FRANCIELE CARLA PETRY

PLANEJAMENTO APLICADO À VERIFICAÇÃO DE BLOQUEIOS EM REDES DE PETRI

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Fabiano Silva

CURITIBA

2008

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Estados de bloqueio impedem a execução de tarefas em sistemas paralelos e concorrentes. Detectar estes estados, caso existam, é essencial para que se possa definir métodos capazes de eliminá-los. As redes de Petri são tradicionalmente utilizadas como formalismo para análise e modelagem de sistemas paralelos e concorrentes e normalmente se utilizam de técnicas computacionalmente caras para verificação de bloqueios. Neste trabalho se apresenta o problema de verificação de bloqueios em redes de Petri na forma de um problema de planejamento em Inteligência Artificial. A principal motivação é se aproveitar dos recentes e eficientes algoritmos para esta área. Tais técnicas foram definidas, automatizadas e comparadas à técnicas de verificação de bloqueios por desdobramento da rede.

Palavras Chave: Redes de Petri, Planejamento em Inteligência Artificial, Bloqueios *(Deadlocks)*.

ABSTRACT

Blocking states prevent the execution of tasks in parallel and concurrent systems. Detect these states, if any, is essential to establish methods that are able to eliminate them. Petri nets are traditionally used as a formalism for analysis and modeling of parallel and concurrent systems and often make use of computationally expensive techniques for checking for blockages. This paper presents the problem of verifying blocks in a Petri nets as a problem of planning in Artificial Intelligence. The main motivation is to take advantage of recent and efficient algorithms for this area. Such techniques have been developed, and compared to automated techniques for verification of blockades by splitting the network.

Key-words: Petri nets, Planning in Artificial Intelligence, Deadlocks

CAPÍTULO 1

INTRODUÇÃO

Na análise das características e comportamentos de sistemas paralelos e concorrentes o problema de se identificar a existência de bloqueios *(deadlocks)* é de fundamental importância, uma vez que estados de bloqueios impedem os sistemas de executarem suas tarefas de forma correta. Caso existam bloqueios nos sistemas, identificar os caminhos que levam até estes estados auxilia no desenvolvimento de uma solução para eliminá-los.

Neste âmbito, redes de Petri é uma ferramenta que tem por finalidade modelar e analisar sistemas concorrentes e paralelos. É possível por meio destas, representar o comportamento dinâmico do sistema e extrair características importantes sobre a sua funcionalidade, como por exemplo identificar a presença de bloqueios [24].

Neste trabalho se propõem uma maneira de analisar a existência de bloqueios em uma rede de Petri pela modelagem deste problema na forma de outro equivalente definido como um problema de Planejamento em Inteligência Artificial. Na verdade, este último pode ser definido por encontrar uma sequência de ações de um dado conjunto que quando aplicadas a um estado inicial leva a um estado final desejado. Se este for um estado de bloqueio do sistema é possível, encontrar o caminho que leva até ele, caso este exista.

A abordagem proposta por Silva [29] utiliza uma ação de planejamento que modela um problema de alcançabilidade em rede de Petri.

Em uma outra abordagem, Edelkamp e Jabbar [6] propõem a verificação de bloqueios em redes de Petri utilizando planejamento, onde através de ações de Planejamento a dinâmica da rede de Petri é representada.

Dada as duas abordagens, os objetivos do presente trabalho são: o desenvolvimento de novas modelagens para o problema de verificação de bloqueios em redes de Petri, para isso se fez inicialmente o estudo das modelagens apresentadas por Edelkamp, Jabbar e Silva. E em seguida a proposta de novos modelos de tradução de problema de bloqueio

em rede de Petri usando planejamento. Também se faz um comparativo de desempenho entre as modelagens apresentadas. Finalmente se compara os resultados com a técnica de verificação de bloqueios em desdobramento de rede.

O presente trabalho está assim dividido: no capítulo 2 são introduzidos os conceitos de redes de Petri, onde são ressaltadas as propriedades de alcançabilidade e existência de bloqueios. São apresentadas as abordagens clássicas encontradas na literatura.

No capítulo 3 é apresentado o conceito de Planejamento em Inteligência Artificial e as linguagens de descrição de domínios (STRIPS, ADL e PDDL).

O capítulo 4 apresenta inicialmente a modelagem de problemas de alcançabilidade em redes de Petri para problemas de planejamento proposta por Silva, bem como modelagens propostas por Edelkamp e Jabbar. As novas modelagens de verificação de bloqueios em redes de Petri utilizando planejamento também são apresentadas.

Os experimentos, a automatização das ferramentas desenvolvidas e o resultado obtido entre a comparação das modelagens, são detalhadas no capítulo 5.

As conclusões e sugestões de trabalhos futuros encontram-se no capítulo 6.

CAPÍTULO 2

REDES DE PETRI

Este capítulo apresenta os conceitos fundamentais sobre redes de Petri, onde as características de alcançabilidade e presença de bloqueio são evidenciadas, sendo estas de fundamental importância para o entendimento do trabalho realizado. Apresenta-se também as técnicas clássicas para a verificação de bloqueios em redes de Petri.

2.1 Conceitos fundamentais

Através da tese de Carl Adam Petri em 1962, intitulada "*Kommunication Mit Automaten*" (Comunicação com Autômatos), as redes de Petri tiveram seu marco inicial. Esta publicação teve por objetivo desenvolver um modelo em que as máquinas de estado fossem capazes de comunicar-se umas com as outras, o que possibilita a representação de concorrência [3].

Rede de Petri é uma ferramenta de modelagem que permite a representação de sistemas utilizando como alicerce uma forte base matemática. Elas possuem a particularidade de permitir a modelagem de sistemas paralelos, concorrentes, assíncronos e não-determinísticos [24].

As principais características das redes de Petri são: utilização de vários níveis de abstração para modelagem, eventos sincronizados, forma sistemática de verificação de propriedades do sistema, possibilidade de modelagem de sistemas concorrentes ou paralelos, representação gráfica, estados e eventos são representados de modo explícito, flexibilidade na descrição de ordem parcial entre vários eventos.

Uma rede de Petri é composta por: lugares, que representam uma condição, uma atividade ou um recurso; fichas (marcas), que representam o estado de um sistema; transições que representam um evento ou as mudanças de estado; arcos que indicam os lugares de entrada ou saída relativos às transições.

Formalmente uma rede de Petri é formada por uma quíntupla $N = (P, T, Pre, Pos, M)$ onde:

- $P = (P_1, P_2, ..., P_n)$: é o conjunto finito de lugares;
- $T = (T_1, T_2, ..., T_n)$: é o conjunto finito de transições, sendo $P \cap T = \emptyset$;
- Pre : P x $T \rightarrow \mathbb{N}$: é a função de incidência e descreve os arcos orientados que conectam lugares a transições;
- Pos : P x $T \rightarrow \mathbb{N}$: descreve os arcos orientados que fazem a conexão das transições aos lugares;
- $M : P \rightarrow \mathbb{N}$: é a marcação da rede.

A figura 2.1 apresenta uma rede onde $T = (t)$ e $P = (p, q, r)$

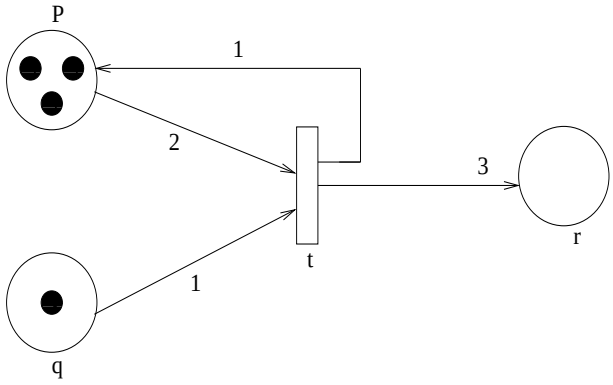

Figura 2.1: Rede de Petri marcada

A função de incidência anterior Pre descreve os arcos orientados que ligam lugares a transições. $Pre\ (p, t)$ representa o peso do arco (p, t). Se $Pre\ (p, t) = 0$, então não existe um arco saindo de um lugar p e entrando em uma transição t.

A função de incidência posterior Pos descreve os arcos orientados que ligam transições a lugares. $Pos\ (p, t)$ representa o peso do arco (t, p). Se $Pos\ (p, t) = 0$, então não existe um arco saindo de uma transição t e entrando em um lugar p.

Na rede da figura 2.1 as funções de incidência anterior são: $Pre(p, t) = 2$, $Pre(q, t) = 1$, $Pre(r, t) = 0$; E as funções de incidência posterior são: $Pos(p, t) = 1$, $Pos(q, t) = 0$ e $Pos(r, t) = 3$.

O comportamento de um sistema pode ser descrito pelas mudanças em seus estados. Um sistema modelado em uma rede de Petri simula essa dinâmica de acordo com uma regra de disparo de transição [24], que determina a mudança de estado através do consumo e geração de marcas.

Como as funções de incidência podem ser vistas como matrizes, a partir destas pode-se calcular a matriz de incidência C, ($C = Pos$ - Pre), que fornece o balanço das marcas na rede quando efetuado o disparo das transições.

Para fins de exemplificação da matriz de incidência será utilizando a rede da figura 2.1. A representação matricial da rede segue:

$$Pre = \begin{array}{c} t \\ \left[\begin{array}{c} 2 \\ 1 \\ 0 \end{array}\right] \end{array} \begin{array}{l} p \\ q \\ r \end{array} \qquad Pos = \begin{array}{c} t \\ \left[\begin{array}{c} 1 \\ 0 \\ 3 \end{array}\right] \end{array} \begin{array}{l} p \\ q \\ r \end{array}$$

$$C = \begin{array}{c} t \\ \left[\begin{array}{r} 1 \\ -1 \\ 3 \end{array}\right] \end{array} \begin{array}{l} p \\ q \\ r \end{array}$$

Para representar a marcação da rede utiliza-se um vetor coluna M, cujo tamanho é a quantidade de lugares. $M(p)$ define o número de marcas em um lugar p por exemplo, assim, $M(p) = 3$, $M(q) = 1$, $M(r) = 0$. Portanto, a marcação inicial da rede é o vetor transposto $M_0 = [3\ 1\ 0]^t$.

2.2 Alcançabilidade em redes de Petri

Diz-se que uma transição t em uma rede de Petri está sensibilizada (ou habilitada) se: $\forall p \in P, M(p) \geq Pre(p,t)$, ou seja, se para t o número de marcas em cada um dos lugares de entrada for maior ou igual ao peso do arco que liga este lugar à transição t.

Esta equação pode ser escrita de forma alternativa como:

$$M \geq Pre(.,t)$$

onde o vetor coluna $Pre(.,t)$ é a coluna da matriz Pre referente a transição t em M.

Pode-se definir $Pre(.,t)$ como o conjunto contendo todos os lugares que antecedem uma transição t, ou seja, seu pré-conjunto, e $Pos(.,t)$ como o conjunto contendo todos os lugares que sucedem uma transição t, ou seja, seu pós-conjunto [3].

Se t é uma transição sensibilizada por uma marcação M, uma nova marcação M' é obtida através do disparo de t. A nova marcação M' é dada pela equação:

$$M' = M - Pre(.,t) + Pos(.,t)$$

Como $C = Pos - Pre$, também pode-se escrever assim:

$$M' = M + C(.,t)$$

O disparo de t representa uma operação que consiste em consumir de $Pre(p,t)$ marcas de cada lugar precedente p (peso do arco de entrada) e colocar $Pos(p,t)$ marcas em cada lugar seguinte a p [3].

Assim, baseado na rede da figura 2.1, após o disparo da transição t uma nova marcação é gerada consistindo em $M_0 = [2\ 0\ 3]^t$ que pode ser visualizada através da seguinte figura:

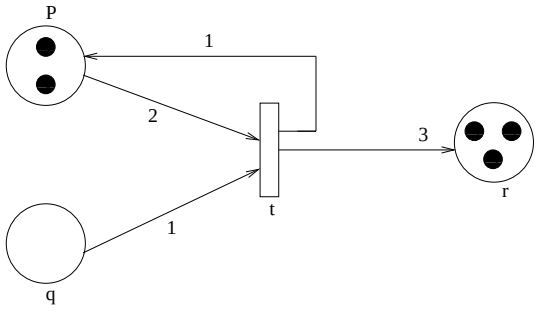

Figura 2.2: Rede de Petri após o disparo da transição t

O disparo de uma transição t sobre uma marcação M obtendo uma marcação M' define uma relação entre as marcações M e M'. Esta relação entre as marcações da rede é chamada de *relação de alcançabilidade*, M' é alcançável a partir de M ($M \xrightarrow{t} M'$).

Quando uma transição está sensibilizada por uma marcação e pode disparar, esta leva à uma nova marcação. Esta nova marcação estando sensibilizada e também disparando, levará a outra nova marcação, o disparo em seqüência das transições é chamado de *seqüência de disparo*.

Para generalizar esta fórmula e calcular uma nova marcação após o disparo de uma seqüência de transições é necessária a utilização de um vetor $\bar{s}$, chamado de vetor característico da sequência s. Este vetor possui dimensão igual ao número de transições da rede; $\bar{s}(t)$ corresponde ao número de vezes que a transição t foi disparada. Tem-se então, a equação fundamental das redes de Petri que é dada por:

$$M' = M + C.\bar{s}$$

Uma relação de alcançabilidade entre marcações de uma transição disparável pode ser estendida para a alcançabilidade do disparo de uma seqüência de transições, ou seja, em uma rede de Petri N é dito que a marcação M_g é alcançável a partir da marcação M se e somente se existe uma seqüência de transições s tal que: $M \xrightarrow{s} M_g$.

Se o conjunto de marcações alcançáveis de uma rede de Petri é finito, é dito que esta é *limitada*, assim, o número de marcas em cada lugar da rede é finito, limitado por um número natural.

Um lugar p de uma rede marcada M é *k-limitado* se, e somente se:

$$\forall M' \in A(N, M), M'(p) \leq k$$

Se $k = 1$ diz-se que o lugar é binário (seguro).

Se o número de marcas em um lugar da rede de Petri é infinito, denota-se que esta é *ilimitada*. Neste caso, para representar a quantidade de marcas neste lugar utiliza-se o símbolo ω.

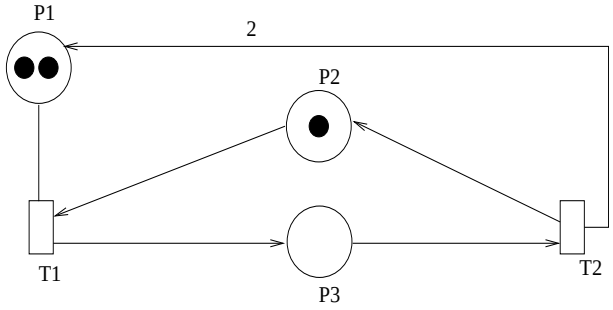

Figura 2.3: Rede de Petri ilimitada

2.3 Bloqueios em redes de Petri

Análises sobre redes de Petri podem revelar importantes informações sobre a estrutura, a dinâmica e o comportamento do sistema modelado. Estas informações podem ser usadas para a avaliação e sugestões de melhorias ou mudanças no mesmo.

Por exemplo, uma rede marcada $N =< R, M >$ é reiniciável se e somente se:

$$\forall M' \in A(N, M), \exists s | M' \xrightarrow{s} M$$

a partir de qualquer marcação acessível M', é possível encontrar uma seqüência de disparo s que leve a rede de volta à marcação inicial M.

Uma rede de Petri marcada $N =< R, M >$ é viva se e somente se todas as suas transições são vivas. Para ser viva, a transição deve poder ser sensibilizada a partir de qualquer marcação, através de uma seqüência de disparos.

Se uma rede de Petri encontra-se em estado de bloqueio, esta não possui nenhuma transição sensibilizada e portanto nenhum estado sucessor, conseqüentemente não poderá mudar de estado. Caso sejam detectados bloqueios na rede, é importante conhecer as seqüências de disparos de transições que levam aos mesmos. A modelagem em redes de Petri da estrutura de um sistema possibilita a identificação de bloqueios, entre outras características importantes.

Song e Lee [30] definem bloqueio em redes de Petri como sendo um estado na rede que pára o fluxo de marcações, esperando por mais recursos, para que possa efetuar mais disparos de transições, porém não há mais recursos que possibilitem a habilitação das

transições para efetuar disparos.

Em outras palavras, não existe nenhuma marcação na rede que sensibilize uma transição, assim nenhuma transição pode ser disparada e o sistema pára [23].

Além disso, um conjunto S de lugares nunca irá ser marcado novamente, depois de ter perdido todas as suas marcas, se e somente se nenhuma transição, que contenha em seu pós-conjunto um lugar pertencente a S, disparar novamente. Ao definir $\bullet S$ como o pré-conjunto composto pelas transições que antecedem um conjunto de lugares S, e $S\bullet$ como o pós-conjunto composto pelas transições que precedem um conjunto de lugares S, pode-se dizer que, em particular, este é o caso em que todas estas transições também contém um lugar pertecente a S em seu pré-conjunto, ou seja

$$\forall t \in T_N : t \in \bullet S \Rightarrow t \in S\bullet$$

ou equilavente $\bullet S \subseteq S\bullet$, um conjunto de lugares que se encontram nessas condições correspondem a um estado de bloqueio na rede de Petri [27].

Na figura 2.4 é ilustrada uma rede de Petri com bloqueio:

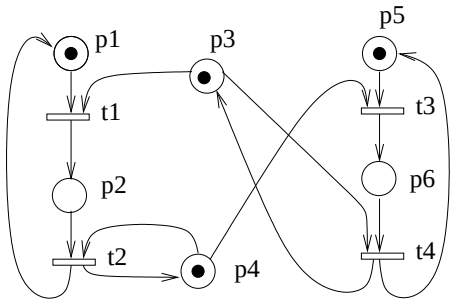

Figura 2.4: Rede de Petri com presença de bloqueios

Na figura pode-se observar que a marcação inicial da rede é $M_0 = [1\ 0\ 1\ 1\ 1\ 0]^t$. Para visualizar as seqüências de disparos que levam ao estado de bloqueio na rede, será utilizado um grafo de alcançabilidade, uma vez que se o conjunto de marcações acessíveis de uma rede de Petri for finito, pode-se representá-lo sob a forma de um grafo $GA(R, M)$, cujos nós são as marcações acessíveis do conjunto $A(R, M)$ [3], assim é ilustrado na figura 2.5 o grafo de alcançabilidade da rede de Petri da figura 2.4 :

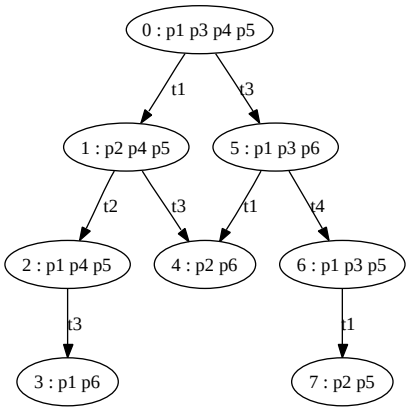

Figura 2.5: Grafo de alcançabilidade da rede de Petri da figura 2.4

Pode-se perceber pelo grafo de alcançabilidade da figura 2.5 que é possível evidênciar as seqüências de disparos de transições que levam a rede de Petri ao estado de bloqueio, assim, neste caso, é possível enumerar estas seqüências de disparos, como segue: Seqüência 1:

$$t_1 - t_3$$

, ou seqüência 2:

$$t_1 - t_2 - t_3$$

, ou seqüência 3:

$$t_3 - t_1$$

, ou seqüência 4:

$$t_3 - t_4 - t_1$$

2.4 Desdobramento

O conceito de desdobramento foi introduzido originalmente por McMillan [21] com o objetivo de evitar o problema da explosão do espaço de estados na análise dos sistemas modelados em redes de Petri.

Em 1996, Esparza e colegas [7, 8] fizeram melhorias no algoritmo original. A técnica é um método para resolver o problema de alcançabilidade que explora e preserva as informações de concorrência da rede. O desdobramento de uma rede é uma outra rede, acíclica, finita e que possui todas as marcações alcançáveis da rede original.

Para entendimento da técnica faz-se necessário algumas definições, as quais podem ser evidênciadas a seguir.

Definição 1 (Rede de Ocorrência) *Uma rede $O = (B, E, F)$ é uma rede de ocorrência se e somente se:*

(i) $\forall b \in B\,, |{}^\bullet b| \leq 1$ *(os lugares têm no máximo uma transição de entrada)*

(ii) F *é acíclico (o fechamento transitivo é irreflexivo)*

(iii) O *é finitamente precedido (*$\forall x \in B \cup E$*, o conjunto de elementos* $y \in B \cup E$*, tal que* $y \leqslant x$*, é finito)*

(iv) Nenhum evento $e \in E$ *está em auto-conflito (um nó* $x \in B \cup E$ *está em auto-conflito se* $x \# x$*)*

Definição 2 (*Processo de ramificação*) *ou simplesmente processos, são desdobramentos de uma rede que representam todos os caminhos possíveis mantendo as informações sobre concorrência e conflitos. O processo de ramificação de uma rede N é o par $\beta = (ON, \varphi)$ onde $ON = (B, E, F)$ é uma rede de ocorrência e φ é uma função de rotulamento que satisfaz as seguintes propriedades:*

(i) $\varphi(B) \subseteq S$ *e* $\varphi(E) \subseteq T$ *(preserva a natureza dos nós)*

(ii) $\forall e \in E$*, a restrição de* φ *para* ${}^\bullet e$ *é uma função bijetora entre* ${}^\bullet e$ *(em* N*) e* ${}^\bullet\varphi(e)$ *(em* β*), e simetricamente para* $e^\bullet$ *e* $\varphi(e)^\bullet$ *(preserva o ambiente das transições)*

(iii) *a restrição de φ para $Min(O)$[1] é uma função bijetora entre $Min(O)$ e M_0 (β começa pela marcação inicial)*

(iv) *$\forall e_1, e_2 \in E$, se ${}^\bullet e_1 = {}^\bullet e_2$ e $\varphi(e_1) = \varphi(e_2)$ então $e_1 = e_2$ (β não duplica as transições de N)*

É possível truncar o processo de ramificação em uma sub-rede, denominado prefixo finito, que possui todas as marcações alcançáveis. Esse prefixo corresponde ao desdobramento, para fins de exemplificação o desdobramento da rede da figura 2.4 é mostrado:

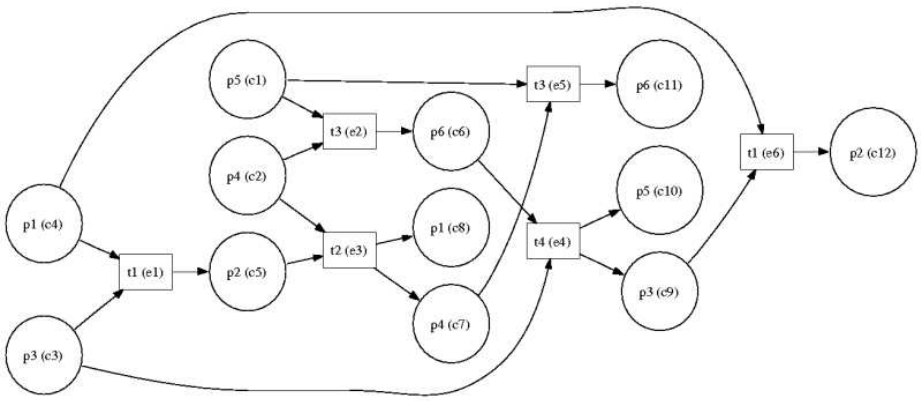

Figura 2.6: Desdobramento da rede de Petri 2.4

[1] $Min(O)$ é o conjunto dos elementos mínimos de $B \cup E$ que possuem seu pré-conjunto vazio. Considerando redes onde não existem transições com pré-conjunto vazio, $Min(O)$ só podem ser condições (B), portando $Min(O) = M_0$.

Ainda, para entendimento da técnica de desdobramento, se faz necessário os conceitos de configuração, configuração local, prefixo finito completo e evento de corte.

Definição 3 (Configuração) *Uma configuração C é um conjunto de transições que satisfazem as seguintes condições:*

(i) $\forall e' \leqslant e,\ e \in C \Rightarrow e' \in C$

(ii) $\forall e_1, e_2 \in C,\ e_1 \neq e_2 \Rightarrow^{\bullet} e_1 \cap^{\bullet} e_2 = \emptyset$

Uma configuração pode ser associada com uma marcação $\texttt{Mark}(C)$ que corresponde a uma marcação alcançável a partir de M_0 após todas as transições de C terem sido disparadas.

$$\texttt{Mark}(C) = p((Min(N) \cup C^{\bullet}) \backslash {}^{\bullet}C)$$

Definição 4 (Configuração Local) *A configuração local de um evento e, denotada por $[e]$, é a configuração mínima que contém e e todos os seus precedentes.*

Definição 5 (Prefixo Finito Completo) *Na maioria dos casos, o desdobramento β da rede é infinito, mas é possível truncá-lo em uma subrede denominada prefixo finito completo β', que possui todas as informações de β. Formalmente, o prefixo β' de β é completo se para cada marcação alcançável M, existe uma configuração $C \in \beta'$ tal que:*

1. $\mathtt{Mark}(C) = M$
2. *para cada transição t habilitada por M, existe uma configuração $C \cup \{e\}$ tal que $e \notin C$ e $\varphi(e) = t$*

Definição 6 (Evento de corte) *Sendo $\prec$ uma ordem adequada (adequate order) nas configurações do desdobramento e seja β o seu prefixo contendo um evento e. O evento e é um evento de corte de β se β contém a configuração local $[e']$ tal que:*

(i) $\mathtt{Mark}([e]) = \mathtt{Mark}([e'])$

(ii) $[e'] \prec [e]$

Quando $[e'] < [e]$, pode-se dizer que $[e']$ é menor que $[e]$ se ele tiver menos eventos, ou seja, $|[e']| < |[e]|$. Como isso é fácil verificar que essa ordem parcial é uma ordem adequada, portanto $[e'] \prec [e]$.

2.4.1 Verificação de bloqueios em redes de Petri

No campo de análise de sistemas concorrentes, a ausência de bloqueios geralmente é uma propriedade desejável. Por esta razão, ao longo dos anos, evidenciaram-se pesquisas que tiveram por objetivo a verificação de bloqueios nestes sistemas [23, 4].

Murata, Shenker e Shatz em 1989 [25] apresentam um método de detecção de bloqueios em programas na linguagem de programação *Ada* [26, 12] usando a análise da estrutura e a dinâmica das redes de Petri.

Primeiramente, um programa *Ada* é traduzido para um rede de Petri denominada *rede Ada*. A verificação de bloqueios na *rede Ada* é baseada na sua estrutura usando invariantes de lugar e invariantes de transição [25].

Um invariante de lugar em uma rede de Petri é uma função da marcação dos lugares cujo valor é uma constante que depende apenas da marcação inicial da rede. Um invariante de transição corresponde a uma seqüência cíclica de eventos que pode ser repetida indefinidamente, os cálculos de invariantes são efetivados mediante a matriz de incidência.

O algoritmo apresentado por eles mostrou-se mais eficiente na análise da dinâmica da rede do que a análise utilizando grafos de alcançabilidade.

McMillan em 1992 [22], apresentou uma maneira de detectar bloqueios em desdobramento de rede analisando as suas configurações. Uma configuração se faz conter todo o histórico das marcações alcançáveis da rede e de todos os eventos que podem ocorrer durante este processo. Ele observa que não existe um bloqueio na rede se cada configuração do desdobramento pode ser estendida para uma configuração contendo ao menos um evento de corte.

Um evento de corte é uma condição para a parada da construção do desdobramento da rede, e consiste em identificar se um evento alcançado durante a construção do desdobramento já foi identificado anteriormente. Caso a resposta seja positiva, a parada da

construção do desdobramento é efetuada.

A rede contém um bloqueio se e somente se existe uma configuração que esteja em conflito com cada evento de corte do desdobramento. Duas transições estão sem conflito estrutural se e somente se elas têm ao menos um lugar de entrada em comum; duas transições estão em conflito efetivo para uma marcação M se e somente se ambas estão em conflito estrutural e estão sensibilizadas. Assim, McMillan define um conjunto que contém os eventos da configuração que estão diretamente em conflito com um evento de corte. Caso esse conjunto esteja vazio ao final da execução do algoritmo, conclui-se que a rede não possui bloqueios. Este algoritmo é exponencial no pior caso.

Melzer e Römer, em 1997, [23] desenvolveram uma versão mais rápida do algoritmo de McMillan e compararam os resultados com um novo método proposto pelos mesmos, baseado em álgebra linear, no qual para cada lugar da rede de Petri é associado uma equação conservativa de marcas. Se a equação não tiver solução quando avaliada em uma seqüência de disparos, conclui-se pela existência de bloqueio.

Por meio de vários testes, utilizando o conjunto de problemas proposto por Corbertt [4], foram obtidos pontos fortes e fracos sobre ambas abordagens, contudo o método utilizando álgebra linear mostrou um melhor desempenho quando comparada ao algoritmo melhorado de McMillan.

Posteriormente, em 1999, uma abordagem similar ao trabalho de Melzer e Römer foi apresentado por Heljanko, onde o problema de verificação de bloqueios de um prefixo completo finito é traduzido para um problema de programação em lógica [14].

Inicialmente, é efetuado um mapeamento do conjunto de eventos, e suas condições, de um prefixo completo finito para um conjunto de átomos em programação em lógica. Para encontrar o bloqueio é modelada a habilitação de cada evento, de corte ou não, que pode levar até outro evento do prefixo, isso permite identificar todas as situações que não levam a um bloqueio. Essas situações que não são bloqueios são eliminadas do modelo, resultando somente nas situações de bloqueio, caso existam.

A abordagem transforma o problema de verificação de bloqueio em redes de Petri 1-safe, um problema PSPACE-completo para um problema de verificação de bloqueios em

um prefixo completo finito NP-completo.

A maior contribuição é o desenvolvimento de um método de transformação de problemas de alcançabilidade e bloqueios de redes de Petri 1-safe para problemas de busca em um modelo de estado em programação em lógica com suas devidas provas.

2.5 Considerações

O presente capítulo apresentou os conceitos fundamentais sobre redes de Petri, relevantes neste texto: alcançabilidade e presença de bloqueio, característica na qual a interrupção de uma seqüência de disparos é evidenciada por não possuir marcas suficientes para habilitar nenhuma transição ocasionando numa parada do sistema.

No campo de análise de bloqueios em redes de Petri, foram vistas as técnicas clássicas da literatura, com destaque para as abordagens utilizando invariantes de lugar e as que usam desdobramento de rede. Também foi visto que a partir destas duas, outras modelagem foram propostas, uma usando prefixo completo finito e álgebra linear e a outra usando prefixo completo finito e programação em lógica.

No presente trabalho são propostas novas técnicas de verficação de bloqueios em redes de Petri com o intuito de agregar ferramentas mais efecientes no campo da análise de sistemas.

CAPÍTULO 3

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Este capítulo apresenta conceitos fundamentais sobre a área de planejamento, linguagens de descrição de domínios e planejadores que representam o estado da arte em planejamento clássico, notadamente o planejador Metric-FF.

3.1 Conceitos fundamentais

Planejamento em IA tem seu marco inicial na década de 60, onde as primeiras abordagens para o problema adotavam como representação a lógica de primeira ordem. Para se encontrar um plano era necessário provar um teorema e resgatar os passos para a obtenção da prova. Nos anos 70, deu-se a passagem da demonstração de teoremas para as técnicas de busca em espaço de estados. O avanço nos anos 90 ocorreu em termos de eficiência e aperfeiçoamento de algoritmos, o que transformou o planejamento em uma área multidisciplinar e dinâmica.

A área de planejamento está sub-dividida em dois segmentos: o planejamento clássico e não-clássico.

No planejamento clássico a preocupação está principalmente na geração de planos que resultem em um conjunto de objetivos previamente definidos para uma situação. O planejamento clássico se distingue por:

- Apresentar resultantes determinísticas;
- Fazer uso de estados estáticos para representação do mundo;
- Mudanças do mundo ocorrem apenas como resultado do efeito de uma ação.

No planejamento não-clássico, o agente se depara com situações imprevisíveis, onde o grau de certeza da ação nem sempre é absoluto. Isso pode ocorrer deviado á variáveis externas, como fatores climáticos, alteração de ambientes, variações de energia entre outros.

Um problema de planejamento em inteligência artificial pode ser definido como o processo para encontrar uma seqüência de ações de um dado conjunto, que quando aplicadas a um estado inicial leva a um estado final desejado. Esta formulação de problema de planejamento é abstrata, ela especifica uma classe de problemas de planejamento parametrizado por linguagens para representar o mundo, os objetivos e as ações.

Planejamento pode ser considerado como um problema de busca em um espaço de estados que representam o mundo. Um estado é um conjunto finito de propriedades ou proposições que descrevem um particular instante do mundo.

Um planejador, ou resolvedor automático de problemas de planejamento é um sistema que busca encontrar um conjunto de ações para solucionar problemas pré-definidos. Assim, a partir de uma informação inicial, procura-se uma combinação correta de ações, que leva um agente do estado inicial ao estado objetivo.

Um algoritmo que resolve um problema de planejamento possui três entradas:

- A descrição do estado inicial do mundo;
- A descrição do objetivo do agente;
- A descrição do conjunto de ações que podem ser executadas pelo agente.

Pela sua natureza computacional estas descrições devem ser fornecidas em uma linguagem formal bem definida. Até 1970 a base era a lógica de primeira ordem, que, à época apresentava problemas inerentes à área de raciocínio sobre ação [19].

Fikes e Nilson alternativamente, propuseram em 1971 [9] o sistema STRIPS (*Stanford Research Institute Problem Solver*), que agregava uma liguagem de representação simples e um algoritmo de busca em um espaço de estados do mundo. Este algoritmo procurava uma solução usando busca em uma árvore onde os nós representavam os estados do mundo e os arcos representavam as ações do plano. O grau de mérito foi a independência entre a linguagem de representação e o algoritmo que soluciona o problema, embora com base em um formalismo menos nobre para representação do conhecimento.

A pesquisa em planejamento durante a década de 80 se restringiu a duas abordagens: a busca em STRIPS e a por prova de teorema. Ambas abordagens recaía-se em busca

exaustiva com a explosão combinatória de estados, tornado-as inificiêntes do ponto de vista da praticidade. Em 1992, Kaltz e Selman propuseram uma tradução do problema de planejamento representado em STRIPS para lógica proporcional. O planejador apresentado denominou-se *Satplan* [17].

Em 1996, Blum e Furst, apresentaram o *GraphPlan*, que a partir das descrições em STRIPS, constrói um grafo que reduz sensivelmente a representação do espaço de busca do problema, redução que se deve ao tratamento de conflitos entre ações do domínio [1].

A partir do *Satplan* e *GraphPlan*, motivados pelas inovações propostas, surgiram novos planejadores que buscavam agregar condições de análise mais precisas através de funções heurísticas para guiar as buscas. Surgiu também os chamados planejadores hídridos combinando elementos do planejamento com heurística à abordagens baseadas em grafos de planos e satisfabilidade.

No ano de 1998, surgiu a competição de planejadores automáticos, nesta competição a linguagem de representação PDDL, derivada da junção das linguagens STRIPS e ADL (*Action Description Language*) foi apresentada aos competidores. [20]. Nesta competição foi apresentado o planejador HSP [2] baseado em busca heurística, capaz de produzir planos com extrema eficiência.

Em 2000, o planejador FF (*Fast-Forward Planning System*), mostrando ser um sistema robusto e rápido [15]. Em 2002, o foco da competição baseou-se em planejamento temporal e domínios métricos, estendendo a linguagem PDDL para a versão 2.1 e tendo como destaque o planejador *Metric*-FF [15].

Em 2004 a competição de planejadores seguiu o mesmo foco da competição de 2002, visando agora problemas de planejamento mais complexos, com características tanto numéricas quanto temporais.

3.1.1 A representação *STRIPS*

O STRIPS foi um dos primeiros sistemas desenvolvidos para o problema de planejamento [9]. Composto por um modelo formal baseado em operações simples de inclusão e remoção em conjuntos, permite descrever e representar as ações e os estados do mundo

em uma linguagem simples, viabilizando a busca por uma solução em modelos clássicos de busca em espaços de estados [28].

Em STRIPS as ações são agrupadas em famílias chamadas de esquemas. A descrição das ações e dos literais dos problemas é feita a partir destes esquemas parametrizados por variáveis que são instanciadas pelas constantes existentes no problema. A representação dos estados do mundo é feita através de conjunções de literais instanciados e as ações instanciadas definem as regras de transformação entre os estados do mundo. Em suma, é uma linguagem proposicional escrita sob a forma de predicados.

Para modelar um problema de planejamento em STRIPS é necessário dividí-lo em duas partes: a descrição do domínio e a descrição do problema. A descrição do domínio contém o conjunto de literais e as ações dadas por:

- Nome e os parâmetros da ação (identificação da ação);
- Pré-condições: diz respeito ao conjunto de literais positivos, ou verdadeiros, para que a ação possar ser aplicada sobre este;
- Efeitos, corresponde a uma conjunção que pode incluir literais positivos ou negativos, que descreve as alterações que devem ser efetuadas sobre um estado do mundo, onde os literais positivos são adicionados no estado e os literias negativos são removidos.

A descrição do problema é dada por:

- Um conjunto de constantes presente no problema;
- O estado inicial do mundo, o qual é descrito por uma conjunção de literais positivos instanciados;
- O objetivo do problema, ou estado final, que é descrito por uma conjunção de literais positivos instanciados.

A descrição completa dos estados do mundo é necessária em STRIPS. O que não é explicitamente descrito é considerado falso, nenhum literal negativo é incluído nas descrições dos estados, as pré-condições das ações são sempre positivas.

Quando for efetuada a entrada da descrição do domínio que contém os literais e as ações, bem como a descrição do problema a qual é composta por constantes, estado inicial e estado final, o planejador retorna uma seqüência de ações que transformam o estado inicial no estado final desejado.

Por exemplo, o problema de transportar uma carga de um lugar à outro, utilizando um avião. A ação que modela o carregamento da carga em um avião em STRIPS pode ser vista a seguir, onde as pré-condições da ação são: estar a carga no aeroporto, estar o avião no aeroporto, a carga, o avião e o aeroporto; Como efeito: a cargar estar no avião e não mais estar no aeroporto.

$ACAO(Carregar(c, p, a)$,
$PRECONDICAO : Estar(c, a) \wedge Estar(p, a) \wedge Carga(c) \wedge Aviao(p) \wedge Aeroporto(a)$
$EFEITO : \neg Estar(c, a) \wedge Em(c, p))$

3.1.2 ADL

Devido à limitações de STRIPS, surgiram derivações com base em sua estrutura, entre elas a linguagem de descrição ADL [28].

ADL tem como base um modelo algébrico para caracterizar os estados do mundo e supõe um replanejamento do cálculo de situações mediante a definição de uma ação *(a)* como uma tupla $< Pre, Add, Del, Update(s, t) >$ onde:

- Pre são as pré-condições da ação a, Add e del representam conjuntos de fórmulas para acrescentar e excluir respectivamente ao executar a ação;

- $Update(s, t)$ faz a descrição do conjunto de pares de estados, de forma que ao executar a ação a no estado s obtêm-se o estado t.

As principais características da linguagem ADL são: (1) nos estados são permitidos literais positivos e negativos, (2) literais não mensionados são desconhecidos. (3) Os objetivos podem ser conjunções e disjunções, (4) suporta quantificador nas variáveis dos esquemas, (5) variáveis podem ter tipos e (6) as ações podem ter efeitos condicionais.

Para o presente trabalho é de fundamental importância a característica disponibilizada pela linguagem ADL que permite literais negativos, uma vez que haverá a necessidade de análise de estados que serão representados por este literais.

Para exemplicar uma ação em ADL utilizar-se-á o mesmo problema descrito em STRIPS, neste caso a ação carregar que pode ser vista a seguir, terá como pré-condições: estar a carga no aeroporto e estar o avião no aeroporto; Como efeito a cargar estar no avião e não mais no aeroporto:

$ACAO(Carregar(c : carga, p : aviao, a : aeroporto)$,
$PRECONDICAO : Estar(c, a) \wedge Estar(p, a)$
$EFEITO : \neg Estar(c, a) \wedge Em(c, p))$

3.1.3 PDDL

A linguagem de definição de domínios em planejamento PDDL foi criada por Drew McDermott em 1998 [20], para a competição de planejadores. Com objetivo de estabelecer um padrão de notação para a descrição de domínios usados como referência na análise de desempenho dos planejadores.

A linguagem PDDL mostrou ser uma linguagem com alta capacidade de representação, a qual permite extensões como: (1) efeitos condicionais, (2) efeitos quantificados, (3) decomposição hieráquica, (4) variáveis tipadas, (5) fluentes (fatos associados a instantes de tempo, ou seja, um fato que é válido em um momento, pode não ser válido em outro), (6) avaliação e comparação de expressões aritméticas, entre outros. Em sua versão 2.1 [11], faz o tratamento de fluentes numéricos, representação de tempo e duração. Já na extensão 2.2 [5] inclui predicados derivados cujos valores são derivados de um conjunto de regras,estes não são afetados por nenhuma das ações disponíveis no plano.

A PDDL é caracterizada por separar a descrição de ações parametrizadas das descrições dos estados iniciais e do objetivo. O problema de planejamento é constituído do arquivo de descrição do domínio e do arquivo de descrição do problema. Uma mesma descrição de domínio pode ser utilizada para a solução de várias descrições de problemas,

dentro do mesmo escopo.

No arquivo de domínio são descritos todos os elementos que são necessários para modelar o seu comportamento. Nele estão definidos os tipos, as funções, os predicados (literais) e as ações do domínio. Nas ações, descritas no domínio, são definidas as pré-condições para execução destas e os efeitos que ocorrem após a execução. Para ações com métricas de tempo, é necessário definir a duração da execução de cada ação.

Para fins de exemplificação, imagine que um veículo tenha que se deslocar de uma cidade a outra. Pontos importantes a serem levados em consideração são, a quantidade de combustível no carro e quantidade de combustível que se gasta para deslocar-se de uma cidade a outra. Tais informações farão parte da modelagem das ações para que um planejador automático resulte em um plano para este domínio.

A seguir é mostrada a modelagem do domínio para o problema citado:

```
(define (domain metricVehicle)
(:requirements :strips)
(:types vehicle location)
(:predicates (at ?v - vehicle ?p - location)
             (acessible ?v - vehicle ?p1 ?p2 - location))

(:functions (fuel-level ?v -vehicle)
            (fuel-used ?v -vehicle)
            (fuel-required ?p1 ?p2 - location)
            (total-fuel-used))

(:action drive
 :parameters (?v - vehicle ?from ?to - location)
 :precondition (and (at ?v ?from)
                    (acessible ?v ?from ?to)
                    (>= (fuel-level ?v) (fuel-required ?from ?to)))
 :effect (and (not (at ?v ?from))
              (at ?v ?to)
              (decrease (fuel-level ?v) (fuel-required ?from ?to))
              (increase (total-fuel-used)(fuel-required ?from ?to))
```

```
(increase (fuel-used ?v) (fuel-required ?from ?to )))))
```

No exemplo, o nome definido para o domínio é *metricVehicle*. *Requirements* define as características de representação usadas no domínio de planejamento modelado. *Types* corresponde a uma lista de constantes, neste caso corresponde a veículo e localização. *Predicates* consiste em uma lista de declarações de predicados, para cada predicado é especificado uma lista de variáveis e seus tipos. Por exemplo, a variável v no predicado *at* diz respeito ao tipo veículo.

A ação *drive* tem como parâmetros o veículo e a sua localização. As pré-condições são: estar o veículo na cidade de partida, ser acessível o caminho da cidade onde está o veículo para a cidade que se deseja ir, ter combustível suficiente para dirigir até a cidade desejada. Como efeitos desta ação, o veículo não está mais na cidade de partida, houve decremento (*decrease*) na quantidade de combustível do veículo e incremento (*increase*) do total de combustível usado.

Dado o domínio, um possível problema para este é o deslocamento de um caminhão e um carro entre as cidades de Paris, Roma, Berlim e Madri, dados os caminhos que ligam as cidades, as unidades de combustiveis que são gastas deslocando-se de uma cidade a outra e a quantidade de combustível existente nos veículos, assim a modelagem deste problema segue:

```
(define (problem metricVehicle1)
 (:domain metricVehicle)
 (:objects
           truck car -vehicle
           Paris Berlin Rome Madrid - location)

(:init
       (at truck Rome)
       (at car Paris)
       (= (fuel-level truck) 100)
       (= (fuel-level car) 100)
       (acessible car Paris Berlin)
       (acessible car Berlin Rome)
```

```
        (acessible car Rome Madrid)
        (acessible truck Rome Paris)
        (acessible truck Rome Berlin)
        (acessible truck Berlin Paris)
        (= (fuel-required Paris Berlin) 40)
        (= (fuel-required Berlin Rome) 30)
        (= (fuel-required Rome Madrid) 50)
        (= (fuel-required Rome Paris) 35)
        (= (fuel-required Rome Berlin) 40)
        (= (fuel-required Berlin Paris) 40)
        (= (total-fuel-used) 0)
        (= (fuel-used car) 0)
        (= (fuel-used truck) 0)
)
(:goal (and (at truck Paris)
            (at car Rome))))
```

O campo *objects* enumera os objetos (constantes) pertencentes ao problema, neste caso, caminhão e carro são objetos do tipo veículo e Paris, Berlim, Roma e Madri são objetos do tipo localização. O estado inicial (*init*) descreve as condições iniciais, como o caminhão estar localizado na cidade de Roma, o carro estar localizado na cidade de Paris, o nível de combustível ser igual a 100 para ambos, serem acessíveis as rotas entre as cidades de Paris e Berlim, Berlim e Roma, Roma e Madri, Roma e Paris, Roma e Berlim e Berlim a Paris. É descrito também no estado inicial a quantidade de combustível que se gasta para ir de uma cidade a outra. Por exemplo, de Paris a Berlim gasta-se 40 unidades de combustível. Ainda, no estado inicial é descrita a quantidade de combustível usada até o momento em cada veículo. Por exemplo o carro estar com 100 unidades de combustível naquele momento.

Em *goal* é descrito o objetivo do problema, neste caso, estar o caminhão localizado em Paris e o carro localizado em Roma.

Utilizando um planejador automático de problemas, um plano válido poderia ser o seguinte: (1) dirigir o carro de Paris a Berlim, (2) dirigir o caminhão de Roma a Paris; e

finalmente (3) dirigir o carro de Berlim até Roma.

3.1.4 O planejador Metric-FF

Algumas linguagens de representação proposicionais, por exemplo STRIPS, não atendem às necessidades quando aplicadas na modelagem de alguns problemas do mundo real. Como por exemplo a execução de ações simultâneas com diferentes durações e recursos contínuos, são complicados ou até mesmo impossíveis de serem representados nessas linguagens [16]. Na tentativa de superar essas limitações algumas linguagens foram criadas, como as linguagens ADL e PDDL, ambas vistas anteriormente.

Também na tentativa de superar essas limitações, algumas técnicas foram desenvolvidas, aprimorando alguns planejadores. O Metric-FF é uma extensão do planejador FF.

O planejador FF é baseado na idéia da heurística que estima as distâncias para o estado o qual deseja-se alcançar, pelo comprimento de uma solução aproximada, esta heurística é obtida através do grafos de planos relaxado.

O grafo de planos relaxado é formado por camadas, cada uma definida pelos nós pertencentes a ela. As camadas pares contêm os nós proposição que representam cada proposição para o problem a ser resolvido. As camadas ímpares contêm os nós ações, que são as instâncias das ações cujas pré-condições são satisfeitas pelas proposições da camada anterior.

No Metric-FF a função heurística foi estendida e introduziu-se restrições e efeitos numéricos.

Em uma tarefa de planejamento numérico, podem existir limitações numéricas (nas pré-condições das ações e nos estados objetivo) e efeitos numéricos (nos efeitos das ações). Limitações e efeitos podem ser de diferentes tipos. Por exemplo, uma limitação pode requerer que o valor de uma variável seja no máximo, ou então no mínimo, igual a uma determinada constante. Os efeitos numéricos podem, incrementar ou decrementar o valor de uma variável [16].

No presente trabalho é efetuada a modelagem de problemas utilizando variáveis numéricas, e proposicionais. Para tal, utiliza-se como planejador o Metric-FF, planejador

que atende às necessidades do presente trabalho.

3.2 Considerações

Neste capítulo apresentou-se alguns conceitos sobre planejamento em Inteligência Artificial.

Além disto, o capítulo trouxe uma breve descrição das linguagens STRIPS, ADL e PDDL. A linguagem PDDL será utilizada na modelagem de problemas de bloqueio em redes de Petri. Assim, o problema de verificação de bloqueios em redes de Petri será modelado como um problema de planejamento; este será detalhado no capítulo 4.

Ao final apresentou-se o planejador Metric-FF o qual será utilizado para geração dos resultados do presente trabalho.

CAPÍTULO 4

VERIFICAÇÃO DE BLOQUEIOS EM REDES DE PETRI USANDO PLANEJAMENTO

Modelar redes de Petri como problemas de planejamento permite utilizar sistemas planejadores para verificar a ocorrência de determinadas situações na rede, como por exemplo, encontrar bloqueios.

Neste capítulo, é apresentado inicialmente uma abordagem sobre alcançabilidade em redes de Petri como problema de planejamento em STRIPS, baseado em Silva [29]. Posteriormente são apresentadas técnicas de verificação de bloqueio em redes de Petri utilizando planejamento proposta por Edelkamp e Jabbar [6]. Ao final do capítulo são apresentadas novas modelagens baseadas nos autores citados, as quais são inéditas.

4.1 Alcançabilidade como planejamento em PDDL

O problema de alcançabilidade em redes de Petri k-limitadas, visto no capítulo 2.2, pode ser modelado como problema de planejamento em PDDL. As redes de Petri, como já mencionado, são representadas por coleções de vetores e matrizes de números naturais. A simulação ou modelagem dessas redes em outras estruturas de representação requer que estas estruturas suportem números naturais e operações aritméticas básicas. No caso de PDDL, a representação de números e operações sobre eles não é direta.

Para a modelagem do problema de alcançabilidade em redes k-limitadas é suficiente representar operações de incremento e decremento sobre números naturais do intervalo $[0, k]$ e as relações "$\geq$" e "$=$".

Estritamente, os números naturais deste intervalo podem ser representados em PDDL pelo conjunto de constantes: $\{n_0, ..., n_k\}$ onde n_i representa o natural i [29].

A relação de ordem "$\geq$" é representada em PDDL pelos literais:

$$geq(n_i, n_j)$$

onde i e $j \in \mathbb{N}$, $i \geq i$, $0 \leq i \leq k$ e $0 \leq i \leq k$

A relação de igualdade "=" é representada em PDDL pelos literais:

$$eq(n_i, n_i)$$

onde $i \in \mathbb{N}$, $0 \leq i \leq k$

A operação de soma é definida por um conjunto de relações de incremento sobre os números naturais representadas pelos literais:

$$inc_s(n_i, n_j)$$

onde s, i e $j \in \mathbb{N}$, $j = (i + s)$, $1 \leq s \leq k$ e $0 \leq 1 \leq (k - s)$. Assim, por exemplo $inc_3(n_1, n_4)$, indica que o incremento de 3 sobre 1 é igual a 4.

A operação de subtração é definida por um conjunto de relações de decremento representadas pelos literais:

$$dec_s(n_i, n_j)$$

onde s, i e $j \in \mathbb{N}$, $j = (i - s)$, $1 \leq s \leq k$ e $0 \leq 1 \leq (k - s)$. Assim por exemplo, o decremento de 2 sobre 3 é igual a 1.

Os lugares da rede de Petri são modelados como conjuntos de proposições que indicam o número de marcas presentes no lugar:

$$place_l(n_i)$$

indica que o lugar l da rede contém i marcas.

As marcações da rede são representadas por conjuntos de proposições $place_l(n_i)$ para todos os lugares l da rede com suas respectivas marcações i.

As transições são modeladas como conjuntos de ações em PDDL. A ação "α_t" que modela a transição t é dada por:

- Parâmetros: $(m_1, m'_1, ..., m_l, m'_l)$. Estas variáveis representam as marcações dos l lugares ligados à transição t, m_i indica a marcação do lugar i antes do disparo de t e m'_i a marcação de i após o disparo.

- Pré-condições: $place_i(m_i) \wedge geq(m_i, Pre(p,t)) \wedge \triangle(m_i, m'_i)$ para todo lugar i. Onde $\triangle(m_i, m'_i)$ é dado por:

  - Se $Pre(p_i,t) > Pos(p_i,t)$: $dec_s(m_i, m'_i)$ sendo $s = Pre(p_i,t) - Pos(p_i,t)$;

  - Se $Pre(p_i,t) < Pos(p_i,t)$: $inc_s(m_i, m'_i)$ sendo $s = Pos(p_i,t) - Pre(p_i,t)$;

  - Se $Pre(p_i,t) = Pos(p_i,t)$: $eq_s(m_i, m'_i)$.

- Efeito: $place_i(m'_i)$ e $\neg\ place_i(m_i)$, para todo lugar $i \in P$, tal que $Pre(p,t) \neq Pos(p,t)$.

Considerando a rede da figura 4.1:

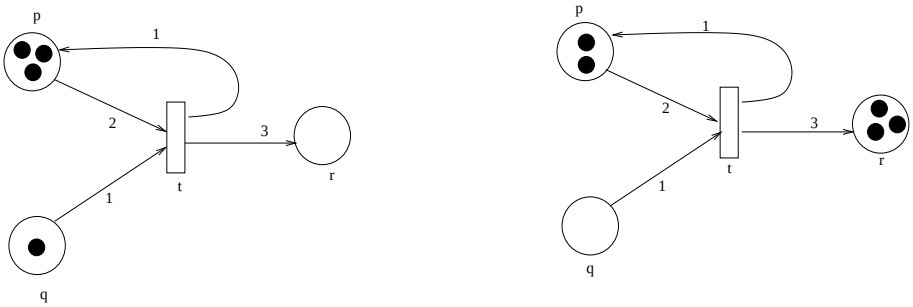

Figura 4.1: Rede de Petri antes (a) e após (b) o disparo da transição t

A ação α_t que modela o disparo da transição t da figura 4.1 é representada em PDDL por:

```
 (:action alpha-t
    :parameters (?x ?y ?z ?w ?a ?b)
    :precondition (and (place-p ?x) (geq ?x n2) (dec1 ?x ?y)
                       (place-q ?z) (geq ?z n1) (dec1 ?z ?w)
                       (place-r ?a) (geq ?a n0) (inc3 ?a ?b)
                  )
    :effect  (and (place-p ?y) (place-q ?w) (place- ?b)
                  (not (place-p ?x)) (not (place-q ?z)) (not (place-r ?a))
             )
)
```

Tem-se então um problema de alcançabilidade dado por uma rede de Petri e duas marcações: a inicial vista na rede da figura 4.1 (a) e a que se deseja atingir ilustrada na figura da rede 4.1 (b). Este problema pode ser modelado como um problema de planejamento:

- A ação "α_t" obtida a partir da transição;
- O estado inicial do problema é dado pela conjunção dos literais que representam a marcação inicial da rede, neste caso o lugar p com 3 marcas, o lugar q com 1 marca e o lugar r sem marcas;
- O objetivo do problema dado pela conjunção dos literais lugar que representam a marcação desejada.

Um plano que resolve o problema apresentado indica que a marcação que se deseja atingir é alcançável a partir da marcação inicial, disponibilizando uma seqüência de disparos das transições da rede que levam até este objetivo. Neste caso, a partir da marcação conforme a figura da rede 4.1 à esquerda através do disparo da transição t obtem-se a nova marcação vista na rede da figura 4.1 à direta.

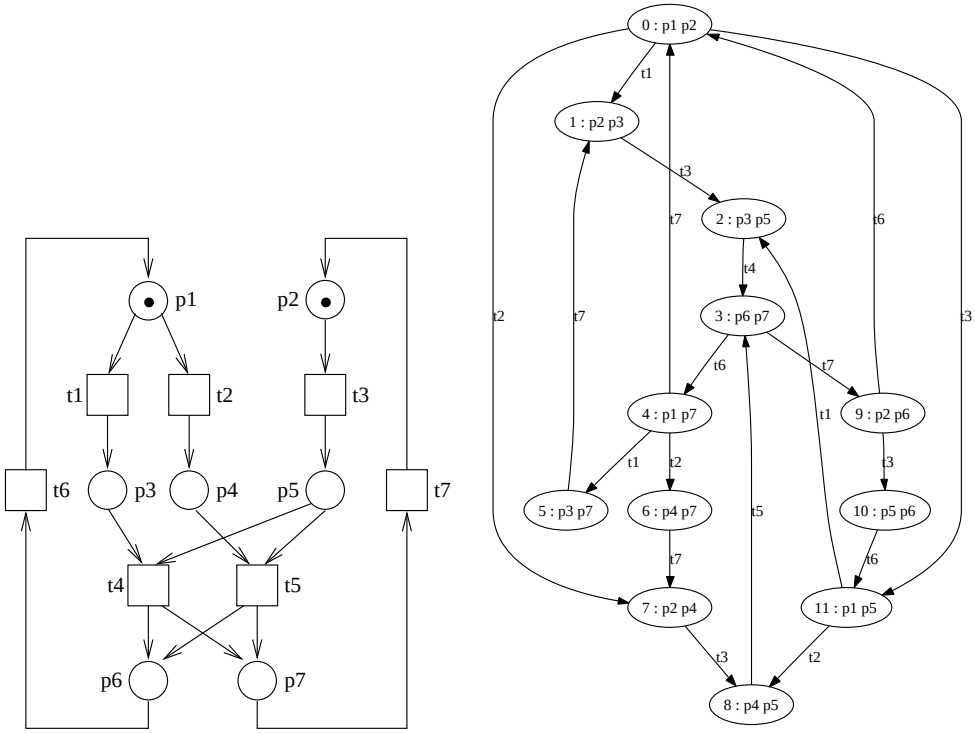

Figura 4.2: (a) Rede de Petri; (b) Grafo de alcançabilidade

Para fins de maior exemplificação, dada a rede 4.2, a modelagem da ação t_1 é demonstrada a seguir, onde os parâmetros da ação são representados por variáveis, que correspondem às quantidades de marcas dos lugares, antes do disparo e após o disparo da transição. As pré-condições trazem os literais verdadeiros para que a ação possa ser aplicada. O efeito corresponde a uma conjunção que pode incluir literais positivos ou negativos, descrevendo as alterações que serão efetuadas após o disparo da transição.

```
(:action fire-t1
    :parameters (?x ?y ?r ?s)
    :precondition (and (place-p1 ?x) (geq ?x n1) (dec1 ?x ?y)
                       (place-p3 ?r) (geq ?r n0) (inc1 ?r ?s)
                  )
    :effect  (and (place-p1 ?y) (place-p3 ?s)
                  (not (place-p1 ?x)) (not (place-p3 ?r))
             )
)
```

A ação *fire-t1* tem como parâmetros as variáveis x, y, r e s que correspondem a

quantidade de marcas do lugar $place_{p1}$ e $place_{p3}$ antes e depois do disparo da transição t_1. Nas pré-condições, a proposição *geq* avalia a quantidade de marcas no lugar $place_{p1}$ e o peso do arco, neste caso, decrementando o número de marcas em x; o lugar $place_{p3}$ que receberá o incremento é modelado em seguida. Os efeitos correspondem a nova marcação dos lugares $place_{p1}$ e $place_{p3}$, bem como a negação da antiga marcação dos lugares $place_{p1}$ e $place_{p3}$. As demais ações que correspondem a modelagem do domínio da rede da figura 4.2 seguem o mesmo princípio que a ação *fire-t1*, que podem ser vistas no `Anexo A - Listagem 1`.

Para esta rede de Petri, um possível problema de alcançabilidade seria a obtenção de uma marca no lugar $place_{p1}$ e uma marca no lugar $place_{p7}$, para isso no arquivo de problema, define-se a descrição do estado inicial da rede, os lugares e a quantidade de marcas em cada lugar, além do objetivo que se deseja alcançar.

```
(:init (place-p1 n1) (place-p2 n1) (place-p3 n0) (place-p4 n0)
       (place-p5 n0) (place-p6 n0) (place-p7 n0)
       (dec1 n1 n0) (inc1 n0 n1)
       (geq n1 n1)  (geq n1 n0) (geq n0 n0))

(:goal (and (place-p1 n1) (place-p7 n1))))
```

Um plano válido para este problema de alcançabilidade em redes de Petri, resultante de um planejador, mostra que uma seqüência de disparos de transições para obtenção do objetivo desejado pode ser: disparos da transição t_3 através da ação *fire-t3* seguido do disparo da transição t_2 através da ação *fire-t2*, seguido do disparo da transição t_5 através da ação *fire-t5* e finalmente seguido do disparo da transição t_6 através da ação *fire-t6*.

Outra seqüência de disparos que leva ao mesmo objetivo pode ser evidenciada, através dos disparos da transição t_1 através da ação *fire-t1* seguido do disparo da transição t_3 através da ação *fire-t3*, seguido do disparo da transição t_4 através da ação *fire-t4* e finalmente seguido do disparo da transição t_6 através da ação *fire-t6*.

Problemas de planejamento em PDDL e de alcançabilidade em redes limitadas são equivalentes em termos de complexidade [29], neste caso, a instância do problema de

planejamento tem, no pior caso, tamanho exponencial em relação ao tamanho da rede. A tradução de cada transição da rede pode gerar k^n ações diferentes, onde n é o número de lugares associados à transição e k é o número máximo de marcas em qualquer lugar da rede.

4.2 Verificação de bloqueios em redes de Petri usando ações em planejamento

Tratar de disparos de uma transição como uma ação aplicada a um conjunto de predicados induzidos por uma estrutura de redes de Petri e sua marcação, possibilita encontrar um plano quando um problema de alcançabilidade é modelado em planejamento.

O objetivo agora, está centrado na busca de possíveis bloqueios na rede, lembrando que um bloqueio em uma rede de Petri ocorre se nenhuma transição puder ser disparada [6].

Edelkamp e Jabbar [6] apresentam modelagens de ações para problemas de verificação de bloqueios em redes de Petri. A seguir são apresentadas as modelagens proposicional, segura e com variáveis numérica tendo como base estes autores.

4.2.1 Modelagem j-binária

A modelagem proposicional que verifica a existência de bloqueios em redes de Petri traduzida para um problema de planejamento proposta por Edelkamp e Jabbar [6] é baseada em uma ação principal chamada *fire-transition* descrita no arquivo de domínio, a qual é ilustrada a seguir:

```
(:action fire-transition
 :parameters (?t - transition)
 :precondition
             (forall (?p - place)
                (or (not (incoming ?p ?t))
                    (exists (?n - number)
                            (and (number-of-tokens ?p ?n)
                                 (is-not-zero ?n))
```

```
                )
            )
    :effect
        (and
            (forall (?p - place ?n1 ?n2 - number)
                    (when
                      (and (incoming ?p ?t) (inc ?n1 ?n2) (number-of-tokens ?p ?n2))
                      (and (not (number-of-tokens ?p ?n2)) (number-of-tokens ?p ?n1))
                     )
              )
              (forall (?p - place ?n1 ?n2 - number)
                      (when
                        (and (outgoing ?t ?p) (inc ?n1 ?n2)  (number-of-tokens ?p ?n1))
                        (and (not (number-of-tokens ?p ?n1)) (number-of-tokens ?p ?n2))
                       )
               )
               (forall (?t2 - transition)
                       (when (blocked ?t2)
                             (not (blocked ?t2)) )
               )
        )
)
```

Nesta ação, uma transição t é passada como parâmetro. As pré-condições da ação dizem que para todos os lugares da rede de Petri, ou, não existe um arco que liga um lugar à uma transição, ou se existe, o número de marcas existente neste lugar não é zero.

Para representar a quantidade de marcas existentes nos lugares da rede um predicado chamado *number-of-tokens* é declarado.

Como efeito desta ação, para todos os lugares da rede, quando existe um arco que liga o lugar à transição, dado o disparo da transição, ocorre um incrementando no número de marcas do lugar posterior a transição, caso contrário, o número de marcas permanece o mesmo em ambos os lugares.

Para o controle das transições que estão sensibilizadas e as que não estão sensibilizadas no estado atual da rede, duas ações são adicionadas a esta modelagem, a primeira é ilustrada a seguir:

```
 (:action do-block
  :parameters (?t - transition ?n - number)
  :precondition
       (exists (?p - place)
         (and (incoming ?p ?t)
              (and (number-of-tokens ?p ?n) (is-zero ?n))
         )
       )
 :effect
        (blocked ?t)
)
```

A ação *do-block* tem como parâmetros uma transição t e um número natural n, sua pré-condição diz que se existe um lugar, e existe um arco que liga este lugar à transição t, mas que a quantidade de marcas nesse lugar é zero, então essa transição está bloqueada, pois não está sensibilizada, impossibilitada de disparar.

Por fim, uma ação que verifica se não existe transições sensibilizadas, é acrescentada no arquivo de domínio:

```
  (:action do-lock
   :parameters ()
   :precondition
           (forall (?t - transition)
                   (blocked ?t))
           :effect
                (deadlock)
  )
```

A ação *do-lock* verifica se existem transições na rede de Petri sensibilizadas, caso não exista nenhuma seu efeito é o bloqueio geral da rede.

O arquivo de domínio completo, pode ser visto no `Anexo A - Listagem 2`, este domínio é fixo, podendo ser utilizado para diversos problemas de verificação de bloqueio em redes de Petri, uma vez que a descrição da rede e sua topologia é dada na descrição do problema.

O arquivo de problema apresentado por esta modelagem, é composto pelos objetos (lugar e transição), pela topologia da rede de Petri, pela descrição do estado inicial da rede e pela descrição do objetivo.

Para exemplificação utilizaremos a rede de Petri que segue:

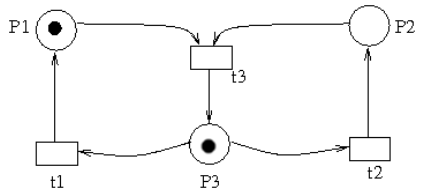

Figura 4.3: Rede de Petri com presença de bloqueios 2

Os objetos correspondem a todas as transições e os lugares que compõem a rede de Petri. O estado inicial é dado pela quantidade de marcas de cada lugar, é declarado os arcos que ligam lugares às transições e os arcos que ligam transições ao lugares; como objetivo temos o bloqueio geral da rede. A seguir podem ser visualizados, o estado inicial e o objetivo descrito no arquivo de problema, o arquivo completo é encontrado no `Anexo A - Listagem 3`:

```
(:init
    (outgoing t1 p1)
    (outgoing t2 p2)
    (outgoing t3 p3)

    (incoming p1 t3)
    (incoming p2 t3)
    (incoming p3 t1)
    (incoming p3 t2)

    (number-of-tokens p1 n1)
    (number-of-tokens p2 n0)
    (number-of-tokens p3 n1)
```

```
)
(:goal (deadlock))
```

Um plano válido para este problema, corresponde ao disparo da transição t_1 que leva a um estado de bloqueio total da rede. Este plano é dado da seguinte maneira: (1) a ação *Fire-Transition* dispara a transição t_1, (2) a ação *do-block* bloqueia a transição t_2 (3) seguida da ação *do-block* que bloqueia a transição t_1, (4) a ação *do-block* bloqueia a transição t_3 e finalmente a ação *do-lock* evidencia o estado de bloqueio total da rede.

Cabe ressaltar que os pesos dos arcos nesta modelagem serão sempre 1, uma vez que não existe nenhum parâmetro que identifique seus valores.

4.2.2 Modelagem j-numérica

Outra modelagem proposta por Edelkamp e Jabbar [6] é a modelagem de verificação de bloqueio em redes de Petri utilizando variáveis numéricas, variáveis estas que armazenam valores numéricos.

O arquivo de domínio é também fixo. Para fins de visualização das diferenças entre as modelagens, em seguida é apresentada a ação *fire-transition*:

```
(:action fire-transition
 :parameters (?t - transition)
 :precondition
        (forall (?p - place)
          (or (not (incoming ?p ?t)) (> (number-of-tokens ?p) 0)))
 :effect
        (and (forall (?p - place)
               (when (incoming ?p ?t)
                     (decrease (number-of-tokens ?p) 1))
             )
             (forall (?p - place)
               (when (outgoing ?t ?p)
                     (increase (number-of-tokens ?p) 1) )
```

```
            )
            (forall (?t2 - transition)
               (when (blocked ?t2)
                     (not (blocked ?t2)) )
            )
        )
)
```

A ação *fire-transition* segue a mesma lógica da modelagem proposicional, porém neste caso, utiliza-se variáveis numéricas, e funções de decremento e incremento, que foram introduzidas através da liguangem PDDL2.1. Assim, para todo lugar da rede de Petri, ou não existe um arco que liga um lugar a uma transição, ou este lugar contém um número de marcas maior que zero.

Como efeito gerado pelo disparo desta transição, decrementa-se o número de marcas nos lugares que antecedem a transição e incrementa-se o número de marcas nos lugares posteriores.

As ações que verificam quais transições não estão sensibilizadas naquele determinado estado da rede e a ação que verifica se todas as transições estão sensibilizadas fazem também parte deste modelo.

O arquivo de domínio completo é encontrado no `Anexo A - Listagem 4`.

O arquivo de descrição do problema é composto pela topologia da rede, pela marcação inicial e descrição do objetivo. A diferença entre os arquivos de problema dessa modelagem e da proposicional, está somente na utilização de variáveis numéricas o arquivo de problema é encontrado no `Anexo A - Listagem 5`.

Um resolvedor automático de problemas, nos garante o mesmo plano descrito na modelagem proposicional, para a rede da figura 4.3.

Para esta modelagem são aceitas somente redes de Petri com peso 1 nos arcos, uma vez que não é evidente a declaração de valores de pesos nos arcos.

4.2.3 Modelagem j-segura

Edelkamp e Jabbar [6], além das modelagens proposicional e com variáveis numéricas, apresentam uma modelagem somente para rede de Petri seguras, nas redes de Petri seguras o número de marcas que um determinado lugar da rede pode possuir não é maior que 1.

A principal ação do domínio é a ação *fire-transition*, que pode ser visualizada a seguir:

```
(:action fire-transition
:parameters (?t - transition)
:precondition
  (forall (?p - place)
    (or (not (incoming ?p ?t))
         (marked ?p)))
:effect
  (and
    (forall (?p - place)
      (when (incoming ?p ?t)
    (not (marked ?p))
       )
     )
    (forall (?p - place)
      (when (outgoing ?t ?p)
    (marked ?p)
      )
    )

   (forall (?t2 - transition)
     (when (blocked ?t2)
           (not (blocked ?t2))
    )
   )
))
```

Como parâmetro desta ação uma variável t que representa as transições é declarada. Sua pré-condição define que para todos os lugares da rede, se existe um arco que liga um

lugar a um transição, e esta transição disparar, o lugar será marcado.

Como efeito a ação marca os lugares de saida t, sensibilizando novas transições da rede.

Além desta ação mais duas ações compõem este modelo de domínio:

```
(:action do-block
 :parameters (?t - transition)
 :precondition
   (exists (?p - place)
     (and (incoming ?p ?t)
                  (not (marked ?p))
             )
    )
 :effect
   (blocked ?t)
)
```

A ação *do-block* tem como parâmetro uma transição t; Se existe um lugar onde não há marcas para sensibilizar uma transição, seu efeito é o bloqueio da transição.

```
(:action do-lock
 :parameters ()
 :precondition
   (forall (?t - transition)
      (blocked ?t)
    )
 :effect
   (deadlock)
)
```

Caso todas as transições da rede de Petri estejam bloqueadas, a ação *do-lock* gera o bloqueio total da rede, uma vez que essa ação verifica se todas as transições da rede de Petri não estão sensibilizadas.

No arquivo de problema, além dos objetos lugares e transições, da topologia da rede e estado final, o diferencial está na declaração da marcação inicial, que é definida por *marked p* para indicar que o lugar p está marcado.

Para exemplificação, a rede da figura 4.4 será utilizada. Uma seqüência de disparos para encontrar o bloqueio desta rede é: (1) a ação *Fire-Transition* dispara para a transição t_5, (2) a ação *do-block* tranca a transição t_5, t_4, t_3, t_2 e t_1 (3) seguida da ação *do-lock* que efetiva o estado de bloqueio total da rede.

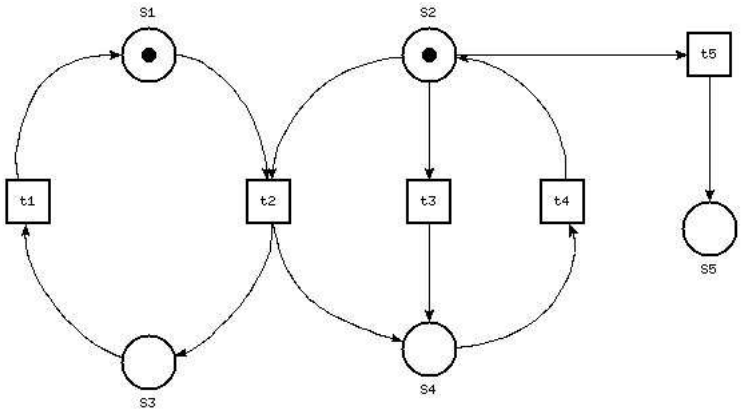

Figura 4.4: Rede de Petri segura

Os arquivos completos de domínio e problema são encontrados no Anexo A - Listagens 6 e 7.

4.3 Novas modelagens

Partindo da afirmação encontrada em Silva [29], de que é possível resolver um problema de alcançabilidade em redes de Petri através de Planejamento em Inteligência Artificial, desenvolveu-se novas abordagens que possibilitam a verificação de bloqueios em redes de Petri.

Além destas, uma nova modelagem foi desenvolvida, baseada na abordagem de Edelkamp e Jabbar [6], visando atender a checagem de bloqueios em redes de Petri contendo

peso nos arcos maior que 1.

Estas novas modelagens serão decritas a seguir.

4.3.1 Modelagem binária-1

Baseado em Silva [29], utilizando as mesmas definições de modelagem de ações em PDDL, desenvolveu-se uma modelagem para verificação de bloqueio em redes de Petri.

Esta é enumerativa, de igual forma à utilizada para solucionar problemas de alcançabilidade por Silva, ou seja, cada transição da rede de Petri corresponde a uma ação do problema de planejamento, tornado assim dinâmico o arquivo de domínio, diferente das propostas por Edelkamp e Jabbar que utilizam um domínio fixo.

No arquivo de problema, são declarados os objetos, o estado inicial da rede de Petri bem como o objetivo que se deseja alcançar. A seguir é descrito o objetivo, neste caso o estado de bloqueio da rede de Petri. Esta foi a primeira modelagem proposicional que o presente trabalho implementou. Para exemplificação utilizou-se a rede de Petri da figura 4.5 vista a seguir.

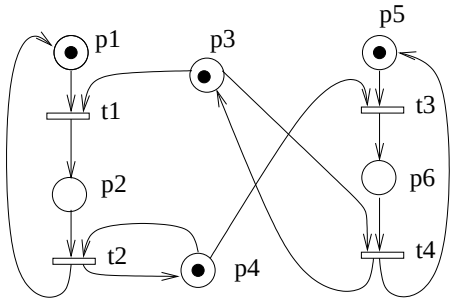

Figura 4.5: Rede de Petri com bloqueios

```
(:goal (and (or (place-p1 n0) (place-p3 n0))
            (or (place-p2 n0) (place-p4 n0))
            (or (place-p4 n0) (place-p5 n0))
            (or (place-p3 n0) (place-p6 n0))
        )
)
```

O estado objetivo verifica, para todas as transições, se os lugares que as prescedem não possuem marcas para sensibilizá-las. Para isso utilizou-se uma conjunção de disjunção. Em outras palavras, para a rede da figura 4.5, o estado objetivo caracterizou-se por analisar se o lugar $place_{p1}$ ou o lugar $place_{p3}$ estavam sem marcas e se o lugar $place_{p2}$ ou lugar $place_{p4}$ estavam sem marcas e o lugar $place_{p4}$ ou o lugar $place_{p5}$ estavam sem marcas e o lugar $place_{p3}$ ou o lugar $place_{p6}$ estavam sem marcas.

Os arquivos completos de domínio e problema desta modelagem encontram-se no `Anexo A - Listagens 8 e 9`.

Um plano válido nos garante que após a disparo da transição t_3 através da ação *fire-t3* seguido do disparo da transição t_1 através da ação *fire-t1* resulta no bloqueio da rede de Petri, por exemplo.

Esta modelagem é direcionada à modelagem de problemas de bloqueios em rede de Petri limitadas. Possibilitando a modelagem de redes de Petri com peso nos arcos maior que 1.

4.3.2 Modelagem binária-2

Esta modelagem é baseada na abordagem de Silva [29] juntamente com a abordagem de Edelkamp e Jabbar [6], proposta pelo presente trabalho.

Este modelo também é enumerativo, ou seja, cada transição da rede de Petri transforma-se em uma ação de planejamento. Utilizando-se da rede de Petri da figura 4.5, para fins de exemplificação, a seguir é mostrada a modelagem de ação $fire - t2$:

```
(:action fire-t2
```

```
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3 ?a4 ?d4)
 :precondition (and (place-p2 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                    (place-p4 ?a2) (geq ?a2 n1) (eq ?a2 ?d2)
                    (place-p1 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
               )
 :effect  (and  (place-p2 ?d1) (place-p1 ?d3)
                (not (place-p2 ?a1)) (not (place-p1 ?a3))
                (not (blocked-t1))  (not (blocked-t2)) (not (blocked-t3))
          )
)
```

Nesta ação, verifica-se que quando existem arcos que contém laço elementar, a marcação naquele determinado lugar, mantém-se a mesma, pois ao ser consumida uma marca através do disparo da transição, outra marca é acrescentada através do mesmo disparo de transição.

A principal diferença entre essa modelagem e a binária-1 está no efeito da ação. Além de gerar um novo estado após o disparo da transição, as transições seguintes serão desbloqueadas.

Nesta abordagem para fins de controle das transições que estão sensibilizadas, duas novas ações são incluidas, as quais seguem:

```
(:action block-t2
 :parameters (?a1 ?a2)
 :precondition  (or (and (place-p2 ?a1) (not (geq ?a1 n1)))
                    (and (place-p4 ?a2) (not (geq ?a2 n1)))
                )
    :effect
          (blocked-t2)
)
```

A ação *block-t2* tem como parâmetros as quantidades de marcas nos lugares $place_{p2}$ e $place_{p4}$, assim a transição terá como efeito o estado de bloqueada, quando os lugares que antecedem a transição não possuirem marcas suficientes para sensibilizá-la.

```
(:action lock
    :parameters ()
    :precondition (and (blocked-t1) (blocked-t2) (blocked-t3) (blocked-t4))
    :effect
         (deadlock)
)
```

A ação *lock* verifica se todas as transições não estão sensibilizada, para resultar no efeito de estado bloqueado da rede.

O arquivo de problema, descreve o estado objetivo de forma reduzida, comparando-se com modelagem binária-1 sugerida pelo presente trabalho, neste caso, o estado objetivo é a existência de bloqueio na rede de Petri:

```
(:goal (deadlock))
```

Através desta modelagem foi possivel encontrar um plano válido, o estado de bloqueio na rede de Petri é encontrado após os disparos das transições t_1 através da ação *fire-t1* seguido do disparo da transição t_3 através da ação *fire-t3*, após isso, as transições t_1, t_2, t_3 e t_4 encontram-se em bloqueio uma vez que a ação *block-t1* verifica que a transição t_1 não está sensibilizada, a ação *block-t2* verifica que a transição t_2 não está sensibilizada, a ação *block-t3* verifica que a transição t_3 não está sensibilizada e por fim a ação *block-t4* verifica que a transição t_4 não está sensibilizada.

Esta abordagem suporta a modelagem de redes de Petri limitadas com pesos nos arcos maiores que 1.

Os arquivos de domínio e problema são encontrados no `Anexo A - Listagens 10 e 11`.

4.3.3 Modelagem numérica

A abordagem que utiliza variáveis numéricas (j-numérica) reproduzida neste trabalho, apresentada por Edelkamp e Jabbar [6] tem a particularidade de modelar redes de Petri

com pesos nos arcos igual a 1, para tanto desenvolveu-se a partir dessa uma nova abordagem, onde a possibilidade de modelar rede de Petri com pesos maiores que 1 em seus arcos é válida.

A principal diferença entre a modelagem j-numérica e esta está, iniciando pela descrição do domínio, na adição de funções que correspondem ao peso do arco que liga um lugar p à uma transição t, e o peso do arco que liga uma transição t à um lugar p.

A ação principal desta abordagem, a ação *fire-transition* é ilustrada a seguir:

```
(:action fire-transition
 :parameters (?t - transition)
 :precondition
   (forall (?p - place)
      (or (not (incoming ?p ?t))
          (>= (number-of-tokens ?p) (fincoming ?p ?t))
      )
   )
:effect
   (and (forall (?p - place)
           (when (incoming ?p ?t)
              (decrease (number-of-tokens ?p) (fincoming ?p ?t))
           )
        )
        (forall (?p - place)
           (when (outgoing ?t ?p)
              (increase (number-of-tokens ?p) (foutgoing ?t ?p))
           )
        )
        (forall (?t2 - transition)
          (when (blocked ?t2)
                (not (blocked ?t2)) ))
   ))
```

A nova ação *fire-transition* tem como parâmetro uma transição t, e como pré-condição que, para todos os lugares da rede de Petri, ou não existe um arco que liga o lugar a

transição, ou o número de marcas tem que ser maior ou igual ao peso do arco que liga este lugar à transição.

Como efeito desta ação, ocorre o incremento e decremento de marcas antes e após o disparo da transição.

Na ação *do-block* é verificado também, se a quantidade de marcas no lugar que antecede a transição satisfaz a necessidade de consumo do arco que a liga à transição. Esta ação pode ser evidênciada a seguir:

```
(:action do-block
 :parameters (?t - transition)
 :precondition
   (exists (?p - place)
     (and (incoming ?p ?t)
        (and (= (number-of-tokens ?p) 0) (< (number-of-tokens ?p) (fincoming ?p ?t)))
        )
    )
 :effect
   (blocked ?t)
)
```

No aquivo de problema, além do já visto através da abordagem j-numérica é necessário a declaração do valores dos pesos dos arcos que ligam transições à lugares e lugares à transições.

Para fins de exemplificação utilizaremos a seguinte rede de Petri:

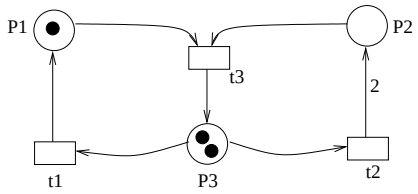

Figura 4.6: Rede de Petri - Pesos nos arcos maior que 1

Os arquivos de domínio e problema completos, podem ser visualizados no `Anexo A - Listagem 12 e 13`.

Um planejador resulta na seguinte sequência para encontrar o bloqueio: (1) inicialmente é identificado que a transição t_3 não está sensibilizada, (2) a transição t_1 dispara através da ação *fire-transition*, (3) a transição t_3 continua bloqueada, (4) a transição t_1 dispara novamente, com isso as transições t_3, t_2 e t_1 não estão sensibilizadas, ocasionando bloqueio geral da rede evidenciado pela (5) ação *do-lock*.

O presente trabalho automatizou as modelagens descritas neste capítulo, com o intuito de comparação entre as modelagens. No próximo capítulo é efetuada a descrição das ferramentas de automatização das modelagens e descreve-se também os resultados obtidos.

4.4 Considerações

O presente capítulo apresentou inicialmente, a modelagem de problemas de alcançabilidade em redes de Petri utilizando Planejamento em Inteligência Artificial, baseado em Silva [29].

Na seqüência apresentou-se as abordagens proposta por Edelkamp e Jabbar [6] as quais transformam problemas de verificação de bloqueios em redes de Petri em problema de Planejamento.

Novas abordagens para o problema de verificação de bloqueios em redes de Petri utilizando planejamento foram propostas. Na modelagem binária-1 para verificar se o estado de bloqueio existe, o estado objetivo analisa se todos os lugares que antecem as transições não contêm marcas suficientes para sensibilizá-la usando uma conjunção de dijunção, esta modelagem é proposicional. A abordagem binária-2 ao disparar uma transição através de uma ação, não bloqueia a transição seguinte a esta transição que disparou, assim para verificar se existe o bloqueio na rede, duas ação são introduzidas, que verificam se existem transições impossibilitadas de disparar.

A abordagem numérica apresentada neste trabalho é uma complementação a abordagem j-numérica, onde funções de pesos de arcos são adicionadas, possibilitando verificar bloqueios em redes de Petri traduzindo para problemas de planejamento, redes de Petri com pesos nos arcos maior que 1.

Através destas novas modelagens é possível modelar o problema de verificação de

bloqueios em redes de Petri como problema de planejamento em redes de Petri com peso no arco maior que 1, principal contribuição deste trabalho à comunidade acadêmica.

O próximo capítulo mostra o resultado da comparação entre as abordagens apresentadas neste capítulo.

CAPÍTULO 5

EXPERIMENTOS

O presente capítulo mostra os resultados obtidos através da aplicação das abordagens apresentadas no capítulo 4, que modelam o problema de verificação de bloqueio em redes de Petri, traduzido em problemas de planejamento em Inteligência Artificial. Para tal, desenvolveu-se a automatização das modelagens, e utilizando-se de um conjunto de problemas efetivou-se a comparação entre estas.

Inicialmente, apresentar-se-á a automatização das modelagens, seguido do conjunto de problemas, dos tempos obtidos através do planejador Metric-FF, e ao final a análise dos resultados.

5.1 Automatização das modelagens

As modelagens apresentadas nos capítulo 4 foram automatizadas, para isso desenvolveu-se ferramentas de tradução de redes de Petri para problemas de verificação de bloqueios em planejamento.

As ferramentas de tradução partem de uma descrição de rede de Petri gerada pela ferramenta *Pep-tool* [13], assim o arquivo de domínio e problema em PDDl com objetivo de verificação de bloqueio é gerado para a dada rede de Petri.

Para a rede de Petri da figura 2.4, o arquivo gerado pela ferramenta *Pep-tool*, contém a declaração de todos os lugares, transições, arcos que ligam, lugares à transições e transições à lugares:

```
PEP
PTNet
FORMAT_N
% --------- Lugares ---------
PL
1"p1"60@220M1m1
2"p2"180@220M0m0
3"p3"60@60M1m1
4"p4"180@60M1m1
5"p5"180@60M1m1
6"p6"180@60M0m0
% --------- Transições ---------
TR
1"t1"180@180
2"t2"120@180
3"t3"120@120
4"t4"120@120
% --------- Declaração de Arcos que ligam Transições à Lugares  ---------
TP
1<2
2<1
2<4
3<6
4<5
4<3
% --------- declaração de Arcos que ligam Lugares á Transições ---------
PT
1>1
2>2
3>1
3>4
4>3
4>2
5>3
6>4
```

Caso o peso do arco que liga um transição à um lugar, ou um lugar à transição tenha peso maior que 1, a declaração de peso no arco pode ser vista a seguir:

```
% --------- Arcos que ligam Lugares -> Transições ---------
PT
1>1
1>4
2>2w2n702
3>3w2n702
3>4
4>5
1>6w2n702
```

Neste caso, o arco que liga o lugar P_2 à transição T_2, por exemplo, tem peso 2, que é denominado depois do caracter w, os caracteres posteriores ao número 2, fazem parte da declaração gráfica da rede de Petri [13].

Para a automatização das modelagens, foi usada a linguagem de programação C++, com paradigma de programação orientado a objeto, em sistema operacional Linux. As ferramentas receberam os nomes de suas abordagens, por exemplo a ferramenta j-numérica reproduz a modelagem j-numérica.

5.2 Problemas modelados

Após efetivada a automatização das modelagens, para fins de testes, utilizou-se um conjunto de problemas coletados por Corbett [4], estes problemas constituem-se de redes de Petri 1-seguras que modelam problemas de comunicação em máquinas de estado [23].

Os problemas utilizados seguem:

- *Dartes Program - DARTES* é o esqueleto de comunicação para um complexo programa em *Ada* com 12 tarefas [18].

- *Dining Philosophers - DP* apesar de não ser um problema muito realístico, contém um bloqueio que não é trivial e é provavelmente o exemplo mais comumente analisado.

- *Elevator* esse programa modela o controle para uma construção com m elevadores, usando tarefas para modelar os mesmos.

- *Hartstone Program - HARTSTONE* é o esqueleto de comunicação de um programa em *Ada*, analisado por [18], onde cada tarefa que começa pára m tarefas em progresso.

- *Distributed Memory Manager - MMGT* é um esqueleto de comunicação de um programa *Ada* que implementa um esquema de gerenciamento de memória [10] com m usuários.

- *User Interface - Q* o esqueleto *Ada* de uma interface de usuário RPC *client/server-based* com 18 tarefas que são usadas por diversas aplicações reais.

- *Sensor Test Program - SENTEST* é o esqueleto de comunicação de um programa *Ada* analisado em [18] que inicia m tarefas para testar sensores.

- *Speed Regulation Program - SPEED* é o esqueleto de comunicação de um programa *Ada* analisado em [18] com 10 tarefas que monitoram e regulam a velocidade de um carro.

Tamanhos diferentes de problemas são usando neste trabalho, por exemplo apresenta-se problemas dos filósofos com seis, oito, dez e doze filósofos.

5.3 Resultados obtidos

Dada a automatização das modelagens e os problemas apresentados, utilizando-se do planejador Metric-FF foi realizado uma bateria de testes, com a finalidade de analisar o desempenho de cada modelagem. A configuração da máquina na qual os testes foram realizados constitui-se de um processador AMD OPTERON de 2.8 GHz, 64 bits, com 32 Gbytes de memória.

A seguir, a tabela de resultados das comparações entre as modelagens apresentadas no capítulo 4 é encontrada:

Tabela 5.1: Resultados

Problema	L	T	J-bin	bin-2	J-num	num	J-seg	mcsmoldes
DARTES(1)	331	257	1.43	*	2.23	2.11	1.26	0.004
DP(6)	36	24	0.00	12.41	0.00	0.00	0.00	0.000
DP(8)	48	32	0.01	35.64	0.01	0.01	0.00	0.000
DP(10)	60	40	0.02	134.54	0.02	0.02	0.01	0.004
DP(12)	72	48	0.02	521.08	0.02	0.03	0.02	0.004
ELEV(1)	63	99	0.18	*	0.9	0.08	0.09	0.001
ELEV(2)	146	299	7.11	*	1.93	1.81	1.50	0.008
HART(25)	127	77	0.08	0.22	0.08	0.08	0.06	0.000
HART(50)	252	152	0.26	0.44	0.51	0.49	0.21	0.000
HART(75)	377	227	1.46	12.72	1.55	2.12	1.90	0.014
HART(100)	502	302	5.86	38.48	3.80	3.59	2.00	0.012
MMGT(3)	122	172	1.49	34.67	0.40	0.38	0.32	0.424
Q(1)	163	194	2.17	*	0.65	0.62	1.56	0.266
SENT(25)	104	55	0.04	0.11	0.03	0.04	0.02	0.000
SENT(50)	179	80	0.11	0.42	0.10	0.09	0.08	0.000
SENT(75)	254	105	0.27	1.15	0.21	0.20	0.17	0.004
SENT(100)	329	130	0.47	2.53	0.38	0.36	0.32	0.000
SPD(1)	33	39	0.01	0.01	0.01	0.01	0.00	0.272

* O planejador resultou em um tempo maior que 600 segundos para estes experimentos.

A tabela anterior mostra os problemas abordados, a quantidade de lugares (L) e a quantidade de transições (T) de cada problema, as modelagens analisadas e o tempo do planejador Metric-FF em segundos para cada modelagem, além do tempo de execução da ferramenta MCSMOLDES que verifica a existência de bloqueios em desdobramento de rede.

Para fins de comparação com outra técnica de verificação de bloqueio, optou-se pela ferramenta MCSMOLDES, proposta por [14], que verifica o estado de bloqueio utilizando o prefixo completo finito, gerado utilizando a ferramenta MOLE no presente trabalho utilizou-se a ferramenta MCSMOLDES na versão 1.7.

5.4 Análise dos resultados

A ferramenta J-binária (J-bin) que reproduz a modelagem proposicional proposta por Edelkamp e Jabbar, obteve uma média de tempo alta comparada com as demais modela-

gens algums problemas como *elevator 2* e *hart 100* levaram 7.11 segundos e 5.86 segundos para que o planejador resultasse um plano válido para encontrar o estado de bloqueio da rede, conforme visto na tabela 5.1. Nota-se que com o crescimento do problema, o tempo necessário para a verificação do bloqueio também aumenta, por exemplo, o problema dos sensores, a medida que cresce o número de lugares e transições, aumenta o tempo de execução do planejador.

A abordagem proposta por Edelkamp e Jabbar utilizando variáveis numéricas (J-num) mostrou tempo melhor em relação a modelagem J-binária os problemas *elevator* e *hart 100* levaram 1.93 segundos e 3.80 segundos para que o planejador resultasse um plano válido para encontrar o estado de bloqueio na rede, como visto na tabela 5.1. Da mesma forma nota-se que a medida que o problema aumenta, dado o crescimento de seu número de lugares e de transições, o tempo de execução do plano gerado pelo planejador automático também aumenta.

A modelagem utilizando variáveis numéricas proposta pelo presente trabalho (numérico), teve um desempenho ainda melhor que a modelagem J-binária e J-numérica. Os tempos gerados pelo planejador ao concluir um plano válido, o qual determina a seqüência que leva o estado de bloqueio da rede de Petri, pode ser visto na tabela 5.1.

A modelagem binária-1 não está descrita na tabela 5.1, esta demonstrou-se eficiente com redes de Petri que continham no máximo quinze lugares, porém com o aumento de quantidade de lugares na rede, a modelagem tornou-se enificiente pois um componente exponencial é evidênciado.

A proposta binária-2 (bin-2) apresentada neste trabalho, por ser enumerativa, consumiu um tempo maior do resolvedor automático de problemas para retornar um plano válido, porém este tempo ainda é considerado satisfatório.

O melhor desempenho de tempo retornado pelo Metric-FF entre as abordagens que traduzem o problema de verificação de bloqueio em redes de Petri utilizando planejamento. Uma vez que as redes desse conjunto são todas seguras, esta modelagem teve melhor desempenho, ocasionando assim numa rápida resposta do planejador.

Comparando-se a técnica de modelagem de verificação de bloqueio em redes de Petri

como problema de planejamento, com a técnica de verificação de bloqueios em desdobramentos de redes, nota-se que o desdobramento teve um desempenho de tempo melhor que as modelagens apresentadas para este conjunto de problemas.

Os resultados obtidos neste trabalho divergiram dos resultados apresentados por Edelkamp e Jabbar, uma vez que para efetuar os teste utilizando a abordagem sobre redes de Petri seguras a máquina utilizada por estes foi um pentium 4 3,2Ghz com 2 Gb de memória; Os resultados foram comparados a abordagem que utilizou um pentium III 450 Mhz.

No presente trabalho todos os teste ocorreram na mesma máquina.

Por não ser conhecido até o presente momento um conjunto de problemas de redes de Petri k-limitados, mas para fins de exemplificação de funcionamento, a seguinte rede de Petri foi modelada. A comparação de tempo resultada do planejador Metric-FF sobre a modelagem numérica e binária-2 obtiveram igual tempo, por ser uma rede que contém poucos lugares e transições a resposta do planejador para encontrar o caminho que leva ao bloqueio foi de 0 segundos para ambas as modelagens. A rede de Petri segue:

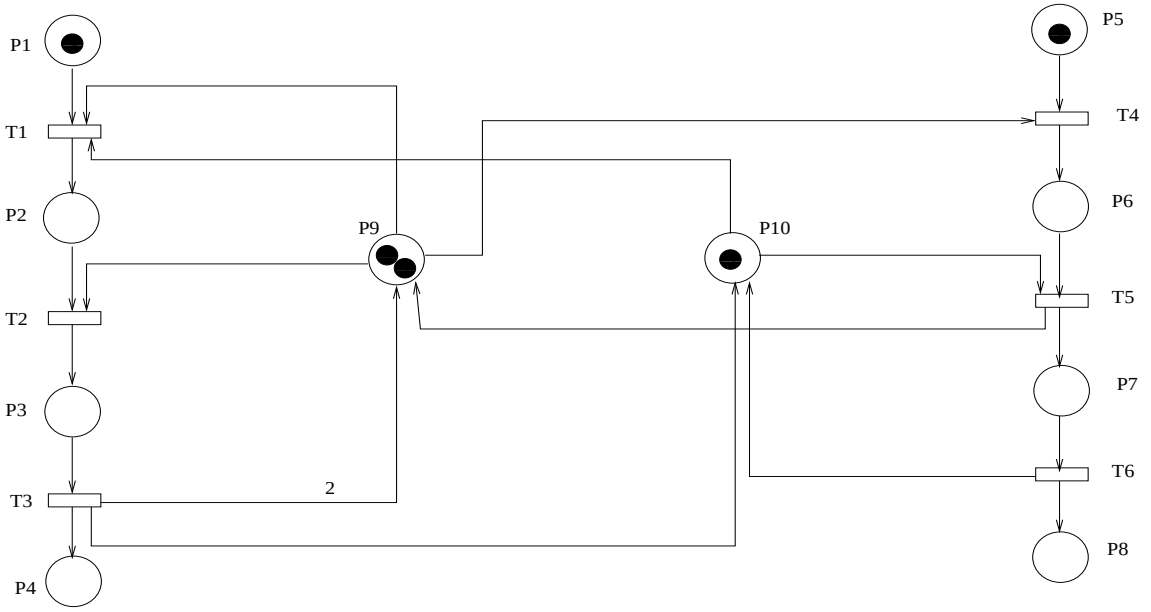

Figura 5.1: Rede de Petri k-limitada

5.5 Considerações

Neste capítulo apresentou-se a automatização das modelagens do capítulo 4, que tem por objetivo verificar a existência de bloqueios em redes de Petri utilizando planejamento.

Através de um conjunto de problemas foram efetuados testes para análise de desempenho entre estas modelagens posteriormente comparou-se esta técnica de verificação de bloqueio em redes de Petri utilizando planejamento com outra técnica que verifica a existência de bloqueios utilizando desdobramento de redes de Petri.

Os resultados obtidos evidenciaram que a técnica usando desdobramento de rede é mais eficiente que a técnica de verificação de bloqueios em redes de Petri como problema de planejamento.

Para fins de teste a modelagem de uma rede de Petri k-limitada com peso nos arcos maior que 1 é apresentada.

CAPÍTULO 6

CONCLUSÃO

O presente trabalho automatizou a tradução de problemas de alcançabilidade em redes de Petri proposta por Silva [29] e, partindo desta abordagem, foram desenvolvidas modelagens para o problema de verificação de bloqueios em redes de Petri, transformando este em um problema de planejamento.

As abordagens de verificação de bloqueio apresentadas por Edelkamp e Jabbar [6] também foram automatizadas. Para fins de teste de desempenho entre estas abordagens foi utilizado um conjunto de problemas de bloqueio em redes de Petri proposto por Corbett [4], como resolvedor automático de problemas foi escolhido o Metric-FF. A modelagem que apresentou o melhor desempenho para este conjunto de problemas foi a J-segura, proposta por Edelkamp e Jabbar [6].

As abordagens apresentadas pelo presente trabalho, baseadas em Silva, obtiveram tempos satisfatórios mediante a este conjunto de problemas. Por ser enumerativa, ou seja, cada transição da rede de Petri é traduzida para uma ação em planejamento, o tempo gasto pelo planejador para resultar um plano válido tornou-se maior.

Ao comparar as técnicas de verificação de bloqueios em redes de Petri usando planejamento, com a técnica proposta por Heljanko [14] que verifica bloqueios em desdobramento de rede, na qual inicialmente utiliza-se a ferramenta MOLE para geração do prefixo completo finito e posteriormente utiliza-se a ferramenta Mcsmoldes para verificação do bloqueio no prefixo completo finito, verificou-se que a abordagem usando desdobramento de rede obteve um melhor desempenho de tempo, divergindo dos resultados expostos por Edelkamp e Jabbar.

Este trabalho apresentou uma mudança na abordagem numérica proposta por Edelkamp e Jabbar, esta mudança consiste na introdução de funções para representar peso nos arcos. Como até o presente momento não é conhecido nenhum conjunto de problemas

com estas características, a modelagem de uma rede de Petri k-limitada com peso nos arcos maior que 1 é apresentada.

Assim, conclui-se pelo presente trabalho que as técnicas de verificação de bloqueios em redes de Petri traduzidas para problemas de planejamento são corretas, e com desempenho de tempo satisfatório. Como principal contribuição são dadas duas modelagens para verificação de bloqueios em redes de Petri que contêm pesos nos arcos maior que 1.

Para trabalhos futuros, sugere-se a modelagem de bloqueios em redes de Petri temporais como problema de planejamento em Inteligência Artificial, uma vez que a linguagem PDDL2.1 disponibiliza a possibilidade de mensuração de tempo de duração de cada ação. Outra sugestão é a transformação do prefixo completo finito em um problema de planejamento. E por fim, análise de modelagens de bloqueios utilizando predicados derivados é sugerida.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Blum, A. and Furst, M. Fast planning through planning graph analysis. *In Proceedings of IJCAI-95*, páginas 1636–1642, 1995.

[2] H. Bonet, B. e Geffner. HSP: Planning as heuristic serach. *Entry at the AIPS-98 Planning Competition*, 1998.

[3] Cardoso, J. e Valette, R. *Redes de Petri*. Editora da UFSC, 1997.

[4] Corbett, J. C. Evaluating deadlock detection methods for concurrent software. *IEEE Transactions on Software Engineering 22*, páginas 161–180, 1996.

[5] Edelkamp, S. e Hoffmann, J. PDDL 2.2: the language for the classical part of IPC-4. *Proceedings of the International Planning Competition - IPC'04*, Whistler, Canada, 2004.

[6] Edelkamp, S. e Jabbar, S. Action planning for directed model checking of petri nets. *Electronic Notes in Theoretical Computer Science*, 149(2):3–18, 2006.

[7] Esparza, J., Römer, S. e Vogler, W. An improvement of McMillan's unfolding algorithm. *Tools and Algorithms for Construction and Analysis of Systems*, páginas 87–106, 1996.

[8] Esparza, J., Römer, S. e Vogler, W. An improvement of McMillan's unfolding algorithm. *Form. Methods Syst. Des.*, 20(3):285–310, 2002.

[9] Fikes, R. e Nilsson, N. STRIPS: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence*, 2(3-4), 1971.

[10] Ford, R. Concurrent algoritms for real-time memory management. *IEEE software*, páginas 10–23, 1988.

[11] Fox, M. e Long, D. PDDL 2.1. *Proceedings of the International Planning Competition - IPC'02*, Toulouse, France, 2002.

[12] Gehani, N. An advanced introduction including reference manual for the ada programming language. *Englewood Cliffs, Nj: Pretice-Hall*, 1984.

[13] Grahlmann, B. The pep tool. *Springer - Lecture Notes in Computer Science*, 1254:440–443, 1997.

[14] Heljanko, K. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. *Fundamenta Informaticae*, páginas 247–268, 1999.

[15] Hoffmann, J. Ff: The fast-forward planning system. *AI Magazine*, 22:57–62, 2001.

[16] Hoffmann, J. The metric-ff planning system: Translating "ignoring delete list"to numeric state variables. *Journal of Artificial Intelligence Research, 20*, páginas 291–341, 2003.

[17] Kautz, H. and Selman, B. Pushing the envelope: Planning, propositional logic, and stochastic search. *In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, páginas 1194–1201, 1996.

[18] Masticola, S. P. e Ryder, B. G. Static infinite wait anomaly detection in polynomial time. *In Proceeding of the 1990 International Conference on Parallel Procedding*, volume II, páginas 78–87, 1990.

[19] McCarthy, J. e Hayes, P. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intellingence*, 4:463–502, 1969.

[20] McDermott, D. PDDL - the planning domain definition language. *AIPS-98 - Planning Competition Comittee*, 1998.

[21] McMillan, K. L. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[22] McMillan, K. L. e Römer, S. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuit. *Proc. 4th Workshop on Computer Aided Verification, LNCS 663*, páginas 164–174, 1992.

[23] Melzer, K. L. Deadlock checking using net unfoldings. *Proc. 4th Workshop on Computer Aided Verification, LNCS 663*, páginas 164–174, 1992.

[24] Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, 1989.

[25] Murata, T., Shenker, B. e Shatz, M. Detection of ada static deadlocks using petri net invariants. *IEEE Trans. Softw. Eng*, páginas 314–326, 1989.

[26] Pyle, I. C. The ada programming language. *Englewood Cliffs, Nj: Pretice-Hall*, 1981.

[27] Reisig, W. Petri nets an introduction. *Monographs on Theoretical Computer Science*, páginas 98–100, 1985.

[28] S. Russell e P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 2002.

[29] Silva, F. *Rede de Planos: Uma proposta para a Solução de Problemas de Planejamento em Inteligência Artificial usando Redes de Petri.* Tese de Doutorado, Centro Federal de Educação Tecnológica do Paraná - CEFET, Curitiba, PR, Brasil, fevereiro de 2005.

[30] Song, Y. e LEE J. Deadlock analysis of petri nets using the transitive matrix. *SICE 2002. Proceedings of the 41st SICE Annual Conference*, 2:689–694, 2002.

ANEXO A

Listagem 1 - Modelagem do domínio da rede de Petri da figura 4.1 - Problema de Alcançabilidade em redes de Petri:

```
(define (domain rede)
(:requirements :strips)
(:predicates
                (place-p1 ?p1)
                (place-p2 ?p2)
                (place-p3 ?p3)
                (place-p4 ?p4)
                (place-p5 ?p5)
                (place-p6 ?p6)
                (place-p7 ?p7)
                (geq ?x ?y)
                (dec1 ?x ?y)
                (inc1 ?x ?y)
)
(:action fire-t1
 :parameters (?x ?y ?r ?s)
 :precondition (and (place-p1 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p3 ?r) (geq ?r n0) (inc1 ?r ?s)
                )
 :effect  (and (place-p1 ?y) (place-p3 ?s)
               (not (place-p1 ?x)) (not (place-p3 ?r))
          )
)
(:action fire-t2
 :parameters (?x ?y ?r ?s)
 :precondition (and (place-p1 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p4 ?r) (geq ?r n0) (inc1 ?r ?s)
                )
 :effect  (and (place-p1 ?y) (place-p4 ?s)
```

```
              (not (place-p1 ?x)) (not (place-p4 ?r))
          )
)
(:action fire-t3
 :parameters (?x ?y ?r ?s)
 :precondition (and (place-p2 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p5 ?r) (geq ?r n0) (inc1 ?r ?s)
               )
 :effect  (and (place-p2 ?y) (place-p5 ?s)
               (not (place-p2 ?x)) (not (place-p5 ?r))
          )
)
(:action fire-t4
 :parameters (?x ?y ?r ?s ?a ?b ?c ?d)
 :precondition (and (place-p3 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p5 ?r) (geq ?r n1) (dec1 ?r ?s)
                    (place-p6 ?a) (geq ?a n0) (inc1 ?a ?b)
                    (place-p7 ?c) (geq ?c n0) (inc1 ?c ?d)
               )
 :effect  (and  (place-p3 ?y) (place-p5 ?s) (place-p6 ?b) (place-p7 ?d)
                (not (place-p3 ?x)) (not (place-p5 ?r))
                (not (place-p6 ?a)) (not (place-p7 ?c))
           )
)
(:action fire-t5
 :parameters (?x ?y ?r ?s ?a ?b ?c ?d)
 :precondition (and (place-p4 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p5 ?r) (geq ?r n1) (dec1 ?r ?s)
                    (place-p6 ?a) (geq ?a n0) (inc1 ?a ?b)
                    (place-p7 ?c) (geq ?c n0) (inc1 ?c ?d)
                )
 :effect  (and  (place-p4 ?y) (place-p5 ?s) (place-p6 ?b) (place-p7 ?d)
                (not (place-p4 ?x)) (not (place-p5 ?r))
                (not (place-p6 ?a)) (not (place-p7 ?c))
           )
```

```
)
(:action fire-t6
 :parameters (?x ?y ?r ?s)
 :precondition (and (place-p6 ?x) (geq ?x n1) (dec1 ?x ?y)
                     (place-p1 ?r) (geq ?r n0) (inc1 ?r ?s)
                )
 :effect  (and (place-p6 ?y) (place-p1 ?s)
               (not (place-p6 ?x)) (not (place-p1 ?r))
           )
)
(:action fire-t7
 :parameters (?x ?y ?r ?s)
 :precondition (and (place-p7 ?x) (geq ?x n1) (dec1 ?x ?y)
                    (place-p2 ?r) (geq ?r n0) (inc1 ?r ?s)
               )
:effect  (and (place-p7 ?y) (place-p2 ?s)
                  (not (place-p7 ?x)) (not (place-p2 ?r))
)))
```

Listagem 2 - Modelagem Proposicional baseada em Edelkamp e Jabbar (J-binária) - Domínio

```
(define (domain rdp)
(:requirements :typing :strips)
(:types place transition)
(:predicates
  (incoming ?p - place ?t - transition)
  (outgoing ?t - transition ?p - place)
  (is-not-zero ?n - number)
  (is-zero ?n - number)
  (inc ?n1 ?n2 - number)
  (number-of-tokens ?p - place ?n - number)
  (blocked ?t - transition)
  (deadlock)
)
```

```
(:action do-block
 :parameters (?t - transition ?n - number)
 :precondition
      (exists (?p - place)
        (and (incoming ?p ?t)
        (and (number-of-tokens ?p ?n) (is-zero ?n))
        )
      )
:effect
   (blocked ?t)
)

(:action do-lock
 :parameters ()
 :precondition
       (forall (?t - transition)
           (blocked ?t))
    :effect
      (deadlock)
)

(:action fire-transition
 :parameters (?t - transition)
 :precondition
        (forall (?p - place)
           (or (not (incoming ?p ?t))
             (exists (?n - number)(and (number-of-tokens ?p ?n)(is-not-zero ?n)))
            )
         )
:effect
      (and
        (forall (?p - place ?n1 ?n2 - number)
           (when
             (and (incoming ?p ?t)(inc ?n1 ?n2)(number-of-tokens ?p ?n2))
             (and (not (number-of-tokens ?p ?n2))(number-of-tokens ?p ?n1))
```

```
            )
           )
         (forall (?p - place ?n1 ?n2 - number)
            (when
             (and (outgoing ?t ?p)(inc ?n1 ?n2)(number-of-tokens ?p ?n1))
             (and (not (number-of-tokens ?p ?n1))(number-of-tokens ?p ?n2))
            )
           )
         (forall (?t2 - transition)
             (when (blocked ?t2) (not (blocked ?t2)))
         )
       )
))
```

Listagem 3 - - Modelagem Proposicional baseada em Edelkamp e Jabbar (J-binária) - Problema, baseado na rede de Petri da figura 4.3

```
(define (problem rdp)
(:domain rdp)
(:objects
         p1 p2 p3 - place
         t1 t2 t3 - transition
         n0 n1 - number
)
(:init
    (inc n0 n1)
    (is-zero n0)
    (is-not-zero n1)

    (outgoing t1 p1)
    (outgoing t2 p2)
    (outgoing t3 p3)

    (incoming p1 t3)
    (incoming p2 t3)
    (incoming p3 t1)
```

```
    (incoming p3 t2)


    (number-of-tokens p1 n1)
    (number-of-tokens p2 n0)
    (number-of-tokens p3 n1)
)
(:goal (deadlock)
))
```

Listagem 4 - Modelagem com variáveis numéricas baseada em Edelkamp e Jabbar (J-numérica) - Domínio

```
(define (domain rdp)
(:requirements :typing :strips)
(:types transition place - object)
(:predicates (incoming ?p - place ?t - transition)
             (outgoing ?t - transition ?p - place)
     (blocked ?t - transition)
     (deadlock)
)


(:functions
   (number-of-tokens ?p - place))


(:action do-block
 :parameters (?t - transition)
 :precondition
          (exists (?p - place)
   (and (incoming ?p ?t)
              (= (number-of-tokens ?p) 0))
           )
 :effect
     (blocked ?t)
)


(:action do-lock
```

```
 :parameters ()
 :precondition
       (forall (?t - transition)
            (blocked ?t))
 :effect
     (deadlock)
)

(:action fire-transition
 :parameters (?t - transition)
 :precondition
       (forall (?p - place)
          (or (not (incoming ?p ?t))
          (> (number-of-tokens ?p) 0)))
 :effect
     (and (forall (?p - place)
            (when (incoming ?p ?t)
              (decrease (number-of-tokens ?p) 1) ))
           (forall (?p - place)
             (when (outgoing ?t ?p)
                (increase (number-of-tokens ?p) 1) ))
           (forall (?t - transition)
              (when (blocked ?t) (not (blocked ?t)) ))
      )

)
```

Listagem 5 - Modelagem com variáveis numéricas baseada em Edelkamp e Jabbar (J-numérica) - Problema, baseado na rede de Petri da figura 4.3

```
(define (problem rdp)
(:domain rdp)
(:objects
         p1 p2 p3 - place
         t1 t2 t3 - transition
         n0 n1 - number
```

```
)
(:init
    (inc n0 n1)
    (is-zero n0)
    (is-not-zero n1)

    (outgoing t1 p1)
    (outgoing t2 p2)
    (outgoing t3 p3)

    (incoming p1 t3)
    (incoming p2 t3)
    (incoming p3 t1)
    (incoming p3 t2)

    (number-of-tokens p1 n1)
    (number-of-tokens p2 n0)
    (number-of-tokens p3 n1)

)
(:goal (deadlock)))
```

Listagem 6 - Modelagem J-segura baseado em Edelkamp e Jabbar - Domínio

```
(define (domain rdp)
(:requirements :typing :strips)
(:types transition place - object)
(:predicates (incoming ?p - place ?t - transition)
             (outgoing ?t - transition ?p - place)
     (blocked ?t - transition)
     (marked ?p - place)
     (deadlock)
)
(:action do-block
 :parameters (?t - transition)
 :precondition
```

```
   (exists (?p - place)
(and (incoming ?p ?t)
            (not (marked ?p))
         )
    )
 :effect
   (blocked ?t)
)
(:action do-lock
 :parameters ()
 :precondition
   (forall (?t - transition)
      (blocked ?t)
    )
 :effect
   (deadlock)
)
(:action fire-transition
 :parameters (?t - transition)
 :precondition
  (forall (?p - place)
    (or (not (incoming ?p ?t))
         (marked ?p)))
 :effect
  (and
    (forall (?p - place)
      (when (incoming ?p ?t)
    (not (marked ?p))
       )
     )

    (forall (?p - place)
      (when (outgoing ?t ?p)
    (marked ?p)
      )
```

```
    )
   (forall (?t2 - transition)
     (when (blocked ?t2)
            (not (blocked ?t2))
    )
   )

)))
```

Listagem 7 - Modelagem J-segura baseada em Edelkamp e Jabbar - Problema, baseado na rede de Petri da figura 4.4

```
(define (problem rdp)
(:domain rdp)
(:objects
         p1 p2 p3 p4 p5 - place
         t1 t2 t3 t4 t5 - transition
 )
(:init

    (outgoing t1 p1)
    (outgoing t2 p3)
    (outgoing t2 p4)
    (outgoing t3 p4)
    (outgoing t4 p2)
    (outgoing t5 p5)

    (incoming p1 t2)
    (incoming p2 t2)
    (incoming p2 t3)
    (incoming p2 t5)
    (incoming p3 t1)
    (incoming p4 t4)

    (marked p1)
    (marked p2)
```

```
)
(:goal (deadlock)))
```

Listagem 8 - Modelagem binária-1 - Domínio, baseado na rede de Petri da figura 4.5

```
(define (domain rede)
(:requirements :strips)
(:predicates
                (place-p1 ?p1)
                (place-p2 ?p2)
                (place-p3 ?p3)
                (place-p4 ?p4)
                (place-p5 ?p5)
                (place-p6 ?p6)
                (geq ?x ?y)
                (dec1 ?a ?d)
                (inc1 ?a ?d)
                (eq ?a ?d)
)
(:action fire-t1
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3)
 :precondition (and (place-p1 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                    (place-p3 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                    (place-p2 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
             )
:effect  (and  (place-p1 ?d1) (place-p3 ?d2) (place-p2 ?d3)
                (not (place-p1 ?a1)) (not (place-p3 ?a2)) (not (place-p2 ?a3))
         )
)
(:action fire-t2
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3 ?a4 ?d4)
 :precondition (and (place-p2 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                     (place-p4 ?a2) (geq ?a2 n1) (eq ?a2 ?d2)
                     (place-p1 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
                                   )
```

```
:effect  (and  (place-p2 ?d1) (place-p1 ?d3)
               (not (place-p2 ?a1)) (not (place-p1 ?a3))
         )
)
(:action fire-t3
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3)
 :precondition (and (place-p4 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                    (place-p5 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                    (place-p6 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
               )
 :effect  (and  (place-p4 ?d1) (place-p5 ?d2) (place-p6 ?d3)
                (not (place-p4 ?a1)) (not (place-p5 ?a2)) (not (place-p6 ?a3))
         )
)
(:action fire-t4
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3 ?a4 ?d4)
 :precondition (and (place-p3 ?a1) (geq ?a1 n1) (eq ?a1 ?d1)
                    (place-p6 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                    (place-p5 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
               )
:effect  (and  (place-p6 ?d2) (place-p5 ?d3)
               (not (place-p6 ?a2)) (not (place-p5 ?a3))
         )
))
```

Listagem 9 - Modelagem binária-1 - Problema, baseado na rede de Petri da figura 4.5

```
(define (problem rede)
(:domain rede)
(:objects n0 n1)

(:init (place-p1 n1) (place-p2 n0) (place-p3 n1) (place-p4 n1)
       (place-p5 n1) (place-p6 n0)
       (dec1 n1 n0)
       (inc1 n0 n1)
       (geq n1 n1)
```

```
       (geq n1 n0)
       (geq n0 n0)
)
(:goal (and (or (place-p1 n0) (place-p3 n0))
            (or (place-p2 n0) (place-p4 n0))
            (or (place-p4 n0) (place-p5 n0))
            (or (place-p3 n0) (place-p6 n0))
)))
```

Listagem 10 - Modelagem binária-2 - Domínio, baseado na rede de Petri da figura 4.5

```
(define (domain rede)
(:requirements :strips)
(:predicates
                (place-p1 ?p1)
                (place-p2 ?p2)
                (place-p3 ?p3)
                (place-p4 ?p4)
                (place-p5 ?p5)
                (place-p6 ?p6)
                (geq ?x ?y)
                (dec1 ?a ?d)
                (inc1 ?a ?d)
                (eq ?a ?d)
                (blocked-t1)
                (blocked-t2)
                (blocked-t3)
                (blocked-t4)
                (deadlock)
)
(:action fire-t1
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3)
 :precondition (and (place-p1 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                    (place-p3 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                    (place-p2 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
               )
```

```
 :effect  (and  (place-p1 ?d1) (place-p3 ?d2) (place-p2 ?d3)
                (not (place-p1 ?a1)) (not (place-p3 ?a2)) (not (place-p2 ?a3))
(not (blocked-t2))
             )
)
(:action fire-t2
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3 ?a4 ?d4)
 :precondition (and (place-p2 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                    (place-p4 ?a2) (geq ?a2 n1) (eq ?a2 ?d2)
                    (place-p1 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
                  )
:effect  (and  (place-p2 ?d1) (place-p1 ?d3)
                (not (place-p2 ?a1)) (not (place-p1 ?a3))
(not (blocked-t1))
(not (blocked-t2))
(not (blocked-t3))
          )
)
(:action fire-t3
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3)
 :precondition (and (place-p4 ?a1) (geq ?a1 n1) (dec1 ?a1 ?d1)
                     (place-p5 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                     (place-p6 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
                )
:effect  (and  (place-p4 ?d1) (place-p5 ?d2) (place-p6 ?d3)
                   (not (place-p4 ?a1)) (not (place-p5 ?a2)) (not (place-p6 ?a3))
   (not (blocked-t4))
             )
)
(:action fire-t4
 :parameters (?a1 ?d1 ?a2 ?d2 ?a3 ?d3 ?a4 ?d4)
 :precondition (and (place-p3 ?a1) (geq ?a1 n1) (eq ?a1 ?d1)
                      (place-p6 ?a2) (geq ?a2 n1) (dec1 ?a2 ?d2)
                      (place-p5 ?a3) (geq ?a3 n0) (inc1 ?a3 ?d3)
                                   )
```

```
:effect  (and  (place-p6 ?d2) (place-p5 ?d3)
                  (not (place-p6 ?a2)) (not (place-p5 ?a3))
  (not (blocked-t1))
  (not (blocked-t3))
  (not (blocked-t4))
          )
)
(:action block-t1
 :parameters ( ?a1 ?a2)
 :precondition  (or (and (place-p1 ?a1) (not (geq ?a1 n1)))
            (and (place-p3 ?a2) (not (geq ?a2 n1)))
)
:effect
  (blocked-t1)
)
(:action block-t2
 :parameters ( ?a1 ?a2)
 :precondition  (or (and (place-p2 ?a1) (not (geq ?a1 n1)))
 (and (place-p4 ?a2) (not (geq ?a2 n1)))
)
 :effect
(blocked-t2)
)
(:action block-t3
 :parameters ( ?a1 ?a2)
 :precondition  (or (and (place-p4 ?a1) (not (geq ?a1 n1)))
 (and (place-p5 ?a2) (not (geq ?a2 n1)))
)
 :effect
(blocked-t3)
)

(:action block-t4
 :parameters ( ?a1 ?a2)
 :precondition  (or (and (place-p3 ?a1) (not (geq ?a1 n1)))
```

```
 (and (place-p6 ?a2) (not (geq ?a2 n1)))
)
 :effect
(blocked-t4)
)
(:action lock
 :parameters ()
 :precondition (and (blocked-t1) (blocked-t2) (blocked-t3) (blocked-t4))
 :effect
(deadlock)
))
```

Listagem 11 - Modelagem binária-2 - Problema, baseado na rede de Petri da figura 4.5

```
(define (problem rede)
(:domain rede)
(:objects n0 n1)
(:init (place-p1 n1) (place-p2 n0) (place-p3 n1) (place-p4 n1)
       (place-p5 n1) (place-p6 n0)
       (dec1 n1 n0)
       (inc1 n0 n1)
       (eq n1 n1)
       (eq n0 n0)
       (geq n1 n1)
       (geq n1 n0)
       (geq n0 n0)
)
(:goal (deadlock)
))
```

Listagem 12 - Nova modelagem utilizando variáveis numéricas (numérica) - Domínio

```
(define (domain rdp)
(:requirements :strips )
(:types place transition)
```

```
(:predicates
            (incoming ?p - place ?t - transition)
            (outgoing ?t - transition ?p - place)
            (blocked ?t - transition)
            (deadlock))

(:functions
            (number-of-tokens ?p - place)
            (fincoming ?p - place ?t - transition)
            (foutgoing ?t - transition ?p - place)
)

(:action do-block
 :parameters (?t - transition)
 :precondition
   (exists (?p - place)
(and (incoming ?p ?t)
         (and (= (number-of-tokens ?p) 0) (< (number-of-tokens ?p) (fincoming ?p ?t) ) )
         )
    )
 :effect
   (blocked ?t)
)

(:action do-lock
 :parameters ()
 :precondition
   (forall (?t - transition)
      (blocked ?t))
 :effect
   (deadlock)
)
(:action fire-transition
 :parameters (?t - transition)
```

```
 :precondition
   (forall (?p - place)
      (or (not (incoming ?p ?t))
          (>= (number-of-tokens ?p) (fincoming ?p ?t))
      )
   )
:effect
   (and (forall (?p - place)
           (when (incoming ?p ?t)
              (decrease (number-of-tokens ?p) (fincoming ?p ?t))
           )
        )
        (forall (?p - place)
           (when (outgoing ?t ?p)
              (increase (number-of-tokens ?p) (foutgoing ?t ?p))
           )
        )
        (forall (?t2 - transition)
          (when (blocked ?t2) (not (blocked ?t2)) ))
   )))
```

Listagem 13 - Nova modelagem numérica - Problema, baseado na rede de Petri da figura 4.6

```
(define (problem rdp)
(:domain rdp)
(:objects
         p1 p2 p3 - place
         t1 t2 t3 - transition
)
(:init

    (outgoing t1 p1)
    (outgoing t2 p2)
    (outgoing t3 p3)
```

```
    (incoming p1 t3)
    (incoming p2 t3)
    (incoming p3 t1)
    (incoming p3 t2)

   (= (foutgoing t1 p1) 1)
   (= (foutgoing t2 p2) 2)
   (= (foutgoing t3 p3) 1)

   (= (fincoming p1 t3) 1)
   (= (fincoming p2 t3) 1)
   (= (fincoming p3 t1) 1)
   (= (fincoming p3 t2) 1)

   (=(number-of-tokens p1 ) 1)
   (=(number-of-tokens p2 ) 0 )
   (=(number-of-tokens p3 ) 2)
)
(:goal (deadlock)))
```

.