Primeira parte
segunda parte

Nome do Autor Blah Bleh Blih

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

Área de Concentração: Ciência da Computação

Orientadora: Profª Drª Nome da orientadora

São Paulo, Abril de 2016

Primeira parte
segunda parte

Este exemplar corresponde à redação
da dissertação a ser defendida por
Autor Blah Bleh Blih.

Banca Examinadora:

- Prof$^{\underline{a}}$ Dr$^{\underline{a}}$ Nome da orientadora – IME-USP.
- Prof. Dr. Nome 222 222 – USP Leste.
- Prof. Dr. Nome 333 333 – FEI.

Dedico esta dissertação de mestrado aos meus

...

...

....

Agradecimentos

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Resumo

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Palavras-chave: planejamento em Inteligência Artificial, reparo de plano, replanejamento.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit

esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: Artificial Intelligence planning, plan repair, replanning.

Sumário

Lista de Abreviaturas

ADL *Action Description Language*

AIPS *International Artificial Intelligence Planning Systems*

BDD *Binary Decision Diagram*

CGP *Conformant Graphplan*

CSP *Constraint Satisfaction Problems*

CWA *Closed World Assumption*

GPG *Greedy Planning Graph*

GPS *General Problem Solver*

HTN *Hierarchical Task Network*

IPEM *Integrated Planning, Execution, and Monitoring*

MDP *Markov Decision Process*

PDDL *Problem Domain Definition Language*

POCL *Partial Order Causal Link*

POMDP *Partially Observable Markov Decision Process*

POP *Partial Order Planner*

POPR *Partial Order Plan Repair*

PRM *Probabilistic Roadmap Method*

SIPE *System for Iteractive Planning and Execution Monitoring*

STN *Simple Temporal Netwaorking*

STRIPS *Stanford Research Institute Planning System*

UCPOP *Partial Order Planner whose step descriptions include Conditional effects and Universal quantification*

VHPOP *Versatile Heuristic Partial Order Planner*

Lista de Símbolos

Σ	Sistema de transição de estados
Σ'	Sistema de transição de estados restrito (estático e determinístico)
Π	Problema de planejamento
ϵ	Evento nulo
η	Função de observação
γ	Função de transição de estados
$\gamma(s, a)$	Progressão, conjunto de estados resultantes da aplicação de a a s
$\gamma^{-1}(s, a)$	Regressão, conjunto de estados que levam a s com a aplicação de a
π	Plano
2^S	Conjunto potência de S
$\mathcal{A}$	Conjunto de todas as ações possíveis
$\mathcal{C}$	Conjunto de símbolos de constantes
$\mathcal{D}$	Domínio de planejamento, conjunto de operadores
$\mathcal{E}$	Conjunto de eventos
$\mathcal{H}$	Histórico de refinamentos
$\mathcal{O}$	Conjunto de todas as observações possíveis
$\mathcal{P}$	Plano parcial
$\mathcal{R}$	Estratégia de refinamento
$\mathcal{S}$	Conjunto de todos os estados possíveis
E_p	Estrutura do plano
G	Conjunto de estados meta
O	Conjunto de observações, sub-conjunto de $\mathcal{S}$
P	Problema de planejamento
S	Conjunto de estados, sub-conjunto de $\mathcal{S}$

S_0	Conjunto de estados iniciais
S_g	Conjunto de estados objetivos
no-op	Ação nula
a	Ação de $\mathcal{A}$
e	Evento de $\mathcal{E}$
o	Observação de $\mathcal{O}$
s	Estado de $\mathcal{S}$
s_0	Estado inicial

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Planejamento é um componente importante do comportamento racional pois trata-se de um processo de síntese que seleciona e organiza ações antecipando seus efeitos. Este processo busca satisfazer, da melhor forma possível, um conjunto de metas pré-definidas.

Muitas tarefas humanas necessitam de planejamento. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. [van der Krogt and de Weerdt, 2004a]. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. O Exemplo 1.0.1 ilustra um problema real de planejamento.

Exemplo 1.0.1. *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*

- *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*
- *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eius-*

mod tempor incididunt ut labore et dolore magna aliqua.

- *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*

Um plano simplificado para o problema do Exemplo 1.0.1, Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Exemplo 1.0.2. *Um exemplo de falha, Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*

Quando o plano falha há, basicamente, duas soluções: identificar a situação como um novo problema e criar um novo plano para a situação (*replanejamento*), ou tentar reparar o plano existente (*reparo de plano*). Um argumento a favor do reparo de planos [van der Krogt, 2005] é considerar a quantidade de trabalho já executada e pensar na quantidade de trabalho que será exigida para se criar um novo plano: lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. No entanto, a tarefa de reparo, no pior caso, pode consumir mais trabalho do que um replanejamento completo [Nebel and Koehler, 1993] .

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. e simulações.

Lorem ipsum dolor sit amet

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo

enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, [Ghallab et al., 2004]. At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, [Fikes and Nilsson, 1971, Mason, 1993], [Nau et al., 1995] xyz, abc e def ghi [Smith et al., 1999] [Rabideau et al., 1999], entre outros.

Sed ut perspiciatis unde omniso

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat.. Entre elas, eabcdef ghijt para reduzir o custo da

busca [Bonet and Geffner, 2001, Hoffmann and Nebel, 2001], Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et [Nau et al., 1999] abcdefg.

1.1 Motivação

A motivação desta dissertação surgiu da possibilidade de aplicação da técnica de Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

1.2 Objetivos

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

1.3 Organização

No Capítulo 2 é apresentado um modelo conceitual . O Capítulo 3 trata de planejamento clássico Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur

aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. No Capítulo 4, é discutido a situação em que um Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. O Capítulo 5 Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Finalmente, o Capítulo 7 encerra esta dissertação com as conclusões, contribuições e recomendações para trabalhos futuros.

Capítulo 2

Modelo para planejamento

At vero eos et accusamus et iusto odio dignissimos ducimus, qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat..

2.1 Modelo conceitual para planejamento

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

Definição 2.1 (Sistema de transição de estados)**.** *Um sistema de transição de estados é uma 4-tupla* $\Sigma = (\mathcal{S}, \mathcal{A}, \mathcal{E}, \gamma)$ *[Ghallab et al., 2004]:*

- $\mathcal{S} = \{s_1, s_2, \ldots\}$ *é um conjunto finito de estados, recursivamente numerável;*
- $\mathcal{A} = \{a_1, a_2, \ldots\}$ *é um conjunto finito de ações, recursivamente numerável;*
- $\mathcal{E} = \{e_1, e_2, \ldots\}$ *é um conjunto finito de eventos, recursivamente numerável, que aqui serão chamados de ações exógenas (ou eventos exógenos);*
- $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \rightarrow 2^s$ *é uma função de transição de estado.*

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

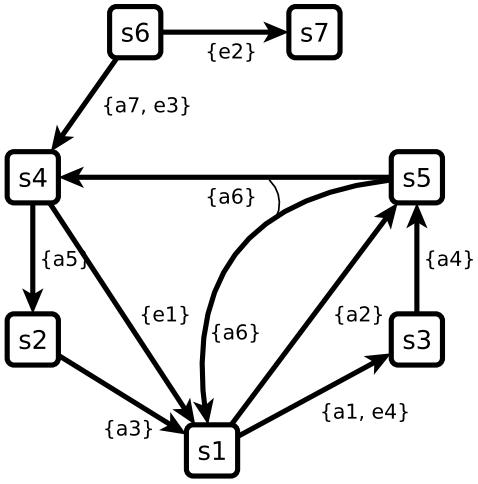

Figura 2.1: Exemplo de sistema de transição de estados.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident.

Eventos são transições *contingentes* , aqui chamados de eventos exógenos (ou ações exógenas), isto é, ações que não são controladas pelo executor do plano. Estas ações devem ser levadas em conta pelo agente de planejamento, mas não estão sob seu controle. Se e é um evento e $\gamma(s, e)$ não é

vazio, então e possivelmente ocorrerá quando o sistema estiver no estado s; e sua ocorrência em s levará o sistema a algum estado em $\gamma(s, e)$. Como o estado exato não pode ser previsível pelo agente planejador, dizemos que este é um tipo de planejamento sob incerteza.

Podem existir vários modelos que definem a semântica da função de transição $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \rightarrow 2^s$. O modelo de Markov supõe que nenhuma ação possa ser executada em estados em que ocorram eventos e vice-versa. Deste modo, $\mathcal{S}$ é dividido em *estados de ação* e *estados de evento* [Koenig et al., 1995]. Um modelo alternativo supõe que ações possam competir com eventos no mesmo estado. Isto é, se for aplicada a ação a ao estado s e $\gamma(s, e)$ não é vazio, então o próximo estado pode ser qualquer elemento de $\gamma(s, a, e)$.

Dado um sistema de transição de estado Σ, o propósito do planejamento é selecionar uma seqüência de ações que, quando executadas no estado inicial, permitam alcançar um objetivo. Chama-se de *plano* a estrutura que representa a seqüência de ações. O objetivo pode ser especificado de várias formas diferentes:

- As especificações mais simples consistem em um *estado-objetivo* s_g ou um conjunto de estados objetivo S_g. Neste caso, o objetivo é alcançado quando qualquer seqüência de transições de estado termina em um dos estados-objetivo.

- O objetivo pode satisfazer alguma condição de acordo com a seqüência de estados a ser seguida pelo sistema. Por exemplo, exigir que estados sejam evitados, especificar estados em que o sistema deverá passar necessariamente durante a execução do plano e estados nos quais ele deverá permanecer.

- Em alguns domínios é possível definir uma função de utilidade ligada aos estados, com penalidades e recompensas. Nesse caso o objetivo é otimizar alguns componentes da função destas utilidades (por exemplo, soma ou máximo) conforme a seqüência de estados seguidos pelo sistema.

- Outra alternativa é especificar o objetivo como tarefas que o sistema deverá executar. Estas tarefas podem ser recursivamente decompostas como conjuntos de ações mais simples.

A Figura 2.2 mostra um modelo conceitual do uso de um planejador em um ambiente. Este modelo é descrito por meio da interação entre três componentes:

1. O *ambiente* denominado Σ possui uma dinâmica que pode ser modelada por meio da função de transição de estado γ, de acordo com a ocorrência dos eventos e das ações.

2. Um *controlador*, dado como entrada a observação do estado em que se encontra o sistema e um plano, fornece como saída uma ação a.

3. Um *planejador* que, dado como entrada um modelo do ambiente Σ[1] (domínio), um estado inicial e um objetivo, sintetiza um plano que satisfaz o objetivo.

2.1.1 Modelo conceitual para planejamento dinâmico

Um elemento importante deste modelo é a observação do controlador sobre o estado atual do sistema durante a execução de um plano. Em geral, esta informação não é completa. O conhecimento parcial sobre o estado pode ser modelado como uma função de observação $\eta : \mathcal{S} \rightarrow \mathcal{O}$, que mapeia $\mathcal{S}$ em um conjunto discreto $\mathcal{O} = \{o_1, o_2, \ldots\}$ de observações possíveis.

Neste modelo, o controlador executa suas tarefas com a dinâmica do sistema de transição de estado e, portanto, diz-se que ele funciona *online* com o ambiente. Por outro lado, o planejador não está diretamente conectado ao ambiente (funcionamento *offline*).

[1]No decorrer do texto o modelo do ambiente Σ será tratado apenas por ambiente Σ ou, algumas vezes, por sistema Σ (de transição de estado).

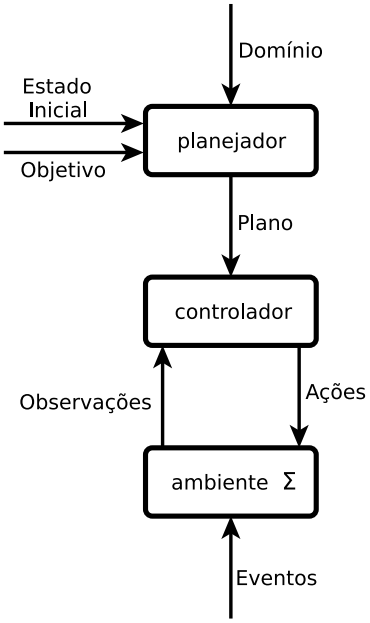

Figura 2.2: Modelo conceitual para planejamento [Ghallab et al., 2004].

Na maior parte das vezes, há diferenças entre o sistema físico (Σ) a ser controlado e seu modelo. Em geral, o planejamento depende das restrições impostas pelo modelo de Σ. Supõe-se que o controlador seja robusto o suficiente para lidar com as diferenças entre Σ e o mundo real. Lidar com observações exige mecanismos de controle mais complexos do que apenas identificar o estado e aplicar a ação correspondente. Um modelo conceitual mais realista intercala planejamento com execução e mecanismos de replanejamento ou reparo de plano. Neste caso é necessária uma ligação mais próxima entre o planejador e o controlador: o controlador deve ser capaz de devolver ao planejador o status da execução do planejamento a fim de comunicar eventuais falhas no plano, como é ilustrado na Figura 2.3 por meio da ligação entre controlador e planejador, rotulada por *status de execução*.

Neste trabalho será implementado um sistema que ora funcionará como um sistema de planejamento, quando ainda não existir um plano, ora como um

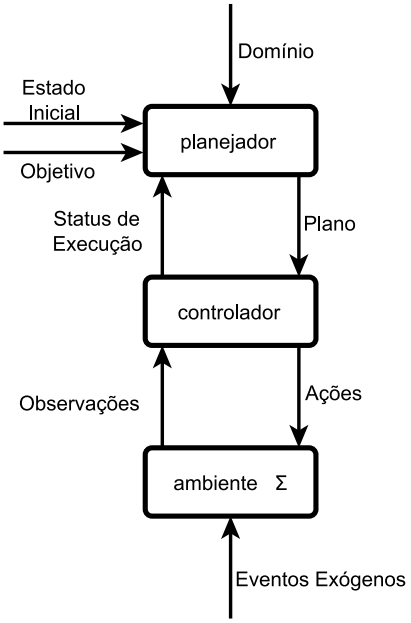

Figura 2.3: Modelo conceitual para planejamento com execução [Ghallab et al., 2004].

sistema de reparo, que dependerá da informação transmitida pelo controlador (status de execução). Esta informação poderá ser de dois tipos:

1. O plano foi executado inteiramente com sucesso.
2. O plano foi parcialmente executado e, nesse caso, o mesmo é devolvido com as indicações das ações que já foram realizadas e da ação cuja execução falhou.

No caso de falha (caso 2), o sistema de reparo é então chamado para: (1) adicionar ações ou (2) remover ações do plano a fim de torná-lo novamente executável, garantindo que este atinja as metas originais com pequenas alterações.

Capítulo 3

Planejamento clássico

Neste capítulo define-se o problema de planejamento clássico como um modelo simplificado para a tarefa de planejamento, que serve como base para futuras extensões de algoritmos para problemas mais gerais de planejamento. Apresenta-se, também, um algoritmo bastante conhecido para planejamento clássico que emprega o método de busca por refinamento de planos.

3.1 Modelo restrito

O modelo conceitual descrito no Capítulo 2 não foi proposto como um modelo operacional. Pelo contrário, ele é usado como referência para a construção de sistemas de planejamento. Por meio dele é possível fazer diferentes suposições restritivas sobre o ambiente em que se deseja planejar [Ghallab et al., 2004]:

- **Suposição A0 (Σ finito)**. O ambiente Σ tem um conjunto finito de estados.
- **Suposição A1 (Σ completamente observável)**. O sistema Σ é completamente observável. Neste caso, a função observação η é a função identidade.

- **Suposição A2 (Σ determinístico)**. O sistema Σ é determinístico, isto é, para cada estado s e para cada evento ou ação u, $|\gamma(s,u)| \leq 1$. Se uma ação é aplicável a um estado, sua execução leva um sistema determinístico a um outro estado único, possivelmente com a ocorrência de um evento exógeno.

- **Suposição A3 (Σ estático)**. O sistema Σ é estático, ou seja, o conjunto de eventos E é vazio. Σ permanece no mesmo estado até que o controlador aplique alguma ação selecionada pelo planejador.

- **Suposição A4 (objetivos restritos)**. O planejador manipula apenas objetivos restritos que são especificados como um estado objetivo explícito s_g ou um conjunto de estados objetivos S_g; sendo que o objetivo é obter qualquer seqüência de transições de estado que termine em um dos estados objetivos.

- **Suposição A5 (planejamento seqüencial)**. Um plano solução para um problema de planejamento é uma seqüência finita de ações de ordem total ou parcial.

- **Suposição A6 (tempo implícito)**. Ações e eventos não têm duração; são transições de estado instantâneas. Esta suposição está intrínseca a um sistema de transição de estado, que não representa tempo explicitamente.

- **Suposição A7 (planejamento offline)**. O planejador não considera qualquer mudança que possa ocorrer em Σ *enquanto* estiver planejando; planeja-se para o estado inicial e o objetivo, independentemente das possíveis falhas na execução do plano.

Uma vez que o sistema é determinístico, se γ for aplicável em s, então $\gamma(s,a)$ corresponde a um único estado s'. Para simplificar a notação, descreve-se $\gamma(s,a) = s'$ em vez de $\gamma(s,a) = \{s'\}$. Para este tipo de sistema, um planejamento é uma seqüência $\{a_1, a_2, \ldots, a_k\}$, tal que $\gamma(\gamma(\ldots \gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k)$ é um estado objetivo. Um exemplo de um grafo de transições de estados em um *modelo restrito*, pode ser observado na Figura 3.1 [do Lago Pereira, 2002].

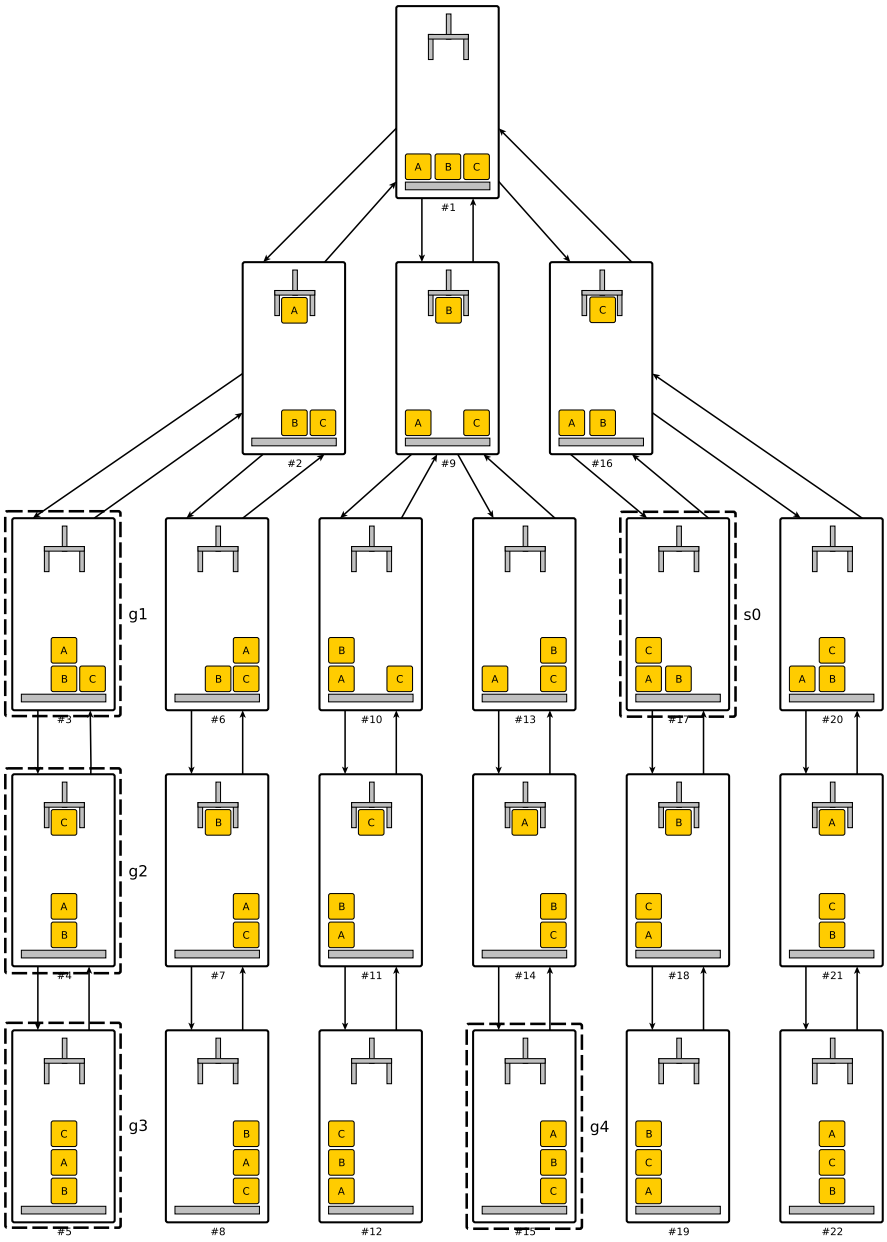

Figura 3.1: Exemplo de sistema de transição de estados no domínio do Mundo dos Blocos. Nesta figura, os nós representam todos os estados do mundo para um sistema com três blocos e cada aresta representa uma transição de estado. Um exemplo de problema de planejamento seria encontrar um plano que a partir do estado s_0 conseguisse alcançar algum dos estados objetivos $\{g_1,\ g_2,\ g_3,\ g_4\}$ [do Lago Pereira, 2002].

A propriedade sobre conhecimento completo é necessária somente no estado inicial s_0 porque o modelo determinístico permite que todos os outros esta-

dos sejam completamente previsíveis, dado que as ações são determinísticas. Uma vez que a execução do plano é incondicional (sempre funciona), o controlador que executa o plano não obtém nenhuma realimentação sobre o estado do sistema, como pode ser observado no modelo conceitual para planejamento visto no Capítulo 2.

Este caso restrito pode parecer simples: planejamento se resume a buscar um caminho em um grafo, sendo que este é um problema conhecido e bem resolvido. Na verdade, se for dado explicitamente o grafo Σ, então não há muito mais a dizer sobre planejamento para este caso. Entretanto, mesmo em um domínio simples, o grafo Σ pode ser tão extenso que especificá-lo explicitamente não é viável. Além disso, este modelo foi proposto como uma base para futuras extensões, como será visto no próximo capítulo.

O *planejamento clássico* se refere genericamente a um planejamento para sistemas restritos de transições de estados.

Definição 3.1 (Planejamento clássico)**.** *Um sistema restrito de transição de estados é aquele que satisfaz todas as suposições de A0 a A7. É um sistema de transição de estados determinístico, estático, finito e completamente observável com objetivos restritos e tempo implícito. Tal sistema é simbolizado por* $\Sigma = (\mathcal{S}, \mathcal{A}, \gamma)$*, e não por* $(\mathcal{S},\ \mathcal{A},\ \mathcal{E},\ \gamma)$*, porque não há eventos exógenos. Aqui,* $\mathcal{S}$*,* $\mathcal{A}$ *e* γ *são finitos, e* $\gamma(s, a)$ *é um estado único quando* a *é aplicável em* s *[Ghallab et al., 2004].*

Definição 3.2 (Problema de planejamento clássico)**.** *Um problema de planejamento clássico para um sistema de transição de estado restrito* $\Sigma = (\mathcal{S},\ \mathcal{A},\ \gamma)$ *é definido como uma tripla* $\mathcal{P} = (\Sigma\ ,\ s_0,\ g)$*, em que* s_0 *é um estado inicial e* g *corresponde a um conjunto de estados objetivos. Uma solução para* $\mathcal{P}$ *é uma seqüência de ações* $(a_1,\ a_2,\ \ldots,\ a_k)$ *correspondente a uma seqüência de transições de estados* $(s_0,\ s_1,\ \ldots,\ s_k)$*, tal que* $s_1 = \gamma(s_0,\ a_1),\ \ldots,\ s_k = \gamma(s_{k-1},\ a_k)$*, onde* s_k *é um estado objetivo. Tal seqüência de ações deve ser sintetizada pelo sistema de planejamento.*

3.2 Linguagens para planejamento

No planejamento independente de domínio, a representação de problemas de planejamento − estados, ações e objetivos − deve ser feita por meio de uma linguagem que seja suficientemente expressiva para descrever uma ampla variedade de problemas, mas restritiva o bastante para permitir que algoritmos eficientes operem sobre ela. A maioria das abordagens de planejamento adotam uma representação baseada em lógica para descrever estados, ações e para definir e computar facilmente o próximo estado $\gamma(s, a)$. A linguagem mais popular usada por planejadores clássicos é conhecida como STRIPS (*Stanford Research Institute Planning System*) [Fikes and Nilsson, 1971], tem sido estendida nos últimos 15 anos para abranger problemas de planejamento não-clássicos, isto é, planejamento para domínios mais complexos. Neste trabalho

3.2.1 Representação de estado

A linguagem STRIPS decompõe o mundo em condições lógicas e representa um estado como uma conjunção de literais positivos[1]. Um exemplo de literal proposicional que pode representar o estado de um agente desaparecido é $Perdido \wedge Incomunicavel$. Já literais de primeira ordem podem ser representados por $Cor(Bloco_1,\ Azul) \wedge Cor(Bloco_2,\ Vermelho) \wedge Sobre(Bloco_1,\ Bloco_2)$. Literais utilizados em descrições de estado de primeira ordem devem ser básicos e livres de funções. Além disso, numa representação de estado, quaisquer condições não mencionadas em um estado são consideradas falsas, esta premissa é conhecida como *Closed World Assumption* (CWA).

3.2.2 Representação de objetivo

Um objetivo na linguagem STRIPS é dado por uma conjunção de literais básicos positivos. Por exemplo, pode-se representar o objetivo de ter um

[1] Em lógica de predicados de primeira ordem, um *literal* é uma sentença atômica (um *literal positivo*) ou uma sentença atômica negada (um *literal negativo*).

bolo e estar com a cozinha limpa por meio de $Bolo \wedge CozinhaLimpa$ ou que o $Bloco_2$ deve estar sobre o $Bloco_1$ por $Sobre(Bloco_2,\ Bloco_1)$. Um estado s *satisfaz* um objetivo g se s contém todos os literais (ou proposições) em g. Por exemplo, o estado $Bolo \wedge CozinhaLimpa \wedge Suco$ satisfaz o objetivo $Bolo \wedge CozinhaLimpa$.

3.2.3 Representação de ações

Uma ação STRIPS é descrita pelas pré-condições (literais positivos) que devem ser válidas antes de a mesma ser aplicada e pelos efeitos após sua execução. Por exemplo, a composição da representação de uma ação equivalente a dirigir um carro de um local para outro pode ser observada na Tabela 3.2.

Tabela 3.1 (Exemplo de representação de uma ação STRIPS)**.**

Ação	*Dirigir(carro, origem, destino)*
Pré-condições	*Em(carro, origem)* $\wedge$ *Veículo(carro)* $\wedge$ *Cidade(origem)* $\wedge$ *Cidade(destino)*
Efeitos	$\neg$ *Em(carro, origem)* $\wedge$ *Em(carro, destino)*

A estrutura de uma ação STRIPS no planejamento clássico consiste de três partes:

- O **nome da ação** e a lista de parâmetros. Por exemplo, *Dirigir*(*carro, origem, destino*) serve para identificar a ação.
- A **pré-condição** é uma conjunção de literais positivos que devem ser verdadeiros em um estado antes da ação ser executada. Qualquer variável da pré-condição também deve aparecer na lista de parâmetros da ação.
- Os **efeitos** da ação são representados por uma conjunção de literais livres de funções que descrevem como o estado se altera quando a ação é executada. Um literal positivo (p) no efeito é considerado verdadeiro no estado resultante da ação, enquanto que sua negação ($\neg p$) significa que ele é falso naquele estado. As variáveis do efeito também devem aparecer na lista de parâmetros da ação.

Deste modo, uma ação é *aplicável* a qualquer estado que satisfaça a pré-condição; caso contrário, a ação não tem nenhum efeito. O resultado da execução de uma ação aplicável a em um estado s é s', o qual é calculado eliminando-se os literais negativos e adicionando-se os literais positivos dos efeitos de a. Assim, se um efeito positivo já estiver em s, ele não será adicionado uma segunda vez, e se um efeito negativo não estiver em s, o mesmo será ignorado. Qualquer literal não mencionado no efeito permanece inalterado.

3.2.4 Domínio do Mundo dos Blocos

O Mundo dos Blocos é um dos domínios de planejamento mais famosos [Winston, 1992]. Este domínio consiste em um conjunto de blocos em forma de cubo, dispostos sobre uma mesa. Os blocos podem ser empilhados, mas apenas um bloco pode ficar diretamente em cima de outro. Um braço robô pode levantar um bloco de cada vez, porém não consegue levantar um bloco que tenha outro em cima dele. O objetivo é construir uma ou mais pilhas de blocos, com especificações exatas de quais blocos devem ficar em cima de que outros blocos. Por exemplo, um objetivo poderia ser colocar o bloco A sobre B e o bloco B sobre C. Um exemplo deste domínio pode ser visto na Figura 3.2, e sua descrição encontra-se no Apêndice A.2.

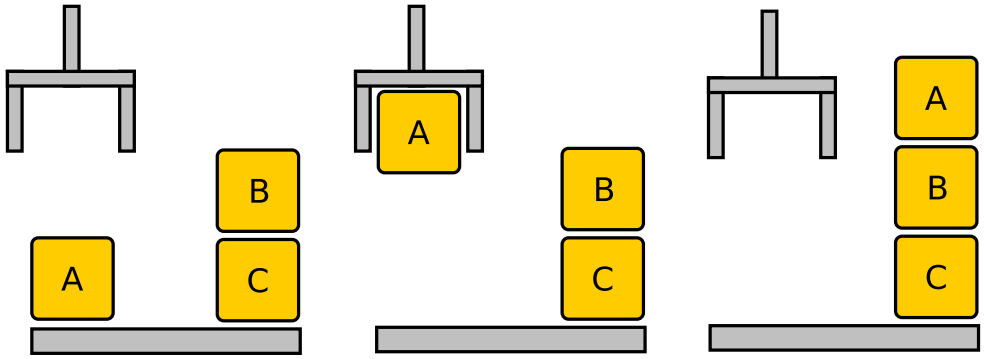

Figura 3.2: Este exemplo demonstra o comportamento de ações comuns no Mundo dos Blocos, ao empilhar o bloco A sobre o bloco B.

A Tabela 3.1 mostra a descrição de todos os estados, em STRIPS, apresentados na Figura 3.1.

Tabela 3.2 (Todos os estados do domínio do Mundo dos Blocos para um problema com 3 blocos)**.**

Estado	*Descrição*
#1	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)* ∧ *Livre(B)* ∧ *Livre(C)*
#2	*Garra(A)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, Mesa)* ∧ *Livre(B)* ∧ *Livre(C)*
#3	*GarraVazia* ∧ *Sobre(A, B)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)* ∧ *Livre(C)*
#4	*Garra(C)* ∧ *Sobre(A, B)* ∧ *Sobre(B, Mesa)* ∧ *Livre(A)*
#5	*GarraVazia* ∧ *Sobre(A, B)* ∧ *Sobre(B, Mesa)* ∧ ∧ *Sobre(C, A)* ∧ *Livre(C)*
#6	*GarraVazia* ∧ *Sobre(A, C)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)* ∧ *Livre(B)*
#7	*Garra(B)* ∧ *Sobre(A, C)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)*
#8	*GarraVazia* ∧ *Sobre(A, C)* ∧ *Sobre(B, A)* ∧ *Sobre(C, Mesa)* ∧ *Livre(C)*
#9	*Garra(B)* ∧ *Sobre(A, Mesa)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)* ∧ *Livre(C)*
#10	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, A)* ∧ *Sobre(C, Mesa)* ∧ *Livre(B)* ∧ *Livre(C)*
#11	*Garra(C)* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, A)* ∧ *Livre(B)*
#12	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, A)* ∧ *Sobre(C, B)* ∧ *Livre(C)*
#13	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, C)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)* ∧ *Livre(B)*
#14	*Garra(A)* ∧ *Sobre(B, C)* ∧ *Sobre(C, Mesa)* ∧ *Livre(B)*
#15	*GarraVazia* ∧ *Sobre(A, B)* ∧ *Sobre(B, C)* ∧ *Sobre(C, Mesa)* ∧ *Livre(A)*
#16	*Garra(C)* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, Mesa)* ∧ *Livre(A)* ∧ *Livre(B)*
#17	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, A)* ∧ *Livre(B)* ∧ *Livre(C)*
#18	*Garra(B)* ∧ *Sobre(A, Mesa)* ∧ *Sobre(C, A)* ∧ *Livre(C)*
#19	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, C)* ∧ *Sobre(C, A)* ∧ *Livre(B)*
#20	*GarraVazia* ∧ *Sobre(A, Mesa)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, B)* ∧ *Livre(A)* ∧ *Livre(C)*
#21	*Garra(A)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, B)* ∧ *Livre(C)*
#22	*GarraVazia* ∧ *Sobre(A, C)* ∧ *Sobre(B, Mesa)* ∧ *Sobre(C, B)* ∧ *Livre(A)*

Na Tabela 3.3 é possível observar as ações STRIPS para o domínio do Mundo dos Blocos.

Tabela 3.3 (Ações STRIPS para o domínio do Mundo dos Blocos)**.**

Ação	*PegarMesa(bloco)*
Pré-condições	*GarraVazia* $\wedge$ *Sobre(bloco, Mesa)* $\wedge$ *Livre(bloco)*
Efeitos	$\neg$ *GarraVazia* $\wedge$ *Garra(bloco)* $\wedge$ $\neg$ *Sobre(bloco, Mesa)* $\wedge$ $\neg$ *Livre(bloco)*
Ação	*ColocarMesa(bloco)*
Pré-condições	*Garra(bloco)*
Efeitos	*GarraVazia* $\wedge$ $\neg$ *Garra(bloco)* $\wedge$ *Sobre(bloco, Mesa)* $\wedge$ *Livre(bloco)*
Ação	*Desempilhar(*$bloco_1$*,* $bloco_2$*)*
Pré-condições	*GarraVazia* $\wedge$ *Sobre(*$bloco_1$*,* $bloco_2$*)* $\wedge$ *Livre(*$bloco_1$*)*
Efeitos	$\neg$ *GarraVazia* $\wedge$ *Garra(*$bloco_1$*)* $\wedge$ $\neg$ *Sobre(*$bloco_1$*,* $bloco_2$*)* $\wedge$ $\neg$ *Livre(*$bloco_1$*)* $\wedge$ *Livre(*$bloco_2$*)*
Ação	*Empilhar(*$bloco_1$*,* $bloco_2$*)*
Pré-condições	*Garra(*$bloco_1$*)* $\wedge$ *Livre(*$bloco_2$*)*
Efeitos	*GarraVazia* $\wedge$ $\neg$ *Garra(*$bloco_1$*)* $\wedge$ *Sobre(*$bloco_1$*,* $bloco_2$*)* $\wedge$ *Livre(*$bloco_1$*)* $\wedge$ $\neg$ *Livre(*$bloco_2$*)*

3.2.5 Linguagens para domínios reais

Com as definições dadas da linguagem STRIPS é possível definir uma *solução* para planejamento como uma seqüência de ações que, quando executadas, resultam em um estado que satisfaz o objetivo.

Para domínios reais, a linguagem STRIPS não é considerada uma linguagem expressiva o suficiente. Devido a isso, foram desenvolvidas muitas variantes de linguagem, sendo uma delas a *Action Description Language* (ADL) [Pednault, 1989]. Em STRIPS só é permitido o uso de literais positivos em estados, enquanto em ADL são permitidos literais positivos e negativos. STRIPS trabalha com a *suposição de mundo fechado*, em que literais não mencionados são falsos. Em contrapartida, a ADL trabalha com a *suposição de mundo aberto*, em que literais não mencionados são desconhecidos. Uma outra diferença importante é que em STRIPS não há funções como para igualdade, e não é possível definir tipos de dados; já ADL apresenta tais características, bem como quantificadores sobre objetos do domínio.

As diversas formas de planejamento em IA podem ser especificadas por

meio de uma sintaxe padrão denominada PDDL (*Problem Domain Definition Language*) [McDermott, 1998] [McDermott and Committee, 1998], que inclui, entre outras, a linguagem STRIPS e ADL. Mais informações sobre PDDL podem ser encontradas no Apêndice A.1.

3.3 Algoritmos tradicionais para planejamento clássico

3.3.1 Busca no espaço de estados

A busca no espaço de estados é empregada por vários algoritmos de planejamento. O espaço de estados pode ser representado por um grafo, em que cada nó corresponde a um estado do mundo e cada aresta a uma transição de estado. O planejamento no espaço de estados divide-se basicamente em algoritmos *progressivos* e *regressivos*.

O algoritmo progressivo de busca parte do estado inicial do plano e aplica de forma não-determinística, a função de transição de estado, isto é, todas as seqüências de ações possíveis, produzindo outros estados e, conseqüentemente, subproblemas. Estes subproblemas buscam soluções parciais que levam ao estado desejado, pois são parte do problema original. A busca é finalizada quando um destes subproblemas consegue alcançar um estado objetivo ou quando não há nenhum plano possível, caracterizando assim uma falha.

Uma das características do planejamento progressivo é o de ter conhecimento completo sobre o estado do mundo a qualquer instante. Isto se deve ao fato de que este opera a partir de um estado inicial completamente especificado e aplica ações aos estados, resultando em mais especificações completas. O Algoritmo 1 exemplifica o processo de planejamento progressivo.

Algoritmo 1: Planejamento progressivo.

Entrada: Estado inicial s_0, Objetivo S_g, Domínio $\mathcal{D}$
Saída: Plano π

```
1  início
2      π ← 0;
3      s ← s_0;
4      repita
5          se s ∈ S_g então
6              devolve π;
7          A ← {a | a é uma ação aplicável a s};
8          se A = 0 então
9              devolve falha;
10         não-deterministicamente escolha a ∈ A;
11         π ← π + a;
12         s ← γ(s, a);
13     até ;
14 fim
```

Também é possível efetuar planejamento no espaço de estados por meio de uma busca regressiva. Neste processo o procedimento é o inverso da busca progressiva, começando pela aplicação da função de transição inversa ao estado objetivo. Com isso são produzidos estados predecessores, nos quais a função de transição é aplicada novamente e assim sucessivamente até que chegue ao estado incial. As representações que seguem o modelo STRIPS tornam esta descrição bastante fácil, porque os conjuntos de estados podem ser descritos pelos literais que devem ser verdadeiros em tais estados (isto é, as pré-condições das ações).

Desde o início das pesquisas, na década de 60, com o *General Problem Solver* (GPS) [Newell and Simon, 1961], os algoritmos de busca progressiva são utilizados, sendo especializados com heurísticas A^* [Hart et al., 1968] e, hoje, correspondem aos melhores planejadores para o modelo restrito (planejamento clássico).

3.3.2 Busca no espaço de planos

Planejamento no espaço de planos é um outro meio de encontrar soluções para problemas de planejamento clássico [Ghallab et al., 2004]. A busca progressiva e regressiva no espaço de estados são formas específicas de busca de planos *totalmente ordenados*. Elas exploram apenas seqüências estritamente lineares de ações conectadas de forma direta ao estado inicial ou ao objetivo, ou seja não podem se beneficiar da decomposição do problema. Ao invés de atuarem sobre cada subproblema separadamente, elas sempre têm de tomar decisões a respeito de como definir seqüências de ações a partir de todos os subproblemas [Russell and Norvig, 2002].

Por outro lado, uma abordagem que apresenta vários sub-objetivos independentes demonstra uma vantagem de flexibilidade na ordem de elaboração do plano. O planejador pode trabalhar primeiro em decisões óbvias ou importantes, onde somente algumas ações são ordenadas com relação às demais (compromissos fracos), em vez de ser forçado a atuar em etapas dispostas em ordem cronológica em relação às ações (compromissos fortes).

Deste modo, qualquer algoritmo de planejamento que possa inserir duas ações em um plano sem especificar qual delas deve ser executada primeiro é definido como *planejamento de ordem parcial*. O planejador de ordem parcial pode ser implementado sob a forma de uma busca no espaço de planos de ordem parcial (em alguns momentos chamados apenas de planos parciais).

O processo se inicia com um plano vazio, em seguida, são considerados meios de aprimorar o plano até que se obtenha um plano completo que resolva o problema. As ações nesta busca não são ações no mundo, mas ações sobre planos: adicionar um passo ao plano, impor uma ordenação que coloque uma ação antes de outra e, assim sucessivamente. Os nós do espaço de busca são planos e, em sua maioria, não concluídos, isto é, planos parciais. Cada plano possui quatro componentes, sendo que os dois primeiros definem os passos do plano e os dois últimos determinam como os planos podem ser

estendidos. Os componentes do plano são descritos a seguir:

- **Um conjunto de ações que compõem os passos do plano**. Estas ações são obtidas do conjunto de todas as ações possíveis para o problema de planejamento,sendo que um plano *vazio* contém apenas as ações *Iniciar* e *Terminar*. *Iniciar* não tem pré-condições e apresenta como efeito todos os literais no estado inicial do problema de planejamento. *Terminar* não tem efeitos e tem como pré-condições os literais de objetivo do problema de planejamento.

- **Um conjunto de restrições de ordenação**. Cada restrição de ordenação tem a forma $A \prec B$, isto significa que a ação A deve ser executado antes de B, mas não imediatamente antes.

- **Um conjunto de vínculos causais**. Um vínculo causal entre as ações A e B no plano é escrito como $A \xrightarrow{p} B$. Isto afirma que p é um efeito da ação A e uma pré-condição de B, e também significa que p deverá permanecer verdadeiro entre as ações A e B. A ação A é chamada de ação que contribui com p, e B a ação que necessita de p. Deste modo, o plano não pode ser estendido adicionando-se uma nova ação C que esteja em conflito com o vínculo causal. A ação C está em conflito com $A \xrightarrow{p} B$, se C tem efeito $\neg p$ e se C pode ocorrer depois de A e antes de B. Vínculos causais também podem ser chamados de *intervalos de proteção*.

- **Um conjunto de pré-condições abertas**. Uma pré-condição é aberta se não é alcançada por alguma ação do plano. Os planejadores trabalham para reduzir o conjunto de pré-condições abertas a um conjunto vazio, sem introduzir conflitos.

Um *plano consistente* é aquele que não possui ciclo nas restrições de ordenação e nenhum conflito com os vínculos causais. Deste modo, um plano consistente sem pré-condições abertas é uma solução. Um exemplo de plano

de ordem parcial e suas linearizações correspondentes em planos de ordem total pode ser visto na Figura 3.3.

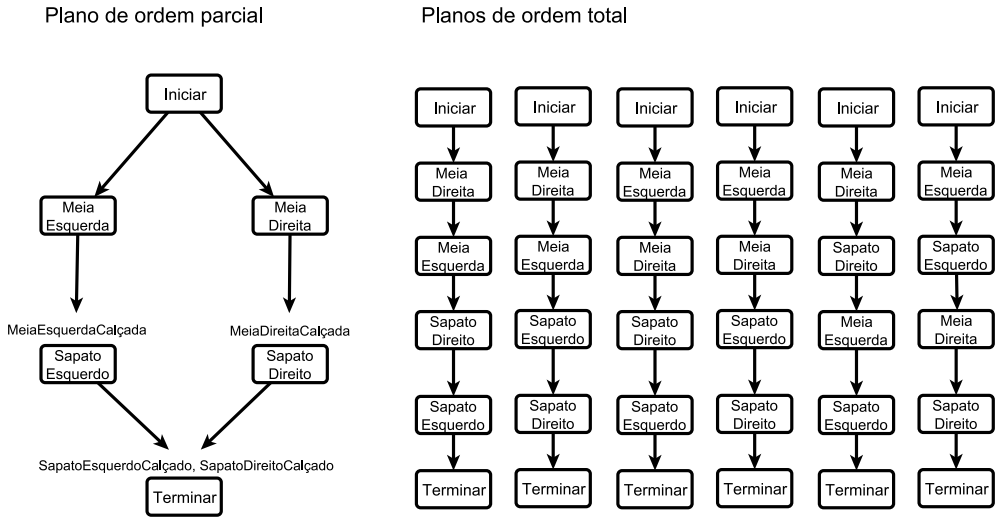

Figura 3.3: Exemplo de um plano de ordem parcial para calçar sapatos e meias, e as seis linearizações correspondentes em planos de ordem total [Russell and Norvig, 2002]. A solução é apresentada em um grafo de ações no qual é possível observar as ações Iniciar e Terminar, que marcam o início e o final do plano. A solução de ordem parcial corresponde a seis planos possíveis de ordem total; cada um desses planos é uma linearização do plano de ordem parcial.

Um algoritmo clássico de planejamento como busca no espaço de planos, completamente provado e correto, é o *Partial Order Planner* (POP) [Russell and Norvig, 2002].

3.4 Planejamento por refinamento

A construção de um plano pode ser vista como um refinamento iterativo do conjunto de todos os planos possíveis. Esta estratégia é chamada de *planejamento por refinamento* ([Kambhampati, 1997]). É possível demonstrar que a maioria dos algoritmos clássicos de planejamento pode ser compreendida desta maneira (incluindo o POP). A idéia principal do planejamento por refinamento é que, iniciando com um conjunto de todas as seqüências possíveis

de ações do domínio (ações STRIPS), num processo iterativo, adicionam-se restrições de modo a reduzir este espaço de planos. As restrições podem impor ordem entre as ações ou que uma proposição particular deva ser verdade em um ponto específico do plano. Cada etapa (ou passo) do plano é identificada por um elemento único e corresponde a uma ação. A seguir, são descritos os tipos básicos de restrições no planejamento por refinamento:

Restrições de ordem

- *Restrição de precedência.* Um passo precede outro passo do plano, sendo que outros passos podem ocorrer entre eles.
- *Restrição de contigüidade.* Um passo deve preceder imediatamente um outro passo, isto é, nenhum outro passo deve ocorrer entre eles.

Restrições auxiliares

- *Restrição de proteção de intervalo ou de vínculo causal.* Essa restrição impõe que uma condição deve permanecer verdadeira sobre um intervalo (nenhuma ação que tem efeito p será permitida em um intervalo em que a condição $\neg p$ deva ser preservada).
- *Restrição de verdade pontual.* A verdade de uma condição em um ponto particular do tempo deve ser preservada.

Neste tipo de planejamento, o processo de adição de restrições continua até que todos os planos que satisfaçam as restrições sejam soluções para o problema. Durante este refinamento não é armazenado o conjunto de todos os planos *candidatos*, mas apenas as restrições. Estas restrições são armazenadas em um *plano parcial*, sendo que, neste caso, um plano parcial $\mathcal{P}$ representa um conjunto de planos candidatos, que será chamado de *candidatos*($\mathcal{P}$).

Uma estratégia de *refinamento* define como um plano parcial deve ser estendido por meio da adição de novas restrições, sendo que tal adição será responsável pelo refinamento do conjunto de planos candidatos. Uma estratégia de refinamento $\mathcal{R}$ é uma função que mapeia um plano parcial $\mathcal{P}$ em um conjunto de planos parciais, isto é, $\mathcal{P} = \mathcal{P}_1, \ldots, \mathcal{P}_i, \ldots, \mathcal{P}_n$, de forma que, para cada um dos planos parciais $\mathcal{P}_i$, o conjunto de candidatos ($\mathcal{P}_i$) seja um subconjunto de *candidatos*($\mathcal{P}$). A Figura 3.4 ilustra a definição geral de um planejador por refinamento.

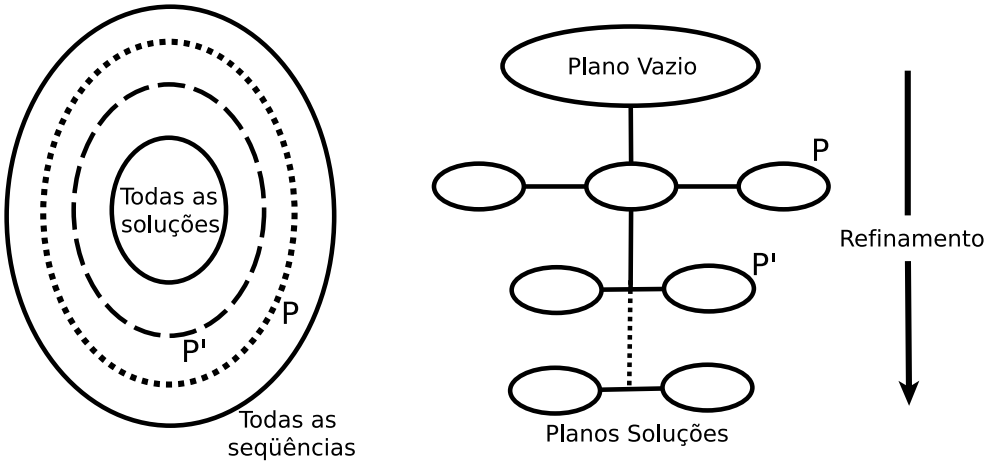

Figura 3.4: Visão geral de planejamento por refinamento [Kambhampati, 1997].

A partir de um conjunto vazio de restrições, representado por um plano parcial vazio $\mathcal{P}$; é possível verificar se um dos candidatos de $\mathcal{P}$ é uma solução do problema. Se for, finaliza-se o processo; caso contrário, aplica-se novamente uma estratégia de refinamento $\mathcal{R}$ para obter uma coleção de planos parciais $\mathcal{P}' = \mathcal{R}(\mathcal{P})$, em que cada plano parcial $\mathcal{P}_i$ possui uma restrição adicional em relação a $\mathcal{P}$. Isto é feito selecionando-se um plano candidato de $\mathcal{P}'$ e verificando novamente se ele é uma solução; caso não seja, aplica-se novamente $\mathcal{R}$. Este processo deve continuar até que se obtenha uma solução ou até que o conjunto de planos parciais esteja vazio e, neste caso, é necessário retroceder. O Algoritmo 2 realiza o processo de planejamento por refinamento.

Algoritmo 2: Planejamento por refinamento

Entrada: Plano parcial $\mathcal{P}$, Problema Π
Saída: Solução para Π ou *falha*

```
1  início
2      se candidatos(P) = ∅ então
3          devolve falha;
4      se existe solucao(P, Π) então
5          devolve solucao(P, Π);
6      Selecionar estratégia de refinamento R;
7      Gerar novo conjunto de planos ⟨P, H'⟩ = R(P);
8      para todo P_i ∈ P escolhido não-deterministicamente faça
9          Planejamento Por Refinamento (P_i, Π, H');
10 fim
```

3.4.1 VHPOP

Um dos planejadores que implementa de forma explícita o planejamento por refinamento é o *Versatile Heuristic Partial Order Planner* (VHPOP).

O VHPOP é um planejador de ordem parcial (*Partial Order Causal Link* - POCL) baseado no UCPOP (*Partial Order Planner with Conditional effects and Universal quantification*) [Penberthy and Weld, 1992]. O VHPOP é resultado de experiências obtidas em meados dos anos 90 no estudo de estratégias para planejamento POCL, combinado com avanços no campo do planejamento independente de domínio, como análise de alcançabilidade [Younes and Simmons, 2003].

Ao incorparar técnicas para restrições temporais, o VHPOP assume a capacidade de fazer planejamento utilizando ações com duração de tempo. Além disso, ele demonstra que as mesmas técnicas heurísticas usadas para auxiliar a execução do planejamento clássico POCL podem ser efetivas em domínios com restrições temporais.

O VHPOP implementa um conjunto de diferentes heurísticas para a escolha de ações durante o planejamento, como Dunf e DSep [Peot and Smith, 1993], LCFR [Joslin and Pollack, 1994] e ZLIFO [Schubert and Gerevini, 1995]. Além disso, durante o processo de escolha, ele pode trabalhar tanto em ações *ground* (totalmente instanciadas) ou *lifted* (parcialmente instanciadas). Deste modo, ele é classificado como um planejador POCL com versatilidade heurística com base em *Constraint Satisfaction Problems* (CSP).

Capítulo 4

Planejamento não-clássico

Nos capítulos anteriores foram considerados apenas domínios de planejamento clássico que são completamente observáveis, estáticos e determinísticos. Além disso, supôs-se que as descrições de ações são sempre corretas e completas. Nestas circunstâncias, um agente pode primeiro planejar e depois executar o plano, sem a necessidade de qualquer percepção. Entretanto, em um ambiente incerto, um agente deve usar suas percepções para descobrir o que está acontecendo enquanto o plano é executado e, possivelmente, modificar ou substituir o plano se algo inesperado acontecer.

Os modelos utilizados pelos planejadores no mundo real são muito mais complexos do que aqueles empregados no planejamento clássico. Eles ampliam os fundamentos em termos de linguagem de representação e também o modo como o planejador interage com o ambiente. Este capítulo mostra como o relaxamento de algumas suposições restritivas pode estender o planejamento clássico de modo que permita uma melhor representação e interação com o domínio, além de conter uma apresentação sobre replanejamento e reparo de plano.

4.1 Modelos estendidos

Muitos modelos interessantes podem ser obtidos quando algumas das suposições restritivas são relaxadas.

- **Suposição A0 relaxada (Σ finito).** Um conjunto, possivelmente infinito, de estados enumeráveis pode ser necessário, por exemplo, para descrever ações que constroem ou inserem novos objetos no universo ou para manipular variáveis de estado numéricas.

- **Suposição A1 relaxada (Σ completamente observável).** O sistema Σ é parcialmente observável. Para cada observação o, pode haver mais de um estado s tal que $\eta(s) = O$. Desconhecendo o estado atual em $\eta^{-1}(o)$, não é possível prever com certeza qual será o estado Σ após cada ação.

- **Suposição A2 relaxada (Σ determinístico).** Num sistema estático, mas não-determinístico[1], cada ação pode levar a diferentes estados, levando o planejador a considerar alternativas. Um plano deve codificar formas de lidar com alternativas.

  Existem diferentes abordagens para lidar com não-determinismo. Algumas delas ampliam as técnicas usadas no planejamento clássico (como planejamento baseado em grafo ou em satisfatibilidade), enquanto outras são projetadas especificamente para lidar com não-determinismo, como o planejamento baseado no *Markov Decision Process* (MDP) e o planejamento com verificação de modelos.

- **Suposição A3 relaxada (Σ estático).** Pode-se facilmente lidar com um sistema dinâmico Σ se este for determinístico e completamente observável, e considerando-se que para cada estado s haja, no máximo, um evento exógeno para o qual existe $\gamma(s, e)$ e que, necessariamente, ocorrerá em s. Tal sistema pode ser mapeado em um modelo restrito: é possível redefinir a transição para uma ação a como $\gamma(\gamma(s, a),\ e)$, em que e é o evento que ocorre no estado $\gamma(s, a)$.

[1] Também conhecido como indeterminismo.

Quando a propriedade A3 é relaxada, além dos eventos que podem ou não ocorrer, existem ações que também podem ser executadas ou não. Sob o ponto de vista de planejamento, tais ações e eventos concorrem em um sistema dinâmico, mesmo quando $|\gamma(s, u)| < 1$. A decisão de aplicar a ação a em s não foca as previsões do planejador para uma única transição de estado.

- **Suposição A4 relaxada (objetivos restritos).** Controlar um sistema pode exigir objetivos mais complexos do que simplesmente alcançar um dado estado. É possível que haja a necessidade de especificar um objetivo ampliado para o planejador, com exigências não apenas no estado final, mas também nos estados s percorridos. Por exemplo, estados críticos a serem evitados, estados pelos quais o sistema deverá passar, estados em que deverá permanecer e outras limitações em sua trajetória. Poderá ser necessário também otimizar funções utilitárias, como modelar um sistema que funcione continuamente por um período de tempo indefinido.

- **Suposição A5 relaxada (plano seqüencial).** Um plano pode ser uma estrutura matemática mais rica do que uma simples seqüência de ações. Pode-se considerar que um plano é um conjunto parcialmente ordenado, uma seqüência de conjuntos, um plano condicional que força rotas alternativas dependendo dos resultados e do contexto atual da execução, um plano universal ou um programa que mapeie estados para adequar ações, ou ainda uma automação que determine qual ação executar, dependendo do histórico de execuções anteriores.

- **Suposição A6 relaxada (tempo implícito).** Em muitos domínios de planejamento a duração e a concorrência das ações precisam ser consideradas, uma vez que ações sempre levam um tempo para serem concluídas e podem demandar recursos. O tempo pode ser necessário também para expressar limitações de objetivos temporários e a ocorrência de eventos em relação a uma referência de tempo absoluto. Entretanto, tempo é uma distância abstrata no modelo de transição de estado. Este modelo conceitual considera ações ou eventos como transições instantâneas: em cada iteração, o controlador lê

sincronicamente a observação para o estado atual, se necessário, e aplica a ação planejada.

- **Suposição A7 relaxada (planejamento offline).** O planejador pode não precisar se preocupar com todos os detalhes da dinâmica atual, mas não pode ignorar completamente como o sistema se desenvolve [Nareyek, 2003]. Sendo assim, precisará checar online se um plano-solução permanece válido e, se necessário, *revisá-lo* ou *replanejá-lo*.

4.2 Planejamento não-determinístico

Em um ambiente com incerteza no efeito das ações, um agente deve usar suas percepções para descobrir o que está acontecendo enquanto o plano é executado e, possivelmente, modificar ou substituir o plano, caso aconteça algo inesperado ou indesejado.

O agente deve lidar com informações *incompletas* ou *incorretas*. A incompletude surge porque o mundo é parcialmente observável, não-determinístico ou ambos. Por exemplo, uma gaveta pode estar trancada ou não; uma das chaves do agente pode abrir a gaveta, se ela estiver trancada, e o agente pode ou não estar ciente destes tipos de incompletude em seu conhecimento. Deste modo, o modelo do mundo é fraco, mas correto. Por outro lado, a incorreção surge porque o mundo não corresponde ao modelo de mundo do agente, por exemplo: ele pode *acreditar* que sua chave abre a gaveta, mas pode estar errado caso a fechadura tenha sido trocada.

A possibilidade de ter conhecimento completo ou correto depende de quanto de não-determinismo existe no mundo. Com o **não-determinismo limitado**, as ações podem ter efeitos imprevisíveis, mas tais efeitos podem ser listados nos axiomas de descrição de ações. Um agente pode lidar com o não-determinismo limitado fazendo planos que funcionam em todas as circunstâncias possíveis ou incluindo ações de percepção do plano (plano condicional). Por outro lado, com o **não-determinismo ilimitado**, o con-

junto de pré-condições ou efeitos possíveis é desconhecido ou grande demais para ser enumerado completamente. Este seria o caso em domínios muito complexos ou dinâmicos. Um agente pode lidar com o não-determinismo ilimitado apenas se estiver preparado para rever seus planos ou sua base de conhecimento. Existem cinco tipos de não-determinismo em planejamento [Russell and Norvig, 2002].

- **Não-determinismo limitado**
  - **Planejamento sem sensores**. Também chamado *planejamento conformante*, é um método que constrói planos seqüenciais que devem ser executados sem percepção. O algoritmo de planejamento sem sensores deve assegurar que o plano atinja o objetivo *em todas as circunstâncias possíveis*, independente do verdadeiro estado inicial e dos resultados das ações. O planejamento sem sensores se baseia na *coerção* − a idéia de que o mundo pode ser forçado a entrar em um determinado estado, mesmo quando o agente só tem informações parciais a respeito do estado atual. A coerção nem sempre é possível e, portanto, o planejamento sem sensores, em geral, é inaplicável. O primeiro planejador com conformação moderadamente eficiente foi o *Conformant Graphplan* (CGP) de [Smith and Weld, 1998].
  - **Planejamento condicional**. Pode ser utilizado nos casos em que todos os prováveis efeitos não-determinísticos das ações são previsíveis e existe a possibilidade de se construir planos que incluam ações de percepção. Também conhecida como *planejamento de contingência*, esta abordagem lida com o não-determinismo limitado, construindo um plano condicional com ramificações distintas para as diferentes contingências que podem surgir. Assim como no planejamento clássico, o agente planeja primeiro e, depois, executa o plano. O agente descobre qual parte do plano deve executar, incluindo ações de percepção no plano para testar a presença das condições apropriadas. O WARPLAN-C [Warren, 1976], uma variante do WARPLAN, foi um dos pri-

meiros planejadores a usar ações condicionais. Uma outra abordagem, na qual são construídos planos condicionais com laços, baseada no *Binary Decision Diagram* (BDD), é descrita em [Hansen and Zilberstein, 2001].

- **Planejamento probabilístico** Nos casos em que é possível identificar distribuições de probabilidades nos efeitos das ações, a técnica predominantemente utilizada é a representação de problemas por meio de um MDP. O objetivo do planejador é obter uma *política* que, conseqüentemente, mapeará ações para cada estado sofre um conjunto de estados iniciais, buscando maximizar a utilidade esperada de planos. [Bonet and Geffner, 2000] descrevem um planejador baseado na heurística no espaço de estados para o *Partially Observable Markov Decision Process* (POMDP). O planejador C-BURIDAN, definido por [Draper et al., 1994], manipula ações com resultados probabilísticos, abordando também o POMDP.

- **Não-determinismo ilimitado**

  - **Replanejamento ou reparo de plano**. Nesta abordagem o agente pode usar qualquer uma das técnicas de planejamento (clássica, conformante, condicional ou probabilística) para construir um plano. Por meio do *monitoramento de execução* é possível julgar se o plano precisa ou não ser revisto. A revisão do plano ocorre quando algo sai errado; então é necessário fazer modificações nele (*reparo de plano*) ou realizar um novo planejamento a partir do estado atual (*replanejamento*). O *System for Iteractive Planning and Execution Monitoring* (SIPE) [Wilkins, 1988] [Wilkins, 1990] foi o primeiro planejador a lidar sistematicamente com o problema de replanejamento.
  - **Planejamento contínuo**. Um planejador contínuo é projetado para persistir ao longo do tempo. Ele pode manipular circunstâncias inesperadas no ambiente, ainda que essas circunstâncias ocorram enquanto o agente está em meio a uma execução ou construção de um plano. Este tipo de planejamento também

deve lidar com mudanças de metas do agente. O *Integrated Planning, Execution, and Monitoring* (IPEM) foi o primeiro sistema a integrar o planejamento de ordem parcial e a execução para produzir [Ingerson and Steel, 1988] um agente de planejamento contínuo.

4.3 Replanejamento e reparo de plano

Este trabalho está interessado em problemas de planejamento na presença de não-determinismo ilimitado porém com ações determinísticas para os quais a técnica de reparo de plano possa levar o agente a satisfazer suas metas com sucesso. Ou seja, a proposta é não representar ações não-determinísticas explicitamente, mas tratar o não-determinismo por meio de reparos de plano gerados a partir de um planejador determinístico. Define-se *replanejamento* como um novo processo de planejamento a partir do estado atual de uma execução de um plano que apresentou uma falha. De acordo com [Nebel and Koehler, 1993], no pior caso, reparar um plano existente não é mais eficaz do que um replanejamento. Entretanto, como uma grande parte do plano geralmente ainda é válida, na prática, o reparo de plano pode ser mais eficaz, [Kambhampati, 1997] pois, além disso, em muitos domínios pode ser mais caro modificar todo o plano, devido a compromissos com outros agentes baseados no plano original [Cushing and Kabhampati, 2005].

O Algoritmo 3 descreve um exemplo simples de um agente que realiza reparo de plano. Ele utiliza um algoritmo de planejamento (que pode ser qualquer um dos apresentados no Capítulo 3, denominado *planejador*, como uma chamada a um método (linhas `3` e `8`). Se as pré-condições da próxima ação não forem satisfeitas, o agente tentará encontrar um caminho que o leve de volta ao plano original. Esse caminho é chamado *reparo*. Se o planjedor for bem-sucedido na descoberta de um reparo, o agente acrescentará o reparo ao plano original, a fim de criar um novo plano. Em seguida, o agente prossegue na execução das ações do novo plano. A Figura 4.1 ilustra a execução de um plano em que é necessário um reparo para que o objetivo seja alcançado.

Algoritmo 3: Agente Reparador de Plano.

Entrada: Percepção do ambiente $\mathcal{O}$, Objetivo S_g, Domínio $\mathcal{D}$
Saída: *Sucesso* ou *falha* na execução do plano

```
1  início
2      s ← η(O);
3      π ← planejador(s, S_g, D);
4      para todas as ações a do π faça
5          se pré-condições(a, s) não-verdadeiras então
6              s_esperado ← calcula(s, a);
7              s ← η(O);
8              π_reparo ← planejador(s, s_esperado, D);
9              se π_reparo ≠ falha então
10                 π ← π_reparo + π;
11             senão
12                 devolve falha;
13         senão
14             executa(a);
15             π ← π − a;
16             s ← η(O);
17     se s ≠ S_g então
18         devolve falha;
19     devolve sucesso;
20 fim
```

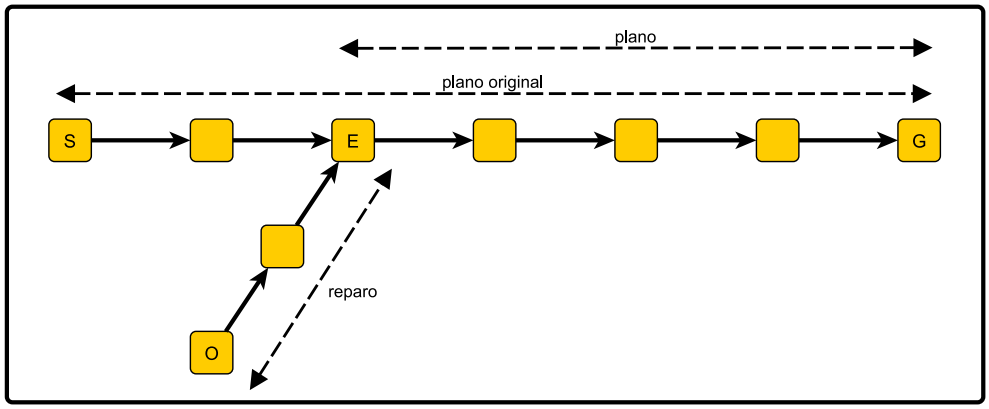

Figura 4.1: Antes da execução o planejador apresenta um plano, aqui chamado de *plano original*, para ir de S até G. O agente executa o plano até o ponto marcado com E. Antes de executar o *plano* restante, ele verifica as pré-condições e descobre que está, de fato, no estado O. Em seguida, ele chama seu algoritmo de planejamento para apresentar um reparo, um plano para ir de O até algum ponto no *plano original*.

A proposta desta dissertação é abordar o método de reparo de plano de forma mais robusta, considerando o processo de reparação a partir de duas operações distintas: (*1*) remover ações do plano que, no momento, tornam mais difícil atingir o(s) objetivo(s) e (*2*) adicionar ações que aproximem o agente do objetivo, sendo que a última operação é muito similar ao planejamento clássico.

4.3.1 Trabalhos correlatos

A utilização de métodos para reparo de plano, ao invés de se fazer o replanejamento, não é recente. Segue uma breve revisão das principais propostas de reparo de plano encontradas na literatura:

- O *Greedy Planning Graph* (GPG) [Gerevini and Serina, 2000] é baseado em grafos de planos. Ele utiliza uma abordagem apoiada no planejador [Blum and Furst, 1997] Graphplan. Quando o plano se torna inválido, o GPG verifica onde ocorrem as inconsistências do plano. O

plano é então dividido em três partes: o topo do plano, que consiste em ações que podem ser executadas a partir do estado inicial; uma parte intermediária, que é composta por um conjunto de ações inconsistentes e as ações entre elas; e uma final, que pode ser utilizada para atingir os objetivos assim que as inconsistências tenham sido resolvidas. Estas três partes podem ser identificadas utilizando-se o grafo de planejamento que foi construído durante a fase de planejamento. A parte intermediária é então descartada e um plano é procurado para preencher a lacuna existente entre o topo e o final do plano. Se este plano não puder ser encontrado a lacuna é ampliada e o processo é repetido. Eventualmente pode ocorrer de todo o plano ser descartado. Neste caso, descarta-se a possibilidade de reparo e um plano completamente novo é construído, se for possível.

- O modelo de planos do REPLAN [Boella and Damiano, 2002] é similar aos planos utilizados nos formalismos de *Hierarchical Task Network* (HTN) [Erol et al., 1994]. Uma rede de tarefas descreve uma possível forma de realizar uma tarefa por meio de sua decomposição em subtarefas ou, eventualmente, em ações primitivas (ou seja, ações que o agente pode executar de forma automática). Para cada tarefa existe pelo menos uma destas redes de tarefas. Um plano é criado pela escolha da rede de tarefas correta para cada tarefa (abstrata), até que cada rede contenha apenas ações primitivas. Por meio deste processo de planejamento, o REPLAN constrói uma árvore de derivação que inclui todas as tarefas escolhidas e demonstra como um plano foi derivado.

  O reparo de plano dentro do REPLAN é chamado de partição. Para cada nó inválido da árvore de derivação, a (menor) sub-árvore que contém este nó é removida. Inicialmente, cada nó considerado inválido é uma ação primitiva, e a raiz da árvore correspondente é a tarefa que continha tal ação. Subseqüentemente, um novo reparo é gerado para esta tarefa. Se o reparo falhar, uma nova etapa é iniciada, na qual sub-árvores para tarefas hierarquicamente mais elevadas são removi-

das e regeradas. No pior caso esse processo continua até que toda a árvore de derivação seja descartada.

- O O-Plan [Drabble et al., 1997] utiliza a estratégia de regras para reparo de plano. Durante a execução o sistema confirma os efeitos de cada ação. Para cada efeito que falha em que uma ação é necessária para que outra seja executada, ações adicionais, na forma de um reparo, são incluídas no plano. Estes reparos de plano são planos pré-construídos que podem reparar determinadas condições de falhas. Por exemplo, os planos de reparo podem incluir um plano para a troca de um pneu furado ou para a substituição de um motor quebrado. Quando uma condição incorreta é encontrada, a execução do plano é interrompida e um reparo de plano é inserido e executado. Após a finalização do reparo, a execução do plano regular recomeça. O O-Plan apenas adiciona ações para reparar falhas e não emprega qualquer tipo de remoção de ações. Deste modo, ele também é incompleto, pois nem todas as falhas podem ser recuperadas. Xuemei Wang e Steve A. Chien [Wang and Chien, 1997] descrevem como a busca pode ser incorporada ao O-Plan para tentar recuperá-lo de falhas para as quais nenhuma estratégia de reparo pré-construída esteja disponível. Entretanto, não considera a remoção de ações de um plano para a recuperação das falhas, mas apenas para a descoberta de quais ações precisam ser executadas novamente.

Capítulo 5

Reparo de plano por refinamento reverso

Este capítulo apresenta a estratégia de reparo de plano por refinamento reverso e do método heurístico desenvolvido neste trabalho.

5.1 Refinamento reverso

Para o reparo de plano não se pode utilizar diretamente o modelo de planejamento por refinamento, pois esta estratégia só permite adicionar ações, enquanto no reparo é preciso, além de adicionar, retirar ações. O reparo de plano constitui de duas atividades distintas: a remoção de ações que estejam impedindo o sucesso do plano e a ampliação do plano, por meio da adição de ações ([van der Krogt and de Weerdt, 2004b]). Por este motivo, é necessário incluir uma *estratégia de refinamento reverso* para o reparo de plano.

A remoção de restrições do plano parcial impede que o plano atinja seus objetivos. A adição de ações pode ser tratada como um planejamento normal, em que o plano parcial é ampliado (refinado) para satisfazer os objetivos. O Exemplo 5.1.1 ilustra uma situação em que apenas o reparo não é suficiente para que o plano atinja seu objetivo, sendo necessário um processo de refinamento reverso a fim de que o plano seja reparado.

Exemplo 5.1.1. *Suponha que exista um plano para um indivíduo ir a um encontro utilizando um carro. Entretanto, ao se aproximar do carro, ele nota que um dos pneus está furado. Um simples reparo para este plano poderia ser adicionar ações para a troca do pneu por um sobressalente e, então, prosseguir com o resto do plano, o que corresponderia à adição de ações ao plano falho. Porém, supondo que este é um encontro muito importante para o qual o indivíduo não quer se atrasar e, neste caso, trocar o pneu poderia levar muito tempo, seria melhor remover do plano a ação de dirigir o carro e substituí-la por ações que utilizem um táxi. Nesse caso, algumas ações do plano seriam removidas e outras, adicionadas.*

Portanto, para reparar um plano, um planejador não deve apenas empregar uma estratégia de refinamento com o intuito de ampliar o plano por meio de ações que atingirão os objetivos. O planejador tem, também, que empregar uma estratégia de *refinamento reverso* para diminuir as restrições do plano parcial (removendo ações do plano que estejam obstruindo uma solução).

No artigo [van der Krogt and de Weerdt, 2005] os autores propõem uma ampliação do modelo do planejamento por refinamento que permite que estratégias de refinamento reverso sejam empregadas. A proposta original de Roman Krogt e Mathijs M. Weerdt pode ser melhor compreendida através da Figura 5.1.

Uma versão adaptada desta proposta de ampliação pode ser vista no Algoritmo 5. Dado um plano π, parcialmente executado, o estado em que o plano apresentou falha, o domínio e a meta do problema, inicialmente, escolhe-se *refinar* o plano, isto é, adicionar refinamentos. Se não for encontrada uma solução, então tenta-se *remover refinamentos* do plano. Para executar o modelo de refinamento reverso no plano, é necessário selecionar uma heurística de refinamento reverso e aplicá-la ao plano parcial. O refinamento ocorre no trecho do algoritmo referente ao planejamento normal.

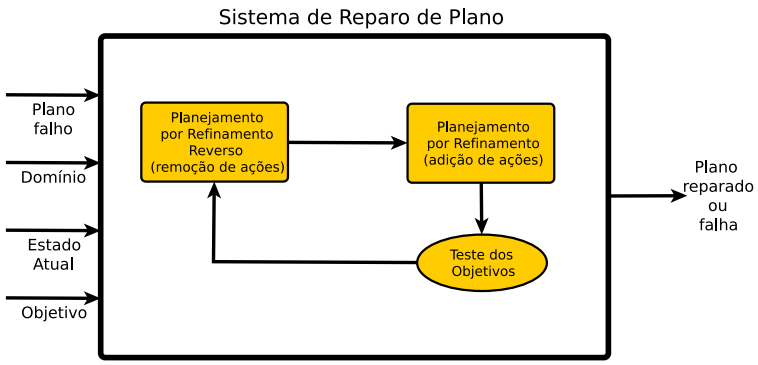

Figura 5.1: Arquitetura de funcionamento do sistema de reparo de plano.

O Algoritmo 5 funciona da seguinte forma: inicialmente ele atribui o estado em que a execução do plano original deveria estar caso não tivesse ocorrido a falha, a uma variável (e_g). A seguir, o algoritmo chama um método de planejamento que tenta encontrar um plano que leve a execução do estado atual s ao estado desejado e_g, atribuindo o resultado a π_r. Na próxima fase, os comandos descritos nas linhas 4 e 5 verificam se o resultado π_r é diferente de falha. Caso seja, o algoritmo devolve o reparo de plano encontrado pelo planejador concatenado ao plano original. Isto significa que somente a adição de novas ações conseguiu reparar o plano falho.

Entretanto, caso o resultado de π_r retorne uma falha, um método heurístico, que recebe como entrada o estado atual s e o plano original π, é chamado. Este método devolve como resposta uma estrutura de dados composta por uma lista de pares de `estado` e `quantidade`, que indica quais conjuntos de ações do plano original devem ser considerados para a remoção. A proposta é fazer com que a execução do plano retorne ao `estado` do plano original após a remoção da `quantidade` necessária de ações. Ocorre, então, um processo de repetição, onde cada par de `estado` e `quantidade` é testado a fim de que o plano saia do estado atual s e consiga alcançar o estado e_g, atribuindo o resultado a π_r.

Os próximos comandos verificam se o resultado π_r é diferente de falha. Em caso afirmativo, remove-se do plano original a `quantidade` de ações. Em

seguida o algoritmo devolve o reparo de plano encontrado pelo planejador concatenado ao plano original. Portanto, neste caso, para reparar o plano foi necessária a remoção de um conjunto de ações do plano original e a adição de um conjunto de novas ações. Se nenhum dos pares for capaz de devolver uma resposta diferente de falha, então um replanejamento é feito. O método de planejamento tenta encontrar um plano que saia do estado atual s e consiga alcançar o objetivo S_g, atribuindo o resultado a π_r. Mais uma vez o algoritmo verifica se o resultado π_r é diferente de falha. Caso seja, o algoritmo devolve o plano encontrado pelo planejador.

Uma versão do Algoritmo 5 sem heurística pode ser visualizada no Algoritmo 4.

Algoritmo 4: Algoritmo de Reparo de Plano sem Heurística.

Entrada: Trecho do plano original ainda não executado π, Estado atual s, Domínio $\mathcal{D}$, Objetivo S_g
Saída: Plano reparado π ou *falha*

```
1  início
2      e_g ← recuperaEstado(π);
3      π_r ← planejador(s, e_g, D);
4      se π_r ≠ falha então
5          devolve π_r + π;
6      senão
7          para todos os conjuntos de ações seqüenciais c_a de π faça
8              se π ≠ c_a então
9                  e_g ← recuperaEstado(π − c_a);
10                 π_r ← planejador(s, e_g, D);
11                 se π_r ≠ falha então
12                     π ← π − c_a;
13                     devolve π_r + π;
14     π_r ← planejador(s, S_g, D);
15     se π_r ≠ falha então
16         devolve π_r;
17     devolve falha;
18 fim
```

Algoritmo 5: Algoritmo de Reparo de Plano.

Entrada: Trecho do plano original ainda não executado π, Estado atual s, Domínio $\mathcal{D}$, Objetivo S_g
Saída: Plano reparado π ou *falha*

```
1  início
2      e_g ← recuperaEstado(π);
3      π_r ← planejador(s, e_g, D);
4      se π_r ≠ falha então
5          devolve π_r + π;
6      senão
7          lista[] ← heurística(s, π);
8          repita
9              e_g ← lista.primeiroElemento().estado;
10             π_r ← planejador(s, e_g, D);
11             se π_r ≠ falha então
12                 π ← π.remove(lista.primeiroElemento().quantidade);
13                 devolve π_r + π;
14             removePrimeiroElemento(lista[]);
15         até lista[] estiver vazia ;
16     π_r ← planejador(s, S_g, D);
17     se π_r ≠ falha então
18         devolve π_r;
19     devolve falha;
20 fim
```

5.2 Heurística

A forma mais comum de se adicionar conhecimento a respeito de um problema ao algoritmo de busca é por meio de funções heurísticas. Uma função heurística pode ser denotada por $h(n)$:

> $h(n)$ = custo estimado do caminho mais econômico do nó n até um nó objetivo.

Se n é um nó objetivo, então $h(n) = 0$.

Uma heurística é *admissível* quando $h(n)$ não superestima o custo para alcançar o objetivo [Russell and Norvig, 2002]. Além disso, ela é otimista por natureza, pois pressupõe que o custo da solução do problema é menor ou igual ao seu custo real. Isto ocorre porque a heurística ajuda a descartar os nós de maior custo, assegurando que o caminho ótimo seja sempre o primeiro a ser escolhido.

Um modo de pensar sobre o problema da criação de heurísticas é ter como base um *problema relaxado*, no qual um problema é encarado com menos restrições. O custo de uma solução ótima para um problema relaxado é uma heurística [van der Krogt and de Weerdt, 2005] admissível para o problema original [Russell and Norvig, 2002].

A idéia básica da construção de uma heurística para o planejamento reverso é observar os efeitos das ações e os objetivos que devem ser alcançados, e avaliar quantas ações serão necessárias para alcançar todos os objetivos.

Uma sugestão inicial é relaxar o problema removendo todas as pré-condições das ações. Então, toda ação será aplicável e qualquer literal poderá ser alcançado em um passo.

Uma heurística mais interessante seria, além de eliminar todas as pré-condições: (1) eliminar os efeitos negativos das ações e (2) considerar as

interações positivas entre ações. Isto é, se uma ação tem o efeito $X \wedge \neg Y$ no problema original, terá somente o efeito X no problema relaxado, ou seja, nenhuma ação pode excluir os literais alcançados por outro e, portanto, não é preciso se preocupar com interações negativas entre subplanos. Para que a união dos efeitos positivos dessas ações satisfaça o objetivo, calcula-se o número mínimo de ações necessárias. Por exemplo, considere o objetivo (X, Y, Z) para o domínio descrito pelas as seguintes ações:

ação(α, EFEITO: $X \wedge Q$)
ação(β, EFEITO: $Y \wedge Z \wedge R$)
ação(δ, EFEITO: $Y \wedge Q \wedge R$)

O conjunto de objetivos $\{X, Y, Z\}$ pode ser alcançado pelas ações $\{\alpha, \beta\}$ sendo a heurística igual a 2, considerando custos unitários. É possível notar que a ação δ não é considerada, pois ela adiciona Y, que já é incluído pela ação β (interação positiva).

A estratégia utilizada neste projeto foi gerar problemas relaxados eliminando-se tanto as pré-condições negativas quanto os efeitos negativos e ainda, considerando uma ordem total entre as ações do plano para o cálculo da heurística.

A abordagem para o uso destas heurísticas de planejamento para reparo de planos é ilustrada na Figura 5.2. De um lado, temos o plano corrente P que não será refinado. É computado, então, o número de planos que resulta da remoção de ações de P. Para cada um destes planos resultantes, utiliza-se uma heurística para estimar a quantidade de trabalho necessária para transformar este plano em um plano válido, isto é, calcula-se um valor heurístico para cada um destes planos. O plano que obtiver o melhor (mais baixo) valor heurístico é selecionado e o planejamento (refinamento) é utilizado para completar este plano. Se o planejador não puder produzir uma solução (o que pode acontecer porque a heurística não é perfeita), um outro refinamento reverso é escolhido. É importante notar que somente se aplica o passo do refinamento reverso ao plano inicial P.

Um outro ponto de vista importante sobre este procedimento que deve ser levado em consideração é o espaço de busca atravessado. Inicialmente, o método de reparo de plano fornece o plano corrente P para ser adaptado. Este plano pode estar localizado em uma parte do espaço de busca em que é muito difícil encontrar uma solução por refinamento (isto é, adicionando apenas ações), se tal solução existir. A heurística de refinamento reverso avalia as condições do espaço de busca ao redor destes planos parciais, isto é, calcula um valor heurístico que expressa a facilidade com que o plano pode ser ampliado. Após ser identificada a melhor opção, é iniciado o processo de refinamento.

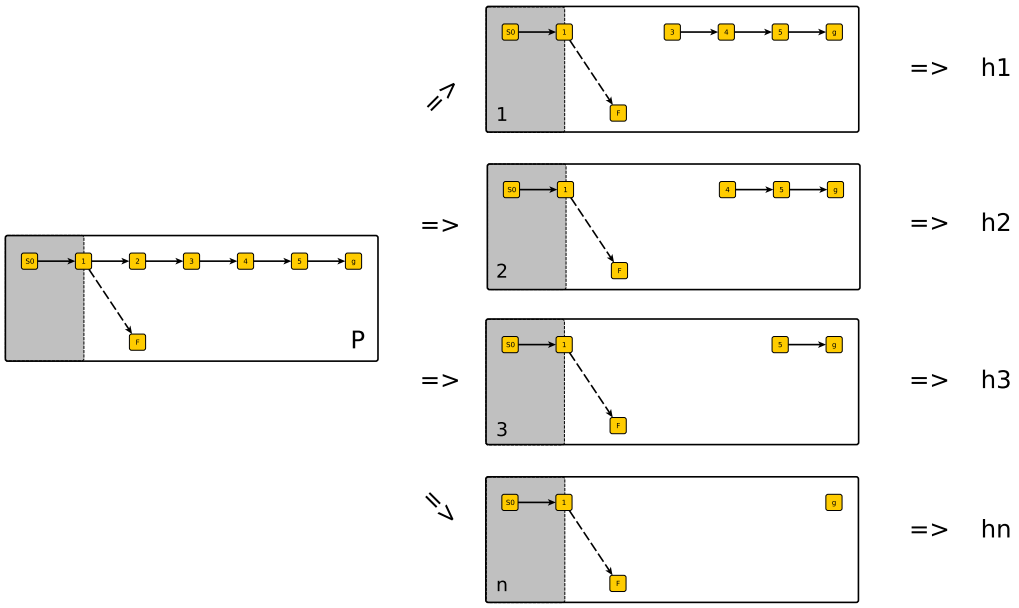

Figura 5.2: Heurística de refinamento reverso. Do plano original à esquerda foram derivados n subplanos e calculados seus respectivos valores heurísticos $(h_1, \ldots, h_n)$. A região cinza representa a parte do plano que já foi executada.

A heurística só é utilizada nos casos em que o refinamento de plano falha ao tentar encontrar um reparo. Seu objetivo é avaliar qual é o melhor conjunto de ações a serem removidas, de modo que, a partir do novo estado inicial, possa ser encontrado um reparo, por meio de refinamento, que satisfaça o

novo estado-objetivo.

A heurítica desenvolvida neste trabalho é baseada na remoção de conjuntos de ações seqüenciais e crescentes. Os conjuntos de ações removidas a serem avaliados começam com a primeira ação não executada. A partir daí, cada novo conjunto é incrementado até que todas as ações tenham sido adicionadas. Inicia-se com uma ação, depois duas e assim sucessivamente até que todas as ações sejam incorporadas. Para cada conjunto de ações removidas um novo estado é gerado. Este estado contém todas as pré-condições necessárias para executar todas as demais ações (não removidas) até o final do plano, como pode ser observado na Figura 5.3.

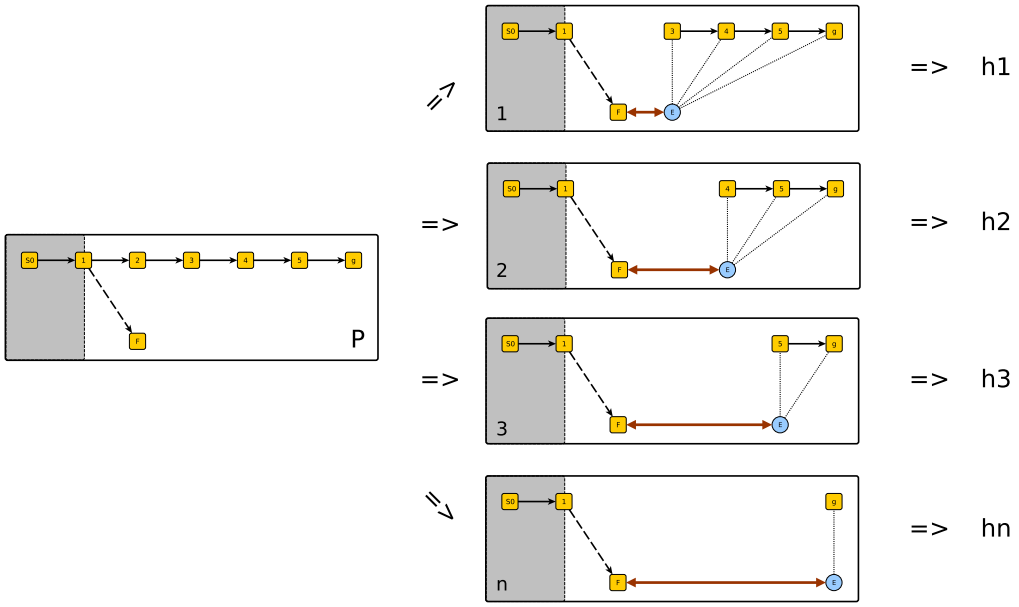

Figura 5.3: Para cada conjunto de ações removidas, um estado novo é gerado. Este estado contém todas as pré-condições necessárias para todas as ações não removidas. A heurística calcula a distância entre o estado atual F e o estado novo E. A região cinza representa a parte do plano que já foi executada.

O Algoritmo 6 tem como objetivo devolver uma lista de pares de `estado` e `valor` e funciona da seguinte forma: para todos os conjuntos de ações

possíveis de serem removidas, um estado com as condições necessárias para executar o restante das ações (não removidas) do plano original é gerado (E_g). O processo é iniciado, marcando-se como visitado o estado atual (s), e este é inserido na fila s com o valor igual a 0. Este valor serve para indicar a distância entre o estado atual e o estado inserido na fila. A partir da linha 6 até a linha 19, para cada elemento f da $fila$, verifica se o mesmo está contido em E_g. Se estiver, insere na lista de $resultados$, e o algoritmo devolve $resultados$ se tiver E_g contido. Seqüencialmente, entre as linhas 12 e 18, para todas as ações a aplicáveis ao estado $f.estado$, atribui a s_{novo} o estado resultante após a execução da ação (linha 15). Sendo que esta execução não considera os efeitos negativos da ação. Verifica se o estado s_{novo} não está marcado como visitado. Caso não esteja, marca como visitado e insere na $fila$ com o valor de $f.valor$ incrementado em um. Deste modo a estrutura da fila terá um conjunto de estados e valores que indicam a distância em relação ao estado atual s. (linha 16 a 18). Sendo que o conjunto de elementos devolvidos pelo algoritmo estará ordenado de acordo com o *valor* heurístico atribuído.

Algoritmo 6: Heurística para Refinamento Reverso

Entrada: Estado Atual s, Plano Original π
Saída: Lista de pares de *Estado e valor*

```
1  início
2      E_g ← geraListaEstados(π);
3
4      visitado(s);
5      fila.insere(0, s);
6      repita
7          f ← fila.retiraInício();
8          se f.estado ⊂ E_g.estado então
9              resultado.insere(f);
10             se E_g ⊆ resultado então
11                 devolve resultado;
12         para toda ação a aplicável ao estado f.estado faça
13             // Executa a ação a no estado f.estado
14             // sem considerar efeitos negativo
15             s_novo ← executa(f.estado, a);
16             se não visitado(s_novo) então
17                 visitado(s_novo);
18                 fila.insere(f.valor +1, s_novo);
19     até fila estiver vazia ;
20     devolve resultado;
21 fim
```

Capítulo 6

Implementação e análise experimental

Como foi dito no Capítulo 5, não existe uma implementação disponível de um sistema de reparo de plano por refinamento reverso. Assim, um dos objetivos deste trabalho é implementar um sistema de reparo desse tipo para disponibilizá-lo, bem como para verificar empiricamente as situações em que o reparo de plano pode ser mais vantajoso do que o replanejamento e como o uso da heurística pode melhorar sua eficiência.

6.1 Sistema de reparo de plano

O sistema de reparo de plano por refinamento reverso implementado neste trabalho utiliza como base o VHPOP, um planejador de código fonte aberto que implementa de forma explícita o planejamento por refinamento. O VHPOP foi desenvolvido por Håkan Younes e R.G. Simmons na Universidade Carneguie Mellon em 2003 [Younes and Simmons, 2003] em C++. Originalmente, o planejador VHPOP recebe como entrada três conjuntos de dados: o domínio, o estado inicial e o objetivo. Todos os dados de entrada estão no formato PDDL. Ao executar o sistema, o planejador produz, se existir, um plano resposta para o problema (Figura 6.1).

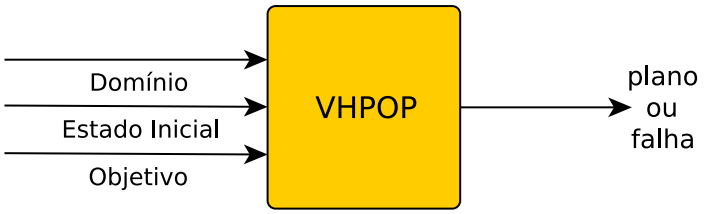

Figura 6.1: Arquitetura de funcionamento do VHPOP.

O sistema de reparo implementado (Figura 6.2) recebe como entrada:

- um plano falho, com indicação das ações já executadas;
- uma descrição do domínio (conjunto de ações do agente);
- o estado atual da simulação;
- uma descrição dos estados objetivos.

Tal sistema devolve como resposta uma falha, se um reparo não for encontrado, caso contrário, uma resposta válida será uma seqüência de ações que atinja o objetivo do problema.

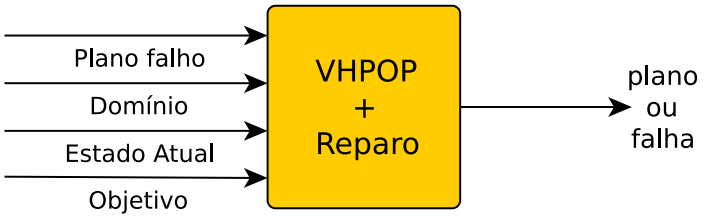

Figura 6.2: Sistema VHPOP-RE.

O sistema de reparo pode ser executado com ou sem o uso de heurística. O sistema sem o uso de heurística (Algoritmo 4 do Capítulo 5) chamado de VHPOP-RE, testa todas as alternativas possíveis para encontrar a solução ótima de reparo. A versão com heurística (Algoritmo 6 do Capítulo 5), chamada de VHPOP-RE-H, tenta encontrar um reparo fazendo o menor número de tentativas. Com isso, espera-se que esse seja um método mais

eficiente de reparo de planos.

Os sistemas VHPOP-RE e VHPOP-RE-H foram implementados de forma a encapsular completamente o planejador VHPOP (Figura 6.3): a idéia é que após remover ações do plano por refinamento reverso, esses sistemas usem o próprio VHPOP para completar o plano.

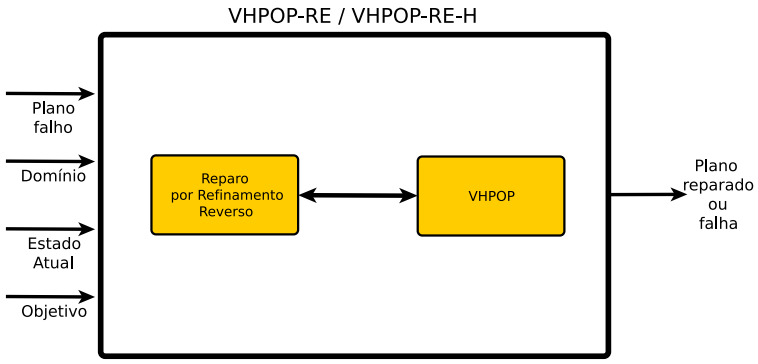

Figura 6.3: Arquitetura de funcionamento do VHPOP-RE.

6.2 Simulador da dinâmica de ambientes de teste

Além da implementação do VHPOP-RE, foi desenvolvido um sistema de simulação da dinâmica de ambientes de planejamento para testar o sistema de reparo, chamado de SIMULA-PLANO. Este sistema tem como objetivo simular a execução de planos em diferentes domínios, com a possibilidade de simular falhas durante a execução de um plano.

O simulador recebe como entrada:

- uma descrição do domínio (conjunto de ações do agente);
- o estado inicial do ambiente;

- um plano para atingir o objetivo do agente;
- um conjunto de falhas.

O conjunto de falhas pode ser informado ao simulador de duas maneiras diferentes: (1) pelo programa de geração de falhas automáticas Seção 6.3, ou (2) pela intervenção de um usuário.

Durante a execução de um plano, o simulador verifica as pré-condições de todas as ações a serem executadas. Ao detectar que uma ou mais pré-condições não são satisfeitas, o simulador chama o sistema VHPOP-RE.

Com o intuito de facilitar a visualização, foi desenvolvida uma interface Web que permite observar a execução das ações do plano com a ocorrência de falhas e as ações de reparo, bem como permite que o usuário interaja com o simulador para a geração de falhas. A Figura 6.4 apresenta uma imagem da interface Web durante a execução de um plano no simulador. É possível visualizar a ocorrência de uma falha que inviabiliza as pré-condições da ação `A1`, uma vez que o estado passou do estado `S1` ao `SR0`. A partir deste ponto um reparo de plano é executado (ações `AR0`, `AR1` e `AR2`) até retomar o plano original a partir da ação `A2`.

6.3 O programa de geração de falhas

Para que os testes pudessem ser realizados foi criado um sistema de geração de falhas independente de domínio. Criar uma alteração específica significa simplesmente invalidar uma pré-condição e ainda manter um estado consistente. Por exemplo, no domínio do Mundo dos Blocos, ao tentar desempilhar um bloco, uma das pré-condições é que o bloco esteja livre, uma alteração específica colocaria um outro bloco sobre o bloco que se quer desempilhar, para cria uma falha. Um dos cuidados que se deve ter é de manter a consistência dos estados: simplesmente fazer com que uma pré-condição se torne falsa, pode gerar estados inconsistentes. Por exemplo, se o $Bloco_A$ estiver sobre o $Bloco_B$, não se pode adicionar a condição $Livre(Bloco_B)$,

sem adicionar a condição $Sobre(Bloco_A, mesa)$ ou $Sobre(Bloco_A, Bloco_C)$, isto é, o $Bloco_A$ deve estar em outro lugar que não seja sobre o $Bloco_B$.

Sendo assim, a alternativa para manter estados conscientes é executar ações exógenas[1] cujos efeitos inviabilizem ao menos uma das pré-condições da ação. O programa de geração de falhas realiza os seguintes passos:

1. sorteia (ou seleciona) uma ação a_f para falhar;
2. seleciona uma ou mais pré-condições de a_f que serão negadas, $p \subseteq$ pré-condições (a_f);
3. chama o VHPOP, o qual recebe como entrada o subconjunto p, o estado s anterior à execução de a_f, o domínio; e devolve como saída um plano de ações exógenas;
4. solicita ao simulador que execute o plano de ações exógenas imediatamente antes da execução de a_f.

6.4 A arquitetura do sistema

Os sistemas VHPOP-RE, VHPOP-RE-H e o simulador foram implementados na linguagem de programação C++, gerando aproximadamente 5300 linhas de código. A interface Web do simulador foi implementada utilizando a linguagem de programação Perl.

A Figura 6.5 ilustra a arquitetura e dados trocados entre cada segmento do sistema. Mais detalhes da implementação podem ser vistos no Apêndice B, o qual mostra o Diagrama de Classe em UML.

[1] Ações exógenas podem ser as mesmas ações do domínio, porém, elas são supostamente executadas por outro agente que tem como objetivo invalidar os planos do agente principal e por isso podem também ser chamadas de ações exógenas.

6.5 Domínios de teste

Os domínios de teste utilizados nesta dissertação são instâncias dos domínios: Mundo dos Blocos Coloridos e Controle de Satélites.

6.5.1 Mundo dos Blocos Coloridos

O domínio do Mundo dos Blocos Coloridos é uma variação do domínio do Mundo dos Blocos [Winston, 1992] que foi definido nessa disseratação para testar problemas interessantes de reparo de planos.

A diferença entre esses dois domínios é que no Mundo dos Blocos Coloridos, além de cada bloco possuir uma identificação única, ele possui uma cor pré-determinada. Ao empilhar um bloco sobre o outro, o bloco superior assume a cor do bloco inferior e, mesmo que os blocos sejam separados, a modificação de cor permanecerá até que o bloco seja disposto em cima de um outro bloco de cor diferente e uma nova alteração ocorra. Um exemplo deste domínio pode ser visto na Figura 6.6, e sua descrição encontra-se na Tabela 6.1.

Tabela 6.1. *Descrição das ações no domínio do Mundo dos Blocos Coloridos*

Ação	***Descrição***
pick-up(?x)	
precondição:	(and (clear ?x) (ontable ?x) (handempty))
efeito:	(and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x))
put-down(?x)	
precondição:	(holding ?x)
efeito:	(and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x))
stack-amarelo(?x ?y)	
precondição:	(and (holding ?x) (clear ?y) (amarelo ?y))
efeito:	(and (not (holding ?x)) (not (clear ?y)) (clear ?x) (handempty) (on ?x ?y) (amarelo ?x) (not (azul ?x)) (not (vermelho ?x)))
stack-azul(?x ?y)	
precondição:	(and (holding ?x) (clear ?y) (azul ?y))
efeito:	(and (not (holding ?x)) (not (clear ?y)) (clear ?x) (handempty) (on ?x ?y) (not (amarelo ?x)) (azul ?x) (not (vermelho ?x)))
stack-vermelho(?x ?y)	
precondição:	(and (holding ?x) (clear ?y) (vermelho ?y))
efeito:	(and (not (holding ?x)) (not (clear ?y)) (clear ?x) (handempty) (on ?x ?y) (not (amarelo ?x)) (not (azul ?x)) (vermelho ?x))
unstack(?x ?y)	
precondição:	(and (on ?x ?y) (clear ?x) (handempty))
efeito:	(and (holding ?x) (clear ?y) (not (clear ?x)) (not (handempty)) (not (on ?x ?y)))

Este domínio oferece a possibilidade de testar, de modo claro e eficiente, o comportamento do sistema durante a ocorrência de falhas em que somente o refinamento (adição de ações) não é suficiente para reparar o plano, exi-

gindo assim, a execução de refinamento reverso.

Exemplo 6.5.1. *Deseja-se criar uma pilha de três blocos onde todos sejam de cor amarela ou azul. O estado inicial consiste em três blocos, dispostos livremente sobre a mesa e suas cores são: amarelo, azul e vermelho. O plano original é colocar o bloco vermelho sobre o azul e o amarelo sobre os dois. Entretanto, um evento exógeno poderia fazer com que o bloco azul fosse disposto sobre o vermelho, inviabilizando a execução do plano original. Para solucionar este problema seria necessário executar um sistema de reparo de plano que utiliza refinamento reverso, pois é preciso remover ações do plano original para repará-lo*

6.5.2 Controle de Satélites

A busca por soluções de automatização aplicada às operações espaciais é uma necessidade mundial para diminuir os custos das missões. Encontrar caminhos de automatização para as atividades que envolvem operação de satélites é de extrema importância para se conseguir manter um bom desempenho, mesmo com a escassez de recursos disponíveis [Cardoso, 2006] [Cardoso et al., 2006]. Essa é uma das aplicações práticas para as quais se tem usado planejadores automáticos.

O domínio de Controle de Satélites [McDermott, 2000] [Bacchus, 2001] envolve um conjunto de satélites, e seus respectivos instrumentos de coleta de dados e configurações específicas. Os instrumentos fazem parte de satélites específicos, e cada instrumento pode ter diferentes funcionalidades como, por exemplo, um infravermelho ou uma câmera fotográfica. O satélite pode ser posicionado em direções específicas, e os instrumentos calibrados para atuarem na mesma direção em que aponta o satélite. Sendo assim, um satélite posicionado na direção de um corpo celeste pode ter seu infravermelho calibrado para tirar uma fotografia deste astro. A descrição das ações do domínio pode ser obeservada na Tabela 6.4.

Entretanto, para utilizar um instrumento de coleta de dados é necessário

que este esteja ligado no sistema de energia do satélite e esteja calibrado. Toda vez que um instrumento é desligado e ligado novamente, ele é descalibrado.

Tabela 6.2. *Descrição das ações no domínio de Controle de Satélites*

Ação	***Descrição***
`turn_to(?s ?d_new ?d_prev)`	
`precondição:`	`(pointing ?s ?d_prev)`
`efeito:`	`(and (pointing ?s ?d_new)`
	`(not (pointing ?s ?d_prev)))`
`switch_on(?i ?s)`	
`precondição:`	`(and (on_board ?i ?s)`
	`(power_avail ?s)`
`efeito:`	`(and (power_on ?i) (not (calibrated ?i))`
	`(not (power_avail ?s)))`
`switch_off(?i ?s)`	
`precondição:`	`(and (on_board ?i ?s) (power_on ?i) )`
`efeito:`	`(and (not (power_on ?i)) (power_avail ?s))`
`calibrate(?s ?d ?i ?m)`	
`precondição:`	`(and (on_board ?i ?s) (calibration_target ?i ?d)`
	`(pointing ?s ?d) (power_on ?i)`
`efeito:`	`(calibrated ?i)`
`take_image(?s ?d ?i ?m)`	
`precondição:`	`(and (calibrated ?i) (on_board ?i ?s)`
	`(supports ?i ?m) (power_on ?i)`
	`(pointing ?s ?d) (power_on ?i))`
`efeito:`	`(have_image ?d ?m))`

O objetivo é encontrar um plano que faça a coleta de dados, otimizando a utilização dos recursos do satélite como, por exemplo, tirar fotos com diferentes câmeras e infravermelhos de diversos corpos celestes, da forma mais eficiente possível.

O domínio de Controle de Satélites permite testar o comportamento do sis-

tema durante a ocorrência de falhas que precisa da remoção de ações para reparar o plano, pois um instrumento pode se tornar inapto para utilização.

6.6 Análise experimental

Os experimentos foram executados em um computador Dell, com processador Intel 6400 Core 2 Duo com clock de 2.13 GHz, 2 GB de memória e sistema operacional Linux Ubuntu 7.10 (*Gutsy Gibbon*). O código-fonte foi compilado com GCC 4.1 com o *flag* `-O2` habilitado.

Foram comparados os sistemas VHPOP-RE, VHPOP-RE-H (versão com heurística) e VHPOP (usado para replanejamento a partir do estado em que ocorre uma falha). Realizou-se dois tipos de análise experimental:

1 Análise do tempo médio de execução; e

2 Análise de aproveitamento do plano original mantido após o reparo.

Os resultados das análises referentes ao domínio do Mundo dos Blocos Coloridos e Controle de Satélites podem ser observados nas Figuras 6.7 e 6.8. Os problemas foram divididos em diferentes categorias, conforme a descrição das Tabelas 6.3 e 6.4.

Tabela 6.3. *Descrição da natureza dos problemas utilizados para teste do domínio de Mundo dos Blocos Coloridos*

Falha	Descrição	Objetivo
1 – 8	*3 blocos coloridos*	*construir 1 pilha de blocos*
9 – 16	*6 blocos coloridos*	*construir 1 pilha de blocos*
17 – 24	*6 blocos coloridos*	*construir 3 pilhas de blocos*

Tabela 6.4. *Descrição da natureza dos problemas utilizados para testes do domínio de Controle de Satélites*

Falha	Descrição
1 − 6	*1 satélite - 2 instrumentos - 2 aparelhos*
7 − 12	*2 satélites - 1 instrumento - 2 aparelhos*
13 − 18	*2 satélites - 2 instrumentos - 1 aparelho*

6.6.1 Análise de tempo

Observa-se nas Figuras 6.7 e 6.8 que, em quase todos os pontos do gráfico, o sistema de reparo de plano utilizando heurística é mais eficiente do que o reparo de plano sem nenhuma heurística, VHPOP-RE, ou mesmo o replanejamento completo. Isto se deve principalmente ao fato de que, mesmo após ocorrer uma falha, grande parte do plano original geralmente ainda é válida. Sendo que, com a utilização de uma heurística, é possível utilizar a parte ainda válida do plano sem desperdiçar muito processamento na busca, ou mesmo descartando e refazendo todo o plano com um replanejamento completo.

Entretanto, ainda nas Figuras 6.7 e 6.8, observa-se que em alguns casos, o sistema VHPOP-RE-H consome mais tempo para encontrar a solução do que o VHPOP. Isto se dá porque no pior caso, o VHPOP-RE-H testa um conjunto de possibilidades antes de descartar completamente o plano e construir um novo (como na falha 22 da Figura 6.7, ou nas falhas 3 e 6 da Figura 6.8).

6.6.2 Análise de aproveitamento do plano

A Figura 6.9 apresenta um gráfico onde nota-se que o número de ações do plano original mantido pelo reparo de plano com heurística é bem superior ao replanejamento completo, pois ao detectar que ocorreu uma falha, o sistema de replanejamento descarta todo o plano e gera um novo, enquanto

o sistema de reparo de plano desenvolvido neste trabalho tenta encontrar, por meio da heurística, um reparo que o leve de volta ao plano original. No pior dos casos o VHPOP-RE-H testa todas as possibilidades e encontra a mesma solução que o sistema VHPOP, ou seja, remove todas as ações do plano original, como pode ser observado nas Figuras 6.9 e 6.11, falhas 3 e 6. Comparando o sistema de reparo com heurística ao sem, observa-se que o algoritmo sem heurística mantém um maior número de ações do plano original. Isto ocorre porque a heurística utilizada não é perfeita, enquanto o algoritmo implementado no sistema de reparo sem heurística testa todas possibilidades de retorno ao plano original.

Em alguns casos, entretanto, é possível que o VHPOP-RE-H adicione um maior número de ações do que um replanejamento completo, pois sua prioridade é manter um compromisso com o plano original.

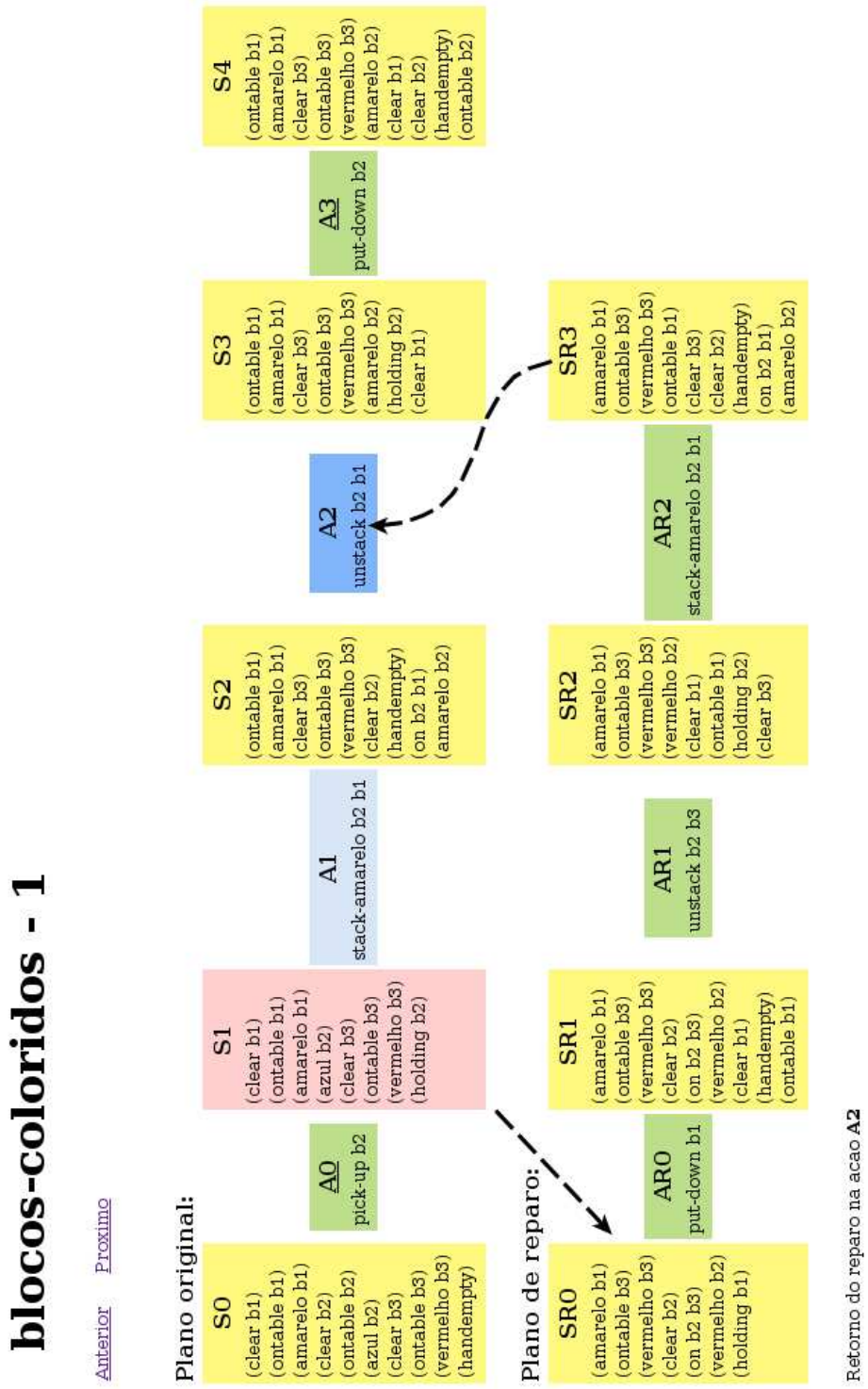

Figura 6.4: O plano original é mostrado acima do reparo. Ações já executadas do plano original aparecem com nomes sublinhados. Ações removidas do plano original são ilustradas por retângulos de cor cinza claro. As setas sobrepostas à foto da tela indicam em que ponto do plano original ocorreu a falha, e qual é o ponto da retorno após a execução do reparo.

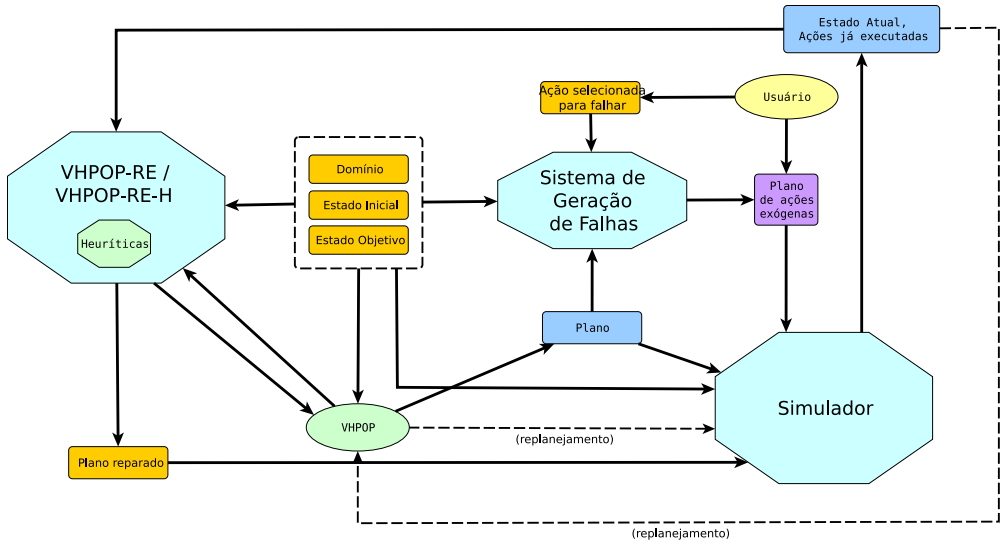

Figura 6.5: Arquitetura do sistema de reparo de plano por refinamento reverso. Os octágonos representam os sistemas implementados nesse trabalho.

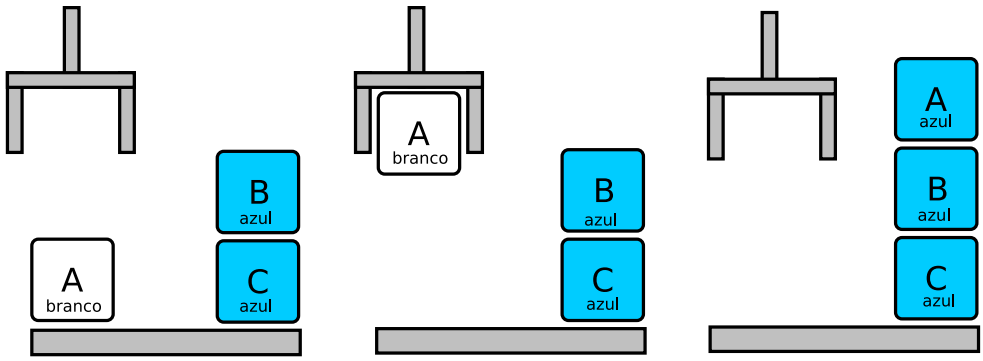

Figura 6.6: Este exemplo mostra o comportamento da ação empilhar no Mundo dos Blocos Coloridos. Ao empilhar o bloco branco A sobre o bloco azul B, o bloco A muda da cor branca para a cor azul.

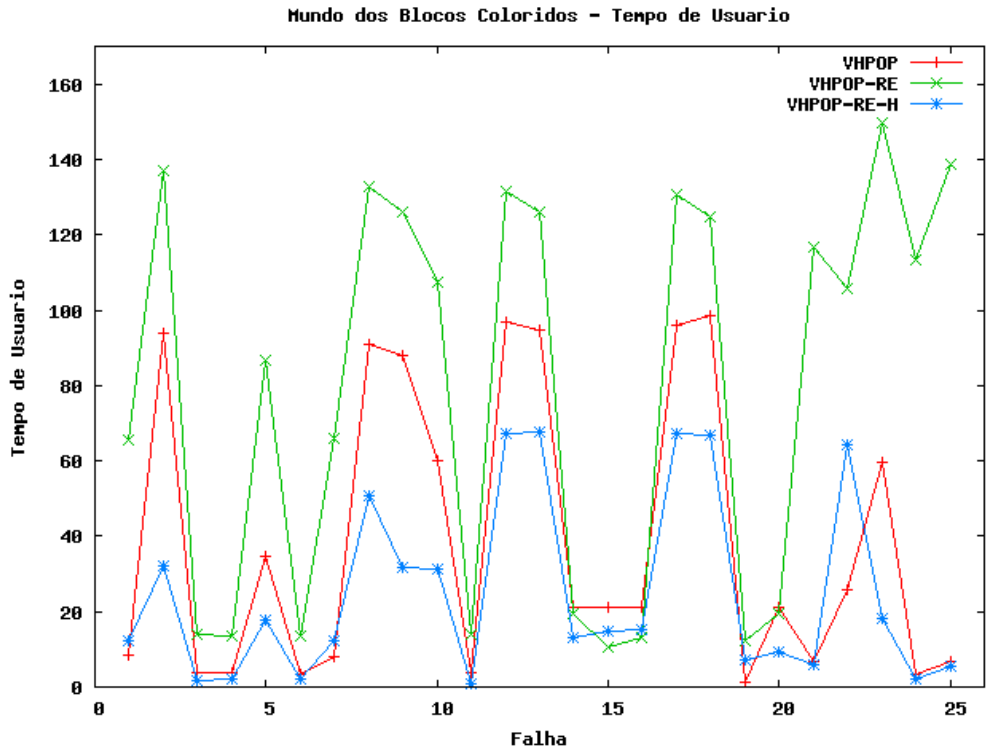

Figura 6.7: Tempo médio de execução para o domínio do Mundo dos Blocos Coloridos.

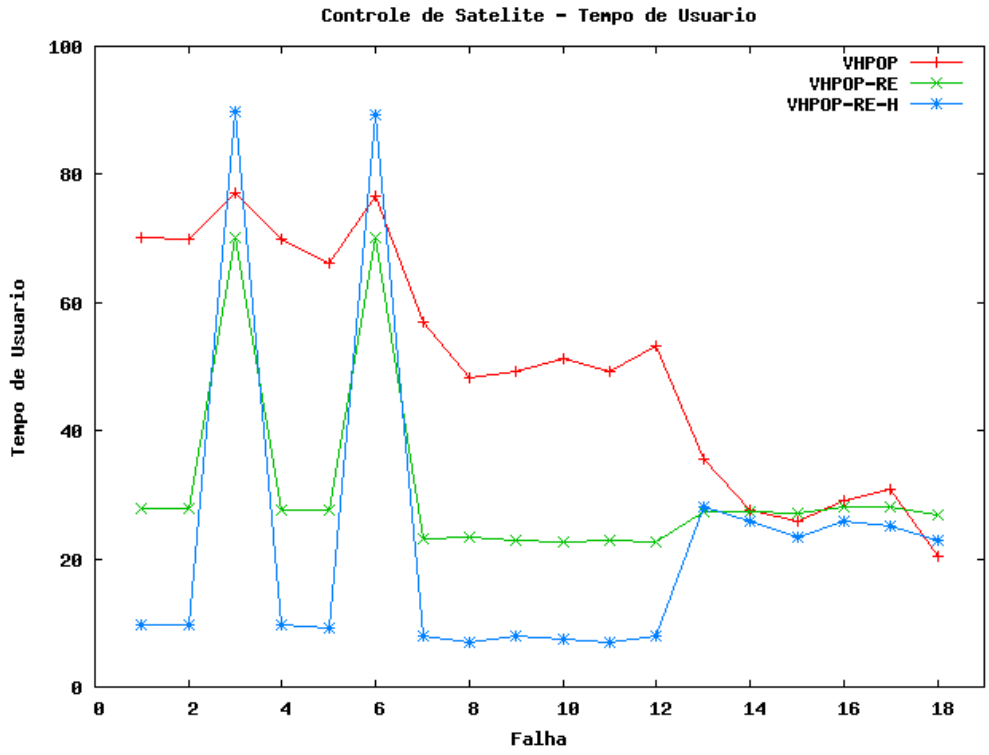

Figura 6.8: Tempo médio de execução para o domínio de Controle de Satélites.

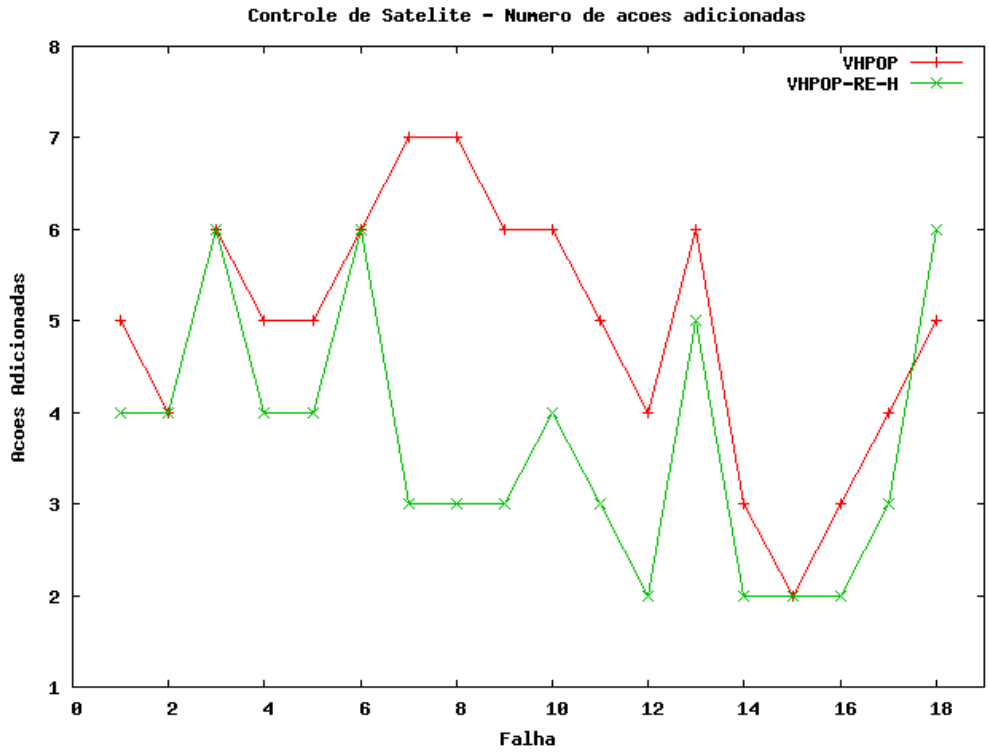

Figura 6.9: úmero de ações adicionadas após a execução do reparo no domínio de Controle de Satelites.

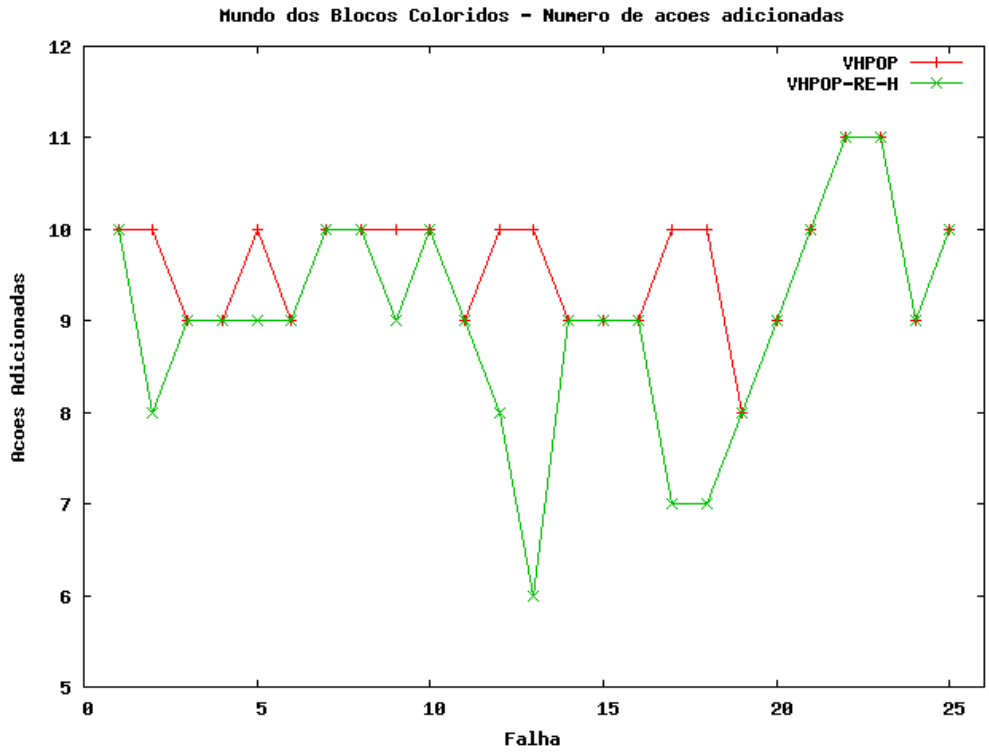

Figura 6.10: Número de ações adicionadas após a execução do reparo no domínio do Mundo dos Blocos Coloridos.

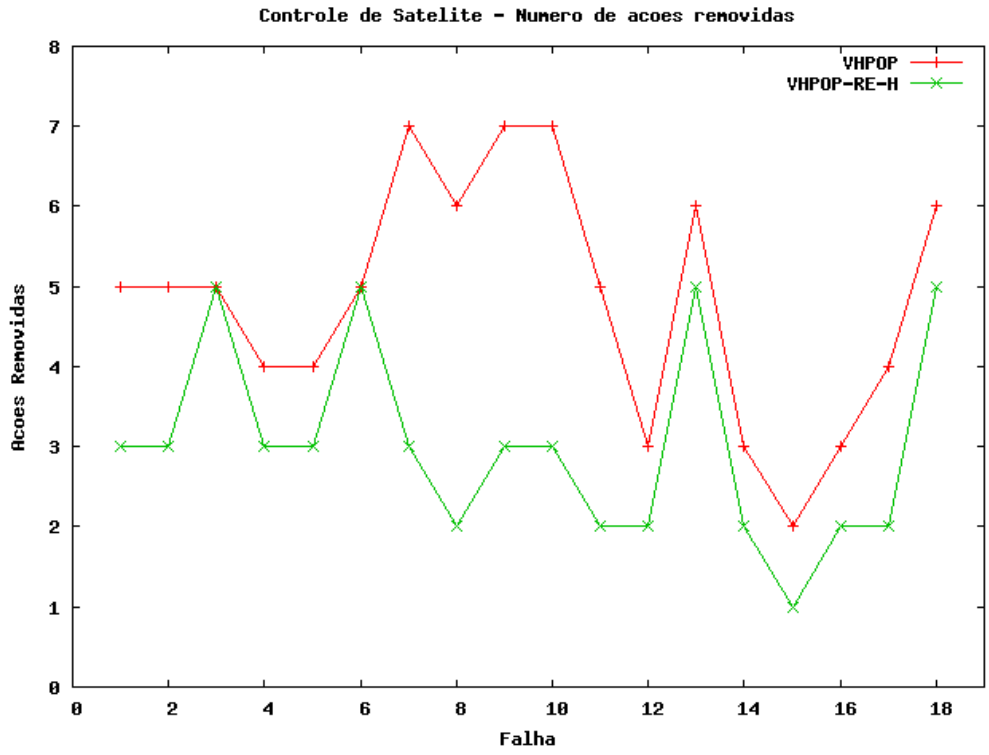

Figura 6.11: Número de ações removidas após a execução do reparo no domínio de Controle de Satelites.

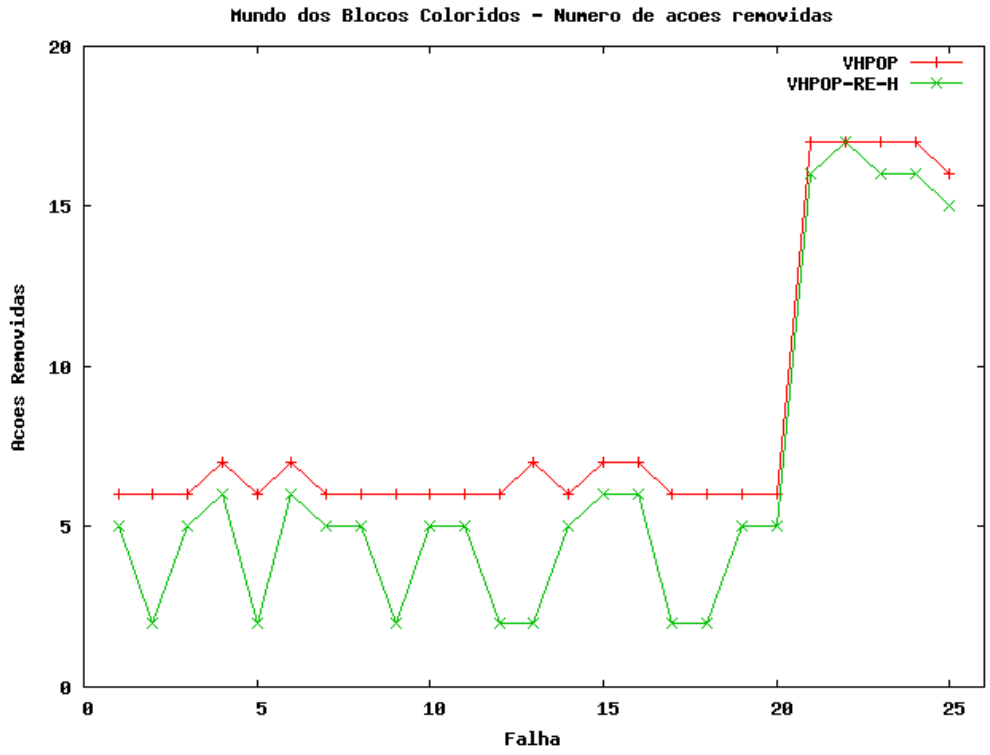

Figura 6.12: Número de ações removidas após a execução do reparo no domínio do Mundo dos Blocos Coloridos.

Capítulo 7

Conclusão

Nos últimos anos, planejamento automatizado vem sendo cada vez mais aplicado em problemas práticos de diversas áreas que requerem soluções confiáveis. Além disso, um plano pode falhar durante sua execução devido à interferências de outros agentes (eventos exógenos).

Neste trabalho foram estudadas diferentes abordagens [Cushing and Kabhampati, 2005] [van der Krogt and de Weerdt, 2004b] [van der Krogt and de Weerdt, 2005] para tratar planejamento não-determinístico, por meio de monitoração de execução e reparo de planos.

No reparo de planos faz-se a suposição de que o custo envolvido no replanejamento completo, a partir do estado que ocorreu a falha é maior do que o reparo do plano tentando-se aproveitar ao máximo do plano original.

De acordo com [Nebel and Koehler, 1993], no pior caso, reparar um plano existente não é mais eficaz do que um novo replanejamento completo. Entretanto, na prática, o reparo de plano provou ser mais eficaz, uma vez que grande parte do plano ainda é válida na maioria dos casos. Como visto no Capítulo 6, utilizar um plano que já existe, ainda que seja necessário ajustá-lo, certamente demanda menos recursos do que construir um plano completamente novo. Além disso, em muitos domínios pode ser mais dispendioso modificar todo o plano devido a compromissos com outros agentes

baseados no plano original [Cushing and Kabhampati, 2005].

Assim, o objetivo desse trabalho foi investigar as vantagens entre replanejamento e reparo por meio da implementação de um sistema de reparo (VHPOP-RE-H) e compará-lo com o replanejamento utilizando o planejador clássico VHPOP.

7.1 Principais contribuições

Embora o VHPOP [Younes and Simmons, 2003] tenha uma implementação de domínio público, não foram disponibilizados planejadores não-determínisticos que executem reparo de plano por meio de refinamento reverso explícito [van der Krogt and de Weerdt, 2005]. Um dos objetivos do estudo foi prover uma implementação das técnicas exibidas nas seções anteriores e possibilitar a comparação empírica com outras abordagens.

Outra contribuição desse trabalho foi o desenvolvimento de uma heurística que viabilizasse a implementação do modelo de reparo de plano por refinamento reverso.

Finalmente, a análise experimental mostrou que o sistema VHPOP-RE-H é capaz de reparar, a maioria dos planos, de forma mais eficiente do que os sistemas VHPOP-RE e VHPOP, pois ele gasta menos tempo para encontrar uma solução e remove, no máximo, o mesmo número de ações que o replanejamento completo.

7.2 Trabalhos futuros

Algumas extensões possíveis desse trabalho são:

- Usar uma heurística clássica de busca no espaço de estados, como por exemplo FF ou HSP, para estimar o custo do replanejamento total.

- Monitorar se uma ou mais ações sempre falham em determinadas condições, para que o agente deixe de incluí-las em seus planos.

- Uma biblioteca de fragmentos de planos [van der Krogt et al., 2001], ou planejamento baseado em casos [Tonidandel, 2003].

- Um conjunto de macro-ações [Botea et al., 2004] pode melhorar o desempenho em ambientes onde existam informações prévias, possibilitando, inclusive, a utilização de alguma técnica reativa [de Castro Aranha, 2005] [Boella and Damiano, 2002] para reparo de planos.

Apêndice A

Domínios de teste

A.1 PDDL - Linguagem de Definição de Domínio de Planejamento

Em 1998 foi criada a *Problem Domain Definition Language* (PDDL) [McDermott, 1998] [McDermott and Committee, 1998] [McDermott, 2000]. Esta linguagem tem como principal objetivo representar os domínios do mundo real por meio de uma estrutura capaz de ser entendida e interpretada por um planejador. A maioria dos planejadores desenvolvidos hoje são capazes de utilizar a PDDL como representação de entrada do domínio para a geração de uma solução ou de um plano, já que esta linguagem tornou-se um padrão na área de planejamento automático. A representação do modelo do domínio deve ser a mais próxima possível do domínio real, contendo a descrição das ações possíveis, suas pré e pós-condições, as informações sobre o estado inicial do domínio e o estado objetivo (metas), para que o planejador possa processar o modelo.

Algumas características principais da PDDL são:

- A PDDL é uma representação direcionada às ações do domínio;
- As ações representadas em PDDL são baseadas em ações do modelo *Stanford Research Institute Planning System* (STRIPS) [Fikes and

Nilsson, 1971] em que as pré-condições e os efeitos de uma ação representam a dinâmica da execução desta no domínio;

- Possui traços da linguagem ADL [Pednault, 1989] que incluem a representação de efeitos condicionais nas ações, assim como qualificadores e quantificadores universais;
- Definição de restrições;
- Especificação de ações hierárquicas compostas [McDermott, 1998] por subações;
- Devido ao fato da linguagem ser padronizada, é possível que um mesmo problema representado em PDDL seja tratado por vários planejadores diferentes (portabilidade de problemas entre agentes planejadores).

A PDDL está em constante evolução desde a sua criação. Uma das principais evoluções da PDDL foi a chamada PDDL 2.1 [Fox and Long, 2003] que foi desenvolvida com a intenção de representar domínios de planejamento determinísticos que envolvem tempo e que necessitam de recursos de manipulação algébrica, incorporando também características da linguagem ADL [Pednault, 1989]. A PDDL 2.1 pode ser classificada em cinco níveis: o nível 1 é definido pela primeira versão da PDDL desenvolvida para a competição de *International Artificial Intelligence Planning Systems* (AIPS) em 1998. O nível 2 é um complemento ao nível 1, permitindo a utilização de recursos numéricos como a comparação entre variáveis numéricas e a atualização dos valores das mesmas. Os níveis 3 e 4 definem a representação de ações com dependência temporal tanto com efeitos não contínuos (nível 3) como com efeitos contínuos (nível 4). O nível 5 é uma extensão do nível 4 capaz de representar domínios contínuos e discretos em tempo real.

A representação de um modelo de domínio de planejamento em PDDL é dividida em duas partes. A primeira possui a definição do domínio em que são encontradas, principalmente, as ações possíveis no domínio assim como a declaração dos tipos de objetos existentes. Já a segunda parte possui a

definição do problema de planejamento a ser resolvido onde são fornecidos os estados iniciais do problema e o objetivo a ser atingido. Cada uma dessas partes é fornecida ao planejador na forma de arquivo (`.pddl`). A separação da definição do domínio e dos problemas é um fator positivo já que, para uma mesma definição de domínio, é possível raciocinar sobre diversos problemas [Vaquero, 2007].

A PDDL possui um formalismo para as definições do domínio e do problema. Toda a definição formal da linguagem pode ser encontrada na especificação da PDDL 2.1 [Fox and Long, 2003] ou em versões posteriores como a PDDL 2.2 [Edelkamp and Hoffmann, 2004] e PDDL 3.0 [Gerevini and Long, 2005].

A.2 Domínio do Mundo dos Blocos

```
01 ; The 4-operator blocks world domain from the 2nd International
02 ; Planning Competition.
03
04 (define (domain blocks)
05  (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (handempty)
06               (holding ?x))
07  (:action pick-up
08           :parameters (?x)
09           :precondition (and (clear ?x) (ontable ?x) (handempty))
10           :effect (and (not (ontable ?x)) (not (clear ?x))
11           (not (handempty)) (holding ?x)))
12  (:action put-down
13           :parameters (?x)
14           :precondition (holding ?x)
15           :effect (and (not (holding ?x)) (clear ?x) (handempty)
16                        (ontable ?x)))
17  (:action stack
18           :parameters (?x ?y)
19           :precondition (and (holding ?x) (clear ?y))
```

```
20          :effect (and (not (holding ?x)) (not (clear ?y))
21                       (clear ?x) (handempty) (on ?x ?y)))
22  (:action unstack
23          :parameters (?x ?y)
24          :precondition (and (on ?x ?y) (clear ?x) (handempty))
25          :effect (and (holding ?x) (clear ?y) (not (clear ?x))
26                       (not (handempty)) (not (on ?x ?y)))))
```

Apêndice B

Arquitetura do Sistema

B.1 Diagrama de implementação

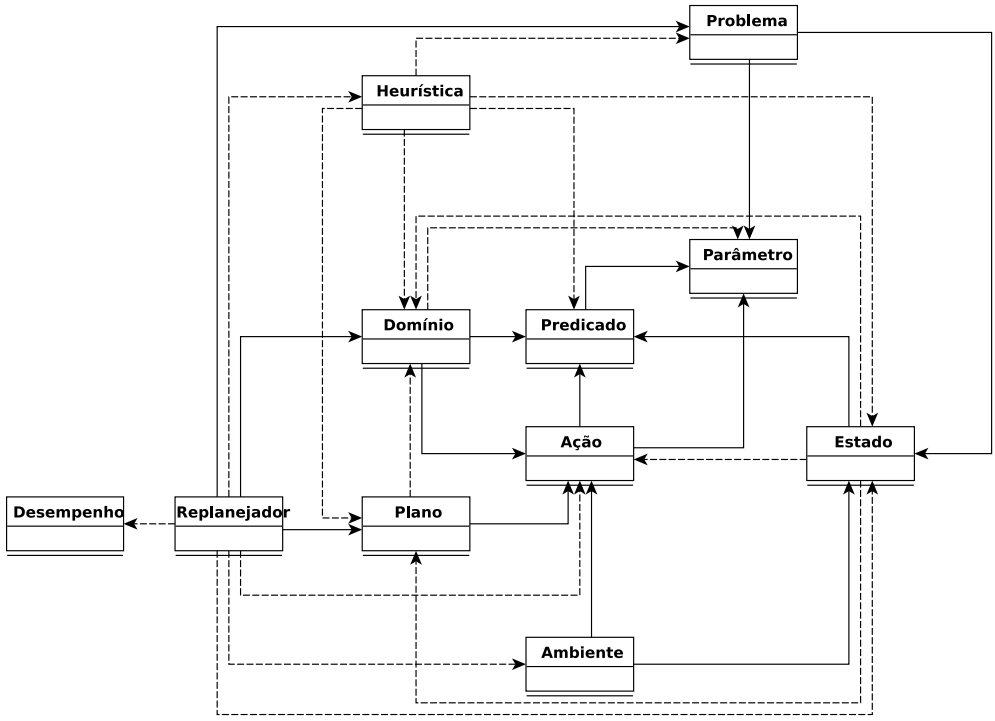

Figura B.1: Diagrama de Classe[1] do sistema de reparo VHPOP-RE

Referências Bibliográficas

[Bacchus, 2001] Bacchus, F. (2001). The aips'00 planning competition. 22 (3):47–56.

[Blum and Furst, 1997] Blum, A. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300.

[Boella and Damiano, 2002] Boella, G. and Damiano, R. (2002). A replanning algorithm for a reactive agent architecture. In *AIMSA*, pages 183–192.

[Bonet and Geffner, 2000] Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In *AIPS*, pages 52–61.

[Bonet and Geffner, 2001] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129:5–33.

[Botea et al., 2004] Botea, A., Müller, M., and Schaeffer, J. (2004). Using component abstraction for automatic generation of macro-actions. In *ICAPS*, pages 181–190.

[Cardoso, 2006] Cardoso, L. S. (2006). Aplicação da tecnologia de agentes de planejamento em operações de satélites. Master's thesis, Instituto Nacional de Pesquisas Espaciais.

[Cardoso et al., 2006] Cardoso, L. S., Orlando, V., calves Vieira Ferreira, M. G., and Biancho, A. C. (2006). Aplicação da tecnologia de planejamento em operações de satélites. *VII Simpósio Brasileiro de Automação Iteligente (SBAI)*.

[Cushing and Kabhampati, 2005] Cushing, W. and Kabhampati, S. (2005). Replanning: a new perspective. *International Conference on Automated Planning and Scheduling*.

[de Castro Aranha, 2005] de Castro Aranha, C. (2005). Sistema de substituição: uma técnica reativa para auto-reparo e auto-diagnóstico de planos. Master's thesis, Instituto de Computação - Universidade Estadual de Campinas.

[do Lago Pereira, 2002] do Lago Pereira, S. (2002). Planejamento abdutivo no cálculo de eventos. Master's thesis, Instituto de Matemática e Estatística - Universidade de São Paulo.

[Drabble et al., 1997] Drabble, Dalton, and Tate, A. (1997). Repairing plans on the fly. *NASA Workshop on planning and Scheduling for Space*.

[Draper et al., 1994] Draper, D., Hanks, S., and Weld, D. S. (1994). Probabilistic planning with information gathering and contingent execution. In *AIPS*, pages 31–36.

[Edelkamp and Hoffmann, 2004] Edelkamp, S. and Hoffmann, J. (2004). Pddl 2.2: The language for classical part of the 4th international planning competition.

[Erol et al., 1994] Erol, K., Hendler, J., and Nau, D. S. (1994). Semantics for hierarchical task network planning.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. In *IJCAI*, pages 608–620.

[Fowler and Scott, 2000] Fowler, M. and Scott, K. (2000). *UML Essencial*. Addison Wesley Longman, Inc, 2 edition.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). Pddl 2.1: An extension to pddl for expressing temporal planning domains. *JAIR*, 20:61–124.

[Gerevini and Long, 2005] Gerevini, A. and Long, D. (2005). Plan constraints and preferences in pddl3 - the language of fifth international planning competition.

[Gerevini and Serina, 2000] Gerevini, A. and Serina, I. (2000). Fast plan adaptation through planning graphs: Local and systematic search techniques. In *AIPS*, pages 112–121.

[Ghallab et al., 2004] Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan-Kauffman Publishers.

[Hansen and Zilberstein, 2001] Hansen, E. A. and Zilberstein, S. (2001). Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62.

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

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302.

[Ingerson and Steel, 1988] Ingerson, A. and Steel, S. (1988). Integratirig planning, execution and monitoring. *Conference on Artificial Intelligence*.

[Joslin and Pollack, 1994] Joslin, D. and Pollack, M. E. (1994). Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *AAAI*, pages 1004–1009.

[Kambhampati, 1997] Kambhampati, S. (1997). Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18:67–97.

[Koenig et al., 1995] Koenig, S., Goodwin, R., and Simmons, R. G. (1995). Robot navigation with markov models: A framework for path planning and learning with limited computational resources. In *Reasoning with Uncertainty in Robotics*, pages 322–337.

[Mason, 1993] Mason, M. T. (1993). Kicking the sensing habit. *AI Magazine*, 14 (1):58–59.

[McDermott, 1998] McDermott, D. (1998). The pddl planning domain definition language. In *AIPS*.

[McDermott, 2000] McDermott, D. (2000). The 1998 ai planning systems competition. *AI Magazine*, 2:21.

[McDermott and Committee, 1998] McDermott, D. and Committee, A. P. C. (1998). Pddl - the planning domain definition language. Technical report.

[Nareyek, 2003] Nareyek, A. (2003). Planning in dynamic worlds: More than external events. In *IJCAI*, pages 30–35.

[Nau et al., 1999] Nau, D. S., Cao, Y., Lotem, A., and Muñoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. In *IJCAI*, pages 968–975.

[Nau et al., 1995] Nau, D. S., Gupta, S. K., and Regli, W. C. (1995). Ai planning versus manufacturing-operation planning: A case study. In *IJCAI*, pages 1670–1676.

[Nebel and Koehler, 1993] Nebel, B. and Koehler, J. (1993). Plan modification versus plan generation: A complexity-theoretic perspective. In *IJCAI*, pages 1436–1444.

[Newell and Simon, 1961] Newell, A. and Simon, H. A. (1961). Gps, a program that simulates human thought. In *Lernende Automaten*, pages 109–124.

[Pednault, 1989] Pednault, E. P. D. (1989). Adl: Exploring the middle ground between strips and the situation calculus. In *KR*, pages 324–332.

[Penberthy and Weld, 1992] Penberthy, J. S. and Weld, D. S. (1992). Ucpop: A sound, complete, partial order planner for adl. In *KR*, pages 103–114.

[Peot and Smith, 1993] Peot, M. A. and Smith, D. E. (1993). Threat-removal strategies for partial-order planning. In *AAAI*.

[Rabideau et al., 1999] Rabideau, G., Knight, R., Chien, S., Fukunaga, A., and Govindjee, A. (1999). Iterative repair planning for spacecraft operations using the aspen system. *Artificial Intelligence, Robotics and Automa-*

tion in Space, Proceedings of the Fifth International Symposium, ISAI-RAS '99, held 1-3 June, 1999 in ESTEC, Noordwijk, the Netherlands. Edited by M. Perry. Paris: European Space Agency., ESA SP-440:99.

[Russell and Norvig, 2002] Russell, S. J. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[Schubert and Gerevini, 1995] Schubert, L. K. and Gerevini, A. (1995). Accelerating partial order planners by improving plan and goal choices. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 442–450, Herndon, Virginia. IEEE Computer Society Press.

[Smith et al., 1999] Smith, B., Millar, W., Wen, Y. J. D., Nayak, P., and Clark, M. (1999). Validation and verification of the remote agent for space-craft autonomy. In *IEEE Aerospace Conference*.

[Smith and Weld, 1998] Smith, D. E. and Weld, D. S. (1998). Conformant grafplan. *Conference on Artificial Intelligence*.

[Tonidandel, 2003] Tonidandel, F. (2003). *Desenvolvimento e Implementação de um Sistema de Planejamento Baseado em Casos*. PhD thesis, Escola Politécnica - Universidade de São Paulo.

[van der Krogt, 2005] van der Krogt, R. (2005). *Plan Repair in Single-Agent and Multi-Agent Systems*. PhD thesis, Delft University of Technology, Delft, The Netherlands.

[van der Krogt et al., 2001] van der Krogt, R., Bos, A., and Witteveen, C. (2001). Plan fragment libraries. *Belgium-Netherlands Artificial Intelligence Conference*.

[van der Krogt and de Weerdt, 2004a] van der Krogt, R. and de Weerdt, M. M. (2004a). Plan repair: A framework and a new heuristic with applications to logistics. In *Proceedings of the 8th TRAIL Congress*.

[van der Krogt and de Weerdt, 2004b] van der Krogt, R. and de Weerdt, M. M. (2004b). The two faces of plan repair. *Proceedings of the BNAIC*, pages 147–154.

[van der Krogt and de Weerdt, 2005] van der Krogt, R. and de Weerdt, M. M. (2005). Plan repair as an extension of planning. In *ICAPS*, pages 161–170.

[Vaquero, 2007] Vaquero, T. S. (2007). Itsimple: Ambiente integrado de modelagem e análise de domínos de planejamento automático. Master's thesis, Escola Politécnica - Universidade de São Paulo.

[Wang and Chien, 1997] Wang, X. and Chien, S. A. (1997). Replanning using hierarchical task network and operator-based planning. In *ECP*, pages 427–439.

[Warren, 1976] Warren, D. H. D. (1976). Generating conditional plans and programs. *AISB Summer Conference*.

[Wilkins, 1988] Wilkins, D. E. (1988). Pratical planning: Extending the ai planning paradigm.

[Wilkins, 1990] Wilkins, D. E. (1990). Can ai planners solve practical problems? *Computational Intelligence*.

[Winston, 1992] Winston, P. H. (1992). *Inteligência Artificial*. Livros Técnicos e Científicos.

[Younes and Simmons, 2003] Younes, H. L. S. and Simmons, R. G. (2003). Vhpop: Vesatile heuristic partial order planner. *Jair*, 20:405–430.