Planejamento Hierárquico com Ações Não-Determinísticas

Ricardo Guimarães Herrmann
<herrmann@ime.usp.br>

Projeto de dissertação a ser apresentado como requisito parcial para qualificação no curso de Mestrado em Ciência da Computação do IME-USP

Orientadora: Prof$^{\underline{a}}$ Dr$^{\underline{a}}$ Leliane Nunes de Barros

– Abril de 2007 –

Resumo

Planejamento em Inteligência Artificial (IA) permite que agentes inteligentes busquem planos de ações para problemas envolvendo ambientes dinâmicos. O planejamento clássico busca a satisfação de um conjunto de propriedades no ambiente.

O planejamento hierárquico, por outro lado, tem como objetivo decompor uma rede de tarefas em ações executáveis, isto é, gerar um plano de ações que, ao ser executado a partir de uma dada situação inicial, realize um conjunto de tarefas. Por tirar proveito da natureza hierárquica dos domínios práticos, esta técnica permite resolver problemas que necessitam de planos com milhares de ações. Entretanto, uma solução desta magnitude pode apresentar dificuldades em tempo de execução, tanto devido à informação omitida ou incorreta sobre o mundo, quanto à presença de outros agentes manipulando o mesmo ambiente.

As ações em planejamento podem ser determinísticas ou não-determinísticas. A resolução de problemas de planejameno sob incerteza, ou planejamento não-determinístico, utiliza técnicas computacionalmente caras, geralmente de maior interesse teórico. Por este motivo, encontrar planos para problemas de grande porte ainda é um grande desafio.

A verificação de modelos simbólicos provê alguns meios para tratar a representação de incertezas de modo compacto, possibilitando um ganho de desempenho em domínios que apresentam incerteza Knightiana, ou seja, sem probabilidades associadas.

Este trabalho se baseia em um estudo [KNPT05] sobre a integração das técnicas de planejamento hierárquico, planejamento não-determinístico e planejamento com verificação de modelos simbólicos, que resulta em um meio mais eficiente de escolha de ações, para diversos domínios de aplicação. Esta técnica permite ao planejamento hierárquico fazer predições sobre as eventuais contingências que venham a ocorrer durante a execução de planos.

Sumário

Resumo **iii**

1 Introdução **1**
- 1.1 Motivação 1
- 1.2 Objetivos 3
- 1.3 Organização 3

2 Planejamento Clássico **5**
- 2.1 Modelo Conceitual para Planejamento 5
- 2.2 Planejamento Clássico 7
  - 2.2.1 Algoritmos para Planejamento Clássico 8

3 Planejamento Hierárquico **13**
- 3.1 Planejamento com Redes Hierárquicas de Tarefas 14
- 3.2 Planejamento com Redes Simples de Tarefas 16
- 3.3 Extensões 19
  - 3.3.1 Extensões Adicionais 20
  - 3.3.2 Planejadores Hierárquicos Existentes 20
  - 3.3.3 Vantagens e Desvantagens 21

4 Planejamento sob Incerteza **23**
- 4.1 Suposições 23
  - 4.1.1 Determinismo nos Efeitos de Ações 23
  - 4.1.2 Observabilidade Parcial 24
- 4.2 Técnicas de Planejamento sob Incerteza 24
- 4.3 Modelo 25
- 4.4 Planejamento Forte, Forte Cíclico e Fraco 26
  - 4.4.1 Planejamento Forte 27
  - 4.4.2 Planejamento Forte Cíclico 27
  - 4.4.3 Planejamento Fraco 28
- 4.5 Planejamento baseado em Verificação de Modelos 28
  - 4.5.1 Verificação de Modelos Simbólicos 29

5 Planejamento Hierárquico sob Incerteza **31**
- 5.1 "Indeterminizando" Planejadores Progressivos 31
- 5.2 Planejador YoYo 34

5.2.1 Algoritmo 34
5.2.2 Planejamento Forte 37
5.2.3 Utilização de BDDs 38

6 Proposta da Dissertação **41**
6.1 Implementação 41
6.2 Atividades 43
6.3 Cronograma 43

Lista de Figuras

2.1 Exemplo de sistema de transição de estados 6
2.2 Modelo conceitual para planejamento 7

3.1 Exemplo de decomposição de tarefas em ordem total 17

4.1 Sistema de transição de estados não-determinístico 26

Lista de Algoritmos

1 PlanejamentoProgressivo $(s_0, S_g, \mathcal{D})$ . . . 9
2 PlanejamentoRegressivo $(s_0, S_g, \mathcal{D})$ . . . 10
3 DecomposiçãoOrdemParcial $(s, w, \mathcal{D}, \mathcal{M})$ . . . 19
4 VerificaForte$(\pi, S_{abertos}, S_g, S_0)$ . . . 27
5 VerificaForteCíclico$(\pi, S_{abertos}, S_g, S_0)$ . . . 28
6 VerificaFraco$(\pi, S_{abertos}, S_g, S_0)$ . . . 29
7 FCP $(s_0, g, \mathcal{D}, \alpha)$ . . . 32
8 ND-FCP $(S_0, g, \mathcal{D}', \alpha')$ . . . 33
9 YoYo$(\mathcal{D}, S_0, S_g, w, \mathcal{M})$ . . . 34
10 YoYoAux$(\mathcal{D}, X, S_g, \mathcal{M}, \pi, x_0)$ . . . 35
11 PodaSituações(X, S_g, π) . . . 35
12 EfetuaDecomposição$(S, w, \mathcal{D}, \mathcal{M})$ . . . 36
13 EncontraSucessores(F, X) . . . 37
14 VerificaPolítica_Forte(π, X, S_g, x_0) . . . 38

Nomenclatura

Σ	sistema de transição de estados
Σ'	sistema de transição de estados restrito (estático e determinístico)
Σ_π	estrutura de execução de uma política
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, política
σ	substituição lógica
.	composição de seqüências
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
$\mathcal{F}$	conjunto de símbolos de tarefas primitivas
$\mathcal{K}$	estrutura de Kripke
$\mathcal{M}$	conjunto de todos os métodos possíveis
$\mathcal{N}$	conjunto de rótulos para redes de tarefas
$\mathcal{O}$	conjunto de todas as observações possíveis
$\mathcal{P}$	conjunto de proposições de um domínio de planejamento
$\mathcal{P}$	conjunto de símbolos de predicados
$\mathcal{S}$	conjunto de todos os estados possíveis
$\mathcal{T}$	conjunto de símbolos de tarefas compostas
$\mathcal{V}$	conjunto de símbolos de variáveis
E	conjunto de arestas
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 meta

U	conjunto de nós
no-op	ação nula
a	ação de $\mathcal{A}$
e	evento de $\mathcal{E}$
k	índice
l	literal
m	método
n	rótulo de $\mathcal{N}$
r	termo
s	estado de $\mathcal{S}$
s_0	estado inicial
t_u	tarefa associada a um nó de uma rede de tarefas
u, v	nós de uma rede de tarefas
w	rede de tarefas
x	variável
$\langle \ldots \rangle$	tupla
$(\ldots)$	lista
$\{\ldots\}$	conjunto
$\cup$	união de conjuntos
$\cap$	interseção de conjuntos
$\setminus$	subtração de conjuntos

Lista de Acrônimos

IA Inteligência Artificial

STN Rede Simples de Tarefas (*Simple Task Network*)

HTN Rede Hierárquica de Tarefas (*Hierarchical Task Network*)

TFD Decomposição Progressiva em Ordem Total (*Total-order Forward Decomposition*)

PFD Decomposigão Progressiva em Ordee Parcial (*Partial-order Forward Decomposition*)

FCP Planejamento Progressivo (*Forward Chaining Planning*)

ND-FCP *Non-Deterministic Forward Chaining Planning*

SHOP *Simple Hierarchical Ordered Planner*

STRIPS *Stanford Research Institute Planning System*

UMCP *Universal Method Composition Planner*

NOAH *Nets Of Action Hierarchies*

SOUP *Semantics Of User Programs*

CTL Lógica de Tempo Ramificado (*Computation Tree Logic*)

LTL Logica de Tempo Linear (*Linear Temporal Logic*)

TAL Lógicas de Ação Temporal (*Temporal Action Logics*)

BDD Diagrama de Decisão Binária (*Binary Decision Diagram*)

OBDD Diagrama de Decisão Binária Ordenado (*Ordered Binary Decision Diagram*)

MDP Processo de Decisão de Markov (*Markov Decision Process*)

Capítulo 1

Introdução

Planejamento em IA [GNT04] consiste na busca automatizada de planos de ações que visam alcançar metas pré-estabelecidas em mundos descritos formalmente. O raciocínio de planejamento envolve a antecipação das conseqüências de ações, antes mesmo de executá-las, a fim de selecionar ações que satisfaçam as metas do agente. Planejamento é, portanto, um componente importante na construção de agentes inteligentes autônomos.

Planos podem ser de diferentes tipos: os mais simples podem ser representados como sequências de ações, totalmente ou parcialmente ordenadas. Planos condicionais possuem ramificações, seguidas de acordo com condições que venham a ocorrer durante sua execução. Políticas, que mapeiam estados do mundo em ações, representam planos na forma de um agente reativo específico para o problema, com uma ação para todo estado possível de ser alcançado a despeito dos eventos externos. Existem ainda planos estendidos, que associam contextos de execução a estados, aumentando o poder de expressão de políticas.

Há diversas formas de planejamento para fins específicos, como planejamento de projeto, de sensoriamento, navegação robótica, entre outras. Estas abordagens *específicas de domínio* são muito bem sucedidas em suas áreas de aplicação, mas o planejamento em IA concentra-se em formas de planejamento *independentes de domínio*, reutilizáveis, com o intuito de reduzir o esforço necessário à criação de novos planejadores [GNT04].

1.1 Motivação

O *planejamento clássico* [GNT04] faz suposições restritivas sobre: o modelo do mundo (ou modelo do ambiente); o tipo de solução desejada; as características da meta; a capacidade do agente em termos de observação do estado do mundo e interação com o ambiente. Mais especificamente, no planejamento clássico faz-se a suposição de que o ambiente possa ser modelado como um sistema de transição de estados determinístico, estático, finito, completamente observável e com tempo implícito, com metas especificadas através de estados desejados (metas de alcançabilidade), sem realimentação do controlador sobre o estado da execução e com soluções que possam ser dadas por seqüências de ações. Mesmo com tais suposições, o planejamento clássico é um problema NP-completo [ENS95]. Entretanto, problemas de interesse prático, em sua maioria, não podem ser resolvidos sob tais suposições.

Uma variação de planejamento clássico é o *planejamento hierárquico* [Yan90] [EHN94b], em que o controle de busca é efetuado pela decomposição de *tarefas de alto nível* em tarefas menores, até o nível de ações primitivas (isto é, ações que podem ser executadas diretamente pelo sistema de controle). Os diferentes métodos de decomposição de uma tarefa em sub-tarefas são especificados pelo projetista do domínio. Os métodos de decomposição permitem maior controle sobre a busca de soluções, o que contribui para o fato de que planejamento hierárquico é atualmente a técnica mais utilizada em problemas práticos. A utilização deste conhecimento adicional sobre o domínio permite derivar planejadores que obtêm soluções em tempo polinomial em relação ao tamanho do problema para casos em que, através do planejamento clássico, só poderiam ser solucionados em tempo exponencial.

Outra característica presente em problemas práticos de planejamento é a incerteza sobre o estado do mundo e conseqüências de ações. O modelo de sensores do agente raramente pode prover uma especificação completa do estado do mundo. Ações podem falhar ou ter conseqüências diferentes em situações específicas, sem mencionar ações nas quais o indeterminismo é intrínseco, como o lançamento de um dado. O estado do mundo também pode ser modificado por fatores externos, sobre os quais o agente não tem controle ou até mesmo conhecimento da existência. Estas são suposições comuns a problemas práticos e devem ser levadas em consideração para que os planos sintetizados tenham alguma validade, ou seja, sejam corretos.

Existem extensões do planejamento clássico para lidar com domínios de planejamento *não-determinístico*, em que uma ação pode levar o agente a diferentes estados do mundo. Em determinados domínios e problemas, ainda que o planejador não possa prever qual destes estados irá ocorrer após a execução da ação, é possível encontrar um plano que satisfaça a meta do problema.

Quando é possível identificar distribuições de probabilidades nos efeitos das ações e associar recompensas aos estados do mundo, a técnica predominantemente utilizada é a representação de problemas através de um Processo de Decisão de Markov (*Markov Decision Process*) (MDP) [How60]. Nesse caso, o objetivo do planejador é obter uma política que maximiza a recompensa dos estados visitados com base nas probabilidades dos efeitos das ações [BG01].

A técnica mais utilizada para o tratamento de problemas de planejamento não-determinísticos (isto é, sem probabilidades associadas aos efeitos das ações) é a de *verificação de modelos* [Rov01]. Soluções podem ser políticas de diferentes níveis de garantias de alcance de metas, caracterizadas, em nível decrescente de garantia, como *fortes*, *fortes cíclicas* ou *fracas*. A *verificação de modelos simbólicos* utiliza formas compactas de representação de estados e ações, permitindo que algoritmos trabalhem com conjuntos de estados que possuem características comuns, raciocinando sobre conjuntos de estados e seus sucessores de acordo com escolhas de ações. Esta técnica baseia-se no cálculo da pré-imagem da função de transição [Rov01].

Uma outra abordagem promissora para resolver problemas de planejamento não-determinístico é a de *planejamento progressivo hierárquico*, originalmente proposta para fornecer heurísticas eficientes de controle de busca para domínios com efeitos determinísticos [NAI$^+$03]. Em [KN04] é proposta uma forma de generalizar essa abordagem para tratar o caso não-determinístico, junto com uma classe de outras técnicas de planejamento progressivo. Esta técnica baseia-se na propriedade de que planejadores que utili-

zam busca progressiva no espaço de estados sempre têm conhecimento completo do estado do mundo, enquanto o emprego de planejamento hierárquico possibilita uma redução do espaço de busca. Entretanto, não existe uma implementação disponível do planejador descrito em [KN04] e [KNPT05]. As técnicas de planejamento hierárquico e planejamento não-determinístico apresentam suas características próprias, bem como fazem diferentes suposições sobre as linguagens de representação de domínios de planejamento. É, portanto, de grande utilidade integrar essas técnicas diferentes de planejamento, como é a proposta do algoritmo YoYo [KNPT05].

1.2 Objetivos

O objetivo deste trabalho é estudar essas duas técnicas de planejamento e implementar o algoritmo YoYo para resolver problemas que envolvam planejamento não-determinístico e hierárquico com busca progressiva.

O sistema desenvolvido será comparado em termos de desempenho com algoritmos clássicos de planejamento não-determinístico.

1.3 Organização

O Capítulo 2 apresenta a forma geral do modelo utilizado em planejamento e as suposições restritivas do planejamento clássico, além de meios de representação simbólica de estados e ações.

Em seguida, no Capítulo 3, são apresentadas duas formas de planejamento hierárquico, uma mais geral e outra especializada para lidar com busca progressiva. Além disso, enumera algumas possíveis extensões e explica suas vantagens e desvantagens.

O Capítulo 4 introduz o conceito de não-determinismo e suas consequências em planejamento, causadas por ações não determinísticas e observabilidade nula ou parcial, o que caracteriza o planejamento não-determinístico. Também expõe as diferentes classes de soluções como políticas e respectivos níveis de garantia de alcance de metas. Além disso, provê uma visão rápida do papel da verificação de modelos em sistemas de planejamento mais recentes e técnicas de modelagem simbólica associadas.

Meios de união das técnicas listadas anteriormente são exibidos no Capítulo 5, que mostra como é possível obter um planejador hierárquico não-determinístico baseado em busca progressiva e em seguida estendê-lo para que faça uso de técnicas utilizadas em verificação de modelos simbólicos em busca de maior eficiência.

Finalmente, o Capítulo 6 apresenta a abordagem tomada neste trabalho, seguida do cronograma de trabalho.

Capítulo 2

Planejamento Clássico

Planejamento em geral envolve muitos fatores como tempo, recursos, otimização, entre outros. Além disso, há diversas maneiras de representação de ações e seus resultados. Um meio efetivo para a compreensão de problemas complexos é criar uma versão restrita do problema. O planejamento clássico busca prover um modelo simplificado de planejamento, bem fundamentado e caracterizado, para servir de base para técnicas mais sofisticadas de planejamento. Este capítulo apresentada o planejamento clássico, além de uma breve revisão sobre os principais algoritmos da área.

2.1 Modelo Conceitual para Planejamento

Definição 2.1.1. O formalismo mais comum para ambientes de planejamento é um *sistema de transição de estados*, uma 4-tupla $\Sigma = \langle \mathcal{S}, \mathcal{A}, \mathcal{E}, \gamma \rangle$, onde:

- $\mathcal{S} = \{s_1, s_2, \ldots\}$ é um conjunto finito ou recursivamente enumerável de estados;
- $\mathcal{A} = \{a_1, a_2, \ldots\}$ é um conjunto finito ou recursivamente enumerável de ações;
- $\mathcal{E} = \{e_1, e_2, \ldots\}$ é um conjunto finito ou recursivamente enumerável de eventos;
- $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \rightarrow 2^{\mathcal{S}}$ é uma função de transição de estados.

Estados representam situações possíveis de ambientes. Ações e eventos representam os acontecimentos ocorridos que levam um ambiente a mudar de estado. A diferença entre ações e eventos é que estes últimos não estão sob controle do agente e podem ocorrer independentemente de suas ações.

Um sistema de transição de estados pode ser representado como um *grafo* (Fig. 2.1), onde nós representam estados e transições são arestas anotadas com pares ação-evento. Também utiliza-se a ação nula $\mathsf{no\text{-}op} \in \mathcal{A}$ e o evento nulo $\epsilon \in \mathcal{E}$ para denotar transições causadas exclusivamente por eventos ou ações, respectivamente (onde a omissão de uma ação ou evento na notação de transições implica o uso destes).

Definição 2.1.2. Uma ação a é *aplicável* a um estado s se e só se $\gamma(s, a) \neq \emptyset$. Se a é aplicada no estado s, leva o ambiente a um estado $s' \in \gamma(s, a, e)$.

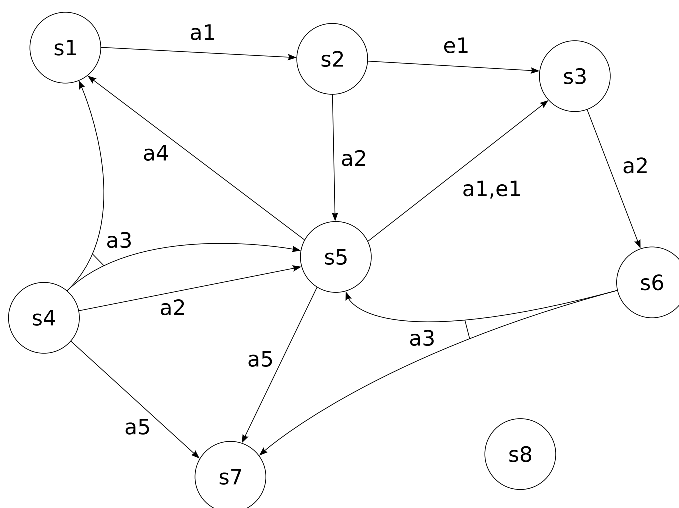

Figura 2.1: Exemplo de sistema de transição de estados

É conveniente visualizar este modelo através de uma interação entre três componentes (Fig. 2.2):

1. Um *sistema de transição de estados* Σ, que evolui de acordo com a sua função de transição de estados γ, reagindo aos eventos e ações às quais é submetido;

2. Um *controlador*, que recebe como entrada o estado s do sistema (através de observações) e provê como saída uma ação a de acordo com algum plano;

3. Um *planejador*, que tem como entrada uma descrição do sistema Σ, uma situação inicial e um objetivo. Sintetiza um plano para ser repassado ao controlador, com o intuito de atingir o objetivo. Em alguns tipos de planejamento, o planejador necessita de uma realimentação por parte do controlador, através do fornecimento do *estado da execução* do plano.

O controlador trabalha *online* com Σ. Já o planejador não se comunica diretamente com Σ e trabalha *offline*. Este último concentra-se não no estado atual do mundo em tempo de planejamento, mas em quais estados o ambiente modelado possa vir a estar enquanto o controlador executa o plano.

Outro elemento importante pode ser adicionado ao modelo: a informação que o controlador possui sobre o estado atual do mundo pode não ser completa. O conhecimento parcial do estado pode ser modelado como uma função de observação $\eta : \mathcal{S} \rightarrow 2^{\mathcal{O}}$ que associa a cada estado s um conjunto $\eta(s) = \{o_1, o_2, \ldots\}$ de possíveis observações. As percepções do controlador tornam-se então a observação $O = \eta(s)$.

Na maioria das vezes há diferenças entre o sistema físico a ser controlado e seu modelo, e o planejamento limita-se a este último. Portanto, é comum supor que o controlador seja robusto o suficiente para lidar com as diferenças entre Σ e o mundo real.

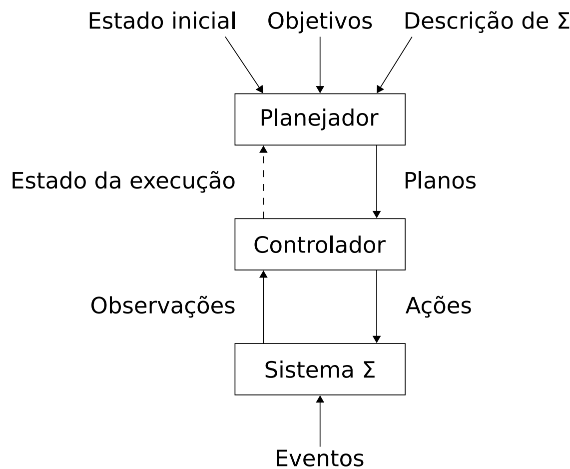

Figura 2.2: Modelo conceitual para planejamento

2.2 Planejamento Clássico

O formalismo clássico para planejamento baseia-se em um modelo restrito, a fim de manter uma semântica bem definida, e utiliza o conceito de estados e operadores do sistema *Stanford Research Institute Planning System* (STRIPS) [FN71], considerado um dos pioneiros em planejamento automatizado.

Definição 2.2.1. Um *sistema de transição de estados restrito* é uma 3-tupla $\Sigma' = \langle \mathcal{S}, \mathcal{A}, \gamma \rangle$, onde:

- $\mathcal{S} = \{s_1, s_2, \ldots\}$ é o conjunto finito ou recursivamente enumerável de estados;
- $\mathcal{A} = \{a_1, a_2, \ldots\}$ é o conjunto finito ou recursivamente enumerável de ações;
- $\gamma : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ é a função (parcial) de transição de estados.

Definição 2.2.2. Um *problema de planejamento clássico* é a 3-tupla $\mathcal{P} = \langle \Sigma', s_0, S_g \rangle$, onde:

- Σ' um sistema de transição de estados restrito, ou *domínio de planejamento*;
- s_0 é um *estado inicial*;
- S_g é uma *meta* (ou *objetivo*) de alcançabilidade, formada por *estados meta*.

Um *planejador clássico* deve, dado um problema, devolver um *plano*, uma seqüência de n ações $\pi = (a_1, \ldots, a_n)$ que, se executadas a partir de s_0, levam a um estado meta, ou seja, $\gamma(s_0, a_1) = s_1, \gamma(s_1, a_2) = s_2, \ldots, \gamma(s_{n-1}, a_n) = s_n$, onde $s_n \in S_g$. O planejamento clássico parte das seguintes suposições restritivas:

- Σ' **finito**, onde há um número finito de estados;
- Σ' **completamente observável**, significando que é sempre possível saber com exatidão qual o estado atual do mundo;

- Σ' **determinístico**, onde ações possuem apenas um efeito possível;
- Σ' **estático**, em que não ocorrem eventos externos ao agente, ou seja, somente o agente manipula o mundo;
- **Metas restritas**, denotadas por um conjunto de estados meta;
- **Planos seqüenciais** representados simplesmente como uma seqüência de ações a serem executadas na ordem apresentada;
- **Tempo implícito** abstraindo o tempo em um sistema episódico (tempo discreto), onde apenas uma ação ocorre por vez e não há informação de duração;
- **Planejamento *offline*** caracterizando a independência do planejador em relação ao executor, sem realimentação sobre a execução do plano.

Em suma, o planejamento clássico implica em uma busca por *planos seqüenciais em um sistema de transição de estados determinístico, estático, finito e completamente observável, com metas de alcançabilidade e tempo implícito, sem realimentação do controlador sobre o estado da execução.*

Representação Simbólica de Estados e Ações

Problemas de planejamento de interesse prático geralmente envolvem uma grande quantidade de estados e ações, o que inviabiliza a representação explícita de Σ. Portanto, é prática comum representar simbolicamente estados e ações, utilizando lógica de primeira ordem, com estados representados como conjuntos de literais verdadeiros sob alguma interpretação. Entretanto, apesar do uso de uma linguagem lógica de primeira ordem, um estado não é representado por um conjunto de fórmulas, e sim, apenas, por um conjunto de átomos totalmente instanciados. Além de permitir representações mais compactas, a representação simbólica de estados e ações promove o desacoplamento entre domínios e problemas de planejamento, separando-os em sua estrutura de ações e suas constantes, respectivamente.

Definição 2.2.3. *Operadores* são representações simbólicas de ações que alteram o valor-verdade de literais, através de listas de adição e remoção destes (chamadas *efeitos* do operador), sendo instâncias de operadores aplicáveis a um estado se este respeita uma lista de literais verdadeiros chamados *pré-condições* do operador.

O planejamento clássico costuma utilizar a *suposição de mundo fechado*: se um átomo não aparece na especificação de um estado, então é tido como falso.

2.2.1 Algoritmos para Planejamento Clássico

Busca no Espaço de Estados

Os algoritmos mais simples para planejamento clássico efetuam busca no espaço de estados. Nestes, o espaço de busca é um subconjunto do espaço de estados, onde cada nó corresponde a um estado do mundo e cada aresta a uma transição. O planejamento no espaço de estados divide-se basicamente em algoritmos *progressivos* e *regressivos*.

O procedimento de busca progressiva começa pelo estado inicial e aplica, não-deterministicamente, a função de transição de estados, produzindo sub-problemas. Estes sub-problemas buscam *soluções parciais*, pois são parte da solução final devolvida pela invocação do problema original. A busca termina quando um destes sub-problemas alcança um estado objetivo, ou *falha* se não houver nenhum plano possível.

Uma característica importante do planejamento progressivo (Algoritmo 1) é que permite ao planejador ter conhecimento sobre o estado completo do mundo a qualquer instante. Isto deve-se ao fato de que este opera a partir de um estado inicial completamente especificado e aplica sempre ações totalmente instanciadas aos estados, resultando em mais especificações completas de estados. É possível tirar proveito desta propriedade de diversas maneiras e este trabalho explora algumas destas possibilidades.

Algoritmo 1: PLANEJAMENTOPROGRESSIVO $(s_0, S_g, \mathcal{D})$

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

```
1  início
2  |  π ← ∅; s ← s0;
3  |  repita
4  |  |  se s ∈ Sg então devolva π;
5  |  |  A ← {a | a é uma ação (instância total de um operador em D) e a é
   |  |  aplicável a s};
6  |  |  se A = ∅ então devolva falha;
7  |  |  não-deterministicamente escolha a ∈ A;
8  |  |  π ← π.a;
9  |  |  s ← γ(s, a);
10 |  fim
11 fim
```

O planejamento no espaço de estados também pode ser efetuado por uma busca regressiva (Algoritmo 2). O procedimento começa pelos estados-objetivo e aplica a função inversa da transição, produzindo sub-objetivos. A busca termina quando um destes sub-objetivos é satisfeito pelo estado inicial.

Busca no Espaço de Planos

Outro meio de encontrar planos para problemas de planejamento clássico é utilizando *planejamento no espaço de planos* [GNT04]. Neste, o espaço de busca é o de *planos parcialmente especificados*, onde nós representam planos parciais e as arestas entre eles representam *operações de refinamento* de planos. Uma operação de refinamento adiciona restrições à forma do plano, através de uma entre *introdução de ações*, *vínculos causais* e mais dois tipos de restrições: de *ordem de ações* ou de *atribuição de variáveis*. Todas estas formas de refinamento evitam adicionar ao plano restrições que não sejam estritamente necessárias, o que caracteriza o *princípio do comprometimento mínimo*. A busca ocorre de forma progressiva, iniciando-se a partir de um nó inicial, representando o plano vazio, e prossegue com o refinamento, até que um nó contendo um plano que alcance as metas

Algoritmo 2: PLANEJAMENTOREGRESSIVO $(s_0, S_g, \mathcal{D})$

```
   Entrada: Estado inicial s0, Objetivo Sg, Domínio D
   Saída: Plano π
1  início
2  |  π ← ∅;
3  |  repita
4  |  |  se s ∈ Sg então devolva π;
5  |  |  A ← {a | a é uma ação (instância total de um operador em D) e a é
   |  |  relevante para g};
6  |  |  se A = ∅ então devolva falha;
7  |  |  não-deterministicamente escolha a ∈ A;
8  |  |  π ← a.π;
9  |  |  Sg ← γ^{-1}(Sg, a);
10 |  fim
11 fim
```

de planejamento seja encontrado. Não há como efetuar uma busca regressiva no espaço de planos, pois o nó objetivo representaria o próprio plano que se está buscando.

Grafo de Planejamento

Graphplan [BF97] é outro método de planejamento clássico, que utiliza como espaço de busca um *grafo de planejamento*. Tem como saída uma seqüência de conjuntos de ações, da forma

$$\pi = (\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, \ldots\}, \ldots),$$

que representa todas as seqüências começando em a_1 e a_2 em qualquer ordem, seguidas por a_3 e a_4, também em qualquer ordem, e assim por diante. Diferentemente do planejamento no espaço de planos, estabelece fortes compromissos: ações são consideradas totalmente instanciadas e devem ocorrer em passos específicos. Esta técnica de planejamento depende da *análise de alcançabilidade* e do *refinamento disjuntivo*. A análise de alcançabilidade serve para determinar se um estado é alcançável a partir de algum outro estado s_0. O refinamento disjuntivo consiste em lidar com uma ou mais falhas através de uma disjunção de soluções. Como em geral falhas não são independentes e suas soluções podem interferir, relações de dependência são colocadas como restrições a serem respeitadas em um estágio posterior.

A estrutura utilizada é um grafo em camadas alternadas de proposições e ações, onde cada camada de proposições representa, através de sua combinação, os possíveis estados alcançáveis a partir do estado inicial, pela aplicação de ações das camadas anteriores. Arestas são adicionadas ligando uma proposição a uma ação quando uma proposição é pré-condição desta. Do mesmo modo, ações ligam-se a proposições para denotar efeitos. Além disso, são adicionadas *restrições de exclusão mútua* entre pares de ações ou de proposições, a fim de representar escolhas de ações que não podem ocorrer simultaneamente em uma camada. Esta forma de planejamento apresenta duas fases distintas: uma de expansão do grafo de planejamento, adicionando um par de camadas de ações e propo-

sições, e outra de tentativa de extração de planos, respeitando as restrições de exclusão mútua. Estas fases alternam-se iterativamente até que algum plano seja encontrado.

Planejamento como Satisfazibilidade

A idéia por detrás do planejamento como satisfazibilidade é mapear um problema de planejamento a um outro problema bem conhecido, para o qual já existem algoritmos eficientes. Um plano é então extraído da solução deste outro problema. Mais especificamente, a idéia é reduzir um problema de planejamento a um *problema de satisfazibilidade proposicional* (também conhecido como SAT), ou seja, que procura dizer se uma fórmula proposicional é satisfazível. Avanços no desempenho de algoritmos para SAT permitem lidar com problemas mais complexos sem que seja necessário introduzir melhorias em algoritmos de planejamento. Existem diferentes formas de codificação de um problema de planejamento como SAT. Esta técnica foi originalmente proposta pelo planejador SatPlan [KS92]. A codificação como SAT também pode ser utilizada na fase de busca de planos de Graphplan, resultando no planejador Blackbox [KS99].

Capítulo 3

Planejamento Hierárquico

O *planejamento hierárquico*, também conhecido como Rede Hierárquica de Tarefas (*Hierarchical Task Network*) (HTN), provê uma forma particular de especificação de regras de controle de busca, aproveitando-se da natureza hierárquica inerente a muitos problemas práticos. Difere do planejamento clássico quanto a seu tipo de meta: o objetivo é, ao invés de alcançar um estado meta, realizar um conjunto de *tarefas* através de sua *decomposição* em sub-tarefas menores, recursivamente, até que seja possível realizá-las diretamente através de ações primitivas. Cada decomposição possível representa uma ramificação na árvore de busca do problema. O projetista do domínio deve especificar diferentes *métodos* de decomposição para cada *tarefa composta*, junto com suas pré-condições para que sejam aplicáveis. Esses conjuntos de tarefas são representados através de *redes de tarefas*, representando restrições sobre sua ordem de execução, que pode ser tanto uma *ordem total* quanto uma *ordem parcial*; esta última permite maior expressividade, mas, em contrapartida, aumenta a complexidade do problema de planejamento.

Assim como no planejamento clássico, cada estado do mundo é representado por um conjunto de átomos, e cada ação corresponde a uma transição de estados determinística. A entrada para o sistema de planejamento inclui um conjunto de operadores similares aos do planejamento clássico e também um conjunto de métodos, cada um deles uma prescrição de como decompor alguma tarefa em um conjunto de *sub-tarefas* (tarefas menores). Um planejador HTN irá aplicar um operador somente se orientado por um método a fazê-lo; portanto, cada método é parte da especificação de uma função de ramificação.

O planejamento hierárquico, também conhecido como planejamento prático, é a técnica mais utilizada para solucionar problemas reais, nas mais diversas áreas de aplicação, como *web* semântica [SPW$^+$04], otimização de navegação de robôs [BHH03] e geração automatizada de cursos baseada em tarefas pedagógicas [Ull05], entre outras.

O poder de expressão do planejamento hierárquico é maior que o do planejamento clássico, permitindo que todo problema clássico seja representado por um hierárquico, onde o contrário não é verdade [EHN94a]. A função do planejador hierárquico é encontrar escolhas de métodos para cada tarefa, decompondo-as de modo com que pré-condições sejam respeitadas. Isso reduz drasticamente o espaço de busca, por utilizar seqüências de ações como escolhas de possíveis componentes do plano, ao invés de escolher ações

individualmente. Basicamente permite maior controle do feixe de busca.

Planejamento HTN tem sido mais largamente utilizado em aplicações práticas que qualquer outra das técnicas aqui discutidas. Isto deve-se parcialmente ao fato de que métodos HTN provêm um meio conveniente de representar “receitas” de soluções de problemas que correspondem ao modo com que especialistas de domínio humanos pensam ao planejar.

Há diferentes formalismos para planejamento hierárquico [Yan90] [EHN94b], pois as contribuições partiram de mais de um planejador, onde pode-se citar: *Nets Of Action Hierarchies* (NOAH) [Sac75], utilizando a linguagem *Semantics Of User Programs* (SOUP), com busca no espaço de planos e planos em ordem parcial; NONLIN [Tat77] com vínculos causais; SIPE-2 [Wil90], permitindo ordens arbitrárias de ações; O-Plan [CT91], permitindo maior modularidade e flexibilidade; *Universal Method Composition Planner* (UMCP) [EHN94c], com um sólido formalismo. Existe também uma abordagem híbrida [KMS98].

Além da decomposição hierárquica, outras abordagens diferentes de controle incluem poda do espaço de busca utilizando lógica temporal, onde pode-se citar: TLPlan [BK00], utilizando Logica de Tempo Linear (*Linear Temporal Logic*) (LTL) [Pnu77], e TAL-planner [DK99], utilizando Lógicas de Ação Temporal (*Temporal Action Logics*) (TAL) [DGKK98].

3.1 Planejamento com Redes Hierárquicas de Tarefas

O formalismo para HTN utilizado neste trabalho é o mesmo de [EHN96]. O uso de *críticas* [EHNT95] é deixado de lado, pois certos tipos de críticas podem comprometer a corretude e a completude do algoritmo.

A linguagem $\mathcal{L}$ para planejamento HTN é uma linguagem de primeira ordem com algumas extensões, similar a NONLIN [Tat77]. O vocabulário de $\mathcal{L}$ é a 6-tupla $\langle \mathcal{V}, \mathcal{C}, \mathcal{P}, \mathcal{F}, \mathcal{T}, \mathcal{N} \rangle$, onde:

- $\mathcal{V}$ é um conjunto infinito de símbolos de variáveis;
- $\mathcal{C}$ é um conjunto finito de símbolos de constantes;
- $\mathcal{P}$ é um conjunto finito de símbolos de predicados;
- $\mathcal{F}$ é um conjunto finito de símbolos de *tarefas primitivas* (denotando ações);
- $\mathcal{T}$ é um conjunto finito de símbolos de *tarefas compostas*;
- $\mathcal{N}$ é um conjunto infinito de símbolos utilizados para rotular tarefas.

Todos os conjuntos de símbolos de $\mathcal{L}$ são mutuamente disjuntos.

Como no planejamento clássico, um estado é uma lista de átomos totalmente instanciados. Os átomos pertencentes à lista são verdadeiros no estado e os que não aparecem são falsos neste estado.

Um novo tipo de símbolo é um *símbolo de tarefa*. Todo símbolo de operador é um símbolo de tarefa primitiva, e existem alguns símbolos de tarefas adicionais chamados de *símbolos de tarefas compostas*. Uma *tarefa* é uma expressão da forma $t(r_1, \ldots, r_k)$ tal

que t é um símbolo de tarefa, e $r_1, \ldots, r_k$ são termos. Se t é um símbolo de operador, então a tarefa é *primitiva*; caso contrário, a tarefa é *não-primitiva*. A tarefa é *totalmente instanciada* se todos os termos são totalmente instanciados; caso contrário, ela é *parcialmente instanciada*. Uma ação $a = (\text{nome}(a), \text{pre}(a), \text{efeitos}(a))$ *executa* uma tarefa primitiva totalmente instanciada t em um estado s se $\text{nome}(a) = t$ e a é aplicável a s.

Definição 3.1.1. Tipos de tarefas:

- Uma *tarefa primitiva* é uma construção sintática da forma $\text{executar}[f(r_1, \ldots, r_k)]$, onde $f \in \mathcal{F}$ e $r_1, \ldots, r_k$ são termos;
- Uma *tarefa meta* é uma construção sintática da forma $\text{obter}[l]$, onde l é um literal;
- Uma *tarefa composta* é uma construção sintática da forma $\text{realizar}[f(r_1, \ldots, r_k)]$, onde $t \in \mathcal{T}$ e $r_1, \ldots, r_n$ são termos.

Tarefas meta e tarefas compostas são ambas também denominadas *tarefas não-primitivas*.

Definição 3.1.2. Uma *rede de tarefas* é uma construção sintática da forma $[(n_1 : \alpha_1) \ldots (n_m : \alpha_m), \phi]$, onde:

- Cada α_i é uma tarefa;
- $n_i \in \mathcal{N}$ é um *rótulo* para α_i (para distinguir diferentes ocorrências de α_i na rede);
- ϕ é uma fórmula booleana construída a partir de:
  - **restrições de substituição de variáveis**, tais como $(v = v')$ e $(v = c)$, onde $v, v' \in \mathcal{V}$ e $c \in \mathcal{C}$;
  - **restrições de ordem**, tais como $(n \prec n')$, onde $n, n' \in \mathcal{N}$, significando que a tarefa rotulada n deve preceder a tarefa rotulada n';
  - **restrições sobre estados**, tais como (n, l), (l, n') e (n, l, n'), onde l é um literal e $v, v' \in \mathcal{V}$, significando que l deve ser verdadeiro no estado imediatamente posterior a n, imediatamente anterior a n e em todos os estados entre n e n', respectivamente.

Para que a decomposição de redes de tarefas possa proceder em qualquer ordem, faz-se necessário um mecanismo de manutenção de histórico. Este deve permitir a representação de restrições que o algoritmo de planejamento ainda não tenha analisado. A manutenção de histórico em HTN é feita através da representação de restrições na rede de tarefas.

Uma rede de tarefas contendo apenas tarefas primitivas é denominada *rede de tarefas primitiva*. Um *plano* é uma seqüência π de tarefas primitivas totalmente instanciadas.

Definição 3.1.3. Um *operador* tem a forma $[\text{operador } f(v_1, \ldots, v_k)(\text{pre} : l_1, \ldots, l_m)(\text{efeitos} : l'_1, \ldots, l'_n)]$, onde:

- f é um símbolo de tarefa primitiva;

- $l_1, \ldots, l_m$ são literais descrevendo as condições para que f seja executável;
- $l'_1, \ldots, l'_n$ são literais descrevendo os efeitos de f;
- $v_1, \ldots, v_k$ são os símbolos de variáveis que aparecem nos literais.

Definição 3.1.4. Um *método* HTN é uma construção da forma (α, d), onde α é uma tarefa não-primitiva e d é uma rede de tarefas. Significa que um modo de relizar a tarefa α é completar a rede de tarefas d, ou seja, realizar todas as sub-tarefas da rede de tarefas sem violar sua fórmula de restrição.

Para cada tarefa meta existe, implicitamente, um método $(\text{obter}[l], [(n : \text{executar}[f]), (l, n)])$ contendo apenas uma tarefa primitiva, fictícia f, sem efeitos, e uma restrição de que o objetivo l seja verdadeiro no estado imediatamente anterior a executar$[f]$.

A descrição de métodos HTN omite a especificação de pré-condições, pois estas são representadas através de restrições do tipo (l, n).

3.2 Planejamento com Redes Simples de Tarefas

Esta seção descreve uma linguagem para uma versão simplificada, um caso restrito de planejamento HTN, denominada Rede Simples de Tarefas (*Simple Task Network*) (STN) [GNT04]. No planejamento STN, associa-se dois tipos de restrições a um método: restrições de ordem e de pré-condições. Representa-se restrições de ordem diretamente na rede de tarefas através das arestas de um dígrafo. Diferente de HTN, não se mantém histórico destas pré-condições na rede de tarefas: ao invés, pré-condições são *forçadas* através da construção das redes de tarefas, de modo a satisfazê-las. Este é o propósito do conjunto de arestas $\{(v, u') \mid u' \in succ_1(u)\}$ na definição 3.2.1. O único tipo de procedimento de planejamento viável para uso desta técnica é a decomposição progressiva [GNT04].

As definições de termos, literais, operadores, ações e planos são as mesmas do planejamento clássico. A definição de $\gamma(s, a)$, o resultado da aplicação de uma ação a a um estado s, é também a mesma do planejamento clássico. No entanto, a linguagem inclui também tarefas, métodos e redes de tarefas, de um modo simplificado.

Definição 3.2.1. Uma *rede simples de tarefas* é um *grafo direcionado* (ou *dígrafo*) acíclico $w = (U, E)$ onde U é o conjunto de nós, E é o conjunto de arestas e cada nó $u \in U$ contém uma tarefa t_u. w é *totalmente instanciada* se todas as tarefas $\{t_u \mid u \in U\}$ são totalmente instanciadas; caso contrário, w é parcialmente instanciada. w é *primitiva* se todas as tarefas $\{t_u \mid u \in U\}$ são primitivas; caso contrário, w é *não-primitiva* (ou *composta*).

As arestas de w definem uma ordem parcial de U. Formalmente, $u \prec v$ se, e somente se, existe um caminho de u para v. Se a ordem parcial é também uma ordem total, dizemos que w é *totalmente ordenada*. Neste caso, algumas vezes dispensa-se a notação de grafo para w e, ao invés, escreve-se w como a seqüência de tarefas $w = \langle t_1, \ldots, t_k \rangle$, onde t_1 é a tarefa no primeiro nó de U, t_2 é a tarefa no segundo nó de U e assim por diante. Se w é totalmente instanciada, primitiva, totalmente ordenada e existem ações $a_1, \ldots, a_k$ cujos nomes são $t_1, \ldots, t_k$, então w corresponde ao plano $\pi_w = \langle a_1, \ldots, a_k \rangle$.

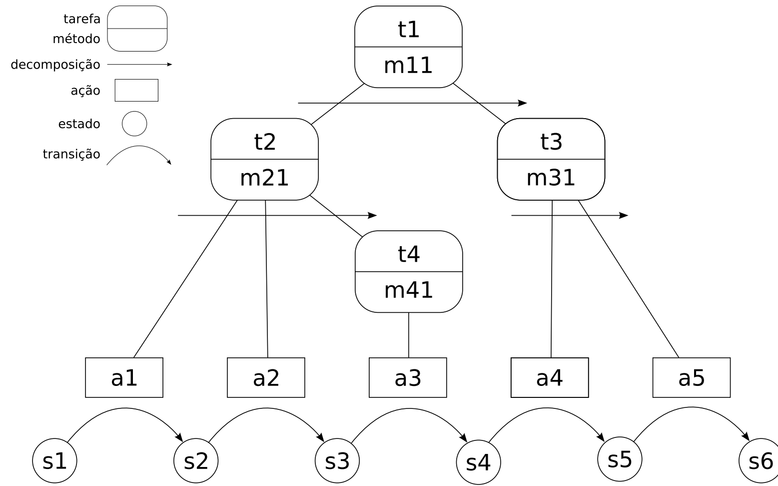

Figura 3.1: Exemplo de decomposição de tarefas em ordem total

Definição 3.2.2. Um *método STN* é a 4-tupla $m = \langle \text{nome}(m), \text{tarefa}(m), \text{pre}(m), \text{rede}(m) \rangle$, onde:

- nome(m) é o nome do método, uma expressão do tipo $m(x_1, \ldots, x_k)$, onde $x_1, \ldots, x_k$ são todos os símbolos de variáveis que ocorrem em m;
- tarefa(m) é uma tarefa não-primitiva;
- pre(m) é um conjunto de literais especificando as *pré-condições* do método;
- rede(m) é uma rede de tarefas contendo as *sub-tarefas* de m.

Na definição (3.2.2), nome(m) tem o mesmo propósito do nome de um operador de planejamento clássico; permite referência desambígua a instâncias de substituição do método, sem que seja necessário descrever explicitamente as pré-condições e efeitos. tarefa(m) diz a que tipo de tarefa m pode ser aplicado, pre(m) especifica que condições o estado atual deve satisfazer para que m seja aplicável, e rede(m) especifica as sub-tarefas a realizar para que complete tarefa(m).

Um método m é *totalmente ordenado* se rede(m) é totalmente ordenada. Neste caso, ao invés de utilizar um dígrafo, é possível utilizar uma lista sub-tarefas$(m) = \langle t_1, \ldots, t_k \rangle$ como representação compacta de:

$$\text{rede}(m) = (\{u_1, \ldots, u_k\}, \{(u_1, u_2), (u_2, u_3), \ldots, (u_{k-1}, u_k)\}).$$

Definição 3.2.3. Uma instância de método m é *aplicável* a um estado s se $\text{pre}^+(m) \subseteq s$ e $\text{pre}^-(m) \cap s = \emptyset$.

Um planejador STN efetua uma busca progressiva no espaço de estados, guiada pela decomposição da rede de tarefas inicial (Fig. 3.1). A decomposição aplica métodos a

tarefas sem predecessores, até o nível de tarefas primitivas, e busca extrair planos válidos (Algoritmo 3).

Quando um método m é utilizado para decompor uma tarefa t, tal que t é parte de um nó u de uma rede de tarefas w, obtemos a nova rede de tarefas $\delta(w, u, m, \sigma)$ (Def. 3.2.4). Intuitivamente, u é removido de w, uma cópia w_u de sub-tarefas(m) é inserida em w no lugar de u, e toda restrição de ordem que se aplicava a u passa a valer para todo nó de w_u.

Definição 3.2.4. Seja $w = (U, E)$ uma rede de tarefas, u um nó de w sem predecessores, e m um método relevante para t_u sob alguma substituição σ. Seja $succ(u)$ o conjunto de todos os sucessores imediatos de u, isto é, $succ(u) = \{u' \in U \mid (u, u') \in E\}$, e $succ_1(u)$ o conjunto de sucesores imediatos para os quais u é o único predecessor. Seja (U', E') o resultado da remoção de u e todas as arestas que contenham u e seja (U_m, E_m) uma cópia de rede(m). Se (U_m, E_m) não é vazio, então o resultado da decomposição de u em w sob σ é o conjunto de redes de tarefas:

$$\delta(w, u, m, \sigma) = \{(\sigma(U' \cup U_m), \sigma(E_v)) \mid v \in \text{sub-tarefas}(m)\},$$

onde

$$E_v = E_m \cup (U_m \times succ(u)) \cup \{(v, u') \mid u' \in succ_1(u)\}.$$

Caso contrário, $\delta(w, u, m, \sigma) = \{(\sigma(U'), \sigma(E'))\}$.

Devido a essa estratégia progressiva o planejador sempre mantém controle do estado atual do mundo (em tempo de planejamento) e pode se beneficiar desta informação (para extração de heurísticas, calcular custos de planos, entre demais usos). Também é possível ver STN como uma busca regressiva, onde as tarefas correspondem aos objetivos e a decomposição age como regressão.

O algoritmo para planejamento hierárquico em ordem parcial (Algoritmo 3) baseia-se nos três casos em que um plano $\pi_w = (a_1, \ldots, a_k)$ é uma solução para um problema $\mathcal{P} = (s_0, w, \mathcal{D}, \mathcal{M})$:

1. Se w é vazia, então π é uma solução para $\mathcal{P}$ se π é vazio, ou seja, $k = 0$;

2. Se existe um nó de tarefa primitiva $u \in w$ que não tem predecessores em w, então π é uma solução para $\mathcal{P}$ se a_1 é aplicável a t_u em s_0 e o plano $\pi' = (a_2, \ldots, a_k)$ é uma solução para o problema $\mathcal{P}' = (\gamma(s_0, a_1), w \setminus \{u\}, \mathcal{D}, \mathcal{M})$. Intuitivamente, $\mathcal{P}'$ é o problema produzido pela execução da primeira ação de π e remoção da tarefa correspondente de w;

3. Existe um nó de tarefa não-primitiva $u \in w$ que não tem predecessores em w. Supondo que existe uma instância m de algum método em $\mathcal{M}$ tal que m é relevante para t_u e aplicável em s_0, então π é uma solução para $\mathcal{P}$ se existe uma rede de tarefas $w' \in \delta(w, u, m, \sigma)$ tal que π é uma solução para $\mathcal{P}' = (s_0, w', \mathcal{D}, \mathcal{M})$.

Se π é uma solução para $\mathcal{P} = (s_0, w, \mathcal{D}, \mathcal{M})$, então para cada nó de tarefa $u \in U$ existe uma *árvore de decomposição* cujas folhas são as ações de π.

Algoritmo 3: DecomposiçãoOrdemParcial $(s, w, \mathcal{D}, \mathcal{M})$

Entrada: Estado inicial s, Rede de tarefas w, Domínio $\mathcal{D}$, Conjunto de métodos $\mathcal{M}$

Saída: Plano π

```
1  início
2    se w = ∅ então devolva plano vazio;
3    não-deterministicamente escolha u ∈ w, sem predecessores em w;
4    se t_u é uma tarefa primitiva então
5      A ← {(a, σ) | a é uma ação de D, σ é uma substituição tal que
         nome(a) = σ(t_u) e a é aplicável a s};
6      se a = ∅ então devolva falha;
7      não-deterministicamente escolha (a, σ) ∈ A;
8      π ←DecomposiçãoOrdemParcial(γ(s, a), σ(w \ {u}), D, M);
9      se π = falha então devolva falha;
10     senão devolva a.π;
11   senão
12     M ← {(m, σ) | m é um método totalmente instanciado de M, σ é uma
         substituição tal que nome(m) = σ(t_u) e m é aplicável a s};
13     se a = ∅ então devolva falha;
14     não-deterministicamente escolha (m, σ) ∈ M;
15     não-deterministicamente escolha w' ∈ δ(w, u, m, σ);
16     devolva DecomposiçãoOrdemParcial(s, w', D, M);
17 fim
```

3.3 Extensões

É relativamente fácil incorporar extensões ao planejamento hierárquico, visto que os procedimentos de decomposição STN planejam progressivamente a partir do estado inicial. Têm conhecimento em tempo de planejamento do estado atual do mundo a cada passo do planejamento. Isto torna fácil executar procedimentos arbitrários envolvendo o estado atual, incluindo computações numéricas complexas, inferência axiomática e chamadas a pacotes de software específicos de domínio (tal como sistemas de bancos de dados e raciocinadores probabilísticos). Dentre os problemas envolvidos na incorporação de técnicas adicionais, pode-se citar:

- **Símbolos de Funções.** Se deixamos a linguagem $\mathcal{L}$ de planejamento HTN conter símbolos de funções, então os argumentos de um átomo ou tarefa não são mais restritos a ser símbolos constantes e símbolos variáveis. Ao invés, estes podem ser termos arbitrariamente complexos, tornando o planejamento indecidível.

- **Axiomas.** Para incorporar inferência axiomática, é necessário utilizar um provador de teoremas como uma sub-rotina do procedimento de planejamento. A abordagem mais simples é restringir os axiomas a cláusulas de Horn e utilizar um provador de teoremas baseado nestas.

- **Procedimentos Anexados.** Pode-se modificar o algoritmo de avaliação de pré-condições (que, no caso de se ter também inferência axiomática incorporada, seria o provador de teoremas em cláusulas de Horn) para reconhecer que certos termos ou símbolos de predicados devem ser avaliados através do uso de procedimentos anexados ao invés do uso do mecanismo normal de prova de teoremas.

3.3.1 Extensões Adicionais

- **Efeitos de Alto Nível.** Alguns planejadores HTN permitem ao usuário declarar na descrição do domínio de planejamento que várias tarefas não-primitivas, ou os métodos para estas tarefas, alcançarão vários efeitos.

- **Pré-condições Externas.** Supondo que para decompor alguma tarefa t, decide-se usar algum método m. Além disso, supondo que existe alguma condição c tal que, não importando qual sub-plano seja produzido abaixo de m, pelo menos uma ação no sub-plano irá requerer c como pré-condição e o sub-plano não irá conter nenhuma ação que alcança c. Neste caso, para que o sub-plano apareça como parte de um plano solução π, a pré-condição c deve de algum modo ser alcançada em algum outro lugar em π. Portanto, dizemos que a pré-condição c é *externa* ao método m.

- **Tempo.** É possível generalizar processos de decomposição de tarefas para fazer certos tipos de planejamento temporal, por exemplo, lidar com ações que têm durações de tempo e podem se sobrepor entre si.

- **Grafos de Planejamento.** Esta extensão é diferente das demais por ser uma extensão do algoritmo de planejamento mas não da representação HTN: é possível modificar o algoritmo de decomposição para fazer uso de grafos de planejamento. O algoritmo modificado gera uma árvore de decomposição e um grafo de planejamento. O tamanho do grafo de planejamento é reduzido através da geração de somente aquelas ações que fazem par com tarefas na árvore de decomposição, e o tamanho da árvore de decomposição é reduzido através da decomposição de tarefas somente se o grafo de planejamento pode verificar todos os seus predecessores. [MK98]

3.3.2 Planejadores Hierárquicos Existentes

A família *Simple Hierarchical Ordered Planner* (SHOP) [NCLMA99] de planejadores permite planejamento hierárquico utilizando o caso restrito de HTN denominado STN, em ordem total (SHOP) e em ordem parcial (SHOP2) [NAI+03]. Dentre suas propriedades interessantes está o fato de que a busca é feita progressivamente, de modo que o planejador conheça sempre o estado atual. Isto torna possível um maior poder de expressão: axiomas, manipulação simbólica e numérica, execução de procedimentos externos durante o planejamento.

JSHOP2 [ON04] é uma implementação em Java do planejador SHOP2 (sendo este último escrito em Common Lisp). Adicionalmente, o JSHOP2 possui uma fase de compilação, que, a partir da descrição de um domínio, gera código Java de um planejador

dependente do domínio. Este planejador gerado é utilizado para resolver problemas específicos deste domínio e se aproveita de técnicas na geração do código para torná-lo mais eficiente quanto às suas estruturas internas de representação. Resultados experimentais sugerem que o ganho obtido pela técnica de compilação é de uma ordem de magnitude [ON04].

Os planejadores SHOP têm sido utilizados em aplicações diversas [NAI+05], como planejamento de evacuação, combate a incêndio em florestas, integração de sistemas de software, controle de veículos autônomos e seleção de materiais para manufatura, além de diversar aplicações à *web* semântica [WPS+03] [WSH+03] [SPW+04] [AKN05].

O modelo de decomposição de tarefas de JSHOP2 é a base para a função EFETUA-DECOMPOSIÇÃO (Algoritmo 12) do algoritmo do planejador YOYO.

3.3.3 Vantagens e Desvantagens

Comparados aos planejadores clássicos, a principal vantagem da abordagem HTN é sua sofisticada representação de conhecimento e raciocínio sob ações compostas. Eles podem representar e solucionar uma variedade de problemas de planejamento; com um bom conjunto de métodos de decomposição, eles podem solucionar problemas de planejamento clássico com maior eficiência. A principal desvantagem de planejadores HTN é que autores de domínios necessitam especificar um conjunto de métodos. O planejamento clássico necessita apenas do conjunto de operadores de planejamento.

Comparando HTNs com regras de controle [BK00], é difícil dizer que tipo de técnica de controle é mais efetiva. Planejadores hierárquicos têm sido mais utilizados em aplicações práticas, mas isto deve-se parcialmente ao fato de serem mais antigos. HTNs dão a um planejador informação sobre quais opções considerar, e regras de controle sobre quais opções *não* considerar. Os dois tipos de informação são úteis em diferentes situações e combiná-las pode ser vantajoso [SBNM02].

Capítulo 4

Planejamento sob Incerteza

Agentes raramente têm acesso à informação completa sobre o ambiente. Assim, um agente inteligente deve ser capaz de agir sob *incerteza*. A *decisão racional* depende tanto da importância relativa de vários objetivos a serem alcançados quanto da probabilidade de que (e do grau em que) estes serão alcançados. A decisão sobre o compromisso entre exatidão e a utilidade da teoria do agente, todavia, parece exigir ela própria um raciocínio sobre a incerteza [RN03].

4.1 Suposições

O *planejamento sob incerteza* relaxa as suposições clássicas de *determinismo*, *observabilidade total* e *metas de alcançabilidade* descritas na seção 2.2.

- Quanto ao efeito das ações, um problema de planejamento pode ser *determinístico*, *probabilístico*, ou *não-determinístico* (também chamado de *"possibilístico"* [vR01] ou sob *incerteza Knightiana* [Kni21] [TCdB07]).
- A observabilidade pode ser *total*, *parcial* ou *nula*.
- Metas podem ser de *alcançabilidade*, *estendidas* ou de *maximização de utilidade* (ou *otimização*).

4.1.1 Determinismo nos Efeitos de Ações

Determinismo é uma visão simplificada do mundo que supõe que este evolui ao longo de um único caminho, totalmente previsível dado o conhecimento de seu histórico. Em alguns casos, modelar um sistema de forma determinística pode ser uma abstração útil. Segundo a visão determinística, é necessário apenas estender propriamente um modelo para se prever tudo o que venha a acontecer no ambiente modelado. Com um modelo perfeito, o lançamento de um dado seria totalmente determinado, mas isto é uma suposição irrealista e impraticável.

A suposição de ações *não-determinísticas* assume uma postura mais realista. É mais útil modelar todos os possíveis resultados do lançamento de um dado e raciocinar sobre múltiplas possibilidades, sendo que nenhuma delas é certa. Pode ser útil modelar o fato

de que um componente possa parar ou falhar e planejar para “tratamento de exceções” ou mecanismos de recuperação.

O planejamento em domínios não-determinísticos apresenta uma característica que o difere do planejamento clássico: um plano pode resultar em muitos caminhos diferentes de execução. Como conseqüência, não é possível especificar planos seqüenciais, mas políticas que mapeiam estados em ações.

O indeterminismo pode ser também representado pela associação de probabilidades aos possíveis resultados de ações. Isto permite modelar o fato de que alguns resultados têm maior chance de ocorrer que outros.

Em aplicações mais complexas, pode ser muito difícil obter as distribuições de probabilidades corretas, tanto por falta de dados estatísticos quanto pelo fato de que, em alguns casos, a meta é, garantir que certas condições ocorram. Nesse trabalho, o interesse é encontrar políticas para domínios com incerteza nos efeitos das ações, sem conhecer as distribuições de probabilidade.

4.1.2 Observabilidade Parcial

Em várias aplicações, o estado de um sistema é apenas parcialmente visível em tempo de execução e, como conseqüência, diferentes estados do sistema são indistingüíveis do ponto de vista do controlador.

Muitas aplicações apresentam uma ou mais das seguintes situações:

- algumas variáveis do sistema podem nunca ser observáveis pelo controlador,
- algumas variáveis podem ser observáveis somente em alguns estados ou após alguma ação de sensoriamento tenha sido executada.

A principal conseqüência técnica da observabilidade parcial é que observações devolvem conjuntos de estados ao invés de estados únicos. Isto torna o espaço de busca não mais o conjunto de estados do domínio, mas seu conjunto potência. Adicionalmente, no caso do planejamento probabilístico, observações podem devolver distribuições de probabilidades sobre conjuntos de estados, o que torna o espaço de busca infinito. Este trabalho trata apenas de problemas de planejamento sob observabilidade total.

4.2 Técnicas de Planejamento sob Incerteza

Existem basicamente 4 métodos de planejamento para lidar com indeterminismo [RN03]. Domínios podem ser classificados quanto à sua quantidade de indeterminismo em:

Indeterminismo Limitado, onde as ações podem ter efeitos imprevisíveis, mas os efeitos possíveis podem ser listados nos axiomas de descrição de ações. Existem dois tipos conhecidos de planejamento sob indeterminismo limitado:

Planejamento Conformante, ou planejamento sem sensores, constrói planos seqüenciais padrão que devem ser executados sem percepção. Visa encontrar soluções fortes sob observabilidade nula, ou seja, garante atingir um estado

meta apesar do indeterminismo do domínio, para todos os possíveis estados iniciais e todos os efeitos incertos de ações. Nenhuma ação deve ser executada em estados que não satisfazem as pré-condições e qualquer estado que resulte da execução do plano deve ser levado a um estado meta.

Planejamento Contingente, ou condicional, lida com o indeterminismo limitado construindo um plano condicional com diferentes ramificações para as diferentes contingências que poderiam surgir. Permite superar a incerteza até certo ponto, mas apenas se ações de sensoriamento puderem obter a informação necessária, e somente se não houver muitas contingências diferentes. Esse trabalho lida com este tipo de planejamento.

Indeterminismo Ilimitado, onde o conjunto de pré-condições ou efeitos possíveis é desconhecido ou grande demais para ser enumerado completamente. Existem duas técnicas para lidar com esse tipo de indeterminismo:

Monitoramento e Replanejamento, onde o agente pode usar qualquer das técnicas de planejamento precedentes para construir um plano, mas também utiliza o monitoramento de execução para julgar se o plano tem uma provisão referente à situação real ou precisa ser revisto. Presumivelmente, tais planos funcionarão a maior parte do tempo, mas surgirão problemas quando os acontecimentos contrariarem a teoria do agente, obrigando-o a buscar um novo plano;

Planejamento Contínuo, projetado para persistir ao longo de inúmeras execuções, pode lidar com o abandono e criação de novas metas, respeitando prioridades entre elas.

4.3 Modelo

Definição 4.3.1. O formalismo para planejamento não-determinístico baseia-se em um *sistema de transição de estados não-determinístico* (Fig. 4.1), uma 3-tupla $\Sigma'' = \langle \mathcal{S}, \mathcal{A}, \gamma \rangle$, onde:

- $\mathcal{S} = \{s_1, s_2, \ldots\}$ é o conjunto finito ou recursivamente enumerável de estados;
- $\mathcal{A} = \{a_1, a_2, \ldots\}$ é o conjunto finito ou recursivamente enumerável de ações;
- $\gamma : \mathcal{S} \times \mathcal{A} \rightarrow 2^{\mathcal{S}}$ é a função de transição de estados.

Planos seqüenciais não são suficientemente expressivos para representar contingências. Assim, utiliza-se a noção de *política* π, que diz qual é a melhor ação, dado um determinado estado. Uma função parcial de estados a ações. Geralmente, são representadas através de um conjunto $\pi = \{(s_i, a_i)\}_{i=1}^{k}$, onde $k \geq 0$ é seu tamanho. Os estados para os quais π tem alguma ação associada é denotado pelo conjunto $S_\pi = \{s \mid (s, a) \in \pi\}$; Políticas são ditas *determinísticas* se mapeiam apenas uma ação para cada estado.

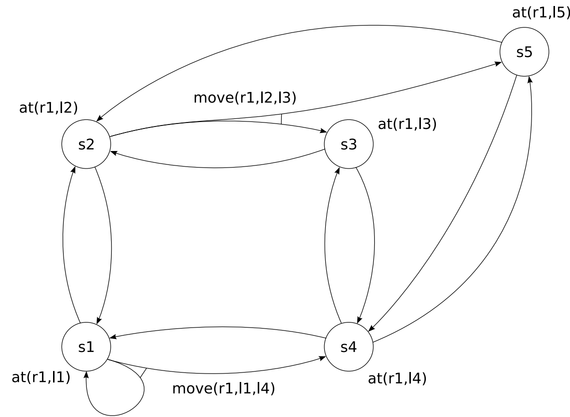

Figura 4.1: Sistema de transição de estados não-determinístico

Definição 4.3.2. A *estrutura de execução* Σ_π de uma política π é um grafo direcionado (ou *dígrafo*) no qual nós representam estados alcançáveis através de ações de π e arestas são possíveis transições de estados causadas por ações de π. Se e somente se existe um caminho em Σ_π de s_1 até s_2, então s_1 é π-*ancestral* de s_2 e s_2 é π-*descendente* de s_1.

4.4 Planejamento Forte, Forte Cíclico e Fraco

Políticas podem ser caracterizadas nas seguintes classes de soluções [CPRT03] [dLP06]:

Soluções Fortes são políticas que atingem a meta, garantidamente, a despeito do não-determinismo, ou seja, todos os caminhos no dígrafo Σ_π são finitos e atingem a meta. Formalmente, uma política π é uma solução *forte* para $\mathcal{P}$ se e somente se Σ_π é acíclico e todo terminal em Σ_π está em S_g;

Soluções Fortes Cíclicas são políticas que atingem a meta, supondo que os ciclos sejam eventualmente interrompidos. Formalmente, uma política π é uma solução *forte cíclica* para $\mathcal{P}$ se e somente se existe, para cada nó em Σ_π, um caminho até um terminal em Σ_π e todo terminal em Σ_π está em S_g;

Soluções Fracas são políticas que podem atingir a meta, mas não garantidamente. Formalmente, uma política π é uma solução *fraca* para $\mathcal{P}$ se e somente se, para cada estado em S_0, existe um caminho que leva a um estado em S_g que é terminal em Σ_π.

Toda solução forte é também uma solução forte cíclica e toda solução forte cíclica é também uma solução fraca [CPRT03]. Soluções estritamente fracas e soluções fortes correspondem a dois casos extremos de satisfação de metas simples. Intuitivamente, soluções estritamente fracas correspondem a "planos otimistas" e soluções fortes correspondem a "planos seguros". Entretanto, na prática, há casos em que soluções estritamente fracas não são aceitáveis e soluções fortes não existem. Nestes casos, soluções estritamente fortes cíclicas podem ser uma alternativa viável.

Nos algoritmos seguintes, $S_{abertos}$ representa um conjunto de estados que ainda não foram considerados pelo planejador, permitindo com que possam operar sobre políticas parciais, ainda em construção.

4.4.1 Planejamento Forte

Definição 4.4.1. A *pré-imagem forte* de um conjunto S de estados é definida como:

$$\textsc{PréImagemForte}(S) = \{(s,a) \mid \gamma(s,a) \neq \emptyset \wedge \gamma(s,a) \subseteq S\}.$$

Intuitivamente, a pré-imagem forte de um conjunto de estados S é o conjunto de todo estado s, tal que todos os seus sucessores (resultantes da aplicaçao de a) encontrem-se em S.

No planejamento forte, uma política π deve apresentar um caminho acíclico até um estado meta para todo estado alcançável a partir dos estados iniciais. Portanto, a estrutura de execução Σ_π deve ser acíclica. O algoritmo que verifica se uma política é forte (Algoritmo 4) efetua uma busca regressiva, começando a partir dos estados meta e e estados em aberto, aplicando a função de pré-imagem forte (Definição 4.4.1) até alcançar os estados iniciais. A cada iteração são removidos pares de estado-ação de π que já foram verificados e pertencem à pré-imagem forte dos estados meta. Ao final da busca, se ainda existe algum par de estado-ação na política π, então a política induz um ciclo na estrutura de execução Σ_π e, portanto, não é uma solução forte.

Algoritmo 4: $\textsc{VerificaForte}(\pi, S_{abertos}, S_g, S_0)$

Entrada: Política π, Conjunto de estados em aberto $S_{abertos}$, Conjunto de estados meta S_g, Conjunto de estados iniciais S_0

```
1 início
2    S′ ← ∅; S ← S_g ∪ S_abertos;
3    enquanto S′ ≠ S faça
4       S′ ← S;
5       S ← S ∪ {s′ | (s′, a) ∈ π e γ(s′, a) ⊆ S};
6       π ← π \ {(s, a) | s ∈ S e (s, a) ∈ π};
7    se S_0 ⊆ S e π = ∅ então devolva verdadeiro;
8    devolva falso;
9 fim
```

4.4.2 Planejamento Forte Cíclico

O algoritmo que determina se uma política é forte cíclica (Algoritmo 5) efetua uma busca regressiva em Σ_π. Consiste em certificar-se de que todo par estado-ação na política parcial π é verificado pela busca. Ao final da busca, se existe algum par de estado-ação não removido de π, então π viola a suposição do planejamento forte-cíclico de que ciclos sejam eventualmente interrompidos, ou seja, existe um ciclo induzido por π a partir do qual não há possibilidade de alcançar estados meta ou em aberto.

Algoritmo 5: VERIFICAFORTECÍCLICO($\pi, S_{abertos}, S_g, S_0$)

Entrada: Política π, Conjunto de estados em aberto $S_{abertos}$, Conjunto de estados meta S_g, Conjunto de estados iniciais S_0

1 **início**
2 $S' \leftarrow \emptyset; S \leftarrow S_g \cup S_{abertos}$;
3 **enquanto** $S' \neq S$ **faça**
4 $S' \leftarrow S$;
5 $S \leftarrow S \cup \{s' \mid (s', a) \in \pi \text{ e } S \cap \gamma(s', a) \neq \emptyset\}$;
6 $\pi \leftarrow \pi \setminus \{(s, a) \mid s \in S \text{ e } (s, a) \in \pi\}$;
7 **se** $S_0 \subseteq S$ *e* $\pi = \emptyset$ **então devolva** *verdadeiro*;
8 **devolva** *falso*;
9 **fim**

4.4.3 Planejamento Fraco

Definição 4.4.2. A *pré-imagem fraca* de um conjunto S de estados é definida como:

$$\text{PRÉIMAGEMFRACA}(S) = \{(s, a) \mid \gamma(s, a) \neq \emptyset \wedge \gamma(s, a) \cap S \neq \emptyset\}.$$

Intuitivamente, a pré-imagem fraca de um conjunto de estados S é o conjunto de todo estado s, tal que pelo menos um de seus sucessores (resultantes da aplicaçao de a) encontre-se em S.

O algoritmo para verificar se uma política é fraca (Algoritmo 6) efetua uma busca por Σ_π por caminhos acíclicos, que, a partir de cada estado inicial $s \in S_0$ de $\mathcal{P}$, levam a um estado meta. Para isso, o algoritmo utiliza uma busca regressiva, a partir dos estados meta e dos estados em $S_{abertos}$, até os estados iniciais S_0. Esta busca regressiva é baseada na função de pré-imagem fraca (Definição 4.4.2). A busca termina quando todos os estados que podem ser alcançados (através da pré-imagem) do objetivo e de $S_{abertos}$ estão em S. A este ponto, S contém todos os estados a partir dos quais existe um caminho acíclico até um estado meta. Depois disso, simplesmente verifica se os estados iniciais estão todos em S. Em caso afirmativo, a política parcial π não viola os requisitos do planejamento fraco.

4.5 Planejamento baseado em Verificação de Modelos

Planejamento baseado em Verificação de Modelos [GNT04] é uma abordagem para planejamento sob incerteza que lida com indeterminismo, observabilidade parcial e metas estendidas. A idéia principal é resolver problemas de planejamento na teoria dos modelos. Dados um sistema de transição de estados e uma fórmula temporal, planejamento baseado em verificação de modelos gera planos que "controlam" a evolução do sistema tal que todos os comportamentos do modelo façam com que a fórmula seja verdadeira.

A principal vantagem do planejamento baseado em verificação de modelos é a habilidade de planejar sob incerteza de modo prático. O indeterminismo leva à necessidade de lidar com diferentes resultados de ações, ou seja, diferentes transições possíveis. Observabilidade parcial leva à necessidade de lidar com observações que correspondem a mais

Algoritmo 6: VERIFICAFRACO($\pi, S_{abertos}, S_g, S_0$)

Entrada: Política π, Conjunto de estados em aberto $S_{abertos}$, Conjunto de estados meta S_g, Conjunto de estados iniciais S_0

1 **início**
2 $S'' \leftarrow \emptyset; S \leftarrow S_g \cup S_{abertos}$;
3 **enquanto** $S'' \neq S$ **faça**
4 $S'' \leftarrow S$;
5 $S' \leftarrow \{s' \mid (s', a) \in \pi \text{ e } \gamma(s', a) \cap S \neq \emptyset\}$;
6 $\pi \leftarrow \pi \setminus \{(s, a) \mid s \in S \text{ e } (s, a) \in \pi\}$;
7 $S \leftarrow S \cup S''$;
8 **se** $S_0 \subseteq S$ **então devolva** *verdadeiro*;
9 **devolva** *falso*;
10 **fim**

de um estado. Ao invés de deliberar sobre estados individuais, o planejamento baseado em verificação de modelos efetua a busca sobre conjuntos de estados e conjuntos de transições de uma só vez. Estes conjuntos são representados e manipulados simbolicamente: conjuntos extremamente grandes de estados ou imensos conjuntos de transições são geralmente representados de forma bem compacta, e em alguns casos são manipulados com um baixo custo computacional.

4.5.1 Verificação de Modelos Simbólicos

Modelos realistas necessitam de um enorme número de estados. Uma abordagem simbólica [JEK+90] para verificação de modelos utiliza fórmulas proposicionais para representar, de modo compacto, modelos de estados finitos e transformações sobre fórmulas. A exploração do espaço de estados utiliza então estas formas compactas para ganhar eficiência.

Diferente dos algoritmos tradicionais, que verificam estados individualmente, os algoritmos de verificação de modelos simbólicos efetuam busca regressiva sobre conjuntos de estados e transições. É demonstrado que, em alguns casos, o uso destas representações compactas melhora exponencialmente

Um tratado sobre planejamento em domínios não-determinísticos, utilizando verificação de modelos simbólicos, pode ser encontrado em [Rov01].

Capítulo 5

Planejamento Hierárquico sob Incerteza

Em domínios de planejamento determinísticos, muito trabalho tem sido feito com o objetivo de melhorar a eficiência de planejadores, procurando preveni-los de visitar estados não promissores. A maior parte desses trabalhos concentra-se em planejadores que utilizam busca progressiva no espaço de estados, tais como HSP [BG99], FF [HN01], TLPlan [BK00], TALplanner [DK99], e SHOP2 [NAI⁺03]. Estes planejadores conhecem sempre o estado atual do mundo, o que facilita o uso de algumas técnicas poderosas de poda do espaço de busca, especialmente em planejadores que permitem especificar formas de poda específicas de domínio.

Em [KN04], é apresentado um meio de transportar estas melhorias de eficiência para planejamento não-determinístico. Dentre as contribuições de [KN04], pode-se citar uma técnica geral de "indeterminizar" planejadores progressivos, ou seja, estendê-los para que possam trabalhar em domínios não-determinísticos. Esta forma de transformação preserva a corretude e a completude do algoritmo original. Em seguida, é descrito YoYo [KNPT05], que incorpora ao planejador hierárquico SHOP2 essa técnica, preparando-o para lidar com planejamento não-determinístico. Além disso, YoYo utiliza técnicas de verificação de modelos simbólicos para conseguir um maior ganho de desempenho.

5.1 "Indeterminizando" Planejadores Progressivos

Em [KN04], é descrita uma técnica geral para produzir, a partir de planejadores progressivos para domínios determinísticos, versões "indeterminizadas" destes, ou seja, transformá-los em planejadores que lidam com problemas não-determinísticos. A técnica apresentada produz novos planejadores que tentam devolver soluções cíclicas fortes, se existirem, para problemas não-determinísticos.

A base para a técnica é coposta de dois procedimentos abstratos de planejamento: Planejamento Progressivo (*Forward Chaining Planning*) (FCP) (Algoritmo 7), para domínios determinísticos, e um procedimento correspondente, com alguas modificações, *Non-Deterministic Forward Chaining Planning* (ND-FCP) (Algoritmo 8), para domínios não-determinísticos.

O propósito da utilização de procedimentos abstratos é permitir demonstrar a trans-

```
Algoritmo 7: FCP (s_0, g, D, α)
   Entrada: Estado inicial s_0, Objetivo g, Domínio D, Função de seleção de ações α
   Saída: Plano π
 1 início
 2   π ← ∅; s ← s_0;
 3   repita
 4     se s satisfaz g então devolva π;
 5     A ← {(s, a) | a é uma instância total de um operador em D, a é aplicável a
       s, e a ∈ α(s)};
 6     se A = ∅ então devolva falha;
 7     não-deterministicamente escolha (s, a) ∈ A;
 8     π ← π ∪ {(s, a)};
 9     s ← γ(s, a);
10   fim
11 fim
```

formação de forma mais clara: em suma, como planejadores progressivos podem ser preparados, de forma sistemática, para lidar com ações não-determinísticas. Formalmente, um planejador Λ é uma *instância* de FCP se existe uma substituição de certas partes do procedimento FCP de modo que este se comporte como Λ. Se um planejador Λ pode ser descrito como uma instância do procedimento FCP (um *algoritmo progressivo determinístico*), a instância correspondente ND-Λ de ND-FCP (*algoritmo progressivo não-determinístico*) será a *"indeterminização"* cíclica forte de Λ. ND-SHOP2 [KN04] é resultante da transformação de SHOP2 [NAI$^+$03].

Em instâncias de FCP, a relação de satisfação de objetivos e a função α variam, de acordo com o algoritmo de planejamento Λ usado como base. A relação s satisfaz g depende de Λ e pode ir além da simples verificação $s \in S_g$. A *função de seleção de ações* α é utilizada para podar o espaço de busca: devolve um sub-conjunto das ações a aplicáveis a s, que são denominadas ações que *satisfazem* $\alpha(s)$. Alguns exemplos de instâncias de FCP podem ser:

- No caso mais simples, planejamento STRIPS, $\alpha(s)$ apenas devolve as ações a aplicáveis a s e a relação de satisfação $s \in S_g$ se mantém, onde g é substituído por S_g.

- Já em SHOP2 [NAI$^+$03], a busca é controlada pelo conjunto de métodos utilizados para decompor sub-tarefas, e $\alpha(s)$ nos procedimentos denota o conjunto de ações que podem ser produzidas por estes métodos. Em casos com metas puramente de decomposição, qualquer s, se alcançado através das decomposições, satisfaz g. Em outras palavras, se a rede de tarefas w alcançada junto a s é vazia, então s satisfaz g.

- Outro planejador que pode ser utilizado é TLPlan [BK00], que tem a busca controlada por uma fórmula ϕ especificada em LTL [Pnu77]. Para cada estado s, $\alpha(s)$ é o conjunto de todas as ações a aplicáveis a s, tal que o estado sucessor $\gamma(s, a)$ satisfaça a fórmula.

Um ponto importante a notar é que a função α pode vir a carregar consigo um estado interno. No caso do planejamento hierárquico, a manutenção da hierarquia de decomposição corrente é de responsabilidade de α, que pode mudar durante a escolha não-determinística. Por se tratar de procedimentos abstratos, FCP e ND-FCP mantêm este fato implícito.

FCP (Algoritmo 7) começa com um estado inicial s_0 e efetua uma busca progressiva (Algoritmo 1), explorando outros estados, sucessivamente, através da escolha e aplicação de ações. A busca termina quando um estado que satisfaz a meta g é alcançado. Ao invés de examinar todas as ações aplicáveis a s, FCP considera somente o sub-conjunto destas que satisfaz $\alpha(s)$.

Como FCP, ND-FCP (Algoritmo 8) também planeja progressivamente a partir dos estados em S_0, mas o critério de sucesso é mais complexo. Em FCP, um plano π é uma solução se seu estado final satisfaz a condição meta g. Para que uma política em ND-FCP seja uma solução, *todo* estado na estrutura de execução de π deve ter pelo menos um π-descendente que satisfaz a meta. Estados que já têm um caminho em Σ_π que leve a um estado meta são denotados estados *resolvidos*. Do contrário, são chamados de estados *em aberto*. A busca termina quando todos os estados alcançáveis são resolvidos, se existir uma política solução. Caso contrário, devolve *falha*.

Algoritmo 8: ND-FCP $(S_0, g, \mathcal{D}', \alpha')$

Entrada: Conjunto inicial S_0, Objetivo g, Domínio $\mathcal{D}'$, Função de seleção de ações α'
Saída: Plano π

```
1  início
2    π ← ∅; S ← S_0; S_resolvidos ← ∅;
3    repita
4      se S = ∅ então devolva π;
5      selecione um estado s ∈ S e remova-o de S;
6      se s satisfaz g então insira s em S_resolvidos;
7      senão se s ∉ S_π então
8        A ← {(s, a) | a é uma instância total de um operador em D', a é
           aplicável a s, e a ∈ α'(s)};
9        se A = ∅ então devolva falha;
10       não-deterministicamente escolha (s, a) ∈ A;
11       π ← π ∪ {(s, a)};
12       S ← S ∪ γ(s, a);
13     senão se s não tem π-descendentes em (S ∪ S_resolvidos) \ S_π então
         devolva falha ;
14   fim
15 fim
```

5.2 Planejador YoYo

Nesta seção é descrito YoYo [KNPT05], um algoritmo de planejamento hierárquico progressivo que lida com ações não-determinísticas. Sua vantagem sobre ND-SHOP2 [KN04] é que combina o planejamento hierárquico com técnicas de verificação de modelos simbólicos. YoYo utiliza um Diagrama de Decisão Binária (*Binary Decision Diagram*) (BDD) [Bry92] para representar um conjunto de estados. Esta forma de representação busca obter maior eficiência, pois permite tratar transições de conjuntos de estados a seus sucessores de uma só vez.

Um exemplo de domínio em que YoYo obtém vantagem é uma variação de *presas e predadores* [KS95]: um predador e várias presas movem-se em uma grade quadrada de tamanho $n \times n$, deslocando-se para regiões adjacentes através de movimentos para o norte, sul, leste ou oeste. As presas, por não serem tão velozes quanto o predador, possuem uma ação adicional descansar, que faz com que não se desloquem. O predador também possui a ação agarrar, executável desde que este ocupe a mesma região de uma presa. O objetivo é planejar ações para o predador de modo a agarrar todas as presas. O movimento aleatório das presas (incluindo a ação de descanso) é responsável pelo indeterminismo do domínio. Como mostrado em [KNPT05], métodos de planejamento hierárquico podem podar o espaço de busca, utilizando a estratégia de tentar seguir uma presa por vez. Conforme aumenta o número de presas, a representação de estados e ações por BDDs evita um aumento exponencial do espaço de busca.

A estrutura do algoritmo é apresentada nesta seção. Em seguida, é descrito como YoYo faz uso de BDDs nos algoritmos.

5.2.1 Algoritmo

Algoritmo 9: YoYo($\mathcal{D}, S_0, S_g, w, \mathcal{M}$)

Entrada: Domínio de planejamento $\mathcal{D}$, Conjunto de estados iniciais S_0, Conjunto de estados meta S_g, Rede de tarefas w, Conjunto de métodos $\mathcal{M}$

1 **início**
2   **devolva** YoYoAux($\mathcal{D}, \{(S_0, w)\}, S_g, \mathcal{M}, \emptyset, \{(S_0, w)\}$);
3 **fim**

A entrada para YoYo (Algoritmo 9) consiste no problema de planejamento $\mathcal{P} = \langle \mathcal{D}, S_0, S_g \rangle$ em um domínio não-determinístico $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \gamma \rangle$, uma rede de tarefas inicial w e um conjunto $\mathcal{M}$ de métodos para o domínio $\mathcal{D}$. O algoritmo explora tuplas da forma (S, w), denominadas *situações*, que são resolvidas pela decomposição da rede de tarefas w nos estados de S.

Começando a partir da situação inicial (S_0, w), YoYo recursivamente gera sucessivos conjuntos de situações até que uma solução para o problema de planejamento dado seja encontrada. A cada iteração do processo de planejamento YoYo primeiro examina o conjunto X de situações à procura de ciclos e estados meta, utilizando PodaSituações. Para cada situação $(S, w) \in X$, PodaSituações verifica o conjunto de estados S e remove qualquer estado que já apareça na política (já possuindo, portanto, uma ação planejada para tal) ou no conjunto de estados meta (e, portanto, nenhuma ação deve ser

Algoritmo 10: YoYoAux($\mathcal{D}, X, S_g, \mathcal{M}, \pi, x_0$)

Entrada: Domínio de planejamento $\mathcal{D}$, Conjunto de situações X, Conjunto de estados meta S_g, Conjunto de métodos $\mathcal{M}$, Política parcial π, Situação inicial x_0

1 **início**
2 $X \leftarrow$ PodaSituações(X, S_g, π);
3 **se** *existe uma situação* $(S, w = \emptyset) \in X$ *tal que* $S \nsubseteq S_g$ **então devolva** *falha*;
4 **se** VerificaPolítica(π, X, S_g, x_0) **então devolva** *falha*;
5 **se** $X = \emptyset$ **então devolva** π;
6 escolha uma situação (S, w) de X e remova-a;
7 $F \leftarrow$ EfetuaDecomposição($S, w, \mathcal{D}, \mathcal{M}$);
8 **se** $F = \emptyset$ **então devolva** *falha*;
9 $X' \leftarrow$ EncontraSucessores(F, X);
10 $\pi' \leftarrow \pi \cup \{(s, a) \mid (S', a, w') \in F \text{ e } s \in S'\}$;
11 $\pi \leftarrow$ YoYoAux($\mathcal{D}, X', S_g, \mathcal{M}, \pi', x_0$);
12 **se** $\pi =$ *falha* **então devolva** *falha*;
13 **devolva** π;
14 **fim**

planejada para tal). Como resultado, o conjunto de situações devolvido por PodaSituações é realmente o conjunto de situações que necessitam ser exploradas e progredidas em sucessoras na busca. Tais situações são denominadas *situações abertas* do feixe de busca em questão.

Algoritmo 11: PodaSituações(X, S_g, π)

Entrada: Conjunto de situações X, Conjunto de estados meta S_g, Política parcial π

1 **início**
2 $X' \leftarrow \emptyset$;
3 **para toda** *situação* $(S, w) \in X$ **faça**
4 $S' \leftarrow S \setminus (S_g \cup S_\pi)$;
5 **se** $S' = \emptyset$ **então** $X' \leftarrow X' \cup \{(S', w)\}$;
6 **devolva** X';
7 **fim**

Após gerar as situações abertas a serem exploradas, YoYo verifica se existe uma situação aberta (S, w) tal que não existam mais tarefas a ser executadas em w mas a meta ainda não tenha sido alcançada (ou seja, $S \nsubseteq S_g$). Neste caso tem-se uma falha na busca e, portanto, YoYo devolve *falha* a partir do feixe de busca corrente. Caso contrário, YoYo usa VerificaPolítica para verificar se a política parcial gerada até o momento adequa-se aos requisitos do tipo de solução buscada. A definição formal desta rotina depende de se estar buscando soluções fortes ou cíclicas fortes; portanto, a discussão é deixada para a próxima seção.

Se a política parcial π não adequa-se aos requisitos de uma solução para o problema

de entrada, então YoYo retorna do feixe de busca corrente com *falha*. Caso contrário, e se não há mais situações abertas a serem exploradas, π é uma solução para o problema de planejamento. Isto deve-se ao fato de que π não viola os requisitos do problema de entrada, pois passou pela verificação de VerificaPolítica no passo anterior, e não tem mais tarefas para decompor.

Algoritmo 12: EfetuaDecomposição($S, w, \mathcal{D}, \mathcal{M}$)

Entrada: Conjunto de estados S, Rede de tarefas w, Domínio de planejamento $\mathcal{D}$, Conjunto de métodos $\mathcal{M}$

1 **início**
2   $F \leftarrow \emptyset; X \leftarrow \{(S, w)\}$;
3   **repita**
4   **se** $X = \emptyset$ **então devolva** F;
5   escolha uma tupla $(S, w) \in X$ e remova-a;
6   escolha uma tarefa t que não tenha predecessores em w;
7   **se** t *é uma tarefa primitiva* **então**
8       $A \leftarrow \{a \mid a \in \mathcal{A}$ é uma ação para t, e $S \subseteq S_a\}$;
9   **se** $A = \emptyset$ **então devolva** $\emptyset$;
10   escolha uma ação a de A;
11       $F \leftarrow F \cup \{(S, a, w \setminus \{t\})\}$;
12   **senão**
13       $M \leftarrow \{m \mid m$ é um método em $\mathcal{M}$ para t, e $S \cap S_m \neq \emptyset\}$;
14   **se** $M = \emptyset$ **então devolva** $\emptyset$;
15   escolha um método m de M;
16       $X \leftarrow X \cup \{(S \cap S_m, (w \setminus \{t\}) \cup w')\}$;
17   **se** $S \setminus S_m \neq \emptyset$ **então** $X \leftarrow X \cup \{(S \setminus S_m, w)\}$;
18   **fim**
19 **fim**

Caso ainda existam situações abertas a explorar, então YoYo seleciona uma delas, (S, w) e procura planejar uma ação para cada estado em S. EfetuaDecomposição, basicamente um planejador STN, é responsável por esta operação, como segue: em uma situação (S, w), seja t uma tarefa que não tem predecessores em w. Se t é uma tarefa primitiva, então t pode ser executada diretamente no ambiente. Seja a uma ação correspondente a t, e a possa ser aplicada a cada estado de S, ou seja, $S \subseteq S_a$. Ressalta-se o fato de que aplicar uma ação a a um conjunto de estados S não gera nenhuma nova situação aberta: S deve ser um sub-conjunto de S_a, pois, do contrário, existe pelo menos um estado em S ao qual nenhuma ação é aplicável, e este é um ponto de falha no planejamento.

Se t não é primitiva, então sucessivamente aplica-se métodos às tarefas não-primitivas em w, até que uma ação seja gerada. Supondo a escolha da aplicação de um método m a t, surgem duas possíveis situações:

1. A situação que surge da decomposição de t por m nos estados $S \cap S_m$, nos quais m é aplicável;

2. A situação que especifica os estados nos quais m não é aplicável, isto é, as situações $(S \setminus S_m, w)$.

No primeiro caso, procede-se com a decomposição de sub-tarefas de t como especifica m. No segundo caso, por outro lado, outros métodos para t devem ser utilizados. Se não existem outros métodos para t em situações como $(S \setminus S_m, w)$, então EfetuaDecomposição devolve o conjunto vazio, fazendo YoYo reportar falha.

EfetuaDecomposição devolve um conjunto F da forma $\{(S_i, a_i, w_i)\}_{i=0}^{k}$. Se $F = \emptyset$, então o processo de decomposição falha, já que existe um estado $s \in S$ tal que não é possível escolher uma ação para s utilizando os métodos provenientes do domínio de planejamento. Se $F \neq \emptyset$ então a rotina gerou uma ação a_i para cada estado em S, ou seja, $S = \bigcup_i S_i$, e uma rede de tarefas w_i a ser completada após a aplicação desta ação.

Supondo que EfetuaDecomposição tenha devolvido um conjunto F não-vazio de tuplas da forma (S', a, w'), YoYo deve proceder computando as situações sucessoras a explorar, utilizando EfetuaDecomposição como segue: para cada tupla $(S', a, w') \in F$, primeiro gera-se o conjunto de estados possíveis da aplicação de a a S' a partir da função

$$\textsc{succ}(S', a) = \{s'' \mid s' \in S' \text{e} (s', a, s'') \in \gamma\},$$

onde γ é a relação de transição de estados do domínio de planejamento. A situação seguinte à aplicação de ações é definida como $(\textsc{succ}(S', a), w')$.

Algoritmo 13: EncontraSucessores(F, X)

Entrada: Conjunto F de ações e redes de tarefas para estados, Conjunto de situações X

1 **início**
2 $X' \leftarrow X \cup \{(\textsc{succ}(S, a), w) \mid (S, a, w) \in F\}$;
3 $X' \leftarrow \{(\textsc{Compõe}(w, X'), w) \mid (S, w) \in X'\}$;
4 **devolva** X';
5 **fim**

Uma vez que YoYo tenha gerado o conjunto de todas as situações sucessoras a explorar, são compostas as novas situações, provenientes das redes de tarefas especificadas. Formalmente,

$$\textsc{Compõe}(w, X) = \{s \mid (S, w) \in X \text{ e } s \in S\}.$$

A composição de um conjunto de situações é um passo de otimização do processo de planejamento. A progressão de situações abertas pode criar um conjunto de situações nas quais mais de uma situação pode especificar a mesma rede de tarefas. A composição de tais situações não é requerida para a corretude, mas tem a vantagem de permitir utilizar representações mais compactas, pelo fato de que estes estados geralmente possuem características comuns.

5.2.2 Planejamento Forte

No planejamento forte, uma política deve induzir uma estrutura de execução até um estado meta, a partir de todo estado alcançável pela aplicação de ações da política,

começando pelos estados iniciais. Além disso, não deve haver ciclos nessa estrutura de execução. Esta condição pode ser verificada através de VERIFICAPOLÍTICA_FORTE, construída a partir da função de pré-imagem regressiva forte de [CPRT03]. Iniciando a partir do conjunto de estados nas situações abertas e os estados meta, calcula o conjunto de estados na política a partir da qual um estado aberto ou um estado meta é alcançável. Durante o processo, remove os pares estado-ação para estes estados calculados pela pré-imagem regressiva forte. Ao final do processo, se resta algum par estado-ação na política, isto significa que a política induz um ciclo na estrutura de execução e, portanto, não pode ser uma solução forte para o problema de planejamento.

VERIFICAPOLÍTICA_FORTE utiliza uma função denominada ESTADOS, que devolve o conjunto de todos os estados que aparecem em um dado conjunto de situações. Mais formalmente:

$$\text{ESTADOS}(X) = \{s \mid (S, w) \in X \text{ e } s \in S\}.$$

Algoritmo 14: VERIFICAPOLÍTICA_FORTE(π, X, S_g, x_0)

```
1  início
2     S' ← ∅;
3     S0 ← ESTADOS(x0);
4     S ← Sg ∪ ESTADOS(X);
5     enquanto S' ≠ S faça
6        S' ← S;
7        S ← S ∪ {s' | (s', a) ∈ π e γ(s', a) ⊆ S};
8        π ← π \ {(s, a) | s ∈ S e (s, a) ∈ π};
9     se S0 ⊆ S e π = ∅ então devolva falso;
10    devolva verdadeiro;
11 fim
```

5.2.3 Utilização de BDDs

Um Diagrama de Decisão Binária (*Binary Decision Diagram*) (BDD) [Bry92] representa uma fórmula booleana como um grafo direcionado acíclico (dígrafo), com raiz (apenas um nó sem predecessores), onde nós internos, não-terminais, correspondem a variáveis da função e nós terminais são rotulados com os valores $\bot$ ou $\top$. Cada nó interno v é rotulado por uma variável $var(v)$ e possui arestas ligando-o a dois outros nós: $lo(v)$, que representa o caso em que a variável é falsa, e $hi(v)$ caso contrário. Para uma dada valoração, o valor-verdade da função é determinado através de um caminho a partir da raiz até um nó terminal, seguindo as arestas indicadas pelos valores das variáveis. O valor da função é então dado pelo valor do nó terminal.

Em planejamento, um vetor $\overline{s}$ de proposições representa o estado atual do ambiente. Um *estado* é uma valoração de $\{\bot, \top\}$ para cada proposição em $\overline{s}$. Baseado nesta formulação, um conjunto de estados S corresponde à fórmula $S(\overline{s})$, tal que:

$$S(\overline{s}) = \bigvee_{s \in S} s(\overline{s}).$$

Esta definição permite redefinir o algoritmo YoYo em termos de representações de conjuntos de estados na forma de BDDs.

Outro vetor $\overline{s}'$ de variáveis proposicionais é utilizado para representar os estados sucessores do ambiente. Similarmente, utiliza-se um vetor $\overline{a}$ de variáveis de ações para representar um conjunto de ações. Uma política π pode ser representada pela fórmula $\pi(\overline{s}, \overline{a})$ nas variáveis $\overline{s}$ e $\overline{a}$. Denota-se um conjunto de estados S com uma fórmula $S(\overline{s})$ no vetor de estados $\overline{s}$. Representa-se uma situação como um par da forma $(S(\overline{s}), w)$, onde w é uma rede de tarefas.

A situação inicial pode ser representada por $x_0 = \{(I(\overline{s}), w)\}$, onde $I(\overline{s})$ representa o conjunto de estados iniciais e w é a rede de tarefas inicial. Similarmente, o conjunto de estados meta é representado pela fórmula $S_g(\overline{s})$ e as relações de transição de estados por $R(\overline{s}, \overline{a}, \overline{s}')$.

As operações de desigualdade entre conjuntos, diferença de conjuntos e relação de sub-conjuntos constituem as primitivas básicas utilizadas em condicionais nos algoritmos da seção 5.2. Estas operações podem ser codificadas em termos de manipulações lógicas nas fórmulas descritas anteriormente.

O resultado da aplicação de uma ação a ao conjunto de estados S pode ser representado pela fórmula: $\exists \overline{s}' : S(\overline{s}) \wedge R(\overline{s}, \overline{a}, \overline{s}')\,[\overline{s}', \overline{s}]$, onde $[\overline{s}', \overline{s}]$ é chamada de operação de *deslocamento progressivo*. Esta fórmula representa a função $\textsc{succ}(S, a)$.

A função Estados pode ser representada pelo operador de união de conjuntos sobre as situações de interesse. Em particular, buscar o conjunto de todos os estados de $X = \{x_1, x_2, \ldots, x_n\}$ corresponde à fórmula $S_1(\overline{s}) \vee S_2(\overline{s}) \vee \ldots \vee S_n(\overline{s})$, onde $x_i = (S_i, w_i)$. PodaSituações pode ser representada como: $S(\overline{s}) \wedge \neg(S_g(\overline{s}) \vee \exists \overline{a} : \pi(\overline{s}, \overline{a}))$.

As funções VerificaPolítica para planejamento forte e forte cíclico são baseadas em duas primitivas para o cálculo pré-imagens fracas e fortes de um conjunto particular de estados. Estes cálculos correspondem a $\exists \overline{a} \exists \overline{s}'.\pi(\overline{s}, \overline{a}) \wedge S(\overline{s}')$ e $\exists \overline{a} \exists \overline{s}'.\pi(\overline{s}, \overline{a}) \implies S(\overline{s}')$, respectivamente.

EncontraSucessores corresponde às fórmulas $S(\overline{s}) \implies S_a(\overline{s})$ e $S(\overline{s}) \wedge S_m(\overline{s})$, onde $S(\overline{s})$ representa o conjunto de estados nos quais a verificação ocorre e $S_a(\overline{s})$ e $S_m(\overline{s})$ representam o conjunto de todos os estados nos quais a ação a e o método m são aplicáveis, respectivamente.

A atualização de uma política π é efetuada por um conjunto de pares estado-ação π' através da fórmula $\pi(\overline{s}, \overline{a}) \vee \pi'(\overline{s}, \overline{a})$.

Capítulo 6

Proposta da Dissertação

A integração das técnicas de planejamento hierárquico, planejamento não-determinístico com observabilidade total e planejamento com verificação de modelos simbólicos resulta em um meio mais eficiente de escolha de ações, para diversos domínios de aplicação, e é capaz de planejar fazendo predições sobre as eventuais contingências que venham a ocorrer durante a execução de planos. Com a união das técnicas é possível tirar proveito de seus benefícios.

Não existe uma implementação de domínio público dos planejadores ND-SHOP2 [KN04] ou YoYo [KNPT05]. Um dos objetivos do estudo é prover uma implementação das técnicas exibidas nas seções anteriores, a fim de possibilitar a comparação empírica com outras abordagens.

Alguns resultados empíricos [KN04] mostram as condições em que o tempo de execução cresce apenas polinomialmente em relação à versão determinística do problema, contribuindo para que a técnica possa ser utilizada em casos práticos. Assim, uma das propostas desse trabalho é a verificação desse fato com uma análise experimental. Para isso, serão usados alguns casos de teste da competição de planejamento.

6.1 Implementação

Para a implementação da técnica apresentada de planejamento hierárquico sob incerteza, será utilizada a linguagem Haskell [HHJW07], uma linguagem de programação funcional pura, com tipagem estática, funções de alta ordem e avaliação preguiçosa. A programação funcional, fundamentada no cálculo lambda [Bar88], permite um estilo declarativo de programação [Hug89]. As características da linguagem tornam-se vantagens para uma certa classe de problemas, que lidam mais com manipulaçòes simbólicas do que entrada/saída. Outro trabalho [Erw04] mostra que Haskell pode ser tão adequado quanto Prolog para resolver problemas de lógica.

A tipagem estática permite, junto com a inferência automática de tipos, verificação da utilização correta de funções e estruturas de dados em tempo de compilação. Programas errados são detectados prematuramente, o que resulta em um ganho de produtividade. A verificação estática de tipos também evita que estes testes precisem ser efetuados em tempo de execução, o que resulta em melhor desempenho. Testes podem ser efetuados de forma mais simples, pois a ausência de efeitos colaterais nas funções garante que

resultados de funções dependam exclusivamente dos argumentos da função.

Outra vantagem proporcionada por esse paradigma de programação é que estruturas de dados puramente funcionais exibem a propriedade de *persistência* [Oka98], ou seja, são estruturas de dados imutáveis. Operações de modificação destas devem devolver novas cópias das estruturas, com a alteração desejada. Na prática, entretanto, é possível compartilhar as partes da estrutura original que não necessitam ser alteradas. Isso facilita a implementação de busca com retrocesso (*backtracking*), sem que seja necessário manter controle das estruturas de dados explicitamente. Os ramos da computação não-determinística recebem as cópias modificadas dessas estruturas.

Um planejador pode ser representado como uma função que recebe um problema de planejamento (domínio, constantes do problema, estados iniciais e estados meta) e devolve uma lista, possivelmente infinita, de planos solução. Quando o primeiro plano da lista é solicitado (para impressão na tela, por exemplo) então a primeira ação a ser impressa vai disparar a avaliação de toda a computação estritamente necessária para a geração do plano. Isto ocorre pois é necessário efetuar a busca completa até os estados meta para garantir que esta primeira ação seja correta. Se apenas um plano é suficiente e o resto da lista de planos é descartada, então não haverá nenhuma computação adicional.

É possível tirar vantagem da avaliação preguiçosa também para facilitar a criação de instâncias de operadores somente quando necessárias. Um operador $\mathsf{mover}(x, y, z)$, com pré-condições $(\mathsf{livre}(x), \mathsf{livre}(y), \mathsf{sobre}(x, y))$, especificado por alguma tarefa primitiva, terá as variáveis x, y e z instanciadas no momento de aplicação da ação. As instâncias aplicáveis serão derivadas através da unificação lógica do operador com os átomos especificados pelo estado atual do mundo.

Bibliotecas existentes para manipulação de BDDs em Haskell vão desde interfaces para bibliotecas externas em outras linguagens [Gam] até implementações totalmente nativas [Chr06]. Implementações tradicionais efetuam a construção de toda a estrutura do BDD sem levar em consideração quais partes serão realmente necessárias. A utilização de avaliação preguiçosa embutida na linguagem de programação permite efetuar a construção de BDDs construção sob demanda de modo mais fácil.

Para a leitura de domínios e problemas, será utilizada a biblioteca Parsec [Lei], que permite implementar parsers diretamente em Haskell. O objetivo é ler uma versão simplificada da linguagem de JSHOP2, contendo especificação de operadores, tarefas e métodos. Isto descarta funcionalidades adicionais como, por exemplo, especificação de axiomas, símbolos de funções e procedimentos anexados.

A representação por funções de mais altas ordens permite que seja utilizada uma forma de especialização de programas [Jon92], através de sua avaliação parcial. Se um programa recebe mais de uma entrada e algumas destas variam com menos freqüência que outras, então a especialização do programa, em respeito a estas entradas, produz uma versão mais específica do programa. No caso de planejamento, é possível criar funções para planejadores específicos de domínio apenas fornecendo o argumento do domínio para a função de planejamento. Deste modo, é possível tirar proveito de otimizações feitas pelo compilador e produzir planejadores específicos de domínio, especializados e mais eficientes.

6.2 Atividades

1. Modelo de estados, operadores, transições, tarefas, métodos e unificação lógica;
2. Leitor de domínios e problemas no formato de SHOP2, estendido para representar ações não-determinísticas;
3. Planejamento clássico através de busca progressiva utilizando apenas operadores de planejamento;
4. Decomposição de tarefas em ordem total (métodos como listas de tarefas);
5. Decomposição de tarefas em ordem parcial (métodos como redes de tarefas);
6. Utilização de BDDs em representações de estados e ações;
7. Testes comparativos de performance utilizando domínios não-determinísticos;
8. Redação da dissertação;
9. Entrega da dissertação e defesa.

6.3 Cronograma

Atividade	2004	2005	2006		2007		
			1o Sem	2o Sem	1o Tri	2o Tri	3o Tri
Créditos	√	√					
Bibliografia		√	√	√			
Implementação				√	√	√	√
Testes						√	√
Redação da Dissertação					√	√	√
Defesa							√

Referências Bibliográficas

[AKN05] Tsz-Chiu Au, Ugur Kuter, and Dana S. Nau. Web service composition with volatile information. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2005.

[Bar88] H. P. Barendregt. Introduction to lambda calculus. In *Aspenæs Workshop on Implementation of Functional Languages, Göteborg*. Programming Methodology Group, University of Göteborg and Chalmers University of Technology, 1988.

[BF97] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[BG99] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In Susanne Biundo and Maria Fox, editors, *ECP*, volume 1809 of *Lecture Notes in Computer Science*, pages 360–372. Springer, 1999.

[BG01] Blai Bonet and Hector Geffner. Planning and control in artificial intelligence: A unifying perspective. *Appl. Intell.*, 14(3):237–252, 2001.

[BHH03] Thorsten Belker, Martin Hammel, and Joachim Hertzberg. Learning to optimize mobile robot navigation based on HTN plans. In *ICRA*, pages 4136–4141. IEEE, 2003.

[BK00] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

[Chr06] Jan Christiansen. *A purely functional implementation of ROBDDs in Haskell*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2006.

[CPRT03] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1-2):35–84, 2003.

[CT91] Ken Currie and Austin Tate. O-plan: The open planning architecture. *Artif. Intell.*, 52(1):49–86, 1991.

[Dea99] Thomas Dean, editor. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages.* Morgan Kaufmann, 1999.

[DGKK98] Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. TAL: Temporal Action Logics language specification and tutorial. *Electron. Trans. Artif. Intell.*, 2:273–306, 1998.

[DK99] Patrick Doherty and Jonas Kvarnström. TALplanner: An empirical investigation of a temporal logic-based forward chaining planner. In *TIME*, pages 47–54, 1999.

[dLP06] Sílvio do Lago Pereira. Planejamento não-determinístico baseado em verificação de modelos. 2006.

[EHN94a] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[EHN94b] Kutluhan Erol, James Hendler, and Dana S. Nau. Semantics for HTN planning. Technical Report CS-TR-3239, 1994.

[EHN94c] Kutluhan Erol, James A. Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254, 1994.

[EHN96] Kutluhan Erol, James A. Hendler, and Dana S. Nau. Complexity results for HTN planning. *Ann. Math. Artif. Intell.*, 18(1):69–93, 1996.

[EHNT95] Kutluhan Erol, James A. Hendler, Dana S. Nau, and Reiko Tsuneto. A critical look at critics in HTN planning. In *IJCAI*, pages 1592–1598, 1995.

[ENS95] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

[Erw04] Martin Erwig. Escape from zurg: an exercise in logic programming. *J. Funct. Program.*, 14(3):253–261, 2004.

[FN71] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[Gam] Peter Gammie. A Haskell binding to Long's BDD library.

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufman, San Francisco, CA, 2004.

[HHJW07] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of Haskell: being lazy with class. 2007.

[HN01] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302, 2001.

[How60] R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, 1960.

[Hug89] J. Hughes. Why Functional Programming Matters. *Computer Journal*, 32(2):98–107, 1989.

[JEK$^+$90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[Jon92] Neil D. Jones. Partial evaluation and the generation of program generators. 1992.

[KMS98] Subbarao Kambhampati, Amol Dattatraya Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *AAAI/IAAI*, pages 882–888, 1998.

[KN04] Ugur Kuter and Dana S. Nau. Forward-chaining planning in nondeterministic domains. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 513–518. AAAI Press / The MIT Press, 2004.

[Kni21] Frank H. Knight. *Risk, Uncertainty and Profit.* Hart, Schaffner, and Marx, 1921.

[KNPT05] Ugur Kuter, Dana S. Nau, Marco Pistore, and Paolo Traverso. A hierarchical task-network planner based on symbolic model checking. In Susanne Biundo, Karen L. Myers, and Kanna Rajan, editors, *ICAPS*, pages 300–309. AAAI, 2005.

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

[KS95] Sven Koenig and Reid G. Simmons. Real-time search in non-deterministic domains. In *IJCAI*, pages 1660–1669, 1995.

[KS99] Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In Dean [Dea99], pages 318–325.

[Lei] Daan Leijen. Parsec, a fast combinator parser.

[MK98] Amol Dattatraya Mali and Subbarao Kambhampati. Encoding HTN planning in propositional logic. In *Artificial Intelligence Planning Systems*, pages 190–198, 1998.

[NAI+03] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[NAI+05] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Héctor Muñoz-Avila, J. William Murdock, Dan Wu, and Fusun Yaman. Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20(2):34–41, 2005.

[NCLMA99] Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In Dean [Dea99], pages 968–975.

[Oka98] Chris Okasaki. *Purely Functional Data Structures.* Cambridge University Press, Cambridge, UK, 1998.

[ON04] O. Okhtay and D. Nau. A general approach to synthesize problem-specific planners, 2004.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[Rov01] Marco Roveri. Planning in non-deterministic domains via symbolic model checking, 2001.

[Sac75] Earl D. Sacerdoti. The nonlinear nature of plans. In *IJCAI*, pages 206–214, 1975.

[SBNM02] T. Son, C. Baral, T. Nam, and S. McIlraith. Domain-dependent knowledge in answer set planning, 2002.

[SPW+04] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using SHOP2. *J. Web Sem.*, 1(4):377–396, 2004.

[Tat77] Austin Tate. Generating project networks. In *IJCAI*, pages 888–893, 1977.

[TCdB07] Felipe W. Trevizan, Fabio G. Cozman, and Leliane N. de Barros. Planning under risk and knightian uncertainty. In *IJCAI*, 2007.

[Ull05] Carsten Ullrich. Course generation based on HTN planning. In Mathias Bauer, Boris Brandherm, Johannes Fürnkranz, Gunter Grieser, Andreas Hotho, Andreas Jedlitschka, and Alexander Kröner, editors, *LWA*, pages 74–79. DFKI, 2005.

[vR01] Femke van Raamsdonk. Book review - logic in computer science: Modelling and reasoning about systems by michael r. a. huth and mark d. ryan, cambridge university press, 2000, isbn 0521652006, (hardback), isbn 0521656028, (paperback). *TPLP*, 1(1):123–125, 2001.

[Wil90] David E. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, 6:232–246, 1990.

[WPS+03] Dan Wu, Bijan Parsia, Evren Sirin, James A. Hendler, and Dana S. Nau. Automating DAML-S web services composition using SHOP2. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2003.

[WSH+03] Dan Wu, Evren Sirin, James A. Hendler, Dana S. Nau, and Bijan Parsia. Automatic web services composition using SHOP2. In *WWW (Posters)*, 2003.

[Yan90] Qiang Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6:12–24, 1990.