na página 6.

A expressão que traduz o significado do operador *rw* é apresentada abaixo, considerando a aplicação do operador *rw* ao autômato P0 e que tal autômato possui os seguintes estados: *Thinking, Left* e *Right*.

```
(a_P0.state = s_Thinking ? 0 : (a_P0.state = s_Right ? 1 : (a_P0.state = s_Left ?
2 : -1)));
```

Essa expressão inicialmente verifica se o estado da instância *a_P0* é s_*Thinking*, e em caso positivo retorna o valor de *reward* 0; em caso negativo, ela realiza outra verificação: se o estado da instância *a_P0* é s_*Right*, e em caso positivo retorna o valor de *reward* 1; em caso negativo, ela realiza a terceira e última verificação: se o estado da instância *a_P0* é s_*Left*, e em caso positivo retorna o valor de *reward* 2; e em caso negativo retorna o valor -1, que não tem significado algum e nem é esperado que seja retornado tal valor, pois todos os estados do módulo ad_P0 foram considerados e verificados, portanto não existe a possibilidade de tal instância não estar em um dos estados verificados.

6.5 Algoritmo de tradução de expressões SAN

A flexibilidade que SAN permite descrever uma expressão não é diretamente mapeada para a linguagem do NuSMV. Tal flexibilidade permite que tanto os operadores matemáticos quanto os booleanos aceitem como argumentos valores numéricos e booleanos. A expressão abaixo exemplificada é válida segundo a linguagem SAN.

```
2 >= 3 + 4 && 7 > 6 == 8 - 1 / 9
```

A fim de representar uma expressão na linguagem do NuSMV que tenha o mesmo poder de expressão, como a descrita acima, foi necessário desenvolver um algoritmo que realizasse tal tradução. Além deste objetivo, o algoritmo de tradução

de expressões SAN para a linguagem do NuSMV também objetiva manter a precedência original da expressão SAN – visto que ambas as linguagens possuem conceitos de precedência diferentes.

Este algoritmo decompõe a expressão SAN de entrada, cria uma árvore binária que a representa e, a partir dela, retorna uma expressão que é compatível a linguagem do NuSMV e que representa o significado da expressão de entrada.

A árvore correspondente àquela expressão SAN exemplificada é apresentada abaixo. A tradução é realizada a partir das folhas de maior profundidade desta árvore, verificando sempre qual o tipo dos argumentos que o operador necessita. Caso o operador necessite de um argumento de um determinado tipo e pelo menos um deles for de outro, é realizada a conversão deste(s) argumento(s) para possuir o tipo que o operador requer. A conversão de um argumento booleano para numérico é realizada, dentro de uma expressão, através de uma operação ternária. Já a conversão inversa é realizada através de uma verificação – se o argumento é superior à zero.

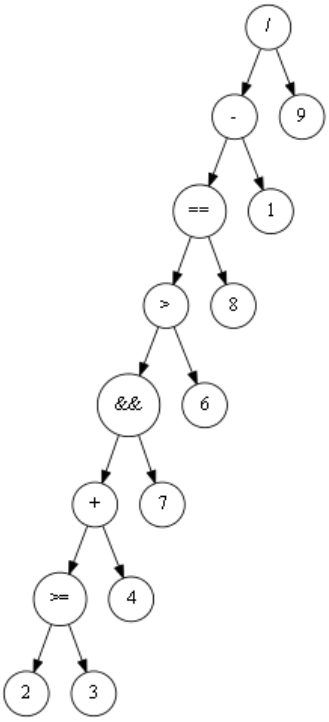


Figura 3: Árvore binária representando uma expressão válida segundo SAN

Por exemplo, o operador >= requer que ambos argumentos sejam do mesmo tipo, a fim de realizar a comparação. Neste exemplo, ambos os argumentos são numéricos – valores “2” e “3” – logo não há necessidade de conversão dos tipos dos argumentos.

Já no próximo caso o operador da soma necessita de dois argumentos numéricos, sendo um deles o resultado da operação >=, ou seja, um valor booleano, e o outro um valor numérico – valor “4”. Portanto, é necessário converter esta parte da expressão com o objetivo de manter o mesmo significado da expressão original e para a linguagem do NuSMV reconhecer como válida esta expressão. A tradução da sub expressão “2 >= 3 + 4” resulta na seguinte expressão “(2 >= 3 ? 1 : 0) + 4” aceita pelo NuSMV.

Continuando a traduzir esta árvore é verificado que o operador da conjunção lógica exige que seus dois parâmetros sejam booleanos. Entretanto ambos, neste exemplo, são numéricos, sendo necessário converter o tipo de ambos os argumentos para booleanos. A tradução da sub expressão “2 >= 3 + 4 && 7” resulta na seguinte expressão “(((2 >= 3 ? 1 : 0) + 4) > 0) & (7 > 0)” aceita pelo NuSMV.

Seguindo este algoritmo até realizar por completa a tradução, aquela expressão utilizada como exemplo será traduzida para a linguagem do NuSMV na seguinte expressão.

```
((((((((((2 >= 3)? 1: 0) + 4) > 0) & (7 > 0))? 1: 0) > 6)? 1: 0) = 8)? 1: 0) - 1 / 9
```

6.6 Desenvolvimento do tradutor

Para desenvolver o tradutor da linguagem SAN para a linguagem do NuSMV foi utilizado duas ferramentas: um analisador léxico e um analisador sintático. Considerando que a estrutura de dados interna do tratutor foi modelada para utilizar-se do paradigma de programação orientado a objetos, foram escolhidas as seguintes ferramentas para desenvolver o tradutor: JFlex [17] e o BYACC/J [6], sendo o analisador léxico e o sintático respectivamente, pois ambos geram código em Java. Através da definição das regras léxicas de SAN – utilizadas pelo JFlex – e da definição das regras sintáticas – utilizadas pelo BYACC/J – são geradas classes em Java as quais constituem o tradutor de SAN para a linguagem do NuSMV.

Conforme já citado na Seção 6.3, a gramática aceita pelo tradutor desenvolvido encontra-se no Apêndice A. A tradução de expressões CTL dentro de um modelo SAN não é comtemplado pelo tradutor, de forma que a fim de aplicar alguma propriedade CTL no modelo traduzido deve-se seguir o procedimento explicado na Seção 4.1.

Tabela 3: Conceitos SAN mapeados atualmente e para trabalhos futuros

Conceitos presentes	Conceitos para trabalhos futuros
st <aut_name>	nb [<aut_name> ... <aut_name>] <stt_name>
@ <aut_name>	sum_rw [<aut_name> ... <aut_name>]
nb <stt_name>	sum_rw [<aut_name> ... <aut_name>] <stt_name>
rw <aut_name>	prod_rw [<aut_name> ... <aut_name>]
<exp1> + <exp2>	prod_rw [<aut_name> ... <aut_name>] <stt_name>
<exp1> - <exp2>	min (<exp1> , <exp2>)
<exp1> * <exp2>	max (<exp1> , <exp2>)
<exp1> / <exp2>	<exp1> ^ <exp2>
<exp1> == <exp2>	
<exp1> != <exp2>	
<exp1> < <exp2>	
<exp1> <= <exp2>	
<exp1> > <exp2>	
<exp1> >= <exp2>	
! <exp1>	
<exp1> || <exp2>	
<exp1> && <exp2>	

Considerando os conceitos apresentados como na Tabela 2.1 em [3], a Tabela 3 desta dissertação explicita os conceitos presentes no tradutor atual e os conceitos os quais serão mapeados em trabalhos futuros. Em [3] é possível encontrar a explicação detalhada de cada um destes conceitos SAN bem como exemplos de sua utilização.

A estrutura de dados utilizada pelo tradutor para carregar um modelo SAN e, a partir dela, gerar um modelo na linguagem do NuSMV é apresentada na Firgura 4.

O tradutor inicialmente lê linha por linha do modelo textual SAN, extrai a informação relativa àquela linha e a guarda na estrutura de dados. Após ler a última linha do modelo textual SAN, a estrutura de dados está completamente carregada. Então o tradutor inicia o processo de criação do modelo textual para o NuSMV sempre consultando a estrutura de dados que representa o modelo SAN. O modelo de saída do tradutor é gerado linha por linha de forma sequencial iniciando pela seção dos módulos e finalizando pela seção dos *DEFINEs*, conforme a Figura 2 da Seção 6.2 apresenta.

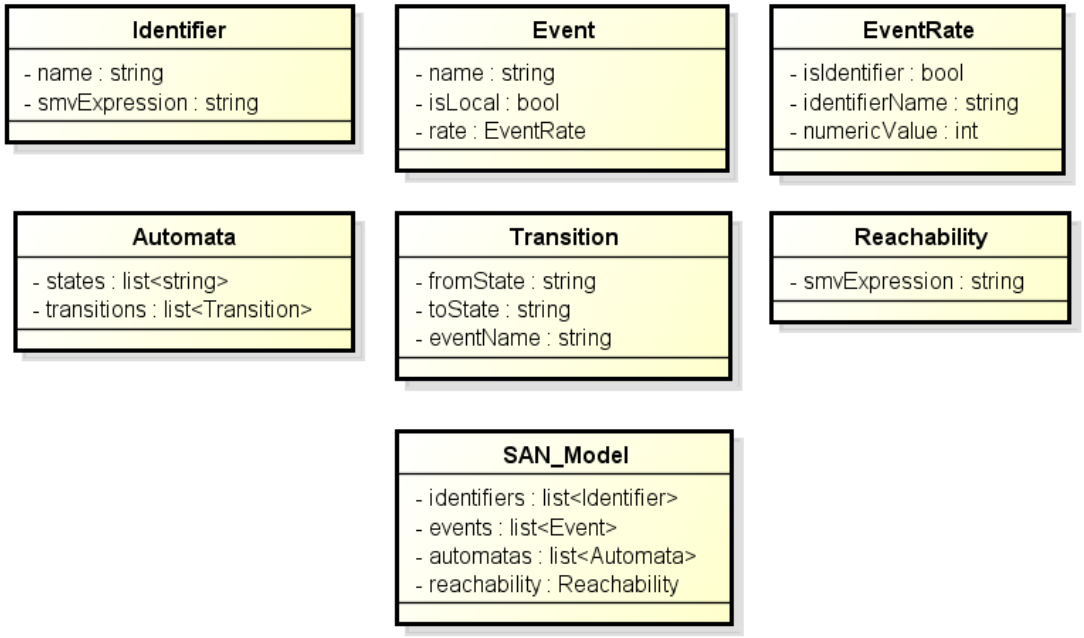


Figura 4: Representação dos dados de um modelo SAN

7 DISCUSSÃO DA CORRETUDE

A argumentação de corretude da tradução acontece idealmente através de uma prova de que esta tradução está correta. Para um exemplo de complexidade de tal prova, tome-se [26] onde se demonstra que uma tradução de Gramática de Grafos para PROMELA é correta. Devido a tal complexidade, uma prova de corretude da tradução não faz parte do escopo deste trabalho e se constitui trabalho futuro.

Entretanto, este trabalho reúne fortes indicativos da corretude da tradução proposta. Estes indicativos se dão com base na experimentação de traduções de dezessete modelos. Para todas estas traduções é observado que o sistema de transição de estados definido pelo modelo SAN (a Cadeia de Markov equivalente, desprezando-se informação quantitativa) tem a mesma topologia do sistema de transição de estados gerado pelo modelo NuSMV. A seguir, detalham-se estes experimentos.

Foram traduzidos quatro tipos de modelos: o jantar dos filósofos [5] [24], o protocolo de rede de sensor sem fio [13] [24], linha de produção [23] e o primeiro servidor disponível [5] [24]. A partir destes quatro tipos foram derivados dezessete modelos SAN os quais foram todos traduzidos para modelos aceitos pelo NuSMV. Estes quatro tipos de modelos são os únicos em [24] que possuem todos os seus conceitos dentro do escopo do tradutor, pois a maior parte dos outros modelos utiliza-se do conceito de replicação – sendo considerado como trabalho futuro adicionar este conceito SAN ao tradutor.

A fim de verificar similaridade entre os modelos, foram verificadas as seguintes características dos modelos de entrada e de saída do tradutor: a quantidade de estados totais, a quantidade de estados atingíveis e o número de transições.

O primeiro tipo de modelo descreve o problema do jantar dos filósofos. Foram traduzidos sete derivações deste modelo, variando a quantidade de filósofos para cada modelo (*philXX*, onde *XX* é o número de filósofos) ou a sua estrutura interna. Das sete variações, apenas dois modelos variam sua estrutura interna mantendo o mesmo número de filósofos – *phil04f* e *phil04c*, ou seja, um utiliza apenas eventos

locais (*phil04f*) enquanto o outro (*phil04c*) utiliza, além de eventos locais, eventos sincronizantes

O segundo tipo de modelo descreve um protocolo de uma rede de sensor sem fio (*adXX*, onde *XX* é o número de nodos). Foram traduzidos três modelos derivados, sendo duas delas com modificações no valor de identificadores e uma com um aumento no número de nodos.

O modelo da linha de produção é o terceiro tipo, variando apenas o número de estações em cada modelo (*plXX*, onde *XX* é o número de estações). Foram traduzidos três variações deste modelo.

O modelo de primeiro servidor disponível é o quarto tipo, variando apenas o número de servidores em cada modelo (*fasXXc*, onde *XX* é o número de servidores). Foram traduzidos quatro variações deste modelo.

Na Tabela 4 é possível verificar as características coletadas de cada modelo.

Tabela 4: Comparação de características dos modelos SAN e dos respectivos modelos traduzidos utilizados pelo NuSMV

Modelo	Estados Totais		Estados Atingíveis		Transições	
	SAN	NuSMV	SAN	NuSMV	SAN	Algoritmo de caminhamento
phil03f	27	27	12	12	22	22
phil04f	81	81	29	29	72	72
phil04c	81	81	29	29	72	72
phil10	59.049	59.049	5.741	5.741	36.518	36.518
phil12	531.441	531.441	33.461	33.461	256.104	256.104
phil15	14.348.907	1.43489e+007	470.832	470.832	4.516.760	-
phil20	3.486.784.401	3.48678e+009	38.613.965	3.8614e+007	495.238.728	-
ad04c	36	36	6	6	6	6
ad04f	36	36	6	6	6	6
ad10f	26.244	26.244	98	98	186	186
pl4	125	125	115	115	334	334
pl5	625	625	551	551	1.961	1.961
pl10	1.953.125	1.95313e+006	1.391.275	1.39128e+006	9.505.540	-
fas5c	32	32	32	32	111	111
fas6c	64	64	64	64	255	255
fas7c	128	128	128	128	575	575
fas10c	1.024	1.024	1.024	1.024	6.143	6.143

As informações de estados totais e atingíveis dos modelos para o NuSMV foram obtidas através do comando *print_reachable_states*, o qual informa os valores de forma precisa até “10^7 – 1”. Qualquer valor que seja superior a este limite de precisão, o valor é informado em notação científica com cinco números após a vírgula. Por isso os modelos *phil15*, *phil20* e *pl10* apresentam as diferenças grifadas acima. Por exemplo, o valor informado pelo NuSMV de estados totais do modelo

phil20 é 3,48678e+009. O número de transições não foi obtido para os modelos *phil15*, *phil20* e *pl10* devido à complexidade espacial do algoritmo de caminhamento, conforme é explicado na Seção 7.1.

7.1 Algoritmo de caminhamento

Através do NuSMV não é possível obter o número de transições que determinado modelo possui, entretanto é possível realizar através dele um caminhamento no modelo de forma interativa, ou seja, a partir de um estado atual o NuSMV mostra as próximas possibilidades de estados atingíveis permitindo que o usuário escolha o próximo estado. Por isso foi necessário desenvolver um algoritmo que interagisse com o NuSMV a fim de explorar todas as possibilidades de transições que um modelo possui. Através deste algoritmo de caminhamento é possível obter o sistema de transição completo gerado pelo modelo, no qual consta o número exato de estados atingíveis bem como o número de transições.

Este algoritmo realiza um caminhamento em profundidade no modelo e possui o seguinte funcionamento: inicialmente ele executa o NuSMV informando um determinado modelo; inicializa o sistema para a verificação de modelos através do comando "*go*" que lê o modelo informado; utiliza o comando "*pick_state –r*" a fim selecionar de forma não deterministicamente um estado global inicial; verifica o estado global inicial escolhido através do comando "*print_current_state -v*" e guarda este estado global não visitado em uma estrutura de dados; então a partir deste momento o algoritmo entra em um laço que apenas é finalizado quando não há mais estados globais que não foram visitados; dentro deste laço é utilizado o comando "*simulate –i –a –k 1*" para o NuSMV informar quais são todos os próximos estados globais possíveis a partir do estado atual do modelo; para cada estado global possível que o comando "*simulate –i –a –k 1*" retorna, o algoritmo salva tal estado na estrutura de dados (isso se ele já não está salvo anteriormente) e o deixa marcado como um estado que ainda não foi visitado; após salvar todos os próximos estados globais possíveis que o comando "*simulate –i –a –k 1*" retorna, o algoritmo marca o estado atual na estrutura de dados como visitado; o próximo estado é selecionado – o algoritmo informa um valor numérico ao NuSMV referente as opções que ele gerou ao processar o comando de simulação – dentre as possibilidades retornadas pelo comando "*simulate –i –a –k 1*" e que ainda não foi

visitado; então o algoritmo volta ao início do laço; quando todos os estados globais retornados pelo comando "*simulate –i –a –k 1*" já estão salvos na estrutura de dados e já foram visitados então o algoritmo percorre a estrutura de dados procurando um estado global que ainda não foi visitado; ao encontrar tal estado o algoritmo utiliza o comando "*goto_state*" para trocar o estado global do NuSMV – para um estado que já foi descoberto mas não visitado – voltando ao início do laço; por fim, ao sair do laço, o algoritmo informa a quantidade de estados globais distintos, o número de transições existentes entre estes estados e gera um arquivo textual que é, posteriormente, utilizado pelo Graphviz [15] a fim de gerar imagens.

Por necessitar visitar todos os estados globais do modelo, o laço acima descrito é executado n vezes, onde n é o número de estados globais que o modelo possui. Portanto este algoritmo possui uma complexidade temporal de O (n). Ao analisar a complexidade espacial é possível verificar que o pior caso é aquele que a partir de qualquer estado global é possível atingir todos os outros estados globais. Considerando o pior caso onde todos os nodos do modelo possuem uma transição para cada nodo do modelo, a complexidade espacial é de: O (n(n-1)), ou seja, $O(n^2 - n)$ que é o mesmo que $O(n^2)$ onde o n é o número de estados globais atingíveis que o modelo possui. A complexidade espacial deste algoritmo de caminhamento é a razão pela qual não foi possível obter as informações de número de transições para os modelos *phil15, phil20* e *pl10* – conforme é possível observar na Tabela 4 – os quais possuem mais de 470 mil estados globais atingíveis e mais que quatro milhões e meio de transições.

A estrutura de dados utilizada para este algoritmo foi uma lista de objetos que possui as seguintes informações: um identificador único (representação de um estado global do modelo), uma propriedade que informa se o objeto já foi visitado, uma lista de outros estados globais com os quais há transição, duas propriedades numéricas as quais guardam o trace number e o state number informados pelo NuSMV e necessários para utilizar o comando "goto_state" do NuSMV. A partir dessa estrutura é possível gerar um arquivo textual – que contém todos estados globais e suas interações – aceito pelo Graphviz [15] a fim de gerar imagens. Desta forma é possível comparar a topologia da cadeia de Markov de um determinado modelo SAN com o sistema de transição da tradução deste modelo SAN, conforme é possível visualizar na Seção 8.2 bem como nos Apêndices B, C, D, E, F, G, H, I, J e K. Em todos estes exemplos a topologia de ambos é igual, tornando-se, cada uma

destas comparações, mais um indício da corretude do mapeamento de conceitos e do tradutor desenvolvido.

8 EXEMPLIFICAÇÃO

A tradução de dois modelos, retirados de [5] [13] [24], é apresentada abaixo em detalhes, bem como um conjunto de propriedades CTL para estes dois modelos.

8.1 Tradução do modelo textual phil04c.san

O modelo apresentado abaixo é resultado da tradução do modelo SAN apresentado na Seção 2.1 e citado em diversas partes na Seção 6.3.

Conforme explicado na Seção 6, é necessário modificar os valores reais para que sejam valores inteiros. Para realizar a tradução do modelo SAN da Seção 2.1, é necessário alterar os valores de *lambda* (1.000000) e de *mu* (2.000000) para 1 e 2, respectivamente. Abaixo está a tradução para a linguagem do NuSMV.

```
MODULE ad_P3()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P3 : ad_P3();
  a_P2 : ad_P2();
  a_P1 : ad_P1();
  a_P0 : ad_P0();

INIT
  ( nb_s_Thinking = i_P )
TRANS
  next_local_step_ad_P3
  |  synchronized_steps
  |  next_local_step_ad_P2
  |  next_local_step_ad_P1
  |  next_local_step_ad_P0
```

```
DEFINE
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Left ? 1 :  (   a_P3.state = s_Right ? 2 :  -1  )  )   );
  -- ================== ad_P3 automata local steps section ==================
  local_step_ad_P3 := (
    -- connection s_Right - s_Thinking
    ( ( a_P3.state = s_Right & e_r_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P3.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P0.unchanged
  );
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_t_r_0
        &
        (
        -- connection s_Thinking - s_Thinking
         ( a_P1.state = s_Thinking & next(a_P1.state) = s_Thinking )
        -- connection s_Right - s_Right
        | ( a_P1.state = s_Right & next(a_P1.state) = s_Right )
```

```
    )
    &
    (
    -- connection s_Thinking - s_Right
     ( a_P0.state = s_Thinking & next(a_P0.state) = s_Right )
    )
    & a_P3.unchanged
    & a_P2.unchanged
)
|
(
    e_r_l_0
    &
    (
    -- connection s_Thinking - s_Thinking
     ( a_P3.state = s_Thinking & next(a_P3.state) = s_Thinking )
    -- connection s_Left - s_Left
    | ( a_P3.state = s_Left & next(a_P3.state) = s_Left )
    )
    &
    (
    -- connection s_Right - s_Left
     ( a_P0.state = s_Right & next(a_P0.state) = s_Left )
    )
    & a_P2.unchanged
    & a_P1.unchanged
)
|
(
    e_t_r_1
    &
    (
    -- connection s_Thinking - s_Thinking
     ( a_P2.state = s_Thinking & next(a_P2.state) = s_Thinking )
    -- connection s_Right - s_Right
    | ( a_P2.state = s_Right & next(a_P2.state) = s_Right )
    )
    &
    (
    -- connection s_Thinking - s_Right
     ( a_P1.state = s_Thinking & next(a_P1.state) = s_Right )
    )
    & a_P3.unchanged
    & a_P0.unchanged
)
|
(
    e_r_l_1
    &
    (
    -- connection s_Right - s_Left
     ( a_P1.state = s_Right & next(a_P1.state) = s_Left )
    )
    &
    (
    -- connection s_Thinking - s_Thinking
     ( a_P0.state = s_Thinking & next(a_P0.state) = s_Thinking )
    )
    & a_P3.unchanged
```

```
        & a_P2.unchanged
    )
    |
    (
        e_t_r_2
        &
        (
        -- connection s_Thinking - s_Thinking
         ( a_P3.state = s_Thinking & next(a_P3.state) = s_Thinking )
        )
        &
        (
        -- connection s_Thinking - s_Right
         ( a_P2.state = s_Thinking & next(a_P2.state) = s_Right )
        )
        & a_P1.unchanged
        & a_P0.unchanged
    )
    |
    (
        e_r_l_2
        &
        (
        -- connection s_Right - s_Left
         ( a_P2.state = s_Right & next(a_P2.state) = s_Left )
        )
        &
        (
        -- connection s_Thinking - s_Thinking
         ( a_P1.state = s_Thinking & next(a_P1.state) = s_Thinking )
        )
        & a_P3.unchanged
        & a_P0.unchanged
    )
    |
    (
        e_t_l_3
        &
        (
        -- connection s_Thinking - s_Left
         ( a_P3.state = s_Thinking & next(a_P3.state) = s_Left )
        )
        &
        (
        -- connection s_Thinking - s_Thinking
         ( a_P2.state = s_Thinking & next(a_P2.state) = s_Thinking )
        )
        & a_P1.unchanged
        & a_P0.unchanged
    )
    |
    (
        e_l_r_3
        &
        (
        -- connection s_Left - s_Right
         ( a_P3.state = s_Left & next(a_P3.state) = s_Right )
        )
        &
```

```
      (
      -- connection s_Thinking - s_Thinking
       ( a_P0.state = s_Thinking & next(a_P0.state) = s_Thinking )
      -- connection s_Right - s_Right
      | ( a_P0.state = s_Right & next(a_P0.state) = s_Right )
      )
      & a_P2.unchanged
      & a_P1.unchanged
  )
);

-- ================== identifiers section ==================
i_P := ( 4 );

i_lambda := ( 1 );

i_mu := ( 2 );

-- ================== nb operators section ==================
nb_s_Thinking := (
  ( a_P3.state = s_Thinking ? 1 : 0 )  +
  ( a_P2.state = s_Thinking ? 1 : 0 )  +
  ( a_P1.state = s_Thinking ? 1 : 0 )  +
  ( a_P0.state = s_Thinking ? 1 : 0 )
);

-- ================== events section ==================
e_t_r_0 := ( i_lambda > 0 );

e_r_l_0 := ( i_lambda > 0 );

e_l_t_0 := ( i_mu > 0 );

e_t_r_1 := ( i_lambda > 0 );

e_r_l_1 := ( i_lambda > 0 );

e_l_t_1 := ( i_mu > 0 );

e_t_r_2 := ( i_lambda > 0 );

e_r_l_2 := ( i_lambda > 0 );

e_l_t_2 := ( i_mu > 0 );

e_t_l_3 := ( i_lambda > 0 );

e_l_r_3 := ( i_lambda > 0 );

e_r_t_3 := ( i_mu > 0 );
```

A cadeia de Markov equivalente ao modelo SAN apresentada no Apêndice B foi obtida com através do San-Lite-Solver [27] e do Graphviz [15]. O sistema de transição de estados do modelo traduzido para o NuSMV é apresentado no Apêndice C. A imagem do Apêndice C foi obtida através do algoritmo de

caminhamento – que é explicado na Seção 7.1 – juntamente com o Graphviz. As taxas neste modelo não são relevantes visto que o NuSMV realiza uma análise qualitativa e não quantitativa.

8.2 Tradução do modelo textual ad04f.san

Este modelo foi utilizado no artigo [13]. Os comentários iniciais do modelo que informam como citar tal modelo foram retirados, como também a seção *result*.

```
identifiers
  bandwidth_b = 2000000;                    // bandwidth in bits/seconds
  bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
  bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
  size_pack = 1500;                           // package size in bytes
  RTS = 40;                                 // Request To Send header in bytes
  CTS_ACK = 39;                             // Clear To Send and Acknowledge
headers in bytes
  MAC = 47;                                 // Medium Access Control header in
bytes
  IFT = 50+(3*10)+280;                      // InterFrame Time (DIFS + 3*SIFS +
Average Backoff Time) in microseconds (approximated)
  IFS = ((bandwidth_B/1000000)*IFT);        // InterFrame Size cost in bytes
(approximated)
  overhead = RTS+CTS_ACK+MAC+IFS;           // headers
  mu = (bandwidth_B)/(size_pack+overhead); // maximum throughput rate
(theoretically as many package as the medium can handle)
  lambda = 50000;                           // package generation rate (one
package/DIFS)

  r12 = lambda * ((st MN_3 == I) && (st MN_4 == I));
  r23 = lambda * ((st MN_1 == I) && (st MN_4 == I));
  r34 = lambda * ((st MN_1 == I) && (st MN_2 == I));

events
  loc tx1 mu;                               // transmission of one package
  loc tx2 mu;                               // transmission of one package
  syn tx3 mu;                               // transmission of one package

  syn g12 r12;                              // package generated from 1 to 2
  syn g23 r23;                              // package routed from 2 to 3
  syn g34 r34;                              // package routed from 3 to 4

partial reachability = (nb I == 4);         // initial state

network Ad (continuous)
  aut MN_1
    stt I to (T) g12
    stt T to (I) tx1

  aut MN_2
    stt I to (R) g12
    stt R to (T) g23
    stt T to (I) tx2
```

```
aut MN_3
  stt I to (R) g23
  stt R to (T) g34
  stt T to (I) tx3

aut MN_4
  stt I to (R) g34
        stt R to (I) tx3
```

Na imagem e tabela abaixo é possível verificar a estrutura de cada autômato e os eventos do modelo, respectivamente. A função de alcançabilidade é (nb I == 4).

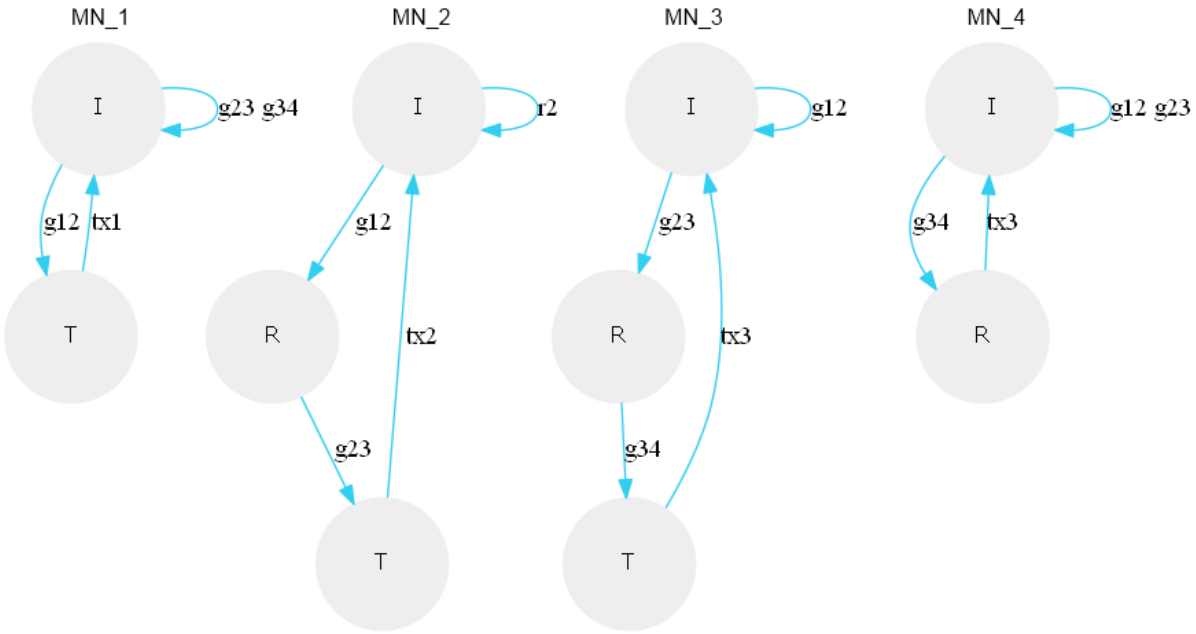


Figura 5: Representação gráfica de ad04f.san

Tabela 5: Eventos de ad04f.san

Evento	Tipo	Taxa
tx1	Local	145,6876
tx2	Local	145,6876
tx3	sincronizante	145,6876
g12	sincronizante	50000 * ((st MN_3 == I) && (st MN_4 == I))
g23	sincronizante	50000 * ((st MN_1 == I) && (st MN_4 == I))
g34	sincronizante	50000 * ((st MN_1 == I) && (st MN_2 == I))

Abaixo está a tradução para a linguagem do NuSMV do modelo acima:

```
MODULE ad_MN_1()
VAR state : {s_I,s_T};
DEFINE
```

```
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_2()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_3()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_4()
VAR state : {s_I,s_R};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_MN_1 : ad_MN_1();
  a_MN_2 : ad_MN_2();
  a_MN_3 : ad_MN_3();
  a_MN_4 : ad_MN_4();

INIT
  ( nb_s_I = 4 )
TRANS
  next_local_step_ad_MN_1
  |  synchronized_steps
  |  next_local_step_ad_MN_2
DEFINE
  a_MN_1_getCurrentIndex := (   a_MN_1.state = s_I ? 0 :  (   a_MN_1.state = s_T ?
1 :  -1  )   );
  -- ================ ad_MN_1 automata local steps section ================
  local_step_ad_MN_1 := (
    -- connection s_T - s_I
    ( ( a_MN_1.state = s_T & e_tx1 ) & ( next(a_MN_1.state) = s_I ) )
  );
  next_local_step_ad_MN_1 := ( local_step_ad_MN_1
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
  );
  a_MN_2_getCurrentIndex := (   a_MN_2.state = s_I ? 0 :  (   a_MN_2.state = s_R ?
1 :  (   a_MN_2.state = s_T ? 2 :  -1  )  )   );
  -- ================ ad_MN_2 automata local steps section ================
  local_step_ad_MN_2 := (
    -- connection s_T - s_I
    ( ( a_MN_2.state = s_T & e_tx2 ) & ( next(a_MN_2.state) = s_I ) )
  );
  next_local_step_ad_MN_2 := ( local_step_ad_MN_2
    & a_MN_1.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
  );
  a_MN_3_getCurrentIndex := (   a_MN_3.state = s_I ? 0 :  (   a_MN_3.state = s_R ?
1 :  (   a_MN_3.state = s_T ? 2 :  -1  )  )   );
  a_MN_4_getCurrentIndex := (   a_MN_4.state = s_I ? 0 :  (   a_MN_4.state = s_R ?
1 :  -1  )   );
```

```
-- ================== SYNCHRONIZED STEPS ==================
synchronized_steps := (
  (
      e_tx3
      &
      (
      -- connection s_T - s_I
       ( a_MN_3.state = s_T & next(a_MN_3.state) = s_I )
      )
      &
      (
      -- connection s_R - s_I
       ( a_MN_4.state = s_R & next(a_MN_4.state) = s_I )
      )
      & a_MN_1.unchanged
      & a_MN_2.unchanged
  )
  |
  (
      e_g12
      -- connection s_I - s_T
      & ( a_MN_1.state = s_I & next(a_MN_1.state) = s_T )
      &
      (
      -- connection s_I - s_R
       ( a_MN_2.state = s_I & next(a_MN_2.state) = s_R )
      )
      & a_MN_3.unchanged
      & a_MN_4.unchanged
  )
  |
  (
      e_g23
      &
      (
      -- connection s_R - s_T
       ( a_MN_2.state = s_R & next(a_MN_2.state) = s_T )
      )
      &
      (
      -- connection s_I - s_R
       ( a_MN_3.state = s_I & next(a_MN_3.state) = s_R )
      )
      & a_MN_1.unchanged
      & a_MN_4.unchanged
  )
  |
  (
      e_g34
      &
      (
      -- connection s_R - s_T
       ( a_MN_3.state = s_R & next(a_MN_3.state) = s_T )
      )
      &
      (
      -- connection s_I - s_R
       ( a_MN_4.state = s_I & next(a_MN_4.state) = s_R )
      )
```

```
        & a_MN_1.unchanged
        & a_MN_2.unchanged
    )
  );
  -- ================== identifiers section ==================
  i_bandwidth_b := ( 2000000 );

  i_bandwidth_B := ( i_bandwidth_b / 8 );

  i_bandwidth_Mbps := ( i_bandwidth_b / 1000000 );

  i_size_pack := ( 1500 );

  i_RTS := ( 40 );

  i_CTS_ACK := ( 39 );

  i_MAC := ( 47 );

  i_IFT := ( ( 50 + ( 3 * 10 ) ) + 280 );

  i_IFS := ( ( ( i_bandwidth_B / 1000000 ) * i_IFT ) );

  i_overhead := ( ( ( i_RTS + i_CTS_ACK ) + i_MAC ) + i_IFS );

  i_mu := ( ( i_bandwidth_B ) / ( i_size_pack + i_overhead ) );

  i_lambda := ( 50000 );

  i_r12 := ( i_lambda * ( ( ( ( a_MN_3.state = s_I ) & ( a_MN_4.state = s_I ) ) )
? 1 : 0 ) );

  i_r23 := ( i_lambda * ( ( ( ( a_MN_1.state = s_I ) & ( a_MN_4.state = s_I ) ) )
? 1 : 0 ) );

  i_r34 := ( i_lambda * ( ( ( ( a_MN_1.state = s_I ) & ( a_MN_2.state = s_I ) ) )
? 1 : 0 ) );

  -- ================== nb operators section ==================
  nb_s_I := (
    ( a_MN_1.state = s_I ? 1 : 0 )  +
    ( a_MN_2.state = s_I ? 1 : 0 )  +
    ( a_MN_3.state = s_I ? 1 : 0 )  +
    ( a_MN_4.state = s_I ? 1 : 0 )
  );

  -- ================== events section ==================
  e_tx1 := ( i_mu > 0 );

  e_tx2 := ( i_mu > 0 );

  e_tx3 := ( i_mu > 0 );

  e_g12 := ( i_r12 > 0 );

  e_g23 := ( i_r23 > 0 );

  e_g34 := ( i_r34 > 0 );
```

A cadeia de markov equivalente ao modelo SAN apresentada abaixo foi obtida com através do San-Lite-Solver [27] e do Graphviz [15].

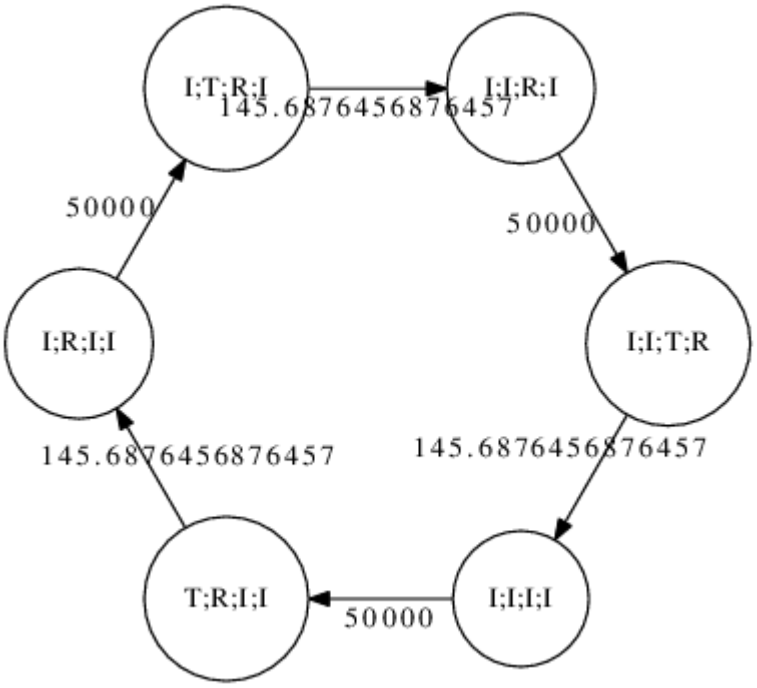


Figura 6: Cadeia de Markov equivalente ao modelo ad04f.san

O sistema de transição de estados do modelo traduzido para o NuSMV é apresentado pela imagem abaixo. Esta imagem foi obtida através do algoritmo de caminhamento – que é explicado na Seção 7.1 – juntamente com o Graphviz. As taxas neste modelo não são relevantes visto que o NuSMV realiza uma análise qualitativa e não quantitativa.

Figura 7: Sistema de transição de estados do modelo ad04f.smv – modelo traduzido

8.3 Avaliação qualitativa em modelos SAN

Um conjunto de propriedades CTL foi desenvolvido para alguns modelos SAN a fim de realizar a análise qualitativa nos modelos equivalentes traduzidos. Estas oito propriedades foram desenvolvidas para o modelo *phil3.san*. Tais propriedades também foram executadas para modelos maiores, que modelam o mesmo problema do jantar dos filósofos, presentes na Tabela 4.

É importante ressaltar que todas as propriedades apresentadas nesta seção são executadas quase que instantaneamente no NuSMV, pois visivelmente todas elas demoram menos que um segundo para o NuSMV retornar o contra exemplo ou afirmar que a propriedade é verdadeira. As oito propriedades apresentadas a seguir também foram executadas de forma instantânea com o modelo *phil20.san* que possui três bilhões de estados totais, 38 milhões de estados atingíveis e 495 milhões de transições, conforme é possível verificar na Tabela 4. Por uma questão de limitação de espaço, é apresentado os contra exemplos para as próximas propriedades para o modelo *phil3.san* em vez do *phil20.san*.

1. EF EG st P0 == Right
2. AG (st P0 == Left → AF (st P1 != Left && st P2 != Right))
3. AG st P1 == Right
4. E [st P0 == Thinking U EG (st P1 == Right)]
5. (! EX st P0 == Thinking) | (! EF EG st P1 == Right)
6. E [TRUE U EX(st P0 == Thinking)] & E [TRUE U EG (st P1 == Right)]
7. AG AX st P2 == Right
8. AG ((st P0 == Thinking || st P0 == Right) → AF (st P1 == Left))

As primeiras três propriedades possuem significado, já as demais foram desenvolvidas apenas para verificar o contraexemplo e testemunhas gerado pelo NuSMV. A primeira propriedade assegura que existe algum caminho onde o filósofo 0 sempre quer comer. Esta propriedade também é verdadeira para todos os outros filósofos do modelo. A segunda propriedade significa que enquanto o filósofo 0 está comendo os demais não estão. Esta propriedade, da mesma forma que a primeira, também é verdadeira para os demais filósofos do modelo. A terceira verifica se globalmente o filósofo 1 quer comer.

A fim de verificar estas propriedades com o NuSMV é necessário traduzir as proposições atômicas, que estão descritas em SAN, de forma que sejam compatíveis com a linguagem do NuSMV e com o modelo traduzido, conforme explicado na Seção 4.1.

No NuSMV não é possível criar uma testemunha no NuSMV para uma propriedade CTL verdadeira. Portanto a fim de obter uma testemunha para cada propriedade verdadeira, cada propriedade verdadeira foi negada ao ser verificada pelo NuSMV. Desta forma, foi possível obter um contraexemplo da negação de uma propriedade verdadeira, sendo equivalente a uma testemunha da propriedade verdadeira. Abaixo é apresentado: a tradução de cada uma das oito propriedades acima, bem como o contraexemplo ou testemunha de cada propriedade.

1. EF EG a_P0.state = s_Right; Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Thinking
-- Loop starts here
-> State: 1.2 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Right
-> State: 1.3 <-
  a_P2.state = s_Left
  a_P1.state = s_Thinking
  a_P0.state = s_Right
-> State: 1.4 <-
  a_P2.state = s_Right
  a_P1.state = s_Thinking
  a_P0.state = s_Right
-> State: 1.5 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Right
```

2. AG (a_P0.state = s_Left → AF (a_P1.state != s_Left & a_P2.state != s_Right)); Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Thinking
```

3. AG a_P1.state = s_Right; Esta propriedade é falsa conforme é apresentado o seu contraexemplo:

```
-> State: 1.1 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Thinking
```

4. E [a_P0.state = s_Thinking U EG (a_P1.state = s_Right)]; Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Thinking
-- Loop starts here
-> State: 1.2 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Right
  a_P0.state = s_Thinking
-> State: 1.3 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Right
  a_P0.state = s_Right
-> State: 1.4 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Right
  a_P0.state = s_Left
-> State: 1.5 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Right
  a_P0.state = s_Thinking
```

5. (! EX a_P0.state = s_Thinking) | (! EF EG a_P1.state = s_Right); Esta propriedade é falsa conforme é apresentado o seu contraexemplo:

```
-> State: 1.1 <-                    -> State: 1.2 <-
  a_P2.state = s_Thinking             a_P2.state = s_Left
  a_P1.state = s_Thinking             a_P1.state = s_Thinking
  a_P0.state = s_Thinking             a_P0.state = s_Thinking
```

6. E [TRUE U EX(a_P0.state = s_Thinking)] & E [TRUE U EG (a_P1.state = s_Right)]; Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_P2.state = s_Thinking
  a_P1.state = s_Thinking
  a_P0.state = s_Thinking
```

7. AG AX a_P2.state = s_Right; Esta propriedade é falsa conforme é apresentado o seu contraexemplo:

```
-> State: 1.1 <-                    -> State: 1.2 <-
  a_P2.state = s_Thinking             a_P2.state = s_Left
  a_P1.state = s_Thinking             a_P1.state = s_Thinking
  a_P0.state = s_Thinking             a_P0.state = s_Thinking
```

8. AG ((a_P0.state = s_Thinking | a_P0.state = s_Right) → AF (a_P1.state = s_Left)); Esta propriedade é falsa conforme é apresentado o seu contraexemplo:

```
-- Loop starts here                 -> State: 1.3 <-
-> State: 1.1 <-                     a_P2.state = s_Right
  a_P2.state = s_Thinking             a_P1.state = s_Thinking
  a_P1.state = s_Thinking             a_P0.state = s_Thinking
  a_P0.state = s_Thinking            -> State: 1.4 <-
-> State: 1.2 <-                     a_P2.state = s_Thinking
  a_P2.state = s_Left                 a_P1.state = s_Thinking
  a_P1.state = s_Thinking             a_P0.state = s_Thinking
  a_P0.state = s_Thinking
```

Estas três propriedades foram desenvolvidas para o modelo *ad10.san*.

1. AG (st MN_1 == T → AF (st MN_10 == R))
2. EG (st MN_1 == T → EF (st MN_10 == R))
3. E ([! (st MN_10 == R) U (st MN_1 == T)])

A primeira propriedade significa que para todos os caminhos no modelo sempre que ocorrer uma transmissão do nodo MN_1 no futuro o nodo MN_10 vai recebê-la. A segunda possui o mesmo significado da primeira com a única diferença que nesta ao gerar a testemunha apresenta um caminho onde um envio do nodo MN_1 é recebido pelo nodo MN_10. A terceira propriedade assegura que o nodo MN_10 não receberá a transmissão até que ocorra uma transmissão do nodo MN_1.

A tradução destas três propriedades segue abaixo juntamente com os contraexemplos e testemunhas.

1. AG (a_MN_1.state = s_T → AF (a_MN_10.state = s_R)); Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
```

2. EG (a_MN_1.state = s_T → EF (a_MN_10.state = s_R)); Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-- Loop starts here
-> State: 1.1 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.2 <-
  a_MN_1.state = s_T
  a_MN_2.state = s_R
-> State: 1.11 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_R
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.12 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
```

```
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.3 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_R
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.4 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_T
  a_MN_3.state = s_R
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.5 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_R
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.6 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_T
  a_MN_4.state = s_R
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.7 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_R
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_T
  a_MN_7.state = s_R
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.13 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_R
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.14 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_T
  a_MN_8.state = s_R
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.15 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_R
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.16 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_T
  a_MN_9.state = s_R
  a_MN_10.state = s_I
-> State: 1.17 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
```

```
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.8 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_T
  a_MN_5.state = s_R
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.9 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_R
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
-> State: 1.10 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_T
  a_MN_6.state = s_R
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
```

```
  a_MN_8.state = s_I
  a_MN_9.state = s_R
  a_MN_10.state = s_I
-> State: 1.18 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_T
  a_MN_10.state = s_R
-> State: 1.19 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
```

3. E ([! (a_MN_10.state = s_R) U (a_MN_1.state = s_T)]); Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é:

```
-> State: 1.1 <-
  a_MN_1.state = s_I
  a_MN_2.state = s_I
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_I
```

```
-> State: 1.2 <-
  a_MN_1.state = s_T
  a_MN_2.state = s_R
  a_MN_3.state = s_I
  a_MN_4.state = s_I
  a_MN_5.state = s_I
  a_MN_6.state = s_I
  a_MN_7.state = s_I
  a_MN_8.state = s_I
  a_MN_9.state = s_I
  a_MN_10.state = s_t
```

Para o modelo *pl3.san* [23] foi desenvolvido a propriedade abaixo a fim de obter a sua testemunha. Esta propriedade também foi executada para modelos maiores, que modelam o mesmo problema da linha de produção, presentes na Tabela 4.

1. E [TRUE U (st M3 == st_1_2)]

A tradução desta propriedade segue abaixo juntamente com sua testemunha.

1. E [TRUE U (a_M3.state = s_st_1_2)] ; Esta propriedade é verdadeira. A testemunha – contraexemplo da negação – desta propriedade é a mesma apresentada na quinta propriedade desta lista;

```
-> State: 1.1 <-
  a_M2.state = s_st_0_1
  a_M3.state = s_st_0_0
-> State: 1.2 <-
  a_M2.state = s_st_1_1
  a_M3.state = s_st_0_0
-> State: 1.3 <-
  a_M2.state = s_st_1_2
  a_M3.state = s_st_0_0
-> State: 1.4 <-
  a_M2.state = s_st_1_1
  a_M3.state = s_st_0_1
-> State: 1.5 <-
  a_M2.state = s_st_0_1
  a_M3.state = s_st_1_1
-> State: 1.6 <-
  a_M2.state = s_st_0_1
  a_M3.state = s_st_1_2
```

9 CONCLUSÃO

Esta dissertação propõe uma abordagem a fim de realizar avaliações qualitativas de modelos SAN. A Seção 6 apresenta e detalha o primeiro mapeamento existente dos conceitos da linguagem SAN para os conceitos de um verificador de modelos existente. O verificador de modelos utilizado nesta dissertação foi o NuSMV. A partir deste mapeamento foi implementado um tradutor de modelos textuais SAN para modelos textuais utilizados pelo NuSMV, o qual é descrito na Seção 6.6.

Conforme é discutido na Seção 7, as características comparadas – estados totais, estados atingíveis e transições – dos modelos SAN e respectivos modelos traduzidos são idênticas, o que dá fortes indícios de que a tradução está correta. Além disso, as topologias – os sistemas de transições de estados – dos modelos SAN e suas traduções são isomórficas, conforme é possível verificar nas imagens dos apêndices, reforçando a hipótese da corretude da tradução.

Dois terços da Seção 8 apresentam dois modelos SAN e seus respectivos modelos traduzidos, desta forma exemplificando de forma completa a tradução de modelos SAN. A Seção 8.3 aborda a realização de análises qualitativas através de propriedades CTL em modelos resultantes da tradução. Desta forma é possível verificar propriedades CTL descritos para três modelos SAN e os seus respectivos contra exemplos (para as propriedades que são falsas) gerados pelo NuSMV. Conforme a Seção 8.3 as análises qualitativas foram realizadas de forma quase instantânea (aproximadamente um segundo) até mesmo para modelos com três bilhões de estados totais, 38 milhões de estados atingíveis e 495 milhões de transições.

Portanto esta dissertação possibilita realizar avaliações qualitativas de modelos SAN através de sua tradução para a linguagem do NuSMV e da utilização do modelo traduzido pelo NuSMV. Entretanto, conforme é detalhado na Seção 6.3 e principalmente na Tabela 3, ainda existem conceitos de SAN que ainda não foram mapeados para a linguagem do NuSMV, sendo a maior parte deles baseados no conceito SAN de replicação.

Consequentemente o mapeamento dos conceitos restantes de SAN para a linguagem do NuSMV é considerado como um dos trabalhos futuros. A

argumentação formal da corretude da tradução, conforme citado na Seção 7, também está inclusa nos trabalhos futuros. Além destes, o aprimoramento do tradutor através da criação de uma camada de abstração similar à apresentada em [1] é outro aspecto que pode ser desenvolvido futuramente. Através desta camada o usuário SAN não necessitaria entender da sintaxe e da semântica da linguagem do NuSMV bem como da utilização deste *Model Checker*. O desenvolvimento desta camanda de abstração pode ser dividido em duas partes: a primeira é a expansão do tradutor a fim de traduzir uma nova seção em SAN que aceite a especificação de propriedades CTL descritos na sintaxe SAN, já a segunda é capacidade de traduzir os contra exemplos gerados pelo NuSMV para a sintaxe utilizada no modelo SAN. Para ambas as partes, a lógica apresentada na Seção 6 desta dissertação será fundamental para desenvolver esta nova camada de abstração.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Arcaini, P.; Gargantini, A.; Riccobene, E. "AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications.", In: Abstract State Machines, Alloy, B and Z. Springer Berlin Heidelberg, 2010. p. 61-74.

[2] Baier, C.; Katoen, J.P. "Principles of Model Checking", Cambridge: MIT Press, 2008, vol. 26202649, pp. 313-449.

[3] Benoit, A.; Brenner, L.; Fernandes, P.; Stewart, W. J. “Peps 2003 User Manual”, 2003, pp. 1-14.

[4] Bobbio, A.; Horváth, A. "Model checking time Petri nets using NuSMV", In: Proceedings of the Fifth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS2001), 2001.

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

[6] "BYACC/J Home Page". Capturado em: http://byaccj.sourceforge.net/, Outubro 2011.

[7] Cavada, R.; Cimatti, A.; Jochim, C. A.; Keighren, G.; Olivetti, E.; Pistore, M.; Roveri, M.; Tchaltsev, A. “NuSMV 2.5 User Manual”, http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf, 2010.

[8] Cavada, R.; Cimatti, A.; Keighren, G.; Olivetti, E.; Pistore, M.; Roveri, M. “NuSMV 2.5 Tutorial”, http://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf, 2011.

[9] Cimatti, A.; Clarke, E.; Giunchiglia, E.; Giunchiglia, F.; Pistore, M.; Roveri, M.; Sebastiani, R.; Tacchella, A. "Nusmv 2: An opensource tool for symbolic model checking", In Computer Aided Verification, 2002, pp. 241-268.

[10] Cimatti, A.; Clarke, E.; Giunchiglia, F.; Roveri, M. "NuSMV: a new symbolic model checker", International Journal on Software Tools for Technology Transfer (STTT), vol. 2, 2000, pp. 410-425.

[11] Clarke, E. M.; Emerson, E. A. "Design and synthesis of synchronization skeletons using branching time temporal logic." In Logic of Programs, vol. 131, 1982, pp. 52–71.

[12] Correa, C. M.; Dotti, F. L.; Fernandes, P.; Maruani, E.; Oleksinski, L. G.; Sales, A. "Um Verificador de Modelos Descritos em Redes de Autômatos Estocásticos", XIII Workshop de Testes e Tolerância a Falhas, Abril 2012, pp. 115-128.

[13] Dotti, F. L.; Fernandes, P.; Sales, A.; Santos, O. M. "Modular analytical performance models for ad hoc wireless networks", Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, IEEE Press, 2005, pp. 164-173.

[14] Fernandes, P.; Plateau, B. "Modeling finite capacity queueing networks with stochastic automata networks.", In Fourth International Workshop on Queueing Networks with Finite Capacity (QNETs 2000), 2000, pp. 1-12.

[15] "Graphviz - Graph Visualization Software". Capturado em: http://www.graphviz.org/, Setembro 2011.

[16] Grobelna, I.; Grobelny, M.; Adamski, M. "Petri Nets and activity diagrams in logic controller specification-transformation and verification", In Mixed Design of Integrated Circuits and Systems (MIXDES), 2010 Proceedings of the 17th International Conference IEEE, 2010, pp. 607-612.

[17] "JFlex - The Fast Scanner Generator for Java". Capturado em: http://jflex.de/, Outubro 2011.

[18] Jiang, Y.; Qiu, Z. "S2N: model transformation from SPIN to NuSMV", In: Model Checking Software. Springer - Berlin Heidelberg, 2012, pp. 255-260.

[19] McMillan, K.L. “Symbolic model checking - An approach to the state explosion problem”. Tese de Doutorado, School of Computer Science - Carnegie Mellon University In Kluwer, 1992.

[20] "Mission | Open Source Initiative". Capturado em: http://www.opensource.org/, Janeiro 2012.

[21] "NuSMV home page". Capturado em: http://nusmv.fbk.eu, Março 2011.

[22] Pan, J. "Software testing", In: Dependable Embedded Systems, vol. 5, 1999, pp. 2006.

[23] Papadopoulos, C.T.; Fernandes, P.; Sales, A.; O'Kelly, M.E.J. "Modeling Exponential Reliable Production Lines using Kronecker Descriptors." In: VIII Conference on Stochastic Models of Manufacturing and Service Operations, 2011, Kusadasi, Izmir, Turkey. Stochastic Models of Manufacturing and Service Operations (SMMSO 2011). Istanbul: Koç University, 2011. p. 253-260.

[24] Performance Evaluation Group. “PEG”. Capturado em: http://www.inf.pucrs.br/peg/, Março 2011.

[25] Queille, J. P.; Sifakis, J. "Specification and verification of concurrent systems in CESAR." In: 5th International Symposium on Programming, vol. 137, 1982, pp 337–351.

[26] Ribeiro, L.; dos Santos, O.M.; Dotti, F.L.; Foss, L. "Correct transformation: From object-based graph grammars to PROMELA", Science of Computer Programming, March 2012, vol. 77, issue 3, pp. 214–246

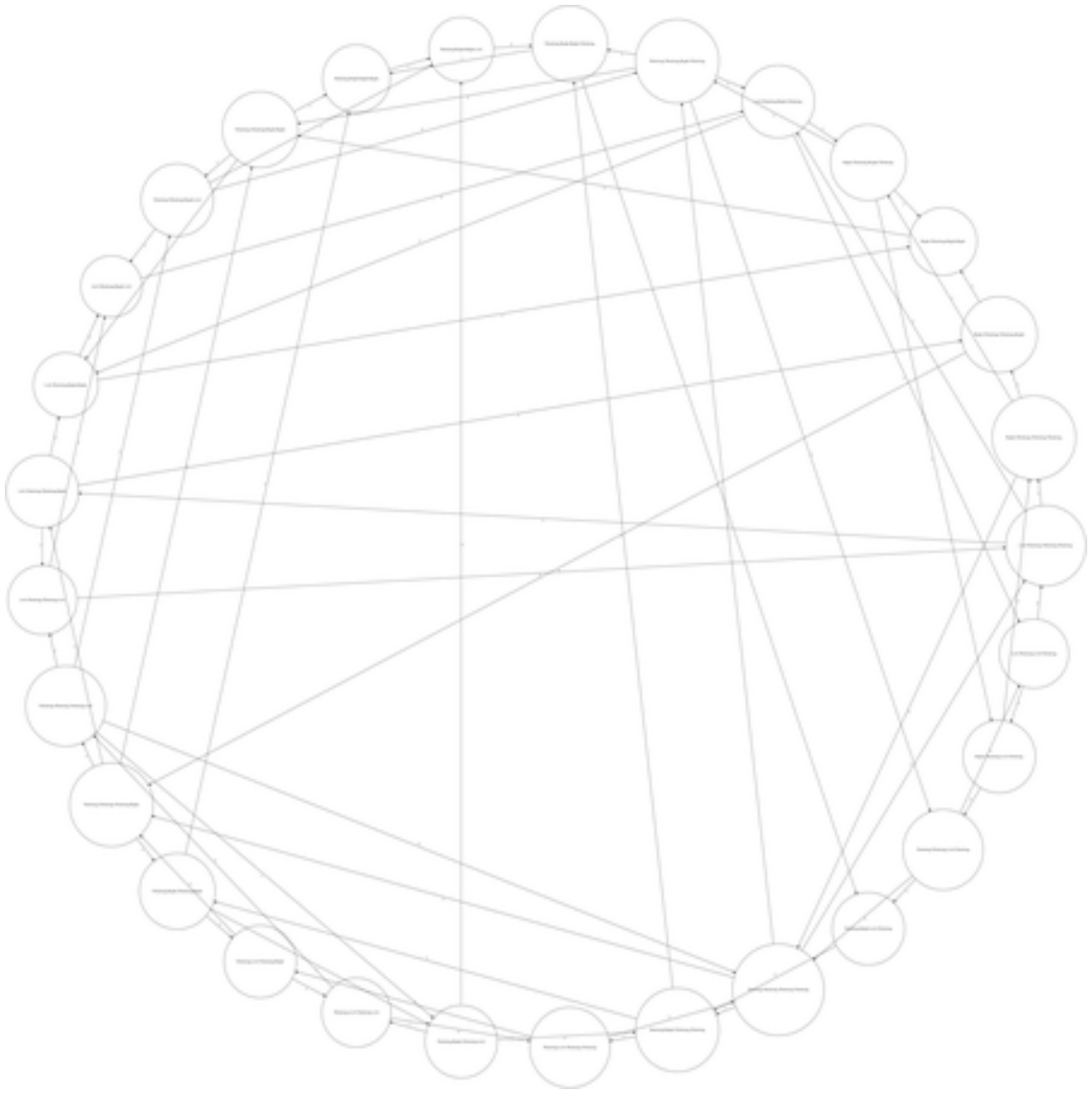
[27] "Tools - Afonso Sales". Capturado em: https://sites.google.com/site/afonsosales/tools, Março 2012.

APÊNDICE A – GRAMÁTICA ACEITA PELO TRADUTOR DA LINGUAGEM SAN PARA A LINGUAGEM DO NUSMV

```
SANModel : identifiersSection eventsSection reachabilitySection networkSection
automataList resultsSection ;

 identifiersSection : IDENTIFIERS identifierList ;
  identifierList : identifierDeclaration | identifierDeclaration identifierList |;
   identifierDeclaration : ID '=' expression ';' ;

 eventsSection : EVENTS eventsList ;
  eventsList : eventDeclaration | eventDeclaration eventsList | ;
  eventDeclaration : eventType eventName eventRate ';' ;
   eventType : LOC | SYN ;
   eventName : ID ;
   eventRate : NUMBER | ID | '(' NUMBER ')' | '(' ID ')' ;

 reachabilitySection : PARTIAL reachabilityDeclaration | reachabilityDeclaration ;
  reachabilityDeclaration : REACHABILITY '=' expression ';' ;

 networkSection : NETWORK networkName '(' networkType ')' ;
  networkName : ID ;
  networkType : CONTINUOUS | DISCRETE ;

 automataList : automataDeclaration | automataDeclaration automataList ;
  automataDeclaration : automataHeader automataBody ;
   automataHeader : AUT automataName ;
    automataName : ID ;
   automataBody : automataStateList ;
    automataStateList : automataStateHeader automataToList | automataStateHeader
automataToList automataStateList ;
     automataStateHeader : STT automataStateName | STT automataStateName '('
automataReward ')' ;
      automataStateName : ID ;
      automataReward :  ID | NUMBER ;
     automataToList : automataToDeclaration | automataToDeclaration automataToList
;
      automataToDeclaration : TO '(' automataStateName ')' automataEventList ;
       automataEventList : automataEventDeclaration | automataEventDeclaration
automataEventList ;
        automataEventDeclaration : eventName | eventName '(' eventProbability ')';
        eventProbability : NUMBER | ID ;

 resultsSection : RESULTS resultList | ;
  resultList : resultDeclaration | resultDeclaration resultList ;
   resultDeclaration : ID '=' expression ';' ;

    expression : constant
   | automataName
   | operatorWithOneArgBeforeArg expressionArgument
   | expressionArgument operatorWithOneArgAfterArg
   | expressionArgument operatorWithTwoArguments expressionArgument
   | expressionArgument operatorWithTwoArguments expression
   | expression operatorWithTwoArguments expressionArgument
   | '('  expression ')' ;

 constant : NUMBER | '(' constant ')' ;
```

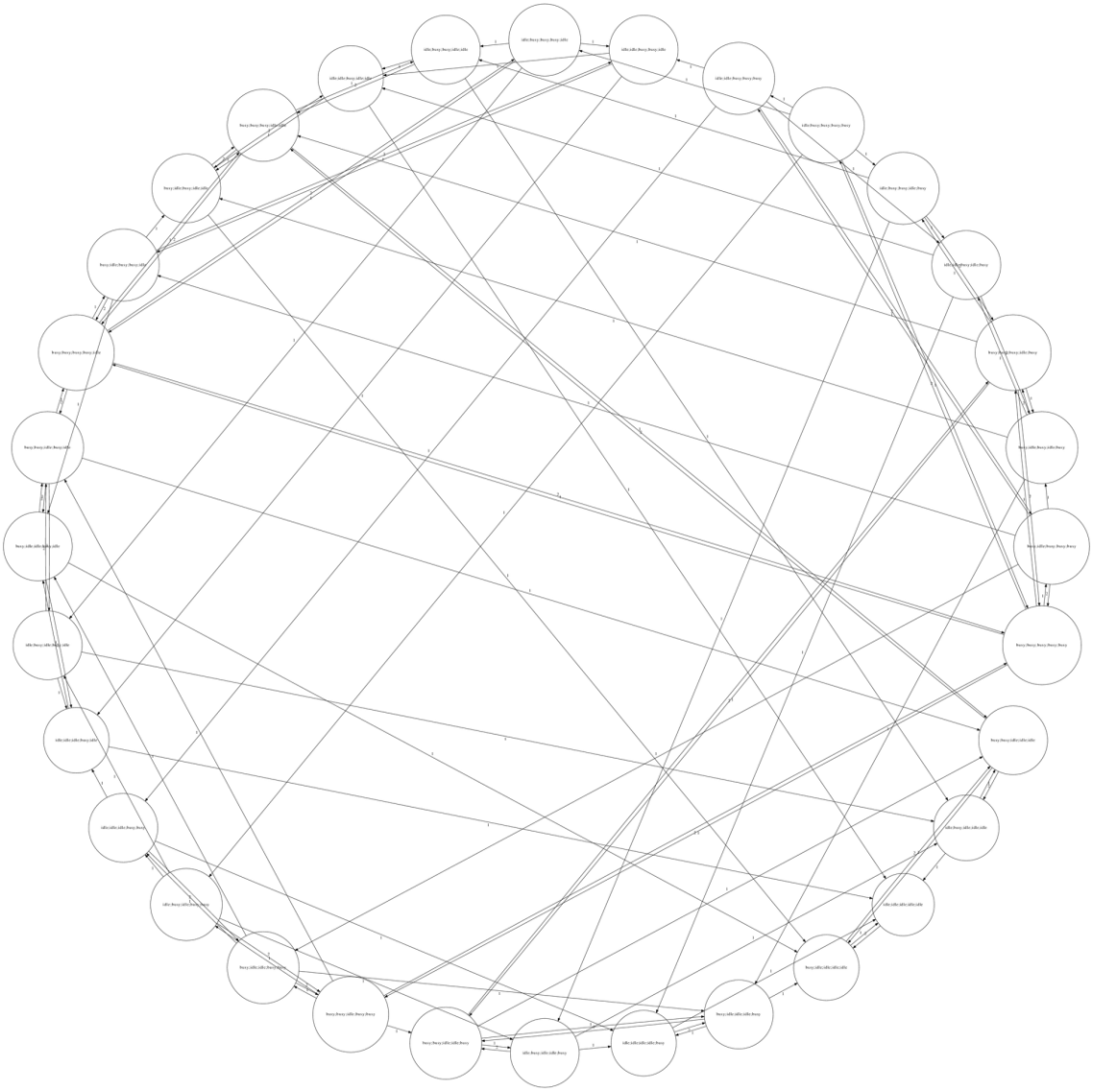
```
 expressionArgument : NUMBER | automataName | expression | '(' expressionArgument
')' ;

 operatorWithTwoArguments : "+" | "-" | "*" | "/" | EQUAL | DIFFERENT | "<" |
LESS_THAN_OR_EQUAL | ">" | GREATER_THAN_OR_EQUAL | LOGIC_OR | LOGIC_AND |
LOGIC_IMPLICATION | XOR | '(' operatorWithTwoArguments ')' ;

 operatorWithOneArgBeforeArg : ST | "@" | NB | RW | "!" | '('
operatorWithOneArgBeforeArg ')' ;

 operatorWithOneArgAfterArg : "!" | '(' operatorWithOneArgAfterArg ')' ;

LETTER=[A-Za-z_]+
ID={LETTER}({LETTER}|{POSITIVENUMBER})*
POSITIVENUMBER=[0-9]+
NUMBER=-?{POSITIVENUMBER}
IDENTIFIERS = "identifiers"
EVENTS = "events"
LOC = "loc"
SYN = "syn"
PARTIAL = "partial"
REACHABILITY = "reachability"
NETWORK = "network"
RESULTS = "results"
CONTINUOUS ="continuous"
DISCRETE = "discrete"
AUT = "aut"
STT = "stt"
TO = "to"
FROM = "from"
ST = "st"
NB = "nb"
RW = "rw"
GREATER_THAN_OR_EQUAL = ">="
LESS_THAN_OR_EQUAL = "<="
XOR = "^"
EQUAL = "=="
DIFFERENT = "!="
LOGIC_AND = "&&"
LOGIC_OR = "||"
LOGIC_EQUIVALENCE = "<->"
LOGIC_IMPLICATION = "->"
```

APÊNDICE B – CADEIA DE MARKOV EQUIVALENTE AO MODELO PHIL04C.SAN

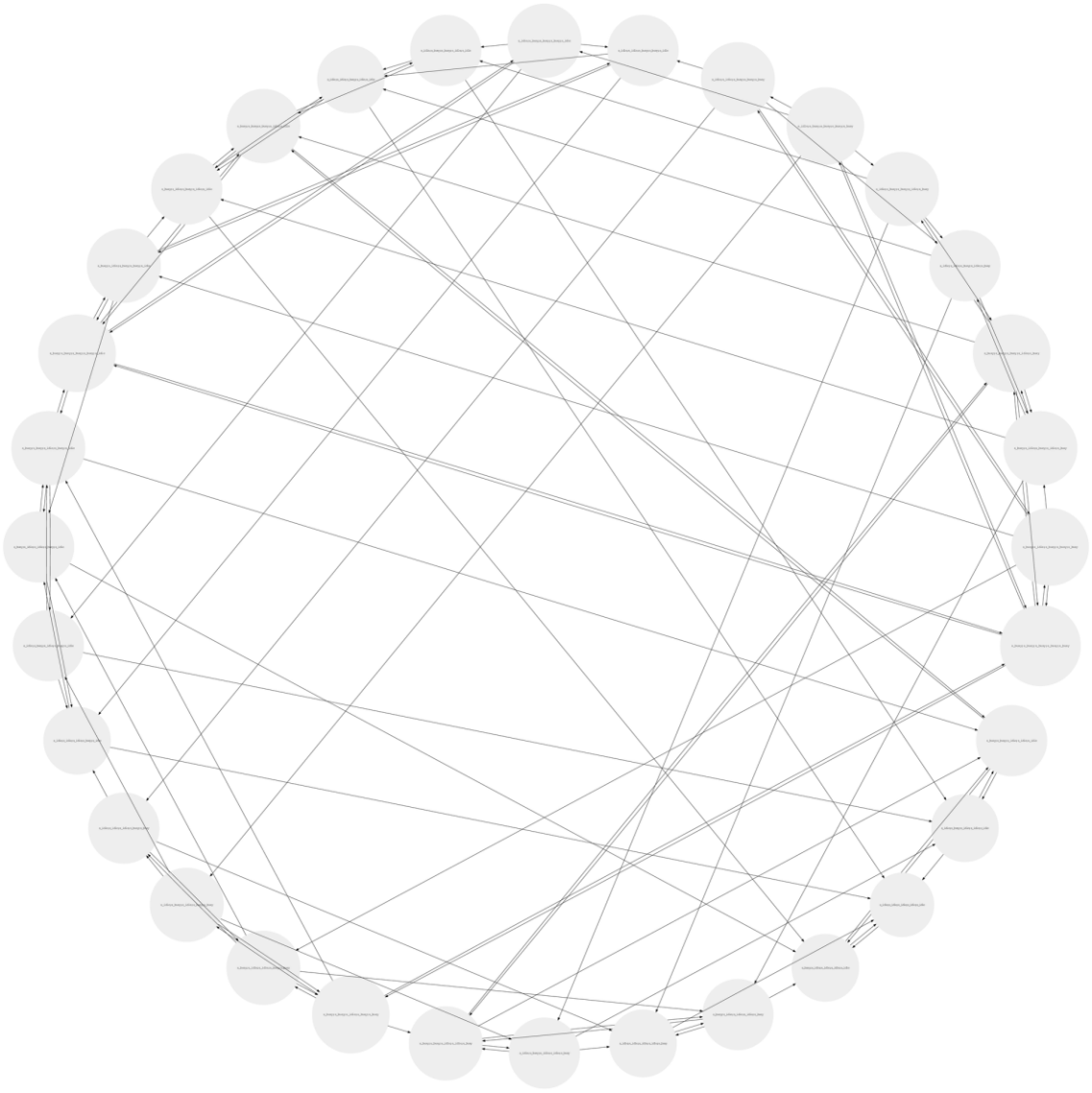


APÊNDICE C – SISTEMA DE TRANSIÇÃO DE ESTADOS DO MODELO PHIL04C.SMV – TRADUÇÃO DE PHIL04C.SAN

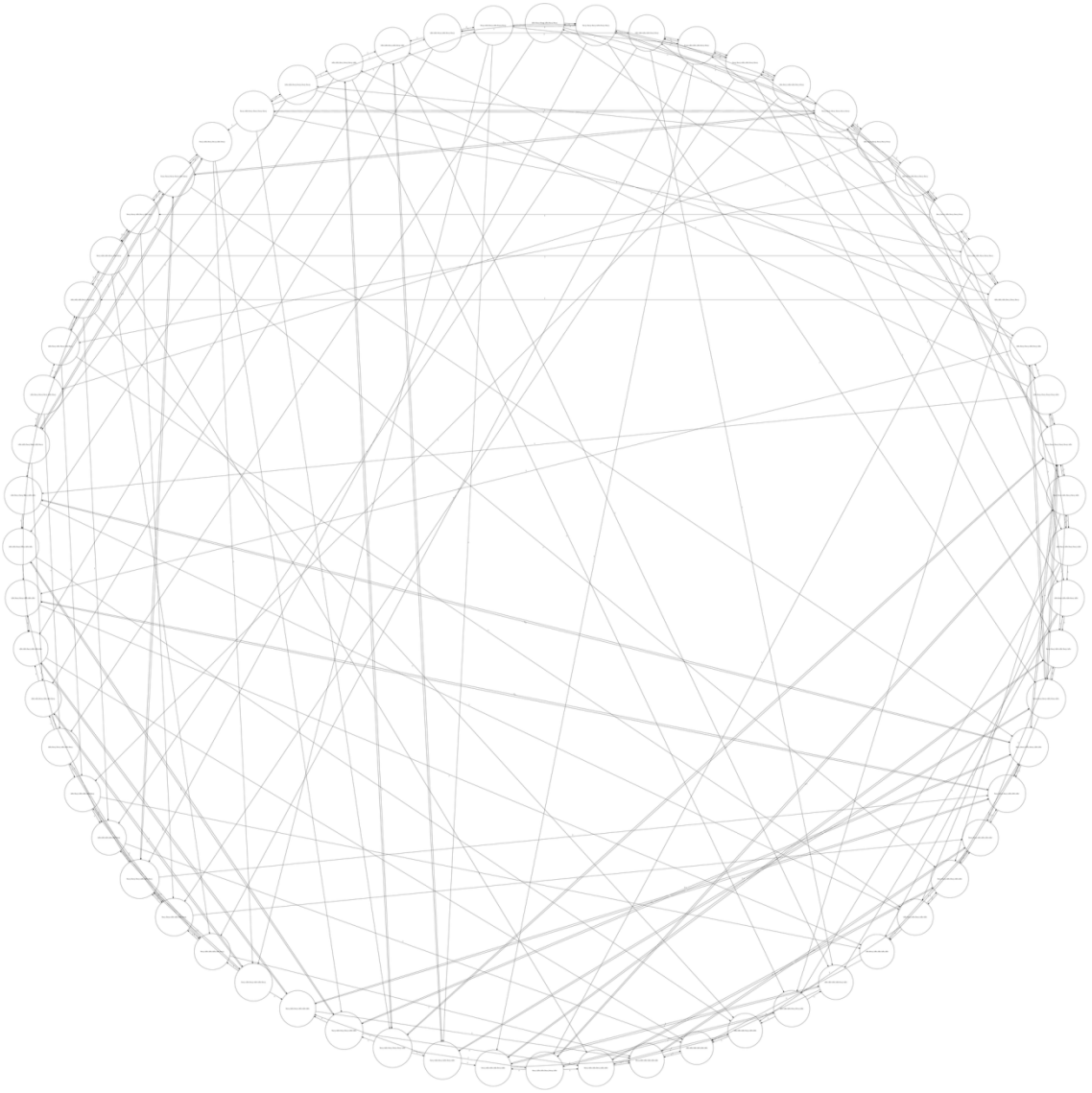
APÊNDICE D – CADEIA DE MARKOV EQUIVALENTE AO MODELO FAS5C.SAN



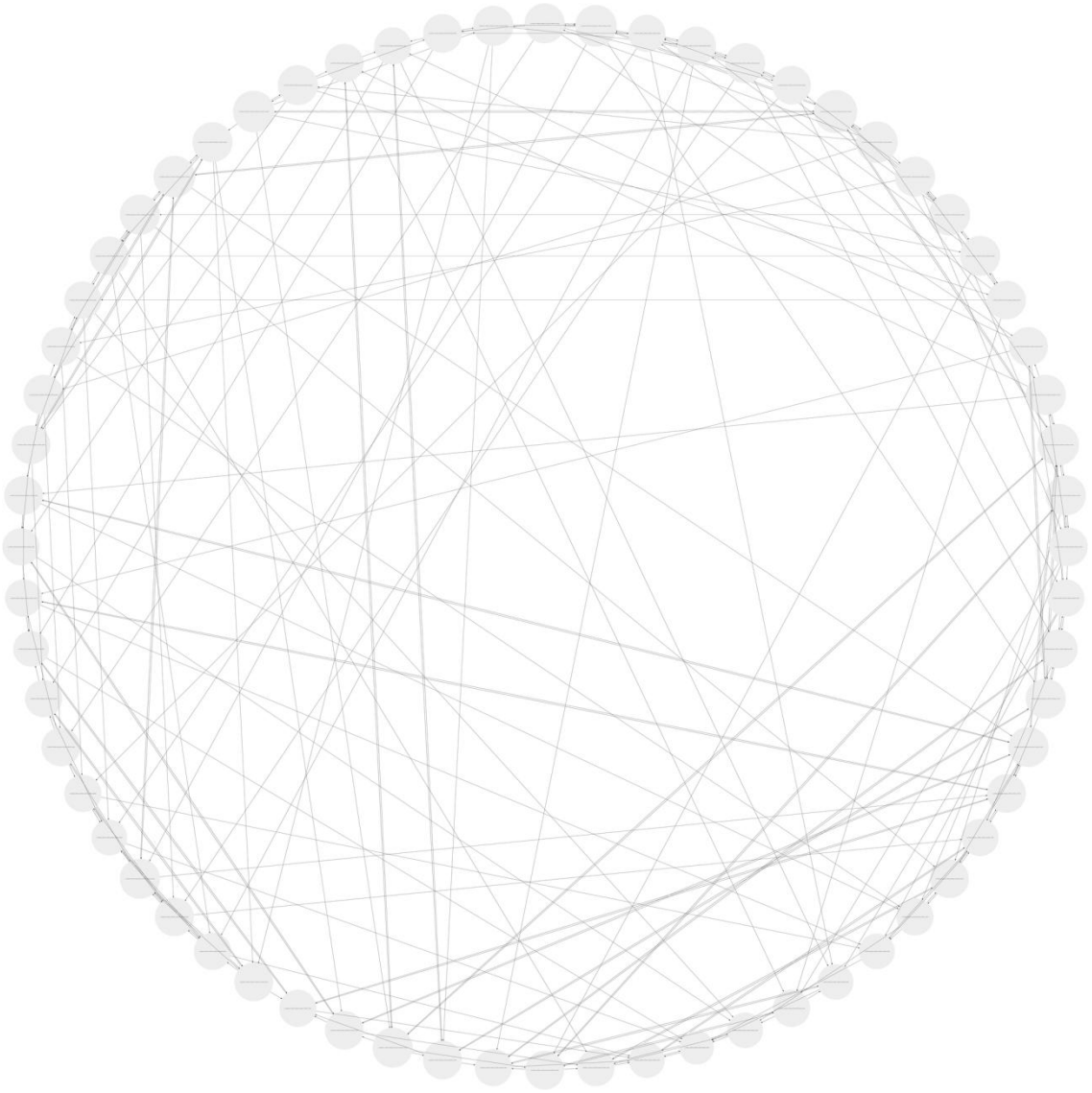
APÊNDICE E – SISTEMA DE TRANSIÇÃO DE ESTADOS DO MODELO FAS5C.SMV – TRADUÇÃO DE FAS5C.SAN



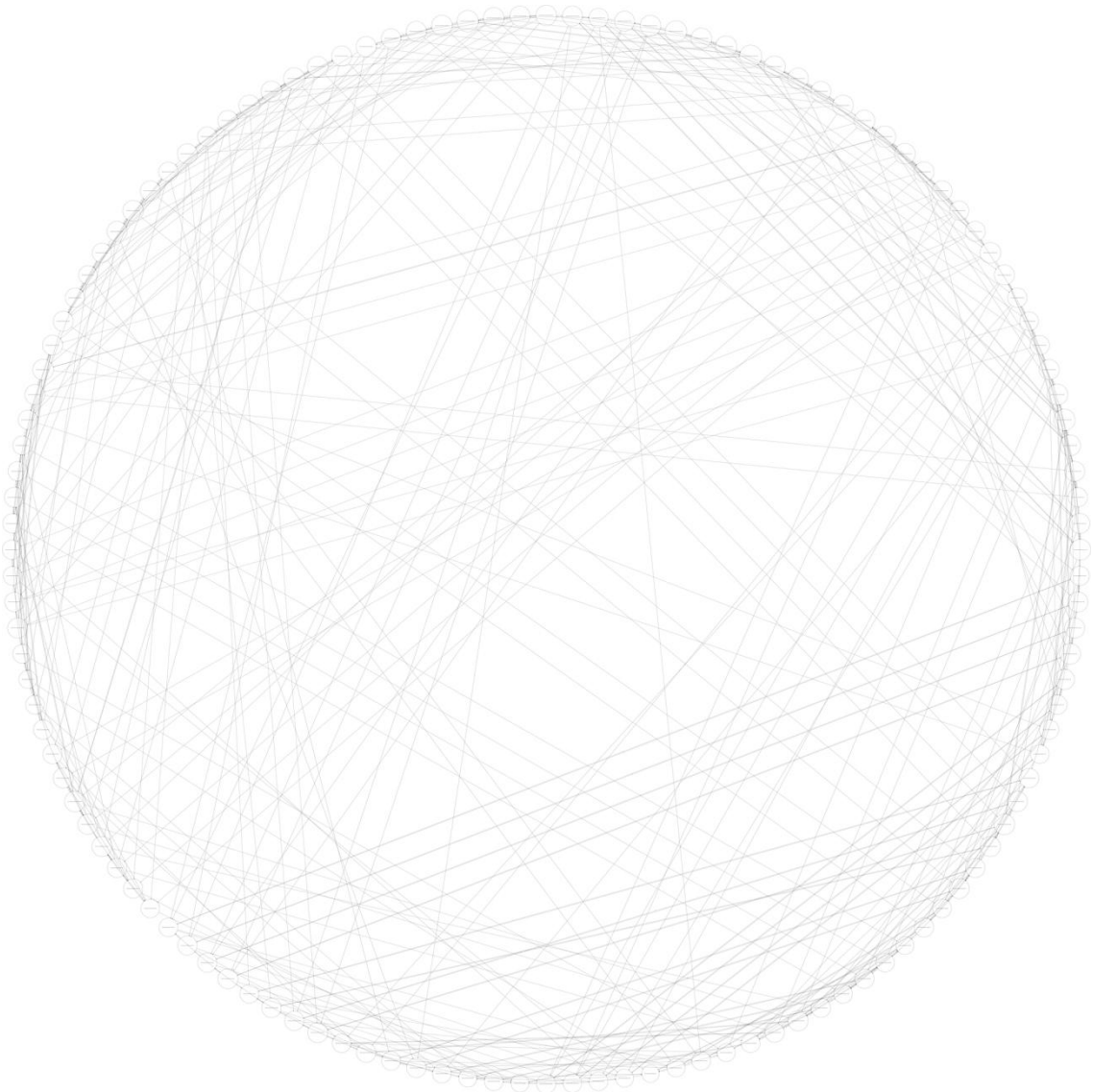
APÊNDICE F – CADEIA DE MARKOV EQUIVALENTE AO MODELO FAS6C.SAN



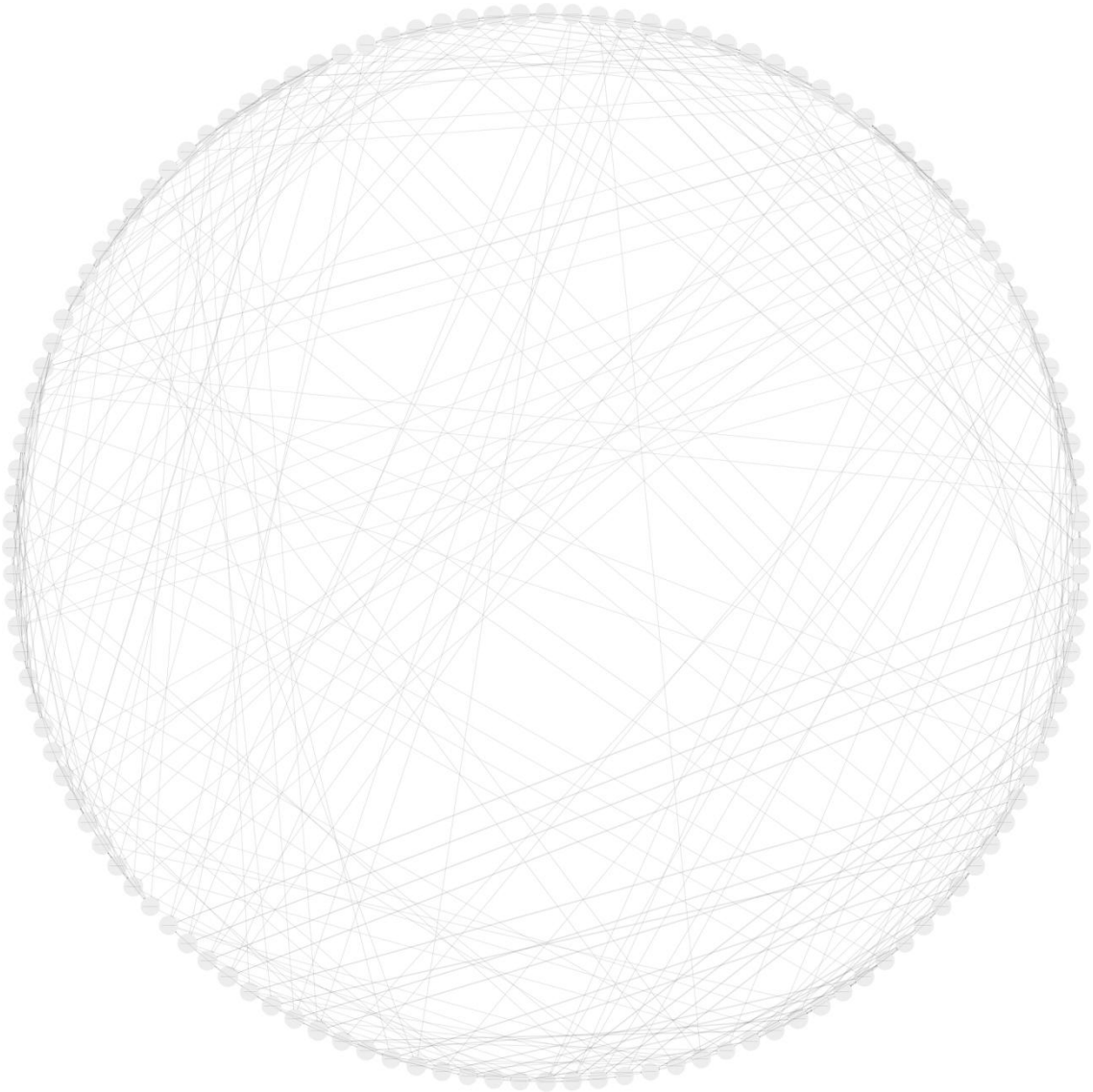
APÊNDICE G – SISTEMA DE TRANSIÇÃO DE ESTADOS DA TRADUÇÃO DE FAS6C.SAN PARA FAS6C.SMV



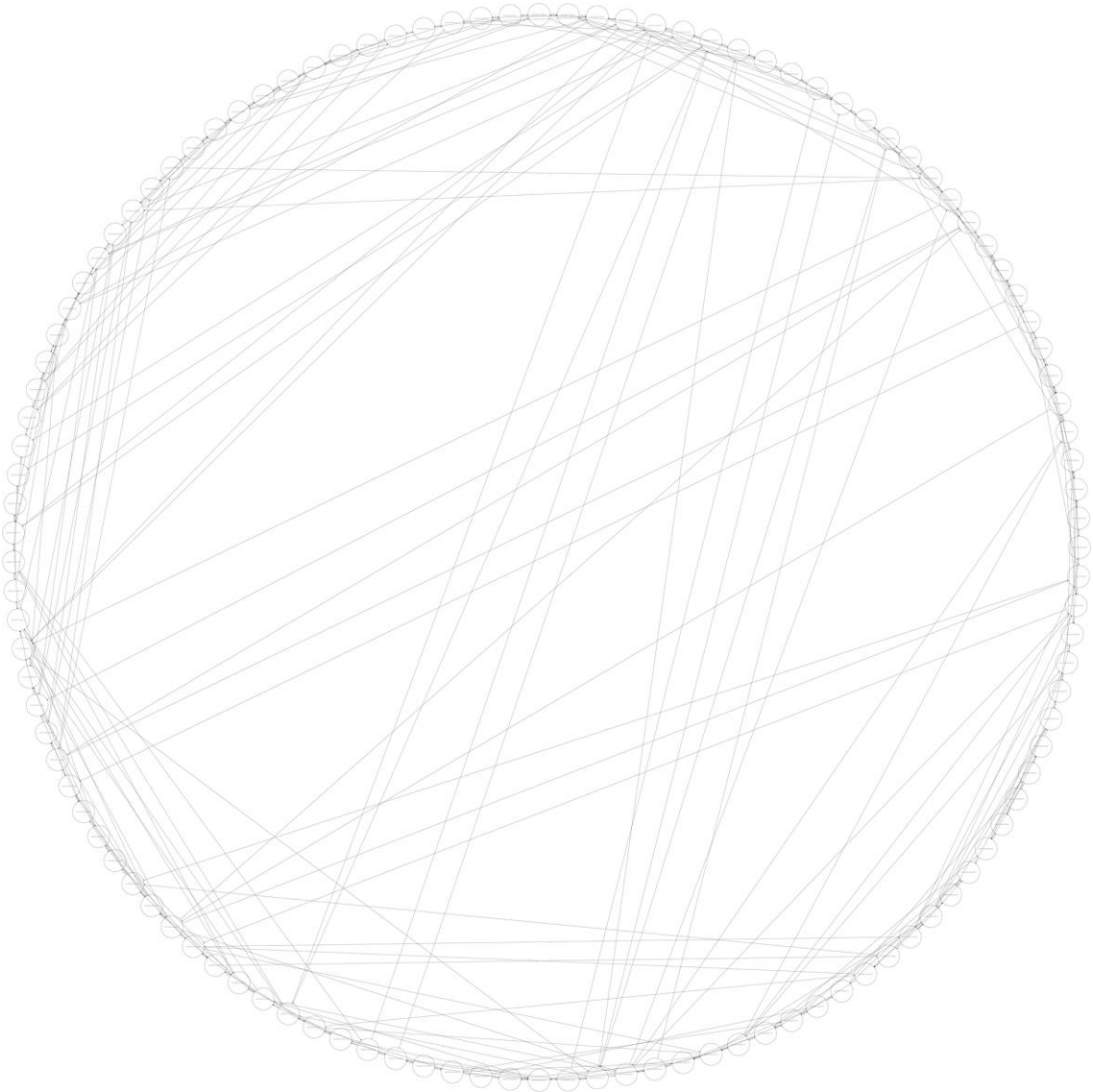
APÊNDICE H – CADEIA DE MARKOV EQUIVALENTE AO MODELO FAS7C.SAN



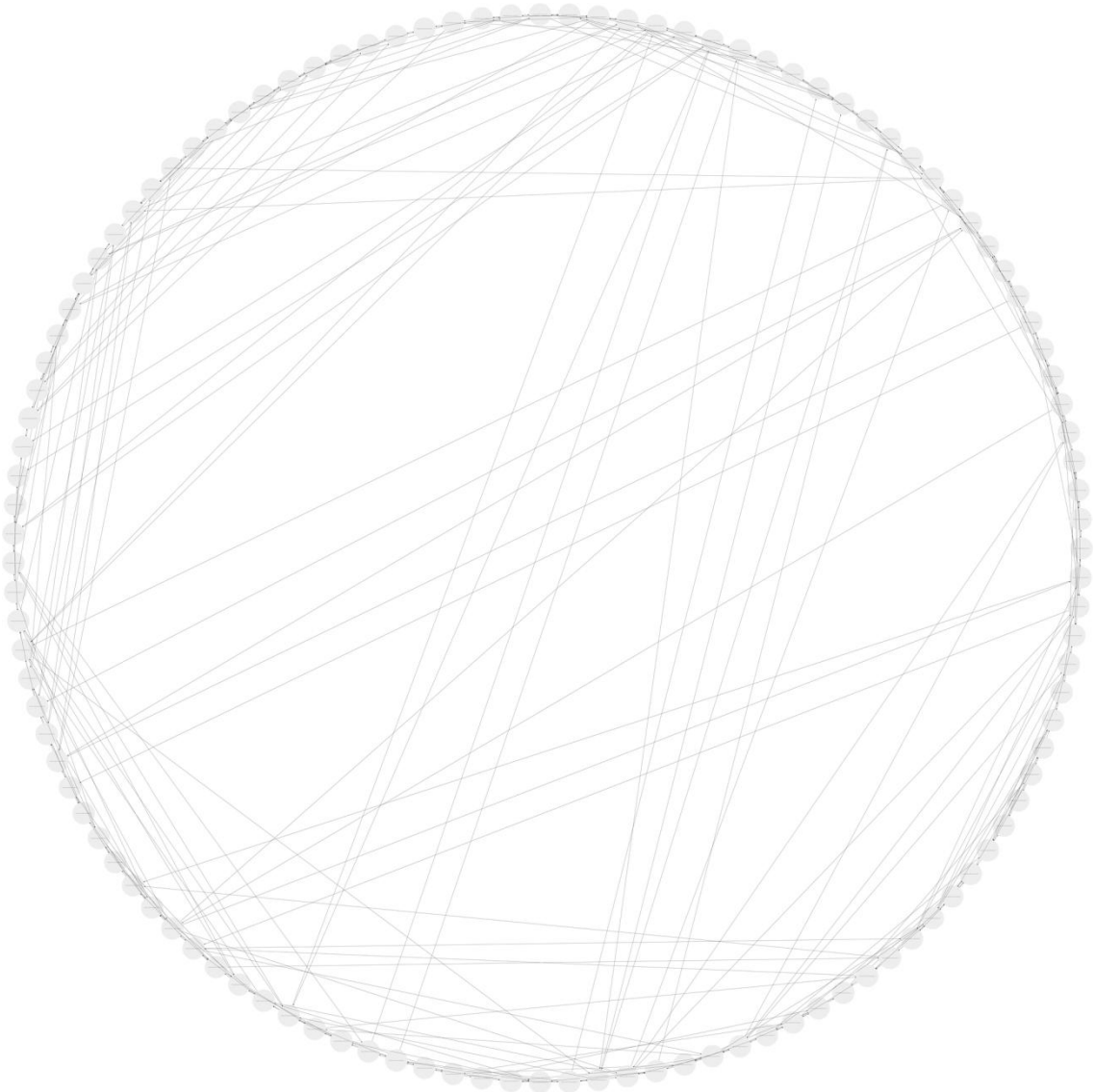
APÊNDICE I – SISTEMA DE TRANSIÇÃO DE ESTADOS DO MODELO FAS7.SMV - TRADUÇÃO DE FAS7C.SAN



APÊNDICE J – CADEIA DE MARKOV EQUIVALENTE AO MODELO PL4.SAN



APÊNDICE K – SISTEMA DE TRANSIÇÃO DE ESTADOS DO MODELO PL4.SMV - TRADUÇÃO DE PL4.SAN



APÊNDICE L – ARQUIVO TEXTUAL DO MODELO PHIL03F.SAN

```
identifiers

  // Number of philosophers
  P = 3;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P2 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 == Thinking) * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Left_2  = (st P1 == Thinking) * lambda;
  Left_to_Right_2     = (st P0 != Left)     * lambda;

events

  loc t_r_0 (Thinking_to_Right_0);
  loc r_l_0 (Right_to_Left_0);
  loc l_t_0 (mu);

  loc t_r_1 (Thinking_to_Right_1);
  loc r_l_1 (Right_to_Left_1);
  loc l_t_1 (mu);

  loc t_l_2 (Thinking_to_Left_2);
  loc l_r_2 (Left_to_Right_2);
  loc r_t_2 (mu);

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

network PHILOSOPHERS (continuous)

  aut P2
    stt Thinking to (Left)     t_l_2
    stt Left     to (Right)    l_r_2
    stt Right    to (Thinking) r_t_2

  aut P1
    stt Thinking to (Right)    t_r_1
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0
```

APÊNDICE M – ARQUIVO TEXTUAL DO MODELO PHIL03F.SMV

```
MODULE ad_P2()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P2 : ad_P2();
  a_P1 : ad_P1();
  a_P0 : ad_P0();

INIT
  ( nb_s_Thinking = i_P )
TRANS
  next_local_step_ad_P2
  |  next_local_step_ad_P1
  |  next_local_step_ad_P0
DEFINE
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Left ? 1 :  (   a_P2.state = s_Right ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Thinking - s_Left
    ( ( a_P2.state = s_Thinking & e_t_l_2 ) & ( next(a_P2.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P2.state = s_Left & e_l_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P2.state = s_Right & e_r_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
```

```
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P2.unchanged
    & a_P0.unchanged
  );
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P2.unchanged
    & a_P1.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 3 );

  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_0 := ( ( ( ( a_P2.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Left_to_Right_2 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  nb_s_Thinking := (
    ( a_P2.state = s_Thinking ? 1 : 0 )  +
    ( a_P1.state = s_Thinking ? 1 : 0 )  +
    ( a_P0.state = s_Thinking ? 1 : 0 )
  );

  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );

  e_r_l_0 := ( i_Right_to_Left_0 > 0 );

  e_l_t_0 := ( i_mu > 0 );
```

```
  e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

  e_r_l_1 := ( i_Right_to_Left_1 > 0 );

  e_l_t_1 := ( i_mu > 0 );

  e_t_l_2 := ( i_Thinking_to_Left_2 > 0 );

  e_l_r_2 := ( i_Left_to_Right_2 > 0 );

  e_r_t_2 := ( i_mu > 0 );
```

APÊNDICE N – ARQUIVO TEXTUAL DO MODELO PHIL04F.SAN

```
identifiers

  // Number of philosophers
  P = 4;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P3 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 != Left)     * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Right_2 = (st P3 == Thinking) * lambda;
  Right_to_Left_2     = (st P1 == Thinking) * lambda;

  Thinking_to_Left_3  = (st P2 == Thinking) * lambda;
  Left_to_Right_3     = (st P0 != Left)     * lambda;

events

  loc t_r_0 (Thinking_to_Right_0);
  loc r_l_0 (Right_to_Left_0);
  loc l_t_0 (mu);

  loc t_r_1 (Thinking_to_Right_1);
  loc r_l_1 (Right_to_Left_1);
  loc l_t_1 (mu);

  loc t_r_2 (Thinking_to_Right_2);
  loc r_l_2 (Right_to_Left_2);
  loc l_t_2 (mu);

  loc t_l_3 (Thinking_to_Left_3);
  loc l_r_3 (Left_to_Right_3);
  loc r_t_3 (mu);

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

network PHILOSOPHERS (continuous)

  aut P3
    stt Thinking to (Left)     t_l_3
    stt Left     to (Right)    l_r_3
    stt Right    to (Thinking) r_t_3

  aut P2
    stt Thinking to (Right)    t_r_2
    stt Right    to (Left)     r_l_2
    stt Left     to (Thinking) l_t_2

  aut P1
    stt Thinking to (Right)    t_r_1
```

```
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0
```

APÊNDICE O – ARQUIVO TEXTUAL DO MODELO PHIL04F.SMV

```
MODULE ad_P3()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P3 : ad_P3();
  a_P2 : ad_P2();
  a_P1 : ad_P1();
  a_P0 : ad_P0();

INIT
  ( nb_s_Thinking = i_P )
TRANS
  next_local_step_ad_P3
  |  next_local_step_ad_P2
  |  next_local_step_ad_P1
  |  next_local_step_ad_P0
DEFINE
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Left ? 1 :  (   a_P3.state = s_Right ? 2 :  -1  )  )   );
  -- ================== ad_P3 automata local steps section ==================
  local_step_ad_P3 := (
    -- connection s_Thinking - s_Left
    ( ( a_P3.state = s_Thinking & e_t_l_3 ) & ( next(a_P3.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P3.state = s_Left & e_l_r_3 ) & ( next(a_P3.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P3.state = s_Right & e_r_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
```

```
    -- connection s_Thinking - s_Right
    ( ( a_P2.state = s_Thinking & e_t_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P2.state = s_Right & e_r_l_2 ) & ( next(a_P2.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P3.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P0.unchanged
  );
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 4 );

  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);
```

```
  i_Right_to_Left_0 := ( ( ( ( a_P3.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_2 := ( ( ( ( a_P3.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Right_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Left_3 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Left_to_Right_3 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  nb_s_Thinking := (
    ( a_P3.state = s_Thinking ? 1 : 0 )  +
    ( a_P2.state = s_Thinking ? 1 : 0 )  +
    ( a_P1.state = s_Thinking ? 1 : 0 )  +
    ( a_P0.state = s_Thinking ? 1 : 0 )
  );

  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );

  e_r_l_0 := ( i_Right_to_Left_0 > 0 );

  e_l_t_0 := ( i_mu > 0 );

  e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

  e_r_l_1 := ( i_Right_to_Left_1 > 0 );

  e_l_t_1 := ( i_mu > 0 );

  e_t_r_2 := ( i_Thinking_to_Right_2 > 0 );

  e_r_l_2 := ( i_Right_to_Left_2 > 0 );

  e_l_t_2 := ( i_mu > 0 );

  e_t_l_3 := ( i_Thinking_to_Left_3 > 0 );

  e_l_r_3 := ( i_Left_to_Right_3 > 0 );

  e_r_t_3 := ( i_mu > 0 );
```

APÊNDICE P – ARQUIVO TEXTUAL DO MODELO PHIL10.SAN

```
identifiers

  // Number of philosophers
  P = 10;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P9 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 != Left)     * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Right_2 = (st P3 != Left)     * lambda;
  Right_to_Left_2     = (st P1 == Thinking) * lambda;

  Thinking_to_Right_3 = (st P4 != Left)     * lambda;
  Right_to_Left_3     = (st P2 == Thinking) * lambda;

  Thinking_to_Right_4 = (st P5 != Left)     * lambda;
  Right_to_Left_4     = (st P3 == Thinking) * lambda;

  Thinking_to_Right_5 = (st P6 != Left)     * lambda;
  Right_to_Left_5     = (st P4 == Thinking) * lambda;

  Thinking_to_Right_6 = (st P7 != Left)     * lambda;
  Right_to_Left_6     = (st P5 == Thinking) * lambda;

  Thinking_to_Right_7 = (st P8 != Left)     * lambda;
  Right_to_Left_7     = (st P6 == Thinking) * lambda;

  Thinking_to_Right_8 = (st P9 == Thinking) * lambda;
  Right_to_Left_8     = (st P7 == Thinking) * lambda;

  Thinking_to_Left_9  = (st P8 == Thinking) * lambda;
  Left_to_Right_9     = (st P0 != Left)     * lambda;

events

  loc t_r_0 (Thinking_to_Right_0);
  loc r_l_0 (Right_to_Left_0);
  loc l_t_0 (mu);

  loc t_r_1 (Thinking_to_Right_1);
  loc r_l_1 (Right_to_Left_1);
  loc l_t_1 (mu);

  loc t_r_2 (Thinking_to_Right_2);
  loc r_l_2 (Right_to_Left_2);
  loc l_t_2 (mu);

  loc t_r_3 (Thinking_to_Right_3);
  loc r_l_3 (Right_to_Left_3);
```

```
  loc l_t_3 (mu);

  loc t_r_4 (Thinking_to_Right_4);
  loc r_l_4 (Right_to_Left_4);
  loc l_t_4 (mu);

  loc t_r_5 (Thinking_to_Right_5);
  loc r_l_5 (Right_to_Left_5);
  loc l_t_5 (mu);

  loc t_r_6 (Thinking_to_Right_6);
  loc r_l_6 (Right_to_Left_6);
  loc l_t_6 (mu);

  loc t_r_7 (Thinking_to_Right_7);
  loc r_l_7 (Right_to_Left_7);
  loc l_t_7 (mu);

  loc t_r_8 (Thinking_to_Right_8);
  loc r_l_8 (Right_to_Left_8);
  loc l_t_8 (mu);

  loc t_l_9 (Thinking_to_Left_9);
  loc l_r_9 (Left_to_Right_9);
  loc r_t_9 (mu);

reachability = ((nb Thinking) == P);

network PHILOSOPHERS (continuous)

  aut P9
    stt Thinking to (Left)     t_l_9
    stt Left     to (Right)    l_r_9
    stt Right    to (Thinking) r_t_9

  aut P8
    stt Thinking to (Right)    t_r_8
    stt Right    to (Left)     r_l_8
    stt Left     to (Thinking) l_t_8

  aut P7
    stt Thinking to (Right)    t_r_7
    stt Right    to (Left)     r_l_7
    stt Left     to (Thinking) l_t_7

  aut P6
    stt Thinking to (Right)    t_r_6
    stt Right    to (Left)     r_l_6
    stt Left     to (Thinking) l_t_6

  aut P5
    stt Thinking to (Right)    t_r_5
    stt Right    to (Left)     r_l_5
    stt Left     to (Thinking) l_t_5

  aut P4
    stt Thinking to (Right)    t_r_4
    stt Right    to (Left)     r_l_4
    stt Left     to (Thinking) l_t_4
```

```
  aut P3
    stt Thinking to (Right)    t_r_3
    stt Right    to (Left)     r_l_3
    stt Left     to (Thinking) l_t_3

  aut P2
    stt Thinking to (Right)    t_r_2
    stt Right    to (Left)     r_l_2
    stt Left     to (Thinking) l_t_2

  aut P1
    stt Thinking to (Right)    t_r_1
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0
```

APÊNDICE Q – ARQUIVO TEXTUAL DO MODELO PHIL10.SMV

```
MODULE ad_P9()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P8()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P7()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P6()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P5()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P4()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P3()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P9 : ad_P9();
  a_P8 : ad_P8();
  a_P7 : ad_P7();
```

```
  a_P6 : ad_P6();
  a_P5 : ad_P5();
  a_P4 : ad_P4();
  a_P3 : ad_P3();
  a_P2 : ad_P2();
  a_P1 : ad_P1();
  a_P0 : ad_P0();

INIT
  ( nb_s_Thinking = i_P )
TRANS
  next_local_step_ad_P9
  |  next_local_step_ad_P8
  |  next_local_step_ad_P7
  |  next_local_step_ad_P6
  |  next_local_step_ad_P5
  |  next_local_step_ad_P4
  |  next_local_step_ad_P3
  |  next_local_step_ad_P2
  |  next_local_step_ad_P1
  |  next_local_step_ad_P0
DEFINE
  a_P9_getCurrentIndex := (   a_P9.state = s_Thinking ? 0 :  (   a_P9.state =
s_Left ? 1 :  (   a_P9.state = s_Right ? 2 :  -1  )  )   );
  -- ================= ad_P9 automata local steps section =================
  local_step_ad_P9 := (
    -- connection s_Thinking - s_Left
    ( ( a_P9.state = s_Thinking & e_t_l_9 ) & ( next(a_P9.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P9.state = s_Left & e_l_r_9 ) & ( next(a_P9.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P9.state = s_Right & e_r_t_9 ) & ( next(a_P9.state) = s_Thinking ) )
  );
  next_local_step_ad_P9 := ( local_step_ad_P9
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P8_getCurrentIndex := (   a_P8.state = s_Thinking ? 0 :  (   a_P8.state =
s_Right ? 1 :  (   a_P8.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P8 automata local steps section =================
  local_step_ad_P8 := (
    -- connection s_Thinking - s_Right
    ( ( a_P8.state = s_Thinking & e_t_r_8 ) & ( next(a_P8.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P8.state = s_Right & e_r_l_8 ) & ( next(a_P8.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P8.state = s_Left & e_l_t_8 ) & ( next(a_P8.state) = s_Thinking ) )
  );
```

```
  next_local_step_ad_P8 := ( local_step_ad_P8
    & a_P9.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P7_getCurrentIndex := (   a_P7.state = s_Thinking ? 0 :  (   a_P7.state =
s_Right ? 1 :  (   a_P7.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P7 automata local steps section ==================
  local_step_ad_P7 := (
    -- connection s_Thinking - s_Right
    ( ( a_P7.state = s_Thinking & e_t_r_7 ) & ( next(a_P7.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P7.state = s_Right & e_r_l_7 ) & ( next(a_P7.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P7.state = s_Left & e_l_t_7 ) & ( next(a_P7.state) = s_Thinking ) )
  );
  next_local_step_ad_P7 := ( local_step_ad_P7
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P6_getCurrentIndex := (   a_P6.state = s_Thinking ? 0 :  (   a_P6.state =
s_Right ? 1 :  (   a_P6.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P6 automata local steps section ==================
  local_step_ad_P6 := (
    -- connection s_Thinking - s_Right
    ( ( a_P6.state = s_Thinking & e_t_r_6 ) & ( next(a_P6.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P6.state = s_Right & e_r_l_6 ) & ( next(a_P6.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P6.state = s_Left & e_l_t_6 ) & ( next(a_P6.state) = s_Thinking ) )
  );
  next_local_step_ad_P6 := ( local_step_ad_P6
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
```

```
  a_P5_getCurrentIndex := (   a_P5.state = s_Thinking ? 0 :  (   a_P5.state =
s_Right ? 1 :  (   a_P5.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P5 automata local steps section =================
  local_step_ad_P5 := (
    -- connection s_Thinking - s_Right
    ( ( a_P5.state = s_Thinking & e_t_r_5 ) & ( next(a_P5.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P5.state = s_Right & e_r_l_5 ) & ( next(a_P5.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P5.state = s_Left & e_l_t_5 ) & ( next(a_P5.state) = s_Thinking ) )
  );
  next_local_step_ad_P5 := ( local_step_ad_P5
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P4_getCurrentIndex := (   a_P4.state = s_Thinking ? 0 :  (   a_P4.state =
s_Right ? 1 :  (   a_P4.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P4 automata local steps section =================
  local_step_ad_P4 := (
    -- connection s_Thinking - s_Right
    ( ( a_P4.state = s_Thinking & e_t_r_4 ) & ( next(a_P4.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P4.state = s_Right & e_r_l_4 ) & ( next(a_P4.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P4.state = s_Left & e_l_t_4 ) & ( next(a_P4.state) = s_Thinking ) )
  );
  next_local_step_ad_P4 := ( local_step_ad_P4
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Right ? 1 :  (   a_P3.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P3 automata local steps section =================
  local_step_ad_P3 := (
    -- connection s_Thinking - s_Right
    ( ( a_P3.state = s_Thinking & e_t_r_3 ) & ( next(a_P3.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P3.state = s_Right & e_r_l_3 ) & ( next(a_P3.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
```

```
    ( ( a_P3.state = s_Left & e_l_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Thinking - s_Right
    ( ( a_P2.state = s_Thinking & e_t_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P2.state = s_Right & e_r_l_2 ) & ( next(a_P2.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P1.unchanged
    & a_P0.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
```

```
    & a_P0.unchanged
  );
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P9.unchanged
    & a_P8.unchanged
    & a_P7.unchanged
    & a_P6.unchanged
    & a_P5.unchanged
    & a_P4.unchanged
    & a_P3.unchanged
    & a_P2.unchanged
    & a_P1.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 10 );

  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_0 := ( ( ( ( a_P9.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_2 := ( ( ( ( a_P3.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_3 := ( ( ( ( a_P4.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_3 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_4 := ( ( ( ( a_P5.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_4 := ( ( ( ( a_P3.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );
```

```
  i_Thinking_to_Right_5 := ( ( ( ( a_P6.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_5 := ( ( ( ( a_P4.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_6 := ( ( ( ( a_P7.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_6 := ( ( ( ( a_P5.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_7 := ( ( ( ( a_P8.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_7 := ( ( ( ( a_P6.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_8 := ( ( ( ( a_P9.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Right_to_Left_8 := ( ( ( ( a_P7.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Left_9 := ( ( ( ( a_P8.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Left_to_Right_9 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  nb_s_Thinking := (
    ( a_P9.state = s_Thinking ? 1 : 0 )  +
    ( a_P8.state = s_Thinking ? 1 : 0 )  +
    ( a_P7.state = s_Thinking ? 1 : 0 )  +
    ( a_P6.state = s_Thinking ? 1 : 0 )  +
    ( a_P5.state = s_Thinking ? 1 : 0 )  +
    ( a_P4.state = s_Thinking ? 1 : 0 )  +
    ( a_P3.state = s_Thinking ? 1 : 0 )  +
    ( a_P2.state = s_Thinking ? 1 : 0 )  +
    ( a_P1.state = s_Thinking ? 1 : 0 )  +
    ( a_P0.state = s_Thinking ? 1 : 0 )
  );

  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );

  e_r_l_0 := ( i_Right_to_Left_0 > 0 );

  e_l_t_0 := ( i_mu > 0 );

  e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

  e_r_l_1 := ( i_Right_to_Left_1 > 0 );

  e_l_t_1 := ( i_mu > 0 );

  e_t_r_2 := ( i_Thinking_to_Right_2 > 0 );

  e_r_l_2 := ( i_Right_to_Left_2 > 0 );

  e_l_t_2 := ( i_mu > 0 );

  e_t_r_3 := ( i_Thinking_to_Right_3 > 0 );
```

```
e_r_l_3 := ( i_Right_to_Left_3 > 0 );

e_l_t_3 := ( i_mu > 0 );

e_t_r_4 := ( i_Thinking_to_Right_4 > 0 );

e_r_l_4 := ( i_Right_to_Left_4 > 0 );

e_l_t_4 := ( i_mu > 0 );

e_t_r_5 := ( i_Thinking_to_Right_5 > 0 );

e_r_l_5 := ( i_Right_to_Left_5 > 0 );

e_l_t_5 := ( i_mu > 0 );

e_t_r_6 := ( i_Thinking_to_Right_6 > 0 );

e_r_l_6 := ( i_Right_to_Left_6 > 0 );

e_l_t_6 := ( i_mu > 0 );

e_t_r_7 := ( i_Thinking_to_Right_7 > 0 );

e_r_l_7 := ( i_Right_to_Left_7 > 0 );

e_l_t_7 := ( i_mu > 0 );

e_t_r_8 := ( i_Thinking_to_Right_8 > 0 );

e_r_l_8 := ( i_Right_to_Left_8 > 0 );

e_l_t_8 := ( i_mu > 0 );

e_t_l_9 := ( i_Thinking_to_Left_9 > 0 );

e_l_r_9 := ( i_Left_to_Right_9 > 0 );

e_r_t_9 := ( i_mu > 0 );
```

APÊNDICE R – ARQUIVO TEXTUAL DO MODELO PHIL12.SAN

```
identifiers

  // Number of philosophers
  P = 12;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P11 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 != Left)     * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Right_2 = (st P3 != Left)     * lambda;
  Right_to_Left_2     = (st P1 == Thinking) * lambda;

  Thinking_to_Right_3 = (st P4 != Left)     * lambda;
  Right_to_Left_3     = (st P2 == Thinking) * lambda;

  Thinking_to_Right_4 = (st P5 != Left)     * lambda;
  Right_to_Left_4     = (st P3 == Thinking) * lambda;

  Thinking_to_Right_5 = (st P6 != Left)     * lambda;
  Right_to_Left_5     = (st P4 == Thinking) * lambda;

  Thinking_to_Right_6 = (st P7 != Left)     * lambda;
  Right_to_Left_6     = (st P5 == Thinking) * lambda;

  Thinking_to_Right_7 = (st P8 != Left)     * lambda;
  Right_to_Left_7     = (st P6 == Thinking) * lambda;

  Thinking_to_Right_8 = (st P9 != Left)     * lambda;
  Right_to_Left_8     = (st P7 == Thinking) * lambda;

  Thinking_to_Right_9 = (st P10 != Left)     * lambda;
  Right_to_Left_9     = (st P8 == Thinking) * lambda;

  Thinking_to_Right_10 = (st P11 == Thinking) * lambda;
  Right_to_Left_10     = (st P9 == Thinking) * lambda;

  Thinking_to_Left_11  = (st P10 == Thinking) * lambda;
  Left_to_Right_11     = (st P0 != Left)     * lambda;

events

  loc t_r_0 (Thinking_to_Right_0);
  loc r_l_0 (Right_to_Left_0);
  loc l_t_0 (mu);

  loc t_r_1 (Thinking_to_Right_1);
  loc r_l_1 (Right_to_Left_1);
  loc l_t_1 (mu);
```

```
  loc t_r_2 (Thinking_to_Right_2);
  loc r_l_2 (Right_to_Left_2);
  loc l_t_2 (mu);

  loc t_r_3 (Thinking_to_Right_3);
  loc r_l_3 (Right_to_Left_3);
  loc l_t_3 (mu);

  loc t_r_4 (Thinking_to_Right_4);
  loc r_l_4 (Right_to_Left_4);
  loc l_t_4 (mu);

  loc t_r_5 (Thinking_to_Right_5);
  loc r_l_5 (Right_to_Left_5);
  loc l_t_5 (mu);

  loc t_r_6 (Thinking_to_Right_6);
  loc r_l_6 (Right_to_Left_6);
  loc l_t_6 (mu);

  loc t_r_7 (Thinking_to_Right_7);
  loc r_l_7 (Right_to_Left_7);
  loc l_t_7 (mu);

  loc t_r_8 (Thinking_to_Right_8);
  loc r_l_8 (Right_to_Left_8);
  loc l_t_8 (mu);

  loc t_r_9 (Thinking_to_Right_9);
  loc r_l_9 (Right_to_Left_9);
  loc l_t_9 (mu);

  loc t_r_10 (Thinking_to_Right_10);
  loc r_l_10 (Right_to_Left_10);
  loc l_t_10 (mu);

  loc t_l_11 (Thinking_to_Left_11);
  loc l_r_11 (Left_to_Right_11);
  loc r_t_11 (mu);

partial reachability = ((st P0 == Thinking) && (st P1 == Thinking) && (st P2 ==
Thinking) && (st P3 == Thinking) && (st P4 == Thinking) && (st P5 == Thinking) &&
(st P6 == Thinking) && (st P7 == Thinking) && (st P8 == Thinking) && (st P9 ==
Thinking) && (st P10 == Thinking) && (st P11 == Thinking));

network PHIL12f (continuous)

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0

  aut P1
    stt Thinking to (Right)    t_r_1
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P2
    stt Thinking to (Right)    t_r_2
```

```
  stt Right    to (Left)     r_l_2
  stt Left     to (Thinking) l_t_2

aut P3
  stt Thinking to (Right)    t_r_3
  stt Right    to (Left)     r_l_3
  stt Left     to (Thinking) l_t_3

aut P4
  stt Thinking to (Right)    t_r_4
  stt Right    to (Left)     r_l_4
  stt Left     to (Thinking) l_t_4

aut P5
  stt Thinking to (Right)    t_r_5
  stt Right    to (Left)     r_l_5
  stt Left     to (Thinking) l_t_5

aut P6
  stt Thinking to (Right)    t_r_6
  stt Right    to (Left)     r_l_6
  stt Left     to (Thinking) l_t_6

aut P7
  stt Thinking to (Right)    t_r_7
  stt Right    to (Left)     r_l_7
  stt Left     to (Thinking) l_t_7

aut P8
  stt Thinking to (Right)    t_r_8
  stt Right    to (Left)     r_l_8
  stt Left     to (Thinking) l_t_8

aut P9
  stt Thinking to (Right)    t_r_9
  stt Right    to (Left)     r_l_9
  stt Left     to (Thinking) l_t_9

aut P10
  stt Thinking to (Right)    t_r_10
  stt Right    to (Left)     r_l_10
  stt Left     to (Thinking) l_t_10

aut P11
  stt Thinking to (Left)     t_l_11
  stt Left     to (Right)    l_r_11
  stt Right    to (Thinking) r_t_11
```

APÊNDICE S – ARQUIVO TEXTUAL DO MODELO PHIL12.SMV

```
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P3()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P4()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P5()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P6()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P7()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P8()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P9()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P10()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
```

```
MODULE ad_P11()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P0 : ad_P0();
  a_P1 : ad_P1();
  a_P2 : ad_P2();
  a_P3 : ad_P3();
  a_P4 : ad_P4();
  a_P5 : ad_P5();
  a_P6 : ad_P6();
  a_P7 : ad_P7();
  a_P8 : ad_P8();
  a_P9 : ad_P9();
  a_P10 : ad_P10();
  a_P11 : ad_P11();

INIT
  ( ( ( ( ( ( ( ( ( ( a_P0.state = s_Thinking ) & ( a_P1.state = s_Thinking ) )
& ( a_P2.state = s_Thinking ) ) & ( a_P3.state = s_Thinking ) ) & ( a_P4.state =
s_Thinking ) ) & ( a_P5.state = s_Thinking ) ) & ( a_P6.state = s_Thinking ) ) & (
a_P7.state = s_Thinking ) ) & ( a_P8.state = s_Thinking ) ) & ( a_P9.state =
s_Thinking ) ) & ( a_P10.state = s_Thinking ) ) & ( a_P11.state = s_Thinking )
TRANS
  next_local_step_ad_P0
  |  next_local_step_ad_P1
  |  next_local_step_ad_P2
  |  next_local_step_ad_P3
  |  next_local_step_ad_P4
  |  next_local_step_ad_P5
  |  next_local_step_ad_P6
  |  next_local_step_ad_P7
  |  next_local_step_ad_P8
  |  next_local_step_ad_P9
  |  next_local_step_ad_P10
  |  next_local_step_ad_P11
DEFINE
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
```

```
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P0.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Thinking - s_Right
    ( ( a_P2.state = s_Thinking & e_t_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P2.state = s_Right & e_r_l_2 ) & ( next(a_P2.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
```

```
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Right ? 1 :  (   a_P3.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P3 automata local steps section =================
  local_step_ad_P3 := (
    -- connection s_Thinking - s_Right
    ( ( a_P3.state = s_Thinking & e_t_r_3 ) & ( next(a_P3.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P3.state = s_Right & e_r_l_3 ) & ( next(a_P3.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P3.state = s_Left & e_l_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P4_getCurrentIndex := (   a_P4.state = s_Thinking ? 0 :  (   a_P4.state =
s_Right ? 1 :  (   a_P4.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P4 automata local steps section =================
  local_step_ad_P4 := (
    -- connection s_Thinking - s_Right
    ( ( a_P4.state = s_Thinking & e_t_r_4 ) & ( next(a_P4.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P4.state = s_Right & e_r_l_4 ) & ( next(a_P4.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P4.state = s_Left & e_l_t_4 ) & ( next(a_P4.state) = s_Thinking ) )
  );
  next_local_step_ad_P4 := ( local_step_ad_P4
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P5_getCurrentIndex := (   a_P5.state = s_Thinking ? 0 :  (   a_P5.state =
s_Right ? 1 :  (   a_P5.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P5 automata local steps section =================
  local_step_ad_P5 := (
    -- connection s_Thinking - s_Right
    ( ( a_P5.state = s_Thinking & e_t_r_5 ) & ( next(a_P5.state) = s_Right ) )
    |
```

```
    -- connection s_Right - s_Left
    ( ( a_P5.state = s_Right & e_r_l_5 ) & ( next(a_P5.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P5.state = s_Left & e_l_t_5 ) & ( next(a_P5.state) = s_Thinking ) )
  );
  next_local_step_ad_P5 := ( local_step_ad_P5
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P6_getCurrentIndex := (   a_P6.state = s_Thinking ? 0 :  (   a_P6.state =
s_Right ? 1 :  (   a_P6.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P6 automata local steps section ==================
  local_step_ad_P6 := (
    -- connection s_Thinking - s_Right
    ( ( a_P6.state = s_Thinking & e_t_r_6 ) & ( next(a_P6.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P6.state = s_Right & e_r_l_6 ) & ( next(a_P6.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P6.state = s_Left & e_l_t_6 ) & ( next(a_P6.state) = s_Thinking ) )
  );
  next_local_step_ad_P6 := ( local_step_ad_P6
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P7_getCurrentIndex := (   a_P7.state = s_Thinking ? 0 :  (   a_P7.state =
s_Right ? 1 :  (   a_P7.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P7 automata local steps section ==================
  local_step_ad_P7 := (
    -- connection s_Thinking - s_Right
    ( ( a_P7.state = s_Thinking & e_t_r_7 ) & ( next(a_P7.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P7.state = s_Right & e_r_l_7 ) & ( next(a_P7.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P7.state = s_Left & e_l_t_7 ) & ( next(a_P7.state) = s_Thinking ) )
  );
  next_local_step_ad_P7 := ( local_step_ad_P7
```

```
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P8_getCurrentIndex := (   a_P8.state = s_Thinking ? 0 :  (   a_P8.state =
s_Right ? 1 :  (   a_P8.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P8 automata local steps section ==================
  local_step_ad_P8 := (
    -- connection s_Thinking - s_Right
    ( ( a_P8.state = s_Thinking & e_t_r_8 ) & ( next(a_P8.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P8.state = s_Right & e_r_l_8 ) & ( next(a_P8.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P8.state = s_Left & e_l_t_8 ) & ( next(a_P8.state) = s_Thinking ) )
  );
  next_local_step_ad_P8 := ( local_step_ad_P8
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P9_getCurrentIndex := (   a_P9.state = s_Thinking ? 0 :  (   a_P9.state =
s_Right ? 1 :  (   a_P9.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P9 automata local steps section ==================
  local_step_ad_P9 := (
    -- connection s_Thinking - s_Right
    ( ( a_P9.state = s_Thinking & e_t_r_9 ) & ( next(a_P9.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P9.state = s_Right & e_r_l_9 ) & ( next(a_P9.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P9.state = s_Left & e_l_t_9 ) & ( next(a_P9.state) = s_Thinking ) )
  );
  next_local_step_ad_P9 := ( local_step_ad_P9
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
```

```
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
  );
  a_P10_getCurrentIndex := (   a_P10.state = s_Thinking ? 0 :  (   a_P10.state =
s_Right ? 1 :  (   a_P10.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P10 automata local steps section ==================
  local_step_ad_P10 := (
    -- connection s_Thinking - s_Right
    ( ( a_P10.state = s_Thinking & e_t_r_10 ) & ( next(a_P10.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P10.state = s_Right & e_r_l_10 ) & ( next(a_P10.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P10.state = s_Left & e_l_t_10 ) & ( next(a_P10.state) = s_Thinking ) )
  );
  next_local_step_ad_P10 := ( local_step_ad_P10
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P11.unchanged
  );
  a_P11_getCurrentIndex := (   a_P11.state = s_Thinking ? 0 :  (   a_P11.state =
s_Left ? 1 :  (   a_P11.state = s_Right ? 2 :  -1  )  )   );
  -- ================== ad_P11 automata local steps section ==================
  local_step_ad_P11 := (
    -- connection s_Thinking - s_Left
    ( ( a_P11.state = s_Thinking & e_t_l_11 ) & ( next(a_P11.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P11.state = s_Left & e_l_r_11 ) & ( next(a_P11.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P11.state = s_Right & e_r_t_11 ) & ( next(a_P11.state) = s_Thinking ) )
  );
  next_local_step_ad_P11 := ( local_step_ad_P11
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 12 );
```

```
  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_0 := ( ( ( ( a_P11.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_2 := ( ( ( ( a_P3.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_3 := ( ( ( ( a_P4.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_3 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_4 := ( ( ( ( a_P5.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_4 := ( ( ( ( a_P3.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_5 := ( ( ( ( a_P6.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_5 := ( ( ( ( a_P4.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_6 := ( ( ( ( a_P7.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_6 := ( ( ( ( a_P5.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_7 := ( ( ( ( a_P8.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_7 := ( ( ( ( a_P6.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_8 := ( ( ( ( a_P9.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_8 := ( ( ( ( a_P7.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_9 := ( ( ( ( a_P10.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_9 := ( ( ( ( a_P8.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_10 := ( ( ( ( a_P11.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );
```

```
  i_Right_to_Left_10 := ( ( ( ( a_P9.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Left_11 := ( ( ( ( a_P10.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Left_to_Right_11 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );

  e_r_l_0 := ( i_Right_to_Left_0 > 0 );

  e_l_t_0 := ( i_mu > 0 );

  e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

  e_r_l_1 := ( i_Right_to_Left_1 > 0 );

  e_l_t_1 := ( i_mu > 0 );

  e_t_r_2 := ( i_Thinking_to_Right_2 > 0 );

  e_r_l_2 := ( i_Right_to_Left_2 > 0 );

  e_l_t_2 := ( i_mu > 0 );

  e_t_r_3 := ( i_Thinking_to_Right_3 > 0 );

  e_r_l_3 := ( i_Right_to_Left_3 > 0 );

  e_l_t_3 := ( i_mu > 0 );

  e_t_r_4 := ( i_Thinking_to_Right_4 > 0 );

  e_r_l_4 := ( i_Right_to_Left_4 > 0 );

  e_l_t_4 := ( i_mu > 0 );

  e_t_r_5 := ( i_Thinking_to_Right_5 > 0 );

  e_r_l_5 := ( i_Right_to_Left_5 > 0 );

  e_l_t_5 := ( i_mu > 0 );

  e_t_r_6 := ( i_Thinking_to_Right_6 > 0 );

  e_r_l_6 := ( i_Right_to_Left_6 > 0 );

  e_l_t_6 := ( i_mu > 0 );

  e_t_r_7 := ( i_Thinking_to_Right_7 > 0 );

  e_r_l_7 := ( i_Right_to_Left_7 > 0 );

  e_l_t_7 := ( i_mu > 0 );

  e_t_r_8 := ( i_Thinking_to_Right_8 > 0 );
```

```
e_r_l_8 := ( i_Right_to_Left_8 > 0 );

e_l_t_8 := ( i_mu > 0 );

e_t_r_9 := ( i_Thinking_to_Right_9 > 0 );

e_r_l_9 := ( i_Right_to_Left_9 > 0 );

e_l_t_9 := ( i_mu > 0 );

e_t_r_10 := ( i_Thinking_to_Right_10 > 0 );

e_r_l_10 := ( i_Right_to_Left_10 > 0 );

e_l_t_10 := ( i_mu > 0 );

e_t_l_11 := ( i_Thinking_to_Left_11 > 0 );

e_l_r_11 := ( i_Left_to_Right_11 > 0 );

e_r_t_11 := ( i_mu > 0 );
```

APÊNDICE T – ARQUIVO TEXTUAL DO MODELO PHIL15.SAN

```
identifiers

  // Number of philosophers
  P = 15;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P14 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 != Left)     * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Right_2 = (st P3 != Left)     * lambda;
  Right_to_Left_2     = (st P1 == Thinking) * lambda;

  Thinking_to_Right_3 = (st P4 != Left)     * lambda;
  Right_to_Left_3     = (st P2 == Thinking) * lambda;

  Thinking_to_Right_4 = (st P5 != Left)     * lambda;
  Right_to_Left_4     = (st P3 == Thinking) * lambda;

  Thinking_to_Right_5 = (st P6 != Left)     * lambda;
  Right_to_Left_5     = (st P4 == Thinking) * lambda;

  Thinking_to_Right_6 = (st P7 != Left)     * lambda;
  Right_to_Left_6     = (st P5 == Thinking) * lambda;

  Thinking_to_Right_7 = (st P8 != Left)     * lambda;
  Right_to_Left_7     = (st P6 == Thinking) * lambda;

  Thinking_to_Right_8 = (st P9 != Left)     * lambda;
  Right_to_Left_8     = (st P7 == Thinking) * lambda;

  Thinking_to_Right_9 = (st P10 != Left)     * lambda;
  Right_to_Left_9     = (st P8 == Thinking) * lambda;

  Thinking_to_Right_10 = (st P11 != Left)     * lambda;
  Right_to_Left_10     = (st P9 == Thinking) * lambda;

  Thinking_to_Right_11 = (st P12 != Left)     * lambda;
  Right_to_Left_11     = (st P10 == Thinking) * lambda;

  Thinking_to_Right_12 = (st P13 != Left)     * lambda;
  Right_to_Left_12     = (st P11 == Thinking) * lambda;

  Thinking_to_Right_13 = (st P14 == Thinking) * lambda;
  Right_to_Left_13     = (st P12 == Thinking) * lambda;

  Thinking_to_Left_14  = (st P13 == Thinking) * lambda;
  Left_to_Right_14     = (st P0 != Left)     * lambda;

events
```

```
loc t_r_0 (Thinking_to_Right_0);
loc r_l_0 (Right_to_Left_0);
loc l_t_0 (mu);

loc t_r_1 (Thinking_to_Right_1);
loc r_l_1 (Right_to_Left_1);
loc l_t_1 (mu);

loc t_r_2 (Thinking_to_Right_2);
loc r_l_2 (Right_to_Left_2);
loc l_t_2 (mu);

loc t_r_3 (Thinking_to_Right_3);
loc r_l_3 (Right_to_Left_3);
loc l_t_3 (mu);

loc t_r_4 (Thinking_to_Right_4);
loc r_l_4 (Right_to_Left_4);
loc l_t_4 (mu);

loc t_r_5 (Thinking_to_Right_5);
loc r_l_5 (Right_to_Left_5);
loc l_t_5 (mu);

loc t_r_6 (Thinking_to_Right_6);
loc r_l_6 (Right_to_Left_6);
loc l_t_6 (mu);

loc t_r_7 (Thinking_to_Right_7);
loc r_l_7 (Right_to_Left_7);
loc l_t_7 (mu);

loc t_r_8 (Thinking_to_Right_8);
loc r_l_8 (Right_to_Left_8);
loc l_t_8 (mu);

loc t_r_9 (Thinking_to_Right_9);
loc r_l_9 (Right_to_Left_9);
loc l_t_9 (mu);

loc t_r_10 (Thinking_to_Right_10);
loc r_l_10 (Right_to_Left_10);
loc l_t_10 (mu);

loc t_r_11 (Thinking_to_Right_11);
loc r_l_11 (Right_to_Left_11);
loc l_t_11 (mu);

loc t_r_12 (Thinking_to_Right_12);
loc r_l_12 (Right_to_Left_12);
loc l_t_12 (mu);

loc t_r_13 (Thinking_to_Right_13);
loc r_l_13 (Right_to_Left_13);
loc l_t_13 (mu);

loc t_l_14 (Thinking_to_Left_14);
loc l_r_14 (Left_to_Right_14);
```

```
  loc r_t_14 (mu);

partial reachability = ((st P0 == Thinking) && (st P1 == Thinking) && (st P2 ==
Thinking) && (st P3 == Thinking) && (st P4 == Thinking) && (st P5 == Thinking) &&
(st P6 == Thinking) && (st P7 == Thinking) && (st P8 == Thinking) && (st P9 ==
Thinking) && (st P10 == Thinking) && (st P11 == Thinking) && (st P12 == Thinking)
&& (st P13 == Thinking) && (st P14 == Thinking));

network PHIL15f (continuous)

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0

  aut P1
    stt Thinking to (Right)    t_r_1
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P2
    stt Thinking to (Right)    t_r_2
    stt Right    to (Left)     r_l_2
    stt Left     to (Thinking) l_t_2

  aut P3
    stt Thinking to (Right)    t_r_3
    stt Right    to (Left)     r_l_3
    stt Left     to (Thinking) l_t_3

  aut P4
    stt Thinking to (Right)    t_r_4
    stt Right    to (Left)     r_l_4
    stt Left     to (Thinking) l_t_4

  aut P5
    stt Thinking to (Right)    t_r_5
    stt Right    to (Left)     r_l_5
    stt Left     to (Thinking) l_t_5

  aut P6
    stt Thinking to (Right)    t_r_6
    stt Right    to (Left)     r_l_6
    stt Left     to (Thinking) l_t_6

  aut P7
    stt Thinking to (Right)    t_r_7
    stt Right    to (Left)     r_l_7
    stt Left     to (Thinking) l_t_7

  aut P8
    stt Thinking to (Right)    t_r_8
    stt Right    to (Left)     r_l_8
    stt Left     to (Thinking) l_t_8

  aut P9
    stt Thinking to (Right)    t_r_9
    stt Right    to (Left)     r_l_9
    stt Left     to (Thinking) l_t_9
```

```
aut P10
  stt Thinking to (Right)    t_r_10
  stt Right    to (Left)     r_l_10
  stt Left     to (Thinking) l_t_10

aut P11
  stt Thinking to (Right)    t_r_11
  stt Right    to (Left)     r_l_11
  stt Left     to (Thinking) l_t_11

aut P12
  stt Thinking to (Right)    t_r_12
  stt Right    to (Left)     r_l_12
  stt Left     to (Thinking) l_t_12

aut P13
  stt Thinking to (Right)    t_r_13
  stt Right    to (Left)     r_l_13
  stt Left     to (Thinking) l_t_13

aut P14
  stt Thinking to (Left)     t_l_14
  stt Left     to (Right)    l_r_14
  stt Right    to (Thinking) r_t_14
```

APÊNDICE U – ARQUIVO TEXTUAL DO MODELO PHIL15.SMV

```
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P3()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P4()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P5()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P6()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P7()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P8()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P9()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P10()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
```

```
MODULE ad_P11()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_P12()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_P13()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_P14()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE main
VAR
  a_P0 : ad_P0();
  a_P1 : ad_P1();
  a_P2 : ad_P2();
  a_P3 : ad_P3();
  a_P4 : ad_P4();
  a_P5 : ad_P5();
  a_P6 : ad_P6();
  a_P7 : ad_P7();
  a_P8 : ad_P8();
  a_P9 : ad_P9();
  a_P10 : ad_P10();
  a_P11 : ad_P11();
  a_P12 : ad_P12();
  a_P13 : ad_P13();
  a_P14 : ad_P14();

INIT
  ( ( ( ( ( ( ( ( ( ( ( ( a_P0.state = s_Thinking ) & ( a_P1.state =
s_Thinking ) ) & ( a_P2.state = s_Thinking ) ) & ( a_P3.state = s_Thinking ) ) & (
a_P4.state = s_Thinking ) ) & ( a_P5.state = s_Thinking ) ) & ( a_P6.state =
s_Thinking ) ) & ( a_P7.state = s_Thinking ) ) & ( a_P8.state = s_Thinking ) ) & (
a_P9.state = s_Thinking ) ) & ( a_P10.state = s_Thinking ) ) & ( a_P11.state =
s_Thinking ) ) & ( a_P12.state = s_Thinking ) ) & ( a_P13.state = s_Thinking ) ) &
( a_P14.state = s_Thinking )
TRANS
  next_local_step_ad_P0
  |  next_local_step_ad_P1
  |  next_local_step_ad_P2
  |  next_local_step_ad_P3
  |  next_local_step_ad_P4
  |  next_local_step_ad_P5
  |  next_local_step_ad_P6
  |  next_local_step_ad_P7
  |  next_local_step_ad_P8
  |  next_local_step_ad_P9
  |  next_local_step_ad_P10
  |  next_local_step_ad_P11
```

```
  |  next_local_step_ad_P12
  |  next_local_step_ad_P13
  |  next_local_step_ad_P14
DEFINE
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P0.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
```

```
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Thinking - s_Right
    ( ( a_P2.state = s_Thinking & e_t_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P2.state = s_Right & e_r_l_2 ) & ( next(a_P2.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Right ? 1 :  (   a_P3.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P3 automata local steps section ==================
  local_step_ad_P3 := (
    -- connection s_Thinking - s_Right
    ( ( a_P3.state = s_Thinking & e_t_r_3 ) & ( next(a_P3.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P3.state = s_Right & e_r_l_3 ) & ( next(a_P3.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P3.state = s_Left & e_l_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
```

```
    & a_P14.unchanged
  );
  a_P4_getCurrentIndex := (   a_P4.state = s_Thinking ? 0 :  (   a_P4.state =
s_Right ? 1 :  (   a_P4.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P4 automata local steps section ==================
  local_step_ad_P4 := (
    -- connection s_Thinking - s_Right
    ( ( a_P4.state = s_Thinking & e_t_r_4 ) & ( next(a_P4.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P4.state = s_Right & e_r_l_4 ) & ( next(a_P4.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P4.state = s_Left & e_l_t_4 ) & ( next(a_P4.state) = s_Thinking ) )
  );
  next_local_step_ad_P4 := ( local_step_ad_P4
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P5_getCurrentIndex := (   a_P5.state = s_Thinking ? 0 :  (   a_P5.state =
s_Right ? 1 :  (   a_P5.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P5 automata local steps section ==================
  local_step_ad_P5 := (
    -- connection s_Thinking - s_Right
    ( ( a_P5.state = s_Thinking & e_t_r_5 ) & ( next(a_P5.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P5.state = s_Right & e_r_l_5 ) & ( next(a_P5.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P5.state = s_Left & e_l_t_5 ) & ( next(a_P5.state) = s_Thinking ) )
  );
  next_local_step_ad_P5 := ( local_step_ad_P5
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
```

```
  );
  a_P6_getCurrentIndex := (   a_P6.state = s_Thinking ? 0 :  (   a_P6.state =
s_Right ? 1 :  (   a_P6.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P6 automata local steps section ==================
  local_step_ad_P6 := (
    -- connection s_Thinking - s_Right
    ( ( a_P6.state = s_Thinking & e_t_r_6 ) & ( next(a_P6.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P6.state = s_Right & e_r_l_6 ) & ( next(a_P6.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P6.state = s_Left & e_l_t_6 ) & ( next(a_P6.state) = s_Thinking ) )
  );
  next_local_step_ad_P6 := ( local_step_ad_P6
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P7_getCurrentIndex := (   a_P7.state = s_Thinking ? 0 :  (   a_P7.state =
s_Right ? 1 :  (   a_P7.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P7 automata local steps section ==================
  local_step_ad_P7 := (
    -- connection s_Thinking - s_Right
    ( ( a_P7.state = s_Thinking & e_t_r_7 ) & ( next(a_P7.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P7.state = s_Right & e_r_l_7 ) & ( next(a_P7.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P7.state = s_Left & e_l_t_7 ) & ( next(a_P7.state) = s_Thinking ) )
  );
  next_local_step_ad_P7 := ( local_step_ad_P7
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
```

```
  a_P8_getCurrentIndex := (   a_P8.state = s_Thinking ? 0 :  (   a_P8.state =
s_Right ? 1 :  (   a_P8.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P8 automata local steps section ==================
  local_step_ad_P8 := (
    -- connection s_Thinking - s_Right
    ( ( a_P8.state = s_Thinking & e_t_r_8 ) & ( next(a_P8.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P8.state = s_Right & e_r_l_8 ) & ( next(a_P8.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P8.state = s_Left & e_l_t_8 ) & ( next(a_P8.state) = s_Thinking ) )
  );
  next_local_step_ad_P8 := ( local_step_ad_P8
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P9_getCurrentIndex := (   a_P9.state = s_Thinking ? 0 :  (   a_P9.state =
s_Right ? 1 :  (   a_P9.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P9 automata local steps section ==================
  local_step_ad_P9 := (
    -- connection s_Thinking - s_Right
    ( ( a_P9.state = s_Thinking & e_t_r_9 ) & ( next(a_P9.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P9.state = s_Right & e_r_l_9 ) & ( next(a_P9.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P9.state = s_Left & e_l_t_9 ) & ( next(a_P9.state) = s_Thinking ) )
  );
  next_local_step_ad_P9 := ( local_step_ad_P9
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
```

```
  a_P10_getCurrentIndex := (   a_P10.state = s_Thinking ? 0 :  (   a_P10.state =
s_Right ? 1 :  (   a_P10.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P10 automata local steps section ==================
  local_step_ad_P10 := (
    -- connection s_Thinking - s_Right
    ( ( a_P10.state = s_Thinking & e_t_r_10 ) & ( next(a_P10.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P10.state = s_Right & e_r_l_10 ) & ( next(a_P10.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P10.state = s_Left & e_l_t_10 ) & ( next(a_P10.state) = s_Thinking ) )
  );
  next_local_step_ad_P10 := ( local_step_ad_P10
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P11_getCurrentIndex := (   a_P11.state = s_Thinking ? 0 :  (   a_P11.state =
s_Right ? 1 :  (   a_P11.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P11 automata local steps section ==================
  local_step_ad_P11 := (
    -- connection s_Thinking - s_Right
    ( ( a_P11.state = s_Thinking & e_t_r_11 ) & ( next(a_P11.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P11.state = s_Right & e_r_l_11 ) & ( next(a_P11.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P11.state = s_Left & e_l_t_11 ) & ( next(a_P11.state) = s_Thinking ) )
  );
  next_local_step_ad_P11 := ( local_step_ad_P11
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
```

```
  a_P12_getCurrentIndex := (   a_P12.state = s_Thinking ? 0 :  (   a_P12.state =
s_Right ? 1 :  (   a_P12.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P12 automata local steps section ==================
  local_step_ad_P12 := (
    -- connection s_Thinking - s_Right
    ( ( a_P12.state = s_Thinking & e_t_r_12 ) & ( next(a_P12.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P12.state = s_Right & e_r_l_12 ) & ( next(a_P12.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P12.state = s_Left & e_l_t_12 ) & ( next(a_P12.state) = s_Thinking ) )
  );
  next_local_step_ad_P12 := ( local_step_ad_P12
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
  );
  a_P13_getCurrentIndex := (   a_P13.state = s_Thinking ? 0 :  (   a_P13.state =
s_Right ? 1 :  (   a_P13.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P13 automata local steps section ==================
  local_step_ad_P13 := (
    -- connection s_Thinking - s_Right
    ( ( a_P13.state = s_Thinking & e_t_r_13 ) & ( next(a_P13.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P13.state = s_Right & e_r_l_13 ) & ( next(a_P13.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P13.state = s_Left & e_l_t_13 ) & ( next(a_P13.state) = s_Thinking ) )
  );
  next_local_step_ad_P13 := ( local_step_ad_P13
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P14.unchanged
  );
```

```
  a_P14_getCurrentIndex := (   a_P14.state = s_Thinking ? 0 :  (   a_P14.state =
s_Left ? 1 :  (   a_P14.state = s_Right ? 2 :  -1  )  )   );
  -- ================= ad_P14 automata local steps section =================
  local_step_ad_P14 := (
    -- connection s_Thinking - s_Left
    ( ( a_P14.state = s_Thinking & e_t_l_14 ) & ( next(a_P14.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P14.state = s_Left & e_l_r_14 ) & ( next(a_P14.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P14.state = s_Right & e_r_t_14 ) & ( next(a_P14.state) = s_Thinking ) )
  );
  next_local_step_ad_P14 := ( local_step_ad_P14
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 15 );

  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_0 := ( ( ( ( a_P14.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_2 := ( ( ( ( a_P3.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_3 := ( ( ( ( a_P4.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_3 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_4 := ( ( ( ( a_P5.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);
```

```
  i_Right_to_Left_4 := ( ( ( ( a_P3.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_5 := ( ( ( ( a_P6.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_5 := ( ( ( ( a_P4.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_6 := ( ( ( ( a_P7.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_6 := ( ( ( ( a_P5.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_7 := ( ( ( ( a_P8.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_7 := ( ( ( ( a_P6.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_8 := ( ( ( ( a_P9.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_8 := ( ( ( ( a_P7.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_9 := ( ( ( ( a_P10.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_9 := ( ( ( ( a_P8.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_10 := ( ( ( ( a_P11.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_10 := ( ( ( ( a_P9.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_11 := ( ( ( ( a_P12.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_11 := ( ( ( ( a_P10.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_12 := ( ( ( ( a_P13.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_12 := ( ( ( ( a_P11.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_13 := ( ( ( ( a_P14.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Right_to_Left_13 := ( ( ( ( a_P12.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Left_14 := ( ( ( ( a_P13.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Left_to_Right_14 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );
```

```
e_r_l_0 := ( i_Right_to_Left_0 > 0 );

e_l_t_0 := ( i_mu > 0 );

e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

e_r_l_1 := ( i_Right_to_Left_1 > 0 );

e_l_t_1 := ( i_mu > 0 );

e_t_r_2 := ( i_Thinking_to_Right_2 > 0 );

e_r_l_2 := ( i_Right_to_Left_2 > 0 );

e_l_t_2 := ( i_mu > 0 );

e_t_r_3 := ( i_Thinking_to_Right_3 > 0 );

e_r_l_3 := ( i_Right_to_Left_3 > 0 );

e_l_t_3 := ( i_mu > 0 );

e_t_r_4 := ( i_Thinking_to_Right_4 > 0 );

e_r_l_4 := ( i_Right_to_Left_4 > 0 );

e_l_t_4 := ( i_mu > 0 );

e_t_r_5 := ( i_Thinking_to_Right_5 > 0 );

e_r_l_5 := ( i_Right_to_Left_5 > 0 );

e_l_t_5 := ( i_mu > 0 );

e_t_r_6 := ( i_Thinking_to_Right_6 > 0 );

e_r_l_6 := ( i_Right_to_Left_6 > 0 );

e_l_t_6 := ( i_mu > 0 );

e_t_r_7 := ( i_Thinking_to_Right_7 > 0 );

e_r_l_7 := ( i_Right_to_Left_7 > 0 );

e_l_t_7 := ( i_mu > 0 );

e_t_r_8 := ( i_Thinking_to_Right_8 > 0 );

e_r_l_8 := ( i_Right_to_Left_8 > 0 );

e_l_t_8 := ( i_mu > 0 );

e_t_r_9 := ( i_Thinking_to_Right_9 > 0 );

e_r_l_9 := ( i_Right_to_Left_9 > 0 );

e_l_t_9 := ( i_mu > 0 );
```

```
  e_t_r_10 := ( i_Thinking_to_Right_10 > 0 );
  e_r_l_10 := ( i_Right_to_Left_10 > 0 );
  e_l_t_10 := ( i_mu > 0 );
  e_t_r_11 := ( i_Thinking_to_Right_11 > 0 );
  e_r_l_11 := ( i_Right_to_Left_11 > 0 );
  e_l_t_11 := ( i_mu > 0 );
  e_t_r_12 := ( i_Thinking_to_Right_12 > 0 );
  e_r_l_12 := ( i_Right_to_Left_12 > 0 );
  e_l_t_12 := ( i_mu > 0 );
  e_t_r_13 := ( i_Thinking_to_Right_13 > 0 );
  e_r_l_13 := ( i_Right_to_Left_13 > 0 );
  e_l_t_13 := ( i_mu > 0 );
  e_t_l_14 := ( i_Thinking_to_Left_14 > 0 );
  e_l_r_14 := ( i_Left_to_Right_14 > 0 );
  e_r_t_14 := ( i_mu > 0 );
```

APÊNDICE V – ARQUIVO TEXTUAL DO MODELO PHIL20.SAN

```
identifiers

  // Number of philosophers
  P = 20;
  // Acquisition rate
  lambda = 1;
  // Release rate
  mu = 2;

  Thinking_to_Right_0 = (st P1 != Left)     * lambda;
  Right_to_Left_0     = (st P19 != Right)    * lambda;

  Thinking_to_Right_1 = (st P2 != Left)     * lambda;
  Right_to_Left_1     = (st P0 == Thinking) * lambda;

  Thinking_to_Right_2 = (st P3 != Left)     * lambda;
  Right_to_Left_2     = (st P1 == Thinking) * lambda;

  Thinking_to_Right_3 = (st P4 != Left)     * lambda;
  Right_to_Left_3     = (st P2 == Thinking) * lambda;

  Thinking_to_Right_4 = (st P5 != Left)     * lambda;
  Right_to_Left_4     = (st P3 == Thinking) * lambda;

  Thinking_to_Right_5 = (st P6 != Left)     * lambda;
  Right_to_Left_5     = (st P4 == Thinking) * lambda;

  Thinking_to_Right_6 = (st P7 != Left)     * lambda;
  Right_to_Left_6     = (st P5 == Thinking) * lambda;

  Thinking_to_Right_7 = (st P8 != Left)     * lambda;
  Right_to_Left_7     = (st P6 == Thinking) * lambda;

  Thinking_to_Right_8 = (st P9 != Left)     * lambda;
  Right_to_Left_8     = (st P7 == Thinking) * lambda;

  Thinking_to_Right_9 = (st P10 != Left)     * lambda;
  Right_to_Left_9     = (st P8 == Thinking) * lambda;

  Thinking_to_Right_10 = (st P11 != Left)     * lambda;
  Right_to_Left_10     = (st P9 == Thinking) * lambda;

  Thinking_to_Right_11 = (st P12 != Left)     * lambda;
  Right_to_Left_11     = (st P10 == Thinking) * lambda;

  Thinking_to_Right_12 = (st P13 != Left)     * lambda;
  Right_to_Left_12     = (st P11 == Thinking) * lambda;

  Thinking_to_Right_13 = (st P14 != Left)     * lambda;
  Right_to_Left_13     = (st P12 == Thinking) * lambda;

  Thinking_to_Right_14 = (st P15 != Left)     * lambda;
  Right_to_Left_14     = (st P13 == Thinking) * lambda;

  Thinking_to_Right_15 = (st P16 != Left)     * lambda;
```

```
  Right_to_Left_15     = (st P14 == Thinking) * lambda;

  Thinking_to_Right_16 = (st P17 != Left)     * lambda;
  Right_to_Left_16     = (st P15 == Thinking) * lambda;

  Thinking_to_Right_17 = (st P18 != Left)     * lambda;
  Right_to_Left_17     = (st P16 == Thinking) * lambda;

  Thinking_to_Right_18 = (st P19 == Thinking) * lambda;
  Right_to_Left_18     = (st P17 == Thinking) * lambda;

  Thinking_to_Left_19  = (st P18 == Thinking) * lambda;
  Left_to_Right_19     = (st P0 != Left)     * lambda;

events

  loc t_r_0 (Thinking_to_Right_0);
  loc r_l_0 (Right_to_Left_0);
  loc l_t_0 (mu);

  loc t_r_1 (Thinking_to_Right_1);
  loc r_l_1 (Right_to_Left_1);
  loc l_t_1 (mu);

  loc t_r_2 (Thinking_to_Right_2);
  loc r_l_2 (Right_to_Left_2);
  loc l_t_2 (mu);

  loc t_r_3 (Thinking_to_Right_3);
  loc r_l_3 (Right_to_Left_3);
  loc l_t_3 (mu);

  loc t_r_4 (Thinking_to_Right_4);
  loc r_l_4 (Right_to_Left_4);
  loc l_t_4 (mu);

  loc t_r_5 (Thinking_to_Right_5);
  loc r_l_5 (Right_to_Left_5);
  loc l_t_5 (mu);

  loc t_r_6 (Thinking_to_Right_6);
  loc r_l_6 (Right_to_Left_6);
  loc l_t_6 (mu);

  loc t_r_7 (Thinking_to_Right_7);
  loc r_l_7 (Right_to_Left_7);
  loc l_t_7 (mu);

  loc t_r_8 (Thinking_to_Right_8);
  loc r_l_8 (Right_to_Left_8);
  loc l_t_8 (mu);

  loc t_r_9 (Thinking_to_Right_9);
  loc r_l_9 (Right_to_Left_9);
  loc l_t_9 (mu);

  loc t_r_10 (Thinking_to_Right_10);
  loc r_l_10 (Right_to_Left_10);
  loc l_t_10 (mu);
```

```
  loc t_r_11 (Thinking_to_Right_11);
  loc r_l_11 (Right_to_Left_11);
  loc l_t_11 (mu);

  loc t_r_12 (Thinking_to_Right_12);
  loc r_l_12 (Right_to_Left_12);
  loc l_t_12 (mu);

  loc t_r_13 (Thinking_to_Right_13);
  loc r_l_13 (Right_to_Left_13);
  loc l_t_13 (mu);

  loc t_r_14 (Thinking_to_Right_14);
  loc r_l_14 (Right_to_Left_14);
  loc l_t_14 (mu);

  loc t_r_15 (Thinking_to_Right_15);
  loc r_l_15 (Right_to_Left_15);
  loc l_t_15 (mu);

  loc t_r_16 (Thinking_to_Right_16);
  loc r_l_16 (Right_to_Left_16);
  loc l_t_16 (mu);

  loc t_r_17 (Thinking_to_Right_17);
  loc r_l_17 (Right_to_Left_17);
  loc l_t_17 (mu);

  loc t_r_18 (Thinking_to_Right_18);
  loc r_l_18 (Right_to_Left_18);
  loc l_t_18 (mu);

  loc t_l_19 (Thinking_to_Left_19);
  loc l_r_19 (Left_to_Right_19);
  loc r_t_19 (mu);

partial reachability = ((st P0 == Thinking) && (st P1 == Thinking) && (st P2 ==
Thinking) && (st P3 == Thinking) && (st P4 == Thinking) && (st P5 == Thinking) &&
(st P6 == Thinking) && (st P7 == Thinking) && (st P8 == Thinking) && (st P9 ==
Thinking) && (st P10 == Thinking) && (st P11 == Thinking) && (st P12 == Thinking)
&& (st P13 == Thinking) && (st P14 == Thinking) && (st P15 == Thinking) && (st P16
== Thinking) && (st P17 == Thinking) && (st P18 == Thinking) && (st P19 ==
Thinking));

network PHIL20f (continuous)

  aut P0
    stt Thinking to (Right)    t_r_0
    stt Right    to (Left)     r_l_0
    stt Left     to (Thinking) l_t_0

  aut P1
    stt Thinking to (Right)    t_r_1
    stt Right    to (Left)     r_l_1
    stt Left     to (Thinking) l_t_1

  aut P2
    stt Thinking to (Right)    t_r_2
```

```
    stt Right    to (Left)     r_l_2
    stt Left     to (Thinking) l_t_2

  aut P3
    stt Thinking to (Right)    t_r_3
    stt Right    to (Left)     r_l_3
    stt Left     to (Thinking) l_t_3

  aut P4
    stt Thinking to (Right)    t_r_4
    stt Right    to (Left)     r_l_4
    stt Left     to (Thinking) l_t_4

  aut P5
    stt Thinking to (Right)    t_r_5
    stt Right    to (Left)     r_l_5
    stt Left     to (Thinking) l_t_5

  aut P6
    stt Thinking to (Right)    t_r_6
    stt Right    to (Left)     r_l_6
    stt Left     to (Thinking) l_t_6

  aut P7
    stt Thinking to (Right)    t_r_7
    stt Right    to (Left)     r_l_7
    stt Left     to (Thinking) l_t_7

  aut P8
    stt Thinking to (Right)    t_r_8
    stt Right    to (Left)     r_l_8
    stt Left     to (Thinking) l_t_8

  aut P9
    stt Thinking to (Right)    t_r_9
    stt Right    to (Left)     r_l_9
    stt Left     to (Thinking) l_t_9

  aut P10
    stt Thinking to (Right)    t_r_10
    stt Right    to (Left)     r_l_10
    stt Left     to (Thinking) l_t_10

  aut P11
    stt Thinking to (Right)    t_r_11
    stt Right    to (Left)     r_l_11
    stt Left     to (Thinking) l_t_11

  aut P12
    stt Thinking to (Right)    t_r_12
    stt Right    to (Left)     r_l_12
    stt Left     to (Thinking) l_t_12

  aut P13
    stt Thinking to (Right)    t_r_13
    stt Right    to (Left)     r_l_13
    stt Left     to (Thinking) l_t_13

  aut P14
```

```
  stt Thinking to (Right)    t_r_14
  stt Right    to (Left)     r_l_14
  stt Left     to (Thinking) l_t_14

aut P15
  stt Thinking to (Right)    t_r_15
  stt Right    to (Left)     r_l_15
  stt Left     to (Thinking) l_t_15

aut P16
  stt Thinking to (Right)    t_r_16
  stt Right    to (Left)     r_l_16
  stt Left     to (Thinking) l_t_16

aut P17
  stt Thinking to (Right)    t_r_17
  stt Right    to (Left)     r_l_17
  stt Left     to (Thinking) l_t_17

aut P18
  stt Thinking to (Right)    t_r_18
  stt Right    to (Left)     r_l_18
  stt Left     to (Thinking) l_t_18

aut P19
  stt Thinking to (Left)     t_l_19
  stt Left     to (Right)    l_r_19
  stt Right    to (Thinking) r_t_19
```

APÊNDICE W – ARQUIVO TEXTUAL DO MODELO PHIL20.SMV

```
MODULE ad_P0()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P1()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P2()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P3()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P4()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P5()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P6()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P7()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P8()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P9()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P10()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
```

```
MODULE ad_P11()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P12()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P13()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P14()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P15()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P16()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P17()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P18()
VAR state : {s_Thinking,s_Right,s_Left};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_P19()
VAR state : {s_Thinking,s_Left,s_Right};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_P0 : ad_P0();
  a_P1 : ad_P1();
  a_P2 : ad_P2();
  a_P3 : ad_P3();
  a_P4 : ad_P4();
  a_P5 : ad_P5();
  a_P6 : ad_P6();
  a_P7 : ad_P7();
  a_P8 : ad_P8();
  a_P9 : ad_P9();
  a_P10 : ad_P10();
  a_P11 : ad_P11();
```

```
  a_P12 : ad_P12();
  a_P13 : ad_P13();
  a_P14 : ad_P14();
  a_P15 : ad_P15();
  a_P16 : ad_P16();
  a_P17 : ad_P17();
  a_P18 : ad_P18();
  a_P19 : ad_P19();

INIT
  ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( a_P0.state = s_Thinking ) & ( a_P1.state =
s_Thinking ) ) & ( a_P2.state = s_Thinking ) ) & ( a_P3.state = s_Thinking ) ) & (
a_P4.state = s_Thinking ) ) & ( a_P5.state = s_Thinking ) ) & ( a_P6.state =
s_Thinking ) ) & ( a_P7.state = s_Thinking ) ) & ( a_P8.state = s_Thinking ) ) & (
a_P9.state = s_Thinking ) ) & ( a_P10.state = s_Thinking ) ) & ( a_P11.state =
s_Thinking ) ) & ( a_P12.state = s_Thinking ) ) & ( a_P13.state = s_Thinking ) ) &
( a_P14.state = s_Thinking ) ) & ( a_P15.state = s_Thinking ) ) & ( a_P16.state =
s_Thinking ) ) & ( a_P17.state = s_Thinking ) ) & ( a_P18.state = s_Thinking ) ) &
( a_P19.state = s_Thinking )
TRANS
  next_local_step_ad_P0
  |  next_local_step_ad_P1
  |  next_local_step_ad_P2
  |  next_local_step_ad_P3
  |  next_local_step_ad_P4
  |  next_local_step_ad_P5
  |  next_local_step_ad_P6
  |  next_local_step_ad_P7
  |  next_local_step_ad_P8
  |  next_local_step_ad_P9
  |  next_local_step_ad_P10
  |  next_local_step_ad_P11
  |  next_local_step_ad_P12
  |  next_local_step_ad_P13
  |  next_local_step_ad_P14
  |  next_local_step_ad_P15
  |  next_local_step_ad_P16
  |  next_local_step_ad_P17
  |  next_local_step_ad_P18
  |  next_local_step_ad_P19
DEFINE
  a_P0_getCurrentIndex := (   a_P0.state = s_Thinking ? 0 :  (   a_P0.state =
s_Right ? 1 :  (   a_P0.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P0 automata local steps section ==================
  local_step_ad_P0 := (
    -- connection s_Thinking - s_Right
    ( ( a_P0.state = s_Thinking & e_t_r_0 ) & ( next(a_P0.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P0.state = s_Right & e_r_l_0 ) & ( next(a_P0.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P0.state = s_Left & e_l_t_0 ) & ( next(a_P0.state) = s_Thinking ) )
  );
  next_local_step_ad_P0 := ( local_step_ad_P0
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
```

```
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P1_getCurrentIndex := (   a_P1.state = s_Thinking ? 0 :  (   a_P1.state =
s_Right ? 1 :  (   a_P1.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P1 automata local steps section ==================
  local_step_ad_P1 := (
    -- connection s_Thinking - s_Right
    ( ( a_P1.state = s_Thinking & e_t_r_1 ) & ( next(a_P1.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P1.state = s_Right & e_r_l_1 ) & ( next(a_P1.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P1.state = s_Left & e_l_t_1 ) & ( next(a_P1.state) = s_Thinking ) )
  );
  next_local_step_ad_P1 := ( local_step_ad_P1
    & a_P0.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P2_getCurrentIndex := (   a_P2.state = s_Thinking ? 0 :  (   a_P2.state =
s_Right ? 1 :  (   a_P2.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P2 automata local steps section ==================
  local_step_ad_P2 := (
    -- connection s_Thinking - s_Right
    ( ( a_P2.state = s_Thinking & e_t_r_2 ) & ( next(a_P2.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P2.state = s_Right & e_r_l_2 ) & ( next(a_P2.state) = s_Left ) )
```

```
    |
    -- connection s_Left - s_Thinking
    ( ( a_P2.state = s_Left & e_l_t_2 ) & ( next(a_P2.state) = s_Thinking ) )
  );
  next_local_step_ad_P2 := ( local_step_ad_P2
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P3_getCurrentIndex := (   a_P3.state = s_Thinking ? 0 :  (   a_P3.state =
s_Right ? 1 :  (   a_P3.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P3 automata local steps section ==================
  local_step_ad_P3 := (
    -- connection s_Thinking - s_Right
    ( ( a_P3.state = s_Thinking & e_t_r_3 ) & ( next(a_P3.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P3.state = s_Right & e_r_l_3 ) & ( next(a_P3.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P3.state = s_Left & e_l_t_3 ) & ( next(a_P3.state) = s_Thinking ) )
  );
  next_local_step_ad_P3 := ( local_step_ad_P3
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
```

```
  a_P4_getCurrentIndex := (   a_P4.state = s_Thinking ? 0 :  (   a_P4.state =
s_Right ? 1 :  (   a_P4.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P4 automata local steps section ==================
  local_step_ad_P4 := (
    -- connection s_Thinking - s_Right
    ( ( a_P4.state = s_Thinking & e_t_r_4 ) & ( next(a_P4.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P4.state = s_Right & e_r_l_4 ) & ( next(a_P4.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P4.state = s_Left & e_l_t_4 ) & ( next(a_P4.state) = s_Thinking ) )
  );
  next_local_step_ad_P4 := ( local_step_ad_P4
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P5_getCurrentIndex := (   a_P5.state = s_Thinking ? 0 :  (   a_P5.state =
s_Right ? 1 :  (   a_P5.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P5 automata local steps section ==================
  local_step_ad_P5 := (
    -- connection s_Thinking - s_Right
    ( ( a_P5.state = s_Thinking & e_t_r_5 ) & ( next(a_P5.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P5.state = s_Right & e_r_l_5 ) & ( next(a_P5.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P5.state = s_Left & e_l_t_5 ) & ( next(a_P5.state) = s_Thinking ) )
  );
  next_local_step_ad_P5 := ( local_step_ad_P5
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
```

```
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P6_getCurrentIndex := (   a_P6.state = s_Thinking ? 0 :  (   a_P6.state =
s_Right ? 1 :  (   a_P6.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P6 automata local steps section ==================
  local_step_ad_P6 := (
    -- connection s_Thinking - s_Right
    ( ( a_P6.state = s_Thinking & e_t_r_6 ) & ( next(a_P6.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P6.state = s_Right & e_r_l_6 ) & ( next(a_P6.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P6.state = s_Left & e_l_t_6 ) & ( next(a_P6.state) = s_Thinking ) )
  );
  next_local_step_ad_P6 := ( local_step_ad_P6
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P7_getCurrentIndex := (   a_P7.state = s_Thinking ? 0 :  (   a_P7.state =
s_Right ? 1 :  (   a_P7.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P7 automata local steps section ==================
  local_step_ad_P7 := (
    -- connection s_Thinking - s_Right
    ( ( a_P7.state = s_Thinking & e_t_r_7 ) & ( next(a_P7.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P7.state = s_Right & e_r_l_7 ) & ( next(a_P7.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P7.state = s_Left & e_l_t_7 ) & ( next(a_P7.state) = s_Thinking ) )
  );
  next_local_step_ad_P7 := ( local_step_ad_P7
    & a_P0.unchanged
    & a_P1.unchanged
```

```
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P8_getCurrentIndex := (   a_P8.state = s_Thinking ? 0 :  (   a_P8.state =
s_Right ? 1 :  (   a_P8.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P8 automata local steps section =================
  local_step_ad_P8 := (
    -- connection s_Thinking - s_Right
    ( ( a_P8.state = s_Thinking & e_t_r_8 ) & ( next(a_P8.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P8.state = s_Right & e_r_l_8 ) & ( next(a_P8.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P8.state = s_Left & e_l_t_8 ) & ( next(a_P8.state) = s_Thinking ) )
  );
  next_local_step_ad_P8 := ( local_step_ad_P8
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P9_getCurrentIndex := (   a_P9.state = s_Thinking ? 0 :  (   a_P9.state =
s_Right ? 1 :  (   a_P9.state = s_Left ? 2 :  -1  )  )   );
  -- ================= ad_P9 automata local steps section =================
  local_step_ad_P9 := (
    -- connection s_Thinking - s_Right
    ( ( a_P9.state = s_Thinking & e_t_r_9 ) & ( next(a_P9.state) = s_Right ) )
    |
```

```
    -- connection s_Right - s_Left
    ( ( a_P9.state = s_Right & e_r_l_9 ) & ( next(a_P9.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P9.state = s_Left & e_l_t_9 ) & ( next(a_P9.state) = s_Thinking ) )
  );
  next_local_step_ad_P9 := ( local_step_ad_P9
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P10_getCurrentIndex := (   a_P10.state = s_Thinking ? 0 :  (   a_P10.state =
s_Right ? 1 :  (   a_P10.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P10 automata local steps section ==================
  local_step_ad_P10 := (
    -- connection s_Thinking - s_Right
    ( ( a_P10.state = s_Thinking & e_t_r_10 ) & ( next(a_P10.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P10.state = s_Right & e_r_l_10 ) & ( next(a_P10.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P10.state = s_Left & e_l_t_10 ) & ( next(a_P10.state) = s_Thinking ) )
  );
  next_local_step_ad_P10 := ( local_step_ad_P10
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
```

```
    & a_P19.unchanged
  );
  a_P11_getCurrentIndex := (   a_P11.state = s_Thinking ? 0 :  (   a_P11.state =
s_Right ? 1 :  (   a_P11.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P11 automata local steps section ==================
  local_step_ad_P11 := (
    -- connection s_Thinking - s_Right
    ( ( a_P11.state = s_Thinking & e_t_r_11 ) & ( next(a_P11.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P11.state = s_Right & e_r_l_11 ) & ( next(a_P11.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P11.state = s_Left & e_l_t_11 ) & ( next(a_P11.state) = s_Thinking ) )
  );
  next_local_step_ad_P11 := ( local_step_ad_P11
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P12_getCurrentIndex := (   a_P12.state = s_Thinking ? 0 :  (   a_P12.state =
s_Right ? 1 :  (   a_P12.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P12 automata local steps section ==================
  local_step_ad_P12 := (
    -- connection s_Thinking - s_Right
    ( ( a_P12.state = s_Thinking & e_t_r_12 ) & ( next(a_P12.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P12.state = s_Right & e_r_l_12 ) & ( next(a_P12.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P12.state = s_Left & e_l_t_12 ) & ( next(a_P12.state) = s_Thinking ) )
  );
  next_local_step_ad_P12 := ( local_step_ad_P12
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
```

```
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P13_getCurrentIndex := (   a_P13.state = s_Thinking ? 0 :  (   a_P13.state =
s_Right ? 1 :  (   a_P13.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P13 automata local steps section ==================
  local_step_ad_P13 := (
    -- connection s_Thinking - s_Right
    ( ( a_P13.state = s_Thinking & e_t_r_13 ) & ( next(a_P13.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P13.state = s_Right & e_r_l_13 ) & ( next(a_P13.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P13.state = s_Left & e_l_t_13 ) & ( next(a_P13.state) = s_Thinking ) )
  );
  next_local_step_ad_P13 := ( local_step_ad_P13
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P14_getCurrentIndex := (   a_P14.state = s_Thinking ? 0 :  (   a_P14.state =
s_Right ? 1 :  (   a_P14.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P14 automata local steps section ==================
  local_step_ad_P14 := (
    -- connection s_Thinking - s_Right
    ( ( a_P14.state = s_Thinking & e_t_r_14 ) & ( next(a_P14.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P14.state = s_Right & e_r_l_14 ) & ( next(a_P14.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P14.state = s_Left & e_l_t_14 ) & ( next(a_P14.state) = s_Thinking ) )
  );
  next_local_step_ad_P14 := ( local_step_ad_P14
```

```
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P15_getCurrentIndex := (   a_P15.state = s_Thinking ? 0 :  (   a_P15.state =
s_Right ? 1 :  (   a_P15.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P15 automata local steps section ==================
  local_step_ad_P15 := (
    -- connection s_Thinking - s_Right
    ( ( a_P15.state = s_Thinking & e_t_r_15 ) & ( next(a_P15.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P15.state = s_Right & e_r_l_15 ) & ( next(a_P15.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P15.state = s_Left & e_l_t_15 ) & ( next(a_P15.state) = s_Thinking ) )
  );
  next_local_step_ad_P15 := ( local_step_ad_P15
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P16_getCurrentIndex := (   a_P16.state = s_Thinking ? 0 :  (   a_P16.state =
s_Right ? 1 :  (   a_P16.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P16 automata local steps section ==================
  local_step_ad_P16 := (
    -- connection s_Thinking - s_Right
```

```
    ( ( a_P16.state = s_Thinking & e_t_r_16 ) & ( next(a_P16.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P16.state = s_Right & e_r_l_16 ) & ( next(a_P16.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P16.state = s_Left & e_l_t_16 ) & ( next(a_P16.state) = s_Thinking ) )
  );
  next_local_step_ad_P16 := ( local_step_ad_P16
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P17_getCurrentIndex := (   a_P17.state = s_Thinking ? 0 :  (   a_P17.state =
s_Right ? 1 :  (   a_P17.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P17 automata local steps section ==================
  local_step_ad_P17 := (
    -- connection s_Thinking - s_Right
    ( ( a_P17.state = s_Thinking & e_t_r_17 ) & ( next(a_P17.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P17.state = s_Right & e_r_l_17 ) & ( next(a_P17.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P17.state = s_Left & e_l_t_17 ) & ( next(a_P17.state) = s_Thinking ) )
  );
  next_local_step_ad_P17 := ( local_step_ad_P17
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
```

```
    & a_P16.unchanged
    & a_P18.unchanged
    & a_P19.unchanged
  );
  a_P18_getCurrentIndex := (   a_P18.state = s_Thinking ? 0 :  (   a_P18.state =
s_Right ? 1 :  (   a_P18.state = s_Left ? 2 :  -1  )  )   );
  -- ================== ad_P18 automata local steps section ==================
  local_step_ad_P18 := (
    -- connection s_Thinking - s_Right
    ( ( a_P18.state = s_Thinking & e_t_r_18 ) & ( next(a_P18.state) = s_Right ) )
    |
    -- connection s_Right - s_Left
    ( ( a_P18.state = s_Right & e_r_l_18 ) & ( next(a_P18.state) = s_Left ) )
    |
    -- connection s_Left - s_Thinking
    ( ( a_P18.state = s_Left & e_l_t_18 ) & ( next(a_P18.state) = s_Thinking ) )
  );
  next_local_step_ad_P18 := ( local_step_ad_P18
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P19.unchanged
  );
  a_P19_getCurrentIndex := (   a_P19.state = s_Thinking ? 0 :  (   a_P19.state =
s_Left ? 1 :  (   a_P19.state = s_Right ? 2 :  -1  )  )   );
  -- ================== ad_P19 automata local steps section ==================
  local_step_ad_P19 := (
    -- connection s_Thinking - s_Left
    ( ( a_P19.state = s_Thinking & e_t_l_19 ) & ( next(a_P19.state) = s_Left ) )
    |
    -- connection s_Left - s_Right
    ( ( a_P19.state = s_Left & e_l_r_19 ) & ( next(a_P19.state) = s_Right ) )
    |
    -- connection s_Right - s_Thinking
    ( ( a_P19.state = s_Right & e_r_t_19 ) & ( next(a_P19.state) = s_Thinking ) )
  );
  next_local_step_ad_P19 := ( local_step_ad_P19
    & a_P0.unchanged
    & a_P1.unchanged
    & a_P2.unchanged
    & a_P3.unchanged
    & a_P4.unchanged
    & a_P5.unchanged
    & a_P6.unchanged
```

```
    & a_P7.unchanged
    & a_P8.unchanged
    & a_P9.unchanged
    & a_P10.unchanged
    & a_P11.unchanged
    & a_P12.unchanged
    & a_P13.unchanged
    & a_P14.unchanged
    & a_P15.unchanged
    & a_P16.unchanged
    & a_P17.unchanged
    & a_P18.unchanged
  );
  -- ================== identifiers section ==================
  i_P := ( 20 );

  i_lambda := ( 1 );

  i_mu := ( 2 );

  i_Thinking_to_Right_0 := ( ( ( ( a_P1.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_0 := ( ( ( ( a_P19.state != s_Right ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_1 := ( ( ( ( a_P2.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_1 := ( ( ( ( a_P0.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_2 := ( ( ( ( a_P3.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_2 := ( ( ( ( a_P1.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_3 := ( ( ( ( a_P4.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_3 := ( ( ( ( a_P2.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_4 := ( ( ( ( a_P5.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_4 := ( ( ( ( a_P3.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_5 := ( ( ( ( a_P6.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_5 := ( ( ( ( a_P4.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_6 := ( ( ( ( a_P7.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_6 := ( ( ( ( a_P5.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_7 := ( ( ( ( a_P8.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_7 := ( ( ( ( a_P6.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );
```

```
  i_Thinking_to_Right_8 := ( ( ( ( a_P9.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_8 := ( ( ( ( a_P7.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_9 := ( ( ( ( a_P10.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_9 := ( ( ( ( a_P8.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda );

  i_Thinking_to_Right_10 := ( ( ( ( a_P11.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_10 := ( ( ( ( a_P9.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_11 := ( ( ( ( a_P12.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_11 := ( ( ( ( a_P10.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_12 := ( ( ( ( a_P13.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_12 := ( ( ( ( a_P11.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_13 := ( ( ( ( a_P14.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_13 := ( ( ( ( a_P12.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_14 := ( ( ( ( a_P15.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_14 := ( ( ( ( a_P13.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_15 := ( ( ( ( a_P16.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_15 := ( ( ( ( a_P14.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_16 := ( ( ( ( a_P17.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_16 := ( ( ( ( a_P15.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Right_17 := ( ( ( ( a_P18.state != s_Left ) ) ? 1 : 0 ) * i_lambda
);

  i_Right_to_Left_17 := ( ( ( ( a_P16.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);
```

```
  i_Thinking_to_Right_18 := ( ( ( ( a_P19.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Right_to_Left_18 := ( ( ( ( a_P17.state = s_Thinking ) ) ? 1 : 0 ) * i_lambda
);

  i_Thinking_to_Left_19 := ( ( ( ( a_P18.state = s_Thinking ) ) ? 1 : 0 ) *
i_lambda );

  i_Left_to_Right_19 := ( ( ( ( a_P0.state != s_Left ) ) ? 1 : 0 ) * i_lambda );

  -- ================== nb operators section ==================
  -- ================== events section ==================
  e_t_r_0 := ( i_Thinking_to_Right_0 > 0 );

  e_r_l_0 := ( i_Right_to_Left_0 > 0 );

  e_l_t_0 := ( i_mu > 0 );

  e_t_r_1 := ( i_Thinking_to_Right_1 > 0 );

  e_r_l_1 := ( i_Right_to_Left_1 > 0 );

  e_l_t_1 := ( i_mu > 0 );

  e_t_r_2 := ( i_Thinking_to_Right_2 > 0 );

  e_r_l_2 := ( i_Right_to_Left_2 > 0 );

  e_l_t_2 := ( i_mu > 0 );

  e_t_r_3 := ( i_Thinking_to_Right_3 > 0 );

  e_r_l_3 := ( i_Right_to_Left_3 > 0 );

  e_l_t_3 := ( i_mu > 0 );

  e_t_r_4 := ( i_Thinking_to_Right_4 > 0 );

  e_r_l_4 := ( i_Right_to_Left_4 > 0 );

  e_l_t_4 := ( i_mu > 0 );

  e_t_r_5 := ( i_Thinking_to_Right_5 > 0 );

  e_r_l_5 := ( i_Right_to_Left_5 > 0 );

  e_l_t_5 := ( i_mu > 0 );

  e_t_r_6 := ( i_Thinking_to_Right_6 > 0 );

  e_r_l_6 := ( i_Right_to_Left_6 > 0 );

  e_l_t_6 := ( i_mu > 0 );

  e_t_r_7 := ( i_Thinking_to_Right_7 > 0 );

  e_r_l_7 := ( i_Right_to_Left_7 > 0 );
```

```
e_l_t_7 := ( i_mu > 0 );

e_t_r_8 := ( i_Thinking_to_Right_8 > 0 );

e_r_l_8 := ( i_Right_to_Left_8 > 0 );

e_l_t_8 := ( i_mu > 0 );

e_t_r_9 := ( i_Thinking_to_Right_9 > 0 );

e_r_l_9 := ( i_Right_to_Left_9 > 0 );

e_l_t_9 := ( i_mu > 0 );

e_t_r_10 := ( i_Thinking_to_Right_10 > 0 );

e_r_l_10 := ( i_Right_to_Left_10 > 0 );

e_l_t_10 := ( i_mu > 0 );

e_t_r_11 := ( i_Thinking_to_Right_11 > 0 );

e_r_l_11 := ( i_Right_to_Left_11 > 0 );

e_l_t_11 := ( i_mu > 0 );

e_t_r_12 := ( i_Thinking_to_Right_12 > 0 );

e_r_l_12 := ( i_Right_to_Left_12 > 0 );

e_l_t_12 := ( i_mu > 0 );

e_t_r_13 := ( i_Thinking_to_Right_13 > 0 );

e_r_l_13 := ( i_Right_to_Left_13 > 0 );

e_l_t_13 := ( i_mu > 0 );

e_t_r_14 := ( i_Thinking_to_Right_14 > 0 );

e_r_l_14 := ( i_Right_to_Left_14 > 0 );

e_l_t_14 := ( i_mu > 0 );

e_t_r_15 := ( i_Thinking_to_Right_15 > 0 );

e_r_l_15 := ( i_Right_to_Left_15 > 0 );

e_l_t_15 := ( i_mu > 0 );

e_t_r_16 := ( i_Thinking_to_Right_16 > 0 );

e_r_l_16 := ( i_Right_to_Left_16 > 0 );

e_l_t_16 := ( i_mu > 0 );

e_t_r_17 := ( i_Thinking_to_Right_17 > 0 );

e_r_l_17 := ( i_Right_to_Left_17 > 0 );
```

```
e_l_t_17 := ( i_mu > 0 );

e_t_r_18 := ( i_Thinking_to_Right_18 > 0 );

e_r_l_18 := ( i_Right_to_Left_18 > 0 );

e_l_t_18 := ( i_mu > 0 );

e_t_l_19 := ( i_Thinking_to_Left_19 > 0 );

e_l_r_19 := ( i_Left_to_Right_19 > 0 );

e_r_t_19 := ( i_mu > 0 );
```

APÊNDICE X – ARQUIVO TEXTUAL DO MODELO AD04C.SAN

```
identifiers
  bandwidth_b = 2000000;                    // bandwidth in bits/seconds
  bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
  bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
  size_pack = 1500;                           // package size in bytes
  RTS = 40;                                 // Request To Send header in bytes
  CTS_ACK = 39;                             // Clear To Send and Acknowledge
headers in bytes
  MAC = 47;                                 // Medium Access Control header in
bytes
  IFT = 50+(3*10)+280;                      // InterFrame Time (DIFS + 3*SIFS +
Average Backoff Time) in microseconds (approximated)
  IFS = ((bandwidth_B/1000000)*IFT);        // InterFrame Size cost in bytes
(approximated)
  overhead = RTS+CTS_ACK+MAC+IFS;           // headers
  mu = (bandwidth_B)/(size_pack+overhead); // maximum throughput rate
(theoretically as many package as the medium can handle)
  lambda = 50000;                           // package generation rate (one
package/DIFS)

  r12 = lambda;
  r23 = lambda;
  r34 = lambda;

events
  loc tx1 mu;                               // transmission of one package
  loc tx2 mu;                               // transmission of one package
  syn tx3 mu;                               // transmission of one package

  syn g12 r12;                              // package generated from 1 to 2
  syn g23 r23;                              // package routed from 2 to 3
  syn g34 r34;                              // package routed from 3 to 4

partial reachability = (nb I == 4);         // initial state

network Ad (continuous)
  aut MN_1
    stt I to (T) g12
          to (I) g23 g34
    stt T to (I) tx1

  aut MN_2
    stt I to (R) g12
          to (I) g34
    stt R to (T) g23
    stt T to (I) tx2

  aut MN_3
    stt I to (R) g23
          to (I) g12
    stt R to (T) g34
    stt T to (I) tx3

  aut MN_4
    stt I to (R) g34
```

```
          to (I) g12 g23
    stt R to (I) tx3
```

APÊNDICE Y – ARQUIVO TEXTUAL DO MODELO AD04C.SMV

```
MODULE ad_MN_1()
VAR state : {s_I,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_2()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_3()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_4()
VAR state : {s_I,s_R};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_MN_1 : ad_MN_1();
  a_MN_2 : ad_MN_2();
  a_MN_3 : ad_MN_3();
  a_MN_4 : ad_MN_4();

INIT
  ( nb_s_I = 4 )
TRANS
  next_local_step_ad_MN_1
  |  synchronized_steps
  |  next_local_step_ad_MN_2
DEFINE
  a_MN_1_getCurrentIndex := (   a_MN_1.state = s_I ? 0 :  (   a_MN_1.state = s_T ?
1 :  -1  )   );
  -- ================== ad_MN_1 automata local steps section ==================
  local_step_ad_MN_1 := (
    -- connection s_T - s_I
    ( ( a_MN_1.state = s_T & e_tx1 ) & ( next(a_MN_1.state) = s_I ) )
  );
  next_local_step_ad_MN_1 := ( local_step_ad_MN_1
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
  );
  a_MN_2_getCurrentIndex := (   a_MN_2.state = s_I ? 0 :  (   a_MN_2.state = s_R ?
1 :  (   a_MN_2.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_2 automata local steps section ==================
  local_step_ad_MN_2 := (
    -- connection s_T - s_I
    ( ( a_MN_2.state = s_T & e_tx2 ) & ( next(a_MN_2.state) = s_I ) )
  );
  next_local_step_ad_MN_2 := ( local_step_ad_MN_2
    & a_MN_1.unchanged
```

```
    & a_MN_3.unchanged
    & a_MN_4.unchanged
  );
  a_MN_3_getCurrentIndex := (   a_MN_3.state = s_I ? 0 :  (   a_MN_3.state = s_R ?
1 :  (   a_MN_3.state = s_T ? 2 :  -1  )  )   );
  a_MN_4_getCurrentIndex := (   a_MN_4.state = s_I ? 0 :  (   a_MN_4.state = s_R ?
1 :  -1  )   );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_tx3
        &
        (
        -- connection s_T - s_I
         ( a_MN_3.state = s_T & next(a_MN_3.state) = s_I )
        )
        &
        (
        -- connection s_R - s_I
         ( a_MN_4.state = s_R & next(a_MN_4.state) = s_I )
        )
        & a_MN_1.unchanged
        & a_MN_2.unchanged
    )
    |
    (
        e_g12
        &
        (
        -- connection s_I - s_T
         ( a_MN_1.state = s_I & next(a_MN_1.state) = s_T )
        )
        &
        (
        -- connection s_I - s_R
         ( a_MN_2.state = s_I & next(a_MN_2.state) = s_R )
        )
        &
        (
        -- connection s_I - s_I
         ( a_MN_3.state = s_I & next(a_MN_3.state) = s_I )
        )
        &
        (
        -- connection s_I - s_I
         ( a_MN_4.state = s_I & next(a_MN_4.state) = s_I )
        )
    )
    |
    (
        e_g23
        &
        (
        -- connection s_I - s_I
         ( a_MN_1.state = s_I & next(a_MN_1.state) = s_I )
        )
        &
        (
        -- connection s_R - s_T
```

```
        ( a_MN_2.state = s_R & next(a_MN_2.state) = s_T )
       )
       &
       (
       -- connection s_I - s_R
        ( a_MN_3.state = s_I & next(a_MN_3.state) = s_R )
       )
       &
       (
       -- connection s_I - s_I
        ( a_MN_4.state = s_I & next(a_MN_4.state) = s_I )
       )
  )
  |
  (
       e_g34
       &
       (
       -- connection s_I - s_I
        ( a_MN_1.state = s_I & next(a_MN_1.state) = s_I )
       )
       &
       (
       -- connection s_I - s_I
        ( a_MN_2.state = s_I & next(a_MN_2.state) = s_I )
       )
       &
       (
       -- connection s_R - s_T
        ( a_MN_3.state = s_R & next(a_MN_3.state) = s_T )
       )
       &
       (
       -- connection s_I - s_R
        ( a_MN_4.state = s_I & next(a_MN_4.state) = s_R )
       )
  )
);
-- ================== identifiers section ==================
i_bandwidth_b := ( 2000000 );

i_bandwidth_B := ( i_bandwidth_b / 8 );

i_bandwidth_Mbps := ( i_bandwidth_b / 1000000 );

i_size_pack := ( 1500 );

i_RTS := ( 40 );

i_CTS_ACK := ( 39 );

i_MAC := ( 47 );

i_IFT := ( ( 50 + ( 3 * 10 ) ) + 280 );

i_IFS := ( ( ( i_bandwidth_B / 1000000 ) * i_IFT ) );

i_overhead := ( ( ( i_RTS + i_CTS_ACK ) + i_MAC ) + i_IFS );
```

```
  i_mu := ( ( i_bandwidth_B ) / ( i_size_pack + i_overhead ) );

  i_lambda := ( 50000 );

  i_r12 := ( i_lambda );

  i_r23 := ( i_lambda );

  i_r34 := ( i_lambda );

  -- ================== nb operators section ==================
  nb_s_I := (
    ( a_MN_1.state = s_I ? 1 : 0 )  +
    ( a_MN_2.state = s_I ? 1 : 0 )  +
    ( a_MN_3.state = s_I ? 1 : 0 )  +
    ( a_MN_4.state = s_I ? 1 : 0 )
  );

  -- ================== events section ==================
  e_tx1 := ( i_mu > 0 );

  e_tx2 := ( i_mu > 0 );

  e_tx3 := ( i_mu > 0 );

  e_g12 := ( i_r12 > 0 );

  e_g23 := ( i_r23 > 0 );

  e_g34 := ( i_r34 > 0 );
```

APÊNDICE Z – ARQUIVO TEXTUAL DO MODELO AD10F.SAN

```
identifiers
  bandwidth_b = 2000000;                    // bandwidth in bits/seconds
  bandwidth_B = bandwidth_b/8;              // bandwidth in bytes/seconds
  bandwidth_Mbps = bandwidth_b/1000000;     // bandwidth in bytes/seconds
  size_pack = 1500;                           // package size in bytes
  RTS = 40;                                 // Request To Send header in bytes
  CTS_ACK = 39;                             // Clear To Send and Acknowledge
headers in bytes
  MAC = 47;                                 // Medium Access Control header in
bytes
  IFT = 50+(3*10)+280;                      // InterFrame Time (DIFS + 3*SIFS +
Average Backoff Time) in microseconds (approximated)
  IFS = ((bandwidth_B/1000000)*IFT);        // InterFrame Size cost in bytes
(approximated)
  overhead = RTS+CTS_ACK+MAC+IFS;           // headers
  mu = (bandwidth_B)/(size_pack+overhead); // maximum throughput rate
(theoretically as many package as the medium can handle)
  lambda = 50000;                           // package generation rate (one
package/DIFS)

  r12  = lambda * (                                  (st MN_3 == I) && (st MN_4
== I));
  r23  = lambda * (                  (st MN_1 == I) && (st MN_4 == I) && (st MN_5
== I));
  r34  = lambda * ((st MN_1 == I) && (st MN_2 == I) && (st MN_5 == I) && (st MN_6
== I));
  r45  = lambda * ((st MN_2 == I) && (st MN_3 == I) && (st MN_6 == I) && (st MN_7
== I));
  r56  = lambda * ((st MN_3 == I) && (st MN_4 == I) && (st MN_7 == I) && (st MN_8
== I));
  r67  = lambda * ((st MN_4 == I) && (st MN_5 == I) && (st MN_8 == I) && (st MN_9
== I));
  r78  = lambda * ((st MN_5 == I) && (st MN_6 == I) && (st MN_9 == I) && (st MN_10
== I));
  r89  = lambda * ((st MN_6 == I) && (st MN_7 == I) && (st MN_10 == I));
  r910 = lambda * ((st MN_7 == I) && (st MN_8 == I));

events
  loc tx1 mu;                               // transmission of one package
  loc tx2 mu;                               // transmission of one package
  loc tx3 mu;                               // transmission of one package
  loc tx4 mu;                               // transmission of one package
  loc tx5 mu;                               // transmission of one package
  loc tx6 mu;                               // transmission of one package
  loc tx7 mu;                               // transmission of one package
  loc tx8 mu;                               // transmission of one package
  syn tx9 mu;                               // transmission of one package

  syn g12  r12;                             // package generated from 1 to 2
  syn g23  r23;                             // package routed from 2 to 3
  syn g34  r34;                             // package routed from 3 to 4
  syn g45  r45;                             // package routed from 4 to 5
  syn g56  r56;                             // package routed from 5 to 6
  syn g67  r67;                             // package routed from 6 to 7
  syn g78  r78;                             // package routed from 7 to 8
```

```
  syn g89  r89;                          // package routed from 8 to 9
  syn g910 r910;                         // package routed from 9 to 10

partial reachability = (nb I == 10);        // initial state

network Ad (continuous)
  aut MN_1
    stt I to (T) g12
    stt T to (I) tx1

  aut MN_2
    stt I to (R) g12
    stt R to (T) g23
    stt T to (I) tx2

  aut MN_3
    stt I to (R) g23
    stt R to (T) g34
    stt T to (I) tx3

  aut MN_4
    stt I to (R) g34
    stt R to (T) g45
    stt T to (I) tx4

  aut MN_5
    stt I to (R) g45
    stt R to (T) g56
    stt T to (I) tx5

  aut MN_6
    stt I to (R) g56
    stt R to (T) g67
    stt T to (I) tx6

  aut MN_7
    stt I to (R) g67
    stt R to (T) g78
    stt T to (I) tx7

  aut MN_8
    stt I to (R) g78
    stt R to (T) g89
    stt T to (I) tx8

  aut MN_9
    stt I to (R) g89
    stt R to (T) g910
    stt T to (I) tx9

  aut MN_10
    stt I to (R) g910
    stt R to (I) tx9
```

APÊNDICE AA – ARQUIVO TEXTUAL DO MODELO AD10F.SMV

```
MODULE ad_MN_1()
VAR state : {s_I,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_2()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_3()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_4()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_5()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_6()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_7()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_8()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_9()
VAR state : {s_I,s_R,s_T};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_MN_10()
VAR state : {s_I,s_R};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_MN_1 : ad_MN_1();
  a_MN_2 : ad_MN_2();
  a_MN_3 : ad_MN_3();
```

```
  a_MN_4 : ad_MN_4();
  a_MN_5 : ad_MN_5();
  a_MN_6 : ad_MN_6();
  a_MN_7 : ad_MN_7();
  a_MN_8 : ad_MN_8();
  a_MN_9 : ad_MN_9();
  a_MN_10 : ad_MN_10();

INIT
  ( nb_s_I = 10 )
TRANS
  next_local_step_ad_MN_1
  |  synchronized_steps
  |  next_local_step_ad_MN_2
  |  next_local_step_ad_MN_3
  |  next_local_step_ad_MN_4
  |  next_local_step_ad_MN_5
  |  next_local_step_ad_MN_6
  |  next_local_step_ad_MN_7
  |  next_local_step_ad_MN_8
DEFINE
  a_MN_1_getCurrentIndex := (   a_MN_1.state = s_I ? 0 :  (   a_MN_1.state = s_T ?
1 :  -1  )   );
  -- ================== ad_MN_1 automata local steps section ==================
  local_step_ad_MN_1 := (
    -- connection s_T - s_I
    ( ( a_MN_1.state = s_T & e_tx1 ) & ( next(a_MN_1.state) = s_I ) )
  );
  next_local_step_ad_MN_1 := ( local_step_ad_MN_1
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_2_getCurrentIndex := (   a_MN_2.state = s_I ? 0 :  (   a_MN_2.state = s_R ?
1 :  (   a_MN_2.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_2 automata local steps section ==================
  local_step_ad_MN_2 := (
    -- connection s_T - s_I
    ( ( a_MN_2.state = s_T & e_tx2 ) & ( next(a_MN_2.state) = s_I ) )
  );
  next_local_step_ad_MN_2 := ( local_step_ad_MN_2
    & a_MN_1.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_3_getCurrentIndex := (   a_MN_3.state = s_I ? 0 :  (   a_MN_3.state = s_R ?
1 :  (   a_MN_3.state = s_T ? 2 :  -1  )  )   );
```

```
  -- ================== ad_MN_3 automata local steps section ==================
  local_step_ad_MN_3 := (
    -- connection s_T - s_I
    ( ( a_MN_3.state = s_T & e_tx3 ) & ( next(a_MN_3.state) = s_I ) )
  );
  next_local_step_ad_MN_3 := ( local_step_ad_MN_3
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_4_getCurrentIndex := (   a_MN_4.state = s_I ? 0 :  (   a_MN_4.state = s_R ?
1 :  (   a_MN_4.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_4 automata local steps section ==================
  local_step_ad_MN_4 := (
    -- connection s_T - s_I
    ( ( a_MN_4.state = s_T & e_tx4 ) & ( next(a_MN_4.state) = s_I ) )
  );
  next_local_step_ad_MN_4 := ( local_step_ad_MN_4
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_5_getCurrentIndex := (   a_MN_5.state = s_I ? 0 :  (   a_MN_5.state = s_R ?
1 :  (   a_MN_5.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_5 automata local steps section ==================
  local_step_ad_MN_5 := (
    -- connection s_T - s_I
    ( ( a_MN_5.state = s_T & e_tx5 ) & ( next(a_MN_5.state) = s_I ) )
  );
  next_local_step_ad_MN_5 := ( local_step_ad_MN_5
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_6_getCurrentIndex := (   a_MN_6.state = s_I ? 0 :  (   a_MN_6.state = s_R ?
1 :  (   a_MN_6.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_6 automata local steps section ==================
  local_step_ad_MN_6 := (
    -- connection s_T - s_I
    ( ( a_MN_6.state = s_T & e_tx6 ) & ( next(a_MN_6.state) = s_I ) )
  );
```

```
  next_local_step_ad_MN_6 := ( local_step_ad_MN_6
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_7_getCurrentIndex := (   a_MN_7.state = s_I ? 0 :  (   a_MN_7.state = s_R ?
1 :  (   a_MN_7.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_7 automata local steps section ==================
  local_step_ad_MN_7 := (
    -- connection s_T - s_I
    ( ( a_MN_7.state = s_T & e_tx7 ) & ( next(a_MN_7.state) = s_I ) )
  );
  next_local_step_ad_MN_7 := ( local_step_ad_MN_7
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_8_getCurrentIndex := (   a_MN_8.state = s_I ? 0 :  (   a_MN_8.state = s_R ?
1 :  (   a_MN_8.state = s_T ? 2 :  -1  )  )   );
  -- ================== ad_MN_8 automata local steps section ==================
  local_step_ad_MN_8 := (
    -- connection s_T - s_I
    ( ( a_MN_8.state = s_T & e_tx8 ) & ( next(a_MN_8.state) = s_I ) )
  );
  next_local_step_ad_MN_8 := ( local_step_ad_MN_8
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
  );
  a_MN_9_getCurrentIndex := (   a_MN_9.state = s_I ? 0 :  (   a_MN_9.state = s_R ?
1 :  (   a_MN_9.state = s_T ? 2 :  -1  )  )   );
  a_MN_10_getCurrentIndex := (   a_MN_10.state = s_I ? 0 :  (   a_MN_10.state =
s_R ? 1 :  -1  )   );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_tx9
        &
        (
        -- connection s_T - s_I
         ( a_MN_9.state = s_T & next(a_MN_9.state) = s_I )
```

```
    )
    &
    (
    -- connection s_R - s_I
     ( a_MN_10.state = s_R & next(a_MN_10.state) = s_I )
    )
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
)
|
(
    e_g12
    -- connection s_I - s_T
    & ( a_MN_1.state = s_I & next(a_MN_1.state) = s_T )
    &
    (
    -- connection s_I - s_R
     ( a_MN_2.state = s_I & next(a_MN_2.state) = s_R )
    )
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
)
|
(
    e_g23
    &
    (
    -- connection s_R - s_T
     ( a_MN_2.state = s_R & next(a_MN_2.state) = s_T )
    )
    &
    (
    -- connection s_I - s_R
     ( a_MN_3.state = s_I & next(a_MN_3.state) = s_R )
    )
    & a_MN_1.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_7.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
)
|
(
    e_g34
```

```
        &
        (
        -- connection s_R - s_T
         ( a_MN_3.state = s_R & next(a_MN_3.state) = s_T )
        )
        &
        (
        -- connection s_I - s_R
         ( a_MN_4.state = s_I & next(a_MN_4.state) = s_R )
        )
        & a_MN_1.unchanged
        & a_MN_2.unchanged
        & a_MN_5.unchanged
        & a_MN_6.unchanged
        & a_MN_7.unchanged
        & a_MN_8.unchanged
        & a_MN_9.unchanged
        & a_MN_10.unchanged
    )
    |
    (
        e_g45
        &
        (
        -- connection s_R - s_T
         ( a_MN_4.state = s_R & next(a_MN_4.state) = s_T )
        )
        &
        (
        -- connection s_I - s_R
         ( a_MN_5.state = s_I & next(a_MN_5.state) = s_R )
        )
        & a_MN_1.unchanged
        & a_MN_2.unchanged
        & a_MN_3.unchanged
        & a_MN_6.unchanged
        & a_MN_7.unchanged
        & a_MN_8.unchanged
        & a_MN_9.unchanged
        & a_MN_10.unchanged
    )
    |
    (
        e_g56
        &
        (
        -- connection s_R - s_T
         ( a_MN_5.state = s_R & next(a_MN_5.state) = s_T )
        )
        &
        (
        -- connection s_I - s_R
         ( a_MN_6.state = s_I & next(a_MN_6.state) = s_R )
        )
        & a_MN_1.unchanged
        & a_MN_2.unchanged
        & a_MN_3.unchanged
        & a_MN_4.unchanged
        & a_MN_7.unchanged
```

```
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
)
|
(
    e_g67
    &
    (
    -- connection s_R - s_T
     ( a_MN_6.state = s_R & next(a_MN_6.state) = s_T )
    )
    &
    (
    -- connection s_I - s_R
     ( a_MN_7.state = s_I & next(a_MN_7.state) = s_R )
    )
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_8.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
)
|
(
    e_g78
    &
    (
    -- connection s_R - s_T
     ( a_MN_7.state = s_R & next(a_MN_7.state) = s_T )
    )
    &
    (
    -- connection s_I - s_R
     ( a_MN_8.state = s_I & next(a_MN_8.state) = s_R )
    )
    & a_MN_1.unchanged
    & a_MN_2.unchanged
    & a_MN_3.unchanged
    & a_MN_4.unchanged
    & a_MN_5.unchanged
    & a_MN_6.unchanged
    & a_MN_9.unchanged
    & a_MN_10.unchanged
)
|
(
    e_g89
    &
    (
    -- connection s_R - s_T
     ( a_MN_8.state = s_R & next(a_MN_8.state) = s_T )
    )
    &
    (
    -- connection s_I - s_R
```

```
        ( a_MN_9.state = s_I & next(a_MN_9.state) = s_R )
       )
       & a_MN_1.unchanged
       & a_MN_2.unchanged
       & a_MN_3.unchanged
       & a_MN_4.unchanged
       & a_MN_5.unchanged
       & a_MN_6.unchanged
       & a_MN_7.unchanged
       & a_MN_10.unchanged
  )
  |
  (
       e_g910
       &
       (
       -- connection s_R - s_T
        ( a_MN_9.state = s_R & next(a_MN_9.state) = s_T )
       )
       &
       (
       -- connection s_I - s_R
        ( a_MN_10.state = s_I & next(a_MN_10.state) = s_R )
       )
       & a_MN_1.unchanged
       & a_MN_2.unchanged
       & a_MN_3.unchanged
       & a_MN_4.unchanged
       & a_MN_5.unchanged
       & a_MN_6.unchanged
       & a_MN_7.unchanged
       & a_MN_8.unchanged
  )
);
-- ================== identifiers section ==================
i_bandwidth_b := ( 2000000 );

i_bandwidth_B := ( i_bandwidth_b / 8 );

i_bandwidth_Mbps := ( i_bandwidth_b / 1000000 );

i_size_pack := ( 1500 );

i_RTS := ( 40 );

i_CTS_ACK := ( 39 );

i_MAC := ( 47 );

i_IFT := ( ( 50 + ( 3 * 10 ) ) + 280 );

i_IFS := ( ( ( i_bandwidth_B / 1000000 ) * i_IFT ) );

i_overhead := ( ( ( i_RTS + i_CTS_ACK ) + i_MAC ) + i_IFS );

i_mu := ( ( i_bandwidth_B ) / ( i_size_pack + i_overhead ) );

i_lambda := ( 50000 );
```

```
  i_r12 := ( i_lambda * ( ( ( ( a_MN_3.state = s_I ) & ( a_MN_4.state = s_I ) ) )
? 1 : 0 ) );

  i_r23 := ( i_lambda * ( ( ( ( ( a_MN_1.state = s_I ) & ( a_MN_4.state = s_I ) )
& ( a_MN_5.state = s_I ) ) ) ? 1 : 0 ) );

  i_r34 := ( i_lambda * ( ( ( ( ( ( a_MN_1.state = s_I ) & ( a_MN_2.state = s_I )
) & ( a_MN_5.state = s_I ) ) & ( a_MN_6.state = s_I ) ) ) ? 1 : 0 ) );

  i_r45 := ( i_lambda * ( ( ( ( ( ( a_MN_2.state = s_I ) & ( a_MN_3.state = s_I )
) & ( a_MN_6.state = s_I ) ) & ( a_MN_7.state = s_I ) ) ) ? 1 : 0 ) );

  i_r56 := ( i_lambda * ( ( ( ( ( ( a_MN_3.state = s_I ) & ( a_MN_4.state = s_I )
) & ( a_MN_7.state = s_I ) ) & ( a_MN_8.state = s_I ) ) ) ? 1 : 0 ) );

  i_r67 := ( i_lambda * ( ( ( ( ( ( a_MN_4.state = s_I ) & ( a_MN_5.state = s_I )
) & ( a_MN_8.state = s_I ) ) & ( a_MN_9.state = s_I ) ) ) ? 1 : 0 ) );

  i_r78 := ( i_lambda * ( ( ( ( ( ( a_MN_5.state = s_I ) & ( a_MN_6.state = s_I )
) & ( a_MN_9.state = s_I ) ) & ( a_MN_10.state = s_I ) ) ) ? 1 : 0 ) );

  i_r89 := ( i_lambda * ( ( ( ( ( a_MN_6.state = s_I ) & ( a_MN_7.state = s_I ) )
& ( a_MN_10.state = s_I ) ) ) ? 1 : 0 ) );

  i_r910 := ( i_lambda * ( ( ( ( a_MN_7.state = s_I ) & ( a_MN_8.state = s_I ) ) )
? 1 : 0 ) );

  -- ================== nb operators section ==================
  nb_s_I := (
    ( a_MN_1.state = s_I ? 1 : 0 )  +
    ( a_MN_2.state = s_I ? 1 : 0 )  +
    ( a_MN_3.state = s_I ? 1 : 0 )  +
    ( a_MN_4.state = s_I ? 1 : 0 )  +
    ( a_MN_5.state = s_I ? 1 : 0 )  +
    ( a_MN_6.state = s_I ? 1 : 0 )  +
    ( a_MN_7.state = s_I ? 1 : 0 )  +
    ( a_MN_8.state = s_I ? 1 : 0 )  +
    ( a_MN_9.state = s_I ? 1 : 0 )  +
    ( a_MN_10.state = s_I ? 1 : 0 )
  );

  -- ================== events section ==================
  e_tx1 := ( i_mu > 0 );

  e_tx2 := ( i_mu > 0 );

  e_tx3 := ( i_mu > 0 );

  e_tx4 := ( i_mu > 0 );

  e_tx5 := ( i_mu > 0 );

  e_tx6 := ( i_mu > 0 );

  e_tx7 := ( i_mu > 0 );

  e_tx8 := ( i_mu > 0 );

  e_tx9 := ( i_mu > 0 );
```

```
  e_g12 := ( i_r12 > 0 );
  e_g23 := ( i_r23 > 0 );
  e_g34 := ( i_r34 > 0 );
  e_g45 := ( i_r45 > 0 );
  e_g56 := ( i_r56 > 0 );
  e_g67 := ( i_r67 > 0 );
  e_g78 := ( i_r78 > 0 );
  e_g89 := ( i_r89 > 0 );
  e_g910 := ( i_r910 > 0 );
```

APÊNDICE AB – ARQUIVO TEXTUAL DO MODELO PL4.SAN

```
identifiers

  // Service rates
  mu1 = 1;
  mu2 = 1;
  mu3 = 1;
  mu4 = 1;

  // Blocking functions (allow that station "j" goes to the "blocking" state with
a rate "mu_i", indicating that station "i" is blocked)
  f2_3_b = (st M2 != st_0_0) * mu2;
  f3_4_b = (st M3 != st_0_0) * mu3;

  // Functional probabilities to allow back transitions in station 2
  backM3 = ((st M3 == st_2_2));

  // Functional probabilities to avoid back transitions in station 2
  nbackM3 = !(backM3);

events

  loc r1_2          (mu1);
  syn r2_3          (mu2);
  loc r2_3_b        (f2_3_b);
  syn r2_3_u        (mu2);
  syn r3_4          (mu3);
  loc r3_4_b        (f3_4_b);
  syn r3_4_u        (mu3);
  loc r4_x          (mu4);
  syn r4_x_u        (mu4);

reachability = !((st M2 == st_0_0) && (st M3 == st_2_2)) && !((st M3 == st_0_0) &&
(st M4 == st_2_2));

network B2_2_B3_2_B4_2 (continuous)

  aut M2
    stt st_0_0 to (st_0_1) r1_2
               to (st_0_0) r4_x_u
    stt st_0_1 to (st_1_1) r1_2
               to (st_0_1) r4_x_u(nbackM3)
               to (st_0_0) r2_3 r3_4_u r4_x_u(backM3)
    stt st_1_1 to (st_2_1) r1_2
               to (st_1_1) r4_x_u(nbackM3)
               to (st_0_1) r2_3 r3_4_u r4_x_u(backM3)
    stt st_2_1 to (st_2_2) r1_2
               to (st_2_1) r4_x_u(nbackM3)
               to (st_1_1) r2_3 r3_4_u r4_x_u(backM3)
    stt st_2_2 to (st_2_1) r2_3_u r3_4_u r4_x_u(backM3)
               to (st_2_2) r4_x_u(nbackM3)

  aut M3
    stt st_0_0 to (st_0_1) r2_3 r2_3_u
    stt st_0_1 to (st_1_1) r2_3 r2_3_u
               to (st_0_0) r3_4 r4_x_u
```

```
    stt st_1_1 to (st_2_1) r2_3 r2_3_u
               to (st_0_1) r3_4 r4_x_u
    stt st_2_1 to (st_2_2) r2_3_b
               to (st_1_1) r3_4 r4_x_u
    stt st_2_2 to (st_2_1) r3_4_u r4_x_u

  aut M4
    stt st_0_0 to (st_0_1) r3_4 r3_4_u
    stt st_0_1 to (st_1_1) r3_4 r3_4_u
               to (st_0_0) r4_x
    stt st_1_1 to (st_2_1) r3_4 r3_4_u
               to (st_0_1) r4_x
    stt st_2_1 to (st_2_2) r3_4_b
               to (st_1_1) r4_x
    stt st_2_2 to (st_2_1) r4_x_u
```

APÊNDICE AC – ARQUIVO TEXTUAL DO MODELO PL4.SMV

```
MODULE ad_M2()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ---------------------------------------------------------------------------
MODULE ad_M3()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ---------------------------------------------------------------------------
MODULE ad_M4()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ---------------------------------------------------------------------------
MODULE main
VAR
  a_M2 : ad_M2();
  a_M3 : ad_M3();
  a_M4 : ad_M4();

INIT
  ! ( ( ( a_M2.state = s_st_0_0 ) & ( a_M3.state = s_st_2_2 ) ) ) & ! ( ( (
a_M3.state = s_st_0_0 ) & ( a_M4.state = s_st_2_2 ) ) )
TRANS
  next_local_step_ad_M2
  |  synchronized_steps
  |  next_local_step_ad_M3
  |  next_local_step_ad_M4
DEFINE
  a_M2_getCurrentIndex := (   a_M2.state = s_st_0_0 ? 0 :  (   a_M2.state =
s_st_0_1 ? 1 :  (   a_M2.state = s_st_1_1 ? 2 :  (   a_M2.state = s_st_2_1 ? 3 :
(   a_M2.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================= ad_M2 automata local steps section =================
  local_step_ad_M2 := (
    -- connection s_st_0_0 - s_st_0_1
    ( ( a_M2.state = s_st_0_0 & e_r1_2 ) & ( next(a_M2.state) = s_st_0_1 ) )
    |
    -- connection s_st_0_1 - s_st_1_1
    ( ( a_M2.state = s_st_0_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_1_1 ) )
    |
    -- connection s_st_1_1 - s_st_2_1
    ( ( a_M2.state = s_st_1_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M2.state = s_st_2_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M2 := ( local_step_ad_M2
    & a_M3.unchanged
    & a_M4.unchanged
  );
  a_M3_getCurrentIndex := (   a_M3.state = s_st_0_0 ? 0 :  (   a_M3.state =
s_st_0_1 ? 1 :  (   a_M3.state = s_st_1_1 ? 2 :  (   a_M3.state = s_st_2_1 ? 3 :
(   a_M3.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================= ad_M3 automata local steps section =================
```

```
  local_step_ad_M3 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M3.state = s_st_2_1 & e_r2_3_b ) & ( next(a_M3.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M3 := ( local_step_ad_M3
    & a_M2.unchanged
    & a_M4.unchanged
  );
  a_M4_getCurrentIndex := (   a_M4.state = s_st_0_0 ? 0 :  (   a_M4.state =
s_st_0_1 ? 1 :  (   a_M4.state = s_st_1_1 ? 2 :  (   a_M4.state = s_st_2_1 ? 3 :
(   a_M4.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================= ad_M4 automata local steps section =================
  local_step_ad_M4 := (
    -- connection s_st_0_1 - s_st_0_0
    ( ( a_M4.state = s_st_0_1 & e_r4_x ) & ( next(a_M4.state) = s_st_0_0 ) )
    |
    -- connection s_st_1_1 - s_st_0_1
    ( ( a_M4.state = s_st_1_1 & e_r4_x ) & ( next(a_M4.state) = s_st_0_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M4.state = s_st_2_1 & e_r3_4_b ) & ( next(a_M4.state) = s_st_2_2 ) )
    |
    -- connection s_st_2_1 - s_st_1_1
    ( ( a_M4.state = s_st_2_1 & e_r4_x ) & ( next(a_M4.state) = s_st_1_1 ) )
  );
  next_local_step_ad_M4 := ( local_step_ad_M4
    & a_M2.unchanged
    & a_M3.unchanged
  );
  -- ================= SYNCHRONIZED STEPS =================
  synchronized_steps := (
    (
        e_r2_3
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
        )
        & a_M4.unchanged
    )
    |
    (
        e_r2_3_u
        &
        (
        -- connection s_st_2_2 - s_st_2_1
```

```
     ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
    )
    & a_M4.unchanged
)
|
(
    e_r3_4
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
    -- connection s_st_1_1 - s_st_0_1
    | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
    -- connection s_st_2_1 - s_st_1_1
    | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
    )
    & a_M2.unchanged
)
|
(
    e_r3_4_u
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
    -- connection s_st_1_1 - s_st_0_1
    | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
    -- connection s_st_2_1 - s_st_1_1
    | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
    -- connection s_st_2_2 - s_st_2_1
    | ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_2_2 - s_st_2_1
     ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
```

```
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
        )
    )
    |
    (
        e_r4_x_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M4.state = s_st_2_2 & next(a_M4.state) = s_st_2_1 )
        )
    )
  );
  -- ================== identifiers section ==================
  i_mu1 := ( 1 );
```

```
i_mu2 := ( 1 );

i_mu3 := ( 1 );

i_mu4 := ( 1 );

i_f2_3_b := ( ( ( ( a_M2.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu2 );

i_f3_4_b := ( ( ( ( a_M3.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu3 );

i_backM3 := ( ( ( a_M3.state = s_st_2_2 ) ? 1 : 0 ) );

i_nbackM3 := ( ( ! ( ( i_backM3 ) > 0 ) ? 1 : 0 ) );

-- ================== nb operators section ==================
-- ================== events section ==================
e_r1_2 := ( i_mu1 > 0 );

e_r2_3 := ( i_mu2 > 0 );

e_r2_3_b := ( i_f2_3_b > 0 );

e_r2_3_u := ( i_mu2 > 0 );

e_r3_4 := ( i_mu3 > 0 );

e_r3_4_b := ( i_f3_4_b > 0 );

e_r3_4_u := ( i_mu3 > 0 );

e_r4_x := ( i_mu4 > 0 );

e_r4_x_u := ( i_mu4 > 0 );
```

APÊNDICE AD – ARQUIVO TEXTUAL DO MODELO PL5.SAN

```
identifiers

  // Service rates
  mu1 = 1;
  mu2 = 1;
  mu3 = 1;
  mu4 = 1;
  mu5 = 1;

  // Blocking functions (allow that station "j" goes to the "blocking" state with
a rate "mu_i", indicating that station "i" is blocked)
  f2_3_b = (st M2 != st_0_0) * mu2;
  f3_4_b = (st M3 != st_0_0) * mu3;
  f4_5_b = (st M4 != st_0_0) * mu4;

  // Functional probabilities to allow back transitions in station 2
  backM3 = ((st M3 == st_2_2));
  backM3_4 = ((st M3 == st_2_2) && (st M4 == st_2_2));
  // Functional probabilities to allow back transitions in station 3
  backM4 = ((st M4 == st_2_2));

  // Functional probabilities to avoid back transitions in station 2
  nbackM3 = !(backM3);
  nbackM3_4 = !(backM3_4);
  // Functional probabilities to avoid back transitions in station 3
  nbackM4 = !(backM4);

events

  loc r1_2               (mu1);
  syn r2_3               (mu2);
  loc r2_3_b       (f2_3_b);
  syn r2_3_u       (mu2);
  syn r3_4               (mu3);
  loc r3_4_b       (f3_4_b);
  syn r3_4_u       (mu3);
  syn r4_5               (mu4);
  loc r4_5_b       (f4_5_b);
  syn r4_5_u       (mu4);
  loc r5_x               (mu5);
  syn r5_x_u       (mu5);

reachability = !((st M2 == st_0_0) && (st M3 == st_2_2)) && !((st M3 == st_0_0) &&
(st M4 == st_2_2)) && !((st M4 == st_0_0) && (st M5 == st_2_2));

network B2_2_B3_2_B4_2_B5_2 (continuous)

  aut M2
    stt st_0_0 to (st_0_1) r1_2
               to (st_0_0) r4_5_u r5_x_u
    stt st_0_1 to (st_1_1) r1_2
               to (st_0_1) r4_5_u(nbackM3) r5_x_u(nbackM3_4)
               to (st_0_0) r2_3 r3_4_u r4_5_u(backM3) r5_x_u(backM3_4)
    stt st_1_1 to (st_2_1) r1_2
               to (st_1_1) r4_5_u(nbackM3) r5_x_u(nbackM3_4)
```

```
            to (st_0_1) r2_3 r3_4_u r4_5_u(backM3) r5_x_u(backM3_4)
  stt st_2_1 to (st_2_2) r1_2
            to (st_2_1) r4_5_u(nbackM3) r5_x_u(nbackM3_4)
            to (st_1_1) r2_3 r3_4_u r4_5_u(backM3) r5_x_u(backM3_4)
  stt st_2_2 to (st_2_1) r2_3_u r3_4_u r4_5_u(backM3) r5_x_u(backM3_4)
            to (st_2_2) r4_5_u(nbackM3) r5_x_u(nbackM3_4)

aut M3
  stt st_0_0 to (st_0_1) r2_3 r2_3_u
            to (st_0_0) r5_x_u
  stt st_0_1 to (st_1_1) r2_3 r2_3_u
            to (st_0_1) r5_x_u(nbackM4)
            to (st_0_0) r3_4 r4_5_u r5_x_u(backM4)
  stt st_1_1 to (st_2_1) r2_3 r2_3_u
            to (st_1_1) r5_x_u(nbackM4)
            to (st_0_1) r3_4 r4_5_u r5_x_u(backM4)
  stt st_2_1 to (st_2_2) r2_3_b
            to (st_2_1) r5_x_u(nbackM4)
            to (st_1_1) r3_4 r4_5_u r5_x_u(backM4)
  stt st_2_2 to (st_2_1) r3_4_u r4_5_u r5_x_u(backM4)
            to (st_2_2) r5_x_u(nbackM4)

aut M4
  stt st_0_0 to (st_0_1) r3_4 r3_4_u
  stt st_0_1 to (st_1_1) r3_4 r3_4_u
            to (st_0_0) r4_5 r5_x_u
  stt st_1_1 to (st_2_1) r3_4 r3_4_u
            to (st_0_1) r4_5 r5_x_u
  stt st_2_1 to (st_2_2) r3_4_b
            to (st_1_1) r4_5 r5_x_u
  stt st_2_2 to (st_2_1) r4_5_u r5_x_u

aut M5
  stt st_0_0 to (st_0_1) r4_5 r4_5_u
  stt st_0_1 to (st_1_1) r4_5 r4_5_u
            to (st_0_0) r5_x
  stt st_1_1 to (st_2_1) r4_5 r4_5_u
            to (st_0_1) r5_x
  stt st_2_1 to (st_2_2) r4_5_b
            to (st_1_1) r5_x
  stt st_2_2 to (st_2_1) r5_x_u
```

APÊNDICE AE – ARQUIVO TEXTUAL DO MODELO PL5.SMV

```
MODULE ad_M2()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M3()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M4()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M5()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_M2 : ad_M2();
  a_M3 : ad_M3();
  a_M4 : ad_M4();
  a_M5 : ad_M5();

INIT
  ( ! ( ( ( a_M2.state = s_st_0_0 ) & ( a_M3.state = s_st_2_2 ) ) ) & ! ( ( (
a_M3.state = s_st_0_0 ) & ( a_M4.state = s_st_2_2 ) ) ) & ! ( ( ( a_M4.state =
s_st_0_0 ) & ( a_M5.state = s_st_2_2 ) ) )
TRANS
  next_local_step_ad_M2
  |  synchronized_steps
  |  next_local_step_ad_M3
  |  next_local_step_ad_M4
  |  next_local_step_ad_M5
DEFINE
  a_M2_getCurrentIndex := (   a_M2.state = s_st_0_0 ? 0 :  (   a_M2.state =
s_st_0_1 ? 1 :  (   a_M2.state = s_st_1_1 ? 2 :  (   a_M2.state = s_st_2_1 ? 3 :
(   a_M2.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M2 automata local steps section ==================
  local_step_ad_M2 := (
    -- connection s_st_0_0 - s_st_0_1
    ( ( a_M2.state = s_st_0_0 & e_r1_2 ) & ( next(a_M2.state) = s_st_0_1 ) )
    |
    -- connection s_st_0_1 - s_st_1_1
    ( ( a_M2.state = s_st_0_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_1_1 ) )
    |
    -- connection s_st_1_1 - s_st_2_1
    ( ( a_M2.state = s_st_1_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M2.state = s_st_2_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_2 ) )
  );
```

```
  next_local_step_ad_M2 := ( local_step_ad_M2
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
  );
  a_M3_getCurrentIndex := (   a_M3.state = s_st_0_0 ? 0 :  (   a_M3.state =
s_st_0_1 ? 1 :  (   a_M3.state = s_st_1_1 ? 2 :  (   a_M3.state = s_st_2_1 ? 3 :
(   a_M3.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M3 automata local steps section ==================
  local_step_ad_M3 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M3.state = s_st_2_1 & e_r2_3_b ) & ( next(a_M3.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M3 := ( local_step_ad_M3
    & a_M2.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
  );
  a_M4_getCurrentIndex := (   a_M4.state = s_st_0_0 ? 0 :  (   a_M4.state =
s_st_0_1 ? 1 :  (   a_M4.state = s_st_1_1 ? 2 :  (   a_M4.state = s_st_2_1 ? 3 :
(   a_M4.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M4 automata local steps section ==================
  local_step_ad_M4 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M4.state = s_st_2_1 & e_r3_4_b ) & ( next(a_M4.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M4 := ( local_step_ad_M4
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M5.unchanged
  );
  a_M5_getCurrentIndex := (   a_M5.state = s_st_0_0 ? 0 :  (   a_M5.state =
s_st_0_1 ? 1 :  (   a_M5.state = s_st_1_1 ? 2 :  (   a_M5.state = s_st_2_1 ? 3 :
(   a_M5.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M5 automata local steps section ==================
  local_step_ad_M5 := (
    -- connection s_st_0_1 - s_st_0_0
    ( ( a_M5.state = s_st_0_1 & e_r5_x ) & ( next(a_M5.state) = s_st_0_0 ) )
    |
    -- connection s_st_1_1 - s_st_0_1
    ( ( a_M5.state = s_st_1_1 & e_r5_x ) & ( next(a_M5.state) = s_st_0_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M5.state = s_st_2_1 & e_r4_5_b ) & ( next(a_M5.state) = s_st_2_2 ) )
    |
    -- connection s_st_2_1 - s_st_1_1
    ( ( a_M5.state = s_st_2_1 & e_r5_x ) & ( next(a_M5.state) = s_st_1_1 ) )
  );
  next_local_step_ad_M5 := ( local_step_ad_M5
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_r2_3
        &
        (
```

```
        -- connection s_st_0_1 - s_st_0_0
         ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
        )
        & a_M4.unchanged
        & a_M5.unchanged
    )
    |
    (
        e_r2_3_u
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
        )
        & a_M4.unchanged
        & a_M5.unchanged
    )
    |
    (
        e_r3_4
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
```

```
    )
    & a_M2.unchanged
    & a_M5.unchanged
)
|
(
    e_r3_4_u
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
    -- connection s_st_1_1 - s_st_0_1
    | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
    -- connection s_st_2_1 - s_st_1_1
    | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
    -- connection s_st_2_2 - s_st_2_1
    | ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_2_2 - s_st_2_1
     ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
    )
    & a_M5.unchanged
)
|
(
    e_r4_5
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_0_0 )
    -- connection s_st_1_1 - s_st_0_1
    | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_0_1 )
    -- connection s_st_2_1 - s_st_1_1
    | ( a_M4.state = s_st_2_1 & next(a_M4.state) = s_st_1_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_2_1 )
    )
    & a_M2.unchanged
    & a_M3.unchanged
)
|
```

```
    (
        e_r4_5_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M4.state = s_st_2_2 & next(a_M4.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_2_1 )
        )
    )
    |
    (
```

```
        e_r5_x_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
```

```
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & next(a_M4.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M4.state = s_st_2_2 & next(a_M4.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M5.state = s_st_2_2 & next(a_M5.state) = s_st_2_1 )
        )
    )
  );
  -- ================== identifiers section ==================
  i_mu1 := ( 1 );

  i_mu2 := ( 1 );

  i_mu3 := ( 1 );

  i_mu4 := ( 1 );

  i_mu5 := ( 1 );

  i_f2_3_b := ( ( ( ( a_M2.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu2 );

  i_f3_4_b := ( ( ( ( a_M3.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu3 );

  i_f4_5_b := ( ( ( ( a_M4.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu4 );

  i_backM3 := ( ( ( a_M3.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM3_4 := ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM4 := ( ( ( a_M4.state = s_st_2_2 ) ? 1 : 0 ) );

  i_nbackM3 := ( ( ! ( ( i_backM3 ) > 0 ) ? 1 : 0 ) );

  i_nbackM3_4 := ( ( ! ( ( i_backM3_4 ) > 0 ) ? 1 : 0 ) );

  i_nbackM4 := ( ( ! ( ( i_backM4 ) > 0 ) ? 1 : 0 ) );

  -- ================== nb operators section ==================
  -- ================== events section ==================
  e_r1_2 := ( i_mu1 > 0 );

  e_r2_3 := ( i_mu2 > 0 );

  e_r2_3_b := ( i_f2_3_b > 0 );

  e_r2_3_u := ( i_mu2 > 0 );

  e_r3_4 := ( i_mu3 > 0 );
```

```
e_r3_4_b := ( i_f3_4_b > 0 );

e_r3_4_u := ( i_mu3 > 0 );

e_r4_5 := ( i_mu4 > 0 );

e_r4_5_b := ( i_f4_5_b > 0 );

e_r4_5_u := ( i_mu4 > 0 );

e_r5_x := ( i_mu5 > 0 );

e_r5_x_u := ( i_mu5 > 0 );
```

APÊNDICE AF – ARQUIVO TEXTUAL DO MODELO PL10.SAN

```
identifiers

  // Service rates
  mu1 = 1;
  mu2 = 1;
  mu3 = 1;
  mu4 = 1;
  mu5 = 1;
  mu6 = 1;
  mu7 = 1;
  mu8 = 1;
  mu9 = 1;
  mu10 = 1;

  // Blocking functions (allow that station "j" goes to the "blocking" state with
a rate "mu_i", indicating that station "i" is blocked)
  f2_3_b = (st M2 != st_0_0) * mu2;
  f3_4_b = (st M3 != st_0_0) * mu3;
  f4_5_b = (st M4 != st_0_0) * mu4;
  f5_6_b = (st M5 != st_0_0) * mu5;
  f6_7_b = (st M6 != st_0_0) * mu6;
  f7_8_b = (st M7 != st_0_0) * mu7;
  f8_9_b = (st M8 != st_0_0) * mu8;
  f9_10_b = ( st M9 != st_0_0 );

  // Functional probabilities to allow back transitions in station 2
  backM3 = ((st M3 == st_2_2));
  backM3_4 = ((st M3 == st_2_2) && (st M4 == st_2_2));
  backM3_4_5 = ((st M3 == st_2_2) && (st M4 == st_2_2) && (st M5 == st_2_2));
  backM3_4_5_6 = ((st M3 == st_2_2) && (st M4 == st_2_2) && (st M5 == st_2_2) &&
(st M6 == st_2_2));
  backM3_4_5_6_7 = ((st M3 == st_2_2) && (st M4 == st_2_2) && (st M5 == st_2_2) &&
(st M6 == st_2_2) && (st M7 == st_2_2));
  backM3_4_5_6_7_8 = ((st M3 == st_2_2) && (st M4 == st_2_2) && (st M5 == st_2_2)
&& (st M6 == st_2_2) && (st M7 == st_2_2) && (st M8 == st_2_2));
  backM3_4_5_6_7_8_9 = ((st M3 == st_2_2) && (st M4 == st_2_2) && (st M5 ==
st_2_2) && (st M6 == st_2_2) && (st M7 == st_2_2) && (st M8 == st_2_2) && (st M9
== st_2_2));
  // Functional probabilities to allow back transitions in station 3
  backM4 = ((st M4 == st_2_2));
  backM4_5 = ((st M4 == st_2_2) && (st M5 == st_2_2));
  backM4_5_6 = ((st M4 == st_2_2) && (st M5 == st_2_2) && (st M6 == st_2_2));
  backM4_5_6_7 = ((st M4 == st_2_2) && (st M5 == st_2_2) && (st M6 == st_2_2) &&
(st M7 == st_2_2));
  backM4_5_6_7_8 = ((st M4 == st_2_2) && (st M5 == st_2_2) && (st M6 == st_2_2) &&
(st M7 == st_2_2) && (st M8 == st_2_2));
  backM4_5_6_7_8_9 = ((st M4 == st_2_2) && (st M5 == st_2_2) && (st M6 == st_2_2)
&& (st M7 == st_2_2) && (st M8 == st_2_2) && (st M9 == st_2_2));
  // Functional probabilities to allow back transitions in station 4
  backM5 = ((st M5 == st_2_2));
  backM5_6 = ((st M5 == st_2_2) && (st M6 == st_2_2));
  backM5_6_7 = ((st M5 == st_2_2) && (st M6 == st_2_2) && (st M7 == st_2_2));
  backM5_6_7_8 = ((st M5 == st_2_2) && (st M6 == st_2_2) && (st M7 == st_2_2) &&
(st M8 == st_2_2));
```

```
  backM5_6_7_8_9 = ((st M5 == st_2_2) && (st M6 == st_2_2) && (st M7 == st_2_2) &&
(st M8 == st_2_2) && (st M9 == st_2_2));
  // Functional probabilities to allow back transitions in station 5
  backM6 = ((st M6 == st_2_2));
  backM6_7 = ((st M6 == st_2_2) && (st M7 == st_2_2));
  backM6_7_8 = ((st M6 == st_2_2) && (st M7 == st_2_2) && (st M8 == st_2_2));
  backM6_7_8_9 = ((st M6 == st_2_2) && (st M7 == st_2_2) && (st M8 == st_2_2) &&
(st M9 == st_2_2));
  // Functional probabilities to allow back transitions in station 6
  backM7 = ((st M7 == st_2_2));
  backM7_8 = ((st M7 == st_2_2) && (st M8 == st_2_2));
  backM7_8_9 = ((st M7 == st_2_2) && (st M8 == st_2_2) && (st M9 == st_2_2));
  // Functional probabilities to allow back transitions in station 7
  backM8 = ((st M8 == st_2_2));
  backM8_9 = ((st M8 == st_2_2) && (st M9 == st_2_2));
  // Functional probabilities to allow back transitions in station 8
  backM9 = ((st M9 == st_2_2));

  // Functional probabilities to avoid back transitions in station 2
  nbackM3 = !(backM3);
  nbackM3_4 = !(backM3_4);
  nbackM3_4_5 = !(backM3_4_5);
  nbackM3_4_5_6 = !(backM3_4_5_6);
  nbackM3_4_5_6_7 = !(backM3_4_5_6_7);
  nbackM3_4_5_6_7_8 = !(backM3_4_5_6_7_8);
  nbackM3_4_5_6_7_8_9 = !(backM3_4_5_6_7_8_9);
  // Functional probabilities to avoid back transitions in station 3
  nbackM4 = !(backM4);
  nbackM4_5 = !(backM4_5);
  nbackM4_5_6 = !(backM4_5_6);
  nbackM4_5_6_7 = !(backM4_5_6_7);
  nbackM4_5_6_7_8 = !(backM4_5_6_7_8);
  nbackM4_5_6_7_8_9 = !(backM4_5_6_7_8_9);
  // Functional probabilities to avoid back transitions in station 4
  nbackM5 = !(backM5);
  nbackM5_6 = !(backM5_6);
  nbackM5_6_7 = !(backM5_6_7);
  nbackM5_6_7_8 = !(backM5_6_7_8);
  nbackM5_6_7_8_9 = !(backM5_6_7_8_9);
  // Functional probabilities to avoid back transitions in station 5
  nbackM6 = !(backM6);
  nbackM6_7 = !(backM6_7);
  nbackM6_7_8 = !(backM6_7_8);
  nbackM6_7_8_9 = !(backM6_7_8_9);
  // Functional probabilities to avoid back transitions in station 6
  nbackM7 = !(backM7);
  nbackM7_8 = !(backM7_8);
  nbackM7_8_9 = !(backM7_8_9);
  // Functional probabilities to avoid back transitions in station 7
  nbackM8 = !(backM8);
  nbackM8_9 = !(backM8_9);
  // Functional probabilities to avoid back transitions in station 8
  nbackM9 = !(backM9);

events

  loc r1_2               (mu1);
  syn r2_3               (mu2);
  loc r2_3_b       (f2_3_b);
```

```
  syn r2_3_u       (mu2);
  syn r3_4               (mu3);
  loc r3_4_b       (f3_4_b);
  syn r3_4_u       (mu3);
  syn r4_5               (mu4);
  loc r4_5_b       (f4_5_b);
  syn r4_5_u       (mu4);
  syn r5_6               (mu5);
  loc r5_6_b       (f5_6_b);
  syn r5_6_u       (mu5);
  syn r6_7               (mu6);
  loc r6_7_b       (f6_7_b);
  syn r6_7_u       (mu6);
  syn r7_8               (mu7);
  loc r7_8_b       (f7_8_b);
  syn r7_8_u       (mu7);
  syn r8_9               (mu8);
  loc r8_9_b       (f8_9_b);
  syn r8_9_u       (mu8);
  syn r9_10              (mu9);
  loc r9_10_b            (f9_10_b);
  syn r9_10_u            (mu9);
  loc r10_x              (mu10);
  syn r10_x_u            (mu10);

partial reachability = !((st M2 == st_0_0) && (st M3 == st_2_2)) && !((st M3 ==
st_0_0) && (st M4 == st_2_2)) && !((st M4 == st_0_0) && (st M5 == st_2_2)) &&
!((st M5 == st_0_0) && (st M6 == st_2_2)) && !((st M6 == st_0_0) && (st M7 ==
st_2_2)) && !((st M7 == st_0_0) && (st M8 == st_2_2)) && !((st M8 == st_0_0) &&
(st M9 == st_2_2)) && !((st M9 == st_0_0) && (st M10 == st_2_2));

network B2_2_B3_2_B4_2_B5_2_B6_2_B7_2_B8_2_B9_2_B10_2 (continuous)

  aut M2
    stt st_0_0 to (st_0_1) r1_2
               to (st_0_0) r4_5_u r5_6_u r6_7_u r7_8_u r8_9_u r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r1_2
               to (st_0_1) r4_5_u(nbackM3) r5_6_u(nbackM3_4) r6_7_u(nbackM3_4_5)
r7_8_u(nbackM3_4_5_6) r8_9_u(nbackM3_4_5_6_7) r9_10_u(nbackM3_4_5_6_7_8)
r10_x_u(nbackM3_4_5_6_7_8_9)
               to (st_0_0) r2_3 r3_4_u r4_5_u(backM3) r5_6_u(backM3_4)
r6_7_u(backM3_4_5) r7_8_u(backM3_4_5_6) r8_9_u(backM3_4_5_6_7)
r9_10_u(backM3_4_5_6_7_8) r10_x_u(backM3_4_5_6_7_8_9)
    stt st_1_1 to (st_2_1) r1_2
               to (st_1_1) r4_5_u(nbackM3) r5_6_u(nbackM3_4) r6_7_u(nbackM3_4_5)
r7_8_u(nbackM3_4_5_6) r8_9_u(nbackM3_4_5_6_7) r9_10_u(nbackM3_4_5_6_7_8)
r10_x_u(nbackM3_4_5_6_7_8_9)
               to (st_0_1) r2_3 r3_4_u r4_5_u(backM3) r5_6_u(backM3_4)
r6_7_u(backM3_4_5) r7_8_u(backM3_4_5_6) r8_9_u(backM3_4_5_6_7)
r9_10_u(backM3_4_5_6_7_8) r10_x_u(backM3_4_5_6_7_8_9)
    stt st_2_1 to (st_2_2) r1_2
               to (st_2_1) r4_5_u(nbackM3) r5_6_u(nbackM3_4) r6_7_u(nbackM3_4_5)
r7_8_u(nbackM3_4_5_6) r8_9_u(nbackM3_4_5_6_7) r9_10_u(nbackM3_4_5_6_7_8)
r10_x_u(nbackM3_4_5_6_7_8_9)
               to (st_1_1) r2_3 r3_4_u r4_5_u(backM3) r5_6_u(backM3_4)
r6_7_u(backM3_4_5) r7_8_u(backM3_4_5_6) r8_9_u(backM3_4_5_6_7)
r9_10_u(backM3_4_5_6_7_8) r10_x_u(backM3_4_5_6_7_8_9)
```

```
    stt st_2_2 to (st_2_1) r2_3_u r3_4_u r4_5_u(backM3) r5_6_u(backM3_4)
r6_7_u(backM3_4_5) r7_8_u(backM3_4_5_6) r8_9_u(backM3_4_5_6_7)
r9_10_u(backM3_4_5_6_7_8) r10_x_u(backM3_4_5_6_7_8_9)
                to (st_2_2) r4_5_u(nbackM3) r5_6_u(nbackM3_4) r6_7_u(nbackM3_4_5)
r7_8_u(nbackM3_4_5_6) r8_9_u(nbackM3_4_5_6_7) r9_10_u(nbackM3_4_5_6_7_8)
r10_x_u(nbackM3_4_5_6_7_8_9)

  aut M3
    stt st_0_0 to (st_0_1) r2_3 r2_3_u
                to (st_0_0) r5_6_u r6_7_u r7_8_u r8_9_u r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r2_3 r2_3_u
                to (st_0_1) r5_6_u(nbackM4) r6_7_u(nbackM4_5) r7_8_u(nbackM4_5_6)
r8_9_u(nbackM4_5_6_7) r9_10_u(nbackM4_5_6_7_8) r10_x_u(nbackM4_5_6_7_8_9)
                to (st_0_0) r3_4 r4_5_u r5_6_u(backM4) r6_7_u(backM4_5)
r7_8_u(backM4_5_6) r8_9_u(backM4_5_6_7) r9_10_u(backM4_5_6_7_8)
r10_x_u(backM4_5_6_7_8_9)
    stt st_1_1 to (st_2_1) r2_3 r2_3_u
                to (st_1_1) r5_6_u(nbackM4) r6_7_u(nbackM4_5) r7_8_u(nbackM4_5_6)
r8_9_u(nbackM4_5_6_7) r9_10_u(nbackM4_5_6_7_8) r10_x_u(nbackM4_5_6_7_8_9)
                to (st_0_1) r3_4 r4_5_u r5_6_u(backM4) r6_7_u(backM4_5)
r7_8_u(backM4_5_6) r8_9_u(backM4_5_6_7) r9_10_u(backM4_5_6_7_8)
r10_x_u(backM4_5_6_7_8_9)
    stt st_2_1 to (st_2_2) r2_3_b
                to (st_2_1) r5_6_u(nbackM4) r6_7_u(nbackM4_5) r7_8_u(nbackM4_5_6)
r8_9_u(nbackM4_5_6_7) r9_10_u(nbackM4_5_6_7_8) r10_x_u(nbackM4_5_6_7_8_9)
                to (st_1_1) r3_4 r4_5_u r5_6_u(backM4) r6_7_u(backM4_5)
r7_8_u(backM4_5_6) r8_9_u(backM4_5_6_7) r9_10_u(backM4_5_6_7_8)
r10_x_u(backM4_5_6_7_8_9)
    stt st_2_2 to (st_2_1) r3_4_u r4_5_u r5_6_u(backM4) r6_7_u(backM4_5)
r7_8_u(backM4_5_6) r8_9_u(backM4_5_6_7) r9_10_u(backM4_5_6_7_8)
r10_x_u(backM4_5_6_7_8_9)
                to (st_2_2) r5_6_u(nbackM4) r6_7_u(nbackM4_5) r7_8_u(nbackM4_5_6)
r8_9_u(nbackM4_5_6_7) r9_10_u(nbackM4_5_6_7_8) r10_x_u(nbackM4_5_6_7_8_9)

  aut M4
    stt st_0_0 to (st_0_1) r3_4 r3_4_u
                to (st_0_0) r6_7_u r7_8_u r8_9_u r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r3_4 r3_4_u
                to (st_0_1) r6_7_u(nbackM5) r7_8_u(nbackM5_6) r8_9_u(nbackM5_6_7)
r9_10_u(nbackM5_6_7_8) r10_x_u(nbackM5_6_7_8_9)
                to (st_0_0) r4_5 r5_6_u r6_7_u(backM5) r7_8_u(backM5_6)
r8_9_u(backM5_6_7) r9_10_u(backM5_6_7_8) r10_x_u(backM5_6_7_8_9)
    stt st_1_1 to (st_2_1) r3_4 r3_4_u
                to (st_1_1) r6_7_u(nbackM5) r7_8_u(nbackM5_6) r8_9_u(nbackM5_6_7)
r9_10_u(nbackM5_6_7_8) r10_x_u(nbackM5_6_7_8_9)
                to (st_0_1) r4_5 r5_6_u r6_7_u(backM5) r7_8_u(backM5_6)
r8_9_u(backM5_6_7) r9_10_u(backM5_6_7_8) r10_x_u(backM5_6_7_8_9)
    stt st_2_1 to (st_2_2) r3_4_b
                to (st_2_1) r6_7_u(nbackM5) r7_8_u(nbackM5_6) r8_9_u(nbackM5_6_7)
r9_10_u(nbackM5_6_7_8) r10_x_u(nbackM5_6_7_8_9)
                to (st_1_1) r4_5 r5_6_u r6_7_u(backM5) r7_8_u(backM5_6)
r8_9_u(backM5_6_7) r9_10_u(backM5_6_7_8) r10_x_u(backM5_6_7_8_9)
    stt st_2_2 to (st_2_1) r4_5_u r5_6_u r6_7_u(backM5) r7_8_u(backM5_6)
r8_9_u(backM5_6_7) r9_10_u(backM5_6_7_8) r10_x_u(backM5_6_7_8_9)
                to (st_2_2) r6_7_u(nbackM5) r7_8_u(nbackM5_6) r8_9_u(nbackM5_6_7)
r9_10_u(nbackM5_6_7_8) r10_x_u(nbackM5_6_7_8_9)

  aut M5
    stt st_0_0 to (st_0_1) r4_5 r4_5_u
```

```
              to (st_0_0) r7_8_u r8_9_u r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r4_5 r4_5_u
              to (st_0_1) r7_8_u(nbackM6) r8_9_u(nbackM6_7) r9_10_u(nbackM6_7_8)
r10_x_u(nbackM6_7_8_9)
              to (st_0_0) r5_6 r6_7_u r7_8_u(backM6) r8_9_u(backM6_7)
r9_10_u(backM6_7_8) r10_x_u(backM6_7_8_9)
    stt st_1_1 to (st_2_1) r4_5 r4_5_u
              to (st_1_1) r7_8_u(nbackM6) r8_9_u(nbackM6_7) r9_10_u(nbackM6_7_8)
r10_x_u(nbackM6_7_8_9)
              to (st_0_1) r5_6 r6_7_u r7_8_u(backM6) r8_9_u(backM6_7)
r9_10_u(backM6_7_8) r10_x_u(backM6_7_8_9)
    stt st_2_1 to (st_2_2) r4_5_b
              to (st_2_1) r7_8_u(nbackM6) r8_9_u(nbackM6_7) r9_10_u(nbackM6_7_8)
r10_x_u(nbackM6_7_8_9)
              to (st_1_1) r5_6 r6_7_u r7_8_u(backM6) r8_9_u(backM6_7)
r9_10_u(backM6_7_8) r10_x_u(backM6_7_8_9)
    stt st_2_2 to (st_2_1) r5_6_u r6_7_u r7_8_u(backM6) r8_9_u(backM6_7)
r9_10_u(backM6_7_8) r10_x_u(backM6_7_8_9)
              to (st_2_2) r7_8_u(nbackM6) r8_9_u(nbackM6_7) r9_10_u(nbackM6_7_8)
r10_x_u(nbackM6_7_8_9)

  aut M6
    stt st_0_0 to (st_0_1) r5_6 r5_6_u
              to (st_0_0) r8_9_u r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r5_6 r5_6_u
              to (st_0_1) r8_9_u(nbackM7) r9_10_u(nbackM7_8) r10_x_u(nbackM7_8_9)
              to (st_0_0) r6_7 r7_8_u r8_9_u(backM7) r9_10_u(backM7_8)
r10_x_u(backM7_8_9)
    stt st_1_1 to (st_2_1) r5_6 r5_6_u
              to (st_1_1) r8_9_u(nbackM7) r9_10_u(nbackM7_8) r10_x_u(nbackM7_8_9)
              to (st_0_1) r6_7 r7_8_u r8_9_u(backM7) r9_10_u(backM7_8)
r10_x_u(backM7_8_9)
    stt st_2_1 to (st_2_2) r5_6_b
              to (st_2_1) r8_9_u(nbackM7) r9_10_u(nbackM7_8) r10_x_u(nbackM7_8_9)
              to (st_1_1) r6_7 r7_8_u r8_9_u(backM7) r9_10_u(backM7_8)
r10_x_u(backM7_8_9)
    stt st_2_2 to (st_2_1) r6_7_u r7_8_u r8_9_u(backM7) r9_10_u(backM7_8)
r10_x_u(backM7_8_9)
              to (st_2_2) r8_9_u(nbackM7) r9_10_u(nbackM7_8) r10_x_u(nbackM7_8_9)

  aut M7
    stt st_0_0 to (st_0_1) r6_7 r6_7_u
              to (st_0_0) r9_10_u r10_x_u
    stt st_0_1 to (st_1_1) r6_7 r6_7_u
              to (st_0_1) r9_10_u(nbackM8) r10_x_u(nbackM8_9)
              to (st_0_0) r7_8 r8_9_u r9_10_u(backM8) r10_x_u(backM8_9)
    stt st_1_1 to (st_2_1) r6_7 r6_7_u
              to (st_1_1) r9_10_u(nbackM8) r10_x_u(nbackM8_9)
              to (st_0_1) r7_8 r8_9_u r9_10_u(backM8) r10_x_u(backM8_9)
    stt st_2_1 to (st_2_2) r6_7_b
              to (st_2_1) r9_10_u(nbackM8) r10_x_u(nbackM8_9)
              to (st_1_1) r7_8 r8_9_u r9_10_u(backM8) r10_x_u(backM8_9)
    stt st_2_2 to (st_2_1) r7_8_u r8_9_u r9_10_u(backM8) r10_x_u(backM8_9)
              to (st_2_2) r9_10_u(nbackM8) r10_x_u(nbackM8_9)

  aut M8
    stt st_0_0 to (st_0_1) r7_8 r7_8_u
              to (st_0_0) r10_x_u
    stt st_0_1 to (st_1_1) r7_8 r7_8_u
```

```
            to (st_0_1) r10_x_u(nbackM9)
            to (st_0_0) r8_9 r9_10_u r10_x_u(backM9)
  stt st_1_1 to (st_2_1) r7_8 r7_8_u
            to (st_1_1) r10_x_u(nbackM9)
            to (st_0_1) r8_9 r9_10_u r10_x_u(backM9)
  stt st_2_1 to (st_2_2) r7_8_b
            to (st_2_1) r10_x_u(nbackM9)
            to (st_1_1) r8_9 r9_10_u r10_x_u(backM9)
  stt st_2_2 to (st_2_1) r8_9_u r9_10_u r10_x_u(backM9)
            to (st_2_2) r10_x_u(nbackM9)

aut M9
  stt st_0_0 to (st_0_1) r8_9 r8_9_u
  stt st_0_1 to (st_1_1) r8_9 r8_9_u
            to (st_0_0) r9_10 r10_x_u
  stt st_1_1 to (st_2_1) r8_9 r8_9_u
            to (st_0_1) r9_10 r10_x_u
  stt st_2_1 to (st_2_2) r8_9_b
            to (st_1_1) r9_10 r10_x_u
  stt st_2_2 to (st_2_1) r9_10_u r10_x_u

aut M10
  stt st_0_0 to (st_0_1) r9_10 r9_10_u
  stt st_0_1 to (st_1_1) r9_10 r9_10_u
            to (st_0_0) r10_x
  stt st_1_1 to (st_2_1) r9_10 r9_10_u
            to (st_0_1) r10_x
  stt st_2_1 to (st_2_2) r9_10_b
            to (st_1_1) r10_x
  stt st_2_2 to (st_2_1) r10_x_u
```

APÊNDICE AG – ARQUIVO TEXTUAL DO MODELO PL10.SMV

```
MODULE ad_M2()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M3()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M4()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M5()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M6()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M7()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M8()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M9()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_M10()
VAR state : {s_st_0_0,s_st_0_1,s_st_1_1,s_st_2_1,s_st_2_2};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_M2 : ad_M2();
  a_M3 : ad_M3();
  a_M4 : ad_M4();
  a_M5 : ad_M5();
  a_M6 : ad_M6();
  a_M7 : ad_M7();
  a_M8 : ad_M8();
  a_M9 : ad_M9();
```

```
  a_M10 : ad_M10();

INIT
 ( ( ( ( ( ( ! ( ( ( a_M2.state = s_st_0_0 ) & ( a_M3.state = s_st_2_2 ) ) ) & !
( ( ( a_M3.state = s_st_0_0 ) & ( a_M4.state = s_st_2_2 ) ) ) ) & ! ( ( (
a_M4.state = s_st_0_0 ) & ( a_M5.state = s_st_2_2 ) ) ) ) & ! ( ( ( a_M5.state =
s_st_0_0 ) & ( a_M6.state = s_st_2_2 ) ) ) ) & ! ( ( ( a_M6.state = s_st_0_0 ) & (
a_M7.state = s_st_2_2 ) ) ) ) & ! ( ( ( a_M7.state = s_st_0_0 ) & ( a_M8.state =
s_st_2_2 ) ) ) ) & ! ( ( ( a_M8.state = s_st_0_0 ) & ( a_M9.state = s_st_2_2 ) ) )
) & ! ( ( ( a_M9.state = s_st_0_0 ) & ( a_M10.state = s_st_2_2 ) ) ) )
TRANS
  next_local_step_ad_M2
  |  synchronized_steps
  |  next_local_step_ad_M3
  |  next_local_step_ad_M4
  |  next_local_step_ad_M5
  |  next_local_step_ad_M6
  |  next_local_step_ad_M7
  |  next_local_step_ad_M8
  |  next_local_step_ad_M9
  |  next_local_step_ad_M10
DEFINE
  a_M2_getCurrentIndex := (   a_M2.state = s_st_0_0 ? 0 :  (   a_M2.state =
s_st_0_1 ? 1 :  (   a_M2.state = s_st_1_1 ? 2 :  (   a_M2.state = s_st_2_1 ? 3 :
(   a_M2.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M2 automata local steps section ==================
  local_step_ad_M2 := (
    -- connection s_st_0_0 - s_st_0_1
    ( ( a_M2.state = s_st_0_0 & e_r1_2 ) & ( next(a_M2.state) = s_st_0_1 ) )
    |
    -- connection s_st_0_1 - s_st_1_1
    ( ( a_M2.state = s_st_0_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_1_1 ) )
    |
    -- connection s_st_1_1 - s_st_2_1
    ( ( a_M2.state = s_st_1_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M2.state = s_st_2_1 & e_r1_2 ) & ( next(a_M2.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M2 := ( local_step_ad_M2
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M3_getCurrentIndex := (   a_M3.state = s_st_0_0 ? 0 :  (   a_M3.state =
s_st_0_1 ? 1 :  (   a_M3.state = s_st_1_1 ? 2 :  (   a_M3.state = s_st_2_1 ? 3 :
(   a_M3.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M3 automata local steps section ==================
  local_step_ad_M3 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M3.state = s_st_2_1 & e_r2_3_b ) & ( next(a_M3.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M3 := ( local_step_ad_M3
    & a_M2.unchanged
```

```
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M4_getCurrentIndex := (   a_M4.state = s_st_0_0 ? 0 :  (   a_M4.state =
s_st_0_1 ? 1 :  (   a_M4.state = s_st_1_1 ? 2 :  (   a_M4.state = s_st_2_1 ? 3 :
(   a_M4.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M4 automata local steps section ==================
  local_step_ad_M4 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M4.state = s_st_2_1 & e_r3_4_b ) & ( next(a_M4.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M4 := ( local_step_ad_M4
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M5_getCurrentIndex := (   a_M5.state = s_st_0_0 ? 0 :  (   a_M5.state =
s_st_0_1 ? 1 :  (   a_M5.state = s_st_1_1 ? 2 :  (   a_M5.state = s_st_2_1 ? 3 :
(   a_M5.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M5 automata local steps section ==================
  local_step_ad_M5 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M5.state = s_st_2_1 & e_r4_5_b ) & ( next(a_M5.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M5 := ( local_step_ad_M5
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M6_getCurrentIndex := (   a_M6.state = s_st_0_0 ? 0 :  (   a_M6.state =
s_st_0_1 ? 1 :  (   a_M6.state = s_st_1_1 ? 2 :  (   a_M6.state = s_st_2_1 ? 3 :
(   a_M6.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M6 automata local steps section ==================
  local_step_ad_M6 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M6.state = s_st_2_1 & e_r5_6_b ) & ( next(a_M6.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M6 := ( local_step_ad_M6
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
```

```
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M7_getCurrentIndex := (   a_M7.state = s_st_0_0 ? 0 :  (   a_M7.state =
s_st_0_1 ? 1 :  (   a_M7.state = s_st_1_1 ? 2 :  (   a_M7.state = s_st_2_1 ? 3 :
(   a_M7.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M7 automata local steps section ==================
  local_step_ad_M7 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M7.state = s_st_2_1 & e_r6_7_b ) & ( next(a_M7.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M7 := ( local_step_ad_M7
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M8_getCurrentIndex := (   a_M8.state = s_st_0_0 ? 0 :  (   a_M8.state =
s_st_0_1 ? 1 :  (   a_M8.state = s_st_1_1 ? 2 :  (   a_M8.state = s_st_2_1 ? 3 :
(   a_M8.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M8 automata local steps section ==================
  local_step_ad_M8 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M8.state = s_st_2_1 & e_r7_8_b ) & ( next(a_M8.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M8 := ( local_step_ad_M8
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
  );
  a_M9_getCurrentIndex := (   a_M9.state = s_st_0_0 ? 0 :  (   a_M9.state =
s_st_0_1 ? 1 :  (   a_M9.state = s_st_1_1 ? 2 :  (   a_M9.state = s_st_2_1 ? 3 :
(   a_M9.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M9 automata local steps section ==================
  local_step_ad_M9 := (
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M9.state = s_st_2_1 & e_r8_9_b ) & ( next(a_M9.state) = s_st_2_2 ) )
  );
  next_local_step_ad_M9 := ( local_step_ad_M9
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M10.unchanged
  );
```

```
  a_M10_getCurrentIndex := (   a_M10.state = s_st_0_0 ? 0 :  (   a_M10.state =
s_st_0_1 ? 1 :  (   a_M10.state = s_st_1_1 ? 2 :  (   a_M10.state = s_st_2_1 ? 3 :
(   a_M10.state = s_st_2_2 ? 4 :  -1  )  )  )  )   );
  -- ================== ad_M10 automata local steps section ==================
  local_step_ad_M10 := (
    -- connection s_st_0_1 - s_st_0_0
    ( ( a_M10.state = s_st_0_1 & e_r10_x ) & ( next(a_M10.state) = s_st_0_0 ) )
    |
    -- connection s_st_1_1 - s_st_0_1
    ( ( a_M10.state = s_st_1_1 & e_r10_x ) & ( next(a_M10.state) = s_st_0_1 ) )
    |
    -- connection s_st_2_1 - s_st_2_2
    ( ( a_M10.state = s_st_2_1 & e_r9_10_b ) & ( next(a_M10.state) = s_st_2_2 ) )
    |
    -- connection s_st_2_1 - s_st_1_1
    ( ( a_M10.state = s_st_2_1 & e_r10_x ) & ( next(a_M10.state) = s_st_1_1 ) )
  );
  next_local_step_ad_M10 := ( local_step_ad_M10
    & a_M2.unchanged
    & a_M3.unchanged
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_r2_3
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
        )
        & a_M4.unchanged
        & a_M5.unchanged
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
```

```
    e_r2_3_u
    &
    (
    -- connection s_st_2_2 - s_st_2_1
     ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_2_1 )
    )
    & a_M4.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
)
|
(
    e_r3_4
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
    -- connection s_st_1_1 - s_st_0_1
    | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
    -- connection s_st_2_1 - s_st_1_1
    | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
    )
    &
    (
    -- connection s_st_0_0 - s_st_0_1
     ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
    -- connection s_st_0_1 - s_st_1_1
    | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
    -- connection s_st_1_1 - s_st_2_1
    | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
    )
    & a_M2.unchanged
    & a_M5.unchanged
    & a_M6.unchanged
    & a_M7.unchanged
    & a_M8.unchanged
    & a_M9.unchanged
    & a_M10.unchanged
)
|
(
    e_r3_4_u
    &
    (
    -- connection s_st_0_1 - s_st_0_0
     ( a_M2.state = s_st_0_1 & next(a_M2.state) = s_st_0_0 )
```

```
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & next(a_M2.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & next(a_M2.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & next(a_M2.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_2_1 )
        )
        & a_M5.unchanged
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r4_5
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & next(a_M4.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
```

```
        e_r4_5_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3 > 0 )  & next(a_M2.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M3.state = s_st_0_1 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & next(a_M3.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & next(a_M3.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & next(a_M3.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M4.state = s_st_2_2 & next(a_M4.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_2_1 )
        )
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
```

```
        & a_M10.unchanged
    )
    |
    (
        e_r5_6
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & next(a_M5.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M6.state = s_st_0_0 & next(a_M6.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M6.state = s_st_0_1 & next(a_M6.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M6.state = s_st_1_1 & next(a_M6.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M4.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r5_6_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3_4 > 0 )  & next(a_M2.state) =
s_st_2_1 )
```

```
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4 > 0 )  & next(a_M3.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M4.state = s_st_0_1 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & next(a_M4.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & next(a_M4.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M4.state = s_st_2_2 & next(a_M4.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M5.state = s_st_2_2 & next(a_M5.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M6.state = s_st_0_0 & next(a_M6.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M6.state = s_st_0_1 & next(a_M6.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M6.state = s_st_1_1 & next(a_M6.state) = s_st_2_1 )
        )
        & a_M7.unchanged
```

```
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r6_7
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M6.state = s_st_0_1 & next(a_M6.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M6.state = s_st_1_1 & next(a_M6.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M6.state = s_st_2_1 & next(a_M6.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M7.state = s_st_0_0 & next(a_M7.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M7.state = s_st_0_1 & next(a_M7.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M7.state = s_st_1_1 & next(a_M7.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M4.unchanged
        & a_M5.unchanged
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r6_7_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4_5 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4_5 > 0 )  & next(a_M2.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4_5 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4_5 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4_5 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4_5 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M2.state = s_st_2_2 & (i_backM3_4_5 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4_5 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4_5 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4_5 > 0 )  & next(a_M3.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4_5 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4_5 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4_5 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4_5 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4_5 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4_5 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M4.state = s_st_0_1 & (i_nbackM5 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M4.state = s_st_0_1 & (i_backM5 > 0 )  & next(a_M4.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M4.state = s_st_1_1 & (i_nbackM5 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & (i_backM5 > 0 )  & next(a_M4.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M4.state = s_st_2_1 & (i_nbackM5 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & (i_backM5 > 0 )  & next(a_M4.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M4.state = s_st_2_2 & (i_backM5 > 0 )  & next(a_M4.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M4.state = s_st_2_2 & (i_nbackM5 > 0 )  & next(a_M4.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M5.state = s_st_0_1 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & next(a_M5.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & next(a_M5.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M5.state = s_st_2_2 & next(a_M5.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M6.state = s_st_2_2 & next(a_M6.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M7.state = s_st_0_0 & next(a_M7.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M7.state = s_st_0_1 & next(a_M7.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M7.state = s_st_1_1 & next(a_M7.state) = s_st_2_1 )
        )
        & a_M8.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r7_8
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M7.state = s_st_0_1 & next(a_M7.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M7.state = s_st_1_1 & next(a_M7.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M7.state = s_st_2_1 & next(a_M7.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M8.state = s_st_0_0 & next(a_M8.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M8.state = s_st_0_1 & next(a_M8.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M8.state = s_st_1_1 & next(a_M8.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M4.unchanged
```

```
        & a_M5.unchanged
        & a_M6.unchanged
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r7_8_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4_5_6 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4_5_6 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4_5_6 > 0 )  & next(a_M3.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4_5_6 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4_5_6 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4_5_6 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4_5_6 > 0 )  & next(a_M3.state) =
s_st_1_1 )
```

```
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4_5_6 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4_5_6 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M4.state = s_st_0_1 & (i_nbackM5_6 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M4.state = s_st_0_1 & (i_backM5_6 > 0 )  & next(a_M4.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M4.state = s_st_1_1 & (i_nbackM5_6 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & (i_backM5_6 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M4.state = s_st_2_1 & (i_nbackM5_6 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & (i_backM5_6 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M4.state = s_st_2_2 & (i_backM5_6 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M4.state = s_st_2_2 & (i_nbackM5_6 > 0 )  & next(a_M4.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M5.state = s_st_0_1 & (i_nbackM6 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M5.state = s_st_0_1 & (i_backM6 > 0 )  & next(a_M5.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M5.state = s_st_1_1 & (i_nbackM6 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & (i_backM6 > 0 )  & next(a_M5.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M5.state = s_st_2_1 & (i_nbackM6 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & (i_backM6 > 0 )  & next(a_M5.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M5.state = s_st_2_2 & (i_backM6 > 0 )  & next(a_M5.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M5.state = s_st_2_2 & (i_nbackM6 > 0 )  & next(a_M5.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M6.state = s_st_0_1 & next(a_M6.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M6.state = s_st_1_1 & next(a_M6.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M6.state = s_st_2_1 & next(a_M6.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M6.state = s_st_2_2 & next(a_M6.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M7.state = s_st_2_2 & next(a_M7.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M8.state = s_st_0_0 & next(a_M8.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M8.state = s_st_0_1 & next(a_M8.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M8.state = s_st_1_1 & next(a_M8.state) = s_st_2_1 )
        )
        & a_M9.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r8_9
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M8.state = s_st_0_1 & next(a_M8.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M8.state = s_st_1_1 & next(a_M8.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M8.state = s_st_2_1 & next(a_M8.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M9.state = s_st_0_0 & next(a_M9.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M9.state = s_st_0_1 & next(a_M9.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M9.state = s_st_1_1 & next(a_M9.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M4.unchanged
        & a_M5.unchanged
```

```
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M10.unchanged
    )
    |
    (
        e_r8_9_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4_5_6_7 > 0 )  & next(a_M2.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M3.state = s_st_2_2 & (i_backM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4_5_6_7 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M4.state = s_st_0_1 & (i_nbackM5_6_7 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M4.state = s_st_0_1 & (i_backM5_6_7 > 0 )  & next(a_M4.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M4.state = s_st_1_1 & (i_nbackM5_6_7 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & (i_backM5_6_7 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M4.state = s_st_2_1 & (i_nbackM5_6_7 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & (i_backM5_6_7 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M4.state = s_st_2_2 & (i_backM5_6_7 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M4.state = s_st_2_2 & (i_nbackM5_6_7 > 0 )  & next(a_M4.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M5.state = s_st_0_1 & (i_nbackM6_7 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M5.state = s_st_0_1 & (i_backM6_7 > 0 )  & next(a_M5.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M5.state = s_st_1_1 & (i_nbackM6_7 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & (i_backM6_7 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M5.state = s_st_2_1 & (i_nbackM6_7 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & (i_backM6_7 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M5.state = s_st_2_2 & (i_backM6_7 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M5.state = s_st_2_2 & (i_nbackM6_7 > 0 )  & next(a_M5.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M6.state = s_st_0_0 & next(a_M6.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M6.state = s_st_0_1 & (i_nbackM7 > 0 )  & next(a_M6.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M6.state = s_st_0_1 & (i_backM7 > 0 )  & next(a_M6.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M6.state = s_st_1_1 & (i_nbackM7 > 0 )  & next(a_M6.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M6.state = s_st_1_1 & (i_backM7 > 0 )  & next(a_M6.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M6.state = s_st_2_1 & (i_nbackM7 > 0 )  & next(a_M6.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M6.state = s_st_2_1 & (i_backM7 > 0 )  & next(a_M6.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M6.state = s_st_2_2 & (i_backM7 > 0 )  & next(a_M6.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M6.state = s_st_2_2 & (i_nbackM7 > 0 )  & next(a_M6.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M7.state = s_st_0_1 & next(a_M7.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M7.state = s_st_1_1 & next(a_M7.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M7.state = s_st_2_1 & next(a_M7.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M7.state = s_st_2_2 & next(a_M7.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M8.state = s_st_2_2 & next(a_M8.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M9.state = s_st_0_0 & next(a_M9.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M9.state = s_st_0_1 & next(a_M9.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M9.state = s_st_1_1 & next(a_M9.state) = s_st_2_1 )
```

```
        )
        & a_M10.unchanged
    )
    |
    (
        e_r9_10
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M9.state = s_st_0_1 & next(a_M9.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M9.state = s_st_1_1 & next(a_M9.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M9.state = s_st_2_1 & next(a_M9.state) = s_st_1_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M10.state = s_st_0_0 & next(a_M10.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M10.state = s_st_0_1 & next(a_M10.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M10.state = s_st_1_1 & next(a_M10.state) = s_st_2_1 )
        )
        & a_M2.unchanged
        & a_M3.unchanged
        & a_M4.unchanged
        & a_M5.unchanged
        & a_M6.unchanged
        & a_M7.unchanged
        & a_M8.unchanged
    )
    |
    (
        e_r9_10_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M2.state = s_st_2_2 & (i_backM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4_5_6_7_8 > 0 )  & next(a_M2.state)
= s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4_5_6_7_8 > 0 )  & next(a_M3.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M4.state = s_st_0_1 & (i_nbackM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M4.state = s_st_0_1 & (i_backM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M4.state = s_st_1_1 & (i_nbackM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & (i_backM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M4.state = s_st_2_1 & (i_nbackM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & (i_backM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M4.state = s_st_2_2 & (i_backM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M4.state = s_st_2_2 & (i_nbackM5_6_7_8 > 0 )  & next(a_M4.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M5.state = s_st_0_1 & (i_nbackM6_7_8 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M5.state = s_st_0_1 & (i_backM6_7_8 > 0 )  & next(a_M5.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M5.state = s_st_1_1 & (i_nbackM6_7_8 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & (i_backM6_7_8 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M5.state = s_st_2_1 & (i_nbackM6_7_8 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & (i_backM6_7_8 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M5.state = s_st_2_2 & (i_backM6_7_8 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M5.state = s_st_2_2 & (i_nbackM6_7_8 > 0 )  & next(a_M5.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M6.state = s_st_0_0 & next(a_M6.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M6.state = s_st_0_1 & (i_nbackM7_8 > 0 )  & next(a_M6.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M6.state = s_st_0_1 & (i_backM7_8 > 0 )  & next(a_M6.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M6.state = s_st_1_1 & (i_nbackM7_8 > 0 )  & next(a_M6.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M6.state = s_st_1_1 & (i_backM7_8 > 0 )  & next(a_M6.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M6.state = s_st_2_1 & (i_nbackM7_8 > 0 )  & next(a_M6.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M6.state = s_st_2_1 & (i_backM7_8 > 0 )  & next(a_M6.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
```

```
        | ( a_M6.state = s_st_2_2 & (i_backM7_8 > 0 )  & next(a_M6.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M6.state = s_st_2_2 & (i_nbackM7_8 > 0 )  & next(a_M6.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M7.state = s_st_0_0 & next(a_M7.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M7.state = s_st_0_1 & (i_nbackM8 > 0 )  & next(a_M7.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M7.state = s_st_0_1 & (i_backM8 > 0 )  & next(a_M7.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M7.state = s_st_1_1 & (i_nbackM8 > 0 )  & next(a_M7.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M7.state = s_st_1_1 & (i_backM8 > 0 )  & next(a_M7.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M7.state = s_st_2_1 & (i_nbackM8 > 0 )  & next(a_M7.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M7.state = s_st_2_1 & (i_backM8 > 0 )  & next(a_M7.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M7.state = s_st_2_2 & (i_backM8 > 0 )  & next(a_M7.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M7.state = s_st_2_2 & (i_nbackM8 > 0 )  & next(a_M7.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M8.state = s_st_0_1 & next(a_M8.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M8.state = s_st_1_1 & next(a_M8.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M8.state = s_st_2_1 & next(a_M8.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M8.state = s_st_2_2 & next(a_M8.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M9.state = s_st_2_2 & next(a_M9.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_1
         ( a_M10.state = s_st_0_0 & next(a_M10.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_1_1
        | ( a_M10.state = s_st_0_1 & next(a_M10.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_2_1
        | ( a_M10.state = s_st_1_1 & next(a_M10.state) = s_st_2_1 )
```

```
        )
    )
    |
    (
        e_r10_x_u
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M2.state = s_st_0_0 & next(a_M2.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M2.state = s_st_0_1 & (i_nbackM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M2.state = s_st_0_1 & (i_backM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M2.state = s_st_1_1 & (i_nbackM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M2.state = s_st_1_1 & (i_backM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M2.state = s_st_2_1 & (i_nbackM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M2.state = s_st_2_1 & (i_backM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M2.state = s_st_2_2 & (i_backM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M2.state = s_st_2_2 & (i_nbackM3_4_5_6_7_8_9 > 0 )  &
next(a_M2.state) = s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M3.state = s_st_0_0 & next(a_M3.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M3.state = s_st_0_1 & (i_nbackM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M3.state = s_st_0_1 & (i_backM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M3.state = s_st_1_1 & (i_nbackM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M3.state = s_st_1_1 & (i_backM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M3.state = s_st_2_1 & (i_nbackM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M3.state = s_st_2_1 & (i_backM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M3.state = s_st_2_2 & (i_backM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_2_1 )
```

```
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M3.state = s_st_2_2 & (i_nbackM4_5_6_7_8_9 > 0 )  & next(a_M3.state)
= s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M4.state = s_st_0_0 & next(a_M4.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M4.state = s_st_0_1 & (i_nbackM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M4.state = s_st_0_1 & (i_backM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M4.state = s_st_1_1 & (i_nbackM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M4.state = s_st_1_1 & (i_backM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M4.state = s_st_2_1 & (i_nbackM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M4.state = s_st_2_1 & (i_backM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M4.state = s_st_2_2 & (i_backM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M4.state = s_st_2_2 & (i_nbackM5_6_7_8_9 > 0 )  & next(a_M4.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M5.state = s_st_0_0 & next(a_M5.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M5.state = s_st_0_1 & (i_nbackM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M5.state = s_st_0_1 & (i_backM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M5.state = s_st_1_1 & (i_nbackM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M5.state = s_st_1_1 & (i_backM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M5.state = s_st_2_1 & (i_nbackM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M5.state = s_st_2_1 & (i_backM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M5.state = s_st_2_2 & (i_backM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
```

```
        | ( a_M5.state = s_st_2_2 & (i_nbackM6_7_8_9 > 0 )  & next(a_M5.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M6.state = s_st_0_0 & next(a_M6.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M6.state = s_st_0_1 & (i_nbackM7_8_9 > 0 )  & next(a_M6.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M6.state = s_st_0_1 & (i_backM7_8_9 > 0 )  & next(a_M6.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M6.state = s_st_1_1 & (i_nbackM7_8_9 > 0 )  & next(a_M6.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M6.state = s_st_1_1 & (i_backM7_8_9 > 0 )  & next(a_M6.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M6.state = s_st_2_1 & (i_nbackM7_8_9 > 0 )  & next(a_M6.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M6.state = s_st_2_1 & (i_backM7_8_9 > 0 )  & next(a_M6.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M6.state = s_st_2_2 & (i_backM7_8_9 > 0 )  & next(a_M6.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M6.state = s_st_2_2 & (i_nbackM7_8_9 > 0 )  & next(a_M6.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M7.state = s_st_0_0 & next(a_M7.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M7.state = s_st_0_1 & (i_nbackM8_9 > 0 )  & next(a_M7.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M7.state = s_st_0_1 & (i_backM8_9 > 0 )  & next(a_M7.state) =
s_st_0_0 )
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M7.state = s_st_1_1 & (i_nbackM8_9 > 0 )  & next(a_M7.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M7.state = s_st_1_1 & (i_backM8_9 > 0 )  & next(a_M7.state) =
s_st_0_1 )
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M7.state = s_st_2_1 & (i_nbackM8_9 > 0 )  & next(a_M7.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M7.state = s_st_2_1 & (i_backM8_9 > 0 )  & next(a_M7.state) =
s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M7.state = s_st_2_2 & (i_backM8_9 > 0 )  & next(a_M7.state) =
s_st_2_1 )
        -- connection s_st_2_2 - s_st_2_2
```

```
        | ( a_M7.state = s_st_2_2 & (i_nbackM8_9 > 0 )  & next(a_M7.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_0 - s_st_0_0
         ( a_M8.state = s_st_0_0 & next(a_M8.state) = s_st_0_0 )
        -- connection s_st_0_1 - s_st_0_1
        | ( a_M8.state = s_st_0_1 & (i_nbackM9 > 0 )  & next(a_M8.state) =
s_st_0_1 )
        -- connection s_st_0_1 - s_st_0_0
        | ( a_M8.state = s_st_0_1 & (i_backM9 > 0 )  & next(a_M8.state) = s_st_0_0
)
        -- connection s_st_1_1 - s_st_1_1
        | ( a_M8.state = s_st_1_1 & (i_nbackM9 > 0 )  & next(a_M8.state) =
s_st_1_1 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M8.state = s_st_1_1 & (i_backM9 > 0 )  & next(a_M8.state) = s_st_0_1
)
        -- connection s_st_2_1 - s_st_2_1
        | ( a_M8.state = s_st_2_1 & (i_nbackM9 > 0 )  & next(a_M8.state) =
s_st_2_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M8.state = s_st_2_1 & (i_backM9 > 0 )  & next(a_M8.state) = s_st_1_1
)
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M8.state = s_st_2_2 & (i_backM9 > 0 )  & next(a_M8.state) = s_st_2_1
)
        -- connection s_st_2_2 - s_st_2_2
        | ( a_M8.state = s_st_2_2 & (i_nbackM9 > 0 )  & next(a_M8.state) =
s_st_2_2 )
        )
        &
        (
        -- connection s_st_0_1 - s_st_0_0
         ( a_M9.state = s_st_0_1 & next(a_M9.state) = s_st_0_0 )
        -- connection s_st_1_1 - s_st_0_1
        | ( a_M9.state = s_st_1_1 & next(a_M9.state) = s_st_0_1 )
        -- connection s_st_2_1 - s_st_1_1
        | ( a_M9.state = s_st_2_1 & next(a_M9.state) = s_st_1_1 )
        -- connection s_st_2_2 - s_st_2_1
        | ( a_M9.state = s_st_2_2 & next(a_M9.state) = s_st_2_1 )
        )
        &
        (
        -- connection s_st_2_2 - s_st_2_1
         ( a_M10.state = s_st_2_2 & next(a_M10.state) = s_st_2_1 )
        )
    )
  );
  -- ================== identifiers section ==================
  i_mu1 := ( 1 );

  i_mu2 := ( 1 );

  i_mu3 := ( 1 );

  i_mu4 := ( 1 );
```

```
  i_mu5 := ( 1 );

  i_mu6 := ( 1 );

  i_mu7 := ( 1 );

  i_mu8 := ( 1 );

  i_mu9 := ( 1 );

  i_mu10 := ( 1 );

  i_f2_3_b := ( ( ( ( a_M2.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu2 );

  i_f3_4_b := ( ( ( ( a_M3.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu3 );

  i_f4_5_b := ( ( ( ( a_M4.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu4 );

  i_f5_6_b := ( ( ( ( a_M5.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu5 );

  i_f6_7_b := ( ( ( ( a_M6.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu6 );

  i_f7_8_b := ( ( ( ( a_M7.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu7 );

  i_f8_9_b := ( ( ( ( a_M8.state != s_st_0_0 ) ) ? 1 : 0 ) * i_mu8 );

  i_f9_10_b := ( ( ( a_M9.state != s_st_0_0 ) ? 1 : 0 ) );

  i_backM3 := ( ( ( a_M3.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM3_4 := ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM3_4_5 := ( ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state = s_st_2_2 ) )
& ( a_M5.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM3_4_5_6 := ( ( ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state = s_st_2_2
) ) & ( a_M5.state = s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM3_4_5_6_7 := ( ( ( ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state =
s_st_2_2 ) ) & ( a_M5.state = s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) & (
a_M7.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM3_4_5_6_7_8 := ( ( ( ( ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state =
s_st_2_2 ) ) & ( a_M5.state = s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) & (
a_M7.state = s_st_2_2 ) ) & ( a_M8.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM3_4_5_6_7_8_9 := ( ( ( ( ( ( ( ( ( a_M3.state = s_st_2_2 ) & ( a_M4.state
= s_st_2_2 ) ) & ( a_M5.state = s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) & (
a_M7.state = s_st_2_2 ) ) & ( a_M8.state = s_st_2_2 ) ) & ( a_M9.state = s_st_2_2
) ) ? 1 : 0 ) );

  i_backM4 := ( ( ( a_M4.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM4_5 := ( ( ( ( a_M4.state = s_st_2_2 ) & ( a_M5.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM4_5_6 := ( ( ( ( ( a_M4.state = s_st_2_2 ) & ( a_M5.state = s_st_2_2 ) )
& ( a_M6.state = s_st_2_2 ) ) ? 1 : 0 ) );
```

```
  i_backM4_5_6_7 := ( ( ( ( ( a_M4.state = s_st_2_2 ) & ( a_M5.state = s_st_2_2
) ) & ( a_M6.state = s_st_2_2 ) ) & ( a_M7.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM4_5_6_7_8 := ( ( ( ( ( ( a_M4.state = s_st_2_2 ) & ( a_M5.state =
s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) & ( a_M7.state = s_st_2_2 ) ) & (
a_M8.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM4_5_6_7_8_9 := ( ( ( ( ( ( ( a_M4.state = s_st_2_2 ) & ( a_M5.state =
s_st_2_2 ) ) & ( a_M6.state = s_st_2_2 ) ) & ( a_M7.state = s_st_2_2 ) ) & (
a_M8.state = s_st_2_2 ) ) & ( a_M9.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM5 := ( ( ( a_M5.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM5_6 := ( ( ( ( a_M5.state = s_st_2_2 ) & ( a_M6.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM5_6_7 := ( ( ( ( ( a_M5.state = s_st_2_2 ) & ( a_M6.state = s_st_2_2 ) )
& ( a_M7.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM5_6_7_8 := ( ( ( ( ( ( a_M5.state = s_st_2_2 ) & ( a_M6.state = s_st_2_2
) ) & ( a_M7.state = s_st_2_2 ) ) & ( a_M8.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM5_6_7_8_9 := ( ( ( ( ( ( ( a_M5.state = s_st_2_2 ) & ( a_M6.state =
s_st_2_2 ) ) & ( a_M7.state = s_st_2_2 ) ) & ( a_M8.state = s_st_2_2 ) ) & (
a_M9.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM6 := ( ( ( a_M6.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM6_7 := ( ( ( ( a_M6.state = s_st_2_2 ) & ( a_M7.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM6_7_8 := ( ( ( ( ( a_M6.state = s_st_2_2 ) & ( a_M7.state = s_st_2_2 ) )
& ( a_M8.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM6_7_8_9 := ( ( ( ( ( ( a_M6.state = s_st_2_2 ) & ( a_M7.state = s_st_2_2
) ) & ( a_M8.state = s_st_2_2 ) ) & ( a_M9.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM7 := ( ( ( a_M7.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM7_8 := ( ( ( ( a_M7.state = s_st_2_2 ) & ( a_M8.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM7_8_9 := ( ( ( ( ( a_M7.state = s_st_2_2 ) & ( a_M8.state = s_st_2_2 ) )
& ( a_M9.state = s_st_2_2 ) ) ? 1 : 0 ) );

  i_backM8 := ( ( ( a_M8.state = s_st_2_2 ) ? 1 : 0 ) );

  i_backM8_9 := ( ( ( ( a_M8.state = s_st_2_2 ) & ( a_M9.state = s_st_2_2 ) ) ? 1
: 0 ) );

  i_backM9 := ( ( ( a_M9.state = s_st_2_2 ) ? 1 : 0 ) );

  i_nbackM3 := ( ( ! ( ( i_backM3 ) > 0 ) ? 1 : 0 ) );

  i_nbackM3_4 := ( ( ! ( ( i_backM3_4 ) > 0 ) ? 1 : 0 ) );

  i_nbackM3_4_5 := ( ( ! ( ( i_backM3_4_5 ) > 0 ) ? 1 : 0 ) );
```

```
i_nbackM3_4_5_6 := ( ( ! ( ( i_backM3_4_5_6 ) > 0 ) ? 1 : 0 ) );

i_nbackM3_4_5_6_7 := ( ( ! ( ( i_backM3_4_5_6_7 ) > 0 ) ? 1 : 0 ) );

i_nbackM3_4_5_6_7_8 := ( ( ! ( ( i_backM3_4_5_6_7_8 ) > 0 ) ? 1 : 0 ) );

i_nbackM3_4_5_6_7_8_9 := ( ( ! ( ( i_backM3_4_5_6_7_8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM4 := ( ( ! ( ( i_backM4 ) > 0 ) ? 1 : 0 ) );

i_nbackM4_5 := ( ( ! ( ( i_backM4_5 ) > 0 ) ? 1 : 0 ) );

i_nbackM4_5_6 := ( ( ! ( ( i_backM4_5_6 ) > 0 ) ? 1 : 0 ) );

i_nbackM4_5_6_7 := ( ( ! ( ( i_backM4_5_6_7 ) > 0 ) ? 1 : 0 ) );

i_nbackM4_5_6_7_8 := ( ( ! ( ( i_backM4_5_6_7_8 ) > 0 ) ? 1 : 0 ) );

i_nbackM4_5_6_7_8_9 := ( ( ! ( ( i_backM4_5_6_7_8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM5 := ( ( ! ( ( i_backM5 ) > 0 ) ? 1 : 0 ) );

i_nbackM5_6 := ( ( ! ( ( i_backM5_6 ) > 0 ) ? 1 : 0 ) );

i_nbackM5_6_7 := ( ( ! ( ( i_backM5_6_7 ) > 0 ) ? 1 : 0 ) );

i_nbackM5_6_7_8 := ( ( ! ( ( i_backM5_6_7_8 ) > 0 ) ? 1 : 0 ) );

i_nbackM5_6_7_8_9 := ( ( ! ( ( i_backM5_6_7_8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM6 := ( ( ! ( ( i_backM6 ) > 0 ) ? 1 : 0 ) );

i_nbackM6_7 := ( ( ! ( ( i_backM6_7 ) > 0 ) ? 1 : 0 ) );

i_nbackM6_7_8 := ( ( ! ( ( i_backM6_7_8 ) > 0 ) ? 1 : 0 ) );

i_nbackM6_7_8_9 := ( ( ! ( ( i_backM6_7_8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM7 := ( ( ! ( ( i_backM7 ) > 0 ) ? 1 : 0 ) );

i_nbackM7_8 := ( ( ! ( ( i_backM7_8 ) > 0 ) ? 1 : 0 ) );

i_nbackM7_8_9 := ( ( ! ( ( i_backM7_8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM8 := ( ( ! ( ( i_backM8 ) > 0 ) ? 1 : 0 ) );

i_nbackM8_9 := ( ( ! ( ( i_backM8_9 ) > 0 ) ? 1 : 0 ) );

i_nbackM9 := ( ( ! ( ( i_backM9 ) > 0 ) ? 1 : 0 ) );

-- ================== nb operators section ==================
-- ================== events section ==================
e_r1_2 := ( i_mu1 > 0 );

e_r2_3 := ( i_mu2 > 0 );

e_r2_3_b := ( i_f2_3_b > 0 );

e_r2_3_u := ( i_mu2 > 0 );
```

```
  e_r3_4 := ( i_mu3 > 0 );
  e_r3_4_b := ( i_f3_4_b > 0 );
  e_r3_4_u := ( i_mu3 > 0 );
  e_r4_5 := ( i_mu4 > 0 );
  e_r4_5_b := ( i_f4_5_b > 0 );
  e_r4_5_u := ( i_mu4 > 0 );
  e_r5_6 := ( i_mu5 > 0 );
  e_r5_6_b := ( i_f5_6_b > 0 );
  e_r5_6_u := ( i_mu5 > 0 );
  e_r6_7 := ( i_mu6 > 0 );
  e_r6_7_b := ( i_f6_7_b > 0 );
  e_r6_7_u := ( i_mu6 > 0 );
  e_r7_8 := ( i_mu7 > 0 );
  e_r7_8_b := ( i_f7_8_b > 0 );
  e_r7_8_u := ( i_mu7 > 0 );
  e_r8_9 := ( i_mu8 > 0 );
  e_r8_9_b := ( i_f8_9_b > 0 );
  e_r8_9_u := ( i_mu8 > 0 );
  e_r9_10 := ( i_mu9 > 0 );
  e_r9_10_b := ( i_f9_10_b > 0 );
  e_r9_10_u := ( i_mu9 > 0 );
  e_r10_x := ( i_mu10 > 0 );
  e_r10_x_u := ( i_mu10 > 0 );
```

APÊNDICE AH – ARQUIVO TEXTUAL DO MODELO FAS05C.SAN

```
identifiers

  // Acquisition rate
  lambda = 2;
  // Release rate
  mu = 1;

events

  loc t1 (lambda);
  syn t2 (lambda);
  syn t3 (lambda);
  syn t4 (lambda);
  syn t5 (lambda);
  loc r1 (mu);
  loc r2 (mu);
  loc r3 (mu);
  loc r4 (mu);
  loc r5 (mu);

reachability = 1;

network FAS5c (continuous)

  aut Server1 stt idle to (busy) t1
              stt busy to (idle) r1
                       to (busy) t2 t3 t4 t5

  aut Server2 stt idle to (busy) t2
              stt busy to (idle) r2
                       to (busy) t3 t4 t5

  aut Server3 stt idle to (busy) t3
              stt busy to (idle) r3
                       to (busy) t4 t5

  aut Server4 stt idle to (busy) t4
              stt busy to (idle) r4
                       to (busy) t5

  aut Server5 stt idle to (busy) t5
              stt busy to (idle) r5
```

APÊNDICE AI – ARQUIVO TEXTUAL DO MODELO FAS05C.SMV

```
MODULE ad_Server1()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_Server2()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_Server3()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_Server4()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE ad_Server5()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ================================================================================
MODULE main
VAR
  a_Server1 : ad_Server1();
  a_Server2 : ad_Server2();
  a_Server3 : ad_Server3();
  a_Server4 : ad_Server4();
  a_Server5 : ad_Server5();

INIT
  1 = 1
TRANS
  next_local_step_ad_Server1
  |  synchronized_steps
  |  next_local_step_ad_Server2
  |  next_local_step_ad_Server3
  |  next_local_step_ad_Server4
  |  next_local_step_ad_Server5
DEFINE
  a_Server1_getCurrentIndex := (   a_Server1.state = s_idle ? 0 :  (
a_Server1.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server1 automata local steps section ==================
  local_step_ad_Server1 := (
    -- connection s_idle - s_busy
    ( ( a_Server1.state = s_idle & e_t1 ) & ( next(a_Server1.state) = s_busy ) )
    |
    -- connection s_busy - s_idle
    ( ( a_Server1.state = s_busy & e_r1 ) & ( next(a_Server1.state) = s_idle ) )
  );
  next_local_step_ad_Server1 := ( local_step_ad_Server1
    & a_Server2.unchanged
```

```
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
  );
  a_Server2_getCurrentIndex := (   a_Server2.state = s_idle ? 0 :  (
a_Server2.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server2 automata local steps section ==================
  local_step_ad_Server2 := (
    -- connection s_busy - s_idle
    ( ( a_Server2.state = s_busy & e_r2 ) & ( next(a_Server2.state) = s_idle ) )
  );
  next_local_step_ad_Server2 := ( local_step_ad_Server2
    & a_Server1.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
  );
  a_Server3_getCurrentIndex := (   a_Server3.state = s_idle ? 0 :  (
a_Server3.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server3 automata local steps section ==================
  local_step_ad_Server3 := (
    -- connection s_busy - s_idle
    ( ( a_Server3.state = s_busy & e_r3 ) & ( next(a_Server3.state) = s_idle ) )
  );
  next_local_step_ad_Server3 := ( local_step_ad_Server3
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
  );
  a_Server4_getCurrentIndex := (   a_Server4.state = s_idle ? 0 :  (
a_Server4.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server4 automata local steps section ==================
  local_step_ad_Server4 := (
    -- connection s_busy - s_idle
    ( ( a_Server4.state = s_busy & e_r4 ) & ( next(a_Server4.state) = s_idle ) )
  );
  next_local_step_ad_Server4 := ( local_step_ad_Server4
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server5.unchanged
  );
  a_Server5_getCurrentIndex := (   a_Server5.state = s_idle ? 0 :  (
a_Server5.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server5 automata local steps section ==================
  local_step_ad_Server5 := (
    -- connection s_busy - s_idle
    ( ( a_Server5.state = s_busy & e_r5 ) & ( next(a_Server5.state) = s_idle ) )
  );
  next_local_step_ad_Server5 := ( local_step_ad_Server5
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
```

```
        e_t2
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server2.state = s_idle & next(a_Server2.state) = s_busy )
        )
        & a_Server3.unchanged
        & a_Server4.unchanged
        & a_Server5.unchanged
    )
    |
    (
        e_t3
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server3.state = s_idle & next(a_Server3.state) = s_busy )
        )
        & a_Server4.unchanged
        & a_Server5.unchanged
    )
    |
    (
        e_t4
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server4.state = s_idle & next(a_Server4.state) = s_busy )
        )
```

```
      & a_Server5.unchanged
  )
  |
  (
      e_t5
      &
      (
      -- connection s_busy - s_busy
       ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
      )
      -- connection s_idle - s_busy
      & ( a_Server5.state = s_idle & next(a_Server5.state) = s_busy )
  )
);
-- ================== identifiers section ==================
i_lambda := ( 2 );

i_mu := ( 1 );

-- ================== nb operators section ==================
-- ================== events section ==================
e_t1 := ( i_lambda > 0 );

e_t2 := ( i_lambda > 0 );

e_t3 := ( i_lambda > 0 );

e_t4 := ( i_lambda > 0 );

e_t5 := ( i_lambda > 0 );

e_r1 := ( i_mu > 0 );

e_r2 := ( i_mu > 0 );

e_r3 := ( i_mu > 0 );

e_r4 := ( i_mu > 0 );

e_r5 := ( i_mu > 0 );
```

APÊNDICE AJ – ARQUIVO TEXTUAL DO MODELO FAS06C.SAN

```
identifiers

  // Acquisition rate
  lambda = 2;
  // Release rate
  mu = 1;

events

  loc t1 (lambda);
  syn t2 (lambda);
  syn t3 (lambda);
  syn t4 (lambda);
  syn t5 (lambda);
  syn t6 (lambda);
  loc r1 (mu);
  loc r2 (mu);
  loc r3 (mu);
  loc r4 (mu);
  loc r5 (mu);
  loc r6 (mu);

reachability = 1;

network FAS6c (continuous)

  aut Server1 stt idle to (busy) t1
              stt busy to (idle) r1
                       to (busy) t2 t3 t4 t5 t6

  aut Server2 stt idle to (busy) t2
              stt busy to (idle) r2
                       to (busy) t3 t4 t5 t6

  aut Server3 stt idle to (busy) t3
              stt busy to (idle) r3
                       to (busy) t4 t5 t6

  aut Server4 stt idle to (busy) t4
              stt busy to (idle) r4
                       to (busy) t5 t6

  aut Server5 stt idle to (busy) t5
              stt busy to (idle) r5
                       to (busy) t6

  aut Server6 stt idle to (busy) t6
              stt busy to (idle) r6
```

APÊNDICE AK – ARQUIVO TEXTUAL DO MODELO FAS06C.SMV

```
MODULE ad_Server1()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server2()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server3()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server4()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server5()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server6()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_Server1 : ad_Server1();
  a_Server2 : ad_Server2();
  a_Server3 : ad_Server3();
  a_Server4 : ad_Server4();
  a_Server5 : ad_Server5();
  a_Server6 : ad_Server6();

INIT
  1 = 1
TRANS
  next_local_step_ad_Server1
  |  synchronized_steps
  |  next_local_step_ad_Server2
  |  next_local_step_ad_Server3
  |  next_local_step_ad_Server4
  |  next_local_step_ad_Server5
  |  next_local_step_ad_Server6
DEFINE
  a_Server1_getCurrentIndex := (   a_Server1.state = s_idle ? 0 :  (
a_Server1.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server1 automata local steps section ==================
  local_step_ad_Server1 := (
    -- connection s_idle - s_busy
```

```
    ( ( a_Server1.state = s_idle & e_t1 ) & ( next(a_Server1.state) = s_busy ) )
    |
    -- connection s_busy - s_idle
    ( ( a_Server1.state = s_busy & e_r1 ) & ( next(a_Server1.state) = s_idle ) )
  );
  next_local_step_ad_Server1 := ( local_step_ad_Server1
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
  );
  a_Server2_getCurrentIndex := (   a_Server2.state = s_idle ? 0 :  (
a_Server2.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server2 automata local steps section ==================
  local_step_ad_Server2 := (
    -- connection s_busy - s_idle
    ( ( a_Server2.state = s_busy & e_r2 ) & ( next(a_Server2.state) = s_idle ) )
  );
  next_local_step_ad_Server2 := ( local_step_ad_Server2
    & a_Server1.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
  );
  a_Server3_getCurrentIndex := (   a_Server3.state = s_idle ? 0 :  (
a_Server3.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server3 automata local steps section ==================
  local_step_ad_Server3 := (
    -- connection s_busy - s_idle
    ( ( a_Server3.state = s_busy & e_r3 ) & ( next(a_Server3.state) = s_idle ) )
  );
  next_local_step_ad_Server3 := ( local_step_ad_Server3
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
  );
  a_Server4_getCurrentIndex := (   a_Server4.state = s_idle ? 0 :  (
a_Server4.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server4 automata local steps section ==================
  local_step_ad_Server4 := (
    -- connection s_busy - s_idle
    ( ( a_Server4.state = s_busy & e_r4 ) & ( next(a_Server4.state) = s_idle ) )
  );
  next_local_step_ad_Server4 := ( local_step_ad_Server4
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
  );
  a_Server5_getCurrentIndex := (   a_Server5.state = s_idle ? 0 :  (
a_Server5.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server5 automata local steps section ==================
  local_step_ad_Server5 := (
    -- connection s_busy - s_idle
```

```
    ( ( a_Server5.state = s_busy & e_r5 ) & ( next(a_Server5.state) = s_idle ) )
  );
  next_local_step_ad_Server5 := ( local_step_ad_Server5
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server6.unchanged
  );
  a_Server6_getCurrentIndex := (   a_Server6.state = s_idle ? 0 :  (
a_Server6.state = s_busy ? 1 :  -1  )   );
  -- ================= ad_Server6 automata local steps section ==================
  local_step_ad_Server6 := (
    -- connection s_busy - s_idle
    ( ( a_Server6.state = s_busy & e_r6 ) & ( next(a_Server6.state) = s_idle ) )
  );
  next_local_step_ad_Server6 := ( local_step_ad_Server6
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
  );
  -- ================= SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_t2
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server2.state = s_idle & next(a_Server2.state) = s_busy )
        )
        & a_Server3.unchanged
        & a_Server4.unchanged
        & a_Server5.unchanged
        & a_Server6.unchanged
    )
    |
    (
        e_t3
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server3.state = s_idle & next(a_Server3.state) = s_busy )
```

```
        )
        & a_Server4.unchanged
        & a_Server5.unchanged
        & a_Server6.unchanged
    )
    |
    (
        e_t4
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server4.state = s_idle & next(a_Server4.state) = s_busy )
        )
        & a_Server5.unchanged
        & a_Server6.unchanged
    )
    |
    (
        e_t5
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server5.state = s_idle & next(a_Server5.state) = s_busy )
        )
```

```
      & a_Server6.unchanged
  )
  |
  (
      e_t6
      &
      (
      -- connection s_busy - s_busy
       ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
      )
      -- connection s_idle - s_busy
      & ( a_Server6.state = s_idle & next(a_Server6.state) = s_busy )
  )
);
-- ================== identifiers section ==================
i_lambda := ( 2 );

i_mu := ( 1 );

-- ================== nb operators section ==================
-- ================== events section ==================
e_t1 := ( i_lambda > 0 );

e_t2 := ( i_lambda > 0 );

e_t3 := ( i_lambda > 0 );

e_t4 := ( i_lambda > 0 );

e_t5 := ( i_lambda > 0 );

e_t6 := ( i_lambda > 0 );

e_r1 := ( i_mu > 0 );

e_r2 := ( i_mu > 0 );

e_r3 := ( i_mu > 0 );
```

```
  e_r4 := ( i_mu > 0 );
  e_r5 := ( i_mu > 0 );
  e_r6 := ( i_mu > 0 );
```

APÊNDICE AL – ARQUIVO TEXTUAL DO MODELO FAS07C.SAN

```
identifiers

  // Acquisition rate
  lambda = 2;
  // Release rate
  mu = 1;

events

  loc t1 (lambda);
  syn t2 (lambda);
  syn t3 (lambda);
  syn t4 (lambda);
  syn t5 (lambda);
  syn t6 (lambda);
  syn t7 (lambda);
  loc r1 (mu);
  loc r2 (mu);
  loc r3 (mu);
  loc r4 (mu);
  loc r5 (mu);
  loc r6 (mu);
  loc r7 (mu);

reachability = 1;

network FAS7c (continuous)

  aut Server1 stt idle to (busy) t1
              stt busy to (idle) r1
                       to (busy) t2 t3 t4 t5 t6 t7

  aut Server2 stt idle to (busy) t2
              stt busy to (idle) r2
                       to (busy) t3 t4 t5 t6 t7

  aut Server3 stt idle to (busy) t3
              stt busy to (idle) r3
                       to (busy) t4 t5 t6 t7

  aut Server4 stt idle to (busy) t4
              stt busy to (idle) r4
                       to (busy) t5 t6 t7

  aut Server5 stt idle to (busy) t5
              stt busy to (idle) r5
                       to (busy) t6 t7

  aut Server6 stt idle to (busy) t6
              stt busy to (idle) r6
                       to (busy) t7

  aut Server7 stt idle to (busy) t7
              stt busy to (idle) r7
```

APÊNDICE AM – ARQUIVO TEXTUAL DO MODELO FAS07C.SMV

```
MODULE ad_Server1()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server2()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server3()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server4()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server5()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server6()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server7()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_Server1 : ad_Server1();
  a_Server2 : ad_Server2();
  a_Server3 : ad_Server3();
  a_Server4 : ad_Server4();
  a_Server5 : ad_Server5();
  a_Server6 : ad_Server6();
  a_Server7 : ad_Server7();

INIT
  1 = 1
TRANS
  next_local_step_ad_Server1
  |  synchronized_steps
  |  next_local_step_ad_Server2
  |  next_local_step_ad_Server3
  |  next_local_step_ad_Server4
  |  next_local_step_ad_Server5
  |  next_local_step_ad_Server6
```

```
  |  next_local_step_ad_Server7
DEFINE
  a_Server1_getCurrentIndex := (   a_Server1.state = s_idle ? 0 :  (
a_Server1.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server1 automata local steps section ==================
  local_step_ad_Server1 := (
    -- connection s_idle - s_busy
    ( ( a_Server1.state = s_idle & e_t1 ) & ( next(a_Server1.state) = s_busy ) )
    |
    -- connection s_busy - s_idle
    ( ( a_Server1.state = s_busy & e_r1 ) & ( next(a_Server1.state) = s_idle ) )
  );
  next_local_step_ad_Server1 := ( local_step_ad_Server1
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
  );
  a_Server2_getCurrentIndex := (   a_Server2.state = s_idle ? 0 :  (
a_Server2.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server2 automata local steps section ==================
  local_step_ad_Server2 := (
    -- connection s_busy - s_idle
    ( ( a_Server2.state = s_busy & e_r2 ) & ( next(a_Server2.state) = s_idle ) )
  );
  next_local_step_ad_Server2 := ( local_step_ad_Server2
    & a_Server1.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
  );
  a_Server3_getCurrentIndex := (   a_Server3.state = s_idle ? 0 :  (
a_Server3.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server3 automata local steps section ==================
  local_step_ad_Server3 := (
    -- connection s_busy - s_idle
    ( ( a_Server3.state = s_busy & e_r3 ) & ( next(a_Server3.state) = s_idle ) )
  );
  next_local_step_ad_Server3 := ( local_step_ad_Server3
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
  );
  a_Server4_getCurrentIndex := (   a_Server4.state = s_idle ? 0 :  (
a_Server4.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server4 automata local steps section ==================
  local_step_ad_Server4 := (
    -- connection s_busy - s_idle
    ( ( a_Server4.state = s_busy & e_r4 ) & ( next(a_Server4.state) = s_idle ) )
  );
  next_local_step_ad_Server4 := ( local_step_ad_Server4
    & a_Server1.unchanged
```

```
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
  );
  a_Server5_getCurrentIndex := (   a_Server5.state = s_idle ? 0 :  (
a_Server5.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server5 automata local steps section ==================
  local_step_ad_Server5 := (
    -- connection s_busy - s_idle
    ( ( a_Server5.state = s_busy & e_r5 ) & ( next(a_Server5.state) = s_idle ) )
  );
  next_local_step_ad_Server5 := ( local_step_ad_Server5
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
  );
  a_Server6_getCurrentIndex := (   a_Server6.state = s_idle ? 0 :  (
a_Server6.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server6 automata local steps section ==================
  local_step_ad_Server6 := (
    -- connection s_busy - s_idle
    ( ( a_Server6.state = s_busy & e_r6 ) & ( next(a_Server6.state) = s_idle ) )
  );
  next_local_step_ad_Server6 := ( local_step_ad_Server6
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server7.unchanged
  );
  a_Server7_getCurrentIndex := (   a_Server7.state = s_idle ? 0 :  (
a_Server7.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server7 automata local steps section ==================
  local_step_ad_Server7 := (
    -- connection s_busy - s_idle
    ( ( a_Server7.state = s_busy & e_r7 ) & ( next(a_Server7.state) = s_idle ) )
  );
  next_local_step_ad_Server7 := ( local_step_ad_Server7
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_t2
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
```

```
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server2.state = s_idle & next(a_Server2.state) = s_busy )
    )
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
)
|
(
    e_t3
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server3.state = s_idle & next(a_Server3.state) = s_busy )
    )
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
)
|
(
    e_t4
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server4.state = s_idle & next(a_Server4.state) = s_busy )
    )
    & a_Server5.unchanged
```

```
        & a_Server6.unchanged
        & a_Server7.unchanged
    )
    |
    (
        e_t5
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server5.state = s_idle & next(a_Server5.state) = s_busy )
        )
        & a_Server6.unchanged
        & a_Server7.unchanged
    )
    |
    (
        e_t6
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
```

```
      -- connection s_busy - s_busy
       ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
      )
      &
      (
      -- connection s_idle - s_busy
       ( a_Server6.state = s_idle & next(a_Server6.state) = s_busy )
      )
      & a_Server7.unchanged
  )
  |
  (
      e_t7
      &
      (
      -- connection s_busy - s_busy
       ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
      )
      &
      (
      -- connection s_busy - s_busy
       ( a_Server6.state = s_busy & next(a_Server6.state) = s_busy )
      )
      -- connection s_idle - s_busy
      & ( a_Server7.state = s_idle & next(a_Server7.state) = s_busy )
  )
);
-- ================== identifiers section ==================
i_lambda := ( 2 );

i_mu := ( 1 );

-- ================== nb operators section ==================
-- ================== events section ==================
e_t1 := ( i_lambda > 0 );

e_t2 := ( i_lambda > 0 );

e_t3 := ( i_lambda > 0 );
```

```
  e_t4 := ( i_lambda > 0 );
  e_t5 := ( i_lambda > 0 );
  e_t6 := ( i_lambda > 0 );
  e_t7 := ( i_lambda > 0 );
  e_r1 := ( i_mu > 0 );
  e_r2 := ( i_mu > 0 );
  e_r3 := ( i_mu > 0 );
  e_r4 := ( i_mu > 0 );
  e_r5 := ( i_mu > 0 );
  e_r6 := ( i_mu > 0 );
  e_r7 := ( i_mu > 0 );
```

APÊNDICE AN – ARQUIVO TEXTUAL DO MODELO FAS10C.SAN

```
identifiers

  // Acquisition rate
  lambda = 2;
  // Release rate
  mu = 1;

events

  loc t1 (lambda);
  syn t2 (lambda);
  syn t3 (lambda);
  syn t4 (lambda);
  syn t5 (lambda);
  syn t6 (lambda);
  syn t7 (lambda);
  syn t8 (lambda);
  syn t9 (lambda);
  syn t10 (lambda);
  loc r1 (mu);
  loc r2 (mu);
  loc r3 (mu);
  loc r4 (mu);
  loc r5 (mu);
  loc r6 (mu);
  loc r7 (mu);
  loc r8 (mu);
  loc r9 (mu);
  loc r10 (mu);

reachability = 1;

network FAS10c (continuous)

  aut Server1 stt idle to (busy) t1
              stt busy to (idle) r1
                       to (busy) t2 t3 t4 t5 t6 t7 t8 t9 t10

  aut Server2 stt idle to (busy) t2
              stt busy to (idle) r2
                       to (busy) t3 t4 t5 t6 t7 t8 t9 t10

  aut Server3 stt idle to (busy) t3
              stt busy to (idle) r3
                       to (busy) t4 t5 t6 t7 t8 t9 t10

  aut Server4 stt idle to (busy) t4
              stt busy to (idle) r4
                       to (busy) t5 t6 t7 t8 t9 t10

  aut Server5 stt idle to (busy) t5
              stt busy to (idle) r5
                       to (busy) t6 t7 t8 t9 t10

  aut Server6 stt idle to (busy) t6
```

```
            stt busy to (idle) r6
                     to (busy) t7 t8 t9 t10

aut Server7 stt idle to (busy) t7
            stt busy to (idle) r7
                     to (busy) t8 t9 t10

aut Server8 stt idle to (busy) t8
            stt busy to (idle) r8
                     to (busy) t9 t10

aut Server9 stt idle to (busy) t9
            stt busy to (idle) r9
                     to (busy) t10

aut Server10 stt idle to (busy) t10
             stt busy to (idle) r10
```

APÊNDICE AO – ARQUIVO TEXTUAL DO MODELO FAS10C.SMV

```
MODULE ad_Server1()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server2()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server3()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server4()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server5()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server6()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server7()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server8()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server9()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE ad_Server10()
VAR state : {s_idle,s_busy};
DEFINE
unchanged := state = next(state);
-- ============================================================================
MODULE main
VAR
  a_Server1 : ad_Server1();
  a_Server2 : ad_Server2();
  a_Server3 : ad_Server3();
```

```
  a_Server4 : ad_Server4();
  a_Server5 : ad_Server5();
  a_Server6 : ad_Server6();
  a_Server7 : ad_Server7();
  a_Server8 : ad_Server8();
  a_Server9 : ad_Server9();
  a_Server10 : ad_Server10();

INIT
  1 = 1
TRANS
  next_local_step_ad_Server1
  |  synchronized_steps
  |  next_local_step_ad_Server2
  |  next_local_step_ad_Server3
  |  next_local_step_ad_Server4
  |  next_local_step_ad_Server5
  |  next_local_step_ad_Server6
  |  next_local_step_ad_Server7
  |  next_local_step_ad_Server8
  |  next_local_step_ad_Server9
  |  next_local_step_ad_Server10
DEFINE
  a_Server1_getCurrentIndex := (   a_Server1.state = s_idle ? 0 :  (
a_Server1.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server1 automata local steps section ==================
  local_step_ad_Server1 := (
    -- connection s_idle - s_busy
    ( ( a_Server1.state = s_idle & e_t1 ) & ( next(a_Server1.state) = s_busy ) )
    |
    -- connection s_busy - s_idle
    ( ( a_Server1.state = s_busy & e_r1 ) & ( next(a_Server1.state) = s_idle ) )
  );
  next_local_step_ad_Server1 := ( local_step_ad_Server1
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server2_getCurrentIndex := (   a_Server2.state = s_idle ? 0 :  (
a_Server2.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server2 automata local steps section ==================
  local_step_ad_Server2 := (
    -- connection s_busy - s_idle
    ( ( a_Server2.state = s_busy & e_r2 ) & ( next(a_Server2.state) = s_idle ) )
  );
  next_local_step_ad_Server2 := ( local_step_ad_Server2
    & a_Server1.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
```

```
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server3_getCurrentIndex := (   a_Server3.state = s_idle ? 0 :  (
a_Server3.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server3 automata local steps section ==================
  local_step_ad_Server3 := (
    -- connection s_busy - s_idle
    ( ( a_Server3.state = s_busy & e_r3 ) & ( next(a_Server3.state) = s_idle ) )
  );
  next_local_step_ad_Server3 := ( local_step_ad_Server3
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server4_getCurrentIndex := (   a_Server4.state = s_idle ? 0 :  (
a_Server4.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server4 automata local steps section ==================
  local_step_ad_Server4 := (
    -- connection s_busy - s_idle
    ( ( a_Server4.state = s_busy & e_r4 ) & ( next(a_Server4.state) = s_idle ) )
  );
  next_local_step_ad_Server4 := ( local_step_ad_Server4
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server5_getCurrentIndex := (   a_Server5.state = s_idle ? 0 :  (
a_Server5.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server5 automata local steps section ==================
  local_step_ad_Server5 := (
    -- connection s_busy - s_idle
    ( ( a_Server5.state = s_busy & e_r5 ) & ( next(a_Server5.state) = s_idle ) )
  );
  next_local_step_ad_Server5 := ( local_step_ad_Server5
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server6_getCurrentIndex := (   a_Server6.state = s_idle ? 0 :  (
a_Server6.state = s_busy ? 1 :  -1  )   );
```

```
  -- ================== ad_Server6 automata local steps section ==================
  local_step_ad_Server6 := (
    -- connection s_busy - s_idle
    ( ( a_Server6.state = s_busy & e_r6 ) & ( next(a_Server6.state) = s_idle ) )
  );
  next_local_step_ad_Server6 := ( local_step_ad_Server6
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server7_getCurrentIndex := (   a_Server7.state = s_idle ? 0 :  (
a_Server7.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server7 automata local steps section ==================
  local_step_ad_Server7 := (
    -- connection s_busy - s_idle
    ( ( a_Server7.state = s_busy & e_r7 ) & ( next(a_Server7.state) = s_idle ) )
  );
  next_local_step_ad_Server7 := ( local_step_ad_Server7
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server8_getCurrentIndex := (   a_Server8.state = s_idle ? 0 :  (
a_Server8.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server8 automata local steps section ==================
  local_step_ad_Server8 := (
    -- connection s_busy - s_idle
    ( ( a_Server8.state = s_busy & e_r8 ) & ( next(a_Server8.state) = s_idle ) )
  );
  next_local_step_ad_Server8 := ( local_step_ad_Server8
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
  );
  a_Server9_getCurrentIndex := (   a_Server9.state = s_idle ? 0 :  (
a_Server9.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server9 automata local steps section ==================
  local_step_ad_Server9 := (
    -- connection s_busy - s_idle
    ( ( a_Server9.state = s_busy & e_r9 ) & ( next(a_Server9.state) = s_idle ) )
  );
```

```
  next_local_step_ad_Server9 := ( local_step_ad_Server9
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server10.unchanged
  );
  a_Server10_getCurrentIndex := (   a_Server10.state = s_idle ? 0 :  (
a_Server10.state = s_busy ? 1 :  -1  )   );
  -- ================== ad_Server10 automata local steps section
==================
  local_step_ad_Server10 := (
    -- connection s_busy - s_idle
    ( ( a_Server10.state = s_busy & e_r10 ) & ( next(a_Server10.state) = s_idle )
)
  );
  next_local_step_ad_Server10 := ( local_step_ad_Server10
    & a_Server1.unchanged
    & a_Server2.unchanged
    & a_Server3.unchanged
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
  );
  -- ================== SYNCHRONIZED STEPS ==================
  synchronized_steps := (
    (
        e_t2
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server2.state = s_idle & next(a_Server2.state) = s_busy )
        )
        & a_Server3.unchanged
        & a_Server4.unchanged
        & a_Server5.unchanged
        & a_Server6.unchanged
        & a_Server7.unchanged
        & a_Server8.unchanged
        & a_Server9.unchanged
        & a_Server10.unchanged
    )
    |
    (
        e_t3
        &
        (
```

```
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server3.state = s_idle & next(a_Server3.state) = s_busy )
    )
    & a_Server4.unchanged
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
)
|
(
    e_t4
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server4.state = s_idle & next(a_Server4.state) = s_busy )
    )
    & a_Server5.unchanged
    & a_Server6.unchanged
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
)
|
(
    e_t5
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
```

```
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server5.state = s_idle & next(a_Server5.state) = s_busy )
        )
        & a_Server6.unchanged
        & a_Server7.unchanged
        & a_Server8.unchanged
        & a_Server9.unchanged
        & a_Server10.unchanged
    )
    |
    (
        e_t6
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server6.state = s_idle & next(a_Server6.state) = s_busy )
        )
```

```
    & a_Server7.unchanged
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
)
|
(
    e_t7
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server6.state = s_busy & next(a_Server6.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server7.state = s_idle & next(a_Server7.state) = s_busy )
    )
    & a_Server8.unchanged
    & a_Server9.unchanged
    & a_Server10.unchanged
)
|
(
    e_t8
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
```

```
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server6.state = s_busy & next(a_Server6.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server7.state = s_busy & next(a_Server7.state) = s_busy )
        )
        &
        (
        -- connection s_idle - s_busy
         ( a_Server8.state = s_idle & next(a_Server8.state) = s_busy )
        )
        & a_Server9.unchanged
        & a_Server10.unchanged
    )
    |
    (
        e_t9
        &
        (
        -- connection s_busy - s_busy
         ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
        )
        &
        (
```

```
    -- connection s_busy - s_busy
     ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server6.state = s_busy & next(a_Server6.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server7.state = s_busy & next(a_Server7.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server8.state = s_busy & next(a_Server8.state) = s_busy )
    )
    &
    (
    -- connection s_idle - s_busy
     ( a_Server9.state = s_idle & next(a_Server9.state) = s_busy )
    )
    & a_Server10.unchanged
)
|
(
    e_t10
    &
    (
    -- connection s_busy - s_busy
     ( a_Server1.state = s_busy & next(a_Server1.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server2.state = s_busy & next(a_Server2.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server3.state = s_busy & next(a_Server3.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server4.state = s_busy & next(a_Server4.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server5.state = s_busy & next(a_Server5.state) = s_busy )
    )
    &
    (
    -- connection s_busy - s_busy
     ( a_Server6.state = s_busy & next(a_Server6.state) = s_busy )
    )
    &
```

```
        (
        -- connection s_busy - s_busy
         ( a_Server7.state = s_busy & next(a_Server7.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server8.state = s_busy & next(a_Server8.state) = s_busy )
        )
        &
        (
        -- connection s_busy - s_busy
         ( a_Server9.state = s_busy & next(a_Server9.state) = s_busy )
        )
        -- connection s_idle - s_busy
        & ( a_Server10.state = s_idle & next(a_Server10.state) = s_busy )
    )
);
-- ================== identifiers section ==================
i_lambda := ( 2 );

i_mu := ( 1 );

-- ================== nb operators section ==================
-- ================== events section ==================
e_t1 := ( i_lambda > 0 );

e_t2 := ( i_lambda > 0 );

e_t3 := ( i_lambda > 0 );

e_t4 := ( i_lambda > 0 );

e_t5 := ( i_lambda > 0 );

e_t6 := ( i_lambda > 0 );

e_t7 := ( i_lambda > 0 );

e_t8 := ( i_lambda > 0 );

e_t9 := ( i_lambda > 0 );

e_t10 := ( i_lambda > 0 );

e_r1 := ( i_mu > 0 );

e_r2 := ( i_mu > 0 );

e_r3 := ( i_mu > 0 );

e_r4 := ( i_mu > 0 );

e_r5 := ( i_mu > 0 );

e_r6 := ( i_mu > 0 );

e_r7 := ( i_mu > 0 );
```

```
e_r8 := ( i_mu > 0 );

e_r9 := ( i_mu > 0 );

e_r10 := ( i_mu > 0 );
```