RAZER ANTHOM NIZER ROJAS MONTAÑO

APLICAÇÃO DE FÓRMULAS NÃO-CLAUSAIS EM PLANEJAMENTO COM 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

2006

RAZER ANTHOM NIZER ROJAS MONTAÑO

APLICAÇÃO DE FÓRMULAS NÃO-CLAUSAIS EM PLANEJAMENTO COM 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

2006

RAZER ANTHOM NIZER ROJAS MONTAÑO

APLICAÇÃO DE FÓRMULAS NÃO-CLAUSAIS EM PLANEJAMENTO COM REDES DE PETRI

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientador: Prof. Dr. Fabiano Silva
Departamento de Informática, UFPR

Prof. Dr. Guilherme Bittencourt
Departamento de Automação e Sistemas, UFSC

Prof. Dr. Luiz Künzle
Departamento de Informática, UFPR

Curitiba, 14 de setembro de 2006

AGRADECIMENTOS

Agradeço inicialmente ao meu orientador Fabiano pelo apoio, condução dos trabalhos, discussões e amizade; ao Marcos Castilho pela orientação e mais uma vez na confiança do término deste trabalho. Seu apoio, aceitação e ajuda foram decisivos para o meu trabalho, sem os quais não conseguiria terminá-lo.

Agradeço aos amigos da Cinq, principalmente ao Jayme, Édson e Aldir, que sempre me receberam quando precisei e me apoiaram quando quis seguir em frente nesta etapa da minha carreira; ao Sérgio e ao Adil pela amizade, apoio e compreensão.

Agradeço à minha mãe pelos sacrifícios enfrentados durante sua vida para me trazer até aqui; à minha família em geral por me proporcionar o crescimento necessário para seguir em frente.

Agradeço e dedico este trabalho à Alexandra e à Brunna, que se sacrificaram tantas vezes, principalmente na minha ausência em várias oportunidades.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIAÇÕES

CNF *Conjuntive Normal Form.* É uma forma normal baseada em cláusulas, onde a fórmula deve ser formada por uma conjunção de disjunções e as negações incidem somente sobre variáveis proposicionais.

d-DNNF É uma forma normal baseada na NNF onde a fórmula deve conter as propriedades de determinismo e decomponibilidade.

NNF *Negation Normal Form.* Uma forma normal do cálculo proposicional onde os únicos conectivos permitidos são a conjunção e disjunção, e as negações incidem somente sobre variáveis proposicionais.

RESUMO

Redes de Petri (RdP) podem ser usadas para modelar um problema de planejamento de maneira similar ao uso do grafo de planos. Porém, apesar da representação em RdP apresentar características interessantes para a modelagem, encontrar o plano efetivamente tem sido um problema difícil de se tratar computacionalmente. Isto requer normalmente resolver o problema de Alcançabilidade em Redes de Petri (PSPACE-completo) com milhares/milhões de nodos.

Neste trabalho investiga-se representações e algoritmos para SAT para resolução dos conflitos na rede que levariam à solução do problema de alcançabilidade. A obtenção da instância SAT é feita a partir da RdP e leva ao estudo de formas normais não-clausais e algoritmos eficientes para as fórmulas obtidas.

Palavras chave: Planejamento, Redes de Petri, Alcançabilidade, SAT, *Negation Normal Form*, Tableau KE, d-DNNF, FNNF.

ABSTRACT

Petri Nets (RdP) can be used to model planning problems in similar way to graph of plans. However, despite the representation in RdP presenting interesting characteristics for the modeling, effectively to find the plan has been a hard computational problem. This normally requires to decide the problem of Reachability in Petri Nets (PSPACE-complete) with thousands/millions of nodes.

In this work, we investigate representations and algorithms for SAT for resolution of the conflicts in the net that would lead to the solution of the reachability problem. The SAT instance obtained is generated from the RdP and takes to the study of non-clausal normal forms and efficient algorithms for them.

Key-words: Planning, Petri Nets, Reachability, SAT, Negation Normal Form, Tableau KE, d-DNNF, FNNF.

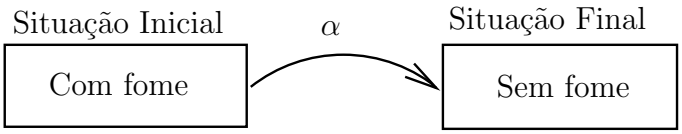
CAPÍTULO 1

INTRODUÇÃO

Na área de raciocínio sobre ações, o *problema de planejamento* define-se por, partindo-se de uma situação inicial conhecida, obter uma determinada situação objetivo, a partir da aplicação de ações pré-definidas. O problema está em descobrir quais ações e em que ordem elas devem ser aplicadas.

No exemplo da Figura 1.1 o problema é o de "saciar a fome". Tem-se uma situação inicial, onde uma pessoa está com fome e a situação final onde esta pessoa está sem fome ou satisfeita. Para se chegar ao objetivo, pode-se agir como indicado:

- Comprando ingredientes;
- Preparando comida;
- Depois da comida pronta, comendo-a.



1. Comprar ingredientes
2. Preparar comida
3. Comer

Figura 1.1: Planejamento

Segundo Weld [36], um algoritmo que resolve um problema de planejamento deve ter como entradas:

- Uma descrição das entidades do mundo e da situação atual;
- Uma descrição dos objetivos;
- Uma descrição das ações que podem ocorrer.

Este algoritmo terá como saída uma seqüência de ações que, quando executadas, levam o mundo do estado atual até o estado objetivo.

O problema de planejamento foi inicialmente modelado computacionalmente por Fikes e Nilson [14], que propuseram uma representação formal simples e um processo de busca, conhecido como sistema STRIPS. Apesar da simplicidade, foi provado que o problema de planejamento em STRIPS é PSPACE-Completo [2]. O algoritmo de Fikes e Nilson não obteve sucesso em resolver mesm problemas bastante simples de planejamento, mas a técnica de modelagem é usada até hoje.

O primeiro algoritmo que resolveu, com desempenho satisfatório, problemas maiores foi o SATPLAN [18] [20], através de uma tradução para satisfatibilidade[31] (SAT) e uso de resolvedores SAT rápidos.

Em 1995, Blum e Furst apresentaram o GRAPHPLAN [1] que modela o problema de planejamento em termos de um grafo conhecido como "Grafo de Planos" e obtiveram sucesso na resolução de uma quantidade maior de problemas. Kautz e Selman [19] mostraram como obter uma instância SAT a partir do grafo de planos. A instância obtida é muito menor do que aquela conseguida pelo SATPLAN.

Estes três algoritmos foram usados como base para uma série de pesquisas que levou à publicação de várias técnicas de modelagem e de algoritmos razoavelmente eficientes para tratar o problema de planejamento em STRIPS, tal como o FF [17]. Exemplos de técnicas de modelagem também são: programação inteira citebockmayr98mixed[35], busca heurística [1], programação por restrições [12] , etc.

Contudo, não se conhece hoje algoritmo eficiente para tratar problemas de planejamento não-clássico, isto é, quando se tem restrições de tempo, não-determinismo, incerteza, recursos, etc. Em suma, o campo de pesquisa ainda é vasto.

Uma representação que naturalmente suporta tempo e recursos são as Redes de Petri (RdP). RdP são normalmente utilizadas para a modelagem de sistemas de produção, mas podem ser utilizadas de maneira bastante similar ao grafo de planos, em termos de representação, conforme proposta de Silva [32]. Neste trabalho, Silva mostra que se pode transformar um grafo de planos em uma RdP.

Com RdP se consegue uma abstração adequada para representação gráfica de sistemas baseados em estados, eventos, pré-condições e pós-condições [25]. Em especial, o estudo da alcançabilidade em redes de Petri resume-se em saber se uma determinada marcação (ou submarcação) pode ser alcançada, a partir de uma marcação inicial.

A modelagem mostrada em [32] foi uma proposta para solução de planejamento utilizando-se as estruturas e algoritmos de RdP. Este modelo consegue mapear para uma RdP toda a estrutura dinâmica que um problema baseado em ações necessita, podendo usufruir do formalismo e técnicas de análise das RdP. Esta solução é conhecida como PETRIPLAN.

O relacionamento estreito entre planejamento e alcançabilidade também foi verificado em [32], possibilitando que um grafo de planos possa ser transformado facilmente em uma RdP ordinária, limitada. A partir disso, o problema de planejamento se torna um problema de alcançabilidade em Redes de Petri, ganhando muito em termos de representação, visto que as meta-informações (exclusões mútuas do grafo de planos) são modeladas pela própria rede, como *conflitos*.

As RdPs possuem um poder de representação maior que o visto em outras soluções propostas, tais como fórmulas em CNF e grafos. A expressividade é tal que se consegue mapear, através da dinâmica da própria rede, as pré-condições e efeitos da execução de uma determinada ação, sem o uso de estruturas auxiliares.

Contudo, resolver o problema de planejamento nesta modelagem é resolver o problema de alcançabilidade que em geral é EXPSPACE [13]. Não se conhece ainda algoritmo eficiente para este problema, em especial se for considerado o tamanho das redes que modelam um problema de planejamento, que normalmente são matrizes com milhares/milhões de nodos, tornando ineficiente ou impraticáveis qualquer solução computacional efetiva.

Através da análise de uma RdP gerada a partir de um problema são obtidas as informações sobre todos os conflitos que incidem sobre a execução de ações. Por conflito, entende-se um ponto de escolha, onde duas ações são mutuamente exclusivas, ou uma ação é executada ou a outra, nunca ambas. A escolha sobre qual ação executar pode levar à solução do problema, ou a um caminho infrutífero, indicando que a escolha feita deve

ser revista. Caso se obtenha uma RdP sem conflitos, então não há escolhas a serem feitas, portanto a solução (ou sua inexistência) é trivial e rápida de ser determinada, bastando uma varredura na RdP. Em RdP sem conflitos e acíclicas e o problema de alcançabilidade é polinomial [13].

O presente trabalho tem como objetivo propor um método para resolver o problema de planejamento modelado como uma RdP, baseado na eliminação de conflitos desta rede. Para isto, a partir da RdP faz-se a geração de uma fórmula lógica, na NNF, que representa a relação existente entre os conflitos para se alcançar os objetivos. Esta relação é tal que, encontrar uma valoração que satisfaça a fórmula gerada equivale a fazer todas as escolhas da RdP, representadas pelos conflitos, de tal forma que os objetivos sejam alcançados.

O que se propõe aqui é a transformação desta fórmula para outra forma normal, conhecida como FNNF, para a qual o procedimento SAT é uma simples varredura na fórmula. Assim, o único ponto com tempo de processamento alto é a transformação de uma fórmula lógica na NNF para FNNF.

No capítulo 2 apresenta-se a fundamentação teórica do trabalho, que trata de RdPs e do algoritmo PETRIPLAN de Silva [32]. O capítulo 3 mostra as representações proposicionais para o problema SAT, tanto para fórmulas clausais como para não-clausais. Para as formas não-clausais são apresentados algoritmos de transformação e de enumeração de modelos. A seguir, no capítulo 4, tem-se a proposta de uma solução para o problema de planejamento, usando-se RdP e um resolvedor SAT para fórmulas na FNNF. Também são mostrados os algoritmos usados na proposta apresentada. Finalmente no capítulo 6 são apresentadas as implementações de vários algoritmos para tratamento de formas não-clausais e as considerações finais do trabalho.

CAPÍTULO 2

PETRIPLAN

Neste capítulo são mostradas as RdP e o problema de alcançabilidade. São feitas correspondências entre esses problemas e os de planejamento. Em particular é mostrado o método proposto por Silva [32], usando o algoritmo PETRIPLAN.

2.1 Redes de Petri

As Redes de Petri (RdP) são um modelo matemático de especificação formal de sistemas discretos, possibilitando a representação e análise matemática de problemas baseados em estados e eventos, permitindo a verificação de propriedades, por exemplo conflitos, paralelismo e concorrência, e corretude dos sistemas. Trata-se de um modelo baseado em estados e eventos, onde os eventos possuem pré-condições e efeitos, que por sua vez, serão pré-condições de outros eventos.

2.1.1 Definições

Uma RdP é composta pelos seguintes elementos [25]:

- **Lugares:** representam os componentes passivos do sistema, como situações do problema modelado. Nele, são depositadas marcas que são acrescidas ou removidas pelo disparo de transições. Um lugar que influencia o disparo de uma transição é uma pré-condição. O lugar que é influenciado pelo disparo de uma transição é uma pós-condição;

- **Transições:** representam os componentes ativos do sistema, como ações. Uma trasição está habilitada (sensibilizada) quando todas as suas pré-condições forem atendidas, isto é, quando houverem marcas suficientes em todos os lugares que são pré-condições desta transição, e desabilitada se pelo menos uma das pré-condições

não for atendida. No caso de estar habilitada, ela pode disparar, retirando um determinado número de marcas das pré-condições e criando outro número de marcas nas pós-condições;

- **Arcos:** guiam o fluxo de marcas pela rede. Conectam lugares a transições e vice-versa;

- **Marcas:** ou fichas ou *tokens*, são componentes dinâmicos do modelo. Seu fluxo determina o comportamento do sistema ao longo do tempo. O número de marcas e sua localização determinam o estado do sistema. As marcas denotam recursos e são representadas graficamente como inscrições dentro dos lugares.

A representação gráfica de uma RdP é feita através de círculos para lugares e barras retangulares para transições. Os círculos pretos dentro dos lugares indicam marcas, e os arcos representam funções de incidência de entrada (quando chegam em um lugar) e de saída (quando saem de um lugar).

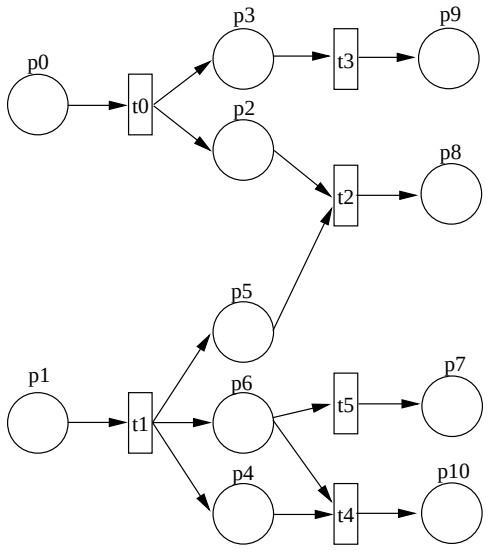


Figura 2.1: Rede de Petri com lugares e transições rotulados

Para cada representação gráfica de uma RdP, existe uma representação algébrica equivalente. A seguir éapresentada a definição de RdP Lugar/Transição [25].

Definição 1 (Rede de Petri) *Uma RdP é uma quádrupla* $N =< P,\ T,\ Pre,\ Pos >$, *onde:*

- $P = p_0,\ p_1,\ ...,\ p_n$ *é um conjunto finito não vazio de lugares.*
- $T = t_0,\ t_1, ...,\ t_m$ *é um conjunto finito de transições, onde* $P \cap T = \emptyset$.
- $Pre : P \times T \rightarrow \mathbb{N}$ *é uma função de incidência de entrada nas transições, indicando pré-condições.*
- $Pos : P \times T \rightarrow \mathbb{N}$ *é uma função de incidência de saída das transições, indicando pós-condições.*

A função de incidência Pre descreve 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 arco entre o lugar p e a transição t.

A função de incidência Pos descreve 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 arco entre a transição t e o lugar p.

O vetor $Pre(.,\ t)$ representa todos os arcos de entrada da transição t, com seus pesos. De forma análoga, o vetor $Pos(.,\ t)$ representa todos arcos de saída da transição t, com seus pesos.

Nesta representação matricial, Pre e Pos são matrizes de n linhas (lugares) e m colunas (transições) e seus elementos pertencem a $\mathbb{N}$.

Assim, para a figura 2.1 tem-se as seguintes matrizes Pre e Pos:

$$Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad Pos = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

2.1.2 Rede de Petri Marcada

Marcas são informações contidas nos lugares. A disposição das marcas e seu número identificam o estado atual do sistema.

Definição 2 (Vetor Marcação) *Seja P o conjunto de lugares de uma RdP. Define-se formalmente* marcação *como um vetor $M = (M(p_1),\ M(p_2),\ ...,\ M(p_n))$, onde $n = |P|$, para todo $p_i \in P$, tal que $M(p_i) \in \mathbb{N}$.*

Definição 3 (RdP Marcada) *Uma* RdP Marcada *é uma tupla $RM = < N,\ M_0 >$, onde N é a estrutura da rede e M_0 é um vetor marcação conhecido como marcação inicial.*

O disparo de transições corresponde a um evento do sistema e faz evoluir a marcação. Esta evolução simula o comportamento dinâmico do sistema.

2.1.3 Equação Fundamental

A dinâmica da RdP é dada pelos disparos de transições habilitadas. Uma transição está habilitada se todas as suas pré-condições forem atendidas, o que significa que, numa rede marcada, todos os lugares que possuem arcos para esta determinada transição devem

possuir um número suficiente de marcas, conforme o peso do arco. Usando-se a representação matricial, uma transição t está habilitada para uma determinada marcação M se, e somente se:

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

ou seja:

$$\forall p \in P, M(p) \geq Pre(p,t)$$

Assim, se M é uma marcação da RdP N, habilitada a transição t, a marcação derivada do disparo de t é:

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

Considerando C a matriz de incidência, dada por:

$$C = Pos - Pre,$$

e o vetor $\overline{s}$, como o vetor característico do disparo de t, dado por $\overline{s} : T \rightarrow \mathbb{N}$, tal que $\overline{s}(t)$ é o número de vezes que a transição t aparece na seqüência s, então uma nova marcação M_g de uma marcação M, após o disparo da seqüência s, é dada por:

$$M_g = M + C.\overline{s}$$

Esta equação é chamada *equação fundamental* de N.

2.1.4 Alcançabilidade

O comportamento dinâmico de uma RdP N é definido pelo conjunto de marcações acessíveis $\mathcal{A}(N;M)$ que pode ser atingido a partir de uma marcação inicial M, através de

uma sequência de disparos s.

$$\mathcal{A}(N;M) = \{M_i, \exists s M \xrightarrow{s} M_i\}$$

Este conjunto de marcações pode ser representado por um grafo $GA(N;M)$, que possui um conjunto de nós, representando as marcações acessíveis $\mathcal{A}(N;M)$. Se existe um arco orientado que liga dois nós M_i e M_j, então existe uma transição sensibilizada que permite passar de uma marcação M_i para M_j.

$$M_i \xrightarrow{t} M_j$$

O problema de alcançabilidade em RdPs, baseado numa marcação inicial M_0, é verificar se uma determinada marcação é alcançável a partir de M_0.

Pode-se usar a equação fundamental para encontrar um vetor $\overline{s}$ dada uma RdP N e duas marcações M (marcação inicial) e M_g (marcação objetivo). A solução que satisfaz a equação deve ser um vetor inteiro não-negativo e sua existência é apenas uma condição necessária para que M_g seja alcançada a partir de M. Esta condição se torna necessária e suficiente para RdPs *acíclicas*.

Assim, uma marcação M_g é dita alcançável a partir de M, se, e somente se, existe uma seqüência de transições s, tal que:

$$M \xrightarrow{s} M_g$$

Convém ressaltar que neste trabalho não há interesse somente em saber se uma determinada marcação é ou não alcançável, mas também deve-se obter uma seqüência ordenada de disparos que leve à marcação final. Muitas vezes a exata marcação esperada não será alcançada, mas sim uma marcação que contém a marcação buscada. Quando uma marcação s está contida em uma marcação M ($s \subseteq M$), diz-se que s é uma submarcação de M. Como as marcações são representadas como vetores, então tem-se que $s \leq M$.

Várias são as soluções para o problema da alcançabilidade. Em [32] é utilizada a

equação fundamental, gerando um sistema de inequações, usando técnicas de programação inteira. Para uma rede acíclica, tem-se:

$$M_g = M_0 + C.\overline{s}$$

fazendo-se algumas operações, obtém-se:

$$C.\overline{s} \leqslant M_0 - S_g$$

onde S_g é uma submarcação que representa o objetivo do problema. Qualquer vetor $\overline{s}$ que é solução para o problema de programação inteira acima, representa quantas vezes cada transição da rede é disparada para obter M_g a partir de M_0, onde $M_g(p) \geq S_g(p)$, para todo lugar p da rede.

Outras técnicas para se resolver o problema de alcançabilidade envolvem o grafo de alcançabilidade. Este grafo define, dada uma marcação, quais são todas as outras possíveis marcações que podem ser obtidas, a partir de um disparo de transição. Assim, consegue-se encontrar um caminho partindo de uma determinada marcação inicial até a marcação que se deseja obter. Este método envolve aplicação de métodos de busca e portanto a aplicação de técnicas heurísticas para guiar estes algoritmos na construção e análise destes grafos.

Uma outra técnica, que é mais detalhadamente apresentada neste texto, transforma a rede de petri em um conjunto de cláusulas do cálculo proposicional. Então, aplica-se resolvedores SAT para encontrar uma determinada valoração, que indica o disparo de transições, levando a uma marcação final.

Claramente um problema de planejamento pode ser tratado como um problema de alcançabilidade em RdP. A situação inicial corresponde à marcação inicial da RdP, bem como a situação objetivo pode ser vista como a marcação final. As ações que podem ser efetuadas correspondem às transições que podem ser disparadas (habilitadas) na RdP. Portanto, técnicas que resolvem problemas de alcançabilidade podem também resolver problemas de planejamento.

Assim, uma maneira de se resolver um problema de planejamento é a sua representação como uma RdP seguida da aplicação de métodos para solução de alcançabilidade. Infelizmente, os atuais métodos para a solução destes problemas envolvem técnicas com custo computacional alto, o que leva à necessidade do desenvolvimento de novos métodos mais eficientes para a solução destes problemas.

2.2 Algoritmo Petriplan

O Petriplan [32] é um algoritmo de planejamento que utiliza RdP como ferramenta de modelagem. A partir da linguagem descritiva de problemas de planejamento PDDL [15], o Petriplan gera uma RdP acíclica, *Rede de Planos* [33], na qual se torna possível a aplicação de métodos de alcançabilidade, em especial buscas e métodos de programação inteira (baseada na equação fundamental). Quando a solução é encontrada, é convertida para a representação de planos, resolvendo assim o problema de planejamento. O Algoritmo 1 mostra o Petriplan.

Algoritmo 1 Petriplan

```
Obtenção da RdP que modela o problema de planejamento
repeat
  Solução da Alcançabilidade
  if não encontrou a solução then
    Expande a RdP
  end if
until solução seja encontrada
```

O uso de RdP para modelar os problemas de planejamento transformando-os em problemas de alcançabilidade traz algumas vantagens em relação ao grafo de planos, como economia na representação e a não necessidade de se usar estruturas externas ao formalismo para representar relações de exclusão mútua.

Todas as caraterísticas dinâmicas do problema são representados pela RdP. Dependências e conflitos podem ser facilmente representados por elementos da própria rede, não necessitando de estruturas auxiliares. Todo o esforço de representação é amparado pela própria dinâmica da rede.

Em especial, resolver um problema de planejamento pode ser visto como resolver um

problema de alcançabilidade. Sabe-se que resolver alcançabilidade para RdPs em geral é EXPSPACE [13]. Para RdPs acíclicas (como é o caso deste trabalho), a complexidade está em NP-Completo. Caso a RdP seja acíclica e não tenha conflitos, então o tempo para resolver alcançabilidade é polinomial.

Portanto, para resolver o problema de alcançabilidade em tempo polinomial deve-se eliminar todos os conflitos da RdP, de tal forma que uma simples varredura obtenha o resultado, que para o problema de planejamento representado é o plano. A proposta aqui é gerar a partir dos conflitos da RdP uma fórmula lógica, para a qual um procedimento similar a SAT procura por uma valoração que a satisfaça. Esta fórmula é construída de forma que valores que a satisfaçam tenham como contrapartida na RdP a completa eliminação dos conflitos.

2.3 Ambiente IPE

O IPE é um ambiente para o desenvolvimento de planejadores [22]. É desenvolvido usando-se orientação a objetos, com uma modelagem de classes flexível ao ponto de permitir a fácil inserção de novas estruturas de representação e novos métodos de planejamento. Com o mesmo ambiente, consegue-se uma medida de eficiência comparativa real, entre diversas técnicas diferentes de planejamento.

O processo efetuado para a solução de um problema de planejamento inicia-se com o *analisador sintático*, que lê um arquivo contendo o problema a ser resolvido (por exemplo em PDDL), alimentando estruturas internas de representação. Baseado nas informações extraídas da leitura do problema, o módulo *codificador* gera uma representação, que pode ser uma instância SAT, Grafo de Planos ou mesmo uma Rede de Petri. Com base nessa representação, um *resolvedor* é aplicado.

O grande objetivo do ambiente IPE é ser uma plataforma na qual se possa desenvolver analisadores sintáticos, codificadores e resolvedores diferentes, podendo-se assim testá-los e medir sua eficiência, disponibilizando material para estudo de novas técnicas de planejamento.

Atualmente, tem-se uma implementação estável de um codificador para RdP e os estu-

dos neste ambiente estão focados no desenvolvimento de novos resolvedores do problema de alcançabilidade. Os algoritmos propostos no capítulo 4 foram implementados nesta plataforma.

CAPÍTULO 3

FORMAS PROPOSICIONAIS DE REPRESENTAÇÃO E ALGORITMOS PARA SAT

Como visto anteriormente, a proposta aqui é, dada uma RdP representando um problema de planejamento, eliminar todos os seus conflitos através de um procedimento baseado em SAT. Com uma rede sem conflitos, resolver alcançabilidade se torna trivial, não exigindo algoritmos elaborados, pois uma varredura simples da rede visitando cada lugar e transição já retorna um resultado.

A modelagem proposta não usa fórmulas na CNF, em virtude do custo computacional, como será descrito mais adiante. Portanto neste capítulo, serão apresentadas outras formas normais para SAT, bem como análise de suas propriedades e algoritmos pertinentes ao problema.

3.1 SAT para CNF

O problema da satisfatibilidade (SAT) é de extrema importância na matemática e na ciência da computação. Na prática, SAT é fundamental na resolução de vários problemas, desde projeto de circuitos integrados até raciocínio automático, portanto, tem um papel fundamental na construção de sistemas computacionais eficientes.

O problema SAT é o centro de uma grande família de problemas intratáveis denominados NP-Completos [16]. Algoritmos elaborados e direcionados foram propostos recentemente para resolver este problema de forma eficiente.

Resolver um problema SAT consiste na busca de uma valoração que satisfaça uma fórmula bem formada do cálculo proposicional. Geralmente esta fórmula está na Forma Normal Conjuntiva (CNF), isto é, com negações incidindo somente sobre variáveis proposicionais e composta por uma conjunção de disjunções. Por exemplo, a fórmula abaixo está na CNF.

$$(p \vee \neg q) \wedge (r \vee s) \wedge \neg t$$

Uma instância SAT é definida por:

- Um conjunto de n variáveis $\mathcal{X} = \{x_1, x_2, ..., x_n\}$
- Um conjunto de literais. Um literal é uma variável ($Q = x$) ou a negação de uma variável ($Q = \neg x$);
- Um conjunto de m distintas cláusulas: $\mathcal{F} = \{C_1, C_2, ..., C_m\}$. Cada cláusula consiste de literais combinados pelo conectivo lógico *ou* ($\vee$).

Seja a seguinte fórmula na forma normal conjuntiva obtida a partir de $\mathcal{F}$

$$\phi = C_1 \wedge C_2 \wedge ... \wedge C_m$$

Uma valoração é uma função $v : \mathcal{X} \rightarrow \{Falso, Verdadeiro\}$. A fórmula é dita *satisfatível* se existe uma valoração satisfazendo ϕ (valor verdade $Verdadeiro$).

Definição 4 (Modelo) *Um modelo para uma fórmula ϕ é uma valoração v para todas as suas variáveis de tal forma ϕ é satisfeita por v.*

O objetivo de um problema SAT é determinar se a fórmula ϕ é satisfatível, isto é, se existe uma atribuição de valores verdade para as variáveis, tal que ϕ seja satisfeita.

Existem várias técnicas para se resolver uma instância SAT, desde variações de procedimentos de busca, até algoritmos paralelos, passando por programação inteira e outras representações. A seguir serão apresentados alguns algoritmos eficientes.

3.1.1 Procedimento de Davis-Putnam

Muitos resolvedores SAT de domínio público propostos recentemente (GRASP[21], SATO[37], WALKSAT[30], CHAFF[24]) implementam variações do procedimento de busca

de Davis-Putnam (DP)[11] sobre fórmulas na CNF. Sendo S um conjunto de cláusulas, o procedimento DP se resume na aplicação das seguintes regras:

- Regra da Tautologia: remover todas as cláusulas de S que são tautológicas. O conjunto restante S' é satisfatível se, e somente se, S for;

- Regra da Cláusula Unitária: se existe uma cláusula unitária L em S, pode-se obter S' de S deletando-se todas as cláusulas unitárias e todas as cláusulas de S que contém L. Se S' é vazio, então S é satisfatível. Caso contrário, obtém-se S'' de S' deletando-se $\neg L$ de S'. S'' é satisfatível se, e somente se, S também for;

- Regra do Literal Puro: um literal L de uma cláusula de S é dito *puro* em S se, e somente se, o literal $\neg L$ não aparece em nenhuma cláusula de S. Se um literal L é puro em S, deleta-se todas as cláusulas contendo L, resultando em S'. S' é satisfatível se, e somente se, S também for;

- Regra da Eliminação de Fórmulas Atômicas: se S pode ser colocado no seguinte formato:

$$(p_1 \vee L) \wedge (p_2 \vee L) \wedge ... \wedge (p_n \vee L) \wedge (q_1 \vee \neg L) \wedge (q_2 \vee \neg L) \wedge ... \wedge (q_m \vee \neg L) \wedge R$$

onde p_i, q_i e R são livres de L e $\neg L$, então pode-se obter os conjuntos:

$$S_1 = p_1 \wedge p_2 \wedge ... \wedge p_n \wedge R$$

e

$$S_2 = q_1 \wedge q_2 \wedge ... \wedge q_m \wedge R.$$

Assim, S é satisfatível se, e somente se, $(S_1 \vee S_2)$ for satisfatível, isto é, se S_1 ou S_2 forem satisfatíveis.

Esse procedimento gasta muito tempo de computação para um conjunto grande de cláusulas, sendo ineficiente mesmo para pequenas instâncias SAT. Assim, cortes no espaço de busca se tornam imprescindíveis. Em especial, o resolvedor CHAFF [24] introduz mecanismos de poda no espaço de busca original do DP e uma estratégia de decisão otimizada para velocidade, o que dá um desempenho muito bom em relação a outras estratégias.

3.1.2 GSAT

O procedimento GSAT [31] é baseado em busca local para encontrar uma valoração que satisfaça um determinado conjunto de cláusulas. Inicialmente, atribui-se um valor aleatório para cada variável proposicional, então, trocam-se alguns valores em busca de uma valoração que torne a maior quantidade de cláusulas satisfatível. Estas trocas são feitas repetidas vezes até que uma valoração que satisfaça todas as cláusulas seja encontrada ou até que um determinado número de trocas seja executado.

Na sua decisão sobre qual valor de variável trocar, o GSAT efetua uma busca local, sempre tentando atribuições satisfatíveis que diferem em uma variável, da atribuição atual.

Por causa do seu procedimento de busca local, GSAT é correto mas não completo, isto é, quando é encontrada uma solução, tem-se certeza que esta solução é correta, mas uma resposta negativa é inconclusiva. Isto acontece por causa do critério de escolha sobre qual variável será trocada. Algumas vezes, pode-se encontrar uma solução seguindo por uma valoração que diminui a quantidade de cláusulas satisfatíveis, indicando que o algoritmo é sensível ao problema de mínimos locais.

3.1.3 CHAFF

O CHAFF [24] é um resolvedor SAT eficiente, baseado no procedimento de busca de Davis-Putnam. Ganhou dois prêmios na categoria industrial do Sat Competition 2004 e esteve entre os mais rápidos resolvedores no Sat Competition 2005.

O processo desenvolvido pelo CHAFF pode ser descrito pelo Algoritmo 2, no qual o procedimento `decide()` seleciona uma variável sem atribuição de valor verdade (*Verda-*

deiro ou *Falso*) e dá um valor a ela. Esta atribuição é conhecida como *decisão* e é de extrema importância saber os pontos de decisão, visto que o procedimento é baseado em *backtracking*.

Algoritmo 2 CHAFF

```
loop
  if !decide() then
    return satisfatível
  end if
  while !bcp() do
    if !resolveConflict() then
      return não-satisfatível
    end if
  end while
end loop
```

Algoritmo 3 resolveConflict

Saída: se conseguiu resolver o conflito

```
d ← decisão mais recente não tentada com Verdadeiro e Falso
if d == NULL then
  return Falso
end if
Trocar o valor de d
Marcar d como tentado com Verdadeiro e Falso
Defazer qualquer implicação inválida
return Verdadeiro
```

O procedimento `bcp()` (*Boolean Constraint Propagation*), além da identificação de cláusulas unitárias, faz a verificação da atribuição de valores, verificando as restrições que essa valoração implica (também chamado de *implicações*). Em outras palavras, dada uma valoração, deve-se descobrir, para as variáveis ainda não valoradas, quais delas terão obrigatoriamente um determinado valor para que as cláusulas sejam satisfatíveis (dedução do valor verdade de variáveis não atribuídas a partir das que já foram atribuídas). Convém ressaltar, que não só restrições podem ser derivadas desse procedimento, mas também *conflitos*, isto é, quando as implicações levam uma variável a, obrigatoriamente, ter o valor lógico $Verdadeiro$ e $Falso$ ao mesmo tempo.

O procedimento `resolveConflict()` resolve conflitos com restrições que foram criadas na última atribuição de valores verdade. Isso é feito trocando valores verdade da

última atribuição que gerou o conflito pelo valor que ainda não foi tentado. Assim, o procedimento `bcp()` pode continuar os cálculos de restrições normalmente. Caso os dois valores verdade já tenham sido tentados para aquela variável, então um *backtracking* deve ser efetuado até o ponto de decisão anterior, onde não foram tentados os dois valores verdade ainda.

Segundo [24], na prática 90% do processamento da maioria dos resolvedores SAT (os que se baseiam no algoritmo de Davis-Putnam) se encontra no procedimento `bcp()`. Portanto, se esse procedimento for escrito de forma eficiente, o resolvedor será eficiente. Assim, o CHAFF implementa técnicas eficientes para encontrar cláusulas que seguem um padrão de atribuiçãode valores verdade específico, a saber, todos os literais da cláusula exceto um com valor já atribuído a $Falso$.

Outra melhoria que o CHAFF introduz é no procedimento de decisão de qual variável será instanciada e para qual valor. Baseado em heurísticas, esse procedimento é de extrema importância para a velocidade de processamento do resolvedor SAT.

A heurística mais difundida é conhecida como *Dynamic Largest Individual Sum* (DLIS), que seleciona para atribuição o literal que aparece mais freqüentemente em cláusulas não resolvidas. Para o CHAFF foi desenvolvida uma nova estratégia, chamada *Variable State Independent Decaying Sum*, onde não só a frequência da variável é considerada, mas também a sua polaridade (sinal). Em cada decisão, a variável e sua polaridade mais freqüente é escolhida.

3.2 SAT para Formas Não-Clausais

Originalmente SAT é definido para fórmulas em CNF e a grande maioria das aplicações se baseia nesta forma normal. Entretanto, muitos problemas, quando são analisados como satisfatibilidade, geram fórmulas que não estão em CNF, sendo necessária uma conversão prévia para que se trate o problema como SAT tradicional.

Converter uma fórmula qualquer para CNF se faz pela aplicação sucessiva da operação distributiva sobre os conectivos lógicos e leis de De Morgan. Esta transformação é especialmente cara computacionalmente, visto que o número de cláusulas geradas pode ser

exponencial, levando a um uso excessivo de memória.

Dado um problema cuja modelagem pode ser feita através de fórmulas não-clausais, sua solução via SAT clássico demanda a aplicação de dois procedimentos potencialmente exponenciais: conversão para CNF e SAT para CNF. Estes dois procedimentos aplicados em conjunto elevam drasticamente o tempo de execução e consumo de memória do resolvedor. Uma solução é construir resolvedores SAT para fórmulas não-clausais.

A seguir é apresentada uma solução para SAT não-clausal baseado no GSAT. Na mesma linha, Stachniak[34] propõe uma variação do WALKSAT para o tratamento de fórmulas não-clausais.

3.2.1 GSAT não-clausal

Sebastiani em [29] propõe uma variação no algoritmo GSAT clássico (para fórmulas na CNF) para que este possa ser aplicado a fórmulas não-clausais. O algoritmo 4 mostra o esquema do GSAT, sobre o qual as alterações de Sebastiani foram propostas.

Algoritmo 4 GSAT

Entrada: Fórmula ϕ
  for $j = 0$ to $MaxTries$ **do**
    $T = initial(\phi)$
    for $k = 1$ to $MaxFlips$ **do**
      if $T \models \phi$ **then**
        return T
      else
        $PossFlips = hill\_climb(\phi, T)$
        $V = pick(PossFlips)$
        $T = flip(V, T)$
        $update\_scores(\phi, V)$
      end if
    end for
  end for

Este algoritmo é baseado em *scores*, onde o *score* de uma fórmula com relação a uma valoração é o número de cláusulas falsificadas por esta valoração. O algoritmo 4 é construído para que, a cada troca de valoração, seja escolhida uma variável que diminuirá o número de cláusulas falsificadas, isto é, fazendo o *score* tender a zero.

Em [29] é feita uma análise deste algoritmo, baseada no fato que as funções $initial()$,

hill_climb(), *pick*() e $flip()$ não dependem da estrutura da fórmula e o cálculo dos *scores* é o único passo no qual é requerido que a fórmula esteja em forma clausal é a chamada da função *update_scores*(). Assim, para a aplicação deste algoritmo para fórmulas não-clausais, basta estender o conceito de *scores* para estas formas, gerando algoritmos eficientes para este cálculo e alterando a implementação da função *update_scores*() para suportar estas fórmulas.

Na solução proposta por Sebastiani o cálculo dos *scores* é feito de forma direta, sem a necessidade de se transformar a fórmula para CNF. O processo se inicia a partir dos literais, sendo que um literal falsifica ou não apenas uma cláusula. Nos nodos internos, caso se tenha uma conjunção, então o número de cláusulas falsificadas é exatamente a soma das cláusulas falsificadas em cada um dos filhos. Em caso de disjunção, o número de cláusulas falsificadas é o produto do número de cláusulas falsificadas pelos filhos.

Convém ressaltar que este procedimento não é completo, isto é, baseia-se em um heurística que visa minimizar o número de cláusulas falsificadas e que não garante a solução do problema, podendo recair em mínimos locais. Observa-se que o problema dos mínimos locais é tratado no algoritmo como uma valoração aleatória a partir da função *initial*(), que é executada cada vez que um máximo de trocas ($MaxFlips$) não gerar uma valoração que satisfaz a fórmula.

3.3 Formas Não-Clausais

Resolver um problema SAT para um conjunto de cláusulas é computacionalmente intratável sem o uso de heurísticas ou métodos mais elaborados. Existem também procedimentos para solução de problemas SAT para fórmulas não-clausais, desde que estejam em uma determinada forma normal. Por exemplo, segundo [9], se a fórmula estiver na forma normal d-DNNF (seção 3.3.2), pode-se fazer o teste de satisfatibilidade, contagem de modelos (*Model Counting*) e enumeração de modelos (*Model Enumeration*) em tempo polinomial.

Este estudo é importante pois a conversão da rede de petri para instância SAT proposta no capítulo 4 gera trivialmente uma instância SAT em forma não-clausal.

Segue uma descrição de formas normais alternativas à CNF e algoritmos relevantes.

3.3.1 *Negation Normal Form*

Uma sentença está na *Negation Normal Form* (NNF) se é construída a partir de literais usando-se somente disjunções e conjunções [7]. Uma representação prática de sentenças na NNF é em termos de um grafo acíclico dirigido (DAG), que possui um nodo raiz, onde:

- cada nodo folha é rotulado: ou é valor lógico $Verdadeiro$, ou $Falso$ ou um literal;
- cada nodo interno é rotulado: ou $\vee$ ou $\wedge$, e pode ter qualquer quantidade de filhos.

Por exemplo, a seguinte fórmula está na NNF:

$$\{[(\neg p \wedge q) \vee (p \wedge \neg q)] \wedge [(r \wedge s) \vee (\neg r \wedge \neg s)]\} \vee \{[(\neg p \wedge \neg q) \vee (p \wedge q)] \wedge [(r \wedge \neg s) \vee (\neg r \wedge s)]\}$$

e sua representação como um DAG pode ser vista na Figura 3.1.

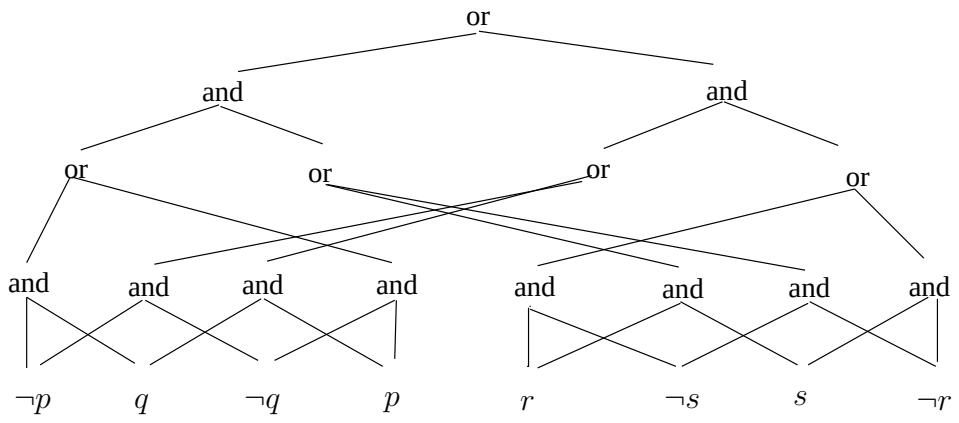


Figura 3.1: DAG de uma fórmula na NNF

Deve-se perceber que existe uma restrição muito forte: o escopo das negações só pode ser variáveis atômicas. Claramente, uma fórmula na Forma Normal Conjuntiva (CNF) está na NNF.

A omissão da negação em nodos internos não limita a representatividade da NNF, visto que qualquer fórmula tem outra equivalente, sem negações sobre $\vee$ ou $\wedge$, simplesmente obtida a partir da aplicação das leis de De Morgan sucessivas vezes.

3.3.2 Decomponibilidade e Determinismo de NNFs

Definição 5 (Decomponibilidade) *Dada uma fórmula na NNF, para cada conjunção* $\alpha_1 \wedge ... \wedge \alpha_n$*, se uma variável não aparece em mais de um termo* α_i*, isto é, os átomos encontrados em um termo desta conjunção não são os mesmos encontrados em outros termos, então ela é decomponível.*

Uma fórmula está na *Decomposable Negation Normal Form* (DNNF) se estiver na NNF e satisfizer a propriedade de decomponibilidade.

Por exemplo, extraindo-se da Figura 3.1 todos os ramos exceto o *and* esquerdo, imediatamente inferior ao nodo raiz, obtém-se o DAG da Figura 3.2. Analisando-o, percebe-se que os dois ramos imediatamente inferiores são:

- $(\neg p \wedge q) \vee (\neg q \wedge p)$
- $(r \wedge s) \vee (\neg s \wedge \neg r)$

os quais não compartilham nenhuma variável proposicional. Portanto, para este ramo *and* tem-se a propriedade de decomponibilidade. Para verificar se a fórmula toda é decomponível, basta aplicar este mesmo processo para cada um dos ramos *and*.

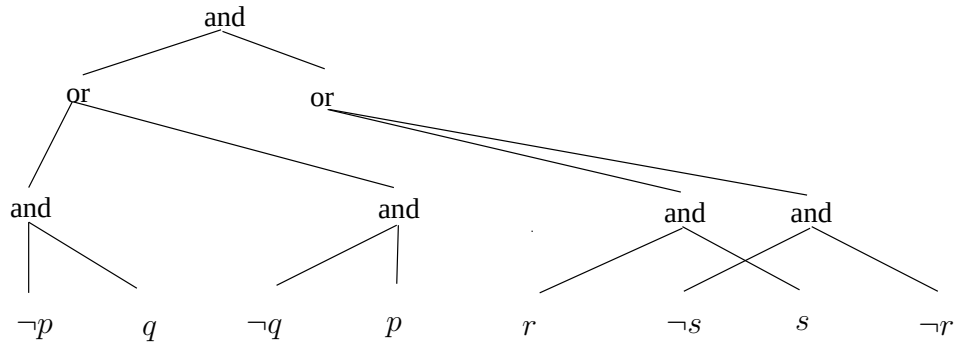


Figura 3.2: Exemplo de decomponibilidade

Em outras palavras, uma fórmula decomponível permite que certos tipos de problemas computacionais sejam decompostos em problemas menores, dado que suas subsentenças não compartilham átomos. Sempre que um nodo *and* for encontrado, cada um de seus ramos terá, seguramente, informações diferentes, podendo ser computado de forma independente.

Decomponibilidade é a propriedade que torna DNNF computacionalmente tratável para o teste de satisfatibilidade. Este teste é como segue:

- se α é um literal, então α é satisfatível;
- se $\alpha = \alpha_1 \vee ... \vee \alpha_n$ (também representado por $\vee_i \alpha_i$), então α é satisfatível se, e somente se, *algum* α_i o for;
- se $\alpha = \alpha_1 \wedge ... \wedge \alpha_n$ (também representado por $\wedge_i \alpha_i$), então α é satisfatível se, e somente se, *todos* os α_i forem.

Analisando-se a definição acima, percebe-se que esta operação é $\mathcal{O}(n)$, onde n é o número de literais da fórmula.

Definição 6 (Determinismo) *Dado uma fórmula na NNF, se para toda disjunção $\vee_i \alpha_i$, todo par de termos α_i e α_j, onde $i \neq j$, $\alpha_i \wedge \alpha_j$ é uma contradição, isto é, são mutuamente exclusivos, então a fórmula é determinística.*

Uma fórmula está na *Deterministic Negation Normal Form* (d-NNF) se ela estiver na NNF e satisfizer a propriedade de determinismo. Uma fórmula que satisfaz as propriedades de determinismo e decomponibilidade está na *Deterministic Decomposible Negation Normal Form* (d-DNNF).

Por exemplo, extraindo-se da Figura 3.1 todos os ramos exceto o *or* mais à esquerda, obtém-se o DAG da Figura 3.3. Analisando-o, percebe-se que os dois ramos imediatamente inferiores são:

- $\neg p \wedge q$
- $\neg q \wedge p$

e que são inconsistentes entre si, isto é:

$$(\neg p \wedge q) \wedge (\neg q \wedge p)$$

é contradição.

Portanto, para este ramo *or* tem-se a propriedade de determinismo. Para verificar se a fórmula toda é determinística, basta aplicar este mesmo processo para cada um dos ramos *or*.

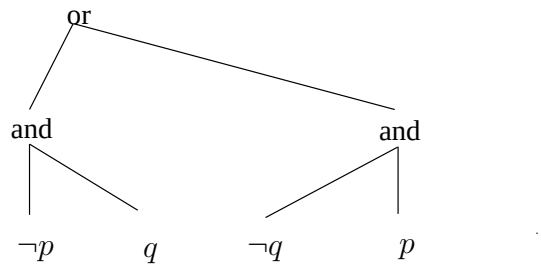


Figura 3.3: Exemplo de determinismo

É a propriedade de determinismo que torna a contagem de modelos (*Model Counting*) e sua enumeração (*Model Enumeration*) tratável, isto é, o número de modelos de uma d-DNNF $\wedge_i \alpha_i$ é o produto do número de modelos de cada termo α_i e o número de modelos de uma d-DNNF $\vee_i \alpha_i$ é a soma do número de modelos de cada termpo α_i.

Da mesma forma, os modelos de uma fórmula na d-DNNF é a união dos modelos dos termos $\vee_i \alpha_i$ e o produto cartesiano entre os modelos dos termos $\wedge_i \alpha_i$. A enumeração de modelos é tratada detalhadamente mais adiante.

3.3.3 Geração de d-DNNF

Para se transformar uma teoria proposicional em d-DNNF [27] deve-se aplicar, recursivamente, uma operação chamada *condicionamento*[1]. Esta operação é derivada da identidade conhecida como Expansão de Shannon [28] dada por:

$$F = (F[Verdadeiro/p] \wedge p) \vee (F[Falso/\neg p] \wedge \neg p)$$

onde $F[x/p]$ indica a troca de todas as ocorrências de p em F por x.

O condicionamento de Δ sobre o literal α, escrito $\Delta \mid \alpha$, é obtido pela troca de cada folha α no DAG por $Verdadeiro$ e cada folha $\neg\alpha$ por $Falso$. Por exemplo, se $\Delta = (p \vee q) \wedge (\neg p \vee r)$, então $\Delta \mid p$ resulta em $(Verdadeiro \vee q) \wedge (Falso \vee r)$. A identidade

[1]Do inglês *conditioning*

que representa a Expansão de Shannon pode ser vista em termos de condicionamento como:

$$F = (F\,|\,p \ \wedge \ p) \vee (F\,|\,\neg p \ \wedge \ \neg p).$$

Portanto, para transformar uma teoria proposicional Δ em d-DNNF, basta:

- enumerar todas as variáveis de Δ, sendo elas $x_1, x_2, ..., x_n$;
- aplicar o condicionamento em x_1, obtendo $(\Delta\,|\,x_1 \ \wedge \ x_1) \vee (\Delta\,|\,\neg x_1 \ \wedge \ \neg x_1)$;
- aplicar condicionamento em $x_2, x_3, ..., x_n$ para $\Delta \mid x_1$ e $\Delta \mid \neg x_1$.

Segundo [27], esta transformação é correta e completa, gerando fórmulas equivalentes.

Seja $\Delta = (p \vee q) \wedge (\neg p \vee r)$. Para gerar uma fórmula na d-DNNF aplicam-se os passos vistos acima. Para melhor visualização, a fórmula será transformada em seu respectivo DAG (Figura 3.4).

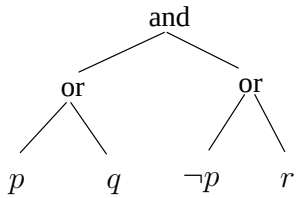


Figura 3.4: Exemplo de transformação d-DNNF

Enumera-se as variáveis para aplicação do condicionamento, na seguinte ordem: p, q e r. Aplica-se a regra $(\Delta\,|\,p \ \wedge \ p) \vee (\Delta\,|\,\neg p \ \wedge \ \neg p)$, obtendo-se o DAG da Figura 3.5.

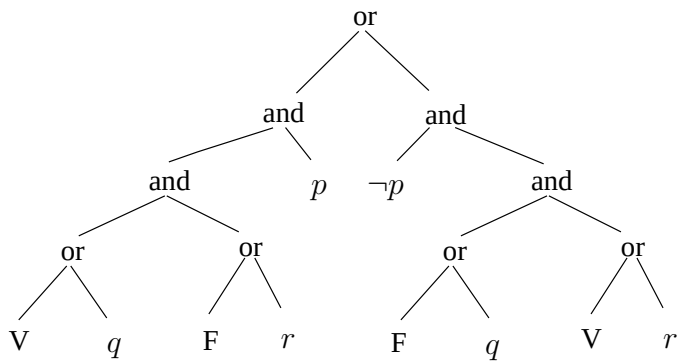


Figura 3.5: Condicionamento em p

Neste momento pode-se aplicar as regras lógicas dos conectivos, para se obter um DAG mais simplificado, ou continuar o condicionamento nas outras variáveis. Caso as simplificações sejam aplicadas, a forma normal obtida é conhecida como *Factored Negation Normal Form* (FNNF) e é tratada com mais detalhes na seção 3.3.4. Neste exemplo, pelo fato da aplicação das regras lógicas simplificar demais o exemplo, optou-se por continuar o condicionamento nas demais variáveis, obtendo-se os DAGs das Figuras 3.6 e 3.7.

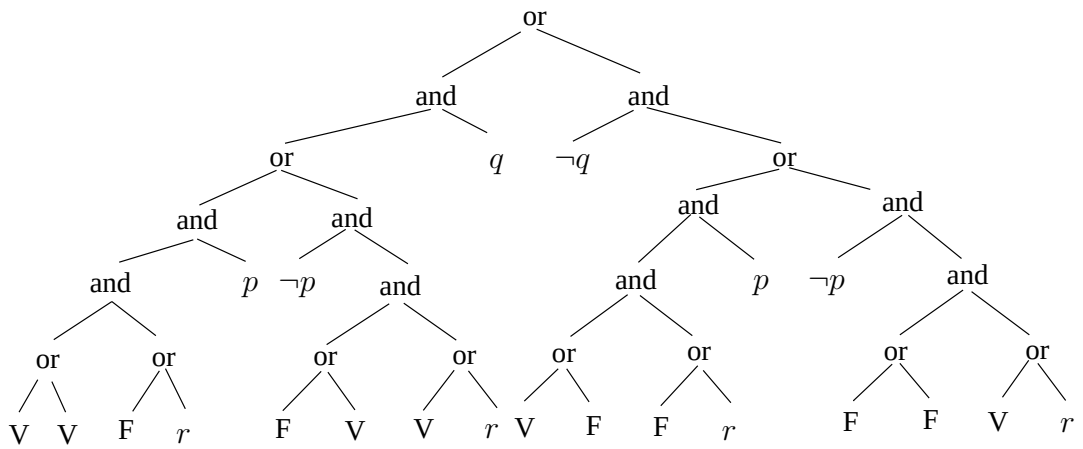


Figura 3.6: Condicionamento em q

Claramente percebe-se que os termos abaixo de um *or*, desconsiderando aqueles rotulados com valores $Verdadeiro$ ou $Falso$, são mutuamente exclusivos (determinismo) e os termos abaixo de um *and* não compartilham variáveis (decomponibilidade), portanto mostrando uma fórmula na d-DNNF. O DAG da Figura 3.8 mostra a aplicação de simplificações baseadas nas tabelas verdade da conjunção e disjunção.

Existe uma correspondência direta entre d-DNNF e BDD. BDD (*Binary Decision Diagram*) é uma estrutura usada para representar uma função booleana, representada por um DAG onde os nodos filhos são $Verdadeiro$ ou $Falso$, e os nodos internos são as variáveis proposicionais da função. Para saber o valor lógico da função, dada uma valoração de todas as suas variáveis, basta caminhar no BDD, da raiz até as folhas, escolhendo os ramos esquerdo ou direito de cada nodo interno conforme a valoração das variáveis proposicionais. Se a valoração indica que a variável p é $Verdadeira$ então escolhe-se o ramo esquerdo no nodo interno rotulado com p. Caso p seja $Falso$ escolhe-se o ramo direito. A Figura 3.9 mostra o BDD correspondente ao DAG da Figura 3.8.

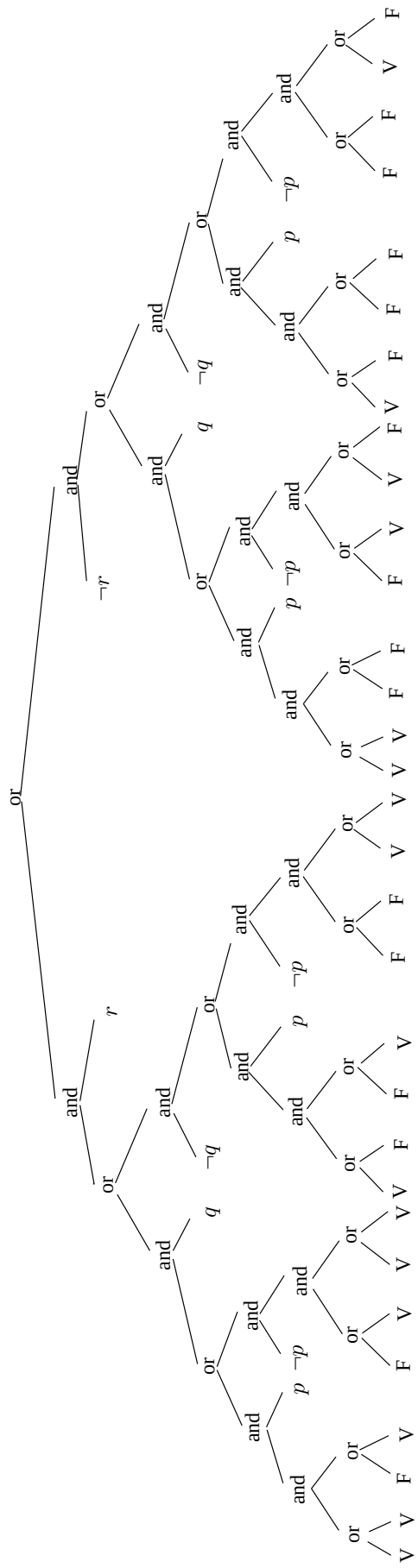


Figura 3.7: Condicionamento em r

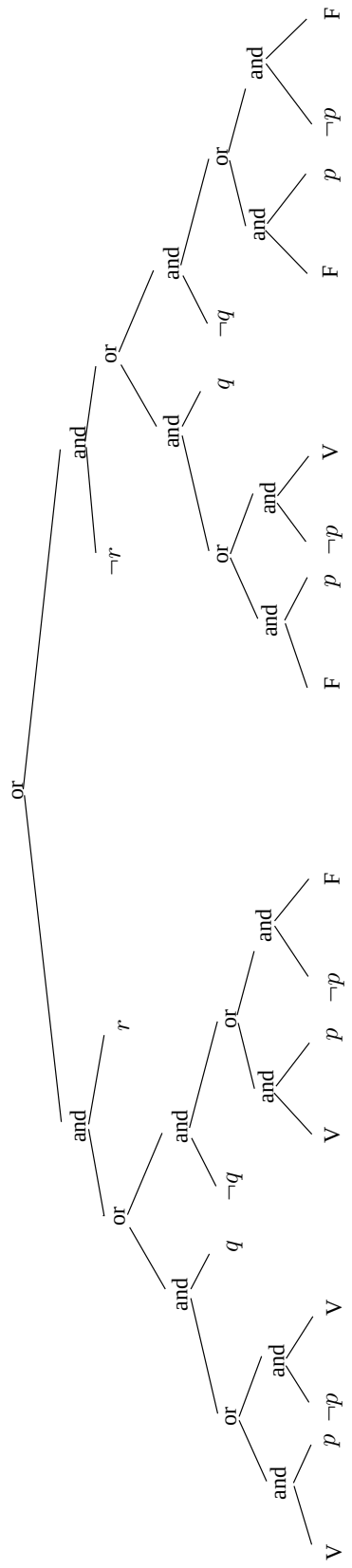


Figura 3.8: Aplicação de tabela verdade em ramos contendo somente valores verdade

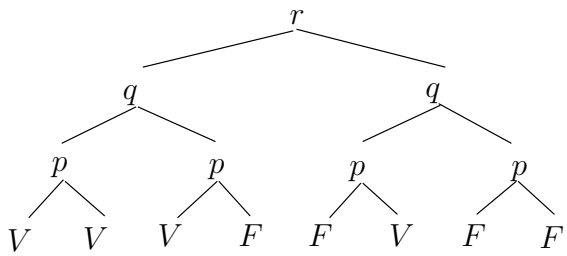


Figura 3.9: BDD

3.3.4 *Factored Negation Normal Form*

Segundo [28], se na geração da d-DNNF, a cada passo de condicionamento, forem aplicadas as seguintes simplificações baseadas em tabela verdade:

- $p \vee p$ troca-se por p
- $p \wedge p$ troca-se por p
- $p \vee Falso$ troca-se por p
- $p \wedge Verdadeiro$ troca-se por p
- $p \vee Verdadeiro$ troca-se por $Verdadeiro$
- $p \wedge Falso$ troca-se por $Falso$
- $p \vee \neg p$ troca-se por $Verdadeiro$
- $p \wedge \neg p$ troca-se por $Falso$

o que se obtém é uma fórmula que está na FNNF (*Factored Negation Normal Form*). Uma fórmula na FNNF também está na d-DNNF, visto que estas simplicações somente diminuem o tamanho da fórmula e não alteram sua estrutura, gerando fórmulas equivalentes.

Para o DAG da Figura 3.7, aplicando-se as regras acima citadas, obtem-se um DAG como o da Figura 3.10.

Outra maneira de se obter uma fórmula na FNNF à partir de uma na NNF é a partir da aplicação do Tableau KE. O tableau KE é um procedimento de prova baseado em tableau, mas que inclui a regra de *cut*, que classicamente não é considerada.

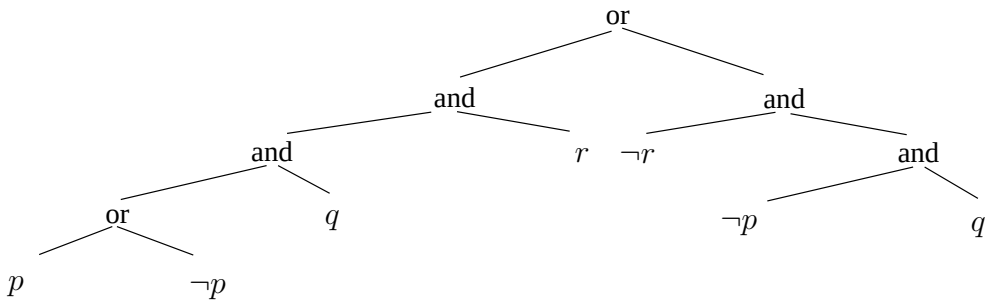


Figura 3.10: Simplificação

Segundo [28] o tableau KE [5] está intimamente ligado à Expansão de Shannon, que é a transformação chave na obtenção da d-DNNF, segundo o método da seção 3.3.3. A aplicação de determinadas regras deste tableau em uma certa ordem sobre uma fórmula na NNF, leva à obtenção de uma fórmula na FNNF.

O tableau KE possui regras α e β, juntamente com a regra de *cut*. A adição da regra *cut* (também conhecido como princípio da bivalência) poderia invalidar o princípio da subfórmula [4] [2]. Assim, no tableu KE foi adicionado o *cut* analítico sendo sua aplicação restrita às subformulas.

No procedimento de transformação para FNNF, somente são usadas a regra *cut* e regras α. As regras β são substituídas pelas regras de simplificação definidas em [23]. Além disso, foram adicionadas regras de simplificação para serem aplicadas no tableau e não somente em fórmulas dentro do tableau. Todas estas regras podem ser vistas na Figura 3.11, as primeiras são regras de simplificação de fórmulas e as demais são regras FNNF para simplificação do tableau.

O procedimento para geração da FNNF pode ser visto no Algoritmo 5.

O tableau resultante terá literais em todos os nodos ou a árvore inteira será reduzida a $Falso$ ou $Verdadeiro$. Este tableau é chamado *Tableau FNNF saturado*. Se na aplicação do *cut* for utilizado sempre a mesma ordem de variáveis, será obtido um FNNF ordenado (OFNNF).

Visto que a aplicação sucessiva da expansão de Shannon pode levar, no pior caso, à duplicação da fórmula, pode-se esperar uma grande utilização de memória, o que na prática

[2]Se existe uma prova de uma fórmula A, então existe uma prova usando somente subfórmulas ou subfórmulas negadas de A.

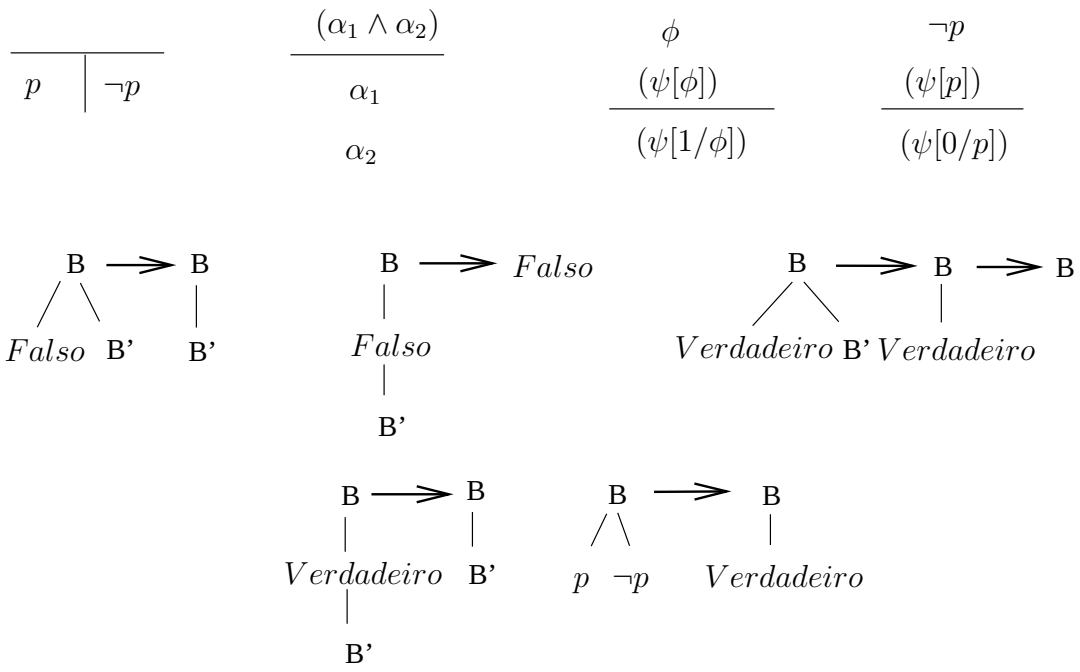


Figura 3.11: Regras para Tableau KE

Algoritmo 5 FNNF

while nem todos os nodos são literais **do**
  Aplicar simplificações baseadas em tabela verdade
  Aplicar regra α
  if não foi possível aplicar a regra α **then**
    Aplicar regras de simplificação de fórmulas: Expansão de Shannon
    if não foi possível aplicar a simplificação **then**
      Aplicar *cut*
    end if
  end if
  Resolve próximo nodo
end while

é inviável. O algoritmo apresentado acima, utilizando-se de busca em profundidade com *backtracking*, é PSPACE [28]. Em comparação com o outro método (aplicação sucessiva da expansão de Shannon), espera-se que a ocupação de memória seja muito mais viável.

3.3.5 Enumeração de Modelos para d-DNNF e FNNF

Enumeração de modelos é a operação que gera todos os modelos para uma fórmula, isto é, todas as valorações que tornam esta fórmula verdadeira. Diferencia-se de SAT, onde se espera simplesmente saber se existe um modelo para uma fórmula.

Segundo [6], a operação de enumeração de modelos para uma fórmula na d-DNNF pode ser efetuada em tempo polinomial. Por causa das propriedades de decomponibilidade e determinismo, basta varrer a fórmula uma vez em busca de todos os modelos. Convém ressaltar que este procedimento é o mesmo para uma fórmula na FNNF (ou mesmo OFNNF), visto que uma fórmula na FNNF também está na d-DNNF.

Seja um modelo representado como um conjunto de literais e suponha que N_i e M_i sejam modelos para uma fórmula (ou subfórmula). O produto cartesiano ($\times$) entre modelos define-se como:

$$\{N_1, ..., N_n\} \times \{M_1, ..., M_m\} \equiv N_1 \cup M_1, N_1 \cup M_2, ..., N_n \cup M_m$$

O conjunto $Models(\Delta)$, conjunto de modelos de uma teoria proposicional Δ, é definido como:

$$Models(\Delta = literal) = \Delta$$
$$Models(\Delta = \bigvee_i \alpha_i) = Models(\alpha_1) \cup Models(\alpha_2) \cup ... \cup Models(\alpha_n)$$
$$Models(\Delta = \bigwedge_i \alpha_i) = Models(\alpha_1) \times Models(\alpha_2) \times ... \times Models(\alpha_n)$$

Se Δ está na d-DNNF e w é uma atribuição de verdade sobre os átomos de Δ, então $w \models \Delta$ se, e somente se, $w \in Models(\Delta)$. A complexidade desta operação é $\mathcal{O}(mn)$, onde m é o tamanho de Δ e $n =\mid Models(\Delta) \mid^2$ [8]. Portanto, se o número de modelos é pequeno o suficiente para ser visto como uma constante, o processo de enumeração leva

um tempo linear sobre o tamanho da fórmula.

Para a fórmula na d-DNNF da Figura 3.10:

$$\Delta = ((p \vee \neg p) \wedge q \wedge r) \vee (\neg r \wedge \neg p \wedge q))$$

obtém-se o conjunto $Models(\Delta)$ abaixo, aplicando-se as regras definidas acima.

- $\{p, q, r\}$
- $\{\neg p, q, r\}$
- $\{\neg p, \neg q, \neg r\}$

Percebe-se neste conjunto de modelos que todas as variáveis proposicionais aparecem em todos os modelos. Caso a regra do *Terceiro Excluído*[3] fosse aplicada ao DAG da Figura 3.10, então seriam obtidos somente duas valorações, a saber:

- $\{q, r\}$
- $\{\neg p, \neg q, \neg r\}$

sendo que a ausência da variável proposicional p no primeiro modelo, indicaria uma condição irrelevante em relação ao valor lógico de p, sendo portanto um implicado primo.

Um algoritmo que percorre uma fórmula na d-DNNF pode ser construído através do caminhamento no DAG correspondente. As operações de produto cartesiano e união são sempre executadas quando um nó é conjunção ou disjunção, respectivamente. O algoritmo para enumeração de modelos pode ser visto no Algoritmo 6.

Todas as considerações sobre enumeração de modelos feitas para d-DNNF valem para FNNF. Esta operação será usada para se resolver o problema de alcançabilidade em redes de petri, em conjunto com a modelagem dos conflitos da rede em uma fórmula não clausal. No próximo capítulo será mostrada a transformação do problema de alcançabilidade em RdP para uma instância SAT baseada em forma não-clausal.

[3] $\alpha \vee \neg\alpha = Verdadeiro$

Algoritmo 6 Models

Entrada: Nodo
Saída: Modelos para Nodo

```
if nodo é um literal then
    return literal
end if
if nodo é disjunção then
    M ← ∅
    for all filho f de Nodo do
        M ← M ∪ Models(f)
    end for
    return M
end if
if nodo é conjunção then
    M ← ∅
    for all filho f de Nodo do
        M ← M × Models(f)
    end for
    return M
end if
```

CAPÍTULO 4

ALCANÇABILIDADE EM REDES DE PETRI VIA SAT PARA FORMAS NÃO-CLAUSAIS

Este trabalho pretende resolver o problema de planejamento usando como substrato formal as Redes de Petri. Neste contexto, o problema de planejamento é visto como um problema de alcançabilidade que é tratado por um resolvedor SAT baseado em fórmulas não-clausais, a partir do seu mapeamento para fórmulas em NNF.

O procedimento proposto, de maneira simples, pode ser visualizado através do Algoritmo 7.

Algoritmo 7 Proposta

```
Carrega o PDDL
repeat
  Expande a rede até que os objetivos sejam encontrados consistentemente
  Gera a fórmula na NNF
  Transforma a fórmula para FNNF
  Aplica resolvedor SAT para FNNF
  if resolvedor SAT encontrou valoração para a fórmula then
    Constrói o plano a partir da valoração
  end if
until encontrar plano
```

4.1 Modelagem

A estrutura de representação usada aqui para manter o problema de planejamento é uma rede de Petri obtida pelo algoritmo Petriplan do capítulo 2. Esta rede possui algumas particularidades, a saber:

- É acíclica;
- Cada transição dispara no máximo uma vez;
- Um lugar possui no máximo dois arcos de saída.

Esta rede representa o comportamento dinâmico da aplicação de ações ao longo do tempo, partindo-se de uma situação inicial, em busca de um estado final. Resolver um problema de planejamento usando esta RdP significa encontrar todas as transições que devem ser disparadas, na ordem correta, para que, a partir de uma marcação inicial da RdP se consiga uma marcação que contenha o estado final. Esta seqüência de ações é chamada *plano*.

A Figura 4.1 mostra uma RdP com uma marcação inicial (círculos pretos nos lugares) e com uma marcação final (lugares com contornos mais fortes).

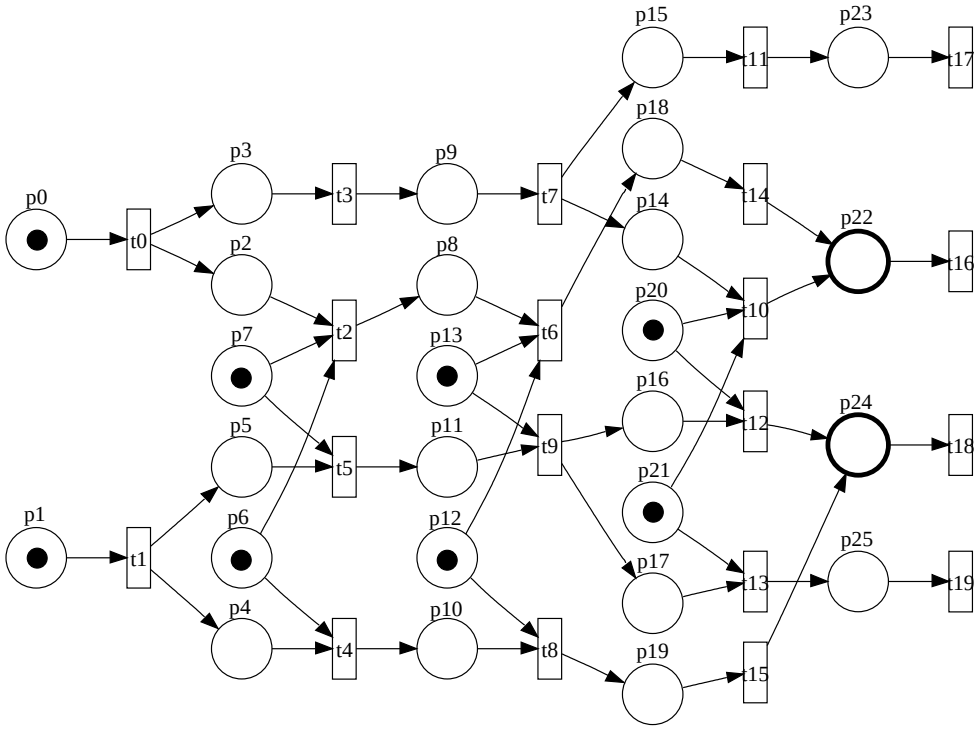


Figura 4.1: Problema completo com marcações

A Figura 4.2 mostra, para a rede da figura 4.1, uma seqüência de disparos de transições que fazem com que, a partir da marcação inicial se consiga uma marcação que contém a marcação final desejada. As transições disparadas estão preenchidas em cinza. Como os lugares objetivo possuem marcas após o disparo destas transições, tem-se um plano.

Para se chegar a um plano, pela análise de alcançabilidade em RdP, deve-se ter especial cuidado com os conflitos. Conflitos em uma RdP indicam que existem transições que não podem ser disparadas para a obtenção do mesmo plano. Pelo fato de serem transições mutuamente exclusivas, uma escolha errada pode levar a um processamento infrutífero,

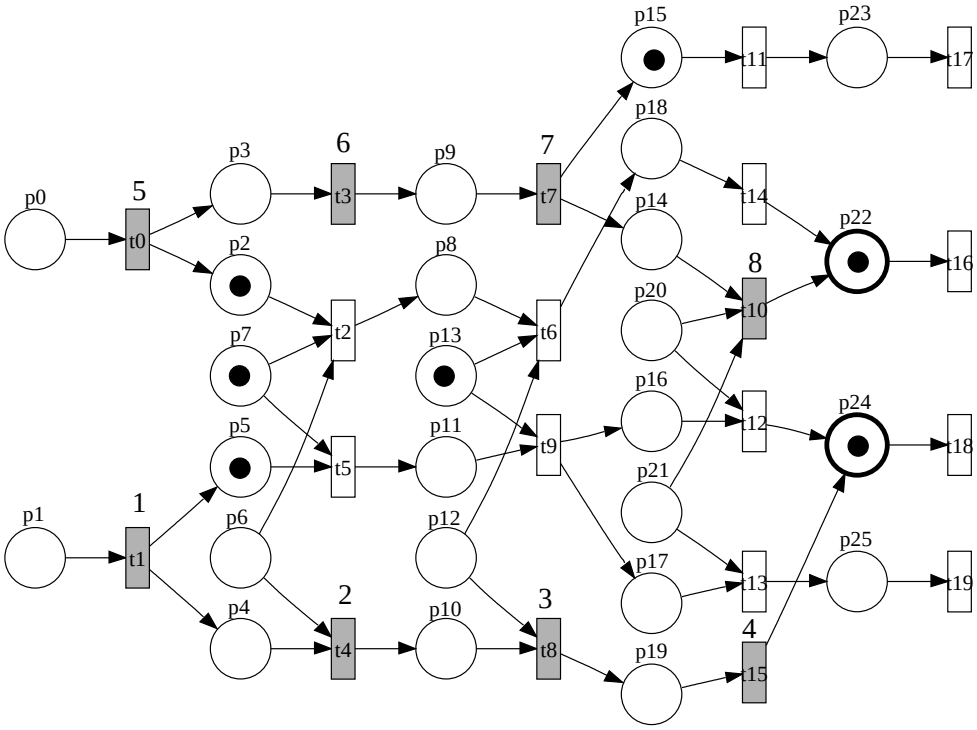


Figura 4.2: Uma possível solução

indicando que outra transição deveria ter sido escolhida para ser disparada. Assim, uma decisão consistente destes conflitos é dado pelo conjunto de escolhas feitas sobre quais transições deveriam ser disparadas em detrimento a outras.

Um conflito em uma RdP é dado por um lugar, ao qual estão conectados duas ou mais transições de saída, como na Figura 4.3.

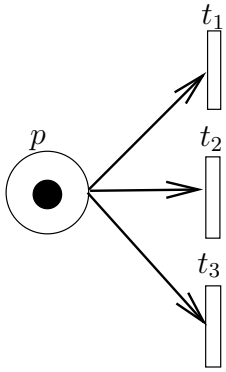


Figura 4.3: Um conflito em RdP

Estas escolhas são de extrema importância, pois se não for conseguida uma decisão consistente destes conflitos, significa que não existe um plano para o problema. Por outro lado, se esta configuração for obtida, as demais transições que não fazem parte de conflitos

não influenciam na obtenção do plano, pois seu disparo ou é obrigatório (por causa da dinâmica de transporte das marcas na RdP) ou seu disparo nunca ocorrerá.

O objetivo é resolver os conflitos de uma dada RdP baseando-se em SAT para se obter uma RdP livre de conflitos. Se isto ocorrer, para se encontrar as transições necessárias para alcançar o estado final a partir do estado inicial, bastaria simular seus disparos na rede sem que haja nenhuma escolha a ser feita.

4.1.1 Geração de NNF a partir de uma Rede de Petri

A técnica proposta é, partir da RdP original, partindo de cada lugar pertencente ao estado objetivo, varre-se a rede até os lugares iniciais construindo uma fórmula lógica onde seus átomos representam unicamente os conflitos. A retirada dos conflitos da rede se dá a partir de um resolvedor SAT para fórmulas não-clausais, que dará valores aos literais desta fórmula. Na estrutura da RdP, esta valoração indica a eliminação dos conflitos através da imposição de uma escolha.

Nas redes de Petri geradas pelo PERIPLAN usado nas implementações, os conflitos são sempre binários (um lugar possuindo no máximo dois arcos de saída), portanto pode-se usar uma variável proposicional para representar cada conflito, como mostra a Figura 4.4

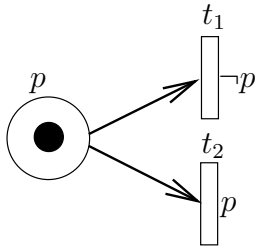


Figura 4.4: Conflito com a proposição α associada

Para cada possível disparo de transição que representa uma escolha, atribui-se um literal: $\neg\alpha$ indica o disparo da transição t_1 e α indica o disparo da transição t_2 na figura 4.4. Para estender esta representação para conflitos com mais de duas transições, usa-se $\lceil \lg_2 x \rceil$ variáveis proposicionais, onde x é o número de transições do conflito, como na Figura 4.5.

Analisando-se a estrutura da RdP, percebe-se que os lugares pertencentes ao estado final podem ser descritos como uma fórmula lógica baseada nas variáveis proposicionais

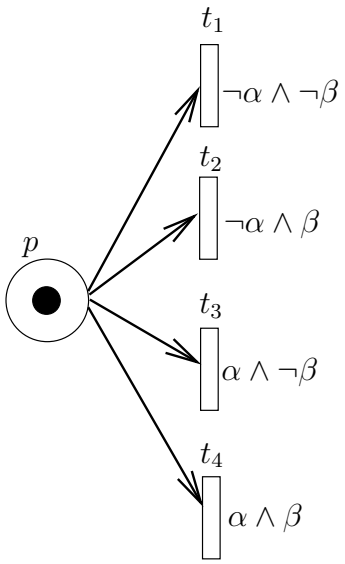


Figura 4.5: Conflito com mais de duas transições

associadas aos conflitos. A Figura 4.6 mostra a ocorrência de uma conjunção na RdP. Caso cada um dos lugares que chegam a t tenham fórmulas lógicas associadas $(\Delta_1, ..., \Delta_n)$, como mostra a Figura 4.7, a fórmula lógica associada a t é $\Delta_1 \wedge ... \wedge \Delta_n$.

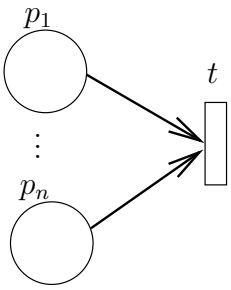


Figura 4.6: Conjunção na RdP

A Figura 4.8 mostra a ocorrência de uma disjunção na RdP. Caso cada uma das transições que chegam a p tenham fórmulas lógicas associadas $(\Delta_1, ..., \Delta_n)$, como mostra a Figura 4.9, a fórmula lógica associada a p é $\Delta_1 \vee ... \vee \Delta_n$.

Percebe-se que para saber a fórmula associada a um lugar, deve-se antes saber quais são as fórmulas associadas às transições que possuem arcos de entrada neste lugar. Analogamente para as transições. Para se conseguir as fórmulas associadas a cada objetivo, faz-se então um passeio na rede, a partir dos lugares objetivos, até os lugares que representam o estado inicial. Quando um lugar tem mais de um arco incidente, uma disjunção é gerada. Quando uma transição tem mais de um arco incidente, uma conjunção é gerada.

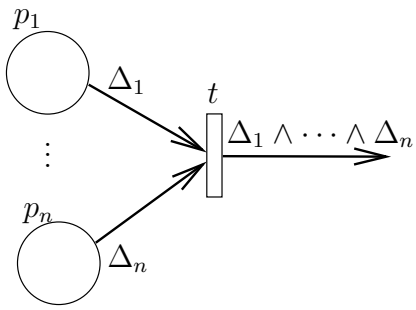


Figura 4.7: Conjunção na RdP com fórmulas

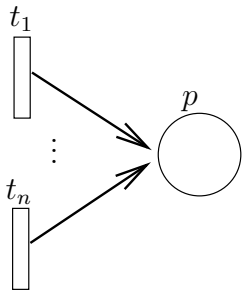


Figura 4.8: Disjunção na RdP

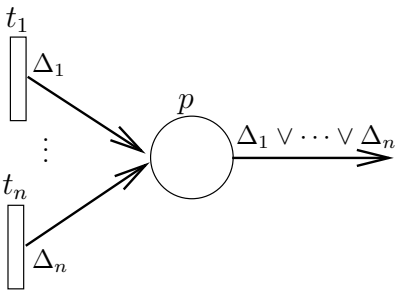


Figura 4.9: Disjunção na RdP com fórmulas

Só são geradas novas variáveis proposicionais quando forem encontrados conflitos.

Seja M_o a marcação final e M_i a marcação inicial da RdP. Seja também $\theta(V)$, definido sobre um vetor V, a quantidade de elementos de V diferentes de zero e $\mu(t)$, definido sobre uma transição t, indicando se t é uma transição de um conflito, isto é, $\mu(t)$ é $Verdadeiro$ se $\exists p,\ Pre(p,t)\ >\ 0\ \wedge\ \theta(Pre(p,.))\ >\ 1$ e $Falso$ caso contrário. Assume-se também que existe uma ordenação qualquer entre as transições, que pode ser a ordem de inserção na fórmula.

A fórmula F que representa os conflitos de uma RdP é dada por:

$$F\ \ =\ \bigwedge_i L(p_i),\ p_i \in M_o$$

$$L(p) = \begin{cases} \bigvee_i T(t_i);\ \forall t_i,\ Pos(p,\ t_i)\ >\ 0 & se\ \theta(Pos(p,\ .))\ \geq\ 1 \\ Verdadeiro & se\ \theta(Pos(p,\ .))\ =\ 0\ e\ p\ \in\ M_i \\ Falso & se\ \theta(Pos(p,\ .))\ =\ 0\ e\ p\ \notin\ M_i \end{cases}$$

$$T(t) = \begin{cases} \bigwedge_i \alpha(p_i,\ t);\ \forall p_i,\ Pre(p_i,\ t)\ >\ 0 & se\ \theta(Pre(.,\ t))\ \geq\ 1 \\ Falso & se\ \theta(Pre(.,\ t))\ =\ 0 \end{cases}$$

$$\alpha(p,\ t) = \begin{cases} L(p) & se\ \neg\mu(t) \\ \beta(p,t)\ \wedge L(p) & se\ \mu(t) \end{cases}$$

$$\beta(p,\ t) = \begin{cases} p & se\ \exists t', Pre(p,\ t')\ >\ 0\ e\ t\ >\ t' \\ \neg p & se\ \exists t',\ Pre(p,\ t')\ >\ 0\ e\ t\ <\ t' \end{cases}$$

Sua implementação se dá, para cada lugar objetivo, executando-se o procedimento *varre-lugar*, mostrado no Algoritmo 8.

Assim, cada lugar objetivo possui uma fórmula associada. Como se espera obter uma seqüência de disparos que alcance todos os lugares do estado final, assume-se que a fórmula lógica que representa o problema de planejamento é a conjunção das fórmulas dos lugares objetivo.

Algoritmo 8 varre-lugar

Entrada: lugar l
Saída: fórmula que representa l

```
formula ← vazio
for all t ∈ Pre(l, .) do
    formula ← formula ∨ varre-transição(t)
end for
if Pos(l, .) == 2 then
    formula ← l
end if
return formula
```

Algoritmo 9 varre-transição

Entrada: transição t
Saída: fórmula que representa t

```
formula ← vazio
for all l ∈ Pre(., t) do
    formula ← formula ∧ varre-lugar(l)
end for
return formula
```

Como essa fórmula é composta somente por literais que representam os conflitos na rede, qualquer modelo (uma valoração que a satisfaça) é interpretado como uma seqüência de disparos não conflitantes. Caso um modelo não seja encontrado, então a disposição dos conflitos não permite que os objetivos sejam alcançados, indicando que não existe um plano para o problema de planejamento associado.

Em virtude da estrutura de geração da fórmula, percebe-se que os únicos conectivos presentes são disjunções e conjunções. Também percebe-se que as negações possuem seu escopo limitado aos átomos das fórmula. Portanto, a fórmula gerada está na NNF [7].

Exemplificando sua aplicação, para a RdP da Figura 2.1 e para o lugar objetivo $p22$, deve-se percorrer a RdP da direita para a esquerda construindo a fórmula que o representa, como mostrado na Figura 4.10.

Assumindo que os conflitos possuem somente dois arcos de saída, convenciona-se que o arco mais acima na rede será o literal negativo (cujo nome é o nome do lugar) e o mais abaixo será positivo. Para a Figura 4.10 tem-se a seguinte formula lógica:

$$p22 = ((\neg p7 \wedge \neg p6) \wedge \neg p12 \wedge \neg p13) \vee (\neg p20 \wedge \neg p21)$$

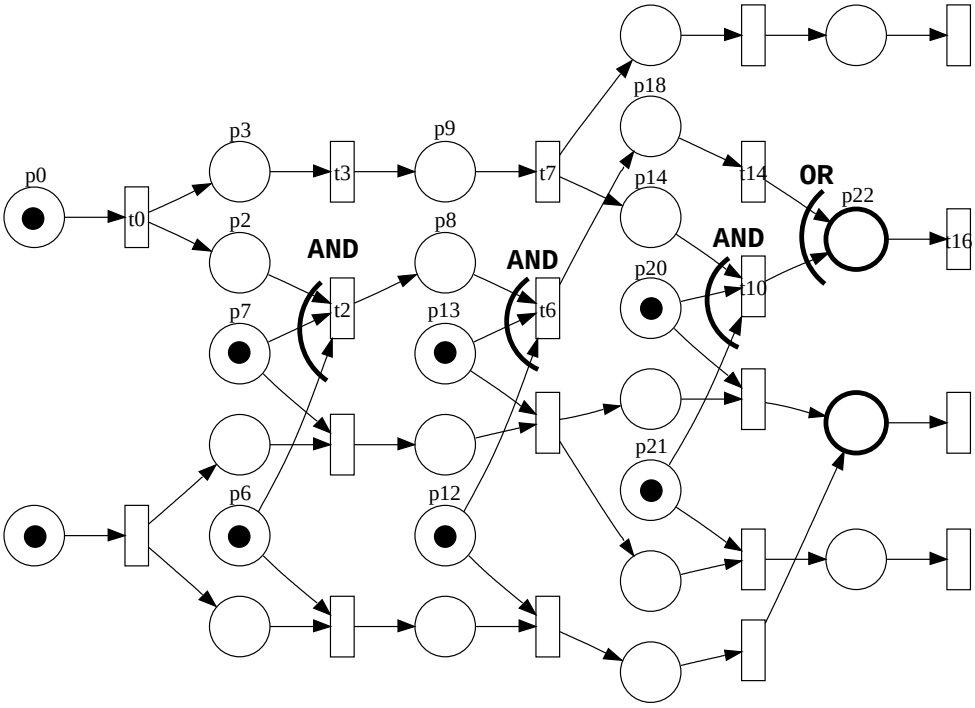


Figura 4.10: Caminhamento reverso para $p22$

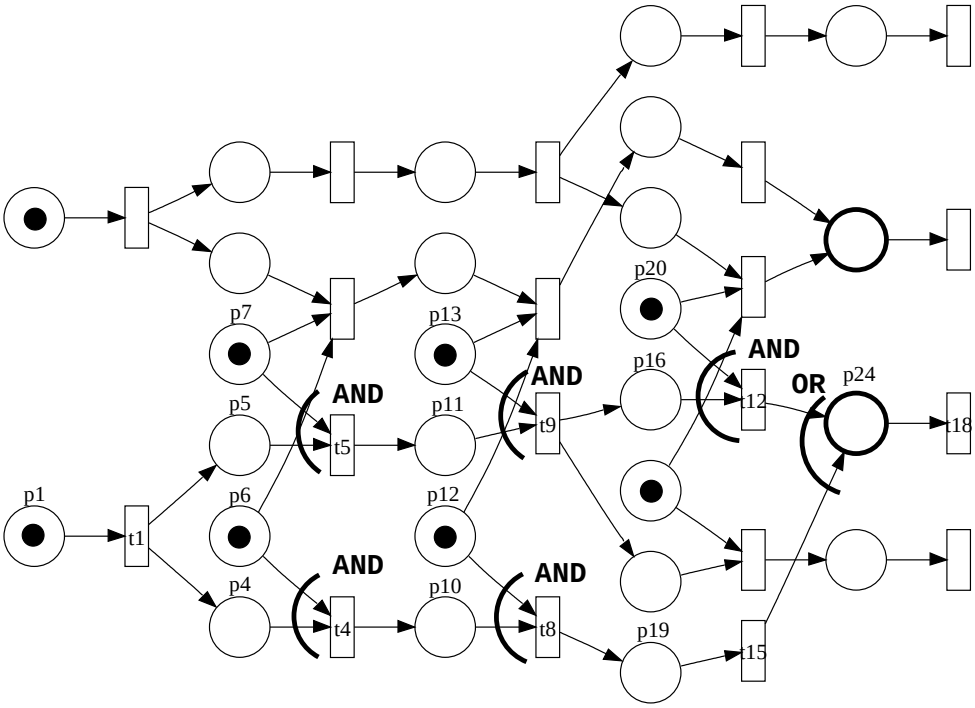


Figura 4.11: Caminhamento reverso para $p24$

A Figura 4.11 representa a aplicação do mesmo raciocínio para a obtenção da fórmula que representa o objetivo $p24$, resultando na seguinte fórmula:

$$p24 = (p6 \wedge p12) \vee (p7 \wedge p13 \wedge p20)$$

A fórmula completa que se quer resolver é $p22 \wedge p24$, gerando o DAG da Figura 4.12.

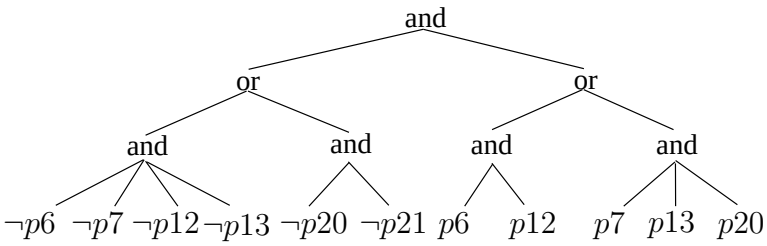


Figura 4.12: DAG para $p22 \wedge p24$

Como a fórmula gerada está na NNF, deve-se aplicar uma transformação para FNNF para que um procedimento eficiente de SAT seja aplicado. Para a fórmula $p22 \wedge p24$, a fórmula resultante na FNNF é a da Figura 4.13.

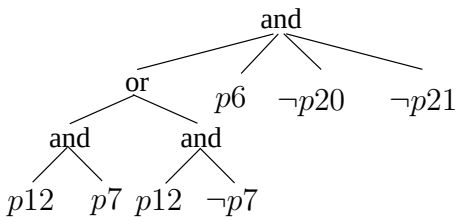


Figura 4.13: FNNF para $p22 \wedge p24$

Sobre a fórmula obtida pode-se aplicar um algoritmo de enumeração de modelos, a fim de se conseguir todas as valorações que satisfaçam a fórmula. Estas valorações, por sua vez, resolvem os conflitos que a rede possui indicando quais são os disparos, em detrimento a outros, que devem ser feitos para se conseguir uma marcação que contenha a marcação final, indicando um plano no problema de planejamento subjacente. Não há a necessidade de se obter todos os modelos possíveis, basta um para que um plano seja conseguido. Claramente, com este modelo qualquer não se pode garantir a qualidade do plano, isto é, se o plano é o menor possível ou um dos menores.

Para se encontrar este modelo (se existir) pode-se usar uma versão reduzida do algoritmo de enumeração de modelos mostrado na seção 3.3.5, o qual não aplica a união entre modelos (que acontece a cada disjunção), sendo que a cada disjunção encontrada, um ramo qualquer é percorrido. Este algoritmo é um resolvedor SAT para FNNF (d-DNNF).

Buscar os modelos na fórmula da Figura 4.13 torna-se uma tarefa muito simples, por causa das propriedades de determinismo e decomponibilidade da FNNF. Aplicando-se o algoritmo visto, consegue-se dois modelos:

- $\{p6, p7, p12, \neg p20, \neg p21\}$
- $\{p6, \neg p7, p12, \neg p20, \neg p21\}$

Cada um desses modelos indica que uma certa combinação dos conflitos leva a um caminho de disparos entre o estado inicial e o estado final da RdP, isto é, a solução de todos os seus conflitos. Com estes conflitos resolvidos, a rede não possui mais nenhum ponto de escolha, pois estas escolhas são feitas através dos modelos obtidos.

Logo após o resolvedor SAT ter encontrado uma ou mais valorações que satisfaçam a fórmula, essas valorações devem ser tratadas como cortes a serem efetuados na rede. Se um mecanismo de busca fizer uma varredura na rede em busca de um plano, então estes cortes eliminam completamente os pontos de escolhas, fazendo com que uma varredura simples na rede retorne o plano desejado.

Visto que o resolvedor SAT pode encontrar mais de uma valoração isso é interpretado como sendo a existência de mais de um plano para o problema de planejamento subjacente. Assim, qualquer modelo encontrado resolverá os conflitos, tornando possível a escolha por qualquer um.

Cada literal do modelo representa uma transição de um conflito. Se o literal é positivo (ou negativo), a transição que ele representa deverá continuar na rede e a outra deverá ser removida da rede, ou mesmo impedida de ser escolhida pelo procedimento que passeia na rede em busca do plano. O não aparecimento da variável no modelo, como é o caso de $p13$, indica uma situação de indiferença, não importando qual transição do conflito for considerada, o resultado será o mesmo.

Para os modelos conseguidos, tem-se as redes das Figuras 4.14 e 4.16. Os arcos marcados com uma linha mais forte são as escolhas feitas. As transições preenchidas em cinza são os disparos que serão feitos para se conseguir a marcação final. Para a variável $p13$ que não aparece nos modelos assume-se qualquer valoração, para que seja obtida uma rede livre de conflitos. Aqui, valora-se como $Verdadeiro$. A aplicação do corte indicado pelos arcos não usados na Figura 4.14 resulta na RdP da Figura 4.15

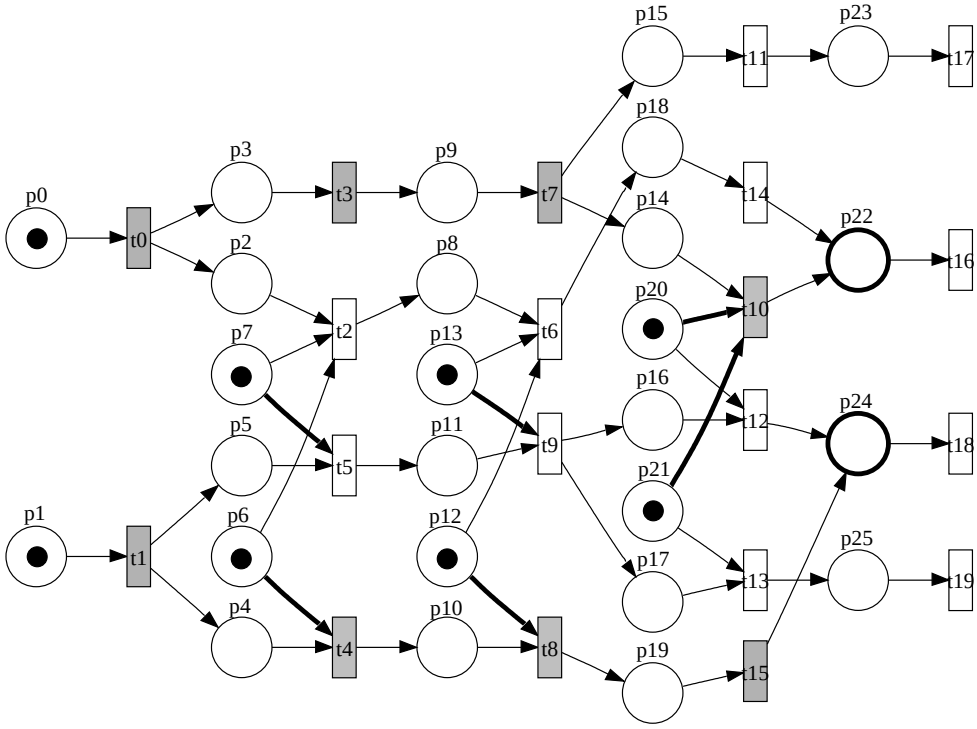


Figura 4.14: Cortes na rede para o primeiro modelo

Com os cortes efetuados na rede, faz-se então um procedimento de busca simples, exaustiva. O principal problema de um algoritmo de busca força-bruta, é a quantidade de *backtrackings* que devem ser feitos até que a solução seja encontrada. Como todas as possibilidades deverão ser tentadas no pior caso, a quantidade de tempo necessário para tal computação cresce exponencialmente com o tamanho do problema.

Com os cortes gerados pelo resolvedor SAT, o algoritmo de busca não terá mais nenhum tipo de *backtracking* a fazer, visto que todos os pontos de conflito já foram resolvidos, como podem ser vistos nas figuras das redes cortadas.

Este algoritmo de busca pode percorrer a rede da direita para a esquerda, partindo dos objetivos até encontrar a situação inicial, isto é, um lugar que tenha uma marca

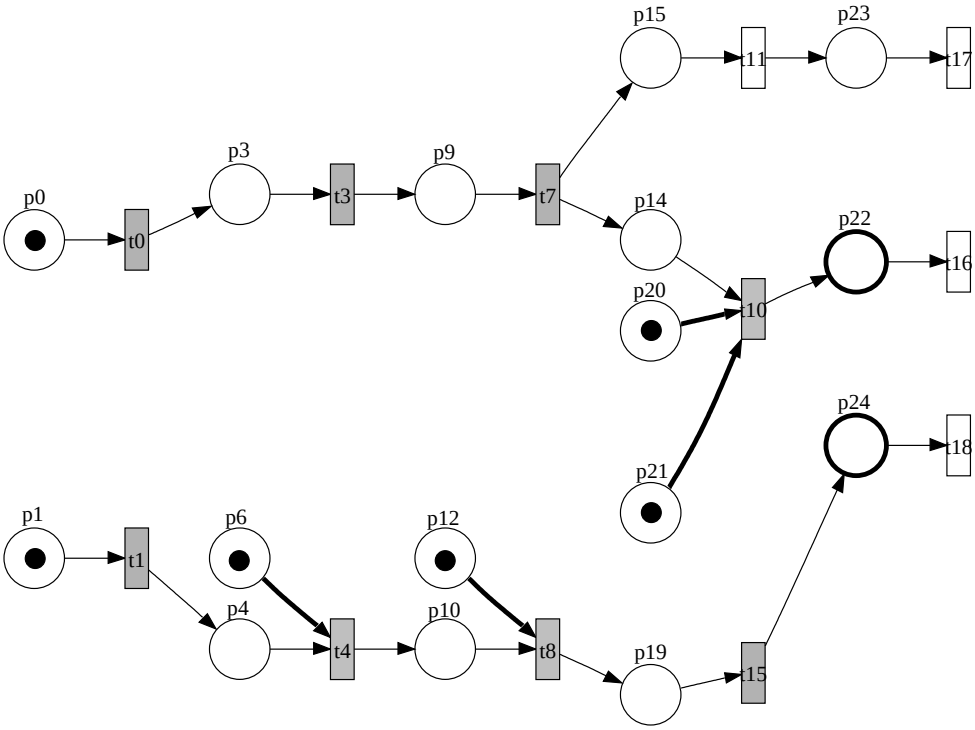


Figura 4.15: Rede de Petri cortada

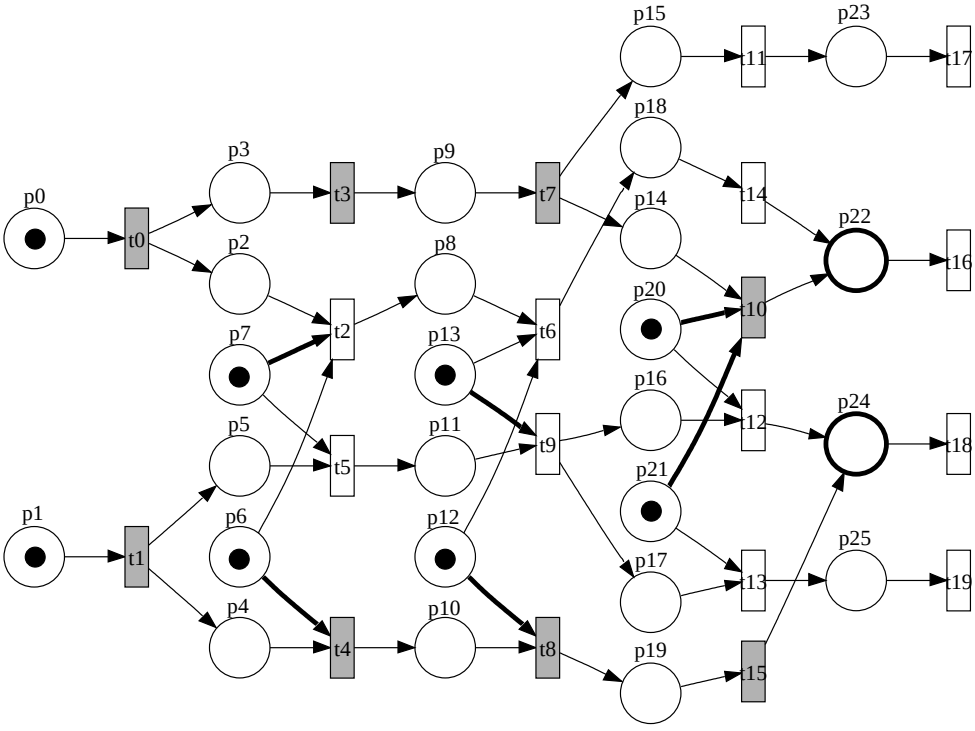


Figura 4.16: Cortes na rede para o segundo modelo

na marcação inicial da rede, com cada nodo visitado sendo armazenado em uma pilha, que no final da busca conterá todo o plano. Pode-se usar também um algoritmo que faz disparos na rede, armazenando cada disparo. Alguns disparos nunca acontecerão, visto que suas estruturas foram removidas da rede. Serão os conflitos resolvidos pela máquina SAT que farão com que os caminhos infrutíferos, aqueles que não levam ao plano, sejam descartados. A rede resultante não possui nenhum conflito, seja com qualquer um dos modelos obtidos pelo SAT.

Caminhar nesta rede sem conflitos é extremamente rápido, $\mathcal{O}(nm)$ (pior caso), onde n é o número de lugares e m é o número de trasições da rede. Após este caminhamento, tem-se a seqüência de transições a ser disparada para que se consiga chegar na marcação final a partir da marcação inicial. Assim, um plano que resolve o problema de planejamento representado pela RdP será encontrado.

Caso um modelo não seja encontrado, sabe-se que os conflitos são inconsistentes entre si, portanto, seguindo o algoritmo PETRIPLAN, deve-se realizar mais uma expansão da RdP.

4.2 Implementações

A seguir são mostrados as implementações feitas usando-se a nova estratégia de resolução do problema de planejamento usando-se formas não-clausais. Esta implementação foi feita no ambiente IPÊ, em C++, usando Linux como sistema operacional.

4.2.1 Soluções usando Condicionamento

A operação de condicionamento basicamente troca um literal em uma fórmula por um valor verdade (*Verdadeiro* ou *Falso*). Aplicações sucessivas do condicionamento, analogamente à aplicação da Expansão de Shannon, em cada variável da fórmula faz com que a fórmula original tenha todos os seus literais valorados e faz com que se crie uma nova fórmula, equivalente à original, mas que está na d-DNNF.

Dada uma fórmula Δ, aplicar condicionamento em Δ sobre o literal L, escrito como

$\Delta \mid L$ significa substituir de toda ocorrência de L por *Verdadeiro* e toda ocorrência de $\neg L$ por *Falso*.

Facilmente verifica-se a seguinte identidade (Expansão de Shannon):

$$\Delta \equiv [(\Delta \mid L) \wedge L] \quad \vee \quad [(\Delta \mid \neg L) \wedge \neg L]$$

A aplicação dessa identidade sobre todas as variáveis de Δ resulta em uma fórmula que está na d-DNNF [27]. A grande vantagem deste método é que a quantidade de vezes que se aplica a Expansão de Shannon é no máximo o número de variáveis da fórmula. O grande problema é o uso de memória excessivo, quando se tem fórmulas de tamanho considerável (40 variáveis ou mais). Percebe-se claramente que, no pior caso, esta identidade levaria à duplicação do espaço ocupado pela fórmula, mesmo que simplificações fossem efetuadas, pois a tendência é que as fórmulas obtidas nos passos intermediários ocupem muito espaço.

Em [10], Darwiche cita que as principais técnicas usadas para uma transformação eficiente de uma teoria em CNF para d-DNNF são a decomposição e o *caching*. Infelizmente estas técnicas são usadas levando-se em conta a estrutura de fórmulas na CNF. Como não se pode garantir que de uma RdP se extraia uma fórmula na CNF, estas técnicas não podem ser usadas aqui.

A implementação aqui mostrada é uma codificação simples e direta da aplicação sucessiva da operação de condicionamento. Como era de se esperar, a limitação de memória, aliada ao tamanho das fórmulas geradas, tornou esta implementação extremamente ineficiente, inviável até para resolução problemas simples[1], mesmo com as simplificações sendo aplicadas a cada condicionamento (geração da FNNF).

A implementação deste método se mostrou muito ineficiente em termos de utilização de memória. Acreditava-se que, pela adição de simplificações lógicas simples, o tamanho da fórmula resultante não aumentaria exponencialmente, sendo que a cada passo de condicionamento várias partes da fórmula seriam reduzidos a $Falso$ ou $Verdadeiro$. Na prática, o ganho com as simplificações não é significativo, visto que alguns dos problemas

[1]Para resolver o problema *gripper* com 4 bolas, a fórmula gerada tem 111 variáveis e todos os 4GB de memória da máquina são usados antes do condicionamento de 50 variáveis.

mais simples não tiveram seu resultado computado devido ao alto consumo de memória.

Uma alternativa foi a troca de recurso de memória por processamento. A idéia principal é fazer o condicionamento só com o literal L, ou seja, apenas com o literal positivo. Caso se consiga um ramo que, ao final de todos os condicionamento, se obtenha valor verdade *Verdadeiro*, então não há mais a necessidade de se obter a fórmula em d-DNNF inteira, pois já se tem em mãos um modelo para a fórmula original. Caso não se consiga *Verdadeiro*, faz-se o condicionamento para o literal $\neg L$. A ordem de escolha das variáveis a serem usados nos condicionamentos foi a mesma do outro método, isto é, a variável proposicional que mais ocorre na fórmula primeiro.

A vantagem desta abordagem é a não necessidade de se obter uma fórmula equivalente em FNNF. Caso se obtenha *Verdadeiro* em um ramo, um modelo já foi encontrado. A desvantagem é que, no pior caso, quando a fórmula não possui nenhum modelo que a satisfaça, todas as combinações são tentadas. Pelo próprio processo de implementação e representação, isto é pior que enumerar a tabela verdade da fórmula.

4.2.2 Tableau KE

Visto que a aplicação sucessiva de condicionamento leva tanto a problemas de memória como de processamento, outra técnica para gerar uma fórmula na d-DNNF ou até mesmo encontrar um modelo foi usada. Em [28], foi mostrado que um procedimento de tableau, conhecido como Tableau KE [5], pode ser usado para este fim.

A FNNF [28] foi introduzida para expressar a aplicação de simplificações no processo de condicionamento. Estas simplificações promovem vários cortes na fórmula, durante o processo de geração da d-DNNF, o que leva à geração de uma fórmula menor e equivalente. Caso a fórmula seja uma contradição, será reduzida a $Falso$. Analogamente, sendo uma tautologia, será reduzida a $Verdadeiro$. Esta representação em FNNF é canônica.

Este novo procedimento usa um fragmento do Sistema KE. KE é um sistema de refutação semelhante ao tableau clássico, mas, ao contrário deste, possui regra *cut*, conhecida como *Princípio da Bivalência* (PB), sendo a única regra de ramificação do tableau. Além disso, para a geração da FNNF, não se faz necessária a aplicação do procedimento

de refutação completo, basta a aplicação das regras de tableau até que se encontre um ramo saturado [2]. Estes ramos saturados serão marcados nas figuras com o símbolo $\odot$.

A este procedimento de tableau, um mecanismo de propagação de restrições é usado, conhecido como *simplificação*[23]. Este mecanismo tem o mesmo propósito que a subjugação para a resolução e que a regra de cláusulas unitárias para o procedimento de Davis-Putnam. A regra de simplificação aplicada a uma fórmula A com respeito a φ, $A[\varphi]$, pode ser vista da seguinte forma ($\sharp$ denota qualquer conectivo lógico):

$$A[\varphi] \rightarrow \begin{cases} Verdadeiro & se\ \varphi\ =\ t.A \\ Falso & se\ \varphi\ =\ f.A \\ \neg(B[\varphi]) & se\ A\ =\ \neg B \\ B[\varphi]\ \sharp\ C[\varphi] & se\ A\ =\ B\ \sharp\ C \\ A & caso\ contrário \end{cases}$$

As fórmulas em um tableau KE podem ser sinalizadas. A regra de simplificação acima trata dessa sinalização, onde $f.A$ indica que A possui o valor lógico $Falso$ e $t.A$ indica que A possui o valor lógico $Verdadeiro$.

De uma forma simples, esta simplificação é análoga ao condicionamento proposto por Darwiche [28]. Na aplicação do condicionamento $\Delta \mid L$ o resultado é uma fórmula Δ' onde toda ocorrência de L é trocada pelo valor lógico $Verdadeiro$ e toda ocorrência de $\neg L$ por $Falso$. Isso equivale a aplicar a regra de simplificação de Massacci como $\Delta[L]$ [28].

Assim, o procedimento de tableau KE pode ser visto como uma outra maneira de se aplicar os condicionamentos propostos para a geração de uma fórmula na d-DNNF. A aplicação sucessiva da simplificação faz com que se obtenha uma fórmula na FNNF.

A implementação do procedimento de tableau KE desenvolvida neste trabalho, baseia-se em uma árvore binária. Cada nodo do tableau é um nodo da árvore e a ele estão

[2]Um ramo saturado do tableau é um ramo onde nenhuma regra pode ser aplicada. Em especial, PB pode ser aplicada até que todas as variáveis sejam consumidas, mas, claramente, percebe-se que, se a única regra possível de ser aplicada for PB e todos os nodos não removidos do tableau possuem fórmulas unitárias, então o ramo está saturado. Um tableau é saturado quando todos seus ramos são saturados.

$$[(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)] \wedge [(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)]$$

Tabela 4.1: Tableau inicial

$$\checkmark[(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)] \wedge [(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)]$$
$$|$$
$$(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)$$
$$(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)$$

Tabela 4.2: Aplicação da regra α

associados os apontadores para outros nodos (filho esquerdo e filho direito) e uma fórmula lógica associada.

A fórmula NNF a seguir é a representação de todos os conflitos de um problema de planejamento muito simples:

$$[(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)] \wedge [(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)])$$

Para gerar a FNNF da fórmula acima, basta atribui-la ao nodo raiz do tableau e iniciar o processo até que um tableau saturado seja encontrado. Aqui, não há necessidade de se obter um tableau saturado pois qualquer ramo saturado deste tableau já é um modelo para a fórmula que está sendo processada. Para a fórmula acima, a Figura 4.1 mostra o tableau inicial, com a fórmula acima atribuída ao nodo raiz.

O procedimento verifica a possibilidade da aplicação da regra α, cujo resultado é o tableau da Figura 4.2. As fórmulas marcadas (com ✓) são fórmulas removidas do tableau, isto é, fórmulas que não são mais elegíveis para a aplicação de qualquer regra. O fato de estas fórmulas serem removidas do processo não implica a perda informação.

Neste ponto, verifica-se que não se pode aplicar a regra α. A simplificação resulta em uma fórmula idêntica à original, portanto, deve-se aplicar *cut* (PB). Para isso, deve-se estabelecer uma ordem na qual os *cuts* acontecerão. Na atual implementação, são usadas primeiro as variáveis que mais aparecem na fórmula original. Esta estratégia é usada supondo-se que variáveis que aparecem com mais freqüência que outras geram menos ramos, pela maior quantidade de variáveis trocadas por valores lógicos e por simplicações de tabela verdade. Atualmente este cálculo é feito na fórmula original, podendo levar à

$$\checkmark[(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)] \wedge [(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)]^{\#1}$$
$$|$$
$$(\neg p20 \wedge \neg p21) \vee (\neg p6 \wedge \neg p7 \wedge \neg p12 \wedge \neg p13)^{\#2}$$
$$(p7 \wedge p13 \wedge p20) \vee (p6 \wedge p12)^{\#3}$$
$$p13^{\#4} \quad \neg p13^{\#5}$$

Tabela 4.3: Aplicação do *cut*

aplicação de condicionamentos inúteis. Estes processamentos não necessários são suprimidos pela verificação de pertinência antes da aplicação do condicionamento propriamente dito. Neste ponto, um *cut* na variável proposicional $p13$ é efetuado, resultando em uma ramificação no tableau: para a esquerda os ramos onde considera-se $p13$ positivo e na direita os ramos onde considera-se $p13$ negativo ($\neg p13$). Obtém-se um tableau como o da Figura 4.3.

Um tableau ramificado como o da Figura 4.3 leva a uma escolha sobre qual ramo se deve continuar o processamento. Aqui decide-se sempre pelo literal positivo, sendo que esta decisão pode ser feita através de uma função heurística. A aplicação de *cut* indica que as fórmulas nos nodos acima ou não puderam ser resolvidos ou estão marcados como removidas. Como o produto do condicionamento são dois ramos com fórmulas unitárias a serem processadas, segundo o algoritmo apresentado, a regra mais relevante neste caso é a aplicação da simplificação de Massacci [23] que simplifica uma fórmula em virtude das fórmulas unitárias geradas.

O procedimento busca, partindo do nodo contendo a fórmula $p13$ até a raiz, por nodos cujas fórmulas contém a variável proposicional $p13$. Encontrado um nodo X que satisfaça a condição de pertinência, este é condicionado, o que significa que uma cópia de sua fórmula é gerada, onde toda ocorrência de $p13$ será trocada por $Verdadeiro$ e toda ocorrência de $\neg p13$ será trocada por $Falso$. Esta nova fórmula é atribuída a um novo nodo, que é inserido como filho do nodo que gerou o condicionamento. O nodo X é marcado como removido.

No tableau da Figura 4.3, percebe-se que existem duas fórmulas que possuem a variável proposicional $p13$ (fórmulas assinaladas com #2 e #3). Após o condicionamento destas duas fórmulas, obtém-se o tableau da Figura 4.4.

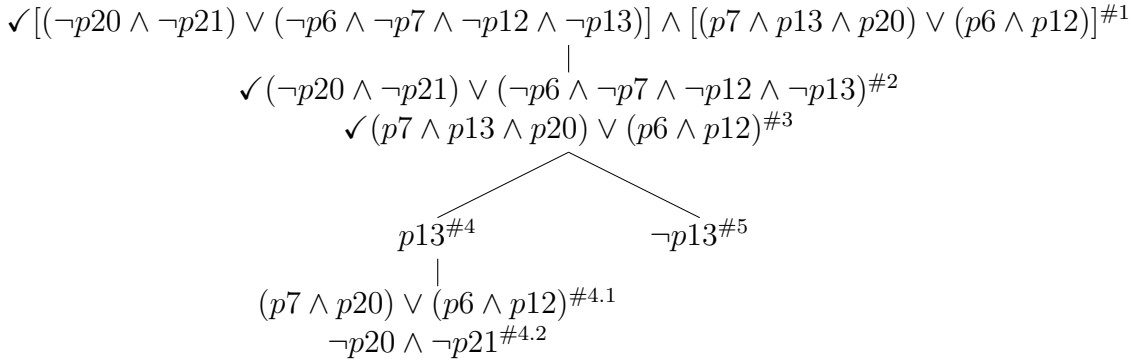


Tabela 4.4: Aplicação de condicionamento

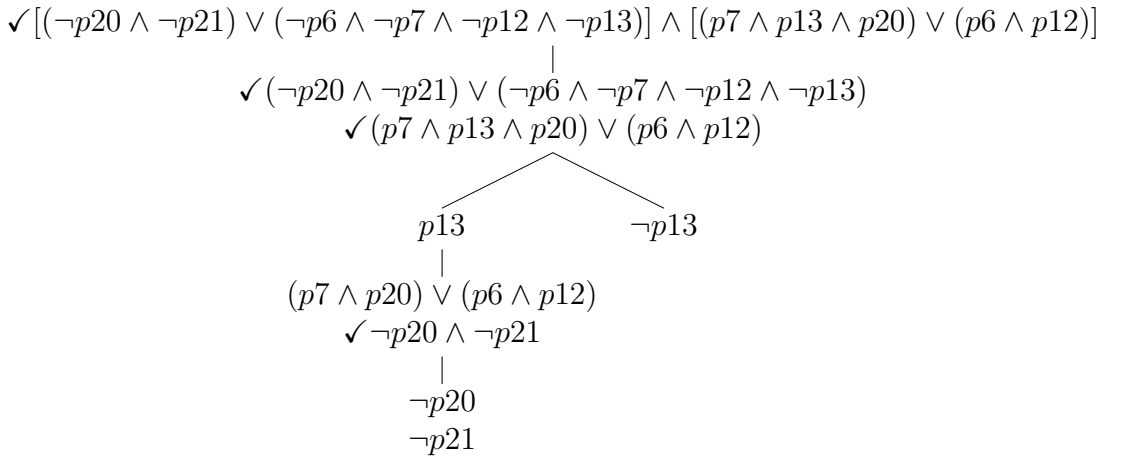


Tabela 4.5: Aplicação de regra α

Analisando este tableau, percebe-se que, das fórmulas geradas do condicionamento em $p13$, a primeira (#4.1) não está sujeita a nenhuma regra de simplificação, mas à segunda (#4.2) pode ser aplicada a regra α, obtendo-se então, o tableau da Figura 4.5.

Aplicando-se condicionamento em $\neg p20$, obtém-se o tableau da Figura 4.6.

Como o nodo contendo $\neg p20$ foi processado, está claro que ao próximo nodo resta a aplicação da regra α, gerando o tableau da Figura 4.7.

No tableau da Figura 4.7 existe um ramo saturado. Este ramo do tableau é um modelo para a fórmula original, portanto não faz-se necessário continuar o processo até que seja obtida a fórmula FNNF completa. Buscando este ramo no tableau e seguindo sua estrutura, obtém-se a seguinte fórmula:

$$p13 \wedge \neg p20 \wedge p6 \wedge p12 \wedge \neg p21$$

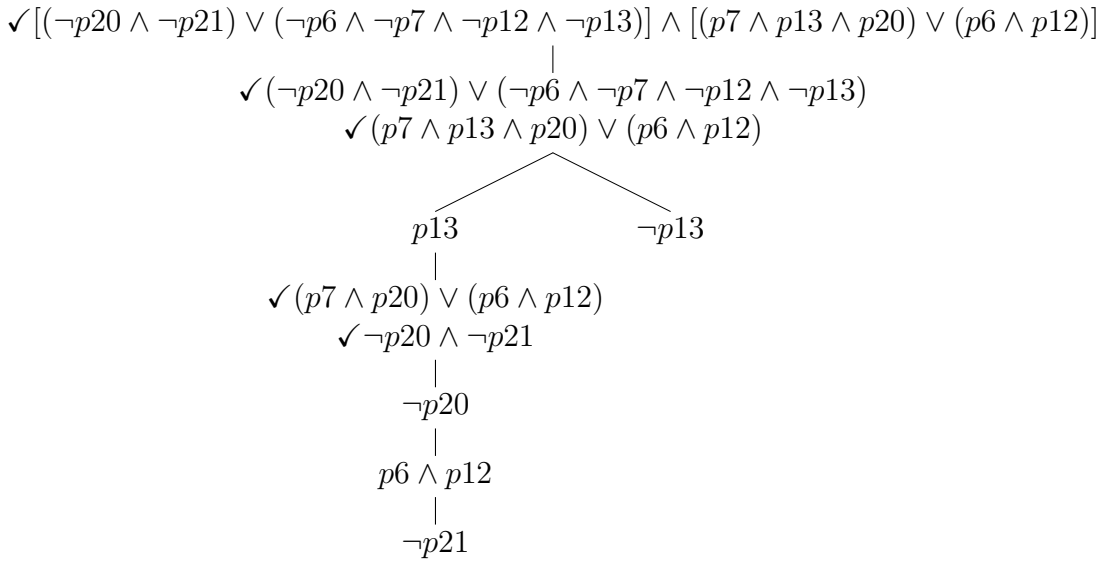


Tabela 4.6: Aplicação de condicionamento

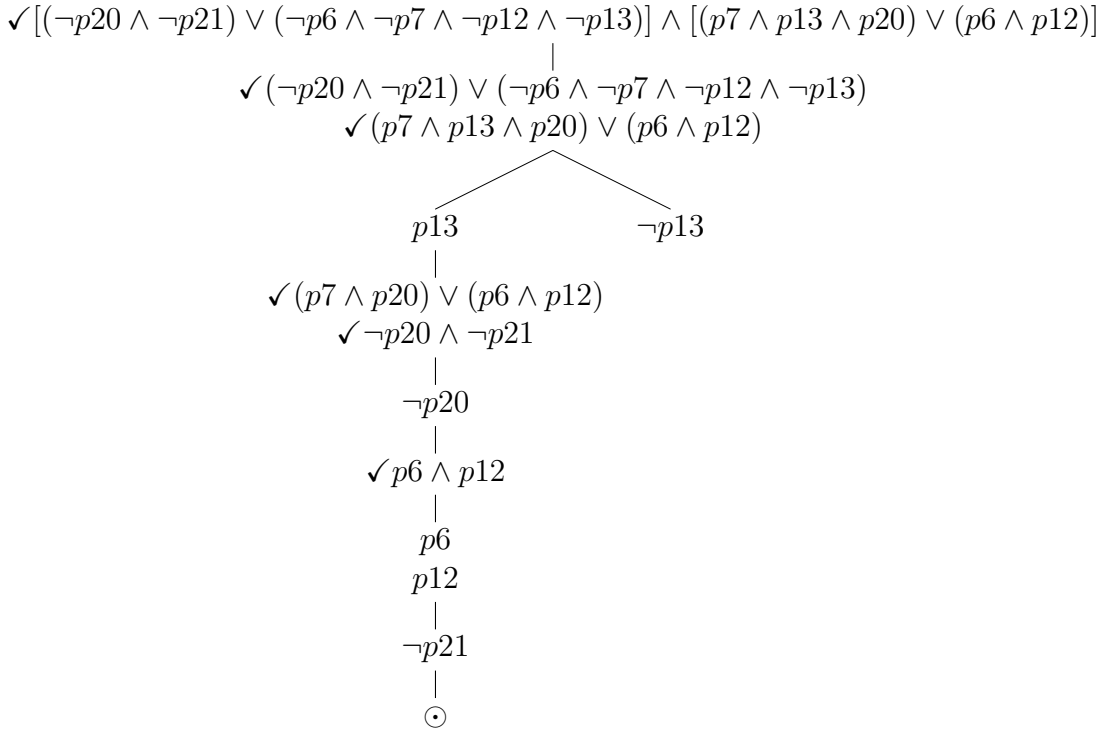


Tabela 4.7: Aplicação de regra α, tableau saturado

Aplicando-se a valoração $p13 = Verdadeiro$, $p20 = Falso$, $p6 = Verdadeiro$, $p12 = Verdadeiro$ e $p21 = Falso$ à fórmula original e efetuando-se as simplificações apropriadas, obtém-se o valor lógico $Verdadeiro$, indicando que $p13 \wedge \neg p20 \wedge p6 \wedge p12 \wedge \neg p21$ é um implicante (se todas as variáveis proposicionais estivessem presentes seria um modelo) para a fórmula original, e por conseguinte, é uma combinação de valores para os conflitos que faz com que não haja inconsistência entre eles.

O algoritmo usado para a implementação do procedimento de Tableau KE é visto no Algoritmo 10.

Algoritmo 10 solve

```
if nodo deletado then
  return filho.solve()
end if
if nodo com valor lógico then
  return valor lógico
end if
if nodo com fórmula unitária then
  condicionamento()
  if ramo do tableau é composto só por fórmulas unitárias then
    return indicando solução
  end if
  if conseguiu condicionar then
    return filho.solve()
  else if nodo possui filho then
    filho.solve()
  else
    cut()
    if não conseguiu aplicar cut then
      return indicando solução
    else
      filho.solve()
    end if
  end if
else
  regraAlfa()
  if conseguiu aplicar regra α then
    filho.solve()
  else if tem algum filho then
    filho.solve()
  else
    cut()
    if conseguiu aplicar cut then
      filho.solve()
    else
      return indicando solução
    end if
  end if
end if
Resolve simplificações de Tableau
```

Algoritmo 11 condicionamento

```
A ← Nodo Atual
x ← Literal de A
for all nodo N acima do nodo atual do
  if N for nodo deletado then
    Tenta próximo nodo
  end if
  if N tem uma fórmula unitária igual a ¬x then
    return  indicando ramo inconsistente
  end if
  if N tem uma fórmula não unitária que contém x ou ¬x then
    Copia fórmula em N
    Aplica simplificação de Massacci na fórmula copiada
    Adiciona nova fórmula em um nodo filho de A
    Marca N como deletado
  end if
end for
```

Algoritmo 12 cut

```
Obtém a próxima variável que não foi condicionada: p
Cria dois nodos filho no nodo atual: p e ¬p
```

Algoritmo 13 regraAlfa

```
if conectivo raiz da fórmula for ∧ then
  Obtém o lado esquerdo do ∧: β1
  Obtém o lado direito do ∧: β2
  Insere β1 como filho do nodo atual
  Insere β2 como filho do nodo contendo β1
  Marca nodo atual como deletado
else
  return  falha na aplicação da regra
end if
```

CAPÍTULO 5

AVALIAÇÃO EXPERIMENTAL

Neste trabalho foram expostas várias técnicas que juntas se propõe a resolver o problema de planejamento. De um arquivo PDDL, descritivo de um problema de planejamento, obtém-se uma Rede de Petri para que, a partir desta, se resolva este problema. Aqui foi proposto um novo método usando cálculo proposicional e formas normais não clausais, usadas em compilação de conhecimento. A Figura 5.1 mostra um resumo das implementações feitas neste trabalho.

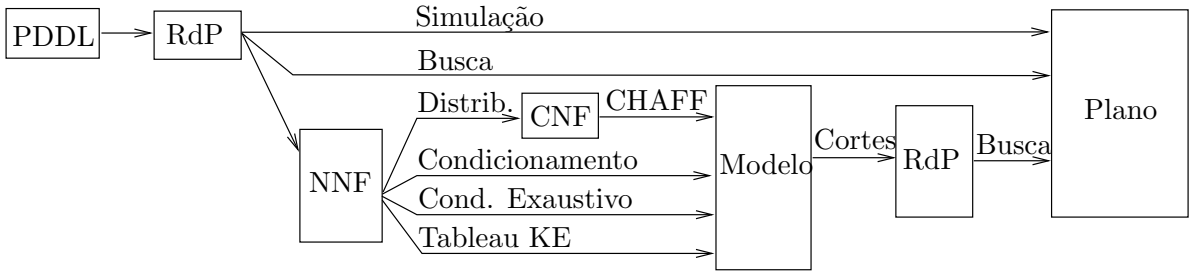


Figura 5.1: Modelo do trabalho

Ao todo foram implementados seis métodos de solução do problema de planejamento, a saber:

- **Simulação**: aplicação da equação fundamental, baseada em operações matriciais;
- **Busca exaustiva**: caminhamento na RdP, construindo-se uma árvore e armazenando as escolhas feitas para cada conflito;
- **SAT com CNF**: geração de uma fórmula na NNF a partir da RdP, sua conversão para CNF e aplicação de um resolvedor SAT para CNF;
- **Condicionamento**: aplicação de condicionamentos sucessivos em uma fórmula NNF gerada a partir da RdP;
- **Condicionamentos parciais**: aplicação dos condicionamentos de forma parcial, em profundidade em uma fórmula NNF gerada a partir da RdP;

- **Tableau KE**: aplicação de Tableau KE e regra de simplificação em uma fórmula NNF gerada a partir da RdP.

A implementação da *Simulação* se baseia em aplicar a dinâmica da RdP, isto é, dado um conjunto de transições disparáveis em uma RdP, dispara-se estas transições e verifica-se se a marcação resultante inclui a marcação objetivo. Caso inclua, pára-se o processo indicando o plano encontrado (o plano é a seqüência de transições disparadas). Caso não inclua, repete-se o processo até encontrar a marcação objetivo ou até não ser possível o disparo de mais nenhuma transição. Os disparos das transições são feitos através da equação fundamental das RdP.

Percebe-se que este processo é lento, em vista da quantidade de operações matriciais necessárias para se obter um plano.

A implementação da *Busca* é baseada na construção de uma árvore de caminhamento na RdP, iniciando-se nos objetivos até chegar nos lugares iniciais (estado inicial e conflitos). A partir de um estado objetivo, varre-se a rede buscando um caminho que leve até os estados iniciais. Quando se está processando um lugar, entende-se que se houver caminho por qualquer um dos arcos que chegam neste lugar, então um caminho é encontrado. Quando se está processando uma transição, entende-se que deve-se encontrar caminho para os lugares iniciais a partir de todos os arcos que chegam até esta transição. É importante notar que, nos casos em que o problema possui mais de um lugar objetivo, os caminhos obtidos quando se busca por um objetivo devem ser consistentes com os demais, o que pode levar a muitos *backtrackings*.

A implementação usando SAT clássico se baseia nos métodos tradicionais para solução de SAT, através de uma fórmula lógica na CNF. Para se obter uma fórmula lógica a partir da rede usou-se um caminhamento dos objetivos até os lugares iniciais. A cada transição, uma conjunção é gerada e a cada lugar, uma disjunção. Quando um lugar tem dois arcos de saída, trata-se de um *conflito*, portanto criando-se uma variável proposicional. Com este procedimento, se obtém uma fórmula lógica que representa todos os *conflitos* da rede e suas dependências.

Como pôde-se observar (Sessão 4.1.1), a fórmula gerada a partir do caminhamento na

RdP não é clausal e faz-se necessário sua transformação para CNF para que um resolvedor SAT clássico possa tratá-la. Este passo de transformação demanda a aplicação da regra distributiva dos conectivos lógicos, que leva, no pior caso, à um crescimento exponencial do tamanho da fórmula original inviabilizando assim sua aplicação para problemas até mesmo pequenos. Além disso, sabe-se que SAT em CNF é NP-Completo, mas o uso de um resolvedor conhecido e eficiente torna esta parte do processo menos onerosa.

Ao se estudar a fórmula gerada pela varredura da RdP, percebeu-se que ela está em uma forma normal conhecida (NNF). Sabendo-se que uma outra forma normal (FNNF), especializada da NNF, apresenta características que tornam a enumeração de modelos uma operação polinomial, o trabalho foi direcionado para o uso de técnicas de conversão de fórmulas lógicas na NNF para FNNF, de modo a contornar o problema de complexidade trazido pela transformação para CNF.

A primeira técnica estudada foi o condicionamento proposto por Darwiche, o qual é baseado na identidade conhecida como Expansão de Shannon. Condicionamento é uma maneira de valorar todos os literais pertencentes à fórmula, de tal forma que, ao ser aplicado para todas as variáveis proposicionais, consegue-se uma fórmula na d-DNNF. Esta forma normal apresenta propriedades na sua estrutura que tornam a operação de enumeração de modelos mais eficiente, não apresentando a mesma complexidade que para as fórmulas na CNF. Se na geração da d-DNNF forem aplicadas simplificações baseadas nas tabelas verdades da disjunção e conjunção, o que se consegue é uma fórmula equivalente na FNNF, que possui as mesmas propriedades que a d-DNNF, mas na prática gasta menos espaço para ser armazenada.

Neste trabalho, abordou-se essa transformação de três formas distintas: em uma delas aplicando condicionamento e duplicando a fórmula a cada passo, em outra aplicando condicionamento em uma recursão à esquerda e na última através do Tableau KE.

A primeira se tornou inviável, visto que as fórmulas mais simples geradas contêm milhares de nodos. Este processo, apesar da aplicação das simplificações baseadas em tabela verdade, causa uma explosão no uso de memória por causa da aplicação do condicionamento, que em sua essência, faz uma duplicação da fórmula com alguns literais

valorados.

A segunda forma de aplicação do condicionamento não faz uso de tantos recursos de memória, visto que não há duplicação de fórmulas. Basicamente, escolhe-se um literal a ser condicionando. Ao invés de se aplicar todo o condicionamento, aplica-se só metade da identidade com o literal positivo, conseguindo-se uma nova fórmula. Caso esse caminho seja infrutífero, um procedimento de *backtracking* fará o processo retornar a este ponto para que o condicionamento com o literal negativo seja aplicado. O processo continua até que um modelo seja encontrado (valoração para todas as variáveis proposicionais) ou até que o procedimento tente todas as possibilidades.

Em contrapartida ao uso de memória da outra variante, o tempo levado para execução deste procedimento é muito grande. No pior caso, onde uma fórmula gerada não é satisfatível (portanto não contém um modelo e o problema de alcançabilidade subjacente não é resolvido) são tentadas todas as combinações possíveis de valores verdade para as variáveis proposicionais.

O procedimento de tableau KE não faz uso excessivo de memória como a primeira solução, nem de tempo de processamento como a segunda. Em vista das regras aplicadas às fórmulas, o procedimento faz uso das estrutura destas para evitar que alguns condicionamentos sejam efetuados em fórmulas muito grandes, reduzindo assim tempo e espaço de armazenamento. Entretanto, pela enorme quantidade de nodos presentes nas fórmulas geradas a partir das RdP, até mesmo este procedimento se mostra inviável.

Convém ressaltar que apesar do estudo ter sido direcionado para que a operação de enumeração de modelos seja aplicada, os procedimentos param assim que o primeiro modelo é encontrado, pois este já é um plano.

5.1 Complexidade

Para o problema de planejamento, as características da linguagem de representação influenciam diretamente na complexidade computacional do problema, implicando desde tempo constante até EXPTIME-Completo[2]. Na representação STRIPS, a complexidade do problema de planejamento é PSPACE-Completo, que é a classe dos problemas mais

difíceis da classe PSPACE. A classe PSPACE é a classe dos problemas que são aceitos por uma máquina de Turing determinística usando apenas um número polinomial de células da fita da máquina durante a computação, em relação ao tamanho da entrada. Na prática, os algoritmos para problemas desta classe tratam espaços de estados exponenciais.

Assim, os métodos baseados em Simulação e Busca Exaustiva tentam, no pior caso, todas as possibilidades de resolução do problema.

Resolver SAT é NP-Completo[3]. Apesar desta indicação de que o problema é intratável, muitos resolvedores SAT hoje em dia resolvem problemas de forma muito rápida, baseados em heurísticas e técnicas bastante avançadas. Esses resolvedores recebem como entrada uma fórmula na CNF, portanto uma conversão se faz necessária. A conversão de uma fórmula lógica, usando-se o método clássico, pode levar a um número exponencial de cláusulas geradas.

A aplicação do Condicionamento é efetuado em um número constante de passos, em relação ao número de variáveis proposicionais da fórmula, mas a cada passo, a fórmula toda é varrida e copiada, fazendo-se as substituições necessárias. No pior caso, a fórmula é duplicada a cada passo. O Condicionamento Parcial usa menos memória em seu processamento, mas, no pior caso, se reduz a um método exaustivo de tentativa e erro, portanto exponencial.

O Tableau KE é PSPACE[28].

5.2 Domínios de Teste

Para os testes que serão relatados a seguir foram usados 3 domínios distintos: Simples, Mundo dos Blocos e Gripper. O domínio simples foi concebido para gerar um problema pequeno, com uma RdP não muito grande de fácil estudo e testes. Abaixo a descrição do domínio:

```
(define (domain simples-strips)
   (:predicates (pre-um ?x - objeto)
                (pre-dois ?x - objeto)
```

```
                (pos-um ?x - objeto)
                (pos-dois ?x - objeto))

  (:action acao-um
      :parameters  (?x - objeto)
      :precondition (and  (pre-um ?x))
      :effect (and  (not (pre-dois ?x))
                    (pos-um ?x)))

  (:action acao-dois
      :parameters (?x - objeto)
      :precondition  (and  (pre-dois ?x))
      :effect (and (pos-dois ?x))))
```

O Mundo dos Blocos é um domínio clássico em planejamento, gerando problemas interessantes para testes em planejadores. Basicamente, tem-se uma mesa com vários blocos dispostos de alguma forma. O objetivo é conseguir uma certa configuração desses blocos, mudando sua posição através de empilhamento e desempilhamento. Aqui foi usado como problema a Anomalia de Sussman para testes. Abaixo a descrição do domínio e do problema:

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
           (ontable ?x - block)
           (clear ?x - block)
           )

  (:action stack
```

```
        :parameters (?x - block ?y - block)
        :precondition (and (clear ?x)
                           (ontable ?x)
                           (clear ?y) )
        :effect
        (and  (not (clear ?y))
              (not (ontable ?x))
          (on ?x ?y)))
 (:action unstack
        :parameters (?x - block ?y - block)
        :precondition (and (on ?x ?y)
                           (clear ?x))
        :effect
        (and   (clear ?y)
           (clear ?x)
           (ontable ?x)
          (not (on ?x ?y)))))

(define (problem BLOCKS-SUSSMANANOMALY)
   (:domain BLOCKS) (:objects B A C - block)
   (:INIT  (CLEAR B)
           (ONTABLE B)
           (ONTABLE A)
           (ON C A)
           (CLEAR C))
   (:goal (AND (ON C B)
               (ON B A))) )
```

O Gripper modela um domínio contendo um robô com uma ou duas garras, e duas salas. O objetivo é levar algumas bolas de uma sala a outra. Abaixo a descrição do modelo e do problema utilizado:

```
(define (domain gripper-strips)
   (:predicates (at-robby ?r - room)
        (at ?b - obj ?r - room)
        (free ?g - gripper)
        (carry ?o - obj  ?g - gripper))

   (:action move
       :parameters  (?from - room ?to - room)
       :precondition (and  (at-robby ?from) )
       :effect (and  (at-robby ?to)
             (not (at-robby ?from))))

   (:action pick
       :parameters (?obj - obj ?room - room ?gripper - gripper)
       :precondition  (and  (at ?obj ?room)
                            (at-robby ?room)
                            (free ?gripper))
       :effect (and (carry ?obj ?gripper)
                    (not (at ?obj ?room))
                    (not (free ?gripper))))

   (:action drop
       :parameters  (?obj - obj   ?room - room ?gripper - gripper)
       :precondition  (and  (carry ?obj ?gripper) (at-robby ?room))
       :effect (and (at ?obj ?room)
            (free ?gripper)
            (not (carry ?obj ?gripper)))))

 (define (problem strips-gripper-x-1)
   (:domain gripper-strips)
```

```
(:objects rooma roomb - room
          ball2 ball1 - obj
          left right - gripper)
(:init (at-robby rooma)
       (free left)
       (free right)
       (at ball2 rooma)
       (at ball1 rooma))
(:goal (and (at ball2 roomb)
            (at ball1 roomb))))
```

5.3 Experimentos

Os experimentos foram realizados em uma máquina dual Intel(R) Xeon(TM) CPU 2.80GHz e 6GB RAM. Os problemas mostrados aqui foram os que terminaram sua execução e não pararam por consumir todos os recursos da máquina.

Apesar de serem usadas técnicas para compilação da fórmula numa forma onde SAT se torna eficiente, esta compilação se mostrou como sendo o grande gargalo de processamento. Em virtude do tamanho das fórmulas geradas e dos algoritmos pouco aprimorados, os tempos de execução e uso de memória não foram satisfatórios. Pesquisas específicas nestas conversões são necessárias para que um planejador usando esta abordagem seja eficiente e consiga tratar de problemas maiores.

A seguir são apresentadas as tabelas com tempos de processamento para os problemas testados. Na Tabela 5.1 tem-se os tempos para o método de Busca Reversa. Na Tabela 5.2 tem-se os tempos conseguidos para o método de Condicionamento segundo Darwiche. Na Tabela 5.3 tem-se os tempos para o método de Condicionamentos Parciais, proposto a partir do Condicionamento. Na Tabela 5.4 tem-se os tempos conseguidos usando-se Tableau KE.

A Tabela 5.5 mostra um comparativo entre os quatro métodos mais relevantes de resolução do problema de planejamento com Redes de Petri. Os tempos são representados

Problema	Domínio	Tempos	
Simples		Busca:	0.0000
		TOTAL:	0.0000
Sussman	Blocos	Busca:	0.0100
		TOTAL:	0.0100
2G 2B	Gripper	Busca:	0.0100
		TOTAL:	0.0100

Tabela 5.1: Resultados: Busca Reversa. 2G 2B indica o problema Gripper com 2 Garras e 2 Bolas.

Problema	Domínio	Tempos	
Simples		Geração da NNF:	0.0000
		Conversão dDNNF:	0.0000
		Enumeração de Modelos:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.0000
Sussman	Blocos	Geração da NNF:	0.0300
		Conversão dDNNF:	0.4100
		Enumeração de Modelos:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.4400
2G 2B	Gripper	Geração da NNF:	0.0800
		Conversão dDNNF:	MEM
		Enumeração de Modelos:	
		Eliminação de Conflitos:	
		Busca na Rede:	
		TOTAL:	

Tabela 5.2: Resultados: Condicionamento. MEM indica que o procedimento esgotou os recursos de memória antes de terminar.

Problema	Domínio	Tempos	
Simples		Geração da NNF:	0.0000
		Aplicação Condicionamentos:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.0000
Sussman	Blocos	Geração da NNF:	0.0300
		Aplicação Condicionamentos:	0.0100
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.0400
2G 2B	Gripper	Geração da NNF:	0.0300
		Aplicação Condicionamentos:	0.0000
		Eliminação de Conflitos:	0.0100
		Busca na Rede:	0.0000
		TOTAL:	0.0400

Tabela 5.3: Resultados: Condicionamentos parciais

Problema	Domínio	Tempos	
Simples		Geração da NNF:	0.0000
		Saturação do Tableau KE:	0.0000
		Busca por Modelo no Tableau:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.0000
Sussman	Blocos	Geração da NNF:	0.0200
		Saturação do Tableau KE:	0.0100
		Busca por Modelo no Tableau:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0100
		TOTAL:	0.0400
2G 2B	Gripper	Geração da NNF:	0.0400
		Saturação do Tableau KE:	0.0100
		Busca por Modelo no Tableau:	0.0000
		Eliminação de Conflitos:	0.0000
		Busca na Rede:	0.0000
		TOTAL:	0.0500

Tabela 5.4: Resultados: Tableau KE

Método	Busca Reversa	Condicionamento	Cond. Parciais	Tableau KE
Simples	0.0000	0.0000	0.0000	0.0000
Sussman	0.0100	0.4400	0.0400	0.0400
Gripper 2G2B	0.0100	MEM	0.0400	0.0500

Tabela 5.5: Resultados: Comparativo entre métodos. MEM indica que o procedimento esgotou os recursos de memória antes de terminar.

em segundos.

Com os tempos conseguidos em cada um destes métodos, algumas conclusões podem ser obtidas. Primeiramente, os algoritmos para obtenção da FNNF aqui apresentados estão preparados para obter estas fórmulas a partir de qualquer RdP acíclica. Faz-se necessário um estudo mais aprofundado para que se possa descobrir particularidades nas RdP que possam ser usadas em favor destes algoritmos.

As causas da explosão no uso de memória se dá por causa da aplicação do condicionamento, que em sua essência, faz uma duplicação da fórmula com alguns literais valorados. O uso de outras técnicas de transformação não fazem uso excessivo de memória, e poderiam levar à solução satisfatória da transformação de NNF para FNNF.

Finalmente, por causa da grande complexidade de espaço exigida no processo, somente problemas pequenos podem ser resolvidos. A implementação de técnicas mais elaboradas fará com que esta abordagem possa ser usada para resolver problemas de planejamento maiores.

CAPÍTULO 6

CONCLUSÃO

Este trabalho se situa na intersecção entre quatro grandes áreas de pesquisa: Planejamento, Redes de Petri, SAT e Compilação do Conhecimento. O objeto de estudo são os problemas de planejamento, representados através de Redes de Petri. RdP é um formalismo muito aderente às características que os problemas de planejamento apresentam, simplificando assim a modelagem.

Um problema de planejamento, modelado como uma RdP, é resolvido como um problema de alcançabilidade. As técnicas para se resolver este problema são computacionalmente intratáveis, para o tamanho das redes aqui geradas, o que motiva este trabalho mostrando uma nova abordagem para alcançabilidade em RdP acíclicas.

Esta nova metodologia envolve a geração de uma fórmula lógica a partir da RdP que, como foi observado, está na NNF. Para a aplicação de SAT clássico seria necessário um passo de conversão para CNF, cuja execução é ineficiente visto que o gasto de memória para a aplicação da operação de distribuição é exponencial, no pior caso. Além disso, a aplicação de um resolvedor SAT tornaria a busca pelo plano duplamente ineficiente, visto que seriam aplicados dois procedimentos potencialmente exponenciais.

Na pesquisa para evitar um dos passos citados (conversão para CNF e resolvedor SAT), encontraram-se formas normais (d-DNNF e FNNF) que possuem algoritmos extremamente eficientes para a enumeração de modelos. Assim, foram implementadas conversões para FNNF e um algoritmo parcial de enumeraçã de modelos, que no tableau KE faz parte do próprio processo de conversão.

O destaque neste trabalho é o uso do Tableau KE na tentativa de se resolver problemas de alcançabilidade em tempo razoável. Infelizmente, em virtude do tamanho das fórmulas NNF geradas, a conversão para FNNF se mostrou ineficiente mesmo para problemas pequenos. Mesmo assim, esta nova abordagem permite uma nova perspectiva ao problema

de planejamento e alcançabilidade, levando ao estudo de novas técnicas para sua solução.

O primeiro estudo futuro a ser realizado é na estrutura de representação da NNF e na sua própria geração. Uma estrutura de dados mais aprimorada, manutenção de subfórmulas repetidas e uso da estrutura da RdP para tirar proveito de tempo de processamento e ocupação de memória, podem trazer possíveis melhoramentos e fazer com que problemas, que nesta implementação são intratáveis, sejam resolvidos em tempo e espaço satisfatórios.

Outro trabalho futuro diz respeito à transformação de uma fórmula na NNF para FNNF. Aqui foram implementados três algoritmos e esperava-se que o algoritmo usando tableau KE tivesse um desempenho satisfatório, resolvendo problemas simples e sem o uso exponencial de memória. Mesmo assim, o tempo de processamento, em virtude das várias possibilidades[1], se mostrou inviável, mostrando que um algoritmo mais elaborado, projetado para se adequar à estrutura da aplicação deve ser implementado.

A proposta que talvez seja a mais promissora é a aplicação da *Dissolução de Caminhos*[2][26]. Nesta abordagem as fórmulas em NNF são representadas através de um grafo semântico, na qual são efetuadas operações para sua conversão em uma estrutura análoga a uma fórmula na FNNF, mas com algumas propriedades mais relaxadas.

Nesta conversão, literais complementares presentes em ramos diferentes de um nodo conjuntivo de uma fórmula na NNF são vistos como uma *ligação*. O processo de dissolução elimina estas ligações entre os nodos conjuntivos, sendo que sem estas ligações, a busca por um modelo apresentada anteriormente pode ser aplicada e retornará um modelo consistente. Já existe um trabalho em andamento que está propondo algoritmos e estruturas eficientes para o tratamento de ligações em fórmulas na NNF.

Analisando-se o processo de geração de FNNF (ou d-DNNF) e a busca por um modelo, percebe-se que a propriedade de determinismo não é necessária, visto que seu principal objetivo é que sejam gerados modelos diferentes em cada ramo de um nodo disjuntivo. Como a busca é por um modelo, não todos, o primeiro que for encontrado é suficiente.

Também percebe-se que a propriedade de decomponibilidade não precisa ser rigoro-

[1]No pior caso 2^n onde n é o número de variáveis da fórmula.
[2]Originalmente *Path Dissolution*.

samente cumprida. Caso se obtenha um nodo conjuntivo, a restrição mais forte a ser cumprida é que os ramos não podem apresentar literais complementares. Mas se possuirem o mesmo literal nenhuma inconsistência será gerada, visto que por idempotência esta duplicação é irrelevante.

A remoção das ligações, como proposto na dissolução de caminhos, faz uso do relaxamento das propriedades de determinismo e decomponibilidade da FNNF, tornando o processo de conversão mais eficiente.

BIBLIOGRAFIA

[1] Avrim Blum e Merrick Furst. Fast planning through planning graph analysis. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, páginas 1636–1642, 1995.

[2] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, páginas 151–158, New York, NY, USA, 1971. ACM Press.

[4] Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, e Joachim Posegga. *Handbook of Tableau Methods*. Addison Wesley, 1999.

[5] Marcello D'Agostino e Marco Mondadori. The taming of the cut: Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[6] A. Darwiche. The tractable counting of theory models and its application to belief revision and truth maintenance. Relatório Técnico D-113, Cognitive Systems Laboratory, UCLA, Ca 90095, 2000. To appear in Journal of Applied Non-Classical Logics.

[7] A. Darwiche e P. Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[8] Adnan Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research*, 8:165–222, 1998.

[9] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

[10] Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. *ECAI*, páginas 328–332, 2004.

[11] Martin Davis e Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[12] Minh Binh Do e Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.

[13] Javier Esparza. Decidability and complexity of petri net problems - an introduction. *Petri Nets*, páginas 374–428, 1996.

[14] R.O. Fikes e N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Relatório Técnico 43r, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, May de 1971. SRI Project 8259.

[15] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, e D. Wilkins. PDDL—the planning domain definition language, 1998.

[16] J. Gu, P. Purdom, J. Franco, e B. Wah. Algorithms for the satisfiability (SAT) problem: a survey. *Satisfiability Problem: Theory and Applications*, páginas 19–152. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1997.

[17] Jörg Hoffmann e Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[18] Henry Kautz e Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. Howard Shrobe e Ted Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, páginas 1194–1201, Menlo Park, California, 1996. AAAI Press.

[19] Henry Kautz e Bart Selman. Unifying SAT-based and graph-based planning. Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[20] Henry A. Kautz e Bart Selman. Planning as satisfiability. *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, páginas 359–363, 1992.

[21] João P. Marques-Silva e Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.*, 48(5):506–521, 1999.

[22] João Eugenio Marynowski. Ambiente de planejamento Ipê. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba PR, Brasil, novembro de 2004.

[23] Fabio Massacci. Simplification: A general constraint propagation technique for propositional and modal tableaux. *Lecture Notes in Computer Science*, 1397:217–??, 1998.

[24] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, e Sharad Malik. Chaff: Engineering an Efficient SAT Solver. *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[25] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April de 1989.

[26] Neil V. Murray e Erik Rosenthal. Dissolution: making paths vanish. *J. ACM*, 40(3):504–535, 1993.

[27] Héctor Palacios, Blai Bonet, Adnan Darwiche, e Hector Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. *ICAPS*, páginas 141–150, 2005.

[28] Reiner Hähnle and Neil Murray and Erik Rosenthal. Normal forms for knowledge compilation. *Lecture Notes in Computer Science*, 3488:304–313, 2005.

[29] Roberto Sebastiani. Applying GSAT to non-clausal formulas (research note). *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[30] Bart Selman, Henry A. Kautz, e Bram Cohen. Local search strategies for satisfiability testing. *Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability*, Providence RI, 1993.

[31] Bart Selman, Hector J. Levesque, e D. Mitchell. A new method for solving hard satisfiability problems. Paul Rosenbloom e Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, páginas 440–446, Menlo Park, California, 1992. AAAI Press.

[32] Fabiano Silva. Algoritmos para planificação baseados em strips. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba PR Brasil, outubro de 2000.

[33] Fabiano Silva. *Rede de Planos: Uma Proposta para a Solução de Problemas de Planejamento em Inteligência Artificial*. Tese de Doutorado, CEFET, Curitiba PR, Brasil, 2005.

[34] Zbigniew Stachniak. Going non-clausal. *SAT2002 Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, páginas 316–322, 2002.

[35] T. Vossen, M. Ball, A. Lotem, e D. Nau. Applying integer programming to AI planning. *Knowledge Engineering Review*, 16:85–100, 2001.

[36] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[37] Hantao Zhang. SATO: an efficient propositional prover. *Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI*, páginas 272–275, 1997.