Planejamento baseado em Verificação Simbólica de Modelos

Viviane Bonadia dos Santos

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação
Orientadora: Prof^a. Dr^a. Leliane Nunes de Barros

Durante parte do desenvolvimento deste trabalho o autor recebeu auxílio financeiro do CNPq

São Paulo, Maio de 2018

Planejamento baseado em Verificação Simbólica de Modelos

Esta é a versão original da dissertação elaborada pela candidata Viviane Bonadia dos Santos, tal como submetida à Comissão Julgadora.

Resumo

SANTOS, V. B **Planejamento baseado em Verificação Simbólica de Modelos**.2018. Dissertação (Mestrado) - Instituto de Matemática e estatística. Universidade de São Paulo, São Paulo, 2018.

Planejamento automatizado em Inteligência Artificial é a área que estuda o processo de escolha e organização de ações (síntese de plano) com o objetivo de alcançar metas preestabelecidas. Planejamento clássico é uma abordagem de planejamento que faz algumas suposições restritivas sobre o ambiente em que o agente atua. Esta abordagem lida com problemas onde o ambiente é completamente observável, finito, estático e não existe incerteza sob os efeitos das ações executadas pelo agente. Embora esta seja uma das abordagens mais estudadas em planejamento, muitos problemas de interesse prático não podem ser resolvidos, dadas suas suposições demasiadamente restritivas. O Planejamento não determinístico relaxa algumas dessas suposições e considera que pode existir incerteza nos efeitos das ações executadas pelo agente. Isso torna o planejamento não determinístico uma abordagem de maior aplicabilidade prática. Uma das principais abordagens para resolver problemas de planejamento não determinístico é a abordagem baseada em verificação de modelos. A maioria dos planejadores que utilizam esta abordagem baseia-se na lógica temporal de tempo ramificado CTL. Contudo, esta lógica possui algumas limitações. Para contornar as limitações da lógica CTL, a qual não considera explicitamente as ações do agente, foi proposta uma nova lógica, a lógica α-CTL, e um planejador capaz de resolver problemas de planejamento não determinísticos pertencentes à classe de problemas FOND (*Fully-Observable Non-Deterministic*), chamado PACTL. Neste trabalho de mestrado, temos por objetivo estender o planejador PACTL com representações e raciocínio simbólico. Chamamos nosso planejador de PACTL-SYM. No PACTL-SYM, os conjuntos de estados e ações são especificados como fórmulas lógicas e computacionalmente representados por meio de diagramas de decisão binária (*Binary Decision Diagram* - BDD).

Palavras-chave: planejamento não determinístico, verificação de modelos, FOND, BDD.

Abstract

SANTOS, V. B **Planning as Model Checking** 2018.Dissertation (MSc.) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2018.

In Artificial Intelligence automated planning is the area that studies the process of choosing and organizing the actions (plan synthesis) with the objective of achieving pre-established goals. Classical planning is an approach to planning that makes restrictive suppositions about the environment that the agent works. This approach deals with problems where the environment is completely observable, finite, static and there is no uncertain under the effects of the executed actions by the agent. Even though this is one of the most studied approaches in planning, many problems of practical interest cannot be solved, given its too restrictive suppositions. The non-deterministic planning relieves some of this suppositions and considers that uncertain can exist in the effects of the actions executed by the agent. It makes non-deterministic planning an approach of higher practical usage. One of the main approaches to solve non-deterministic planning problems is the model checking approach. The majority of the planners that use this approach base itself in the computation tree logic (CTL). This logic has some limitations. To work around the limitations of the CTL, that does not explicitly considers the actions of the agent, a new logic was proposed, the α-CTL logic, and a planner capable of solving Fully-Observable Non-Deterministic problems (FOND), called PACTL. This Master Thesis objective is to extend the PACTL planner with representations and symbolic reasoning. Our planner is called PACTL-SYM. In the PACTL-SYM, the set of states and actions are specified as logical formulas and computationally represented by Binary Decision Diagrams (BDD).

Keywords: non-deterministic planning, models checking, FOND, BDD.

Conteúdo

Lista de Abreviaturas xi

Lista de Figuras xiii

Lista de Tabelas xvii

1 Introdução 1

1.1 Planejamento clássico 1

1.1.1 Busca heurística no espaço de estados 2

1.1.2 Busca simbólica no espaço de estados abstratos 3

1.2 Planejamento não determinístico 3

1.2.1 Planejamento probabilístico 5

1.2.2 Planejamento FOND 5

1.3 Motivação 6

1.4 Objetivo 7

1.5 Contribuições 7

1.6 Organização 8

I Fundamentos 9

2 Representação e raciocínio baseado em teoria de conjuntos 11

2.1 Domínios de planejamento clássico 11

2.1.1 Linguagem STRIPS 11

2.1.2 Progressão de ações STRIPS 13

2.1.3 Regressão de ações STRIPS 13

2.1.4 Algoritmos de planejamento baseados em STRIPS 14

2.2 Domínios de planejamento FOND 15

2.2.1 Representação de estados e ações 15

2.2.2 Progressão de ações não determinísticas 16

2.2.3 Regressão de ações não determinísticas 17

3 Verificação de modelos 19

3.1 Estruturas de Kripke 19

3.2 Lógica de tempo ramificado 20

3.3 Caracterização de ponto fixo CTL 21

3.4 Algoritmos de verificação de modelos 22

4 Planejamento baseado em verificação de modelos 25
4.1 Síntese de políticas 26
4.2 Limitações do formalismo CTL para planejamento 28
4.3 A lógica temporal α-CTL 29
4.3.1 Sintaxe 29
4.3.2 Semântica 29
4.3.3 Metas de planejamento em α-CTL 31
4.4 O planejador PACTL 31
4.4.1 Propriedades formais do planejador PACTL 34
4.4.2 Exemplo de funcionamento do planejador PACTL 35

5 Representação e raciocínio baseado em fórmulas lógicas 37
5.1 Fórmulas Booleanas Quantificadas 37
5.1.1 Sintaxe 37
5.1.2 Semântica 38
5.2 Domínios de planejamento clássico 38
5.2.1 Representação de estados e ações em QBF 38
5.2.2 Progressão de ações determinísticas em QBF 39
5.2.3 Regressão de ações determinísticas em QBF 40
5.2.4 Busca Simbólica 40
5.3 Domínios de planejamento FOND 41
5.3.1 Raciocínio Implícito e Simbólico 42
5.3.2 Raciocínio Explícito e Simbólico 43

6 Diagramas de Decisão Binária 45
6.1 Formalização dos BDDs 45
6.1.1 BDD ordenado e reduzido 46
6.2 Algoritmos para BDDs 48
6.2.1 O algoritmo aplicar 48
6.2.2 O algoritmo restringir 49
6.2.3 O algoritmo quantificação 49

II O planejador PACTL-SYM 51

7 Síntese de soluções fracas e fortes 53
7.1 Síntese da política fraca 54
7.2 Síntese da política forte 56
7.3 Otimizações 57
7.3.1 Eliminação de estados espúrios 57
7.3.2 Heurística 59
7.4 Exemplo de execução do planejador PACTL-SYM 60

8 Síntese de soluções forte-cíclicas **63**
8.1 Síntese da política forte-cíclica do PACTL 63
8.2 Um novo algoritmo para síntese da política forte-cíclica 64

9 Analise Empírica **67**
9.1 Domínios 67
9.1.1 Domínio do robô 67
9.1.2 Domínio dos Primeiros Socorros 68
9.2 Soluções fracas 68
9.3 Soluções fortes 73

10 Conclusões e trabalhos futuros **75**
10.1 Contribuições deste trabalho 75
10.2 Trabalhos futuros 75

Bibliografia **77**

Lista de Abreviaturas

BDD	Diagrama de Decisão Binário (*Binary Decision Diagram*)
CTL	Lógica de Tempo Ramificado (*Computation Tree Logic*)
FOND	*Fully Observable Non-Deterministic Planning*
IA	Inteligência Artificial
MBP	Model Based Planner
PACTL	Planejador Baseado em α-CTL
PACTL-SYM	Planejador Simbólico Baseado em α-CTL
PRP	Planner for Relevant Policies
STRIPS	*Stanford Research Institute Planning System*
VM	Verificação de modelos

Lista de Figuras

1.1 Domínios de planejamento representados por grafos. Arestas pontilhadas e rotuladas pela mesma ação representam efeitos não determinísticos. 4

1.2 Exemplos de soluções fracas, forte e forte-cíclica para o domínio de planejamento representado na Figura 1.2a considerando s_0 como o estado inicial e s_g o estado meta. Nas figuras 1.2b, 1.2c e 1.2d os vértices e arestas em destaque representam os estados e ações que fazem parte da política enquanto os vértices e arestas em cinza claro representam os estados e ações do domínio que não fazem parte da política. Nas figuras, arestas pontilhadas e rotuladas pela mesma ação representam os efeitos não determinísticos da ação. 5

2.1 Domínio do aspirador de pó. 12

2.2 Descrição de ações STRIPS para o domínio do aspirador de pó. 12

2.3 Descrição de ações STRIPS proposicionalizadas para o domínio do aspirador de pó. 12

2.4 Domínio do aspirador de pó não determinístico. 16

2.5 Descrição da ação não determinística **aspirar(sala x)**. 16

3.1 Semântica dos operadores na árvore computacional CTL. Nós em cinza claro representam estados do modelo que satisfazem φ e nós em cinza escuro estados que satisfazem ϕ. 21

3.2 Exemplo da computação de ponto fixo máximo para $\exists \Box p$. 22

3.3 Operações de pré-imagem de verificação de modelos em CTL. 22

3.4 Exemplo de funcionamento do algoritmo para o operador $\forall \Diamond \varphi$ dado que $s_4 \vDash \varphi$. 23

4.1 Exemplo esquemático de um verificador de modelos e de um planejador baseado em verificação de modelos. 25

4.2 Operações de pré-imagem considerando ações. 26

4.3 Funcionamento do algoritmo de síntese da política forte. 27

4.4 Funcionamento do algoritmo de síntese da política fraca. 27

4.5 Semântica dos operadores temporais da lógica α-CTL. Fonte: Menezes (2014) 30

4.6 Domínio de planejamento para meta de alcançabilidade simples. Fonte: (Pereira, 2007). 35

5.1 Fórmula Booleana Quantificada. 38

5.2 Representação de ações determinísticas em QBF 39
5.3 Exemplo de uma ação determinística representada por fórmulas proposicionais. 39
5.4 Representação de ações não determinísticas em QBF. 42
5.5 Operações simbólicas de regressão fraca e forte. 42
5.6 Exemplo de representação simbólica para uma transição. 43
5.7 Exemplo de representação simbólica para um conjunto de transições. . . . 43
5.8 Operações de pré-imagem simbólica considerando ações. 44

6.1 Exemplo de árvore de decisão e BDD, respectivamente, para fórmula $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$. 46
6.2 Exemplo de reduções em BDDs ordenados 47
6.3 Exemplos de BDDs que representam a mesma função porém possuem ordenação de variáveis diferentes. Figura extraída de Bryant (1986). 47
6.4 Exemplo de execução do algoritmo aplicar considerando a seguinte ordenação das variáveis: $x_1 < x_2 < x_3$ 49

7.1 Exemplo esquemático da construção do submodelo M_π para a fórmula $\exists \diamond \varphi$ sendo $s_g \vDash \varphi$ e s_0 o estado inicial. As figuras da esquerda representam a construção do submodelo com pares estado-ação e as figuras da direita o construção do submodelo utilizando a estrutura de camadas. 58
7.2 Exemplo esquemático da extração da política fraca para o submodelo construído na Figura 7.1. 59
7.3 Exemplo da representação de conjuntos de estados quando o algoritmo de síntese utiliza heurística. Os retângulos representam conjuntos de estados com o mesmo valor heurístico. Retângulos em cinza representam conjuntos de estados gerados e explorados enquanto os retângulos em branco representam conjuntos que foram gerados mas não foram explorados. 60
7.4 Domínio do robô de carga 60
7.5 Ações simbólicas do domínio do robô de carga. 61
7.6 Regressão fraca a partir de $\varphi = (rAtA \wedge boxAtA \wedge boxOk)$ segundo a ação $moveBA$. 61
7.7 Submodelo construído pelo Algoritmo 7.1 para o Robô de Carga. 61
7.8 Operação de progressão a partir do estado s_0 considerando todos os efeitos não determinísticos da ação `takeB`. 61

8.1 Domínio de planejamento e política forte cíclica. 63
8.2 Síntese do submodelo que satisfaz $\forall \boxdot \exists \diamond \varphi$. 64
8.3 Árvore de computação do submodelo que satisfaz $\forall \boxdot \exists \diamond \varphi$. 64
8.4 Exemplo esquemático do algoritmo de síntese forte-cíclica. Em 8.4a. é computado um submodelo que satisfaz $\exists \diamond \varphi$. Em 8.4b. são podados os pares estado-ação que levam para fora do submodelo computado. Em 8.4c. é computado novamente um submodelo que satisfaz $\exists \diamond \varphi$ porém considerando apenas o subconjunto resultante dos passos anteriores. Em 8.4d. são podados os pares estado-ação que levam para fora do submodelo computado. . . 65

9.1 Domínio do robô para um problema com 1 caixa. Arestas tracejadas representam ações com efeitos não determinísticos enquanto arestas cheias representa ações com efeitos determinísticos. 69

9.2 Domínio dos primeiros socorros para um problema com uma localização, um incêndio, uma vítima, um caminhão de bombeiro e uma ambulância. Arestas tracejadas representam ações com efeitos não determinísticos enquanto arestas cheias representa ações com efeitos determinísticos. Na figura, as ações de socorrer a vítima na ambulância, socorrer a vítima com equipe de bombeiros e socorrer várias vítimas simultaneamente foram representadas através de uma única ação (socorrer a vítima) uma vez que os estados gerados por estas ações é o mesmo. 69

9.3 Exemplo de estados sucessores após executar as ações de socorrer uma vítima com equipe de bombeiros, socorrer uma vítima com equipe na ambulância e socorrer várias vítimas simultaneamente. 70

9.4 Políticas encontradas pelos planejadores PRP e PACTL-SYM para o problema do robô com 4 caixas. Os retângulos representam estados da política e as setas as transições de um estado para outro após a execução da ação da melhor ação possível. 71

9.5 Tamanho dos BDDs gerados em cada uma das camadas computadas pelo planejador PACTL-SYM heurístico e não heurístico. 72

Lista de Tabelas

6.1 Exemplo de tabela verdade para fórmula $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$. 46

9.1 Tempo (em segundos) para computar uma política para os problemas do domínio do robô. 70

9.2 Quantidade de pares estado-ação das políticas encontradas pelos planejadores PACTL-SYM e PRP para o problema do robô. 71

9.3 Tempo (em segundos) para computar uma política para os problemas do domínio de primeiros socorros com duas localizações com incêndio, uma ambulância e um caminhão de bombeiros, variando o número de vítimas. . 72

9.4 Tempo (em segundos) para computar uma solução forte para os problemas do domínio do robô com modificações. 73

Capítulo 1

Introdução

Em seu cotidiano, o ser humano planeja para determinar e organizar tarefas que precisam ser realizadas para alcançar um determinado objetivo futuro. Visto que esta é uma habilidade fundamental para o desenvolvimento humano bem como para seu comportamento inteligente, criar algoritmos capazes de planejar de maneira autônoma é um dos principais objetivos de diversas pesquisas realizadas em Inteligência Artificial, em particular, na sub-área de *planejamento automatizado.*

Na inteligência artificial, planejamento automatizado consiste no processo de seleção de ações para a síntese de um plano que ao ser executado, a partir de um estado inicial s_0, garante atingir um conjunto de metas preestabelecidas $\mathcal{G}$ (Nau *et al.*, 2004).

O planejamento automatizado busca formas gerais de síntese de planos, ou seja, independentes de domínio de aplicação. Formalmente, o *domínio de planejamento* descreve a dinâmica do ambiente em que o agente de planejamento atua. Dado um domínio de planejamento, um *problema de planejamento* consiste na descrição de uma situação inicial e da especificação de metas a serem alcançadas.

Criar algoritmos eficientes e estruturas de dados para representar e sintetizar planos é um dos grandes desafios de planejamento automatizado. Desde a criação do primeiro planejador automatizado, o GPS (*General Problem Solver*) (Newell *et al.*, 1959), muitas técnicas de planejamento foram desenvolvidas, como Planejamento baseado em Grafo de Planejamento (Blum e Furst, 1995), Planejamento de Ordem Parcial (Sacerdoti, 1975), Planejamento com Busca Heurística (Bonet e Geffner, 1999; P. E. Hart e Raphael, 1968) e Planejamento Baseado em Verificação de Modelos (Cimatti *et al.*, 1997).

1.1 Planejamento clássico

O *planejamento clássico* é uma abordagem que faz diversas suposições restritivas sobre o ambiente em que o agente atua, dentre elas pressupõe-se que: 1) o ambiente é completamente observável; 2) finito; 3) estático; e 4) evolui de forma *determinística*, *i.e.*, não existe incerteza sob os efeitos das ações executadas pelo agente. Em planejamento clássico uma mudança no ambiente só acontece quando o agente atua sobre ele (Russell e Norvig, 2009). Dadas essas suposições, o domínio de planejamento clássico é descrito por um conjunto finito de *estados* do ambiente, um conjunto finito de *ações* que modificam o ambiente e uma *função de transição de estados* que mapeia as mudanças que ocorrem de um estado para outro após a execução de uma ação.

Definição 1.1.1 (Domínio de planejamento clássico) *Formalmente, um domínio de planejamento clássico é definido por uma tupla* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$*, onde:*

- $\mathcal{S}$ *é um conjunto finito de estados possíveis do ambiente;*
- $\mathcal{A}$ *é um conjunto finito de ações que o agente pode executar; e*

- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ *é uma função de transição de estados.*

Um domínio de planejamento pode ser explicitamente representado por meio de um grafo direcionado onde os vértices representam os possíveis estados do domínio e as arestas, rotuladas por ações, representam todas as possíveis transições entre estados. A Figura 1.1a, apresenta um exemplo de um domínio de planejamento clássico representado por um grafo. Neste exemplo s_0, s_1, s_2 e s_3 representam os possíveis estados do domínio e as arestas, rotuladas pelas ações a_1, a_2, a_3, a_4 e a_5, representam as transições de estados. Por exemplo, a ação a_1 quando executada no estado s_0 leva o agente ao estado s_1. Quando uma ação a pode ser executada em um estado s dizemos que a é *aplicável* em s.

Dado um domínio, um problema de planejamento é determinado por um estado que descreve a configuração inicial do ambiente e um conjunto de condições que devem ser satisfeitas (meta). Em geral, um domínio possui mais de uma estado que satisfaz a meta, neste contexto, a meta do problema de planejamento também pode ser expressa pelo conjunto de estados que satisfazem as condições impostas. A definição formal para um problema de planejamento é dada a seguir:

Definição 1.1.2 (Problema de planejamento clássico) *Um problema de planejamento clássico é definido por uma tupla $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, onde:*

- $\mathcal{D}$ *é o domínio de planejamento;*
- $s_0 \in \mathcal{S}$ *é o estado inicial do ambiente; e*
- $\mathcal{G} \subseteq \mathcal{S}$ *é o conjunto de estados meta.*

Seja $\mathcal{D}$ um domínio e $\mathcal{P}$ um problema de planejamento. Um *planejador* é um algoritmo que busca por uma sequência de ações que ao ser executa, a partir do estado inicial, alcança algum estado meta. Sendo assim, a solução para um problema de planejamento é um caminho, também chamado de *plano*, que leva o agente de s_0 até um estado $g \in \mathcal{G}$. Considerando o domínio de planejamento clássico apresentado na Figura 1.1a, dado um problema de planejamento em que s_0 é estado inicial e s_3 um estado meta, um possível plano solução para este problema é a sequência de ações $\langle a_1, a_3 \rangle$. Outro possível plano solução é a sequência de ações $\langle a_2, a_5, a_3 \rangle$.

Planejamento automatizado é uma área de grande interesse dentro da Inteligência Artificial (Russell e Norvig, 2009). Por ser um problema complexo (planejamento clássico é um problema NP-difícil), nas últimas décadas muitas pesquisas foram realizadas com o objetivo de melhorar a eficiência dos algoritmos de planejamento.

A Competição Internacional de Planejamento (IPC) [1], criada em 1998 com objetivo de avançar o *estado-da-arte* em planejamento, ao longo de suas edições contou com diversos planejadores para problemas clássicos que combinaram diferentes técnicas a fim de alcançar maior eficiência. Dentre as abordagens utilizadas, duas que se destacaram foram o planejamento como busca heurística e a busca realizada através do raciocínio simbólico.

1.1.1 Busca heurística no espaço de estados

A ideia geral do uso de heurísticas é resolver o problema de interesse com algumas simplificações (relaxamento de restrições) a fim de obter uma estimativa da distância dos estados do problema até um estado meta mais próximo. Um dos grandes desafios do planejamento heurístico é encontrar formas de extrair heurísticas do problema independentes do domínio que auxiliem a busca para alcançar a meta (Bonet e Geffner, 2001).

[1] http://www.icaps-conference.org/index.php/Main/Competitions

Desde a segunda edição da IPC, em 2000, a abordagem heurística se mostrou promissora apresentando resultados melhores que os demais planejadores na maioria dos domínios. Dentre os competidores o mais bem sucedido desta edição foi o planejador FF (Hoffmann, 2001), que utiliza técnicas de grafos de planejamento sobre o problema relaxado para computar uma heurística bastante informativa. Muitos planejadores eficientes são baseados nessa técnica.

Em diversas edições da competição IPC, a abordagem heurística apresentou bons resultados. O planejador IBaCoP2 (Cenamor *et al.*, 2014), um dos vencedores da competição de 2014, faz parte do grupo de planejadores que se destacaram e que fazem uso de funções heurísticas. Outros planejadores também considerados *estado-da-arte* para planejamento clássico são: *Fast Downward* (Helmert, 2006) vencedor do IPC 2004 e LAMA 2011 (Richter *et al.*, 2011), vencedor em 2011.

1.1.2 Busca simbólica no espaço de estados abstratos

A representação do domínio de planejamento através de um grafo, onde os estados são representados de maneira explícita não é computacionalmente eficiente. Usualmente, um domínio de planejamento é descrito através de uma linguagem formal para representação de estados e ações, como por exemplo a linguagem STRIPS (Fikes e Nilsson, 1971). Muitos planejadores partem deste formalismo e realizam uma busca no espaço de estados gerando estados individualmente para encontrar uma solução.

Ao invés de representar e operar sobre os estados do domínio de forma individual, outra técnica que tem sido explorada é a representação de estados e ações através de fórmulas lógicas. Nesta abordagem, a busca é feita de maneira simbólica raciocinando sobre conjuntos de estados ao invés de estados individuais.

Esses conjuntos de estados são representados por Diagramas de Decisão Binária (BDDs) (Bryant, 1992) que são estruturas de dados eficientes que podem representar os conjuntos de forma compacta e raciocinar eficientemente sobre eles.

Na competição de 2014, quatro dentre os cinco melhores colocados na categoria de planejadores clássicos ótimos (*Sequential Optimal track*) foram planejadores que utilizavam técnicas simbólicas e BDDs (Edelkamp *et al.*, 2015).

Muitos dos planejadores atuais combinam diferentes técnicas para alcançar bons resultados. Por exemplo, o planejador SymBA*-2 (Álvaro Torralba *et al.*, 2014), vencedor da categoria de planejadores ótimos na competição IPC de 2014, combina as técnicas de busca heurística e raciocínio simbólico usando BDDs. Um dos objetivos deste trabalho é explorar essa combinação de técnicas, porém para problemas de planejamento não determinísticos.

1.2 Planejamento não determinístico

Embora planejamento clássico seja uma das abordagens mais estudadas em planejamento automatizado, muitos problemas de interesse prático não podem ser resolvidos com as suposições restritivas feitas por esta abordagem.

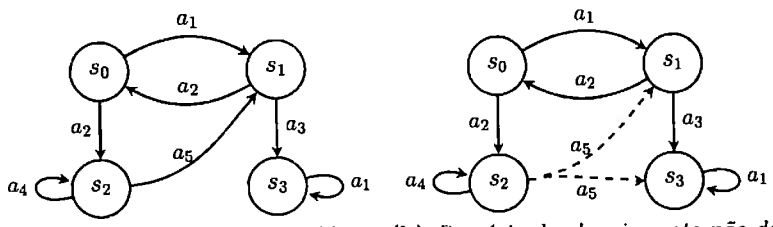
A suposição determinística, por exemplo, feita pelo planejamento clássico é uma visão simplificada do mundo que assume que ele evolui ao longo de um único caminho totalmente previsível (Nau *et al.*, 2004). Dentre os relaxamentos que podem ser feitos com as suposições do planejamento clássico, assumir incerteza nos efeitos das ações é uma das suposições que apresenta uma postura mais realista. Diferentemente da abordagem clássica, o planejamento em domínios não determinísticos lida com ações que possuem efeitos incertos.

O domínio dos primeiros socorros, utilizado na IPC de 2008, é um exemplo que apresenta incerteza sobre os efeitos das ações. Este domínio consiste em resolver emergências de incêndio e prestar socorro à vítimas. Para atender as emergências, existem equipes de

bombeiros que podem transportar água de um local para outro e não deterministicamente apagar incêndios. Além das equipes de bombeiros, existem equipes médicas que podem transportar vítimas, não deterministicamente estabilizá-las quando o atendimento é prestado na ambulância ou em locais que possuem incêndio ou prestar socorro às vítimas no hospital. Neste último caso, a vítima com certeza será estabilizada.

Note que, neste domínio, se o agente tenta, por exemplo, apagar um incêndio em um determinado local, esta ação pode levar à um estado em que não existe mais incêndio ou à um estado em que o incêndio persiste e a equipe não possui mais recursos (água). No primeiro caso, o objetivo de apagar um incêndio é alcançando. No segundo, o agente precisa saber qual a melhor a ação a ser executada. Neste caso, seria carregar o caminhão de bombeiro com mais água (se necessário ir até uma localização que possui água para fazer este carregamento) e tentar apagar o incêndio novamente.

A Figura 1.1b apresenta um exemplo artificial de um domínio de planejamento que possui incerteza nos efeitos das ações. Neste domínio, um agente planejador que se encontra, por exemplo, no estado s_2 ao executar a ação a_5 pode ir tanto para o estado s_1 quanto para o estado s_3, sem haver nenhuma preferência entre eles.



(a) *Domínio de planejamento clássico.* (b) *Domínio de planejamento não determinístico.*

Figura 1.1: *Domínios de planejamento representados por grafos. Arestas pontilhadas e rotuladas pela mesma ação representam efeitos não determinísticos.*

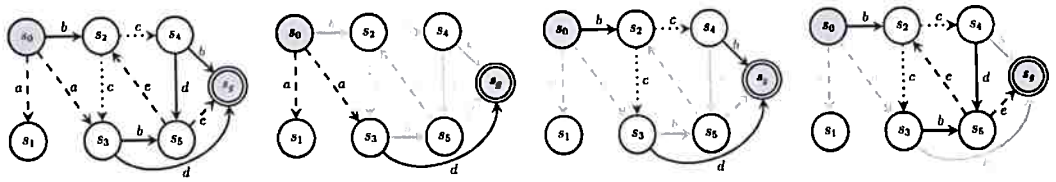
Em decorrência da existência de ações com efeitos incertos, um plano para um problema não determinístico pode resultar em múltiplos caminhos. Como consequência, diferentemente da abordagem clássica onde um plano era uma sequência de ações, um plano solução para um problema de planejamento não determinístico é uma *política*. Uma política é um mapeamento de estados em ações ($\pi : \mathcal{S} \rightarrow \mathcal{A}$), ou seja, a política determina qual é a ação mais apropriada para ser executada em cada estado. Uma política pode ser classificada em três diferentes tipos (Cimatti *et al.*, 2003):

- ***Política fraca***: é uma política que pode alcançar um estado meta, porém em razão do não determinismo, não oferece nenhuma garantia disso. A Figura 1.2b apresenta um exemplo de solução fraca para o domínio da figura 1.2a. A política fraca para este domínio é descrita pelos pares de estado-ação: $\{(s_0, a), (s_3, d)\}$.

- ***Política forte***: é uma política que independentemente do não determinismo garante alcançar um estado meta. Intuitivamente, as soluções fortes correspondem a planos seguros onde todos os caminhos possíveis do plano de execução alcançam a meta em um número finito de passos. A Figura 1.2c apresenta um exemplo de solução forte para o domínio da figura 1.2a. A política forte para este domínio é descrita pelos pares de estado-ação: $\{(s_0, b), (s_2, c), (s_3, d), (s_4, b)\}$.

- ***Política forte-cíclica***: é uma política que sempre alcança um estado meta pressupondo que sua execução eventualmente conseguirá sair de todos os ciclos existentes. Em uma solução forte-cíclica, todos os caminhos de execução alcançam um estado meta. No entanto sua execução pode resultar em uma sequência infinita de estados.

A Figura 1.2d apresenta um exemplo de solução forte-cíclica para o domínio da figura 1.2a. A política forte-cíclica para este domínio é descrita pelos pares de estado-ação: $\{(s_0, b), (s_2, c), (s_3, b), (s_4, d), (s_5, e)\}$.



(a) *Domínio de planejamento.* (b) *Solução fraca.* (c) *Solução forte.* (d) *Solução forte-cíclica.*

Figura 1.2: *Exemplos de soluções fracas, forte e forte-cíclica para o domínio de planejamento representado na Figura 1.2a considerando s_0 como o estado inicial e s_g o estado meta. Nas figuras 1.2b, 1.2c e 1.2d os vértices e arestas em destaque representam os estados e ações que fazem parte da política enquanto os vértices e arestas em cinza claro representam os estados e ações do domínio que não fazem parte da política. Nas figuras, arestas pontilhadas e rotuladas pela mesma ação representam os efeitos não determinísticos da ação.*

Existem dois tipos básicos de incerteza nos efeitos das ações: a incerteza probabilística, que abriga uma classe de problemas chamada de *planejamento probabilístico*, e a incerteza knightiana, que abriga uma classe de problemas chamada de *planejamento* FOND (*Fully-Observable Non-Deterministic*). Cada uma destas classes são descritas com mais detalhes nas seções que seguem.

1.2.1 Planejamento probabilístico

No planejamento probabilístico o agente supõe que existem probabilidades associadas aos efeitos não determinísticos de cada ação. Uma das principais abordagens para resolver problemas de planejamento probabilísticos, é a abordagem baseada em Processos de Decisão Markovianos (MDP) (Puterman, 1994). Além das probabilidades associadas aos possíveis efeitos de cada ação, os problemas de planejamento probabilístico possuem custos não uniformes associados às ações (e/ou valores recompensa associados aos estados alcançados). O problema de planejamento, neste caso, pode ser visto como um problema de otimização onde os planejadores buscam maximizar as recompensas (e/ou minimizar os custos).

1.2.2 Planejamento FOND

Planejamento FOND (*Fully-Observable Non-Deterministic*) é uma classe de planejamento que envolve domínios completamente observáveis (o agente não faz observações no ambiente para determinar em qual estado está) e as ações podem possuir efeitos não determinísticos com incerteza Knightiana.

Em domínios com incerteza Knightiana, uma ação pode ter mais de um efeito possível, porém não existe uma estimativa de qual será efetivamente o estado alcançado quando a ação for executada, *i.e*, não existem probabilidades associadas aos possíveis efeitos de cada ação.

Definição 1.2.1 (Domínio de planejamento FOND) *Um domínio de planejamento FOND é definido por uma tupla $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$, onde:*

- *$\mathcal{S}$ é um conjunto finito de estados possíveis do ambiente;*
- *$\mathcal{A}$ é um conjunto finito de ações que o agente pode executar; e*

- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ *é um função de transição de estados, sendo que dado um estado e uma ação, a função de transição pode levar para um conjunto de estados possíveis.*

Assim como em planejamento clássico, diversas técnicas foram desenvolvidas para resolver problemas FOND, sendo que planejadores atuais combinam essas técnicas para serem mais eficientes. Existem duas principais técnicas utilizadas pelos planejadores que resolvem problemas FOND:

- ***Planejamento* FOND *baseado na determinização das ações:*** os planejadores que utilizam esta técnica criam, para cada ação não determinística do domínio, uma nova ação e utilizam um planejador para problemas de planejamento clássico para encontrar uma solução. Dentre os planejadores que utilizam esta técnica estão o FIP (*Fast Incremental Planner*) (Fu *et al.*, 2011) e o planejador PRP (*Planner for Relevant Policies*) (Muise, 2014; Muise *et al.*, 2012), considerado estado-da-arte dentre os planejadores para problemas FOND.
- ***Planejamento baseado em Verificação de Modelos:*** a ideia principal desta abordagem é resolver problemas de planejamento através de um modelo formal, dando garantia sobre a validade do plano gerado. Dentre os planejadores que utilizam esta técnica estão os planejadores MBP (*Model Based Planner*) Bertoli *et al.* (2001) e o PACTL (Planejador Baseado em α-CTL) (Pereira, 2007).

1.3 Motivação

Para lidar com problemas do mundo real, em geral, exige-se planejadores mais complexos do que os planejadores que resolvem apenas problemas de planejamento clássico (Russell e Norvig, 2009). Os domínios de planejamento FOND relaxam algumas suposições da abordagem clássica tornando esta uma abordagem mais realística do mundo uma vez que é capaz de lidar com a incerteza nos efeitos das ações.

Uma das principais técnicas para lidar com problemas FOND é o planejamento baseado em verificação de modelos. De maneira geral, a ideia principal do planejamento baseado em verificação de modelos é resolver problemas de planejamento através de um modelo formal (Nau *et al.*, 2004). A principal vantagem desta abordagem é a habilidade de planejar de modo formal, com base em linguagens lógicas temporais, dando garantias fortes sobre a validade dos planos gerados. Em particular, através de uma linguagem lógica é possível representar metas de planejamento para os três tipos de políticas de planejamento não determinístico: fraca, forte e forte cíclica.

Ao longo dos anos, diferentes métodos de verificação de modelos foram propostos sendo que, muitos deles, representavam explicitamente o espaço de estados através de tabelas. Contudo, o número de estados do modelo pode crescer exponencialmente, fazendo com que o tamanho das tabelas seja um fator limitante em sistemas realísticos. Conforme citado anteriormente, a verificação simbólica de modelos (McMillan, 1992) é uma abordagem usada para controlar este problema de explosão de estados. Em seu trabalho, Burch *et al.* (1992) mostra resultados empíricos onde a verificação simbólica, com os estados representados através de BDDs, possibilita verificar propriedades de sistemas com 10^{20} estados ou mais, enquanto que a enumeração explícita encontra limitações com 10^{8} estados.

O planejamento baseado em verificação de modelos faz buscas considerando conjuntos de estados ao invés de estados individuais. Esses conjuntos de estados podem ser, e em geral são, extremamente grandes. A representação desses conjuntos usando BDDs resulta em uma representação compacta e que permite operações computacionalmente eficientes (raciocínio simbólico).

A abordagem de planejamento FOND baseada em verificação de modelos além de lidar com a incerteza nos efeitos das ações é capaz de tratar *metas de alcançabilidade estendida.*

Uma meta de alcançabilidade estendida especifica não só um objetivo a ser alcançado no final do planejamento mas também restrições temporais (que devem ser respeitadas ao longo da execução de um plano) representadas através de fórmulas da lógica temporal, em geral através da lógica temporal de tempo ramificado (CTL - *Computation Tree Logic*) (Clarke e Emerson, 1982), ou da lógica de tempo linear (LTL *Linear Time Logic*) (Pnueli, 1977).

Embora CTL seja um dos principais formalismos usado nos algoritmos baseados em verificação de modelos que resolvem problemas FOND, existem situações em que sua semântica não é adequada, por exemplo, como mostra Pereira (2007) em seu trabalho, no tratamento de metas de alcançabilidade estendida. Motivado por isso, ele propôs uma nova versão da lógica CTL, denominada α-CTL (Pereira e de Barros, 2008) e o planejador PACTL, que trata adequadamente o problema de planejamento com metas deste tipo. Além disso, diferentemente de abordagens baseadas em verificação CTL que constroem uma política e depois a validam, o formalismo α-CTL permite sintetizar políticas cuja validade é consequência direta do processo de síntese.

Outra propriedade garantida pelo planejador PACTL refere-se ao tamanho do caminho gerado ao executar uma política forte ou fraca. O PACTL garante que o menor caminho gerado pela execução de uma política fraca encontrada é mínimo no melhor caso e o maior caminho gerado pela execução de uma política forte encontrada é mínimo no pior caso. Diferentemente do planejador PRP que não oferece nenhuma garantia sobre o tamanho dos caminhos gerados a partir da execução de suas soluções.

Em seu trabalho, Pereira (2007) implementou um protótipo do planejador PACTL utilizando a linguagem PROPLOG. Até onde sabemos, não existem planejadores eficientes baseados em α-CTL.

1.4 Objetivo

O principal objetivo deste trabalho de mestrado é propor algoritmos para sintetizar soluções fracas, fortes e forte-cíclicas baseado em verificação simbólica de modelos e α-CTL e comparar o desempenho deles com outros planejadores que resolvem problemas de planejamento FOND considerados estado-da-arte.

1.5 Contribuições

As principais contribuições deste trabalho de mestrado são:

- Adaptação dos algoritmos do planejador PACTL (Planejador baseado em α-CTL) Pereira (2007) para a versão simbólica;
- Análise comparativa da eficiência dos algoritmos de planejamento propostos neste trabalho com outros planejadores que se destacam na resolução de problemas FOND;

Parte do trabalho apresentado nessa dissertação foi publicado nos seguintes artigos:

- Thiago Dias Simão, Ignasi Andres, Viviane Bonadia dos Santos e Leliane Nunes de Barros. Heurísticas para Detecção de Becos sem Saída em Planejamento Probabilístico. Em *XIII Encontro Nacional de Inteligência Artificial (ENIAC 2016)*, Recife, Brasil.
- Viviane Bonadia dos Santos e Leliane Nunes de Barros. PACTL-SYM: um planejador baseado em Verificação Simbólica de Modelos. Em *XIV Encontro Nacional de Inteligência Artificial (ENIAC 2017)*, Minas Gerais, Brasil.

1.6 Organização

Essa dissertação está organizada em duas partes. A primeira parte é apresenta os fundamentos deste trabalho e é composta pelos seguintes capítulos:

Capítulo 2 : Apresenta como planejar representando os estados do domínio individualmente e operando sobre eles com raciocínio baseado em teoria de conjuntos;

Capítulo 3 : Apresenta os conceitos de verificação de modelos;

Capítulo 4 : Apresenta os conceitos de planejamento baseado em verificação de modelos, a lógica α-CTL e algoritmos de planejamento baseados em α-CTL.

Capítulo 5 : Apresenta como planejar representando conjuntos de estados, ao invés de estados individuais, através de fórmulas lógicas e como racionar sobre estes conjuntos;

Capítulo 6 : Apresenta estruturas de dados eficientes para implementar algoritmos que representam conjuntos de estados através de fórmulas lógicas.

A segunda parte apresenta as contribuições deste trabalho e é composta pelos seguintes capítulos:

Capítulo 7 : Apresenta algoritmos baseados em verificação simbólica de modelos para sintetizar soluções fracas e fortes;

Capítulo 8 : Apresenta algoritmos baseados em verificação simbólica de modelos para sintetizar soluções forte cíclicas;

Capítulo 9 : Apresenta uma análise comparativa dos algoritmos de planejamento implementados neste trabalho com outros planejadores;

Capítulo 10 : Apresenta as conclusões e trabalhos futuros deste trabalho.

Parte I

Fundamentos

Capítulo 2

Representação e raciocínio baseado em teoria de conjuntos

Neste capítulo é apresentado como raciocinar sobre ações determinísticas e não determinísticas utilizando uma representação baseada em teoria de conjuntos. Nesta abordagem o raciocínio é feito sobre estados individuais.

2.1 Domínios de planejamento clássico

2.1.1 Linguagem STRIPS

A representação STRIPS (*Stanford Research Institute Problem Solver*) (Fikes e Nilsson, 1971) é uma linguagem formal para representação de estados e ações. Esta linguagem foi proposta no início da década de 70 baseada no cálculo de situações da lógica proposicional. Sua forma de representação teve grande influência na área de planejamento e, em consequência disso, STRIPS é a base de outras linguagens modernas para representar instâncias de problemas de planejamento automatizado (Russell e Norvig, 2009), como por exemplo a linguagem PDDL (*Planning Domain Description Language*), usada na competição internacional de planejamento.

Em STRIPS, os estados são descritos como um conjunto de átomos que denotam as propriedades que valem no mundo e as ações em termos de suas precondições e efeitos. Os efeitos das ações, por sua vez, são divididos em dois subconjuntos: (*i*) efeitos positivos; e (*ii*) efeitos negativos.

Definição 2.1.1 (Representação de ações determinísticas em STRIPS) *Formalmente, uma ação determinística a sobre um conjunto de proposições atômicas $\mathbb{P}$ é especificada por:*

$$a = (Precond(a), Ef^{+}(a), Ef^{-}(a)),$$

em que $Precond(a)$ é um conjunto de precondições que representam as proposições que devem ser verdadeiras no estado corrente para que a ação seja executada, $Ef^{+}(a)$ é o conjunto de proposições que se tornam verdadeiras após a execução da ação, e $Ef^{-}(a)$ é o conjunto de proposições que se tornam falsas após a execução da ação.

Para ilustrar como as ações são representadas em STRIPS, vamos considerar o domínio do aspirador de pó (Russell e Norvig, 2009) (Figura 2.1). Este domínio é formado por duas salas (sala A e sala B) e um agente aspirador de pó capaz de executar duas ações: mover-se de uma sala para outra e aspirar a sala em que se encontra.

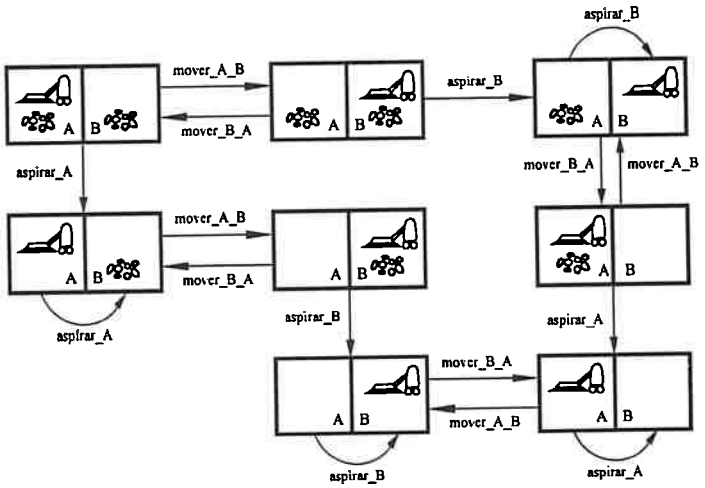


Figura 2.1: *Domínio do aspirador de pó.*

A Figura 2.2 apresenta as duas ações deste domínio formalizadas na linguagem STRIPS. Neste exemplo, *loc-em(x)* representa que o agente está localizado na sala x e *suja(x)* que a sala x possui sujeira. Na Figura 2.2 as ações estão descritas de forma generalizada. Um exemplo das ações deste domínio representadas de maneira proposicional é apresentado na Figura 2.3. Nas ações proposicionalizadas os parâmetros das ações (x e y) são substituídos por objetos do domínio que sejam do mesmo tipo do parâmetro. Por exemplo, no domínio do aspirador de pó A e B são objetos do tipo sala, portanto podem substituir os parâmetros x e y.

mover(sala x, sala y):
$\langle$Precond $= \{loc\text{-}em(x)\}$;
$Ef^+ = \{loc\text{-}em(y)\}$;
$Ef^- = \{loc\text{-}em(x)\}\rangle$,
aspirar(sala x):
$\langle$Precond $= \{suja(x)\}$;
$Ef^+ = \emptyset$;
$Ef^- = \{suja(x)\}\rangle$.

Figura 2.2: *Descrição de ações* STRIPS *para o domínio do aspirador de pó.*

mover_A_B:
$\langle$Precond $= \{loc\text{-}em\text{-}A\}$;
$Ef^+ = \{loc\text{-}em\text{-}B\}$; $Ef^- = \{loc\text{-}em\text{-}A\}\rangle$
mover_B_A:
$\langle$Precond $= \{loc\text{-}em\text{-}B\}$;
$Ef^+ = \{loc\text{-}em\text{-}A\}$; $Ef^- = \{loc\text{-}em\text{-}B\}\rangle$
aspirar_A:
$\langle$Precond $= \{suja\text{-}A\}$;
$Ef^+ = \emptyset$; $Ef^- = \{suja\text{-}A\}\rangle$
aspirar_B:
$\langle$Precond $= \{suja\text{-}B\}$;
$Ef^+ = \emptyset$; $Ef^- = \{suja\text{-}B\}\rangle$

Figura 2.3: *Descrição de ações* STRIPS *proposicionalizadas para o domínio do aspirador de pó.*

Para exemplificar a representação dos estados através de STRIPS, vamos considerar um estado do domínio do espirador de pó em que a sala A e a sala B estão sujas, e o agente aspirador está localizado na sala A. Em STRIPS este estado é representado por um conjunto formado pelas proposições: $\{loc\text{-}em\text{-}A,\ suja\text{-}A,\ suja\text{-}B\}$. Nesta forma de representação, adotamos a suposição de mundo fechado em que apenas as proposições verdadeiras no estado em questão são declaradas. As proposições não declaradas são consideradas falsas (por exemplo, a proposição *loc-em-B* é falsa neste estado).

2.1.2 Progressão de ações STRIPS

Dado um estado s a operação de progressão calcula o estado sucessor s' após a execução de uma ação a no estado s. Em planejamento clássico, como os efeitos das ações são determinísticos, a progressão de um estado produz um único estado sucessor.

Para computar a progressão de um estado s por uma ação a ($Progr(s,a)$), usando o formalismo STRIPS, primeiramente é necessário verificar se a ação a é aplicável no estado. Isto é feito verificando se as precondições da ação a estão contidas no estado atual, ou seja, $Precond(a) \subseteq s$. Se a ação não é aplicável, então não existe estado sucessor. Quando a ação é aplicável o estado sucessor é calculado eliminando do estado s os efeitos negativos da ação e adicionando os efeitos positivos. Formalmente, a progressão de um estado s por uma ação determinística a é dada por:

$$Progr(s,a) = \{s \setminus Ef^{-}(a)) \cup Ef^{+}(a)\},\ \text{se}\ Precond(a) \subseteq s. \tag{2.1}$$

Exemplo 2.1.1 (Progressão STRIPS) *Seja o estado* $s = \{loc\text{-}em\text{-}A,\ suja\text{-}A,\ suja\text{-}B\}$ *e a ação* ***mover_A_B*** *o estado sucessor* s' *é calculado da seguinte forma:*

$$\begin{aligned} s' &= (s \setminus Ef^{-}(a)) \cup Ef^{+}(a) \\ &= \{loc\text{-}em\text{-}A,\ suja\text{-}A,\ suja\text{-}B\} \setminus \{loc\text{-}em\text{-}A\} \cup \{loc\text{-}em\text{-}B\} \\ &= \{suja\text{-}A,\ suja\text{-}B\} \cup \{loc\text{-}em\text{-}B\} \\ &= \{suja\text{-}A,\ suja\text{-}B,\ loc\text{-}em\text{-}B\}. \end{aligned}$$

Vale destacar que as propriedades válidas no estado s e que não foram eliminadas pelos efeitos negativos da ação continuam válidas no estado sucessor s'. No Exemplo 2.1.2, no estado s, tanto a sala A quando a sala B estão sujas. Como a ação `mover` não alteram estas proposições, as salas permanecem sujas no estado s'.

A operação de progressão também pode ser calculada para um conjunto de estados X. Neste caso, a progressão resulta em um conjunto de estados formado pelos sucessores de cada um dos estados $s \in X$. Formalmente, o cálculo da progressão para um conjunto de estados é dado por:

$$Progr(X,a) = \bigcup_{s \in X} \{Progr(s,a)\}. \tag{2.2}$$

2.1.3 Regressão de ações STRIPS

Dado um estado s a operação de regressão por uma ação a, devolve um conjunto de estados predecessores de s, ou seja, um conjunto de estados X nos quais a execução da ação a resulta no estado s.

Para computar a regressão de um estado s por uma ação a ($Regr(s,a)$) é preciso verificar se a ação a é relevante para s. Uma ação a é relevante para um estado s, se seus efeitos positivos estão contidos em s, ou seja $Ef^{+}(a) \subseteq s$, e seus efeitos negativos não estão contidos em s, ou seja $Ef^{-}(a) \cap s = \varnothing$. Além disso, os estados antecessores devem conter as precondições da ação a. Formalmente, o cálculo de estado predecessor é dado por:

$$Regr(s,a) = \{(s \setminus Ef^{+}(a)) \cup Precond(a)\},\ \text{se}\ Ef^{+}(a) \subseteq s\ \text{e}\ Ef^{-}(a) \cap s = \varnothing \tag{2.3}$$

Exemplo 2.1.2 (Regressão STRIPS) *Dado o estado* $s = \{loc\text{-}em\text{-}B,\ suja\text{-}A,\ suja\text{-}B\}$ *e a ação mover_A_B) o estado predecessor é calculado da seguinte forma:*

$$\begin{aligned} X &= (s \setminus Ef^{+}(a)) \cup Precond(a) \\ &= \{loc\text{-}em\text{-}B,\ suja\text{-}A,\ suja\text{-}B\} \setminus \{loc\text{-}em\text{-}B\} \cup \{loc\text{-}em\text{-}A\} \\ &= \{suja\text{-}A,\ suja\text{-}B\} \cup \{loc\text{-}em\text{-}A\} \\ &= \{suja\text{-}A,\ suja\text{-}B,\ loc\text{-}em\text{-}A\}. \end{aligned}$$

Diferentemente da operação de progressão sobre um estado s em que o resultado é um estado s', o resultado da operação de regressão sobre um estado s é um conjunto de estados X, chamado de estado abstrato. X é um conjunto de estados pois representa todos os estados em que as proposições resultantes da operação de regressão são verdadeiras. Por exemplo, no Exemplo 2.1.3, X representa o conjunto de estados em que *suja-A*, *suja-B* e *loc-em-A*, são verdadeiras. As demais proposições do domínio não incluídas em X podem tanto ser verdadeiras quanto falsas. Qualquer estado do domínio representado por X em que a ação a for aplicada, levará ao estado s.

Assim como na progressão, é possível calcular a regressão para um conjunto de estados X obtendo como resultado um conjunto de estados predecessores dos estados $s \in X$. Formalmente, o cálculo de estados predecessores para um conjunto de estados é definido como:

$$Regr(X, a) = \bigcup_{s \in X} \{Regr(s, a)\}. \tag{2.4}$$

2.1.4 Algoritmos de planejamento baseados em STRIPS

Busca progressiva no espaço de estados

Muitos planejadores iniciam sua busca por uma solução para um problema de planejamento a partir do estado inicial, a fim de encontrar uma sequência de ações que alcançam um estado que satisfaz a meta.

A ideia básica dessa abordagem é aplicar sucessivas operações de progressão (apresentada na Equação 2.1) até alcançar um estado meta. O Algoritmo 2.1 apresenta um procedimento que descreve esta ideia.

Algoritmo 2.1: BUSCAPROGRESSIVA($\mathcal{P}$)

```
1  início
2      fronteira ← {s_0}
3      visitados ← ∅
4      π ← ∅
5      enquanto fronteira ≠ ∅ faça
6          seleciona s ∈ fronteira se s ∉ visitados
7          se G ⊆ s então
8              retorna EXTRAISOLUÇÃO(π, s)
9          senão
10             para cada a_i ∈ A e Precond(a_i) ⊆ s faça
11                 s' ← PROGR(s, a_i)
12                 fronteira. INSERE( s' )
13                 π.INSERE(⟨s, a_i, s'⟩)
14         visitados.INSERE(s)
15     retorna FALHA
```

No Algoritmo 2.1 *fronteira* representa o conjunto de estados candidatos a serem expandidos, *visitados* o conjunto de estados que já foram expandidos e π o conjunto de pares

estado ação candidatos ao plano solução. A cada iteração do algoritmo um estado s da fronteira é selecionado e, caso s não satisfaça a meta, seus estados sucessores são gerados e adicionados na $fronteira$. O algoritmo para quando não existem mais estados alcançáveis a partir do estado inicial para serem explorados ou quando um estado que satisfaça a meta é encontrado.

Este algoritmo computa os pares de estado-ação que podem levar à meta a partir do estado inicial. O plano solução, isto é, um conjunto de ações que ao serem aplicadas a partir do estado inicial é extraído de π pela função EXTRAISOLUÇÃO). Esta função é bem simples se armazenarmos, para cada par (s, a) de π, um ponteiro para o estado que gerou o par em questão. Neste caso, basta percorrer a estrutura a partir do estado que satisfaz a meta até alcançar o estado inicial concatenando ao plano solução as ações dos pares que foram visitados neste processo.

Nas primeiras décadas de pesquisa em planejamento (início da década de 60 até o final da década de 90) assumiu-se que a busca progressiva no espaço de estados era muito ineficiente (Russell e Norvig, 2009). Isso porque a busca progressiva é propensa a explorar ações irrelevantes uma vez que expande e examina sistematicamente todos os estados possíveis até que um estado meta seja alcançado ou não existam mais estados a serem explorados. Contudo, é possível melhorarmos o planejamento progressivo utilizando heurísticas.

Planejamento regressivo no espaço de estados

A busca por uma solução também pode ser feita a partir da meta. Neste caso, a busca inicia pela meta e termina quando alcança o estado inicial ou quando não existirem mais ações relevantes. O Algoritmo 15 apresenta um procedimento que descreve esta ideia.

Algoritmo 2.2: BUSCAREGRESSIVA($\mathcal{P}$)

```
1  início
2      fronteira ← {G}
3      visitados ← ∅
4      π ← ∅
5      enquanto fronteira ≠ ∅ faça
6          seleciona s ∈ fronteira se s ∉ visitados
7          se G ⊆ s_0 então
8              retorna EXTRAISOLUÇÃO(π)
9          senão
10             para cada a_i ∈ A e Ef⁺(a_i) ⊆ s e Ef⁻(a_i) ∩ s faça
11                 s' ← REGR(s, a_i)
12                 fronteira.INSERE(s')
13                 π.INSERE((⟨s', a_i, s⟩))
14         visitados.INSERE(s)
15     retorna FALHA
```

2.2 Domínios de planejamento FOND

2.2.1 Representação de estados e ações

Assim como nos problemas de planejamento clássico, os estados dos problemas de planejamento não determinístico podem ser representados como um conjunto de proposições. O raciocínio progressivo e regressivo também pode ser executado por meio de operação entre conjuntos, porém as ações não determinísticas devem ser formalizadas em uma linguagem que estende o formalismo STRIPS (Menezes *et al.*, 2014).

Assim como na representação STRIPS, neste formalismo estendido as ações são descritas em termos de suas precondições e efeitos. Contudo, no caso das ações não determinísticas, os efeitos são representados por um conjunto de efeitos não determinísticos: $Efeitos(a) = \{e_1, ..., e_n\}$ em que cada efeito $e_i \in Efeitos(a)$ possui seu próprio conjunto de efeitos positivos (Ef^+) e negativos (Ef^-).

Para ilustrar a representação de uma ação não determinística, considere o domínio do aspirador de pó com a ação *aspirar(sala x)* modificada (Figura 2.4). Ao executar esta ação, o agente aspirador nem sempre consegue aspirar toda a sujeira. Deste modo, a ação pode ter como efeito uma sala limpa ou uma sala suja (no caso do agente não conseguir aspirar toda a sujeira). A Figura 2.5 apresenta a formalização desta ação não determinística.

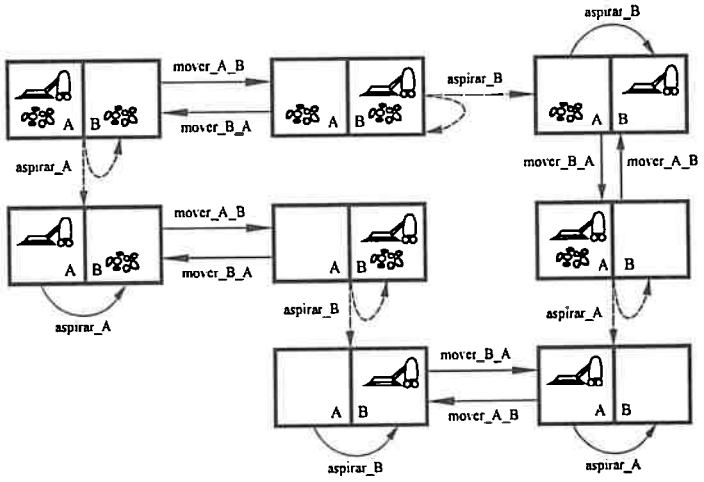


Figura 2.4: *Domínio do aspirador de pó não determinístico.*

aspirar(sala x):
(Precond = $\{suja(x)\}$;
$\{\langle Ef^+ = \emptyset;\ Ef^- = \{suja(x)\}\rangle,$
$\langle Ef^+ = \emptyset;\ Ef^- = \emptyset\rangle\}\rangle$

Figura 2.5: *Descrição da ação não determinística* ***aspirar(sala x).***

2.2.2 Progressão de ações não determinísticas

A progressão de um estado s por uma ação não determinística a ($ProgrND(s,a)$) resulta em um conjunto de possíveis estados sucessores (Menezes, 2014). Dado que a pode ter mais de um efeito o conjunto de estados sucessores é gerado pela união dos estados sucessores calculados para cada um dos efeitos possíveis de a. Formalmente, a progressão de um estado s por uma ação a é dada pela Equação 2.5.

$$ProgrND(s,a) = \bigcup_{e \in Efeitos(a)} \{(s \setminus Ef^-(a,e)) \cup Ef^+(a,e)\},\ \text{se}\ Precond(a) \subseteq s \qquad (2.5)$$

Note que esta equação também pode ser utilizada para calcular a progressão de estados com ações determinísticas. Este é um caso particular onde a ação possui um único efeito.

A Equação 2.6 estende a operação de progressão não determinística para um conjunto de estados $X = \{s_1, s_2, ..., s_n\}$:

$$ProgrND(X,a) = \bigcup_{s \in X} \{ProgrND(s,a)\}. \qquad (2.6)$$

2.2.3 Regressão de ações não determinísticas

A regressão não determinística, assim como a regressão determinística, produz um conjunto de estados predecessores. Dado um estado s e uma ação não determinística a, a ação a é relevante para s se existe algum efeito $e \in Efeitos(a)$ tal que $Ef^+(a,e) \subseteq s$ e $Ef^-(a,e) \cap s = \varnothing$. Formalmente, a regressão não determinística de um estado s por uma ação relevante a ($RegrND(s,a)$) é definida como (Menezes, 2014):

$$RegrND(s,a) = \{(s \setminus Ef^+(a,e)) \cup Precond(a)\}, \\ \text{se } \exists e \in Efeitos(a) \text{ tal que } Ef^+(a,e) \subseteq s \text{ e } Ef^-(a,e) \cap s = \varnothing. \tag{2.7}$$

Note que esta equação também pode ser utilizada para calcular a regressão de um estado por uma ação determinísticas. Este é um caso particular onde a ação possui um único efeito.

Seja S o conjunto de estados do domínio de planejamento, dados um subconjunto X de S e uma ação não determinística relevante para pelo menos um dos estados $s \in X$ é possível calcular a regressão fraca ou forte para o conjunto X:

- *Regressão fraca:* Dado um conjunto de estados X e uma ação não determinística, a regressão fraca computa um conjunto de estados que levam ***possivelmente*** a estados pertencentes a X. O cálculo da regressão fraca para um conjunto de estados a partir de uma ação relevante a é dado por:

$$RegrFraca(X,a) = \{y \in S : ProgrND(y,a) \cap X \neq \varnothing\}, \tag{2.8}$$

  ou seja, são os estados $y \in S$ nos quais o resultado da progressão de y por a tem interseção não nula com o conjunto de estados X.

- *Regressão forte:* Dado um conjunto de estados X e uma ação não determinística, a regressão forte computa um conjunto de estados que levam ***necessariamente*** a estados pertencentes ao conjunto X. O cálculo da regressão forte para um conjunto de estados a partir de uma ação a relevante é dado por:

$$RegrForte(X,a) = \{y \in S : \varnothing \neq ProgrND(y,a) \subseteq X\}, \tag{2.9}$$

  ou seja, são os estados $y \in S$ nos quais o resultado da progressão de y por a são estados que estão contidos no conjunto X.

Capítulo 3

Verificação de modelos

Verificação de modelo (Clarke *et al.*, 1999; Müller-olm *et al.*, 1999) é uma técnica de verificação automatizada de um sistema de transição finito (modelo) com o objetivo de determinar se este sistema finito cumpre uma determinada especificação (propriedade).

Para resolver um problema de verificação de modelos computacionalmente, o sistema de transição finito pode ser representado através de uma estrutura de Kripke e a propriedade a ser verificada por uma fórmula CTL. Neste contexto, um verificador de modelos é um algoritmo que dado um modelo e uma propriedade a ser verificada, o algoritmo visita sistematicamente todos os estados da estrutura verificando a validade da propriedade. Se todos os estados satisfazem a propriedade especificada, o algoritmo devolve sucesso. Caso contrário, devolve um contraexemplo (*i.e.* um exemplo de execução no qual é possível verificar a falha da propriedade).

3.1 Estruturas de Kripke

Um estrutura de Kripke é um grafo de transição de estados finito. Tal estrutura é formada por um conjunto finito de estados, um conjunto de transições entre estados, e uma função que rotula cada estado com um conjunto de proposições que são verdadeiras no estado em questão.

Definição 3.1.1 (Estrutura de Kripke) *Seja $\mathbb{P}$ um conjunto finito não vazio de proposições atômicas. Uma estrutura de Kripke sobre $\mathbb{P}$ é uma tupla $\mathcal{K} = (\mathcal{S}, \mathcal{T}, \mathcal{L})$, onde:*

- *$\mathcal{S}$ é um conjunto finito não vazio de estados;*
- *$\mathcal{T} : \mathcal{S} \to 2^{\mathcal{S}}$ é uma função de transição de estados; e*
- *$\mathcal{L} : \mathcal{S} \to 2^{\mathbb{P}}$ é uma função de interpretação de estados.*

Intuitivamente, as proposições em $\mathbb{P}$ representam propriedades dos estados do sistema e $\mathcal{L}(s)$ denota o conjunto de proposições atômicas que descrevem um estado particular $s \in \mathcal{S}$. Assumimos que o conjunto de proposições atômicas $\mathbb{P}$ contém as proposições $\bot$ e $\top$, e para todo estado $s \in \mathcal{S}$, $\top \in \mathcal{L}(s)$ e $\bot \notin \mathcal{L}(s)$.

Dado um estado inicial $s_0 \in \mathcal{K}$, a estrutura de Kripke pode ser transformada em uma árvore de computação infinita (que representa todos os caminhos possíveis de computação do sistema modelado). Para criar a árvore de computação, as relações de transição da estrutura de Kripke devem ser totais. Uma estrutura $\mathcal{K}$ é total se para todo $s \in \mathcal{S}$ existe um $s' \in \mathcal{S}$ tal que $(s, s') \in \mathcal{T}$. Caso a função de transição em $\mathcal{K}$ não seja total, adicionamos a cada estado terminal em $\mathcal{K}$ uma transição reflexiva (os estados terminais devem persistir infinitamente no tempo).

Um caminho em uma árvore de computação obtida a partir de uma estrutura de Kripke com estado inicial s, denotado por $\Upsilon^s_{\mathcal{K}}$, é uma sequência infinita de estados $\rho =$

$(\rho_1, \rho_2, ...) \in \mathcal{S}^\omega$, onde $\mathcal{S}^\omega$ representa o conjunto de todas as sequências infinitas sobre $\mathcal{S}$, tal que $\rho_i \in \mathcal{T}(\rho_{i-1})$, para $i > 0$. Em outras palavras, um caminho é uma sequência infinita de estados que representa uma possível execução do sistema modelado a partir do seu estado inicial. A semântica da lógica CTL, apresentada a seguir, é definida sobre a árvore de computação gerada a partir de uma estrutura Kripke.

3.2 Lógica de tempo ramificado

A Lógica de Árvore Computacional (*Computation Tree Logic* - CTL) (Clarke e Emerson, 1982) é uma lógica temporal de tempo ramificado que permite especificar propriedades quantificadas sobre caminhos em uma árvore de computação. Esta lógica é dita de tempo ramificado pois considera que para cada instante de tempo existem diferentes caminhos no futuro, sendo que qualquer um deles pode ser o caminho que de fato acontece (Huth e Ryan, 2004). Existem quatro operadores nesta lógica: $\circ$ (próximo estado), $\Diamond$ (finalmente, em um estado futuro), $\Box$ (globalmente, em todos os estados) e $\sqcup$ (uma fórmula é válida até que outra seja verdade). Todo operador temporal é precedido de um quantificador existencial $\exists$ (existe algum caminho) ou universal $\forall$ (inevitavelmente, em todos os caminhos).

Definição 3.2.1 (Sintaxe CTL) *O conjunto de fórmulas CTL é definido indutivamente como:*

$\varphi ::= p \in \mathbb{P} \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists \circ \varphi_1 \mid \forall \circ \varphi_1 \mid \exists \Box \varphi_1 \mid \forall \Box \varphi_1 \mid \exists \Diamond \varphi_1 \mid \forall \Diamond \varphi_1 \mid \exists(\varphi_1 \sqcup \varphi_2) \mid \forall(\varphi_1 \sqcup \varphi_2)$

Definição 3.2.2 (Semântica CTL) *Sejam K uma estrutura de Kripke, s um estado dessa estrutura e φ uma fórmula CTL. A semântica das fórmulas CTL é definida como:*

- $s \vDash p$ *sse* $p \in \mathcal{L}(s)$;
- $s \vDash \neg\varphi_1$ *sse* $s \nvDash \varphi_1$;
- $s \vDash \varphi_1 \wedge \varphi_2$ *sse* $s \vDash \varphi_1$ *e* $s \vDash \varphi_2$;
- $s \vDash \varphi_1 \vee \varphi_2$ *sse* $s \vDash \varphi_1$ *ou* $s \vDash \varphi_2$;
- $\exists \circ \varphi_1$ *sse para algum caminho* $\rho \in \Upsilon_K^s$, $\rho_1 \vDash \varphi_1$;
- $\forall \circ \varphi_1$ *sse para todo caminho* $\rho \in \Upsilon_K^s$, $\rho_1 \vDash \varphi_1$;
- $\exists \Box \varphi_1$ *sse para algum caminho* $\rho \in \Upsilon_K^s$ *e todo* $i \geq 1$, $\rho_i \vDash \varphi_1$;
- $\forall \Box \varphi_1$ *sse para todo caminho* $\rho \in \Upsilon_K^s$ *e todo* $i \geq 1$, $\rho_i \vDash \varphi_1$;
- $\exists(\varphi_1 \sqcup \varphi_2)$ *sse para algum caminho* $\rho \in \Upsilon_K^s$, *existe* $j \geq 0$ *tal que* $\rho_j \vDash \varphi_2$ *e, para todo* $i < j$, $\rho_i \vDash \varphi_1$; *e*
- $\forall(\varphi_1 \sqcup \varphi_2)$ *sse para todo caminho* $\rho \in \Upsilon_K^s$, *existe* $j \geq 0$ *tal que* $\rho_j \vDash \varphi_2$ *e, para todo* $i < j$, $\rho_i \vDash \varphi_1$.

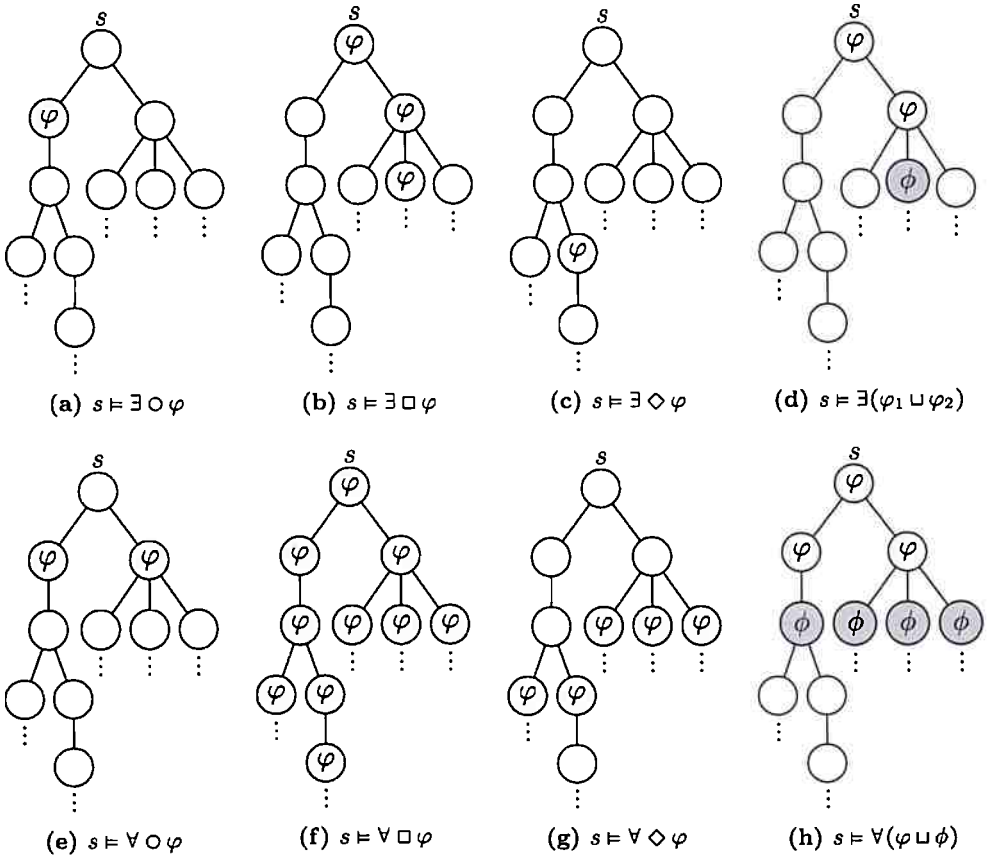


Figura 3.1: *Semântica dos operadores na árvore computacional* CTL. *Nós em cinza claro representam estados do modelo que satisfazem φ e nós em cinza escuro estados que satisfazem ϕ.*

3.3 Caracterização de ponto fixo CTL

Os algoritmos de verificação de modelos CTL são fundamentados na teoria de ponto fixo da CTL. Existe uma correspondência direta entre fórmulas CTL e conjuntos de estados que as satisfazem num determinado modelo temporal. Nesta subseção apresentamos o conceito de ponto fixo e como os operadores temporais CTL podem ser escritos em termos de ponto fixos.

Definição 3.3.1 (Funções monótonas e ponto fixo) *Sejam S um conjunto de estados e $\Gamma : 2^S \to 2^S$ uma função no conjunto de todos os subconjuntos de S:*

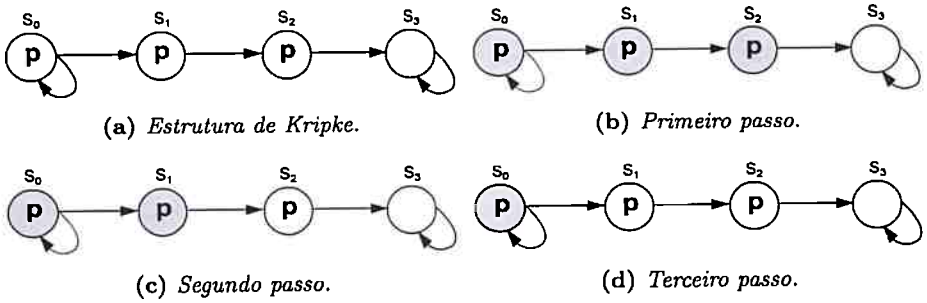
- *Dizemos que Γ é monótona sse $Y \subseteq X$ implica $\Gamma(Y) \subseteq \Gamma(X)$ para todos os subconjuntos Y e X de S*
- *Um subconjunto Y de S é chamado de ponto fixo de Γ sse $\Gamma(Y) = Y$*

Conforme Tarski (1955), se $\Gamma[Y]$ é uma função monótona, então $\Gamma[Y]$ tem um ponto fixo mínimo e um ponto fixo máximo. Seja S um conjunto finito de estados, o ponto fixo mínimo de uma função $\Gamma[Y]$, denotado por $\mu Y.\Gamma[Y]$, é encontrado quando Γ é aplicada sucessivamente a partir do menor subconjunto de S (o conjunto vazio) até encontrar o ponto fixo. O ponto fixo máximo de uma função $\Gamma[Y]$, denotado por $\nu Y.\Gamma[Y]$, é encontrado quando Γ é aplicada sucessivamente a partir do maior subconjunto de S (o próprio conjunto S) até encontrar um ponto fixo.

Teorema 3.3.1 (Clarke-Emerson) *Se S é finito, os operadores temporais globais de CTL podem ser caracterizados em termos de pontos fixos mínimos e máximos:*

- $\exists \Box \varphi = \nu Y.(\varphi \wedge \exists \bigcirc Y)$
- $\forall \Box \varphi = \nu Y.(\varphi \wedge \forall \bigcirc Y)$
- $\exists \Diamond \varphi = \mu Y.(\varphi \vee \exists \bigcirc Y)$
- $\forall \Diamond \varphi = \mu Y.(\varphi \vee \forall \bigcirc Y)$
- $\exists(\varphi_1 \sqcup \varphi_2) = \mu Y.(\varphi_2 \vee (\varphi_1 \wedge \exists \bigcirc Y))$
- $\forall(\varphi_1 \sqcup \varphi_2) = \mu Y.(\varphi_2 \vee (\varphi_1 \wedge \forall \bigcirc Y))$

Exemplo 3.3.1 (Ponto fixo máximo) *(Pereira, 2007) Podemos usar o funcional* $\Gamma[Y] = p \wedge \exists \bigcirc Y$ *para computar o conjunto de estados da estrutura de Kripke na Figura 3.2a que satisfazem a fórmula* $\exists \Box p$. *No primeiro passo temos* $\Gamma^1[\top] = p \wedge \exists \bigcirc \top = \{s_0, s_1, s_2\}$; *na segunda iteração, temos* $\Gamma^2[\top] = p \wedge \exists \bigcirc (p \wedge \exists \bigcirc \top) = \{s_0, s_1\}$; *e, finalmente, na terceira iteração, temos* $\Gamma^3[\top] = p \wedge \exists \bigcirc (p \wedge \exists \bigcirc (p \wedge \exists \bigcirc \top)) = \{s_0\}$ *(que já o ponto fixo máximo pois* $\Gamma^4[\top]$ *produz o mesmo resultado que* $\Gamma^3[\top]$*).*



(a) *Estrutura de Kripke.* (b) *Primeiro passo.*

(c) *Segundo passo.* (d) *Terceiro passo.*

Figura 3.2: *Exemplo da computação de ponto fixo máximo para* $\exists \Box p$.

3.4 Algoritmos de verificação de modelos

Os algoritmos de verificação de modelos CTL baseiam-se na implementação de operações de pré-imagem. Dado um conjunto de estados $X \subseteq \mathcal{S}$, a operação de pré-imagem computa um conjunto de estados Y que alcançam X. Existem dois tipos de pré-imagem: a *pré-imagem fraca* (que computa os estados Y que podem alcançar X, contudo, pode existir uma ramificação a partir de algum estado de Y que não alcança X) e a *pré-imagem forte* (que computa os estados que garantidamente alcançam X).

$$\text{PREIMAGEMFRACA(X)} = \{s \in \mathcal{S} \mid T(s) \cap X \neq \varnothing\}$$
$$\text{PREIMAGEMFORTE(X)} = \{s \in \mathcal{S} \mid \varnothing \neq T(s) \text{ e } T(s) \subseteq X\}$$

Figura 3.3: *Operações de pré-imagem de verificação de modelos em* CTL.

O Algoritmo 3.1 (Giunchiglia e Traverso, 2000) apresenta uma função de um verificador de modelos que verifica se uma estrutura de Kripke satisfaz uma fórmula CTL do tipo $\exists \Diamond \varphi$ a partir de um estado inicial s. Note que para verificar uma fórmula temporal do tipo $\exists \Diamond \varphi$ é usada a pré-imagem fraca.

Para verificar uma fórmula temporal do tipo $\forall \Diamond \varphi$ basta substituir o cálculo de pré-imagem fraca (Linha 7) pelo cálculo de pré-imagem forte. A Figuras 3.4 apresenta um

exemplo de funcionamento do algoritmo que trata do operador $\forall \Diamond \varphi$.

Algoritmo 3.1: VERIFICAEF$(\mathcal{K}, s, \varphi)$

```
1 estadosAtuais ← ∅
2 proximosEstados ← ESTADOS(φ, K)  // estados da estrutura de Kripke que
  satisfazem φ
3 enquanto proximosEstados ≠ estadosAtuais faça
4     se s ∈ proximosEstados então
5         retorna Verdadeiro
6     estadosAtuais ← proximosEstados
7     proximosEstados ← proximosEstados ∪ PREIMAGEMFRACA(proximosEstados)
8 retorna Falso
```

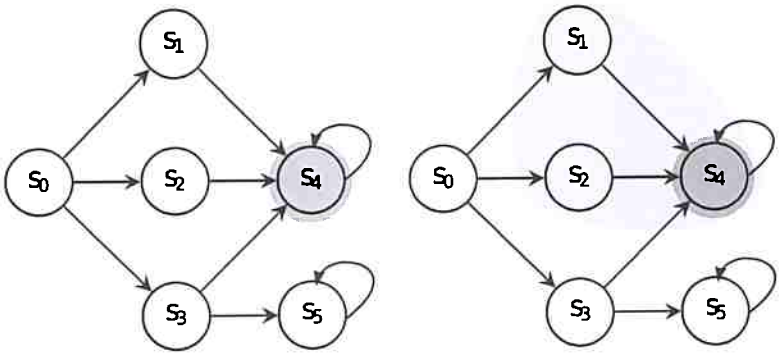


Figura 3.4: *Exemplo de funcionamento do algoritmo para o operador* $\forall \Diamond \varphi$ *dado que* $s_4 \vDash \varphi$.

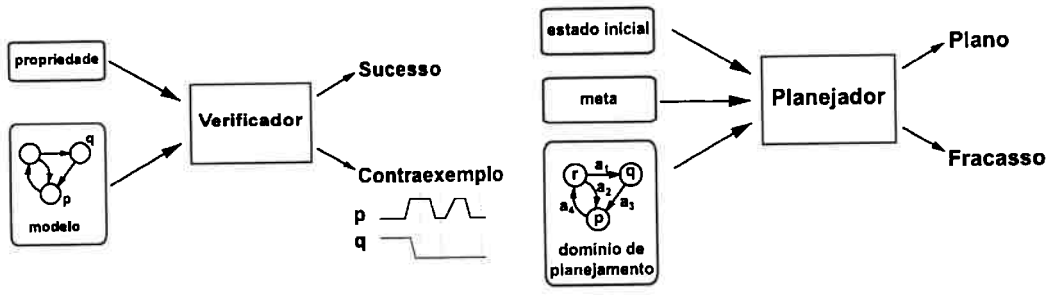
Capítulo 4

Planejamento baseado em verificação de modelos

Planejamento baseado em verificação de modelos é uma abordagem para resolver problemas de planejamento que possuem ações não determinísticas. Além de lidar com o não determinismo, esta técnica é utilizada para resolver problemas com observação parcial e metas estendidas (Nau *et al.*, 2004).

A ideia de planejamento baseado em verificação de modelos é resolver um problema de planejamento de maneira formal herdando técnicas de verificação de modelos para visitar os estados do modelo (Giunchiglia e Traverso, 2000)

No planejamento baseado em verificação de modelos, o modelo descreve o domínio de planejamento $\mathcal{D} = \langle S, \mathcal{A}, \mathcal{T} \rangle$. Note que, ao invés de uma estrutura de Kripke, em planejamento temos um modelo cujas transições são rotuladas pelas ações. Além disso, enquanto um verificador de modelos verifica a validade de determinada propriedade em um modelo $\mathcal{M}$ (Figura 4.1a), um planejador procura um submodelo de $\mathcal{M}$ (política) que satisfaça a meta a partir do estado inicial (Figura 4.1b). Assim, dado um problema de planejamento $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, $\mathcal{D}$ é um modelo de transição de estados rotulado pelas ações, s_0 é o estado inicial e $\mathcal{G}$ é a propriedade φ a ser verificada no modelo.



(a) *Verificador de modelos.* (b) *Planejador baseado em verificação de modelos.*

Figura 4.1: *Exemplo esquemático de um verificador de modelos e de um planejador baseado em verificação de modelos.*

Em geral, os planejadores baseados em verificação de modelos utilizam a lógica temporal CTL para formalizar as propriedades dos algoritmos.

Dado um problema de planejamento $\mathcal{P} = \langle \mathcal{D}, s_0, \mathcal{G} \rangle$, conforme citado anteriormente, uma política π pode ser de três tipos: (i) π é uma solução fraca para $\mathcal{P}$, se existe algum caminho a partir de s_0 no plano solução que alcança um estado final satisfazendo φ; (ii)

é uma solução forte, se todo caminho a partir de s_0 no plano solução é finito e alcança um estado final que satisfaz φ; e (iii) π é uma solução forte-cíclica, se todo caminho a partir de s_0 no plano solução alcança um estado final que satisfaz φ. Formalmente, uma política fraca é um submodelo $M \subseteq \mathcal{D}$ cuja execução satisfaz a fórmula CTL $\exists \Diamond \mathcal{G}$ ($\mathcal{G}$ é uma fórmula proposicional que representa o conjunto de estados que satisfazem a meta). A política forte, por sua vez é um submodelo $M \subseteq \mathcal{D}$ cuja execução satisfaz a fórmula $\forall \Diamond \mathcal{G}$ (Giunchiglia e Traverso, 2000) e, a política forte cíclica é um submodelo $M \subseteq \mathcal{D}$ cuja execução satisfaz a fórmula $\forall \exists \Diamond \mathcal{G}$ (Daniele *et al.*, 2000).

A seguir são apresentados algoritmos baseados em verificação de modelos para síntese de política forte, fraca e forte cíclica que utilizam o formalismo CTL.

4.1 Síntese de políticas

Assim como os algoritmos de verificação, os algoritmos de síntese são baseados em operações de pré-imagem, porém este cálculo é feito levando em consideração as ações que rotulas as transições do modelo (Figura 4.2). O cálculo da pré-imagem fraca de um conjunto de estados $X \subseteq \mathcal{S}$ (usado na síntese de política fraca) computa um conjunto de pares estado-ação que possivelmente alcançam X e o cálculo da pré-imagem forte (usado na síntese de política forte) computa um conjunto de pares estado-ação que garantidamente (a despeito do não determinismo) alcançam X.

$$\text{PRÉIMAGEMFRACA(X)} = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A} \text{ e } \mathcal{T}(s,a) \cap X \neq \emptyset\}$$
$$\text{PRÉIMAGEMFORTE}(X) = \{(s,a) : s \in \mathcal{S}, a \in \mathcal{A} \text{ e } \emptyset \neq \mathcal{T}(s,a) \subseteq X\}$$

Figura 4.2: *Operações de pré-imagem considerando ações.*

Para sintetizar uma solução forte, o algoritmo de PLANEJAMENTOFORTE (Cimatti *et al.*, 1998) recebe como entrada um problema de planejamento $\mathcal{P}$ e devolve um plano contendo uma solução forte, quando existe uma solução forte, ou fracasso, caso contrário. Para encontrar uma solução, o algoritmo parte do conjunto $S' = \mathcal{G}$ e realiza uma busca regressiva até alcançar o estado inicial ou até atingir um ponto fixo mínimo, a partir do qual S' não pode mais ser estendido (neste caso não existe uma solução para o problema). O passo básico desse algoritmo é a função de PRÉIMAGEMFORTE (Figura 4.2).

A PRÉIMAGEMFORTE(X), retorna um conjunto de pares estado-ação (s,a) tal que, a ação a é aplicável no estado s ($\emptyset \neq \mathcal{T}(s,a)$) e a execução dessa ação no estado s (possivelmente não determinística) leva, necessariamente, a um estado que pertence ao conjunto de estados X ($\mathcal{T}(s,a) \subseteq X$).

Algoritmo 4.1: PLANEJAMENTOFORTE($\mathcal{P}$)

```
1  i ← 0
2  π_0 ← ∅
3  X ← G
4  enquanto π_i = π_{i-1} faça
5      se s_0 ⊆ π_i então retorna DETERM(π_i);
6      PI ← PRÉIMAGEMFORTE(X)
7      π_{i+1} ← π_i ∪ PODA(PI, X)
8      X ← X ∪ ESTADOS(PI)
9      i ← i + 1
10 retorna FRACASSO
```

No algoritmo, PODA (Linha 7) elimina da Pré-Imagem (PI) os pares de estado-ação que já estão no conjunto S' e portanto já foram mapeados a alguma ação em uma das

iterações anteriores. Isso garante que a política sintetizada seja ótima e acíclica. Dado um conjunto de pares estado-ação, o método ESTADOS (Linha 8) retorna o conjunto de estados $\in PI$. Iterativamente, o algoritmo faz a união dos novos estados alcançados (PI) com o conjunto de estados já visitados (S') e calcula a pré-imagem até que o estado inicial seja alcançado. Finalmente, para garantir que a política associe no máximo uma ação de $\mathcal{A}$ a cada estado que compõem a política (ou seja, garantir que a política seja determinística), o algoritmo faz uma chamada ao método DETERM(π) (Linha 5) que devolve uma política $\pi' \subseteq \pi$, onde para cada estado da política existe uma única ação associada.

O algoritmo de síntese de soluções fracas é similar ao algoritmo de síntese de soluções fortes, a diferença entre eles está no tipo de pré-imagem utilizada para computar uma solução. Enquanto a síntese de política forte computa a pré-imagem forte, a síntese de política fraca computa a pré-imagem fraca.Para isso, basta substituir a Linha 6 do Algoritmo 4.1 por uma chamada à função de PRÉIMAGEMFRACA (Figura 4.2).

As figuras 4.3 e 4.4 apresentam, respectivamente, o funcionamento dos algoritmos de síntese da política forte e política fraca.

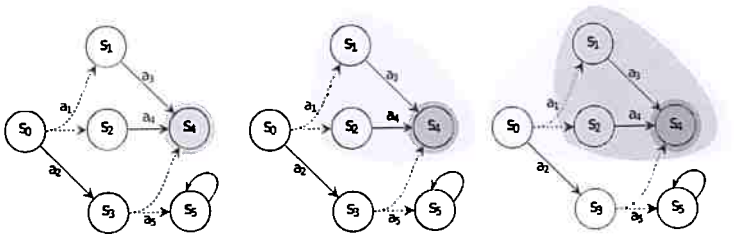


Figura 4.3: *Funcionamento do algoritmo de síntese da política forte.*

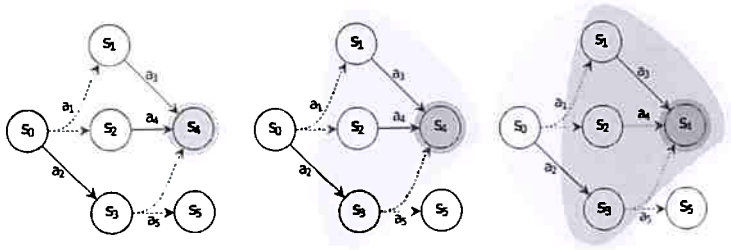


Figura 4.4: *Funcionamento do algoritmo de síntese da política fraca.*

O Algoritmo 4.2 (PLANEJAMENTOFORTECICLICO) (Daniele *et al.*, 2000) mostra como uma política forte-cíclica pode ser computada. Este algoritmo inicia sua execução com um conjunto de todos os pares estado-ação do domínio (exceto estados meta) e iterativamente contrai este conjunto podando, através das funções auxiliares PODAFORA e PODABECO, pares de estado-ação que não garantem alcançar a meta.

A função PODAFORA remove pares de estado-ação que podem levar para estados que não estão contidos no conjunto de estados, a função PODABECO, por sua vez, remove os pares de estado-ação a partir dos quais não é possível alcançar a meta,

Algoritmo 4.2: PLANEJAMENTOFORTECICLICO($\mathcal{P}$)

```
1 I ← I\G
2 SCP ← {⟨s,a⟩ : s ∈ S\G, a ∈ A, a é executável em s}
3 oldSCP ← ⊥
4 enquanto OldSCP6 ≠ SCP faça
5     OldSCP ← SCP
6     SCP ← PODABECO(P, PODAFORA(P, SCP))
7 se I ⊆ STATES(SCP) então retorna SCP senão retorna FALHA;
```

Algoritmo 4.3: PODAFORA($\mathcal{P}, SA$)

```
1 Outgoing ← COMPUTEOUTGOING(P, SA)
2 enquanto Outgoing ≠ ∅ faça
3     SA ← SA\Outgoing
4     Outgoing ← COMPUTEOUTGOING(P, SA)
5 retorna SA
```

Algoritmo 4.4: PODABECO($\mathcal{P}, SA$)

```
1 ConnectedToG ← ∅
2 OldConnectedToG ← ⊥
3 enquanto ConnectedToG ≠ OldConnectedToG faça
4     OldConnectedToG ← ConnectedToG
5     ConnectedToG ← SA ∩ PREIMAGEMFRACA(P, ConnectedToG)
6 retorna ConnectedToG
```

4.2 Limitações do formalismo CTL para planejamento

Conforme citado anteriormente, em geral, os planejadores baseados em verificação de modelos utilizam a lógica temporal CTL para formalizar as propriedades dos algoritmos.

Contudo, a semântica CTL não diferencia as transições causadas pelas diferentes ações. Neste contexto, os planejadores baseados em verificação de modelos que utilizam CTL necessitam de alguns mecanismos extra-lógicos para auxiliar na escolha de ações.

Em seu trabalho, Pereira (2007) mostra que a lógica CTL pode ser usada para validar políticas mas não para sintetizar políticas com base em sua semântica. Além disso, não é possível especificar uma meta de alcançabilidade estendida ramificada usando CTL.

Uma meta de alcançabilidade estendida, além de especificar uma condição a ser satisfeita em um estado, determina uma condição que deve ser preservada (ou evitada) ao longo de toda execução da política.

Definição 4.2.1 (Meta de alcançabilidade estendida) *Formalmente, uma meta de alcançabilidade estendida é um par de fórmulas (φ_1, φ_2), onde φ_1 especifica a condição a ser preservada durante a execução de uma política e φ_2 uma condição a ser alcançada ao final de execução da política (Pereira, 2007).*

Uma meta de alcançabilidade estendida pode ser de dois tipos: linear ou ramificada. Uma meta de alcançabilidade estendida linear φ_1 é descrita por uma fórmula proposicional. Neste caso, a validade de φ_1 depende apenas do caminho específico que leva ao estado meta. Por exemplo, vamos supor que precisamos atravessar uma estrada com buracos. Uma meta de alcançabilidade estendida linear teria como restrição chegar no final da estrada sem ter passado por nenhum buraco. Já uma meta de alcançabilidade estendida ramificada φ_1 é descrita por uma fórmula temporal. Neste caso, a validade de φ_1 não depende apenas do

caminho de execução da política mas também das possíveis ramificações desse caminho. No problema da estrada, um exemplo de meta de alcançabilidade estendida ramificada seria: chegar ao final da estrada passando no mínimo à dois estados de alcançar um estado que tenha buraco. Ou seja, não basta simplesmente não possuir buracos no caminho que leva até a meta é necessário que as ramificações desse caminho não alcancem um estado com buraco em dois passos.

Para contornar as limitações da lógica CTL para planejamento, Pereira (2007) propôs uma extensão da lógica CTL, chamada α-CTLque leva em consideração as ações em sua semântica (do Lago Pereira e de Barros, 2008).

4.3 A lógica temporal α-CTL

4.3.1 Sintaxe

Para diferenciar a sintaxe das fórmulas da lógica α-CTL das fórmulas da lógica CTL, os símbolos dos operadores temporais que compõem esta lógica serão representados com um ponto: $\odot$ (*sucessor imediato*), $\boxdot$ (*invariantemente*), $\diamondsuit$ (*finalmente*) e $\sqcup$ (*até que*).

Definição 4.3.1 *(Sintaxe α-CTL) Seja $p \in \mathbb{P}$ uma proposição atômica. A sintaxe de α-CTL é definida indutivamente como:*

$$\varphi ::= \top \mid p \mid \neg p \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid \exists \odot \varphi \mid \forall \odot \varphi \mid \exists \boxdot \varphi \mid \forall \boxdot \varphi \mid \exists (\varphi_1 \sqcup \varphi_2) \mid \forall (\varphi_1 \sqcup \varphi_2).$$

Em α-CTL, fórmulas bem formadas estão na forma normal negativa (ou seja, a negação está restrita às fórmulas atômicas). Além disso, todos os operadores temporais devem ser prefixados por um quantificador de caminho ($\exists$ ou $\forall$) e os operadores temporais compostos pelo símbolo $\diamondsuit$ são definidos como: $\exists \diamondsuit \varphi_2 \doteq \exists(\top \sqcup \varphi_2)$ e $\forall \diamondsuit \varphi_2 \doteq \forall(\top \sqcup \varphi_2)$.

4.3.2 Semântica

Seja $\mathbb{P}$ um conjunto não vazio de proposições atômicas e $\mathbb{A}$ um conjunto não vazio de ações contendo a ação trivial τ. Um modelo temporal para uma fórmula lógica α-CTL é um grafo de transição de estados $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ com assinatura ($\mathbb{P}$,$\mathbb{A}$), onde os estados são rotulados por subconjuntos não vazios de $\mathbb{P}$ e as transições rotuladas por elementos de $\mathbb{A}$. Nesse modelo, os estados terminais (onde só é possível executar a ação τ) persistem infinitamente no tempo. Informalmente, podemos dizer que um modelo temporal para fórmulas α-CTL é um domínio de planejamento $\mathcal{D}$ ou a estrutura de execução $\mathcal{D}_\pi$ de uma política π.

Intuitivamente, um estado s em um modelo temporal $\mathcal{D}$ satisfaz uma fórmula $\forall \odot \varphi$ se existe uma ação $a \in \mathbb{A}$ que, quando executada em s necessariamente alcança um estado sucessor imediato de s que satisfaz a fórmula φ. Já um estado s em um modelo temporal $\mathcal{D}$ satisfaz uma fórmula $\exists \odot \varphi$ se existe uma ação $a \in \mathbb{A}$ que, quando executada em s, possivelmente alcança um estado sucessor imediato de s que satisfaz a fórmula φ.

Sejam $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ um grafo de transição de estados com assinatura ($\mathbb{P}$,$\mathbb{A}$) e $s \in \mathcal{S}$ o estado inicial. A Figura 4.5, ilustra a semântica dos operadores de α-CTL. Note que as transições são rotuladas pelas ações a, b, c e d.

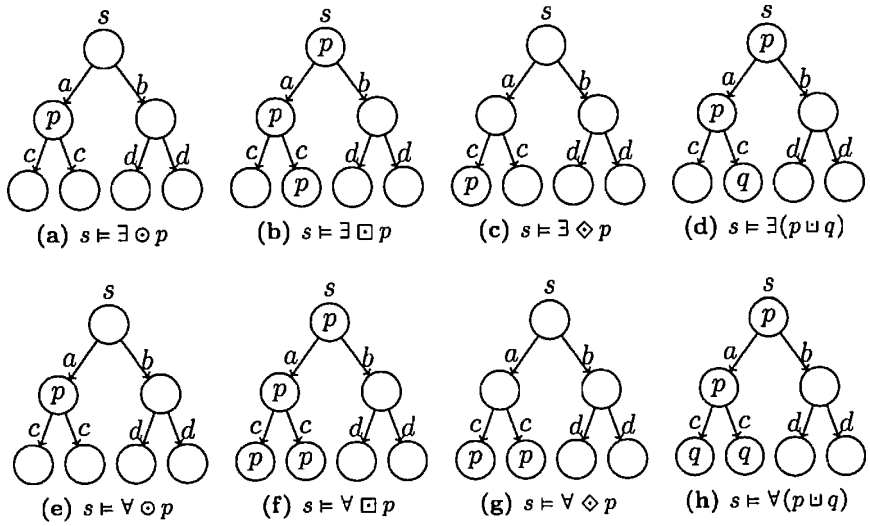


Figura 4.5: *Semântica dos operadores temporais da lógica* α-CTL.
Fonte: Menezes (2014)

- Na Figura 4.5a, $s \vDash \exists \odot p$ expressa que existe um próximo estado, para alguma ação, em que p é verdade (nesse caso, executando-se a ação a).
- Na Figura 4.5e, $s \vDash \forall \odot p$ expressa que em todos os próximos estados, para alguma ação, p é verdade (neste caso, executando-se a ação a). Note que, de acordo com a semântica CTL o resultado para este grafo de transições seria diferente. Segundo a semântica CTL, dado o modelo $\mathcal{D}$ e o estado inicial s (representados pela Figura 4.5e), $(\mathcal{D}, s) \nvDash \forall \odot p$.
- Na Figura 4.5b, $s \vDash \exists \boxdot p$ significa que existe um caminho na árvore no qual em todos os estados, para alguma ação, é válida a proposição p (neste caso, executando-se a ação a).
- Na Figura 4.5f, $s \vDash \forall \boxdot p$ expressa que para todos os caminhos da árvore, partindo de s, para alguma ação, é válida a proposição p (neste caso executando-se a ação a).
- Na Figura 4.5c, $s \vDash \exists \diamond p$ significa que em algum estado futuro, para alguma ação é válida a proposição p (neste caso executando-se a ação a).
- Na Figura 4.5g, $s \vDash \forall \diamond p$ expressa que para todos os estados futuros, para alguma ação, é válida a proposição p (neste caso executando-se a ação a).
- Na Figura 4.5d, $s \vDash \exists (p \sqcup q)$ expressa que algum estado futuro, para alguma ação, a proposição p é válida até que q seja verdade (neste caso executando-se a ação a).
- Na Figura 4.5h, $s \vDash \forall (p \sqcup q)$ expressa que para todos os estados do caminho a partir de s, executando-se alguma ação, a proposição p é verdade até que q seja verdade (neste caso executando-se a ação a).

Formalmente, a semântica dos operadores $\forall\odot$ e $\exists\odot$ é definida utilizando os conceitos de pré-imagem de um conjunto de estados (Definição 4.3.2). A semântica para os operadores $\exists\boxdot$, $\forall\boxdot$, $\exists\sqcup$ e $\forall\sqcup$ é derivada a partir dos operadores $\exists\odot$ e $\forall\odot$, usando operações de ponto fixo mínimo (μ) e máximo (ν) (Pereira, 2007).

Definição 4.3.2 (Pré-Imagem) *Seja $Y \subseteq S$ um conjunto de estados. A pré-imagem fraca de Y, denotada por $\mathcal{T}_{\exists}^{-}(Y)$, é o conjunto $\{s \in S : \exists a \in \mathbb{A} \,.\, \mathcal{T}(s,a) \cap Y \neq \varnothing\}$, e a pré-imagem forte de Y, denotada por $\mathcal{T}_{\forall}^{-}(Y)$, é o conjunto $\{s \in S : \exists a \in \mathbb{A} \,.\, \varnothing \neq \mathcal{T}(s,a) \subseteq Y\}$.*

Definição 4.3.3 (Semântica α-CTL) *Sejam $\mathcal{D} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$ um modelo temporal com assinatura $(\mathbb{P}, \mathbb{A})$. A intenção de uma fórmula α-CTL φ em $\mathcal{D}$ (i.e, o conjunto de estados de $\mathcal{D}$ que satisfazem φ), denotado por $[\![\varphi]\!]_{\mathcal{D}}$, é definida indutivamente como:*

- $[\![p]\!]_{\mathcal{D}} = \{s : p \in \mathcal{L}(s)\}$ $([\![\top]\!]_{\mathcal{D}} = \mathcal{S}$ *and* $[\![\bot]\!]_{\mathcal{D}} = \varnothing)$
- $[\![\neg p]\!]_{\mathcal{D}} = \mathcal{S} \setminus [\![p]\!]_{\mathcal{D}}$
- $[\![(\varphi \wedge \varphi')]\!]_{\mathcal{D}} = [\![\varphi]\!]_{\mathcal{D}} \cap [\![\varphi']\!]_{\mathcal{D}}$
- $[\![(\varphi \vee \varphi')]\!]_{\mathcal{D}} = [\![\varphi]\!]_{\mathcal{D}} \cup [\![\varphi']\!]_{\mathcal{D}}$
- $[\![\exists \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\exists}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\forall \odot \varphi]\!]_{\mathcal{D}} = \mathcal{T}_{\forall}^{-}([\![\varphi]\!]_{\mathcal{D}})$
- $[\![\exists \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y))$
- $[\![\forall \boxdot \varphi]\!]_{\mathcal{D}} = \nu Y.([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y))$
- $[\![\exists(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\exists}^{-}(Y)))$
- $[\![\forall(\varphi \sqcup \varphi')]\!]_{\mathcal{D}} = \mu Y.([\![\varphi']\!]_{\mathcal{D}} \cup ([\![\varphi]\!]_{\mathcal{D}} \cap \mathcal{T}_{\forall}^{-}(Y)))$

4.3.3 Metas de planejamento em α-CTL

Uma meta de alcançabilidade estendida (φ_1 , φ_2) pode ser especificada por uma das três fórmulas α-CTL a seguir, dependendo do tipo de solução desejada:

- $\varphi \doteq \exists(\varphi_1 \sqcup \varphi_2)$, se uma solução fraca é desejada;
- $\varphi \doteq \forall(\varphi_1 \sqcup \varphi_2)$, se uma solução forte é desejada; ou
- $\varphi \doteq \forall \boxdot \exists(\varphi_1 \sqcup \varphi_2)$, se uma solução forte-cíclica é desejada.

4.4 O planejador PACTL

Dado um problema de planejamento $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$, o planejador PACTL (Algoritmo 4.5) primeiramente computa um submodelo $M \subseteq \mathcal{D}$ que satisfaz φ (meta de planejamento especificada em α-CTL). M é um conjunto de estados e pares de estados-ação computado pela função MODELO.

Em seguida, através da função COBERTURA (Linha 2), o algoritmo obtém os estados cobertos pelo submodelo M, isto é, os estados que pertencem ao submodelo M sem nenhuma ação associada. Se o estado inicial s_0 está contido no conjunto de estados cobertos por M (Linha 3), o algoritmo extrai do submodelo M uma política determinística π [1] em que a partir do estado inicial é possível alcançar a meta. Caso contrário ($s_0 \notin C$) o algoritmo devolve *fracasso*.

Algoritmo 4.5: PACTL($\mathcal{P}$) (PEREIRA, 2007)

```
1 M ← MODELO(min, φ)
2 C ← COBERTURA(M)
3 se s0 ∈ C então retorna POLÍTICA(M);
4 retorna fracasso
```

[1] Uma política é determinística se para cada estado $s \in \pi$ existe uma única ação associada.

Algoritmo 4.6: COBERTURA(M) (PEREIRA, 2007)

1 **retorna** $\{s \in \mathcal{S} : s \in M\} \cup \{s \in \mathcal{S} : (s,a) \in M\}$

A função MODELO (Algoritmo 4.7) recebe como entrada uma meta de planejamento φ e um parâmetro e, que indica se o submodelo a ser computado evita ($e = \texttt{min}$) ou permite ciclos ($e = \texttt{max}$), e retorna um submodelo $M \subseteq \mathcal{D}$ sintetizado a partir de φ. Se φ representa uma fórmula proposicional, então a fórmula é tratada diretamente pela função MODELO (linhas 3, 4 e 5).

Algoritmo 4.7: MODELO$[\mathcal{D}](e,\varphi)$ (PEREIRA, 2007)

1 **se** $\varphi \in \mathbb{P}$ **então retorna** $\{s \in \mathcal{S} : \varphi \in \mathcal{L}(s)\}$;
2 **caso** φ **seja**
3 $\quad \neg\varphi$: **retorna** $\mathcal{S} \setminus$ MODELO(e,φ_1)
4 $\quad \varphi_1 \wedge \varphi_2$: **retorna** MODELO$(e,\varphi_1) \cap$ MODELO(e,φ_2)
5 $\quad \varphi_1 \vee \varphi_2$: **retorna** MODELO$(e,\varphi_1) \cup$ MODELO(e,φ_2)
6 $\quad \exists \odot \varphi_1$: **retorna** MODELOEX(e,φ_1)
7 $\quad \forall \odot \varphi_1$: **retorna** MODELOAX(e,φ_1)
8 $\quad \exists \boxdot \varphi_1$: **retorna** MODELOEG(max,φ_1)
9 $\quad \forall \boxdot \varphi_1$: **retorna** MODELOAG(max,φ_1)
10 $\quad \exists \diamondsuit \varphi_2$: **retorna** MODELOEU$(e,\top,\varphi_2)$
11 $\quad \forall \diamondsuit \varphi_2$: **retorna** MODELOAU$(e,\top,\varphi_2)$
12 $\quad \exists(\varphi_1 \sqcup \varphi_2)$: **retorna** MODELOEU(e,φ_1,φ_2)
13 $\quad \forall(\varphi_1 \sqcup \varphi_2)$: **retorna** MODELOAU(e,φ_1,φ_2)

Para tratar os operadores temporais $\exists\odot$ e $\forall\odot$ a função MODELO chama as funções MODELOEX (Algoritmo 4.8), que trata o operador $\exists\odot$, e MODELOAX (Algoritmo 4.9), que trata o operador $\forall\odot$. Estas funções sintetizam um submodelo M_1 a partir da meta φ_1 (Linha 1) e em seguida computam a pré-imagem (I_1) dos estados cobertos pelo submodelo M_1 (C_1). A PRÉIMAGEMFRACA (chamada pela função MODELOEX) computa os estados do domínio que em um passo alcançam os estados pertencentes à M_1. Já a função PRÉIMAGEMFORTE (chamada pela função MODELOAX) computa o conjunto de estados que garantidamente (independente do não determinismo das ações) alcançam os estados pertencentes à M_1 em um passo. Após calcular I_1 as funções MODELOEX e MODELOAX podam de I_1 os estados que já estavam em C_1 (a poda evita que estados $s \in C_1$ sejam mapeados à novas ações). Finalmente, a união do submodelo M_1 (formado pelos estados que satisfazem a propriedade φ) com o conjunto P_1 (formado por um conjunto de pares de estado-ação que alcançam M_1) é devolvido como resposta.

Algoritmo 4.8: MODELOEX$[\mathcal{D}](e,\varphi_1)$ (PEREIRA, 2007)

1 $M_1 \leftarrow$ MODELO(e,φ_1)
2 $I_1 \leftarrow$ PRÉIMAGEMFRACAMAP(COBERTURA(M_1))
3 $P_1 \leftarrow$ PODAX(I_1, M_1)
4 **retorna** $M_1 \cup P_1$

Algoritmo 4.9: MODELOAX$[\mathcal{D}](e,\varphi_1)$ (PEREIRA, 2007)

1 $M_1 \leftarrow$ MODELO(e,φ_1)
2 $I_1 \leftarrow$ PRÉIMAGEMFORTEMAP(COBERTURA(M_1))
3 $P_1 \leftarrow$ PODAX(I_1, M_1)
4 **retorna** $M_1 \cup P_1$

Algoritmo 4.10: PODAX(I_1, M_1) (PEREIRA, 2007)

1 **retorna** $\{(s,a) \in I_1 : s \notin M_1\}$

Algoritmo 4.11: PRÉIMAGEMFRACAMAP$[\mathcal{D}](C)$ (PEREIRA, 2007)

1 **retorna** $\{(s,a) : s \in \mathcal{S},\ a \in \mathbb{A}\ e\ \mathcal{T}(s,a) \cap C \neq \varnothing\}$

Algoritmo 4.12: PRÉIMAGEMFORTEMAP$[\mathcal{D}](C)$ (PEREIRA, 2007)

1 **retorna** $\{(s,a) : s \in \mathcal{S},\ a \in \mathbb{A}\ e\ \varnothing \neq \mathcal{T}(s,a) \subseteq C\}$

As funções auxiliares MODELOEG (Algoritmo 4.13) e MODELOAG (Algoritmo 4.14) tratam dos operadores $\exists\boxdot$ e $\forall$ respectivamente. Estas funções primeiramente computam um submodelo M com os estados que satisfazem a meta φ_1. Em seguida, este modelo é contraído até alcançar um ponto fixo máximo. Durante o processo de contração, a cada iteração as funções calculam a pré-imagem I do modelo (a função MODELOEG calcula a pré-imagem fraca e a função MODELOAG a pré-imagem forte). Após computar I, através da função PODAX, o algoritmo calcula a intersecção dos estados cobertos por M com a pré-imagem I. O resultado da poda passa a ser o modelo corrente e o processo se repete até que um ponto fixo máximo seja alcançado.

Algoritmo 4.13: MODELOEG$[\mathcal{D}](e, \varphi_1)$ (PEREIRA, 2007)

1 $M \leftarrow$ MODELO(e, φ_1)
2 $M' \leftarrow \varnothing$
3 **enquanto** $M' \neq M$ **faça**
4 $\quad M' \leftarrow M$
5 $\quad C \leftarrow$ COBERTURA(M)
6 $\quad I \leftarrow$ PRÉIMAGEMFRACAMAP(C)
7 $\quad M \leftarrow$ PODAG(I, C)
8 **retorna** M

Algoritmo 4.14: MODELOAG$[\mathcal{D}](e, \varphi_1)$ (PEREIRA, 2007)

1 $M \leftarrow$ MODELO(e, φ_1)
2 $M' \leftarrow \varnothing$
3 **enquanto** $M' \neq M$ **faça**
4 $\quad M' \leftarrow M$
5 $\quad C \leftarrow$ COBERTURA(M)
6 $\quad I \leftarrow$ PRÉIMAGEMFORTEMAP(C)
7 $\quad M \leftarrow$ PODAG(I, C)
8 **retorna** M

Algoritmo 4.15: PODAG(I, C) (PEREIRA, 2007)

1 **retorna** $\{(s,a) \in I : s \in C\}$

As funções de ponto fixo mínimo MODELOEU (Algoritmo 4.16) e MODELOAU (Algoritmo 4.17) tratam dos operadores $\exists\sqcup$ e $\forall\sqcup$ respectivamente. Essas funções recebem como parâmetro φ_1, que representa uma meta de planejamento a ser preservada ou evitada ao longo da execução da política, φ_2, que representa uma meta de planejamento a ser alcançada no final da execução de política, e o parâmetro e.

Estas funções primeiramente computam os estados cobertos pelo submodelo que satisfaz a propriedade φ_1 (Linha 1) em seguida calculam o submodelo M_2 que satisfaz a

propriedade φ_2 (Linha 2). Posteriormente, o modelo M_2 é expandido iterativamente até alcançar um ponto fixo. A cada iteração as funções computam a pré-imagem I_2 dos estados cobertos pelo submodelo M_2 (a função MODELOEU calcula a pré-imagem fraca e a função MODELOAU a pré-imagem forte). Em seguida, o conjunto I_2 é podado (através da função PODAU) de modo que apenas estados que satisfazem φ_1 sejam mantidos no pré-componente P_2. Caso ciclos devam ser evitados ($e = \texttt{min}$), a função de poda também elimina de P_2 todos os estados que já pertenciam ao conjunto C_1 antes do cálculo da pré-imagem.

A união do resultado da poda com o submodelo M_2 passa a ser o submodelo M_2 corrente. Este processo se repete até que o ponto fixo seja alcançado.

Algoritmo 4.16: MODELOEU$[\sum](e, \varphi_1, \varphi_2)$ (PEREIRA, 2007)

```
1  C1 ← COBERTURA(MODELO(min, φ1))
2  M2 ← MODELO(e, φ1)
3  M2' ← ∅
4  enquanto M2 ≠ M2' faça
5      M2' ← M2
6      C2 ← COBERTURA(M2)
7      I2 ← PRÉIMAGEMFRACAMAP(C2)
8      P2 ← PODAU(e, I2, C1, C2)
9      M2 ← M2 ∪ P2
10 retorna M2
```

Algoritmo 4.17: MODELOAU$[\sum](e, \varphi_1, \varphi_2)$ (PEREIRA, 2007)

```
1  C1 ← COBERTURA(MODELO(min, φ1))
2  M2 ← MODELO(e, φ1)
3  M2' ← ∅
4  enquanto M2 ≠ M2' faça
5      M2' ← M2
6      C2 ← COBERTURA(M2)
7      I2 ← PRÉIMAGEMFORTEMAP(C2)
8      P2 ← PODAU(e, I2, C1, C2)
9      M2 ← M2 ∪ P2
10 retorna M2
```

Algoritmo 4.18: PODAU(e, I_2, C_1, C_2) (PEREIRA, 2007)

```
1  P2 ← {(s, a) ∈ I2 : s ∈ C1}
2  se e = min então P2 ← {(s, a) ∈ P2 : s ∉ C2} ;
3  retorna P2
```

4.4.1 Propriedades formais do planejador PACTL

Em Pereira (2007) são demostradas as seguintes propriedades formais do planejador PACTL:

Teorema 4.4.1 *Seja $\mathcal{P} = \langle \mathcal{D}, s_0, \exists(\varphi_1 \sqcup \varphi_2) \rangle$ um problema de planejamento. Se $\mathcal{P}$ tem solução, então a política devolvida pelo algoritmo* PACTL*($\mathcal{P}$ é uma solução fraca para $\mathcal{P}$.*

Teorema 4.4.2 *Seja $\mathcal{P} = \langle \mathcal{D}, s_0, \forall(\varphi_1 \sqcup \varphi_2) \rangle$ um problema de planejamento. Se $\mathcal{P}$ tem solução, então a política devolvida pelo algoritmo* PACTL*($\mathcal{P}$ é uma solução forte para $\mathcal{P}$.*

Teorema 4.4.3 *Seja* $\mathcal{P} = \langle \mathcal{D}, s_0, \forall \boxdot \exists(\varphi_1 \sqcup \varphi_2) \rangle$ *um problema de planejamento. Se* $\mathcal{P}$ *tem solução, então a política devolvida pelo algoritmo* PACTL$(\mathcal{P})$ *é uma solução forte-cíclica para* $\mathcal{P}$.

Teorema 4.4.4 *Seja* $\mathcal{P} = \langle \mathcal{D}, s_0, \varphi \rangle$ *um problema de planejamento para uma meta de alcançabilidade estendida* φ. *Então, o algoritmo* PACTL$(\mathcal{P})$ *devolve fracasso se e só se* $\mathcal{P}$ *não tem solução.*

Teorema 4.4.5 *O menor caminho gerado pela execução de uma política* π *obtida por meio da chamada* PACTL$(\langle \mathcal{D}, s_0, \exists(\varphi_1 \sqcup \varphi_2) \rangle)$ *é mínimo no melhor caso.*

Teorema 4.4.6 *O maior caminho gerado pela execução de uma política* π *obtida por meio da chamada* PACTL$(\langle \mathcal{D}, s_0, \forall(\varphi_1 \sqcup \varphi_2) \rangle)$ *é mínimo no melhor caso.*

4.4.2 Exemplo de funcionamento do planejador PACTL

Considere o domínio de planejamento apresentado na Figura 4.6. Suponha que o estado inicial deste problema seja o estado s_0 e a meta de planejamento seja alcançar um estado que satisfaça a propriedade g a partir do estado inicial. Considere que desejamos obter uma solução fraca para este problema. Neste caso, a meta de planejamento pode ser especificada em α-CTL como $\exists \diamond g$. O funcionamento do planejador PACTL para este problema é descrito a seguir:

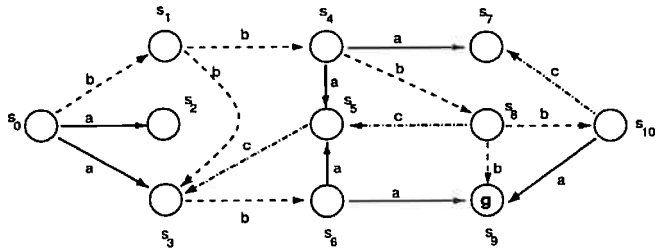


Figura 4.6: *Domínio de planejamento para meta de alcançabilidade simples. Fonte: (Pereira, 2007).*

Para sintetizar o submodelo M, o planejador PACTL chama a função MODELO que por sua vez chama a função MODELOEU (que trata do operador $\exists\diamond$).

- Inicialmente na função MODELOEU temos: $C_1 = \top$, $M_2 = s_9$ (este é um estado do domínio que satisfaz a propriedade g) e $M_2' = \varnothing$
- Após a primeira iteração teremos:
  $M_2' = \{s_9\}$,
  $C_2 = \{s_9\}$,
  $I_2 = \{(s_{10}, a); (s_6, a)\}$,
  $P_2 = \{(s_{10}, a); (s_6, a)\}$ e
  $M_2 = \{s_9; (s_{10}, a); (s_6, a)\}$.
- Após a segunda iteração teremos:
  $M_2' = \{s_9; (s_{10}, a); (s_6, a)\}$,
  $C_2 = \{s_9; s_{10}; s_6\}$,
  $I_2 = \{(s_3, b); (s_8, b); (s_6, a); (s_{10}, a)\}$,
  $P_2 = \{(s_3, b); (s_8, b)\}$ e
  $M_2 = \{s_9; (s_{10}, a); (s_6, a); (s_8, b)\}$.

- Após a terceira iteração teremos:

  $M_2' = \{s_9; (s_{10}, a); (s_6, a); (s_8, b)\}$,

  $C_2 = \{s_9; s_{10}; s_6; s_8\}$,

  $I_2 = \{(s_6, a); (s_{10}, a); (s_3, b); (s_8, b); (s_0, a); (s_1, b); (s_5, c); (s_4, b)\}$,

  $P_2 = \{(s_0, a); (s_1, b); (s_5, c); (s_4, b)\}$, e

  $M_2 = \{s_9; (s_{10}, a); (s_6, a); (s_8, b); (s_0, a); (s_1, b); (s_5, c); (s_4, b)\}$.

A função MODELOEU termina após três iterações uma vez que o submodelo M_2 obtido na terceira iteração é um ponto fixo. A cobertura do submodelo obtido são os estados $C = \{s_9, s_{10}, s_6, s_8, s_0, s_1, s_5, s_4\}$. Como $s_0 \in C$, o algoritmo extrai do submodelo M_2 obtido uma política cuja execução alcança a meta a partir do estado inicial e devolve como solução.

Capítulo 5

Representação e raciocínio baseado em fórmulas lógicas

Este capítulo apresenta como o cálculo de progressão e regressão podem ser feitos de maneira simbólica tanto em problemas de planejamento clássico quanto em problemas FOND. As Fórmulas Booleanas Quantificadas (QBF) fornecem uma notação concisa para fórmulas booleanas. Para raciocinar de maneira simbólica, os estados devem ser representados como fórmulas QBF e o raciocínio progressivo e regressivo realizados através de transformações lógicas sobre estas fórmulas.

5.1 Fórmulas Booleanas Quantificadas

Fórmulas Booleanas Quantificadas (*Quantified Boolean Formulas* - QBF) são uma extensão das fórmulas proposicionais que adicionam o uso dos quantificadores existencial ($\exists$) e universal ($\forall$) (Büning e Bubeck, 2009). Em resumo, podemos dizer que uma fórmula $\exists x\varphi$ é verdadeira se existe algum valor de x (verdadeiro ou falso) que torna φ verdadeira. De forma análoga, uma fórmula $\forall x\varphi$ é verdadeira se para qualquer que seja o valor de x (verdadeiro e falso) φ é verdadeira.

5.1.1 Sintaxe

Definição 5.1.1 (Sintaxe QBF) *Seja $\mathbb{P}$ um conjunto de variáveis proposicionais, a linguagem das fórmulas booleanas quantificadas $\mathcal{L}_{\mathbb{P}}$ é definida como:*

- *Para toda variável proposicional $p \in \mathbb{P}$, $p \in \mathcal{L}_{\mathbb{P}}$;*
- *Para toda constante $t \in \{\bot, \top\}$, $t \in \mathcal{L}_{\mathbb{P}}$;*
- *Se $\varphi \in \mathcal{L}_{\mathbb{P}}$ então $\neg\varphi \in \mathcal{L}_{\mathbb{P}}$;*
- *Se $\varphi_1 \in \mathcal{L}_{\mathbb{P}}$ e $\varphi_2 \in \mathcal{L}_{\mathbb{P}}$ então $\varphi_1 \circ \varphi_2 \in \mathcal{L}_{\mathbb{P}}$ onde $\circ \in \{\wedge, \vee, \rightarrow\}$;*
- *Se $\varphi \in \mathcal{L}_{\mathbb{P}}$, então $Qp\varphi \in \mathcal{L}_{\mathbb{P}}$ onde $Q \in \{\exists, \forall\}$ e $p \in \mathbb{P}$.*

Dado uma fórmula quantificada $Qx\varphi$ onde $Q \in \{\forall, \exists\}$, denotamos por $var(\varphi)$ o conjunto de variáveis pertencentes à φ. Dizemos que $var(\varphi)$ são as variáveis que estão no escopo do quantificador Q e x é a ocorrência quantificada. As variáveis não quantificadas são chamadas de variáveis livres. Quando uma fórmula booleana quantificada não possui nenhuma variável livre dizemos que a QBF é fechada (ver exemplo na Figura 5.1).

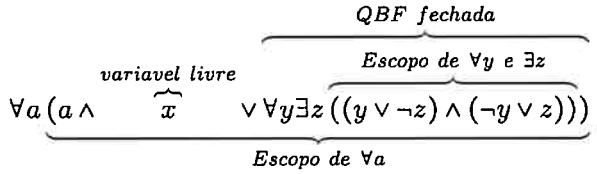


Figura 5.1: *Fórmula Booleana Quantificada.*

5.1.2 Semântica

Definição 5.1.2 (Semântica QBF) *Seja φ uma fórmula booleana quantificada com variáveis livres $z_1, ..., z_n$, uma atribuição verdadeira $\mathfrak{I}$ é um mapeamento $\mathfrak{I} : \{z_1, ..., z_n\} \rightarrow \{0, 1\}$ que satisfaz as seguintes condições:*

- $\varphi = z_i : \mathfrak{I}(\varphi) = \mathfrak{I}(z_i)$, *para* $1 \leq i \leq n$;
- $\varphi = \top : \mathfrak{I}(\varphi) = \top$;
- $\varphi = \bot : \mathfrak{I}(\varphi) = \bot$;
- $\varphi = \neg\varphi' : \mathfrak{I}(\neg\varphi') = \top$ *sse* $\mathfrak{I}(\varphi') = \bot$;
- $\varphi = \varphi_1 \vee \varphi_2 : \mathfrak{I}(\varphi_1 \vee \varphi_2) = \top$ *sse* $\mathfrak{I}(\varphi_1) = \top$ *ou* $\mathfrak{I}(\varphi_2) = \top$;
- $\varphi = \varphi_1 \wedge \varphi_2 : \mathfrak{I}(\varphi_1 \wedge \varphi_2) = \top$ *sse* $\mathfrak{I}(\varphi_1) = \top$ *e* $\mathfrak{I}(\varphi_2) = \top$;
- $\varphi = \forall x\varphi' : \mathfrak{I}(\forall x\varphi') = \top$ *sse* $\mathfrak{I}(\varphi'[\bot/x]) = 1$ *e* $\mathfrak{I}(\varphi'[\top/x]) = \top$; *e*
- $\varphi = \exists y\varphi' : \mathfrak{I}(\exists y\varphi') = \top$ *sse* $\mathfrak{I}(\varphi'[\bot/y]) = \top$ *ou* $\mathfrak{I}(\varphi'[\top/y]) = \top$.

Com base na semântica da linguagem QBF, temos uma série de equivalências das fórmulas booleanas, dentre elas, é importante destacar a eliminação de quantificadores que será utilizada nas operações de progressão e regressão simbólica (Zolda, 2005):

$$\forall x\varphi \equiv \varphi[\bot/x] \wedge \varphi[\top/x];$$
$$\exists x\varphi \equiv \varphi[\bot/x] \vee \varphi[\top/x].$$

5.2 Domínios de planejamento clássico

5.2.1 Representação de estados e ações em QBF

Seja $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$ um domínio de planejamento sobre um conjunto de proposições atômicas $\mathbb{P}$, denotamos por $\mathbb{P}(s)$ o conjunto de literais que descrevem um estado $s \in S$. A codificação de s em QBF é dada pela equação:

$$\xi(s) = \bigwedge_{p \in \mathbb{P}(s)} p. \tag{5.1}$$

Diferentemente da representação STRIPS que faz suposição de mundo fechado, nesta representação fazemos suposição de mundo aberto, isto é, todas os literais que descrevem um estado devem ser declaradas.

A codificação de um conjunto de estados $X \subseteq S$ através de uma fórmula lógica, por sua vez, é definida como:

$$\xi(X) = \bigvee_{s \in X} \xi(s). \tag{5.2}$$

A representação de uma ação determinística a em QBF é um par de fórmulas lógicas $\langle \xi(Precond(a)); \xi(Efeitos(a)) \rangle$ tal que: $\xi(Precond(a))$ é uma conjunção dos literais que

representam as precondições da ação a, e $\xi(Efeitos(a))$ é uma conjunção dos literais que representam os efeitos de a. A Figura 5.2 mostra como representar uma ação a usando QBF:

Ação a:
$\langle \xi(Precond(a)) = \bigwedge_{p \in Precond(a)} p;$
$\xi(Efeitos(a)) = \bigwedge_{t \in Ef^+(a)} t \wedge \bigwedge_{n \in Ef^-(a)} \neg n \rangle$
Mudanças $= \{Ef^+(a) \cup Ef^-(a)\}$.

Figura 5.2: *Representação de ações determinísticas em QBF*

Note que além das precondições e efeitos, para calcular a progressão e regressão em QBF precisamos representar um conjunto com as proposições que pertencem aos efeitos da ação: Mudanças $= \{Ef^+(a) \cup Ef^-(a)\}$.

Seja o conjunto de proposições $\mathbb{P} = \{p, q, t\}$, a Figura 5.3 mostra um exemplo de uma ação determinística a_1 representada por uma fórmula lógica. Esta ação tem como precondição as proposições p e t e como efeitos negativos as proposições p e t. Esta ação não possui efeitos positivos.

Ação a_1:
$\langle \xi(Precond(a_1)) = p \wedge t;$
$\xi(Efeitos(a_1)) = \neg p \wedge \neg t \rangle$
Mudanças$(a_1) = \{p, t\}$.

Figura 5.3: *Exemplo de uma ação determinística representada por fórmulas proposicionais.*

5.2.2 Progressão de ações determinísticas em QBF

Dados um conjunto de estados X e um conjunto de ações determinísticas $\mathcal{A}$ representados por fórmulas proposicionais, Fourman (2000) propôs uma maneira de computar a progressão e regressão de forma simbólica. A progressão de um conjunto de estados X por uma ação a retorna um conjunto de estados Y que pode ser alcançado quando a é executada em estados que estão em X. A equação a seguir mostra como a progressão de um conjunto de estados X através de uma ação a pode ser computada:

$$Progr(\xi(X), a) = \exists Mudanças(a).(\xi(X) \wedge \xi(Precond(a))) \wedge \xi(Efeitos(a)). \quad (5.3)$$

Dado que X representa um conjunto de estados ($X = \{s_1, \ldots, s_n\}$), temos a seguinte equivalência:

$$Progr(\xi(X), a) \equiv \bigvee_{i=1}^{n} Progr(\xi(s_i), a) \quad (5.4)$$

Exemplo 5.2.1 (Progressão QBF) *Seja a ação a_1 (Figura 5.3) e o estado s representado pela fórmula lógica $p \wedge \neg q \wedge t$. A progressão de s por a_1 é calculada como:*

$$\begin{aligned} Progr(s,a_1) &= \exists Mudanças(a_1).(\xi(s) \wedge \xi(Precond(a_1))) \wedge \xi(Efeitos(a_1)) \\ &= \exists p, t((p \wedge \neg q \wedge t) \wedge (p \wedge t)) \wedge \neg p \wedge \neg t \\ &= \exists p, t(p \wedge \neg q \wedge t) \wedge \neg p \wedge \neg t \\ &= \exists t((\bot \wedge \neg q \wedge t) \vee (\top \wedge \neg q \wedge t)) \wedge \neg p \wedge \neg t \\ &= \exists t(\neg q \wedge t) \wedge \neg p \wedge \neg t \end{aligned}$$

$$
\begin{aligned}
&= ((\neg q \wedge \bot) \vee (\neg q \wedge \top)) \wedge \neg p \wedge \neg t \\
&= (\neg q) \wedge \neg p \wedge \neg t \\
&= \neg q \wedge \neg p \wedge \neg t.
\end{aligned}
$$

Logo, dado o estado $\xi(s) = p \wedge \neg q \wedge t$ após a execução da ação a_1 em s, o estado sucessor é $\xi(s') = \neg q \wedge \neg p \wedge \neg t$.

5.2.3 Regressão de ações determinísticas em QBF

Dados um conjunto de estados X e uma ação determinística a representados em QBF, a operação de regressão calcula um conjunto de estados abstratos predecessores Y. A equação a seguir mostra como a regressão de X através de a pode ser computada simbolicamente (Fourman, 2000):

$$Regr(X, a) = \xi(Precond(a)) \wedge \exists Mudanças(a).(\xi(X) \wedge \xi(Efeitos(a))). \tag{5.5}$$

Dado que X representa um conjunto de estados ($X = \{s_1, \ldots, s_n\}$), temos a seguinte equivalência:

$$Regr(\xi(X), a) \equiv \bigvee_{i=1}^{n} Regr(\xi(s_i), a) \tag{5.6}$$

Exemplo 5.2.2 (Regressão QBF) *Seja a ação a_1 (Figura 5.3) e o estado s representado pela fórmula lógica $\neg p \wedge \neg t \wedge \neg q$. A regressão de s por a_1 é calculada como a seguir:*

$$
\begin{aligned}
Regr(s,a_1) &= \xi(Precond(a_1)) \wedge \exists Mudanças(a_1).(\xi(s) \wedge \xi(Efeitos(a))) \\
&= p \wedge t \wedge \exists p, t((\neg p \wedge \neg t \wedge \neg q) \wedge (\neg p \wedge \neg t)) \\
&= p \wedge t \wedge \exists p, t(\neg p \wedge \neg t \wedge \neg q) \\
&= p \wedge t \wedge \exists t((\bot \wedge \neg t \wedge \neg q) \vee (\top \wedge \neg t \wedge \neg q)) \\
&= p \wedge t \wedge \exists t(\neg t \wedge \neg q) \\
&= p \wedge t \wedge ((\bot \wedge \neg q) \vee (\top \wedge \neg q)) \\
&= p \wedge t \wedge (\neg q) \\
&= p \wedge t \wedge \neg q.
\end{aligned}
$$

Logo, $p \wedge t \wedge \neg q$ representa um estado abstrato onde ao executar a ação a_1 é possível alcançar o estado s.

5.2.4 Busca Simbólica

Dado um problema de planejamento clássico $\mathcal{P} = \langle \sum, s_0, \varphi \rangle$, o algoritmo de busca simbólica em largura (Algoritmo 5.1) utiliza operações de progressão para computar os estados sucessores e operações de regressão para recuperar uma política.

Este algoritmo, num primeiro momento constrói uma estrutura em camadas armazenando em cada camada um BDD que representa o conjunto de estados de cada nível da árvore de busca. Além dos conjuntos alcançáveis cada camada armazena também as ações que levam aos sucessores da camada seguinte. O algoritmo para quando alcança a meta.

Cada camada representa um passo do plano solução. Para construir um plano, o algoritmo RECUPERAPLANO (Algoritmo 5.2) visita as camadas construídas durante busca a partir da última camada até alcançar o estado inicial utilizando operações de regressão. A cada passo de regressão o algoritmo escolhe uma ação e adiciona ela ao plano.

5.1

Algoritmo 5.1: BUSCALARGURASIMBOLICA($\mathcal{P}$)

```
1  camada_0 ← ξ(I) // primeira camada contem apenas o estado inicial
2  fechados ← ⊥
3  indice ← 0
4  // repete até que um estado meta seja alcançado
5  enquanto (camada_indice ∧ G) = ⊥ faça
6      camada_indice ← camada_indice ∧ ¬fechados
7      se camada_indice = ⊥ então
8          retorna FALHA
9      // adiciona novos estados ao conjunto de estados já explorados
10     fechados ← fechados ∨ camada_indice
11     camada_indice+1 ← IMAGE(camada_indice)
12     indice ← indice + 1
13 camada_indice ← camada_indice ∧ G
14 retorna RECUPERAPLANO(P, camada, indice)
```

Algoritmo 5.2: RECUPERAPLANO($\mathcal{P}$, *camada*, *indice*)

```
1  π ← ()
2  atual ← camada_indice
3  i ← indice
4  // repete até que um estado meta seja alcançado
5  enquanto (camada_indice ∧ G) = ⊥ faça
6      para i > 1 faça
7          para a ∈ A faça
8              predecessor ← REGR(ATUAL, A)
9              se predecessor ∧ camada_i-1 ≠ ⊥ então
10                 atual ← predecessor ∧ camada_i-1
11                 π.INSERE(a)
12                 encerra loop
13 retorna π
```

5.3 Domínios de planejamento FOND

Um dos principais planejadores baseados em verificação simbólica de modelos é o MBP (Bertoli *et al.*, 2001). Diversos planejadores simbólicos que existem são baseados no MBP. Tanto o MBP quanto os planejadores baseados nele, representam explicitamente os conjuntos de estados e as possíveis transições que ocorrem entre eles. Esta abordagem é amplamente utilizada por muitos planejadores (Cimatti *et al.*, 2003; Giunchiglia e Traverso, 2000; Pereira e de Barros, 2008).

Outra forma de representar e raciocinar simbolicamente é utilizando a representação implícita. Ao contrário da representação explícita, nesta abordagem, representamos apenas as mudanças que podem ocorrer no ambiente através da execução das ações, que são descritas por meio de suas pré-condições e efeitos.

As seções que seguem descrevem com mais detalhes cada uma das abordagens.

5.3.1 Raciocínio Implícito e Simbólico

Representação de estados e ações FOND em QBF

A representação em QBF de uma ação não determinística a é um par de fórmulas lógicas $\langle \xi(Precond(a)); \xi(Efeitos(a)) \rangle$ tal que: $\xi(Precond(a))$ é uma conjunção dos literais que representam as precondições da ação a, e $\xi(Efeitos_{ND}(a))$ é uma disjunção de todos os efeitos não determinísticos de a:

Ação a:

$$\langle \xi(Precond(a)) = \bigwedge_{p \in Precond(a)} p;$$

$$\xi(Efeitos_{ND}(a)) = \bigvee_{e_i \in Efeitos(a)} \Big(\bigwedge_{t \in Ef^{+}(a,e_i)} t \wedge \bigwedge_{n \in Ef^{-}(a,e_i)} \neg n \Big) \rangle$$

Figura 5.4: *Representação de ações não determinísticas em QBF.*

Para computar a progressão e regressão de ações não determinísticas, cada um dos efeitos da ação deve possuir o seu próprio conjunto de *Mudanças* com as proposições que ocorrem nesse efeito. Assim, sendo a uma ação não determinística e $Efeitos(a) = \{e_1, ..., e_n\}$ o conjunto dos efeitos não determinísticos de a, então para cada $e_i \in Efeitos(a)$ há um conjunto $Mudanças(a, e_i)$ correspondente. O conjunto de todas as proposições envolvidas nos efeitos não determinísticos de uma ação a é obtido pela união dos conjuntos $Mudanças(a, e_i)$:

$$Mudanças(Efeitos(a)) = \bigcup_{e_i \in Efeitos(a)} Mudanças(a, e_i). \tag{5.7}$$

Progressão de ações FOND em QBF

A progressão simbólica (Menezes, 2014) de um conjunto de estados X (representado por uma fórmula lógica conforme descrito na Equação 5.2) por uma ação não determinística a pode ser computada da mesma maneira que a progressão simbólica para ações determinísticas (Equação 7.1). Porém, no caso não determinístico as ações são representadas conforme descrito na Figura 5.4.

Regressão de ações FOND em QBF

A operação de regressão por uma ação $a \in \mathcal{A}$ é similar à operação de pré-imagem (Figura 4.2), porém é feita sobre as fórmulas que descrevem as ações, isto é, em termos de precondições e efeitos.

Dado um conjunto de estados X e uma ação não determinística a, a regressão fraca computa um estado abstrato que possivelmente alcança X. A regressão forte de X computa um estado abstrato que garantidamente (a despeito do não determinismo) alcança X

A Figura 5.3.1 mostra como a regressão fraca e forte podem ser computadas por meio de fórmulas lógicas considerando uma representação implícita (Menezes, 2014).

$$RegrFraca(\xi(X), a) = \xi(Precond(a)) \wedge \exists Mudanças(a).(\xi(X) \wedge \xi(Efeitos(a)))$$
$$RegrForte(\xi(X), a) = \xi(Precond(a)) \wedge \forall Mudanças(a).(\xi(Efeitos(a)) \rightarrow \xi(X))$$

Figura 5.5: *Operações simbólicas de regressão fraca e forte.*

Dado um domínio de planejamento, o cálculo da regressão fraca por todas as ações do domínio é equivalente à um passo de pré-imagem fraca; e o cálculo da regressão forte se-

gundo todas as ações do domínio é equivalente à um passo de pré-imagem forte (Rintanen, 2008).

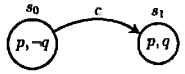
5.3.2 Raciocínio Explícito e Simbólico

Representação Proposicional para Transições

Em um sistema de transição de estados rotulados por ações com assinatura $(\mathbb{P}, \mathbb{A})$ uma transição t é dada por (s, a, s'). Uma transição é representada simbolicamente conforme a equação a seguir:

$$\xi(t) = \xi(s) \wedge \xi(a) \wedge \xi(s') \tag{5.8}$$

Na Equação 5.3.2 $\xi(a)$ representa o rótulo de uma ação do domínio. A Figura 5.6 apresenta um exemplo de uma transição representada por meio de fórmulas lógicas.



(a) *Transição entre estados.*

$$t = (s_0 \wedge a \wedge s_1)$$
$$\xi(t) = (p \wedge \neg q \wedge c \wedge p' \wedge q')$$

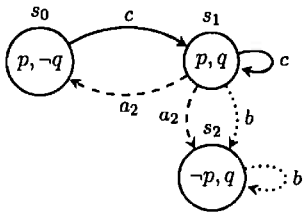
(b) *Fórmula lógica que representa a transição.*

Figura 5.6: *Exemplo de representação simbólica para uma transição.*

Um conjunto de transições é representado conforme a equação a seguir:

$$\xi(\mathcal{T}) = \bigvee_{t \in \mathcal{T}} \xi(t) \tag{5.9}$$

A Figura 5.7 apresenta um exemplo de representação de um conjunto de transições por meio de fórmulas lógicas.



(a) *Transição entre estados.*

$$\mathcal{T} = \{(s_0, c, s_1), (s_1, c, s_1), (s_1, a_2, s_0), (s_1, a_2, s_2), (s_1, b, s_2), (s_0, c, s_1), (s_2, b, s_2)\}$$

$$\xi(\mathcal{T}) = (p \wedge \neg q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge c \wedge p' \wedge q') \vee (p \wedge q \wedge a_2 \wedge p' \wedge \neg q') \vee (p \wedge q \wedge a_2 \wedge \neg p' \wedge q') \vee (p \wedge q \wedge b \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge b \wedge p' \wedge \neg q')$$

(b) *Fórmula lógica que representa a transição.*

Figura 5.7: *Exemplo de representação simbólica para um conjunto de transições.*

Imagem Simbólica de um Conjunto de Estados

Seja um domínio de planejamento $\mathcal{D}$ com assinatura $(\mathbb{P}, \mathbb{A})$ e um conjunto de variáveis $\mathcal{A}$ referente as ações em $\mathbb{A}$.

A operação de imagem de um conjunto de estados X representados por uma uma formula proposicional $(\xi(X))$, dado a representação proposicional das transições do domínio $(\xi(\mathcal{T}))$ computa o conjunto de estados sucessores de X.

Esta operação pode ser computada conforme a equação a seguir:

$$\text{IMAGEM}(X) = \exists \mathcal{A}. \exists \mathbb{P}. (\xi(\mathcal{T}) \wedge \xi(X)). \tag{5.10}$$

Pré-Imagem Simbólica de um Conjunto de Estados

As operações de pré-imagem fraca e forte apresentadas na Figura 4.2 podem ser computadas de maneira simbólica. Os cálculos de pré-imagem por meio de fórmulas lógicas são apresentados na Figura 5.3.2.

$$\textsc{PreImagemFraca}(X) = \exists \mathcal{A} \exists \mathbb{P}'(\xi(\mathcal{T}) \wedge \xi(X'))$$
$$\textsc{PreImagemForte}(X) = \exists \mathcal{A}(\forall \mathbb{P}'(\xi(\mathcal{T}) \rightarrow \xi(X'))) \wedge \exists \mathbb{P}'(\xi(\mathcal{T}))$$

Figura 5.8: *Operações de pré-imagem simbólica considerando ações.*

Capítulo 6

Diagramas de Decisão Binária

Um Diagrama de Decisão Binária (*Binary Decision Diagram* - BDD) é uma estrutura de dados que representa funções booleanas de forma compacta e possibilita realizar operações sobre estas funções de maneira eficiente.

As ideias básicas de BDDs foram introduzidas por Lee (1959), em 1978 Akers (1978) apresentou os BDDs com o nome que utilizamos até hoje, bem como abordagens para gerar BDDs a partir da Expansão de Shannon. A estrutura proposta por Akers ainda possuía algumas redundâncias, então, em 1986 Bryant (1986) impôs restrições sobre a ordem das variáveis booleanas apresentando os BDDs ordenados e reduzidos bem como algoritmos eficientes para manipular tais estruturas. Os BDDs ordenados e reduzidos se mostraram uma abordagem promissora sendo uma alternativa às representações clássicas de funções booleanas antes feitas através de tabelas verdades, que são estrutura muito ineficientes em termos de espaço.

Os BDDs ordenados e reduzidos são amplamente utilizados na implementação de circuitos, verificação formal, planejamento entre outras aplicações. Neste trabalho, essas estruturas são utilizadas para codificar os estados e ações de um domínio de planejamento e realizar as operações de progressão e regressão definidas através das fórmulas booleanas quantificadas.

6.1 Formalização dos BDDs

Formalmente, um BDD é um grafo direcionado acíclico (DAG) que representa funções $f : \{0,1\}^n \rightarrow \{0,1\}$. Um BDD pode ser descrito como o resultado de simplificações em uma árvore de decisão. Uma árvore de decisão, por sua vez, é uma estrutura que possui nós de decisão rotulados com variáveis booleanas e nós terminais rotulados com 1 (verdadeiro) ou 0 (falso). Cada nó n de decisão possui dois nós sucessores: $l(n)$ que está ligado à n por um arco tracejado representando a atribuição do valor falso à variável n, e $h(n)$ que está ligado a n por um arco cheio representando a atribuição do valor verdade à n (Huth e Ryan, 2004).

Definição 6.1.1 (Satisfatibilidade de uma fórmula booleana) *Dado uma árvore de decisão que representa uma fórmula booleana f e uma valoração para as variáveis que ocorrem em f, tal valoração é satisfeita se existe um caminho que inicia no nó raiz da árvore e termina em um nó folha rotulado com 1. Para percorrer a árvore, sempre que a variável possuir um valor falso, deve-se escolher a linha tracejada. Caso contrário, a linha cheia é escolhida.*

Embora árvores de decisão sejam muito úteis para representar funções booleanas, em geral, essa estrutura possui muita redundância. Quando tais redundâncias são eliminadas, dá-se o nome para estrutura resultante dessas eliminações de diagrama de decisão binária.

A exclusão de testes redundantes de uma árvore de decisão consiste no compartilhamento de subgrafos isomorfos presentes na estrutura.

A Tabela 6.1 apresenta a tabela verdade para a expressão booleana $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$ e a Figura 6.1 um exemplo de uma árvore de decisão e de um BDD para a mesma expressão.

x_1	x_2	x_3	$(x_1 \vee x_2) \wedge (x_2 \vee x_3)$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

Tabela 6.1: *Exemplo de tabela verdade para fórmula* $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$.

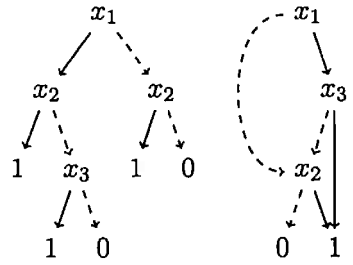


Figura 6.1: *Exemplo de árvore de decisão e BDD, respectivamente, para fórmula* $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$.

6.1.1 BDD ordenado e reduzido

O termo BDD, em geral, refere-se a diagrama de decisão binária ordenado e reduzido (*Reduced Ordered Binary Decision Diagram* - ROBDD). Um ROBDD (Bryant, 1986) é um BDD que possui uma ordenação para suas variáveis e não possui redundância em suas sub-estruturas.

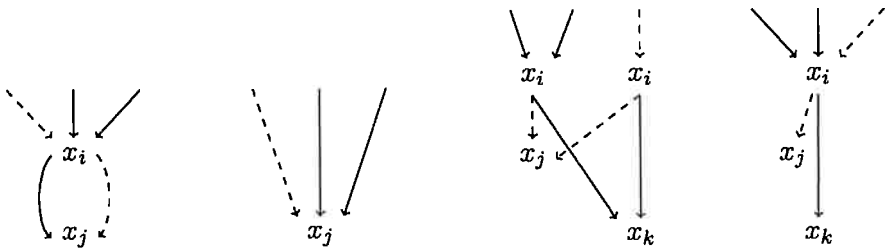
Definição 6.1.2 (Diagrama de decisão binária ordenado) *Dado um conjunto de variáveis booleanas* $\{x_1, x_2, \ldots, x_n\}$, *um* BDD *é ordenado se todos os caminhos da estrutura respeitam uma determinada ordenação para este conjunto* $(x_1 < x_2 < \ldots, < x_n)$.

Segue da definição 6.1.2 que em um BDD ordenado não existe uma variável que ocorre mais de uma vez em um determinado caminho.

A eliminação de redundâncias em um BDD ordenado pode ser resumida em duas regras:

1. Eliminação de nós com sucessores iguais: se um nó não terminal n possui $l(n) = h(n)$, então n deve ser eliminado e os arcos de entrada redirecionados para $l(n)$. Para cada nó eliminado, um par de aresta é eliminado. Um exemplo desta regra de redução pode ser visto na Figura 6.2a.

2. Eliminação de nós duplicados: se os nós não terminais n_1 e n_2 representam a mesma variável booleana, $h(n_1) = h(n_2)$ e $l(n_1) = l(n_2)$, então um dos nós deve ser eliminado e todos seus os arcos de entrada redirecionados para o nó não eliminado. Um exemplo desta regra de redução pode ser visto na Figura 6.2b.



(a) *Eliminação de nós com sucessores iguais em um* BDD *ordenado.*

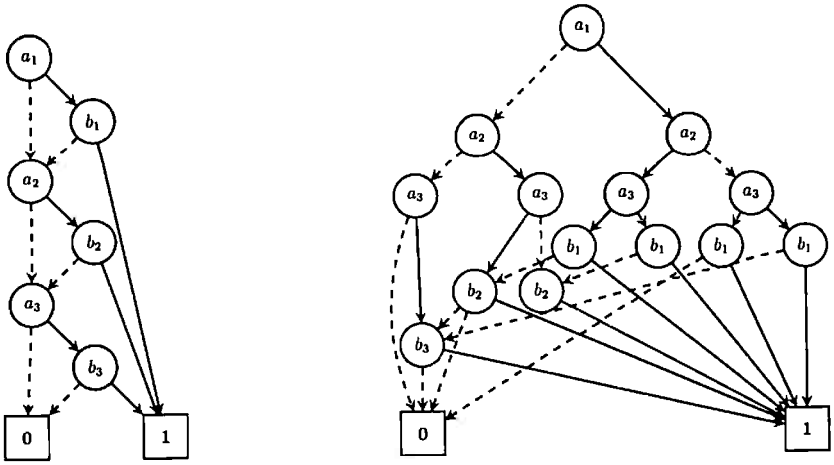
(b) *Eliminação de nós duplicados em um* BDD *ordenado.*

Figura 6.2: *Exemplo de reduções em* BDDs *ordenados*

Quando mais nenhuma das reduções apresentadas puderem ser aplicadas em um BDD ordenado, tem-se um BDD ordenado e reduzido. Além da representação compacta, dada uma determinada ordenação para um conjunto de variáveis $\mathcal{V}$, para para toda expressão booleana existe um único ROBDD que a representa. Deste ponto em diante, por simplicidade, quando a notação BDD for utilizada, na verdade, faz-se referência a ROBDD.

Ordenação das variáveis

Para muitas funções booleanas o BDD correspondente pode ser de tamanho polinomial ou mesmo linear. Contudo, o tamanho do BDD depende da escolha da ordem das variáveis. A Figura 6.3 apresenta dois BDDs que representam a mesma função, porém possuem ordem de variáveis diferentes. Através deste exemplo é possível perceber o impacto na escolha da ordenação das variáveis, porém, escolher a melhor ordenação para um dado conjunto de variáveis é um problema computacionalmente difícil (Sieling, 2002). Para minimizar este custo existem algumas heurísticas que auxiliam na escolha da ordenação (Butler *et al.*, 1991).



(a) BDD *que representa a função* $a_1 \wedge b_1 \vee a_2 \wedge b_2 \vee a_3 \wedge b_3$ *e possui a seguinte ordenação de variáveis:* $a_1 < b_1 < a_2 < b_2 < a_3 < b_3$.

(b) BDD *que representa a função* $a_1 \wedge b_1 \vee a_2 \wedge b_2 \vee a_3 \wedge b_3$ *e possui a seguinte ordenação de variáveis:* $a_1 < a_2 < a_3 < b_1 < b_2 < b_3$.

Figura 6.3: *Exemplos de* BDDs *que representam a mesma função porém possuem ordenação de variáveis diferentes. Figura extraída de Bryant (1986).*

6.2 Algoritmos para BDDs

6.2.1 O algoritmo aplicar

Dados dois BDDs $\mathcal{B}_1$ e $\mathcal{B}_2$ que representam, respectivamente, as fórmulas f_1 e f_2, e possuem a mesma ordenação de variáveis e um operador *op* ($\wedge$, $\vee$ ou $\rightarrow$), a operação aplicar computa um BDD que representa $\mathcal{B} = (f_1\ op\ f_2)$.

O algoritmo aplicar constrói uma fórmula a partir de outras duas de maneira recursiva baseando-se na expansão de Shannon (Shannon, 1938).

Definição 6.2.1 (Expansão de Shannon) *Dado uma fórmula booleana f e uma variável x, a expansão de Shannon é dada por:*

$$f \equiv (\neg x \wedge f[0/x]) \vee (x \wedge f[1/x]). \tag{6.1}$$

Na Equivalência 6.1, $f[0/x]$ é a substituição da variável x por 0 e $f[1/x]$ é a substituição da variável x por 1.

Com base na expansão de shannon, a expansão para composição de funções é definida conforme a Fórmula 6.2. Esta fórmula é usada de maneira recorrente pelo algoritmo aplicar a partir das raízes dos BDDs $\mathcal{B}_1$ e $\mathcal{B}_2$ até alcançar os nós folhas.

$$f_1 \text{ op } f_2 = \neg x_i \wedge (f_1[0/x_i] \text{ op } f_2[0/x_i]) \vee x_i \wedge (f_1[1/x_i] \text{ op } f_1[2/x_i]). \tag{6.2}$$

Sejam r_1 e r_2 raízes dos BDDs $\mathcal{B}_1$ e $\mathcal{B}_2$ respectivamente, a ideia geral do algoritmo é descrita a seguir e ilustrada na Figura 6.4:

1. Se r_1 ou r_2 (ou ambos) são nós terminais então $\mathcal{B}_1$ *op* $\mathcal{B}_2$ pode ser calculada diretamente;

2. Se r_1 e r_2 são nós intermediários e representam a mesma variável booleana x, então um novo nó que representa a variável x é criado com um arco tracejado que aponta para a chamada recursiva `aplicar(op, l(`r_1`), l(`r_2`)` e um arco cheio que aponta para a chamada `aplicar(op, h(`r_1`), h(`r_2`))`. As chamadas recursivas estão baseadas na Fórmula 6.2;

3. Se r_1 e r_2 são nós intermediários e representam, respectivamente as variáveis booleanas x_1 e x_2 sendo a ordem das variáveis $x_1 < x_2$, então um novo nó que representa a variável x_1 é criado com um arco tracejado que aponta para a chamada recursiva `aplicar(op, l(`r_1`), `r_2`)` e um arco cheio que aponta para a chamada `aplicar(op, h(`r_1`), `r_2`)`.

Após a execução do algoritmo aplicar, os passos para reduzir um BDD, descritos anteriormente, devem ser aplicados para garantir que o BDD resultante também seja reduzido. Uma otimização deste algoritmo, apresentada em (Bryant, 1986), é armazenar os resultados das chamadas recursivas de modo que uma mesma operação não seja computada mais de uma vez.

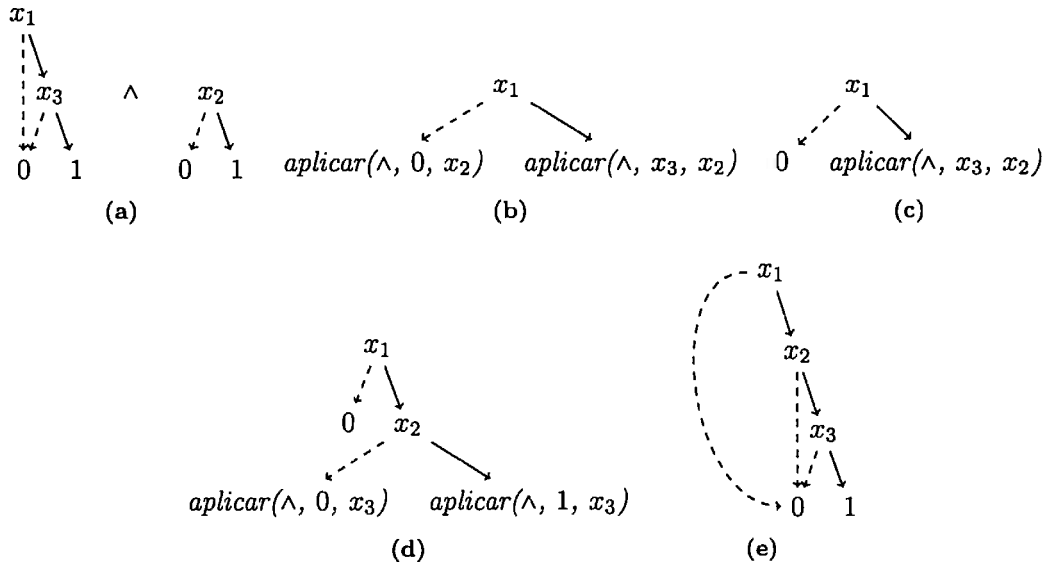


Figura 6.4: *Exemplo de execução do algoritmo aplicar considerando a seguinte ordenação das variáveis:* $x_1 < x_2 < x_3$

6.2.2 O algoritmo restringir

Seja f uma fórmula booleana e $\mathcal{B}$ o BDD que a representa, a operação restringir, computa um BDD reduzido após a substituição da variável $x \in f$ pelo valor verdadeiro ou falso.

- Para computar o BDD reduzido que representa $f[0/x]$, para cada nó n de $\mathcal{B}$ que presenta a variável booleana x, os arcos de entrada são redirecionados para $l(n)$ e o nó removido.
- De forma análoga ao item anterior, para computar o BDD reduzido que representa $f[1/x]$, para cada nó n de $\mathcal{B}$ que presenta a variável booleana x, os arcos de entrada são redirecionados para $h(n)$ e o nó removido.

6.2.3 O algoritmo quantificação

Com base nos algoritmos restringir e aplicar é possível implementar algoritmos para a quantificação universal e existencial.

Seja $\mathcal{B}$ um BDD que representa uma função booleana f em que ocorre a variável x, a operação $\exists x.f$ (que pode ser expressa pela equivalência $f[0/x] \vee f[1/x]$) pode ser implementada como a combinação de duas chamadas para o algoritmo restringir passando como parâmetro os valores verdadeiro e falso ($\mathcal{B}_0 = restringir(0, x, \mathcal{B})$ e $\mathcal{B}_1 = restringir(1, x, \mathcal{B})$ respectivamente). Por fim, o resultado das duas chamadas deve ser combinado em uma chamada para operação aplicar passando como parâmetro o operador **ou** ($\vee$): $\mathcal{B}_\exists = aplicar(\vee, \mathcal{B}_0, \mathcal{B}_1)$.

A quantificação universal ($\forall x.f \equiv f[0/x] \wedge f[1/x]$) pode ser implementada da mesma forma apenas substituindo o operador passado como parâmetro para operação aplicar pelo operador **e** ($\wedge$).

Parte II

O planejador PACTL-SYM

Capítulo 7

Síntese de soluções fracas e fortes

O PACTL-SYM (*Planejador Alpha*-CTL *Simbólico*), proposto neste trabalho de mestrado, é a versão simbólica dos algoritmos de síntese das políticas fraca, forte e forte cíclica do planejador PACTL, apresentados no Capítulo 4.

Os algoritmos de síntese do PACTL-SYM recebem como entrada um problema de planejamento FOND $\mathcal{P}$ e devolvem uma política que satisfaz a meta de planejamento, se existe uma solução, ou falha, caso contrário. Formalmente, a entrada do planejador PACTL-SYM é um problema de planejamento $\mathcal{P} = \langle \mathcal{A}, s_0, \phi \rangle$, onde:

- $\mathcal{A}$ é um conjunto de ações não determinísticas descritas em termos de suas pré-condições e efeitos (Figura 5.4);
- s_0 é o estado inicial representado por uma fórmula QBF, conforme a Equação 5.1; e
- ϕ é a meta de planejamento especificada em α-CTL.

Note que é possível construir o espaço de estados apenas com a especificação do estado inicial, da meta e do conjunto de ações descrito em termos de suas pré-condições e efeitos..

Para sintetizar uma solução, o PACTL-SYM, num primeiro momento, computa um submodelo $M_\pi \subseteq \mathcal{D}$ (M_π é um subconjunto do domínio de estados e ações que podem alcançam a meta). Se $s_0 \subseteq M_\pi$ então uma política π, extraída de M_π, é devolvida, caso contrário o planejador devolve falha.

Diferentemente do planejador PACTL em que o submodelo computado é um conjunto de pares estado-ação, no PACTL-SYM o submodelo não é representado por um conjunto de estados individuais juntamente com as ações aplicáveis em cada estado. Isso porque o planejador PACTL-SYM raciocina sobre conjuntos de estados (representados conforme a Equação 5.2) e explora o espaço de estados utilizando operações de regressão fraca e forte conforme apresentado na Figura 5.3.1. Computacionalmente, os estados e ações são representados por BDDs, sendo que cada proposição do problema é uma variável do BDD.

Assim como o algoritmo de busca simbólica apresentado no Capítulo 5 (Algoritmo 5.1), nossos algoritmos também necessitam criar uma estrutura em camadas para construir uma política. Esta estrutura em camadas é criada na construção do submodelo M_π sendo que cada representa os estados e ações computados em um passo de pré-imagem (cada camada da estrutura representa uma parte do submodelo). Uma camada M_{π_i} é uma estrutura tal que $M_{\pi_i} = \langle \xi(X)^{BDD}, acoes \rangle$ em que:

- $\xi(X)^{BDD}$ é um conjunto de estados X dado por uma fórmula lógica e representado computacionalmente por um BDD; e
- *acoes* é um conjunto de ações (uma lista de ações) que podem ser aplicadas em algum estado de X.

Uma vez que um submodelo foi computado, para extrair uma política o algoritmo percorre as camadas, através de operações de progressão, a partir da primeira camada que contem o estado inicial até alcançar um estado. A cada passo de progressão o algoritmo associa um estado à uma ação. O resultado deste processo é um conjunto de pares estado-ação, ou seja, uma política.

As seções que seguem apresentam em mais detalhes os algoritmos de síntese para políticas fracas e fortes.

7.1 Síntese da política fraca

O Algoritmo 7.1 como sintetizar uma solução fraca ($\exists \Diamond \varphi$).

A construção do submodelo parte dos estados que satisfazem a meta (Linha 1). A primeira camada do submodelo M_π armazena o conjunto de estados $\xi(X)^{BDD}$ que satisfazem $\mathcal{G}$ (Linha 2). A partir do conjunto $\xi(X)^{BDD}$, o algoritmo expande o submodelo iterativamente através de operações de regressão fraca. A cada iteração é computada a regressão fraca por todas as ações $a \in \mathcal{A}$ (Linha 13), através da função auxiliar RegrFraca. Esta função computa a regressão fraca conforme a Figura 5.3.1 e retorna um BDD que representa um conjunto de estados abstratos $\xi(Y)^{BDD}$ que alcançam o conjunto de estados abstratos $\xi(X)^{BDD}$. Note que o cálculo da regressão fraca por todas as ações do problema é equivalente à um passo de pré-imagem fraca (Rintanen, 2008).

O conjunto de estados predecessores $\xi(Y)^{BDD}$ bem como as ações relevantes para o conjunto $\xi(X)^{BDD}$ são armazenados em uma nova camada do submodelo (Linha 22). A próxima iteração i computará os estados predecessores e as ações relevantes para o conjunto recém computado (armazenado na camada $i-1$) de forma que cada nova camada possui um conjunto de estados que pode alcançar o conjunto de estados da camada anterior através das ações armazenadas na camada em questão (camada i).

O algoritmo para quando alcança um ponto fixo. O ponto fixo é alcançado quando todos os estados de uma camada estão contidos em camadas anteriores ou quando não existe nenhuma ação aplicável na camada que deve ser explorada (neste caso, o resultado das regressões será $\xi(X)^{BDD} = \bot$).

Para verificar se um estado abstrato está em alguma camada, o algoritmo executa a função auxiliar Contem (Linha 14). Esta função (Algoritmo 7.2) percorre as camadas do submodelo M_π verificando se um estado abstrato $\xi(R)^{BDD}$ está em alguma camada.

Se o estado inicial está contido em alguma camada do submodelo computado, então existe uma política fraca para o problema. Para extrair a política, o algoritmo de extração (Algoritmo 7.3) visita as camadas do submodelo computado a partir do estado inicial através de operações de progressão.

Dado um submodelo com n camadas, se o estado inicial s_0 é representado por um conjunto de estados de uma camada n_i, o algoritmo seleciona uma ação a da camada n_i e, para cada efeito não determinístico de a, computa os estados sucessores de s_0 (Linha 13). Para cada estado sucessor s' gerado, o algoritmo procura a camada de menor índice (mais próxima da primeira camada) que contém s' (Linha 14) e armazena em uma lista de estados que farão parte da política, o estado s' bem como o índice da camada em que s' se encontra. O índice da camada em que os estados gerados estão são salvos para que nas próximas iterações o algoritmo saiba em qual conjunto de ações buscar a melhor ação para s' (Linha 7). Quando estado s' gerado é um beco-sem-saída, isto é, não está em nenhuma camada e portanto não possui nenhuma ação aplicável, ou é um estado que satisfaz a meta ou já está mapeado na política π (Linha 15), s' não será inserido na lista de estados que farão parte da política.

Se ao menos um estado s' gerado está na camada seguinte à camada que s se encontra (camada n_{i-1}), então a ação a é associada ao estado s (Linha 20). Caso nenhum dos estados gerados pela ação escolhida para um determinado estado s levam à camada seguinte, o

Algoritmo 7.1: SÍNTESEFRACA($\mathcal{P}$)

```
1  ξ(X)^BDD ← φ // conjunto de estados que satisfazem a meta
2  M_π1 ← ⟨ξ(X)^BDD, ∅⟩
3  se (ξ(s_0) ∧ ξ(X)^BDD) ≠ ⊥ então
4      retorna M_π // estado inicial satisfaz a meta
5  i = 2
6  C_s0 ← 0
7  pontofixo ← ⊥
8  enquanto pontofixo ≠ ⊤ e ξ(X)^BDD ≠ ⊥ faça
9      ξ(Y)^BDD ← ⊥
10     acoes ← ∅
11     pontofixo ← ⊤
12     para a ∈ A faça
13         ξ(R)^BDD ← REGRFRACA(ξ(X), a)
14         se CONTEM(M_πi, ξ(R)^BDD) = FALSO então
15             pontofixo ← ⊥
16         se ξ(R)^BDD ≠ ⊥ então
17             ξ(Y)^BDD ← ξ(Y)^BDD ∨ ξ(R)^BDD
18             acoes.INSERE(a)
19     se (ξ(s_0) ∧ ξ(Y)^BDD) ≠ ⊥ e C_s0 = 0 então
20         C_s0 ← i // índice da primeira camada que contém o estado inicial
21     i ← i + 1
22     M_πi ← (⟨ξ(Y)^BDD, acoes⟩)
23     ξ(X)^BDD ← ξ(Y)^BDD
24 se C_s0 = 0 então
25     retorna FALHA
26 senão
27     retorna EXTRAIPOLITICA(P, M_π, C_s0)
```

algoritmo ignora os estados gerados e escolhe outra ação da camada em que s se encontra. Como o conjunto de ações foi gerado a partir de operações de regressão, é garantido que existe ao menos uma ação no conjunto de ações da camada em que s se encontra que, ao ser executada no estado s levará para algum estado contido na camada seguinte.

Este processo progressivo para escolha de ações se repete até que todos os estados gerados ($estados_politica$) sejam mapeados à uma ação.

Dado um estado s e uma ação a, a operação de progressão é computada pelo algoritmo de extração da política (Linha 13) da seguinte forma:

$$Progr(s, a, e_i) = \exists Mudanças(a, e_i).(\xi(s) \wedge \xi(Precond(a))) \wedge \xi(Efeitos(a, e_i)). \quad (7.1)$$

A Figura 7.1 apresenta um exemplo esquemático do funcionamento do algoritmo de construção do submodelo para uma a fórmula $\exists \Diamond \varphi$ sendo que $s_g \vDash \varphi$. A Figura 7.2 apresenta o funcionamento do algoritmo de extração de política considerando o submodelo construído na figura anterior. A seção 7.4 apresenta um exemplo em termos de fórmulas lógicas para o algoritmo de síntese da política fraca.

A estrutura em camadas é importante para conseguirmos construir uma política, uma vez que os estados não são representados individualmente. Outros trabalhos que utilizam as mesmas operações de progressão e regressão que utilizamos mas que não necessitam

Algoritmo 7.2: CONTEM($M_\pi, \xi(R)^{BDD}$)

```
1  para M_{π_i} ∈ M_π faça
2      ξ(X)^BDD ← M_{π_i}.ESTADOS
3      se (ξ(X)^BDD ∧ ξ(R)^BDD) = ξ(X)^BDD então
4          retorna VERDADEIRO
5  retorna FALSO
```

Algoritmo 7.3: EXTRAIPOLITICA($\mathcal{P}, M_\pi, C_{s_0}$)

```
1  π ← ∅
2  estados_politica ← ∅
3  c_i ← C_{s_0} // índice da camada que contém o estado inicial
4  estados_politica.INSERE(⟨ξ(s_0)^BDD, c_i⟩)
5  enquanto estados_politica ≠ ∅ e c_i ≠ 0 faça
6      ⟨ξ(s)^BDD, c_i⟩ ← estados_politica.PRIMEIROELEMENTO
7      acoes ← M_{π_{c_i}}.AÇÕES
8      a ← ∅
9      enquanto a = ∅ faça
10         novos_estados ← ∅
11         a ← SELECIONAAÇÃO(acoes)
12         para e_i ∈ Efeitos(a) faça
13             ξ(s')^BDD ← PROGR(ξ(s), a, e_i)
14             c_j ← ENCONTRACAMADADOESTADO(M_π, ξ(s')^BDD)
15             se c_j > 0 e s' ∉ π então
16                 // verifica se s' não é meta e não está na política
17                 novos_estados.INSERE(⟨ξ(s')^BDD, c_j⟩)
18         se algum estado s' ∈ novos_estados está na camada c_i − 1 então
19             estados_politica.INSERE(novos_estados)
20             π.INSERE(⟨s, a⟩)
21         senão
22             a ← ∅
23 retorna π
```

construir uma política não precisam criar tal estrutura.

Simão (2017), por exemplo, utiliza as operações de regressão para construir um conjunto de estados que alcançam a meta (não são estados becos-sem-saída). O processo de construção deste conjunto é semelhante ao algoritmo 7.1, porém ao invés de separar os conjuntos de estados em camadas a cada iteração, o resultado de todas as iterações até alcançar um ponto-fixo é armazenado em um único BDD.

7.2 Síntese da política forte

O Algoritmo 7.4 apresenta como sintetizar uma solução forte ($\forall \diamond \varphi$). Este algoritmo é muito similar ao algoritmo de síntese da política fraca. A principal diferença entre eles é que, para computar uma solução forte, ao invés de construir o submodelo utilizando operações de regressão fraca, o submodelo é construído com operações de regressão forte.

A regressão forte (Linha 11), computada pela função auxiliar REGRFORTE, é calculada conforme a Figura 5.3.1. Assim como o cálculo da regressão fraca, o cálculo de regressão

forte todas as ações do problema é equivalente à um passo de pré-imagem forte (Rintanen, 2008).

Diferentemente do algoritmo de síntese fraca, a computação do submodelo M_π para quando alcança uma camada que contém o estado inicial ou (caso o problema não possua solução) quando uma camada M_{π_i} não pode mais ser expandida.

Outra diferença importante é que um estado já representado em alguma camada não deve ser inserido em uma nova camada (Linha 12). Esta restrição é feita para evitar ciclos.

Uma vez que um submodelo foi computado, o algoritmo para construir uma política é o mesmo algoritmo para construir uma política a partir de um submodelo fraco (Algoritmo 7.3). A única diferença está na Linha 18. A condição para que uma ação a seja mapeada a um estado s é que ao menos um estado sucessor s' gerado a partir da operação PROGR$(\xi(s)^{BDD}, a, e_i)$ esteja na camada seguinte e todos os demais estados sucessores, considerando os demais efeitos não determinísticos de a, estejam em camadas anteriores.

Algoritmo 7.4: SÍNTESEFORTE$(\mathcal{P})$

```
1  ξ(X)^BDD ← φ // conjunto de estados que satisfazem a meta
2  M_π1 ← ⟨ξ(X)^BDD, ∅⟩
3  se (ξ(s_0) ∧ ξ(X)^BDD) ≠ ⊥ então
4      retorna M_π // estado inicial satisfaz a meta
5  i = 2
6  C_s0 ← 0
7  enquanto (ξ(X)^BDD ∧ ξ(s_0)^BDD = ⊥) e (pontofixo ≠ ⊤) e (ξ(X)^BDD ≠ ⊥) faça
8      ξ(Y)^BDD ← ⊥
9      acoes ← ∅
10     para a ∈ A faça
11         ξ(R)^BDD ← REGRFORTE(ξ(X), a)
12         se ξ(R)^BDD ≠ ⊥ e CONTEM(M_πi, ξ(R)^BDD) = FALSO então
13             ξ(Y)^BDD ← ξ(Y)^BDD ∨ ξ(R)^BDD
14             acoes.INSERE(a)
15     se (ξ(s_0) ∧ ξ(Y)^BDD) ≠ ⊥ e C_s0 = 0 então
16         C_s0 ← i // índice da primeira camada que contém o estado inicial
17     i ← i + 1
18     M_πi ← (⟨ξ(Y)^BDD, acoes⟩)
19     ξ(X)^BDD ← ξ(Y)^BDD
20 se C_s0 = 0 então
21     retorna FALHA
22 senão
23     retorna EXTRAIPOLITICA(P, M_π, C_s0)
```

7.3 Otimizações

7.3.1 Eliminação de estados espúrios

Um dos problemas das operações de regressão é o fato delas gerarem estados espúrios, *i.e*, estados que não são alcançáveis a partir do estado inicial (Alcázar *et al.*, 2013). Um estado s é considerado espúrio quando duas proposições mutuamente exclusivas são verdadeiras nele.

Nos algoritmos em que os estados são representados de maneira explícita, a eliminação de estados espúrios torna a busca no espaço de estados mais eficiente. Na busca simbólica, a eliminação destes estados não oferece garantia de que os conjuntos de estados computados

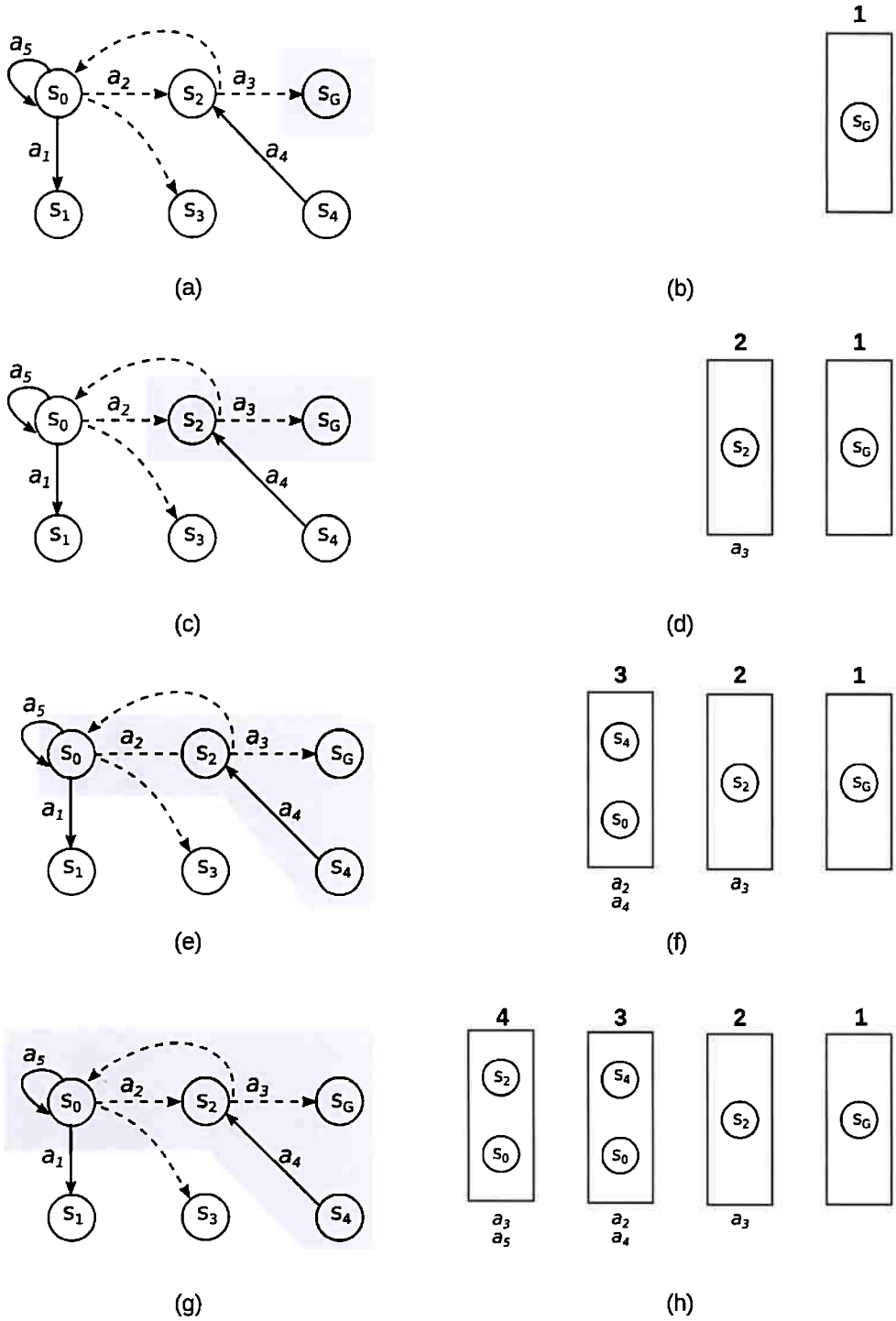


Figura 7.1: *Exemplo esquemático da construção do submodelo M_π para a fórmula $\exists \Diamond \varphi$ sendo $s_g \vDash \varphi$ e s_0 o estado inicial. As figuras da esquerda representam a construção do submodelo com pares estado-ação e as figuras da direita o construção do submodelo utilizando a estrutura de camadas.*

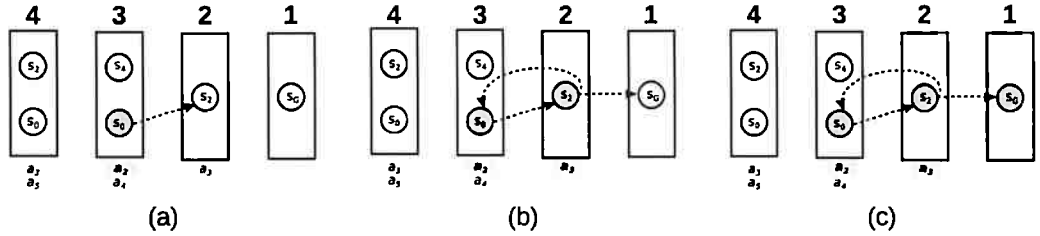


Figura 7.2: *Exemplo esquemático da extração da política fraca para o submodelo construído na Figura 7.1.*

na construção do submodelo M_π serão menores. Isso porque, em geral, o tamanho de um BDD não é proporcional ao número de estados que ele representa. No entanto, em nossos experimentos, a poda de estados espúrios melhorou o desempenho dos algoritmos.

Seja o conjunto de proposições mutuamente exclusivas $C = (p_1, p_2, \ldots, p_n)$ e M todos pares de proposições mutuamente exclusivas em C, a fórmula lógica que representa esta restrição é dada por (Torralba *et al.*, 2017):

$$\xi(c)_i^{BDD} = \left(\bigwedge_{(p_i,p_j)\in M} \neg p_i \vee \neg p_j \right) \wedge \left(\bigwedge_{(p_1,\ldots,p_j)\in C} \neg p_1 \vee \cdots \vee p_n \right) \tag{7.2}$$

A fórmula que representa todas as restrições para que um estado não seja espúrio é dada por:

$$\xi(c)^{BDD} = \xi(c)_1^{BDD} \wedge \xi(c)_2^{BDD} \wedge \cdots \wedge \xi(c)_n^{BDD} \tag{7.3}$$

Um estado abstrato s é espúrio se $\xi(s) \wedge \xi(c)^{BDD} = \bot$. Em nossos algoritmos, adicionamos uma função auxiliar, executada logo após o cálculo de regressão, que verifica se o resultado desta operação não gerou um estado espúrio. Caso o resultado da regressão $(\xi(R)^{BDD})$ não seja um estado válido ele não será representado em nenhuma camada. Em nossos experimentos, assumimos que o conjunto de proposições mutuamente exclusivas é dado para o algoritmo.

7.3.2 Heurística

Outra melhoria que pode ser implementada no planejador PACTL-SYM é o uso de heurísticas. Edelkamp e Reffel (1998) foram os primeiros a introduzir a ideia de busca simbólica com heurística. Em seu trabalho, eles apresentaram uma versão simbólica para o algoritmo A*, chamada BDDA* (P. E. Hart e Raphael, 1968). A ideia do algoritmo BDDA* é agrupar em um BDD os estados com o mesmo valor heurístico de modo que eles possam ser expandidos de uma única vez.

Em nosso algoritmo é possível utilizar heurísticas de maneira similar agrupando, em cada camada, estados com o mesmo valor heurístico. Nesse caso, uma camada não será composta por um único BDD, mas por um conjunto de BDDs que representam estados com o mesmo valor heurístico.

A cada passo de iteração, o algoritmo irá explorar o conjunto de estados com o melhor valor heurístico. Neste caso, as camadas possuem conjuntos de BDDs explorados e não explorados e, a cada iteração, uma nova camada pode ser criada ou uma camada que já existia pode ter novos conjuntos de estados para serem explorados. A Figura 7.3 apresenta um exemplo de como os conjuntos de estados são representados nas camadas quando utilizamos heurísticas.

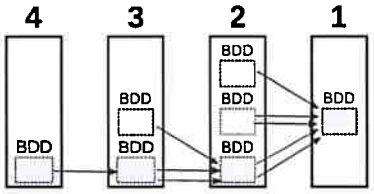


Figura 7.3: *Exemplo da representação de conjuntos de estados quando o algoritmo de síntese utiliza heurística. Os retângulos representam conjuntos de estados com o mesmo valor heurístico. Retângulos em cinza representam conjuntos de estados gerados e explorados enquanto os retângulos em branco representam conjuntos que foram gerados mas não foram explorados.*

Sempre que um estado predecessor s' é gerado, seu valor heurístico é computado. Em seguida, se existe um conjunto de estados na camada de s' que ainda não foi explorado e que possui o mesmo valor heurístico, então é feita a união deste conjunto com s'.

Em nossos experimentos, para computar o valor heurístico de um estado s' consideramos uma função que calcula quão distante s' está do estado inicial s_0.

7.4 Exemplo de execução do planejador PACTL-SYM

Para exemplificar o funcionamento do planejador PACTL-SYM vamos considerar o domínio do Robô de Carga em que um robô deve transportar n caixas de uma localização para outra. Como exemplo, vamos considerar um cenário em que temos 2 salas (sala A e sala B) e 1 caixa que deve ser transportada de uma sala B para sala A (Figura 7.4).

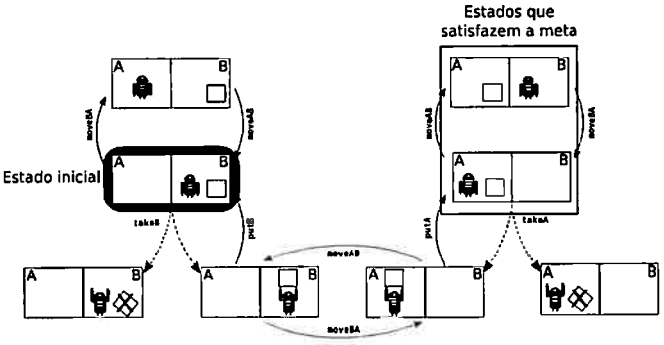


Figura 7.4: *Domínio do robô de carga*

Este problema pode ser representado pelo seguinte conjunto de proposições $\mathbb{P}$: $boxAtA$ (representa que a caixa está no chão da sala A); $boxAtB$ (a caixa está no chão da sala B); $boxOnr$ (a caixa está nos braços do robô); $boxOk$ (a caixa não está quebrada); $rAtA$ (o robô está na sala A); $rAtB$ (o robô está na sala B) e $rFree$ (o robô não está carregando nenhuma caixa). As proposições mutuamente exclusivas deste problema são: $\{rAtA, rAtB\}$ pois o robô não pode estar em duas salas ao mesmo tempo e $\{boxAtA, boxAtB, boxOnr\}$ pois, ou a caixa está no chão de uma das salas, ou nos braços do robô. O robô pode executar seis ações, sendo elas: `moveAB`, move da sala A para sala B; `moveBA`, move da sala B para sala A; `putA`, descarrega a caixa na sala A; `putB`, descarrega a caixa na sala B; `takeA`, carrega a caixa na sala A; e `takeB`, carrega a caixa na sala B. A Figura 7.5 descreve as ações `moveBA`, `putA` e `takeB` de maneira simbólica. Note que, sempre que o robô tenta carregar uma caixa ele poderá derrubá-la no chão e quebrá-la ($\neg boxOk$) e assim o robô não poderá mais transportá-la para outra localização.

Seja o estado inicial $s_0 = (boxAtB \wedge boxOk \wedge rAtB \wedge rFree \wedge \neg boxAtA \wedge \neg boxOnr \wedge \neg rAtA)$ e a meta $\phi = \exists \diamond \varphi$ onde $\varphi = (rAtA \wedge boxAtA \wedge boxOk)$, a Figura 7.6 mostra o resultado da regressão fraca a partir de φ considerando a ação `moveBA` e a Figura 7.7 apresenta o resultado das camadas construídas de acordo com o Algoritmo 7.1. Por simplicidade, neste

$moveBA:$	$\langle\ \xi(Precond(moveBA)) = rAtB;\ \xi(Efeitos_{ND}(moveBA)) = rAtA \wedge \neg rAtB$ Mudanças $= \{rAtA, rAtB\}\rangle$
$putA:$	$\langle\ \xi(Precond(putA)) = boxOnr \wedge rAtA;\ \xi(Efeitos_{ND}(putA)) = boxAtA \wedge rFree \wedge \neg boxOnr$ Mudanças $= \{boxAtA, rFree, boxOnr\}\rangle$
$takeB:$	$\langle\ \xi(Precond(takeB)) = boxAtB \wedge rAtB \wedge rFree \wedge boxOk;$ $\xi(Efeitos_{ND}(takeB)) = (boxOnr \wedge \neg boxAtB \wedge \neg rFree) \vee \neg boxOk;$ Mudanças $= \{boxOnr, boxAtB, rFree, boxOk\}\rangle$

Figura 7.5: *Ações simbólicas do domínio do robô de carga.*

exemplo, mostramos as camadas construídas apenas até alcançar a primeira camada que contém o estado inicial.

$\text{RegrFraca}(\xi(\varphi), moveBA) = \xi(Precond(moveBA)) \wedge \exists Mudanças(moveBA).(\varphi \wedge \xi(Efeitos_{ND}(moveBA)))$
$= rAtB \wedge \exists rAtA, rAtB.((boxAtA \wedge rAtA \wedge boxOk) \wedge (rAtA \wedge \neg rAtB))$
$= rAtB \wedge \exists rAtA, rAtB.(boxAtA \wedge rAtA \wedge boxOk \wedge \neg rAtB) = rAtB \wedge \exists rAtB.(boxAtA \wedge boxOk \wedge \neg rAtB)$
$= rAtB \wedge boxAtA \wedge boxOk$

Figura 7.6: *Regressão fraca a partir de* $\varphi = (rAtA \wedge boxAtA \wedge boxOk)$ *segundo a ação* $moveBA$.

Camada	Conjunto de estados $\xi(X)$	Ações
1	$(boxAtA \wedge rAtA \wedge boxOk)$	∅
2	$(boxAtA \wedge rAtB \wedge boxOk) \vee (boxOnr \wedge rAtA \wedge boxOk)$	`moveAB, putA`
3	$(boxAtA \wedge rAtA \wedge boxOk \wedge rFree) \vee (boxOnr \wedge rAtB \wedge boxOk)$	`moveAB, takeA, moveBA`
4	$(boxAtA \wedge rAtB \wedge boxOk) \vee (boxOnr \wedge rAtA \wedge boxOk) \vee$ $(boxAtB \wedge rAtB \wedge boxOk)$	`moveBA, putA, moveAB, takeB`

Figura 7.7: *Submodelo construído pelo Algoritmo 7.1 para o Robô de Carga.*

A primeira camada que s_0 satisfaz é a camada 4. Para extrair uma política, o algoritmo escolherá alguma ação ação da camada e computará a operação de progressão. A Figura 7.4 mostra o cálculo da progressão de s_0 considerando a ação `takeB`.

$Progr(s, a, e_i) = \exists Mudanças(a, e_i).(\xi(s) \wedge \xi(Precond(a))) \wedge \xi(Efeitos(a, e_i))$
$= \exists boxOnr, boxAtB, rFree.((boxAtB \wedge boxOk \wedge rAtB \wedge rFree \wedge \neg boxAtA \wedge \neg boxOnr \wedge \neg rAtA)$
$\wedge (boxAtB \wedge rAtB \wedge rFree \wedge boxOk)) \wedge (boxOnr \wedge \neg boxAtB \wedge \neg rFree)$
$= rAtB \wedge \neg boxAtA \wedge \neg rAtA \wedge boxOnr \wedge \neg boxAtB \wedge \neg rFree$

(a) *Progressão considerando o efeito não determinístico* $(boxOnr \wedge \neg boxAtB \wedge \neg rFree)$

$Progr(s, a, e_i) = \exists Mudanças(a, e_i).(\xi(s) \wedge \xi(Precond(a))) \wedge \xi(Efeitos(a, e_i))$
$= \exists boxOk.((boxAtB \wedge boxOk \wedge rAtB \wedge rFree \wedge \neg boxAtA \wedge \neg boxOnr \wedge \neg rAtA)$
$\wedge (boxAtB \wedge rAtB \wedge rFree \wedge boxOk)) \wedge (\neg boxOk)$
$= boxAtB \wedge rAtB \wedge rFree \wedge \neg boxAtA \wedge \neg boxOnr \wedge \neg rAtA \neg boxOk$

(b) *Progressão considerando o efeito não determinístico* $(\neg boxOk)$

Figura 7.8: *Operação de progressão a partir do estado* s_0 *considerando todos os efeitos não determinísticos da ação* `takeB`.

O estado gerado na Figura 7.8a está representado na camada 3. Este estado será adicionado na lista *estados_politica* e na próxima iteração o algoritmo computará a operação de progressão deste estado segundo alguma ação da camada 3. Os estados gerados por este cálculo serão adicionados na lista *estados_politica* e assim sucessivamente.

O estado gerado na Figura 7.8b não está representado em nenhuma camada (levando em consideração não apenas as camadas representadas na Figura 7.7 mas todas as camadas até um ponto fixo), logo este estado será ignorado (não será adicionado na lista *estados_politica*).

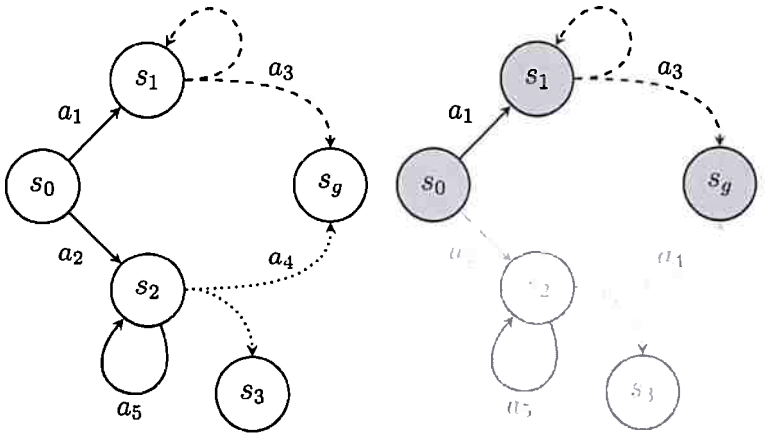
Uma vez computada a progressão a partir de s_0 (Figura 7.4), como um dos estados gerados está em na camada anterior, o par de estado-ação $(s_0, \texttt{takeB})$ será adicionado na política π.

Capítulo 8

Síntese de soluções forte-cíclicas

8.1 Síntese da política forte-cíclica do PACTL

Um plano solução para uma política forte cíclica é um plano que pode conter ações com efeitos não determinísticas que geram ciclos. Consequentemente, a execução de um plano forte cíclico pode resultar em uma sequência infinita de estados. Contundo, ao assumir que sua execução eventualmente conseguirá sair de todos os ciclos existentes é possível afirmar que uma solução forte cíclica sempre alcança a meta (Figura 8.1).



(a) *Domínio de planejamento.*

(b) *Política forte cíclica considerando s_0 como o estado inicial e s_g o estado meta.*

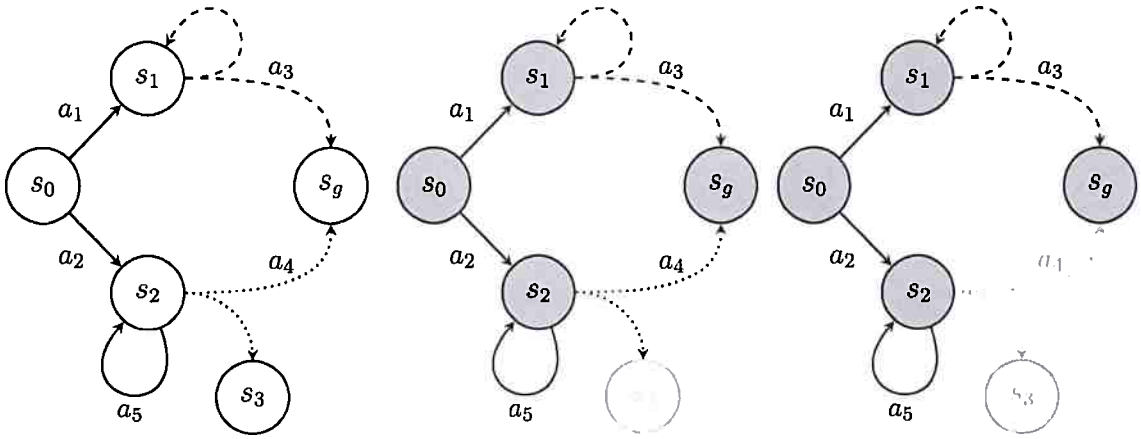
Figura 8.1: *Domínio de planejamento e política forte cíclica.*

Quando uma solução forte cíclica é desejada a meta de planejamento pode ser especificada em α-CTL pela fórmula $\forall \boxdot \exists \diamond \varphi$. Contudo, o submodelo sintetizado pelo planejador PACTL a partir desta fórmula pode resultar em um conjunto de estados que possui becos-sem-saída, isto é, estados a partir dos quais não é possível alcançar a meta.

A Figura 8.2 ilustra a computação do submodelo que satisfaz a fórmula $\forall \boxdot \exists \diamond \varphi$ sendo s_g um estado que satisfaz φ e s_0 o estado inicial. Para sintetizar uma solução forte cíclica, num primeiro momento é computado um submodelo M_1 que satisfaz $\exists \diamond \varphi$ (Figura 8.2b). Em seguida, M_1 é contraído até alcançar um ponto-fixo máximo de forma que os pares de estado-ação que não satisfazem $\forall \boxdot \exists \diamond \varphi$ sejam descartados.

Na etapa de contração o par estado-ação (s_2, a_4) é removido, uma vez que esta transição leva para um estado que não faz parte do submodelo que satisfaz $\exists \diamond \varphi$. O submodelo

sintetizado é apresentado na Figura 8.2c.



(a) *Domínio de planejamento.* (b) *Submodelo* $M_1 \models \exists \diamond \varphi$. (c) *Submodelo* $M_2 \models \forall \boxdot \exists \diamond \varphi$.

Figura 8.2: *Síntese do submodelo que satisfaz* $\forall \boxdot \exists \diamond \varphi$.

Entretanto, a contração do submodelo não exclui o par (s_2, a_3), dessa forma o estado s_2, que se tornou um beco-sem-saída com a eliminação do par (s_2, a_4) é mantido no conjunto computado. Isso ocorre porque esta transição leva para um estado que satisfaz a propriedade $\exists \diamond \varphi$ (no caso, o próprio estado s_2). Ao observarmos a árvore de computação do submodelo sintetizado a partir da fórmula $\forall \diamond \exists \boxdot \varphi$ para o domínio de planejamento apresentado (Figura 8.3) é possível perceber que para todos os caminhos dessa árvore, partindo do estado inicial s_0, é válida a propriedade $\exists \boxdot \varphi$. Ao executar a ação a_2 no estado s_0 o agente alcança o estado s_2 a partir do qual existe uma ação que leva à algum estado futuro que satisfaz a propriedade φ ($s_2 \models \exists \diamond \varphi$, pois ao executar a ação a_4 em s_2 o agente pode alcançar o estado s_g que satisfaz a meta). O submodelo sintetizado (Figura 8.2c) satisfaz a fórmula que expressa uma solução forte cíclica porém não apresenta um resultado esperado pois uma solução forte cíclica deve garantir que a meta eventualmente será alcançada.

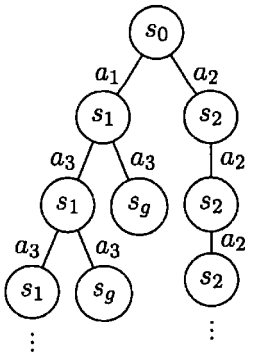


Figura 8.3: *Árvore de computação do submodelo que satisfaz* $\forall \boxdot \exists \diamond \varphi$.

8.2 Um novo algoritmo para síntese da política forte-cíclica

Conforme apresentado na seção anterior, a ideia principal para sintetizar uma política forte-cíclica pode ser resumida em duas etapas: (1) computar um submodelo M_π que satisfaz $\exists \diamond \varphi$; e (2) remover do submodelo M_π os pares estado-ação que levam para algum

estado $s \notin M_1$. A eliminação destes estados é feita através de operações de regressão forte. Contudo, após estas etapas, alguns estados que não alcançam a meta são mantidos em M_1.

Uma forma de garantir que todos os pares de estado-ação que não alcançam a meta sejam removidos é repetir as etapas básicas do algoritmo até alcançar um ponto-fixo mas sempre considerando apenas o subconjunto computado no passo anterior (Simão, 2017). A Figura 8.4 Apresenta um exemplo esquemático do funcionamento do novo algoritmo. Note que o ponto-fixo é alcançado após repetir as etapas básicas do algoritmo duas vezes.

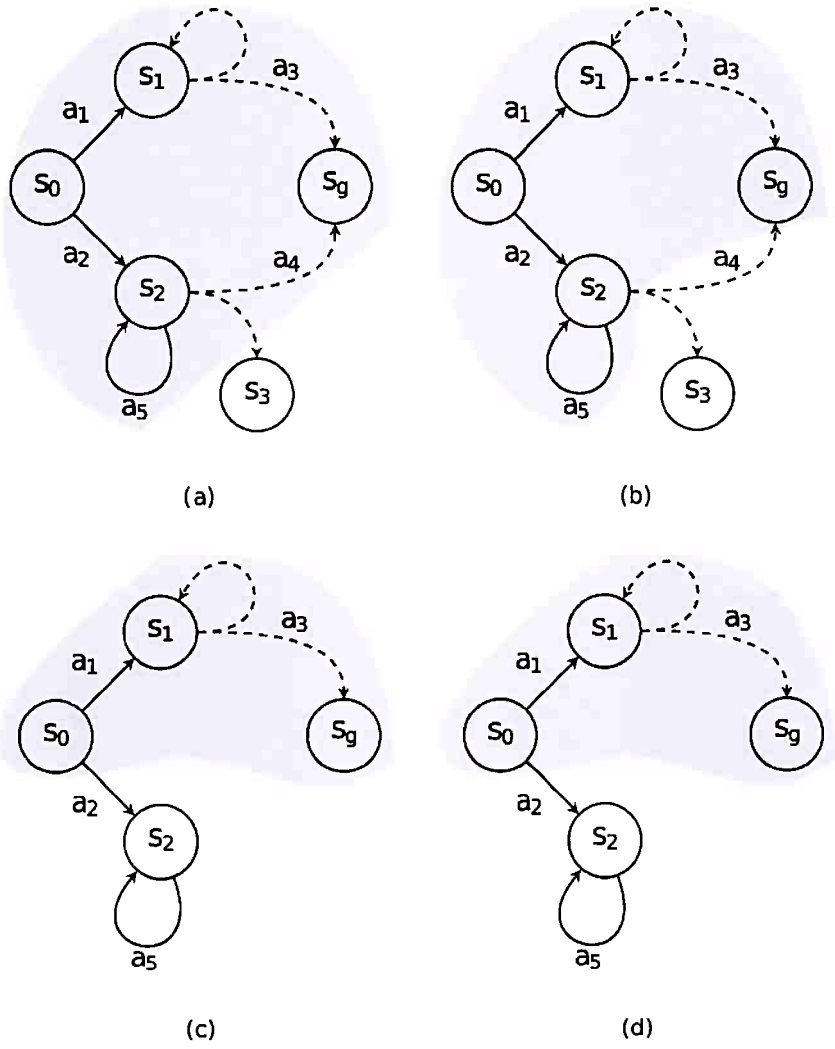


Figura 8.4: *Exemplo esquemático do algoritmo de síntese forte-cíclica. Em 8.4a. é computado um submodelo que satisfaz* $\exists \diamond \varphi$. *Em 8.4b. são podados os pares estado-ação que levam para fora do submodelo computado. Em 8.4c. é computado novamente um submodelo que satisfaz* $\exists \diamond \varphi$ *porém considerando apenas o subconjunto resultante dos passos anteriores. Em 8.4d. são podados os pares estado-ação que levam para fora do submodelo computado.*

É possível construir uma política forte cíclica utilizando uma estrutura em camadas e as operações simbólicas conforme mostra o pseudocódigo 8.1.

O algoritmo de síntese forte-cíclica, num primeiro computa um submodelo $M1_\pi$ exatamente da mesma forma que o Algoritmo 7.1. Em seguida, o algoritmo percorre as camadas do submodelo computando os estados que garantidamente alcançam a meta. Para cada camada de $M1_{\pi_i} = \langle \xi(X)^{BDD}, acoes \rangle$ o algoritmo computa a progressão considerando todas as ações $a \in acoes$ de $\xi(X)^{BDD}$ segundo todos os efeitos não determinísticos de a. Este cálculo é feito através da função auxiliar 12. Se todos os sucessores calculados pertencem

a $M1_\pi$, então $\forall \boxdot M1_\pi$ é satisfeita. Além disso, se ao menos um sucessor está na última camada de $M2_\pi$ então existe a possibilidade de alcançar a meta a partir dos gerados. Se essas duas condições são satisfeitas o algoritmo calcula a regressão fraca de $\xi(X)^{BDD}$ considerando a ação a e armazena o resultado em uma nova camada de $M2_\pi$.

O algoritmo para quando alcança um ponto fixo ($M1_\pi = M2_\pi$).

Para construir uma política a partir do submodelo computado é possível utilizar o mesmo algoritmo de extração da política apresentado no Capítulo 7.

Algoritmo 8.1: SÍNTESEFORTE-CÍLICA($\mathcal{P}$)

```
1  M1_π ← SUBMODELOFRACO(P)
2  M2_π ← ⟨φ, ∅⟩
3  C_{s0} = 0
4  i = 1
5  enquanto M1_π ≠ M2_π faça
6      M2_π ← ⟨φ, ∅⟩
7      para camada ∈ M1_π faça
8          ⟨ξ(X)^BDD, acoes⟩ ← camada
9          ξ(Y)^BDD ← ⊥
10         novas_acoes ← ∅
11         para a ∈ acoes faça
12             sucessores ← SUCESSORES(ξ(X)^BDD, a)
13             se todos os estados s' ∈ sucessores estão em alguma camada
               de M1_π e ao menos um sucessor s' está na última camada de
               M2_π então
14                 ξ(Y)^BDD ← ξ(Y)^BDD ∨ REGRFRACA(ξ(X), a)
15                 novas_acoes.INSERE(a)
16         se (ξ(s0) ∧ ξ(Y)^BDD) ≠ ⊥ e C_{s0} = 0 então
17             C_{s0} ← i // índice da primeira camada que contém o estado
               inicial
18         i ← i + 1
19         M2_{π_i} ← (ξ(Y)^BDD, novas_acoes)
20 se C_{s0} = 0 então
21     retorna FALHA
22 senão
23     retorna EXTRAIPOLITICA(P, M2_π, C_{s0})
```

Capítulo 9

Analise Empírica

Rodamos o planejador PACTL-SYM, desenvolvido neste trabalho, e comparamos os resultados obtidos com os planejadores PRP [1] (Muise *et al.*, 2012), MBP [2] (Bertoli *et al.*, 2001) e a versão não simbólica do planejador PACTL também implementada neste trabalho.

Utilizamos domínios provenientes da competição internacional de planejamento (*International Planning Competition* - IPC [3]) e alguns domínios da competição com modificações feitas por nós. Analisamos os resultados em termos de tempo para computar uma solução e o tamanho da solução encontrada.

Como os planejadores PACTL e MBP requerem que o tipo de solução a ser encontrada seja especificada, dividimos nossos experimentos em grupos de acordo com a solução desejada: fraca, forte ou forte-cíclica. As seções a seguir descrevem os domínios utilizados para encontrar cada uma das soluções bem como os resultados obtidos.

Todos os experimentos foram executados em uma máquina com sistema operacional Debian versão 9.3, com 396 GB de RAM e processador de 2.20GHz. O tempo limite para um planejador encontrar uma solução foi de 1 hora.

9.1 Domínios

9.1.1 Domínio do robô

O domínio do robô é uma variação do domínio Gripper, introduzido na primeira competição de planejamento em 1998 para planejadores clássicos. No domínio do robô existem duas salas (sala A e sala B) e um robô que possui duas garras e deve transportar uma quantidade de caixas de uma sala para outra. Nos problemas que utilizamos para teste, inicialmente todas as caixas estão na sala A e o objetivo é levá-las para a sala B.

Este domínio possui as seguintes ações:

1. **Carregar uma caixa em algum braço:** o robô pode carregar uma caixa no seu braço direito ou carregar uma caixa em seu braço esquerdo.

2. **Descarregar uma caixa:** o robô pode colocar no chão da sala que se encontra uma caixa que está carregando. Sempre que o robô tenta descarregar uma caixa ele pode derrubá-la no chão.

3. **Mover-se de uma sala para outra:** Um robô pode se movimentar livremente entre as salas através desta ação. Se o robô está carregando uma caixa e executa esta ação, a caixa se move de sala junto com ele.

[1] Disponível em bitbucket.org/haz/planner-for-relevant-policies
[2] Disponível em mbp.fbk.eu
[3] www.icaps-conference.org/index.php/Main/Competitions

Os problemas deste domínio podem ser escalados aumentando o número de caixas que devem ser transportadas de uma sala para outra. O número de estados deste problema cresce exponencialmente a medida que o número de caixas aumenta.

9.1.2 Domínio dos Primeiros Socorros

O Domínio dos Primeiros Socorros (*First-Responders Domain*), utilizado na Competição Internacional de Planejamento de 2008, consiste em resolver emergências de incêndio e socorrer vitimas. Para atender as emergências, existem equipes de bombeiros que podem apagar os incêndios e equipes de resgate que podem socorrer as vítimas.

Este domínio possui as seguintes ações:

1. **Mover caminhão de bombeiros:** esta ação faz com que um caminhão de bombeiro que está em uma localização l_1 se mova para uma localização adjacente l_2 desde que não haja incêndio em l_2.

2. **Mover ambulância:** assim como a ação anterior, esta ação move uma ambulância de uma localização l_1 para uma localização adjacente l_2 se não existe incêndio em l_2.

3. **Encher caminhão de bombeiro:** sempre que um caminhão de bombeiro está em uma localização que possui água, seu tanque de água pode ser enchido.

4. **Colocar uma vítima na ambulância:** quando uma vítima e uma ambulância estão no mesmo local, é possível colocar a vítima dentro da ambulância.

5. **Apagar um incêndio:** se um caminhão de bombeiro possui água ele pode ser usado para apagar um incêndio numa localização adjacente (ou no mesmo local) em que o caminhão de bombeiros se encontra. Ao executar esta ação, o caminhão ficará com o tanque de água vazio e existe a possibilidade do fogo não ser apagado.

6. **Tirar uma vítima da ambulância:** se uma ambulância possui uma vítima, esta pode ser tirada da ambulância e colocada no local em que a ambulância se encontra.

7. **Socorrer uma vítima por uma ambulância:** uma vítima que está ferida pode ser socorrida por uma ambulância se essa estiver presente no local. Ao ser socorrida, a vítima pode ou não ficar saudável.

8. **Socorrer uma vítima por um caminhão de bombeiro:** uma vítima que está ferida pode ser socorrida por um caminhão de bombeiro se esse estiver presente no local. Ao ser socorrida, a vítima pode ou não ficar saudável.

9. **Socorrer uma vítima no hospital:** uma vítima pode ser socorrida em um hospital. Quando isso acontece, a vítima pode sempre ficará saudável. Um vítima que está em situação de saúde crítica não pode ser tratada por um caminhão de bombeiro ou por uma ambulância, apenas no hospital.

Os problemas deste domínio podem ser escalados adicionando localizações, incêndios, vítimas, equipes médicas e equipes de bombeiros.

9.2 Soluções fracas

Para avaliar o desempenho dos planejadores ao computar uma solução fraca, utilizamos o domínio do robô e o domínio dos primeiros socorros com algumas modificações descritas a seguir:

- **Domínio do robô:** alteramos a ação 2 de forma que, sempre que o robô tenta descarregar uma caixa e ela cai no chão ela quebra. Quando isso acontece o agente não pode mais carregar uma caixa e consequentemente não consegue alcançar um estado meta.

  A Figura 9.1 apresenta uma instância do domínio com esta modificação para um problema com uma caixa.

- **Domínio dos primeiros socorros:** Eliminamos a ação 9 para evitar que o problema tenha uma solução forte; e adicionamos a ação descrita a seguir:

  - **Socorrer várias vitimas simultaneamente:** quando existe mais de uma vítima em um mesmo local e uma ambulância, as vítimas podem ser atendidas simultaneamente por de uma equipe especializada.

  A Figura 9.2 apresenta uma instância deste domínio. A Figura 9.3 apresenta os estados sucessores gerados quando as ações de socorrer vítimas são executadas em um estado com duas vítimas, uma ambulância e um caminhão de bombeiro.

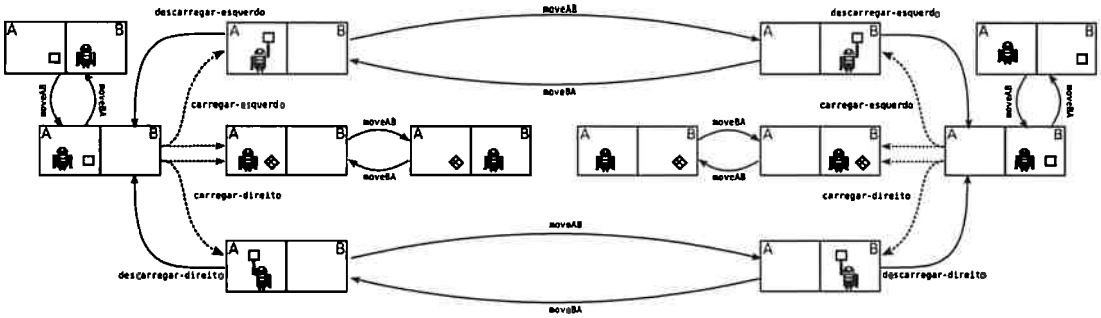


Figura 9.1: *Domínio do robô para um problema com 1 caixa. Arestas tracejadas representam ações com efeitos não determinísticos enquanto arestas cheias representa ações com efeitos determinísticos.*

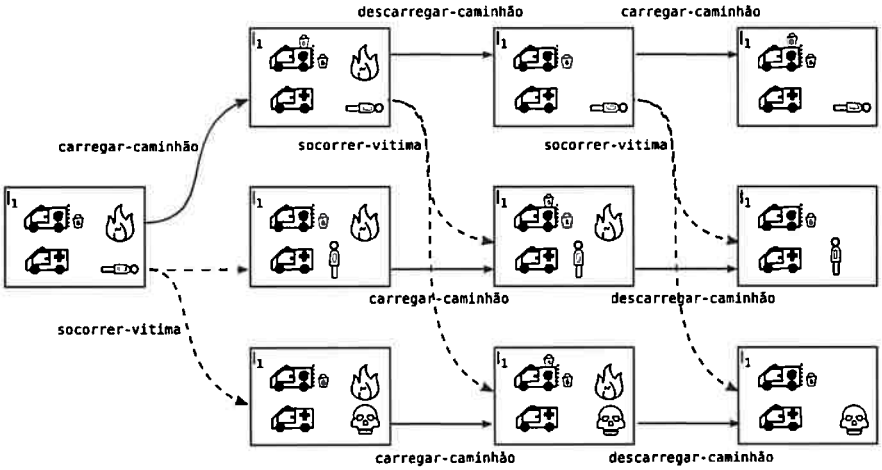


Figura 9.2: *Domínio dos primeiros socorros para um problema com uma localização, um incêndio, uma vítima, um caminhão de bombeiro e uma ambulância. Arestas tracejadas representam ações com efeitos não determinísticos enquanto arestas cheias representa ações com efeitos determinísticos. Na figura, as ações de socorrer a vítima na ambulância, socorrer a vítima com equipe de bombeiros e socorrer várias vítimas simultaneamente foram representadas através de uma única ação (socorrer a vítima) uma vez que os estados gerados por estas ações é o mesmo.*

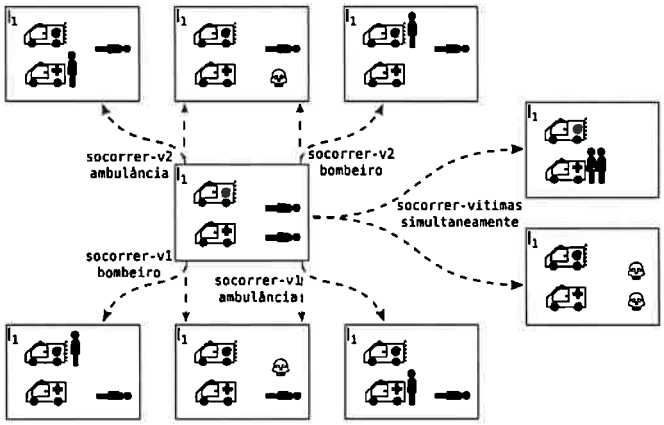


Figura 9.3: *Exemplo de estados sucessores após executar as ações de socorrer uma vítima com equipe de bombeiros, socorrer uma vítima com equipe na ambulância e socorrer várias vítimas simultaneamente.*

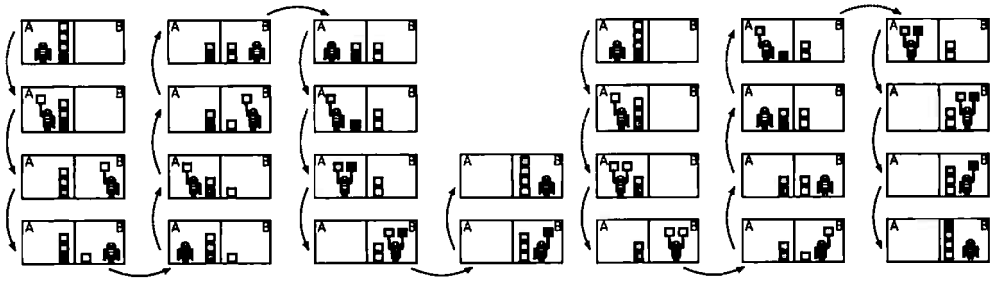
A fim de verificar o desempenho do PACTL-SYM, implementamos no planejador uma heurística específica para o domínio do robô e rodamos os planejadores com problemas de até 30 caixas. A Tabela 9.1 apresenta o tempo de execução de cada um dos planejadores para encontrar uma solução. Na tabela, a segunda coluna representa o tempo de execução do planejador PACTL-SYM, a terceira o tempo do PACTL-SYM com a heurística específica para o domínio do robô, a quarta o tempo do planejador PACTL implementado sem utilizar técnicas verificação simbólica, a sexta o tempo do planejador PRP e a última o tempo do planejador MBP.

Quantidade de Caixas	PACTL-Sym	PACTL-Sym HE	PACTL	PRP	MBP
1	0	0.008	0	0	0.008
2	0.002	0.013	0	0	0.104
3	0.007	0.021	0.137	0	1.456
4	0.019	0.033	11.72	0	84.064
6	0.104	0.086	-	0	-
7	0.198	0.131	-	0.02	-
9	0.701	0.325	-	0.02	-
10	1.142	0.48	-	0.04	-
20	128.693	67.767	-	0.18	-
30	-	-	-	0.56	-

Tabela 9.1: *Tempo (em segundos) para computar uma política para os problemas do domínio do robô.*

Neste domínio, o planejador PRP obteve o melhor desempenho em termos de tempo. Contudo, a solução encontrada por ele não foi a melhor se considerarmos o tamanho da solução. Para todos os problemas acima de 3 caixas a política devolvida por este planejador envolve mais passos para alcançar um estado meta do que o planejador PACTL-SYM, isso porque, diferentemente do PRP, o planejador PACTL-SYM garante encontrar uma política que contém o menor caminho possível para alcançar um estado que satisfaça a meta. A Figura 9.4 apresenta as políticas encontradas pelos planejadores PRP (Figura 9.4a) e PACTL-SYM (Figura 9.4b) para o problema do robô com 4 caixas. Note que, se existem caixas na sala A e o robô está nesta sala, a melhor ação, de acordo com a política encontrada pelo PRP, é carregar uma caixa com o braço direito, mover-se para sala B e descarregar a caixa. Diferentemente da política encontrada pelo PACTL-SYM, onde o robô deve carregar uma caixa com o braço direito, carregar uma segunda caixa com o braço esquerdo para

em seguida se mover para sala B e descarregar as caixas, ou seja, a política devolvida pelo planejador PACTL-SYM possui um caminho no qual sua execução contém o menor número de ações necessárias para alcança a meta. A Tabela 9.2 apresenta o número de pares estado-ação das políticas encontradas pelos planejadores PACTL-SYM e PRP.



(a) *Política encontrada pelo planejador* PRP. (b) *Política encontrada pelo planejador* PACTL-SYM.

Figura 9.4: *Políticas encontradas pelos planejadores* PRP *e* PACTL-SYM *para o problema do robô com 4 caixas. Os retângulos representam estados da política e as setas as transições de um estado para outro após a execução da ação da melhor ação possível.*

Quantidade de Caixas	PACTL-Sym	PRP
1	3	3
2	5	5
3	9	9
4	11	13
6	17	21
7	21	25
9	27	33
10	29	37
20	59	77

Tabela 9.2: *Quantidade de pares estado-ação das políticas encontradas pelos planejadores* PACTL-SYM *e* PRP *para o problema do robô.*

As políticas encontradas pelos planejadores PACTL-SYM com heurística, sem heurística e não simbólico são as mesmas para todos os problemas uma vez o algoritmo, embora implementado utilizando técnicas distintas, é essencialmente o mesmo . A diferença entre as implementações está no tempo em que cada abordagem levou para encontrar uma solução. De modo geral, quanto maior o tamanho do problema, mais eficiente em termos de tempo é o planejador PACTL-SYM com heurística, isso porque os conjuntos de estados que estão mais distantes do estado inicial (necessitam que mais ações sejam executadas a partir do estado inicial para alcançá-los) não são explorados, consequentemente o tamanho dos BDDs em cada camada é significativamente menor. A Figura 9.5 apresenta o tamanho dos BDDs gerados em cada camada do algoritmo com heurística (`PACTL-Sym HE`) e sem heurística (`PACTL-Sym HE`) para o problema com 7 caixas.

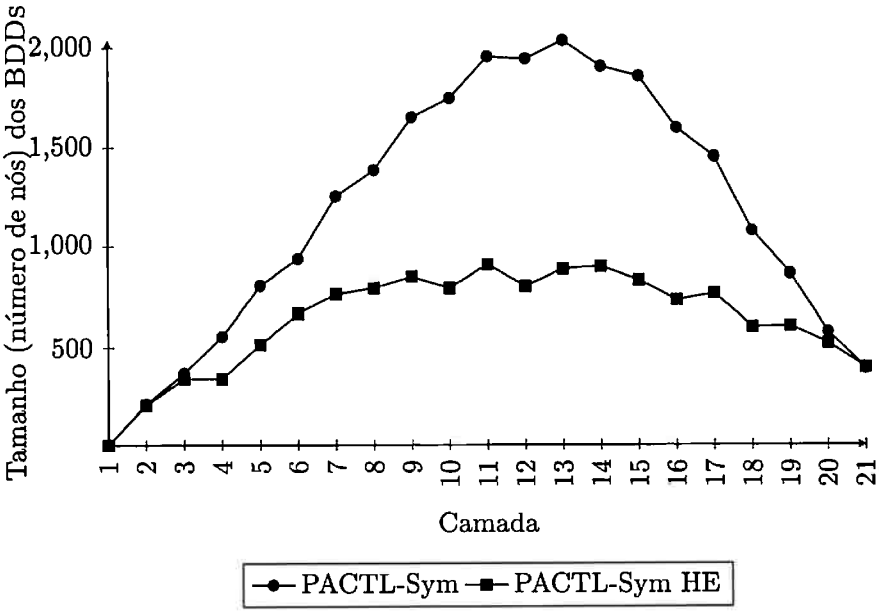


Figura 9.5: *Tamanho dos* BDDs *gerados em cada uma das camadas computadas pelo planejador* PACTL-SYM *heurístico e não heurístico.*

O planejador MBP, assim como o PACTL, também garante encontrar uma política cuja execução alcança o estado meta no menor número de passos possíveis. Contudo, diferentemente do PACTL-SYM, este planejador computa todo o espaço de estados do problema e busca uma solução no espaço de estados completo, o que faz com que este planejador seja menos eficiente que os planejadores PACTL-SYM e PRP que não geram todo o espaço de estados, mas apenas os conjuntos de pares estado-ação que podem alcançar a meta.

Em outro experimento rodamos os planejadores com diversos problemas do domínio dos primeiros socorros com as modificações, todos eles com duas localizações: l_1 e l_2. Em l_1 temos um caminhão de bombeiro e uma ambulância, já em l_2 temos um incêndio e um número variado de vítimas. A Tabela 9.3 apresenta o tempo que cada um dos planejadores levou para encontrar uma solução. Neste domínio, o planejador PACTL-SYM foi o mais eficiente em termos de tempo, além disso, o tamanho da solução encontrada por este planejador é consideravelmente menor quando comparada ao planejador PRP. As soluções devolvidas pelo planejador PACTL-SYM para os problemas deste domínio sempre consistem em socorrer todas as vítimas simultaneamente por uma equipe especializada, enquanto as soluções devolvidas pelo PRP consistem sempre em socorrer uma vítima de cada vez.

Número de Vítimas	PACTL-Sym	PACTL	PRP	MBP
2	0.007	1093.54	0.04	61.848
3	0.025	-	0.04	6566.552
4	0.121	-	0.16	-
5	1.19	-	1.64	-
6	17.979	-	58.76	-
7	371.816	-	-	-
8	-	-	-	-

Tabela 9.3: *Tempo (em segundos) para computar uma política para os problemas do domínio de primeiros socorros com duas localizações com incêndio, uma ambulância e um caminhão de bombeiros, variando o número de vítimas.*

9.3 Soluções fortes

Para avaliar o desempenho dos planejadores ao computar uma solução forte, utilizamos o domínio do robô com algumas modificações descritas a seguir:

- **Domínio do robô:** alteramos a ação 2 de forma que, sempre que o robô tenta descarregar uma caixa com o braço esquerdo ele nunca derruba ela. Quando ele tenta descarregar uma caixa com o braço direito existe a possibilidade da caixa cair no chão e quebrar. Se isso acontecer, o agente não pode mais carregar a caixa e consequentemente não consegue alcançar um estado meta.

Rodamos os planejadores considerando o domínio do robô para problemas de até 30 caixas. A Tabela 9.4 apresenta o tempo de execução de cada um dos planejadores para encontrar uma solução.

Quantidade de caixas	PACTL-Sym	PRP	MBP
1	0	0	0.004
2	0.001	12.88	0.068
3	0.003	13.24	0.788
4	0.007	12.8	14.34
5	0.015	13.54	619.652
6	0.029	14.34	-
7	0.044	14.12	-
9	0.110	15.04	-
10	0.159	15.06	-
15	0.721	16.82	-
20	5.208	17.84	-
25	44.18	20.06	-
30	-	21.68	-

Tabela 9.4: *Tempo (em segundos) para computar uma solução forte para os problemas do domínio do robô com modificações.*

Neste domínio, o planejador PACTL-SYM obteve um desempenho melhor para a maioria dos problemas. Porém, para problemas muito grandes (acima de 25 caixas) o PRP apresentou resultados melhores. Para um problema com 30 caixas, por exemplo, o planejador PACTL-SYM não conseguiu obter uma solução dentro do tempo limite. Assim como nos testes anteriores o MBP foi o planejador que obteve o pior desempenho.

A solução devolvida, isto é, os pares de estado-ação que compõem a política foi a mesma tanto para o PACTL-SYM quanto para o PRP.

Capítulo 10

Conclusões e trabalhos futuros

Um dos grandes desafios de planejamento automatizado é construir algoritmos eficientes que devolvam soluções ótimas. Nos últimos anos diversas técnicas para resolver problemas de planejamento da classe FOND (*Fully Observable Nondeterministic*) foram propostas. Dentre elas, uma técnica que se destaca é a baseada em verificação de modelos. A maioria dos planejadores dentro desta abordagem são baseados na lógica CTL. Porém, esta lógica não é adequada para formalizar algoritmos de síntese de política uma vez que sua semântica é baseada em estruturas de Kripke e portanto não considera os diferentes tipos de transições causadas pelas diferentes ações do domínio.

10.1 Contribuições deste trabalho

Neste trabalho de mestrado apresentamos um planejador para problemas FOND, baseado em verificação simbólica de modelos e na lógica α-CTL (Pereira, 2007), que permite construir algoritmos de síntese de maneira formal.

A abordagem simbólica nos permitiu representar grandes conjuntos de estados e raciocinar sobre eles de forma eficiente com o uso de diagramas de decisão binária (*Binary Decision Diagram* - BDDs).

Nossos algoritmos para síntese de políticas fraca e forte foram baseados no planejador PACTL. Para computador uma política forte-cíclica, apresentamos um novo algoritmo capaz de sintetizar soluções forte-cíclicas corretamente.

Implementamos os algoritmos propostos neste trabalho e fizemos uma análise comparativa do seu desempenho com outros planejadores para problemas FOND. De maneira geral, nosso planejador se mostrou promissor uma vez que os resultados obtidos ficaram próximos aos resultados do planejador PRP (Muise *et al.*, 2012), considerado estado-da-arte.

10.2 Trabalhos futuros

Alguns trabalhos futuros interessantes envolvem:

- Implementar heurísticas simbólicas independentes do domínio no planejador PACTL-SYM;
- Estender os algoritmos simbólicos para tratar todos os operadores da lógica α-CTL;
- Implementar e verificar o desempenho da busca bidirecional na construção das camadas do submodelo computado nos algoritmos de síntese da política;
- Estudar como as operações de regressão podem ser executadas de formas mais eficiente com base nas ideias apresentadas no trabalho *Efficient symbolic search for cost-optimal planning* (Torralba *et al.*, 2017);

- Fazer uma análise comparativa do desempenho do PACTL-SYM considerando problemas de planejamento clássico e planejadores para problemas clássicos vencedores da competição internacional de planejamento.

Bibliografia

Akers(1978) Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, 27(6):509–516. Citado na pág. 45

Alcázar *et al.*(**2013**) Vidal Alcázar, Daniel Borrajo, Susana Fernández e Raquel Fuentetaja. Revisiting regression in planning. Em *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, páginas 2254–2260. AAAI Press. ISBN 978-1-57735-633-2. Citado na pág. 57

Bertoli *et al.*(**2001**) Piergiorgio Bertoli, Alessandro Cimatti, Marco Pistore, Marco Roveri e Paolo Traverso. MBP: a Model Based Planner, 2001. Citado na pág. 6, 41, 67

Blum e Furst(1995) Avrim L. Blum e Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):1636–1642. Citado na pág. 1

Bonet e Geffner(1999) Blai Bonet e Hector Geffner. Planning as heuristic search: New results. Em *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, páginas 360–372. Citado na pág. 1

Bonet e Geffner(2001) Blai Bonet e Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33. Citado na pág. 2

Bryant(1986) Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691. Citado na pág. xiv, 45, 46, 47, 48

Bryant(1992) Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318. Citado na pág. 3

Büning e Bubeck(2009) Hans Kleine Büning e Uwe Bubeck. Theory of quantified boolean formulas. Em *Handbook of Satisfiability*, páginas 735–760. IOS Press. Citado na pág. 37

Burch *et al.*(**1992**) J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill e L. J. Hwang. Symbolic model checking: 1020 states and beyond. *Inf. Comput.*, 98(2):142–170. Citado na pág. 6

Butler *et al.*(**1991**) Kenneth M Butler, Don E Ross, Rohit Kapur e M Mercer. Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. Em *Proceedings of the 28th ACM/IEEE Design Automation Conference*, páginas 417–420. ACM. Citado na pág. 47

Cenamor *et al.*(**2014**) Isabel Cenamor, Tomas de la Rosa e Fernando Fernandez. Ibacop and ibacop2 planner, 2014. Citado na pág. 3

Cimatti *et al.*(**2003**) A. Cimatti, M. Pistore, M. Roveri e P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84. Citado na pág. 4, 41

Cimatti *et al.*(**1997**) Alessandro Cimatti, Enrico Giunchiglia, Fausto Giunchiglia e Paolo Traverso. Planning via model checking: A decision procedure for ar. Em *Proceedings of ECP-97*, páginas 130–142. Springer-Verlag. Citado na pág. 1

Cimatti *et al.*(**1998**) Alessandro Cimatti, Marco Roveri e Paolo Traverso. Strong planning in non-deterministic domains via model checking. Em *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998*, páginas 36–43. Citado na pág. 26

Clarke e Emerson(1982) Edmund M. Clarke e E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. Em *Logic of Programs, Workshop*, páginas 52–71. Springer-Verlag. Citado na pág. 7, 20

Clarke *et al.*(**1999**) Edmund M. Clarke, Orna Grumberg e Doron Peled. *Model checking*. MIT Press. Citado na pág. 19

Daniele *et al.*(**2000**) Marco Daniele, Paolo Traverso e Moshe Y. Vardi. Strong cyclic planning revisited. Em Susanne Biundo e Maria Fox, editors, *Recent Advances in AI Planning*, páginas 35–48, Berlin, Heidelberg. Springer Berlin Heidelberg. Citado na pág. 26, 27

do Lago Pereira e de Barros(2008) Silvio do Lago Pereira e Leliane Nunes de Barros. Using α-ctl to specify complex planning goals. Em Wilfrid Hodges e Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, páginas 260–271, Berlin, Heidelberg. Springer Berlin Heidelberg. Citado na pág. 29

Edelkamp e Reffel(1998) Stefan Edelkamp e Frank Reffel. Obdds in heuristic search. Em Otthein Herzog e Andreas Günter, editors, *KI-98: Advances in Artificial Intelligence*, páginas 81–92, Berlin, Heidelberg. Springer Berlin Heidelberg. Citado na pág. 59

Edelkamp *et al.*(**2015**) Stefan Edelkamp, Peter Kissmann e Alvaro Torralba. Bdds strike back (in ai planning), 2015. Citado na pág. 3

Fikes e Nilsson(1971) Richard E. Fikes e Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Em *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, páginas 608–620. Citado na pág. 3, 11

Fourman(2000) Michael Fourman. Propositional planning. Em *In Proc. Workshop on Model Theoretic Approaches to Planning. AIPS*, páginas 10–17. Citado na pág. 39, 40

Fu *et al.*(**2011**) Jicheng Fu, Vincent Ng, Farokh B Bastani, I-Ling Yen *et al.* Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. Em *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, página 1949. Citado na pág. 6

Giunchiglia e Traverso(2000) Fausto Giunchiglia e Paolo Traverso. Planning as model checking. Em *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*, ECP '99, páginas 1–20. Springer-Verlag. Citado na pág. 22, 25, 26, 41

Helmert(2006) Malte Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246. ISSN 1076-9757. Citado na pág. 3

Hoffmann(2001) Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22:57–62. Citado na pág. 3

Huth e Ryan(2004) Michael Huth e Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press. Citado na pág. 20, 45

Lee(1959) Chang-Yeong Lee. Representation of switching circuits by binary-decision programs. *Bell Labs Technical Journal*, 38(4):985–999. Citado na pág. 45

McMillan(1992) Kenneth Lauchlin McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Tese de Doutorado. UMI Order No. GAX92-24209. Citado na pág. 6

Menezes(2014) M. V. Menezes. *Mudanças em Problemas de Planejamento sem solução.* Tese de Doutorado, University of São Paulo. Citado na pág. xiii, 16, 17, 30, 42

Menezes *et al.***(2014)** M. V. Menezes, L. N. Barro e S. L. Pereira. Symbolic regression for non-deterministic actions. *Learning and Nonlinear Models*, 12:98–114. Citado na pág. 15

Muise(2014) Christian Muise. *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution.* Tese de Doutorado, University of Toronto. Citado na pág. 6

Muise *et al.***(2012)** Christian J Muise, Sheila A McIlraith e J Christopher Beck. Improved non-deterministic planning by exploiting state relevance. Em *ICAPS*. Citado na pág. 6, 67, 75

Müller-olm *et al.***(1999)** Markus Müller-olm, David Schmidt e Bernhard Steffen. Model-checking: A tutorial introduction. Em *In Proc. SAS*. Citado na pág. 19

Nau *et al.***(2004)** Dana Nau, Malik Ghallab e Paolo Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. Citado na pág. 1, 3, 6, 25

Newell *et al.***(1959)** Allen Newell, John C. Shaw e Herbert A. Simon. Report on a general problem-solving program. Em *Proceedings of the International Conference on Information Processing*, páginas 256–264. Citado na pág. 1

P. E. Hart e Raphael(1968) N. J. Nilsson P. E. Hart e B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107. Citado na pág. 1, 59

Pereira(2007) Silvio L. Pereira. *Planejamento sob incerteza para metas de alcançabilidade estendidas.* Tese de Doutorado, University of São Paulo. Citado na pág. xiii, 6, 7, 22, 28, 29, 30, 31, 32, 33, 34, 35, 75

Pereira e de Barros(2008) Silvio Lago Pereira e Leliane Nunes de Barros. A logic-based agent that plans for extended reachability goals. *Autonomous Agents and Multi-Agent Systems*, 16(3):327–344. Citado na pág. 7, 41

Pnueli(1977) Amir Pnueli. The temporal logic of programs. Em *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, páginas 46–57. IEEE Computer Society. Citado na pág. 7

Puterman(1994) Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edição. Citado na pág. 5

Richter *et al.***(2011)** Silvia Richter, Matthias Westphal e Malte Helmert. Lama 2008 and 2011, 2011. Citado na pág. 3

Rintanen(2008) Jussi Rintanen. Regression for classical and nondeterministic planning. Em *European Conference on Artificial Intelligence*, páginas 568–572. Citado na pág. 43, 54, 57

Russell e Norvig(2009) Stuart Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, 3rd edição. Citado na pág. 1, 2, 6, 11, 15

Sacerdoti(1975) Earl D. Sacerdoti. The nonlinear nature of plans. Relatório técnico, Stanford Research Institute. Citado na pág. 1

Shannon(1938) Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723. Citado na pág. 48

Sieling(2002) Detlef Sieling. The nonapproximability of obdd minimization. *Information and Computation*, 172(2):103–138. Citado na pág. 47

Simão(2017) Thiago D. Simão. *Planejamento probabilístico com becos sem saída.* Tese de Doutorado, University of São Paulo. Citado na pág. 56, 65

Tarski(1955) Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309. Citado na pág. 21

Torralba *et al.***(2017)** Álvaro Torralba, Vidal Alcázar, Peter Kissmann e Stefan Edelkamp. Efficient symbolic search for cost-optimal planning. *Artif. Intell.*, 242(C):52–79. ISSN 0004-3702. Citado na pág. 59, 75

Zolda(2005) Michael Zolda. *Comparing Different Prenexing Strategies for Quantified Boolean Formulas.* Tese de Doutorado, Technische Universitat Wien. Citado na pág. 38

Álvaro Torralba *et al.***(2014)** Álvaro Torralba, Vidal Alcázar e Daniel Borrajo. Symba*: A symbolic bidirectional a* planner, 2014. Citado na pág. 3