Universidade Técnica de Lisboa
Instituto Superior Técnico

Planeamento e Acções: Uma Abordagem Integrada

Daniel Jorge Viegas Gonçalves
(Licenciado)

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Tese realizada sob a orientação de
Professor João Pavão Martins
Doutorado

Abril de 2000

Universidade Técnica de Lisboa
Instituto Superior Técnico

Planeamento e Acções: Uma Abordagem Integrada

Daniel Jorge Viegas Gonçalves
(Licenciado)

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Tese realizada sob a orientação de
Professor João Pavão Martins
Doutorado

Abril de 2000

Agradecimentos

Agradeço ao meu orientador, João Pavão Martins, pelas oportunidades e ajuda dadas ao longo do desenvolvimento desta tese.

Não posso deixar de agradecer a todos os restantes membros do Grupo de Inteligência Artificial, em especial ao meu colega de Mestrado Carlos António.

Agradeço à minha mulher, Sónia Patrícia, pela compreensão e paciência demonstrados ao longo destes meses. Só assim foi possível desenvolver o trabalho aqui apresentado.

Finalmente, agradeço aos meus pais e à minha irmã que me souberam motivar e apoiar em todos os pontos do trajecto que conduziu a esta tese.

Este trabalho foi financiado pelo Projecto PRAXIS XXI 2/2.1/TIT/1568/95.

Abril de 2000
Daniel Jorge Viegas Gonçalves

Resumo

A Lógica de Primeira Ordem é um dos formalismos mais usados em Inteligência Artificial. Reveste-se porém de inúmeros problemas. O *Problema da Quiescência* revela a sua inadequação para o raciocínio sobre mudança. Para lidar com ele, surge a *Lógica das Mutações*, que permite efectuar de forma natural raciocínios sobre diferentes instantes de tempo.

Por outro lado, a maioria dos sistemas clássicos são incorpóreos, sem acesso ao mundo. Isso faz com que estejam inerentemente limitados. Sistemas mais recentes estão situados no mundo, mas não tratam uniformemente o raciocínio e a acção. Assim, surge o *OK-BDI*, um formalismo em que o raciocínio e a acção são tratados de forma integrada.

Este trabalho integra esses dois formalismos. Surge assim um formalismo capaz de raciocínio sobre mudança e diferentes situações do mundo, mas ao mesmo tempo situado no mundo sobre o qual raciocina. Foram estudados os problemas e as vantagens que daí resultam.

Para demonstrar a sua utilidade, foi desenvolvido um planeador em que a qualidade dos planos se encontra melhorada em relação à dos planeadores tradicionais pelo facto de o algoritmo ter acesso ao mundo para que se planeia, refinando assim os planos gerados.

Finalmente, o formalismo desenvolvido foi implementado em SNePS.

Palavras-Chave: representação do conhecimento, raciocínio sobre mudança, lógicas não-padrão, planeamento de acções, planeamento condicional, acções, redes semânticas.

Abstract

First Order Logic is one of the most common knowledge representation languages in Artificial Intelligence. However, certain problems arise. The *Frame Problem* shows it's inadequacy to handle reasoning about change. To minimize this the *Mutation Logic* was created. It can reson about change and several time instants in a natural fashion.

On the other hand, most classic systems are unembodied, without access to the world. This is an inherent limitation of those systems. More recent systems are built situated in the world. However, they do not handle reasoning and acting in an integrated way. The *OK-BDI* formalism appears to integrate reasoning and acting.

This work integrates those two formalisms. The result is a formalism able to perform both reasoning about change and at the same time situated in the world it reasons about. The problems and advantages of such a system were studied.

In order to demonstrate it's utility, a planner was developed. It is shown how the quality of the generated plans is greater than that of classical planners due to the embodiement of the planner.

Finally, the new formalism was implemented in SNePS.

Key-Words: knowledge representation, reasoning about change, non-standard logics, planning, conditional planning, acting, semantic networks.

Índice

Capítulo 1

Introdução

Neste capítulo é efectuado um enquadramento da tese nas principais áreas de investigação em Inteligência Artificial. São indicados os objectivos da tese e delineada a abordagem seguida para os atingir.

1.1 Prólogo

A Inteligência Artificial preocupa-se em obter programas que manifestem comportamento inteligente. Embora este objectivo seja simples de enunciar, não existe consenso entre os investigadores da área quanto ao que ele realmente significa e qual a forma correcta para o atingir.

Esta dificuldade provém do facto de que, em última análise, é difícil definir em que consiste o comportamento inteligente. É geralmente aceite que o comportamento inteligente é aquele demonstrado pelos seres humanos quando desempenhando uma tarefa que requer inteligência. No entanto, esta definição só por sí não é suficiente. Como reconheceu pela primeira vez McCarthy [McCarthy 1958], existem programas capazes de resolver problemas complexos, requerendo um elevado grau de inteligência, mas incapazes de resolver alguns dos problemas mais elementares do dia-a-dia. É evidente para qualquer observador que não são, de facto, inteligentes. Assim, McCarthy introduziu a noção de *senso comum*, como algo que um programa inteligente deve manifestar. Segundo ele, um programa com senso comum é aquele que "deduz, automaticamente, por si proprio, uma classe suficientemente vasta de consequências de qualquer coisa que lhe é dita e do que já sabe" [McCarthy 1958]. Ao referir que o programa deve derivar uma classe *suficientemente vasta* de consequências, deixa-se em aberto quantas e de que natureza devem ser essas consequências. Essas características ficam definidas à custa do que os humanos consideram "suficientemente vasto", revertendo-se à definição inicial. Esse conceito implica que o programa deve ser capaz de obter conclusões que não se circunscrevam a um único domínio muito específico, mas sim

que se reportem a um domínio mais vasto.

Duas áreas da Inteligência Artificial intimamente relacionadas para as quais o senso comum se mostra mais necessário são o Planeamento de Acções e o Raciocínio sobre Mudança. O *planeamento de acções* preocupa-se com encontrar sequências de acções (mudanças) que, quando executadas, conduzam de uma qualquer situação inicial a uma situação final em que certas condições se verificam. Isso é algo que qualquer programa inteligente deve ser capaz de conseguir, de forma a atingir determinados objectivos, impostos externamente ou resultantes de processos internos. O *raciocínio sobre mudança* é uma área complementar e preocupa-se com as propriedades formais das mudanças: o que as caracteriza, quando podem ser aplicadas e quais os seus efeitos.

Nessas áreas, grande parte da informação necessária deriva, directamente, do senso comum. A forma em que se processam as mudanças no mundo é, em grande parte, evidente para os seres humanos. Pintar uma parede não altera a sua posição. Mover um livro não altera o seu conteúdo. Para um formalismo dedicado ao planeamento, por outro lado, essa informação não é conhecida à priori, devendo, de alguma forma, ser representada. Isto torna necessário um especial cuidado com a linguagem usada pelo formalismo. Esta deve permitir representar de modo natural a informação necessária e raciocinar relativamente a diferentes instantes de tempo, o que os caracteriza e sobre as mudanças que ocorrem de um instante para o seguinte. Este tipo de raciocínio é designado por *raciocínio diacrónico*, dado que envolve a passagem do mundo por diversos instantes diferentes causada pela execução de mudança e a análise das alterações provocadas ao mesmo. Reciprocamente é possível considerar o *raciocínio sincrónico*, referente aos factos presentes no mundo num dado instante [Janlert 1987].

Existem duas abordagens radicalmente distintas para tentar obter o raciocínio do senso comum necessário para, entre outras aplicações, a criação de planeadores:

1. A *Abordagem Logicista.*
2. A *Abordagem Corpórea.*

A primeira tem as suas origens nos trabalhos de McCarthy e assume que o conhecimento guardado no sistema deve ter uma natureza proposicional. A segunda defende que o comportamento inteligente tem como base os comportamentos simples efectuados pelos sistemas físicos que devem existir como base à entidade inteligente.

1.1.1 A Abordagem Logicista

A abordagem logicista pressupõe a existência de uma linguagem formal bem definida e com expressividade suficiente para representar a informação relevante. É a partir das

proposições representadas nessa linguagem que serão feitas as inferências necessárias para obter as conclusões procuradas.

Esta premissa encontra-se bem explícita na *Hipótese Dos Símbolos Físicos*. Um sistema de símbolos físicos é composto por um conjunto de entidades, chamadas símbolos, que podem ser componentes de entidades mais complexas designadas por expressões. O sistema é caracterizado a cada instante pela presença de um conjunto dessas expressões. Adicionalmente, existem no sistema um conjunto de processos capazes de criar novas expressões a partir de expressões já existentes. Um sistema de símbolos físicos evolui ao longo do tempo alterando as expressões nele contidas mediante a utilização dos processos existentes. A *Hipótese Dos Símbolos Físicos* diz que um sistema de símbolos físicos é condição necessária e suficiente para se obter comportamento inteligente [Newell e Simon 1976].

É, pois, colocado um grande ênfase no papel preponderante da utilização de uma linguagem bem definida, simbólica, para a obtenção de comportamento inteligente. Nada é dito, no entanto quanto à natureza dessa linguagem. McCarthy propôs a utilização da Lógica de Primeira Ordem (LPO). O conhecimento é representado através de um conjunto de proposições, e as conclusões obtidas por aplicação de regras de inferência, num processo semelhante ao da demonstração de teoremas. A LPO tem diversas vantagens, uma das quais é o modo natural em que permite representar o conhecimento, sob a forma de proposições com uma estrutura próxima da usada pelos humanos no seu discurso em língua natural.

Verificou-se no entanto que a LPO não é adequada para efectuar raciocínio diacrónico. A tentativa de a utilizar para esse tipo de raciocínio em aplicações como, por exemplo o planeamento de acções, revelou um dos problemas mais importantes da Inteligência Artificial: o *Problema da Quiescência*, ou *frame-problem*. Esse problema consiste numa dificuldade inerente ao formalismo em nele se representar a mudança. Pela natureza da LPO, quando são especificados os efeitos de uma acção, como por exemplo "ao pintar uma parede de vermelho, passa a ter a cor vermelha", seria, também necessário, explicitar tudo o que *não* muda devido a essa acção. Por exemplo, a cor de uma parede que não foi pintada não muda, tal como não se altera a posição de um móvel. Por senso comum, é evidente que essas alterações não directamente relacionadas com a acção executada não ocorrem. No entanto, é necessário explicitá-las na lógica, o que torna quase incomportável a sua utilização.

Esse problema fez com que se tentassem encontrar formas alternativas de representar a mudança. Isto levou Nilsson e Fikes a criar o `STRIPS` [Fikes e Nilsson 1971], um sistema planeador em que a LPO é usada apenas para a representação daquilo que é estático no mundo, encontrando-se a mudança representada apenas a um meta-nível, de forma independente da restante informação sobre o mundo. Esse planeador deu origem a uma família de sistemas que ainda se encontra em crescimento hoje em dia [Penberty e Weld 1992, Pryor e Collins 1996].

Os defensores da abordagem logicista, porém, argumentam que ao colocar a mudança a um meta nível, se perde algo que podia ser de grande utilidade nalgumas aplicações. Para além disso, o `STRIPS` e os sistemas nele baseados não possuem uma semântica formal bem definida, que permita saber à priori que tipos de problemas conseguem resolver e de que forma o farão. A LPO possui essa semântica rigorosa, o que permite conhecer à priori certos resultados sobre a mesma.

Surgiram, pois, vários trabalhos em que se tentam resolver problemas da LPO mantendo uma abordagem logicista. Estes trabalhos dividem-se em duas grandes famílias:

1. Aquela em que se tenta restringir de alguma forma o modo de representar o conhecimento ou de efectuar inferência;
2. Aquela em que se tentam criar novas lógicas que, extendendo ou alterando de alguma forma a LPO, se mantêm fiéis à filosofia que lhe está subjacente mas são mais adequadas para a representação de certos aspectos em que esta não se comporta tão bem como seria de desejar.

Entre os trabalhos da primeira família encontra-se a escrita de axiomas de quiescência [McCarthy 1958] ou a circunscrição [McCarthy 1980, McCarthy 1986]. De entre os trabalhos da segunda família, destacam-se lógicas como a Lógica de Omissão de Reiter [Reiter 1978] ou a Lógica Auto-Epistémica [Moore 1988]. Surge ainda neste âmbito, para tentar simplificar e tornar natural a representação e raciocínio sobre mudança, a Lógica das Mutações.

A Lógica das Mutações [Pinto-Ferreira 1991] é uma extensão à LPO em que a mudança é reificada. É introduzida uma nova conectiva lógica, designada por *mutação*, que pode ser usada para representar de forma directa as próprias mudanças. Isto permite que as mudanças sejam consideradas como quaisquer outros elementos da linguagem, de modo a ser possível, sempre que necessário, inferir propriedades sobre as mesmas. De igual modo, a aplicação das mudanças encontra-se, agora, ao nível da linguagem, não passando de mais uma regra de inferência. Embora continue a ser uma lógica, dado que a sua criação contemplou especificamente a reificação da mudança, é possível representá-la de forma natural e sem os problemas inerentes a essa representação que a LPO apresenta. A Lógica das Mutações, pela naturalidade com que permite a representação da mudança, permite entre outras coisas a construção de planeadores em que, ao contrário do `STRIPS`, esta não se encontra a um meta-nível.

1.1.2 A Abordagem Corpórea

A segunda abordagem seguida para obter comportamento inteligente com raciocínio do senso comum é radicalmente diferente da abordagem logicista. Os sistemas que seguem a

abordagem clássica seguem uma metodologia *do topo para a base*, ou *top-down*. Tentam representar aquilo que consideram ser o que permite aos humanos demonstrar inteligência: o seu conhecimento e processos de raciocínio de alto nível. A segunda abordagem defende que os sistemas devem ser construidos segundo uma metodologia *da base para o topo*, ou *bottom-up*, em que começam por ser implementados comportamentos simples, reactivos, devendo os comportamentos mais complexos emergir das interacções desses comportamentos mais simples. Pretende, de alguma forma, simular a evolução natural, em que os organismos mais simples foram os que primeiro se desenvolveram, estabelecendo as bases para que outros mais complexos pudessem surgir. Ainda continuando a analogia biológica, tal como uma criança não nasce completamente ensinada, aprendendo progressivamente conceitos mais complexos, não é possível construir um sistema inteligente ao qual sejam dados apenas conhecimentos de alto nível sem uma base de apoio sólida de outros mais simples.

Esta abordagem teve a sua génese os trabalhos de Brooks [Brooks 1991] com insectos robóticos, em que é a arquitectura dos sistemas a responsável pelo comportamento por eles demostrado, e não qualquer conhecimento que neles exista explicitamente representado. Uma das principais críticas efectuadas à abordagem logicista é o facto dos sistemas nela baseados pretenderem modelar inteligências incorpóreas, não situadas num mundo com o qual possam interagir de modo a apreender novas informações ou verificar o resultado das suas acções. É devido a não se encontrarem situados que se torna difícil a representação do conhecimento do senso comum. Para os seres humanos, existe uma série de premissas básicas aprendidas ao longo da vida, resultantes directamente da observação do mundo físico e da experiência. Todos sabemos, por exemplo, que pintar paredes não altera a posição de livros em estantes ou que se algo for empurrado acaba por parar, ou se for atirado ao ar, cai em seguida. Um sistema incorpóreo não possui os meios através dos quais poderia aprender todos estes factos simples do dia-a-dia, pelo que devem ser representados explicitamente no sistema. Isto é habitualmente muito difícil ou mesmo impossível, não só pela quntidade da informação a representar como também pela sua natureza.

Enquadrada na abordagem corpórea, surge uma nova área, a dos *Agentes de Software*, que tenta, de alguma forma, integrar nessa abordagem características da abordagem Logicista. Tal como é usual na abordagem corpórea, a área dos agentes de sobtware Considera entidades individuais, que não são entidades incorpóreas e isoladas, mas sim entidades que se encontram situados num mundo, real ou simulado, com o qual interagem, através de *sensores* e *actuadores*. A essas entidades é dado o nome de *agentes*.

Cada agente, por outro lado, não vai agir unicamente com base nas suas percepções do mundo. Possui um estado interno, definido com base num conjunto de *crenças*, representadas recorrendo a algum formalismo, eventualmente próximo ao usado na abordagem logicista. Não se abandonou totalmente a abordagem logicista. Algumas facetas dessa abordagem continuam a ser usadas de uma nova forma. Para além das crenças, um agente tem *desejos*, resultantes das suas necessidades e que pode obter analisando as suas crenças.

Esses desejos conduzem à *intenção* de levar a cabo alguma acção que pode conduzir à alteração das suas crenças. Isto define um modelo para a estrutura e funcionamento dos agentes, conhecido por modelo *BDI* (do Inglês *Beliefs, Desires and Intentions*).

Embora os agentes de software tentem eliminar o problema que reside no facto de um sistema não se encontrar situado num mundo, a sua grande maioria continua a considerar a mudança como algo situado a um meta-nível. Os agentes possuem a intenção de efectuar alguma acção, e essa será executada por algum outro processo que reconhece as intenções dos agentes e as satisfaz. Normalmente, a acção é usada ao serviço da inferência ou, mais raramente, a inferência ao serviço da acção, mas apenas pontualmente se assiste a uma utlização integrada. Isto ocorre porque os sistemas que normalmente permitem efectuar inferência de forma natural e eficiente não permitem a utilização das acções do mesmo modo. De um modo geral, um bom raciocinador não é um bom actuador e vice-versa. Um bom exemplo disto é que na maior parte dos motores de inferência, por motivos de eficiência, inferências que já foram efectuadas não são repetidas, sendo antes guardadas as suas conclusões para que futuras consultas sejam imediatas. No âmbito de um actuador, por outro lado, faz todo o sentido, e por vezes é mesmo imperativo, executar a mesma acção mais do que uma vez. Assim, tratar a acção como mais uma forma de inferência não é minimamente adequado.

Reconhecendo este problema, Kumar desenvolveu o formalismo OK-BDI [Kumar 1993]. O OK-BDI, do Inglês *Object-Knowledge Beliefs, Desires and Intentions* ou *"Crenças, Desejos, e Intenções Orientadas aos Objectos"* é um sistema que pretende tratar de forma integrada o raciocínio e a acção. É definido um formalismo orientado aos objectos em que as diversas classes permitem representar todas as crenças do agente. Podem ser encontradas na mesma hierarquia as classes que permitem representar as proposições e as acções. Ambos os tipos de entidades são, pois, representados com a mesma linguagem. Este formalismo possui uma expressividade que extende a da LPO. De modo a tratar de forma integrada a acção e a inferência, esta é efectuada através de entidades especiais, designadas por *transformadores*, que permitem associar livremente acções e proposições. De facto, considera-se que a inferência no sentido clássico não é mais do que a execução de um tipo particular de acções: as acções mentais. Dada a integração conseguida, é possível que em qualquer altura, no processo de inferência, surja o desejo de executar uma dada acção de modo a, por exemplo, obter certa informação crucial para o processo de inferência nesse momento. O recíproco também é possível, podendo efectuar-se a inferência durante a tentativa de satisfação de uma intenção do agente, de modo a discernir qual a melhor forma de o fazer.

1.2 Motivação

Duas das características desejáveis num sistema que efectue raciocínio do senso comum são, como foi referido, a *localização no mundo* e a *capacidade de raciocinar sobre a mudança e os diferentes estados por que o mundo passa*. A primeira irá permitir que o agente consiga raciocinar sobre propriedades do mesmo e indicar acções correctas a efectuar de forma mais simples e natural. A segunda é importante para que o agente possa planear qual a melhor estratégia a adoptar antes de realmente a executar no mundo em que, muitas vezes, decisões tomadas são irreversíveis.

Os sistemas logicistas não possuem nenhuma destas características. Como consequência, vários formalismos foram criados para tentar atingir cada uma delas. Dois de entre eles são o objecto de estudo deste trabalho. A Lógica das Mutações permite expressar naturalmente que mudanças podem ocorrer no mundo e quais os seus efeitos, sendo uma base natural para a construção de sistemas de raciocínio diacrónico, de entre os quais se destaca o planeamento de acções. É, no entanto, um formalismo incorpóreo, não possuindo qualquer capacidade de interacção com um eventual mundo. Assim, é possível usar a Lógica das Mutações para planear, mas apenas sobre um modelo abstracto do mundo, gerando-se planos que apenas podem vir a ser executados mediante a intervenção de um agente externo.

O OK-BDI tem uma natureza oposta à da Lógica das Mutações. Encontra-se perfeitamente situado no mundo, podendo interagir com ele de forma transparente. O raciocínio e a acção encontram-se integrados, podendo usar-se indistintamente sempre que necessário. Todavia, não existem mecanismos básicos do sistema que permitam ao formalismo raciocinar sobre instantes de tempo que não o presente. Pode decidir que acção vai executar em seguida, mas não planear um conjunto de acções a executar ou efectuar raciocínio hipotético sobre uma dada situação que não a actual.

Assim, a motivação deste trabalho é a integração da Lógica das Mutações com o OK-BDI, de modo a obter um sistema capaz de agir e raciocinar de forma integrada, podendo o raciocínio reportar-se a diferentes instantes de tempo permitindo o planeamento à priori das acções a executar. Em particular, levantam-se as seguintes questões:

(i) Em que convergem e diferem os dois formalismos a integrar e quais principais as características a manter de cada um deles?

(ii) Até que ponto são compatíveis as diferenças existentes entre os formalismos e qual o seu impacto sobre o formalismo integrado?

(iii) Que alterações são necessárias aos formalismos para que exista uma base comum para a sua integração? Que novas estruturas e processos devem ser considerados?

(iv) Como pode ser usado o formalismo obtido para várias aplicações, com destaque para

o planeamento de acções, e em que medida essas aplicações foram facilitadas pela utilização do novo formalismo?

A primeira questão obriga a um estudo prévio dos dois formalismos e à identificação, dentro das áreas em que se inserem, das suas características principais. Pretende-se aqui encontrar o conjunto de características que define cada um dos formalismos e que deve ser mantido no formalismo final de modo a que tenha o comportamento pretendido.

A resposta à segunda questão consiste em analisar as características obtidas como resposta à anterior e identificar a compatibilidade entre elas. Se algumas se mostrarem incompatíveis, pode ser necessário relaxar alguns constrangimentos e identificar em que medida isso irá afectar as características do formalismo resultante da integração.

Em terceiro lugar, pretendem-se resolver as questões que possam surgir durante o processo de integração propriamente dito. Já foram identificadas as características a integrar, pelo que a sua integração irá passar pela definição e utilização de novas entidades. Tenta-se, aqui, identificar quais as entidades e processos a criar de modo a conseguir obter as características desejadas.

Finalmente, é necessário estudar de que forma o formalismo obtido pode ser usado para melhorar a performance do agente considerado, permitindo-lhe planear as suas acções correctamente antes de as executar. Interessa, também, verificar de que forma pode o processo clássico de planeamento ser melhorado em virtude de estar a ser efectuado com base num sistema inserido no mundo sobre o qual os planos vão ser executados.

1.3 Estrutura da Tese

No capítulo dois (Formalismos a Integrar) é feita uma descrição dos aspectos mais relevantes dos dois formalismos a integrar, a Lógica das Mutações e o OK-BDI. Nesse capítulo são ainda identificados os pontos de contacto e discordância entre eles.

No capítulo três (Formalismo Integrado), é feita a integração dos dois sistemas, obtendo-se o formalismo que permite tratar de forma integrada as acções, o raciocínio sincrónico e o raciocínio diacrónico. São descritas todas as estruturas e processos que isso envolve.

Segue-se o capítulo quatro (O Planeador Integrado) em que o formalismo que acabou de ser definido é utilizado para a criação de um planeador. São indicadas quais as características que esse planeador deve ter, com que problemas deve lidar e como pode ser melhorado, em relação aos planeadores clássicos, em virtude do formalismo em que se baseia.

Finalmente, no capítulo cinco (Conclusões) são apresentadas as principais conclusões e resultados obtidos neste trabalho, delineando-se eventuais caminhos para uma futura investigação nesta área.

Capítulo 2

Formalismos a Integrar

Dado que o objectivo desta tese é a integração do OK-BDI [Kumar 1993] e a Lógica das Mutações [Pinto-Ferreira 1991], é necessário conhecer esses formalismos com algum detalhe, de forma a discernir em que pontos divergem. Neste capítulo é apresentada uma descrição sucinta dos aspectos mais relevantes destes formalismos. Segue-se ainda uma análise comparativa de ambos, que serve de base para o trabalho descrito nos capítulos seguintes.

2.1 O OK-BDI

É sabido que considerar, de forma integrada, o raciocínio e a acção num mesmo sistema não é uma tarefa simples. Usualmente, os sistemas desenvolvidos tendo em vista a representação do conhecimento e o raciocínio não possuem os mecanismos que lhes permitam lidar com acções. De igual modo, um sistema de planeamento/acção não é adequado para efectuar quer a representação do conhecimento quer o raciocínio, de forma simples ou correcta.

Quando se pretende, de alguma forma, usar os dois tipos de sistemas numa mesma aplicação, isto é feito usando dois subsistemas independentes, lidando cada um deles com uma das vertentes acima descritas. No entanto, dado que os subsistemas se mantêm em maior ou menor grau separados entre si, as eventuais potencialidades de um tal sistema acabam por nunca se concretizar.

Um bom exemplo disto é o facto de, normalmente, a inferência ser usada ao serviço da acção. Isto sucede, por exemplo, quando um planeador recorre à inferência para deduzir propriedades do mundo que necessita conhecer para validar a aplicação de um dado operador. Apenas esporadicamente se verifica a situação inversa, em que o actuador é usado ao serviço da inferência, normalmente para verificar a presença de propriedades que permitam aplicar alguma regra de inferência.

Numa tentativa de conseguir um sistema em que, realmente, a acção e o raciocínio se encontrem integrados de forma completa, foi desenvolvido por Deepak Kumar o sistema OK-BDI (*Object-Knowledge Beliefs, Desires and Intentions*) [Kumar 1993], que será descrito em seguida.

2.1.1 Pressupostos do Sistema

De modo a conseguir lidar com o raciocínio e as acções em simultâneo e de forma igualmente correcta, o OK-BDI baseia-se numa série de pressupostos descritos em seguida.

Talvez o mais importante desses pressupostos, dado o objectivo a atingir, seja de natureza epistemológica: *todo o conhecimento necessário pelo sistema para raciocinar, planear, e agir deve encontrar-se representado num único formalismo.* De facto, apenas se tal ocorrer será possível tratar todas essas facetas do sistema de forma uniforme e integrada.

No entanto, apenas um formalismo de representação idêntico para os vários tipos de conhecimento não é condição suficiente para um sistema integrado. Também o modo de utilização dessa representação em cada uma das suas vertentes é de especial importância. Assim, o OK-BDI estabelece um compromisso a nível semântico, que conduz à reformulação do processo de inferência. Assume que a inferência não é mais do que um processo levado a cabo por um tipo especial de acção: as *acções mentais*. Sob este prisma, uma regra de inferência é vista como uma regra que especifica a acção de *acreditar* numa proposição anteriormente não acreditada.

Este novo ponto de vista sobre a inferência pode ser criticado dado que a inferência deixa de ser uma primitiva básica do sistema, passível de optimização na implementação do mesmo. Mas, para além da vantagem da integração dos dois domínios (o raciocínio e a acção), permite ainda lidar com as acções de forma apropriada. De facto, se se tentasse, de algum modo, lidar com as acções com mecanismos similares aos do raciocínio, poderiam não ser obtidos os resultados desejados, dado que muitos sistemas, por motivos de eficiência, não voltam a repetir uma inferência quando esta já foi efectuada anteriormente. Este comportamento não seria aceitável num sistema integrado, dado que é perfeitamente plausível que uma mesma acção seja executada um número indeterminado de vezes.

O OK-BDI implementa, assim, um *motor racional*. Ao contrário de um *motor de inferência*, um motor racional consegue lidar de forma correcta e integrada tanto com a inferência como com as acções.

2.1.2 A Arquitectura do Sistema

O OK-BDI Pode ser visto como dividido em duas partes distintas: o *Formalismo de Representação OK* e o *Motor Racional*.

O formalismo de representação OK é composto por uma hierarquia de classes (dai a designação de *Object Knowledge* ou "conhecimento orientado aos objectos") extensível, que engloba todos os conceitos representáveis no sistema, tais como as proposições e as acções.

O motor racional é definido, essencialmente, pelo conjunto de métodos definidos e especializados ao longo das classes da hierarquia.

2.1.3 O Formalismo OK

A hierarquia base de classes definidas no OK-BDI encontra-se representada graficamente na figura 2.1, indicando-se junto a cada classe as suas propriedades mais importantes.

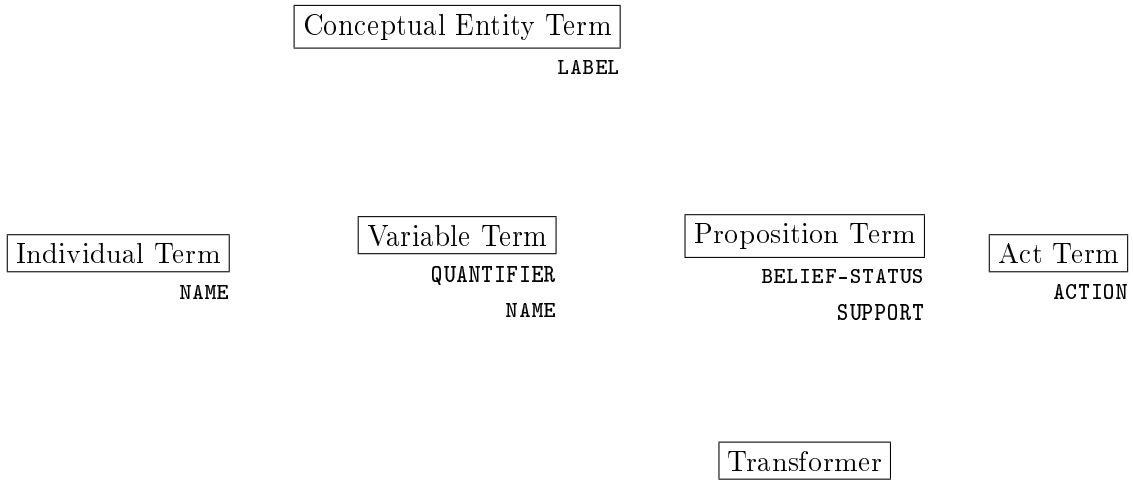


Figura 2.1: Hierarquia base de classes no OK-BDI

Esta hierarquia, pela sua natureza orientada aos objectos, pode ser extendida de acordo com as necessidades particulares de cada utilização do sistema.

De entre as classes pré-existentes, as *entidades conceptuais* (`Conceptual Entity Term`) englobam tudo aquilo a que o agente cognitivo se pode referir, dai ser essa classe o topo da hierarquia. As suas instâncias e das suas sub-classes denotarão conceitos intensionais do sistema. Nesta classe encontra-se definida a propriedade `LABEL`, que será uma etiqueta que identifica univocamente cada conceito dentro do sistema.

É respeitado o *Principio da Unicidade* para todas as entidades conceptuais, ou seja, existe uma correspondência de um para um entre essas entidades e os conceitos intensionais que representam.

A primeira sub-classe de `Conceptual Entity Term` a referir é `Individual Term`, usada para representar *termos individuais* ou concretos. As instâncias desta classe são usadas quando no sistema se quer representar um indivíduo ou entidade concretas (uma dada

pessoa ou lugar, por exemplo). As instâncias desta classe são caracterizadas recorrendo ao seu *nome*, contido no campo `NAME`.

Em seguida, encontramos `Variable Term`. As instâncias desta classe serão *variáveis*, denotando entidades conceptuais arbitrárias. Cada variável pode ter associado, para além do seu nome, um quantificador. Se tal não ocorrer, a variável designa-se como *variável livre*.

A classe `Proposition Term` é, na realidade, uma meta-classe. As suas instâncias são classes cujas instâncias representam *proposições* no sistema OK-BDI.

Um exemplo disto é, por exemplo, a sub-classe `Isa`. Esta classe tem dois campos, `MEMBER` e `CLASS`, e representa o facto da entidade que serve de valor ao primeiro ser um membro da classe representada pela entidade que é o valor do segundo. Assim, apenas as instâncias desta classe representam proposições concretas, representando ela apenas o "padrão" para as diversas proposições desse tipo.

O sistema pode representar proposições em que não acredita. A classe `Proposition Term` contém um campo, `BELIEF-STATUS` que indica se uma dada proposição é ou não acreditada.

O OK-BDI é modelado como um conjunto de *espaços de crenças*, caracterizado pelas proposições acreditadas em cada momento. Mais concretamente, cada espaço de crenças é composto pelas hipóteses acreditadas num dado momento e pelas proposições em que é possível acreditar por derivação a partir dessas hipóteses.

Para ajudar a registar a informação relevante para a manutenção do estado de crença das diversas proposições e da consistência do espaço de crenças, o OK-BDI recorre a um *Sistema de Manutenção de Verdade*, ou TMS[1]. A informação relacionada com o TMS para uma dada proposição (as suposições/justificações na base da crença na proposição) estão contidas no campo `SUPPORT`.

Finalmente note-se que, dado que as proposições são, também elas, subclasses de `Proposition Term`, o OK-BDI pode ter crenças sobre as suas próprias crenças.

Existem, ainda, duas outras classes a considerar: `Act Term` e `Transformer`. Estas duas classes são a base na hierarquia para a representação das acções e das regras de inferência do OK-BDI. Dada a grande importância destas duas classes e dos conceitos que lhes estão associados, serão descritas em maior detalhe nas secções seguintes.

[1] Da nomenclatura anglo-saxónica *Truth Maintenance Sistem*.

As Acções

`Act Term` é a classe cujas subclasses representam *actos* que um agente pode executar sobre alguma entidade conceptual nalgum instante de tempo. Uma *acção* é o resultado de executar um dado acto. De agora em diante, será feita referência apenas a *acções*, como forma de simplificar o discurso, mas mantendo-se sempre presente esta distinção.

As acções são classificadas como *simples* ou *complexas*. As primeiras são as que podem ser executadas directamente. Têm, pois, associados actuadores encarregados de levar a cabo o seu funcionamento. As segundas correspondem a um plano que deve ser decomposto de alguma forma em acções simples antes de ser executado.

Ortogonalmente a esta classificação, as acções encontram-se divididas em *acções físicas*, as *acções mentais* e as *acções de controlo*.

Acções Físicas: Tais acções são específicas do domínio. Actuam directamente sobre o estado do mundo externo ao agente, e no qual este se encontra situado. Por exemplo, se o agente for executar a acção física de pegar num bloco de madeira, então o actuador associado à acção indicada faz, de facto, mover um braço robótico ou qualquer outro dispositivo, de modo a realmente pegar no bloco.

Acções Mentais: Ao contrário das acções físicas, que provocavam mudanças no estado do mundo externo ao agente, as acções mentais afectam o estado do espaço de crenças do agente. Essa alteração normalmente dá-se em resultado da execução de uma acção ou de uma inferência.

Exemplos de acções mentais existentes no sistema OK-BDI são `BELIEVE` e `DISBELIEVE`. A primeira garante que uma dada proposição se torna acreditada no espaço de crenças do agente, tornando-a acreditada nesse espaço, e desacreditando a sua negação, se porventura ela era acreditada no momento. A segunda age de modo recíproco mas análogo.

Estas acções podem, evidentemente, tornar o espaço de crenças inconsistente. Daí a necessidade de usar um sistema de manutenção de verdade (TMS) o que, normalmente, facilita a implementação das duas acções mentais acima descritas, dado serem primitivas usuais destes sistemas.

As Acções de Controlo: Os planos em OK-BDI são, eles mesmos, acções, apesar de compostas, como já vimos anteriormente. São um tipo particular de acções: as *acções de controlo*. Estas acções provocam alterações nas intenções do agente quanto à execução de acções.

Existem vários tipos de acções de controlo, descrevendo um leque bastante alargado e

muito expressivo de diferentes estruturas de planos que o agente pode executar. Nomeadamente, encontramos:

`SEQUENCE(`$\mathtt{a}_1$`,`$\mathtt{a}_2$`)`: Indica que duas acções deve ser executados numa dada ordem. Permite construir *planos lineares*.

`DoONE(`$\mathtt{a}_1$`,...,`$\mathtt{a}_n$`)`: Permite a escolha não-determinística de uma acção de entre um conjunto de acções possíveis.

`DoALL(`$\mathtt{a}_1$`,...,`$\mathtt{a}_n$`)`: Efectua a ordenação não determinística de um conjunto de acções, executando todas as acções desse conjunto por uma ordem não especificada.

`IF((`$\mathtt{p}_1$`,`$\mathtt{a}_1$`),...,(`$\mathtt{p}_n$`,`$\mathtt{a}_n$`))`: Permite a que uma acção de entre um conjunto seja executada condicionalmente, escolhendo-se aquela para a qual a proposição associada $\mathtt{p}_i$ seja verdadeira.

`ITERATE((`$\mathtt{p}_1$`,`$\mathtt{a}_1$`),...,(`$\mathtt{p}_n$`,`$\mathtt{a}_n$`))`: Uma acção de entre um conjunto é executada desde que a proposição $\mathtt{p}_i$ que lhe está associada (diferente para cada acção) for verdadeira. Isto repetir-se-à até que nenhuma condição se verifique.

`ACHIEVE(p)`: Esta acção indica a intenção de tornar uma dada proposição verdadeira, através da execução de uma acção ou plano, se necessário.

`WITHSOME(x,y,...)(p(x,y,...),a(x,y,...))`: Esta é uma *acção qualificada*, isto é, que se refere a entidades conceptuais não completamente especificadas (correspondendo, grosso-modo, aos quantificadores em LPO). procura alguns x,y,etc. que satisfaçam p e aplica a acção a sobre eles.

`WITHALL(x,y,...)(p(x,y,...),a(x,y,...))`: Tal como no caso da acção anterior, estamos perante uma acção qualificada. É similar a `WITHSOME`, mas aplica a acção a todas as instanciações possíveis e não apenas a uma de entre elas.

Os Transformadores

Os *transformador* (*transformer*, no original) são proposições cuja natureza engloba os conceitos usuais de inferência e acção. De um modo geral, um transformador é um par de entidades na forma $(\langle\alpha\rangle, \langle\beta\rangle)$, onde tanto $\langle\alpha\rangle$ como $\langle\beta\rangle$ podem especificar crenças (proposições) ou acções. Quando ambas as partes de um transformador representam crenças, então estamos em presença do equivalente a uma regra de inferência, no sentido clássico. Se uma das partes for uma crença e a outra uma acção, então o transformador pode representar quais as pré-condições ou os efeitos de essa acção, ou ainda que acção deve ser executada pelo sistema em reacção à introdução nele de uma dada proposição. Finalmente, se ambas as partes de um transformador forem acções, então estamos na presença, por exemplo, da definição de uma acção em termos de outras acções.

Os transformadores são usados durante o processo de acção/inferência. O seu funcionamento pode ser ser visto como transformando crenças ou acções noutras crenças ou acções, daí o seu nome. Podem ser usados em inferência progressiva ou regressiva, embora algumas classes de transformadores possam apenas se aplicar de uma destas maneiras e não de ambas. Quando usado em inferência progressiva, um transformador pode ser encarado como representado: "após o agente acreditar ou ter a intenção de efectuar $\langle\alpha\rangle$, passa a acreditar ou a ter a intenção de executar $\langle\beta\rangle$". Quando usado em inferência regressiva, a interpretação é: "se o agente quer acreditar ou executar $\langle\alpha\rangle$, deve primeiro acreditar ou executar $\langle\beta\rangle$".

De acordo com a natureza de $\langle\alpha\rangle$ e $\langle\beta\rangle$, podemos distinguir quatro subclasses de transformadores: *belief-belief*, *belief-act*, *act-belief* e *act-act*.

Transformadores *Belief-Belief*

Os transformadores *Belief-Belief* correspondem às regras de inferência no sentido clássico. Um exemplo de um transformador deste tipo é `AntCq`, que representa as regras de antecedente/consequente, e que pode ser escrito usando a notação

$$\langle\alpha\rangle \rightarrow \langle\beta\rangle$$

Por exemplo, para representar que todos os mamíferos são animais, poderíamos ter

$$\forall \texttt{x[Isa(x,MAMIFERO)} \rightarrow \texttt{Isa(x,ANIMAL)]}$$

Estes transformadores podem ser usados tanto em inferência progressiva, como em inferência regressiva.

Transformadores *Belief-Act*

Estes transformadores, em que $\langle\alpha\rangle$ é uma crença e $\langle\beta\rangle$ é uma acção, são usados para representar as pré-condições de uma acção ou para modelar a reactividade.

Isto é conseguido recorrendo a duas sub-classes. Assim, `PreconditionAct` pode ser usado em inferência regressiva para conhecer as pré-condições de uma dada acção. Por exemplo, para indicar que para abrir uma porta esta deve estar destrancada, podemos escrever

$$\forall \texttt{x[PreconditionAct(Destrancada(x),ABRIR(x))]}$$

Em inferência progressiva, podemos usar o transformador `WhenDo`, usado para especificar como deve o agente reagir a uma dada situação. Por exemplo, se sempre que o agente encontra uma porta a deve abrir, temos

$$\forall \texttt{x[WhenDo(Porta(x),ABRIR(x))]}$$

Transformadores *Act-Belief*

Estes transformadores podem ser usados para especificar os efeitos de uma acção e para designar um plano viável para atingir um dado objectivo. A subclasse `ActEffect` pode ser usada em inferência progressiva para que o agente passa a creditar os efeitos de $\langle\alpha\rangle$. Pode ser usada em inferência regressiva no processo de geração de planos, dado que indica quais as acções a executar para que uma dada proposição se verifique.

Por exemplo, para indicar que o efeito de abrir uma porta é ela passar a estar aberta, deve escrever-se:

$$\forall \texttt{x[ActEffect(ABRIR(x),Aberta(x))]}$$

A outra classe de transformadores de este tipo é `PlanGoal`. Os transformadores deste tipo podem ser usados somente durante a inferência regressiva para decompor o atingir de um objectivo $\langle\beta\rangle$ na execução do plano $\langle\alpha\rangle$. Por exemplo, para representar que um plano para atingir o objectivo de ter a porta A aberta é executar a acção abrir, podemos escrever:

```
PlanGoal(Aberta(A),ABRIR(A))
```

Finalmente, na inferência regressiva pode usar-se uma outra classe de transformadores, os transformadores `DoIf`[2]. Estes podem ser interpretados como representando "se o agente quer saber se acredita em $\langle\beta\rangle$, deve executar $\langle\alpha\rangle$". A informação de que, para saber qual a cor da porta A, devemos olhar para ela, pode ser representada como:

```
DoIf(OLHAR(A),Cor(A,?cor))
```

O actuador associado à acção `OLHAR` encarregar-se-á de instanciar a variável livre `?cor` com o valor correcto.

[2]Designados originalmente por `DoWhen` [Kumar 1993], mas posteriormente apresentados como `DoIf` [Kumar e Shapiro 1994]

Transformadores *Act-Act*

O último tipo de transformadores considerados pelo OK-BDI são os transformadores `Act-Act`. Estes são usados para representar a decomposição de acções complexas em planos. A única subclasse existente tem a designação de `PlanAct`, onde $\langle\beta\rangle$ é um acção complexa e $\langle\alpha\rangle$ um plano que o decompõe em acções mais simples. Por exemplo, para representar que destrancar a fechadura A com a chave B temos que inserir B em A e rodar-la, temos:

```
PlanAct(SEQUENCE(INSERIR(B,A),RODAR(B)),DESTRANCAR(A,B))
```

2.1.4 O Motor Racional

O *motor racional* é a componente operacional do formalismo OK-BDI. É ele o responsável pela integração do raciocínio e acção. É composto por três tipos de métodos, definidos ao longo da hierarquia representacional descrita acima. Assim, seja ϕ uma instância de uma qualquer subclasse de `Proposition Term` e α uma instância de uma subclasse de `Act Term`, temos:

Métodos `BELIEVE`**:** Métodos que são aplicados às crenças por motivos assercionais ou de consulta. Existem duas versões:

`BELIEVE?(`ϕ`,`σ`)`: Retorna uma lista de proposições que unifica com ϕ (após aplicação das substituições em σ) que são acreditadas pelo agente explicitamente ou indirectamente através de inferência regressiva.

`BELIEVE!(`ϕ`,`σ`)`: Similar ao anterior mas retorna todas as proposições obtidas a partir de ϕ através de inferência progressiva.

`INTEND(`α`)`: Função genérica que corresponde ao agente concretizar a sua intenção de executar α.

Métodos `TRANSFORM`**:** Implementam as várias transformações possíveis. Tal como no caso dos métodos `BELIEVE`, existem duas versões:

`TRANSFORM?(`ϕ`,`σ`)`: Efectua a transformação do transformador ϕ durante a inferência regressiva.

`TRANSFORM!(`ϕ`,`σ`)`: Efectua a transformação do transformador ϕ durante a inferência progressiva.

Os métodos `BELIEVE` e `INTEND` podem ser invocados pelo próprio agente ou pelo utilizador que com ele interage. Novas crenças do sobre o mundo exterior podem ser adicionadas às crenças do agente utilizando `BELIEVE!` e o agente pode ser questionado quanto

às suas crenças com `BELIEVE?`. O método `INTEND` pode ser usado para que o agente passe a ter a intenção de executar uma dada acção.

Os métodos `BELIEVE` são usados em conjunto com os transformadores de forma a ser obtida a semântica adequada do sistema, no que diz respeito à inferência. Como estes métodos podem originar alterações no espaço de crenças do agente, é de referir que também eles se encontram intimamente relacionados com o sistema de manutenção de verdade subjacente ao OK-BDI, actualizando-o sempre que o espaço de crenças é alterado.

Estes métodos estão definidos ao longo de toda a hierarquia de classes do formalismo OK-BDI, os `BELIEVE` para `PROPOSITION TERM` e as suas subclasses, `INTEND` para `ACT TERM` e as suas subclasses e os métodos `TRANSFORM` para todos os transformadores. Adicionalmente, são especializados ao longo da hierarquia à medida das necessidades. O formalismo é, assim, extremamente fácil de estender. Basta para tal criar uma nova classe e especializar algum destes métodos para lhe dar a semântica adequada.

2.2 A Lógica das Mutações

A Lógica das Mutações [Pinto-Ferreira 1991] é uma extensão da LPO. Esta lógica foca a sua a sua atenção na *representação de acções* e assegurar o *raciocínio sobre a mudança* induzida por essas acções. Em suma, a Lógica das Mutações tem como objectivo conseguir a reificação da mudança, colocando-a ao nível da linguagem da lógica, e não a um meta-nível, como é mais usual encontrar-se.

A Lógica das Mutações tenta atingir três objectivos principais: a representação de *situações*, descrições da realidade a cada momento; permitir a representação de acções, integrando as suas pré-condições e efeitos; garantir a inferência clássica como um caso particular.

Para representar a mudança é introduzido o conceito de *mutação* (representada pelo símbolo μ), uma conectiva lógica não-standard. Esta permite dois tipos diferentes de raciocínio. O raciocínio *sincrónico*, independente da execução de qualquer acção, correspondendo, grosso modo, à inferência no sentido clássico. O raciocínio *diacrónico*, por outro lado, é o que ocorre sempre que é efectuada uma alteração na descrição da situação actual, devido ao ocorrer de uma mudança.

Em relação a estes dois tipos de inferência, note-se, ainda, que na Lógica das Mutações o conceito clássico de inferência é estendido, sendo possível derivar conjuntos de fórmulas a partir de conjuntos de fórmulas (e não apenas uma fórmula a partir de um conjunto de fórmulas, como é usual). Isto porque levar a cabo uma mudança pode dar origem a mais do que uma alteração na descrição do domínio.

Serão descritos em seguida, de forma sucinta, os vários aspectos relevantes desta lógica,

na sua vertente vocacionada para o planeamento. Começar-se-á por se discutir o seu sistema sintáctico, após o que será focado, de forma muito breve, o sistema semântico. Em seguida, discutir-se-ão, na área do planeamento, quais as possíveis aplicações desta lógica.

2.2.1 O Sistema Sintáctico

O sistema sintáctico da Lógica das Mutações é um sistema de dedução natural. Não existem axiomas, mas apenas regras de inferência. Estas regras normalmente vêm associadas às várias conectivas lógicas existentes, podendo-se encontrar, pelo menos, duas regras referentes a cada conectiva, uma referente à sua introdução, e outra à sua eliminação.

Fórmulas em Lógica das Mutações

Os conceitos de função, predicado, termo, e outros conceitos básicos são idênticos ao encontrado na LPO. Assim, apenas serão descritos aqui aqueles que diferem de algum modo do encontrado nessa lógica.

A linguagem $\mathcal{L}$, da Lógica das Mutações, pode ser vista como composta por dois conjuntos disjuntos de fórmulas: $\mathcal{L}_{LPO}$, correspondendo às fórmulas da LPO, e $\mathcal{L}_{\mu}$, cujos os elementos são as várias mutações representáveis de acordo com as regras de construção de fórmulas bem formadas da Lógica das Mutações.

As fórmulas bem formadas ($fbfs$) da Lógica das Mutações englobam todas as $fbfs$ da LPO, bem como aquelas que utilizam a conectiva μ, construídas do seguinte modo:

Definição 2.1: Regra de criação de mutações

Se $\{B_1, B_2, ..., B_n\}$ e $\{C_1, C_2, ..., C_n\}$ são conjuntos de fórmulas atómicas ou negações de fórmulas atómicas (o que, no seu conjunto, se designa por *fórmulas literais*), tais que uma fórmula e a sua negação não pertencem no mesmo conjunto, então,

$\mu(\{B_1, B_2, ..., B_n\}; \{C_1, C_2, ..., C_n\})$

é uma fbf de $\mathcal{L}_{\mu}$, denominada *mutação*.

□

Os conjuntos $\{B_1, B_2, ..., B_n\}$ e $\{C_1, C_2, ..., C_n\}$ contêm, respectivamente, o que se pode informalmente designar como as *pré-condições* e as *pós-condições* da mutação. As pré-condições são as fórmulas que devem ser verdadeiras na descrição do mundo antes da aplicação da mutação. As pós-condições indicam quais as fórmulas que passam a ser verdadeiras depois de ocorrer a mudança. As fórmulas em $B_1, B_2, ..., B_n$ que não ocorrerem em $C_1, C_2, ..., C_n$ deixarão de ser acreditadas na descrição do mundo após a mudança.

Apesar de se ter já visto como construir as fórmulas bem formadas da Lógica das Mutações, o formalismo não se centra nelas. De facto, consideram-se as *fórmulas bem formadas suportadas*. Torna-se assim possível manter um registo de dependências entre as fórmulas, permitindo distinguir entre as hipóteses e as fórmulas obtidas através de inferência. Terão, ainda, como se verá adiante, um papel preponderante na aplicação da Lógica das Mutações ao planeamento.

Assim, uma fbf suportada na Lógica das Mutações é um par ordenado $\langle A, \xi \rangle$ em que A é uma fórmula de $\mathcal{L}$ e ξ é um conjunto finito de fórmulas de $\mathcal{L}$, designado por *conjunto de suporte de* A. As fórmulas em ξ são aquelas que se encontram na base da derivação de A. Se ξ for um conjunto singular contendo apenas A, diz-se que A é uma *fórmula primitiva* ou *hipótese*. Uma mesma fórmula A pode ter mais do que um conjunto de suporte, correspondendo a várias possíveis *derivações* para A.

As Regras de Inferência

Como já foi referido acima, a inferência na Lógica das Mutações é um sistema de dedução natural, o que envolve duas regras, uma de introdução e outra de eliminação, associadas a cada conectiva lógica.

As regras de inferência usadas pela Lógica das Mutações são, à excepção das relacionadas com as mutações, muito semelhantes às encontradas, por exemplo, na LPO (à excepção, claro de estas deverem lidar com os conjuntos de suporte). Assim, apenas serão aqui mostradas, a título de exemplo, algumas das regras existentes, para além das relacionadas com as mutações (uma descrição mais detalhada das regras existentes pode ser encontrada em [Pinto-Ferreira 1991, Pinto-Ferreira e Martins 1992]).

Na definição das regras de inferência estão envolvidos alguns conceitos auxiliares, definidos em seguida:

Definição 2.2: Conjunto negação

Seja ϵ um conjunto de fórmulas. O *conjunto negação* de ϵ, denotado por $\neg\epsilon$, é constituído pelas fórmulas que resultam da negação de todas as fórmulas de ϵ.

□

Definição 2.3: Disjunção e conjunção de um conjunto de fórmulas

Seja $\epsilon = \{A_1, ..., A_n\}$ um conjunto de fórmulas. As *disjunções e conjunções desses conjuntos*, denotadas, respectivamente, por $\vee\epsilon = \vee(\{A_1, ..., A_n\})$ e $\wedge\epsilon = \wedge(\{A_1, ..., A_n\})$, representam, respectivamente, as fórmulas $(A_1 \vee ... \vee A_n)$ e $(A_1 \wedge ... \wedge A_n)$.

□

Estas definições são de carácter puramente notacional, funcionando como abreviaturas úteis nas definições que se seguem.

Definição 2.4: Conjunto de fórmulas do conjunto de fórmulas suportadas

Seja $\mathcal{B}$ um conjunto de fórmulas suportadas. O *conjunto de fórmulas do conjunto de fórmulas suportadas* $\mathcal{B}$ é, por definição,

$F(\mathcal{B}) = \{x \mid \langle x, \xi \rangle \in \mathcal{B}\}$

□

Definição 2.5: Fórmulas primitivas geradas por um conjunto de fórmulas

Seja φ um conjunto de fórmulas de $\mathcal{L}$, com $\varphi = \{A_1, A_2, ..., A_n\}$. O conjunto de *fórmulas suportadas primitivas geradas pelo conjunto de fórmulas* φ é, por definição,

$\mathcal{P}(\varphi) = \{\langle A_1, \{A_1\}\rangle, \langle A_2, \{A_2\}\rangle, ..., \langle A_n, \{A_n\}\rangle\}$

□

Definição 2.6: Conjunto de fórmulas suportadas acreditadas de um conjunto de fórmulas suportadas

Seja $\mathcal{B}$ um conjunto de fórmulas suportadas de $\mathcal{L}$. O conjunto de fórmulas suportadas *acreditadas* de $\mathcal{B}$ é, por definição,

$\mathcal{A}(\mathcal{B}) = \{\langle x, \xi \rangle \mid [(\langle x, \xi \rangle \in \mathcal{B}) \wedge (\forall y \in \xi, \langle y, \{y\}\rangle \in \mathcal{B})]\}$

O conjunto $\mathcal{A}(\mathcal{B})$ diz-se conjunto de fórmulas suportadas acreditadas.

□

Definição 2.7: Conjunto suportado de um conjunto de fórmulas

Seja $\mathcal{B}$ um conjunto de fórmulas suportadas e seja $\delta = F(\mathcal{B})$. Se $\delta' \subseteq \delta$, o *conjunto suportado do conjunto de fórmulas* δ' *relativamente ao conjunto de fórmulas suportadas* $\mathcal{B}$ é, por definição,

$\mathcal{S}(\delta', \mathcal{B}) = \mathcal{A}(\{\langle x, \xi \rangle \mid (\langle x, \xi \rangle \in \mathcal{B}) \wedge (x \in \delta')\})$

□

Podem, agora, ser enunciadas as regras de inferência. A título de exemplo, temos:

Regra de Introdução de Primitivas

(IPri) A partir do conjunto de $fbfs$suportadas acreditadas de $\mathcal{D}$ e de qualquer fórmula A pode-se inferir

$$\mathcal{D}' = \mathcal{D} \cup \{\langle A, \{A\}\rangle\}$$

Regra de Eliminação de Primitivas

(EPri) A partir do conjunto de fórmulas suportadas acreditadas $\mathcal{D}$ tal que $\langle A, \{A\}\rangle \in \mathcal{D}$ pode-se inferir

$$\mathcal{D}' = \mathcal{A}(\mathcal{D} - \{\langle A, \{A\}\rangle\})$$

Note-se que o conjunto resultante contém exclusivamente as fórmulas cujo conjunto de suporte não contém a primitiva eliminada.

Regra de Introdução da Implicação

($\rightarrow$I) Dado o conjunto de fórmulas suportadas acreditadas $\mathcal{D}$ tal que $\langle B, \omega\rangle \in \mathcal{D}$ e de qualquer fórmula A, pode-se inferir

$$\mathcal{D}' = \mathcal{D} \cup \{\langle A \rightarrow B, \omega - \{A\}\rangle\}$$

Notar que do conjunto de suporte da fórmula suportada inferida é removida a hipótese que constitui o antecedente.

Regra de Modus Ponens

(MP) A partir do conjunto de fórmulas suportadas acreditadas $\mathcal{D}$ tal que $\langle A, \omega_1\rangle \in \mathcal{D}$ e $\langle A \rightarrow B, \omega_2\rangle \in \mathcal{D}$ pode-se inferir

$$\mathcal{D}' = \mathcal{D} \cup \{\langle B, \omega_1 \cup \omega_2\rangle\}$$

Regra de Introdução da Mutação

Esta regra é responsável pela derivação de mutações a partir de mutações já existentes no conjunto de partida $\mathcal{D}$. A mutação obtida corresponde, em termos de efeitos, à execução sequencial das duas mutações, $\mu_1 = \mu(\psi_1; \varphi_1)$ e $\mu_2 = \mu(\psi_2; \varphi_2)$, que lhe dão origem.

Existem várias pré-condições para a utilização desta regra de inferência. Antes que mais, é necessário que as duas mutações sejam sequenciáveis, isto é, que a aplicação da

primeira não remova pré-condições necessárias à aplicação da segunda (garantido pelas duas primeiras condições da lista abaixo).

A mutação resultante irá conter as pré-condições de μ_1, bem como todas as pré-condições de μ_2 não atingidas já pela aplicação de μ_1, ou seja, $\psi_1 \cup (\psi_2 - \varphi_1)$.

As pós-condições da mutação derivada, por outro lado, serão todas as pós-condições de μ_2 e também aquelas de μ_1 que não foram removidas pela aplicação de μ_2, ou seja $\psi_2 \cup (\psi_1 - \varphi_2)$.

Finalmente, para que a mutação obtida seja válida, é necessário que as suas pré e pós-condições cumpram os requisitos especificados na definição da conectiva mutação, ou seja, devem ser conjuntos consistentes de fórmulas literais.

(μI) A Partir de um conjunto de fórmulas suportadas acreditadas $\mathcal{D}$ tal que

$\langle \mu(\psi_1; \varphi_1), \omega_1 \rangle \in \mathcal{D}$ e $\langle \mu(\psi_2; \varphi_2), \omega_2 \rangle \in \mathcal{D}$, pode-se inferir

$\mathcal{D}' = \mathcal{D} \cup \{(\langle \mu(\psi_1 \cup (\psi_2 - \varphi_1); \psi_2 \cup (\psi_1 - \varphi_2)), \omega_1 \cup \omega_2 \rangle\}$, desde que

(i) $\psi_1 \cap (\psi_2 - \varphi_1) = 0$

(ii) $\neg\varphi_1 \cap \psi_2 = 0$

(iii) $(\psi_1 \cup (\psi_2 - \varphi_1))$ e $(\varphi_2 \cup (\varphi_1 - \psi_2))$ sejam conjuntos consistentes de fórmulas literais.

Regra de Eliminação da Mutação

Esta regra corresponde á aplicação de uma mutação. A sua aplicação obriga a que, no conjunto de partida $\mathcal{D}$, esteja presente não só a mutação $\mu = \mu(\psi; \varphi)$ a aplicar, como também as fórmulas que constituem as pré-condições dessa mutação. No conjunto inferido, $\mathcal{D}'$, serão removidas as pré-condições não presentes em φ, transitando para esse conjunto as restantes.

De modo a evitar contradições, todas as fórmulas em $\mathcal{D}$ que sejam a negação de quaisquer pós-condições são previamente removidas de $\mathcal{D}$.

Informalmente, o conjunto inferido $\mathcal{D}'$ é constituído pela união dos seguintes conjuntos de $fbfs$ suportadas:

(i) o conjunto de mutações existentes em $\mathcal{D}$, que não sofre alterações

(ii) o conjunto de fórmulas auto-suportadas correspondente ao conjunto de fórmulas φ

(iii) o conjunto de fórmulas suportadas correspondente ao conjunto de fórmulas σ'', que contém as fórmulas suportadas presentes em $\mathcal{D}$ que não foram afectadas pela mutação.

O conjunto σ'' pode ser calculado de acordo com o processo descrito em seguida.

Inicialmente, considera-se de σ, o conjunto de todas as fórmulas de $\mathcal{L}_{LPO}$ presentes em $\mathcal{D}$. Se esse conjunto for inconsistente com φ é efectuada a mudança mínima em σ de modo a obter um conjunto de fórmulas σ' que não permita derivar (no sentido da LPO), a fórmula $\vee(\neg\varphi)$. Caso contrário, σ' será, simplesmente, igual a σ.

Na posse de σ', é-lhe efectuada uma nova mudança mínima de forma a não ser possível derivar, no sentido da LPO, a fórmula $\vee((\psi - \varphi) \cup (\neg\varphi))$ (isto é, deixe de ser possível, também, derivar as pré-condições que devem ser removidas). Obtem-se, assim, σ''.

O conceito de mudança mínima é descrito em seguida.

Definição 2.8: Subconjuntos máximos a partir dos quais não é possível derivar uma fórmula

Seja σ um conjunto de fórmulas de $\mathcal{L}_{LPO}$ e sejam A e B fórmulas. O *conjunto de todos os subconjuntos máximos do conjunto σ dos quais não é possível derivar a fórmula A* é, por definição:

$(\sigma \downarrow A) = \{\sigma' \subseteq \sigma \mid (\sigma' \nvdash A) \wedge [(B \in \sigma \wedge (B \notin \sigma') \rightarrow \sigma' \vdash (B \rightarrow A)]\}$

Ou seja, o conjunto σ' é tal que $\sigma' \cup \{B\} \vdash A$.
Cada um dos conjuntos de $(\sigma \downarrow A)$ corresponde à *mudança mínima* a efectuar no conjunto σ por forma a que do conjunto resultante σ' não seja possível derivar, em LPO, a fórmula A.

□

Formalmente, temos:

(μE) A partir do conjunto de fórmulas suportadas acreditadas $\mathcal{D}$ tal que

(i) $\mathcal{D} = \mathcal{D}_\mu \cup \mathcal{D}_{LPO}$,
$\mathcal{D}_\mu = \{\langle x, \xi\rangle \in \mathcal{D} \mid x \in \mathcal{L}_\mu\}$ e $\mathcal{D}_{LPO} = \{\langle x, \xi\rangle \in \mathcal{D} \mid x \in \mathcal{L}_{LPO}\}$;

(ii) $\langle \mu(\psi; \varphi), \omega\rangle \in \mathcal{D}_\mu$;

(iii) $\sigma = F(\mathcal{D}_{LPO})$ verifica $\psi \in \sigma$;

(iv) Se ω não for um conjunto unitário, então
$(A, B \in \psi) \wedge (A \neq B) \rightarrow \exists\omega_1\omega_2 \mid (\langle A, \omega_1\rangle, \langle B, \omega_2\rangle \in \mathcal{D}) \wedge (\omega_1 \cap \omega_2) = 0$,

pode-se inferir

$\mathcal{D} = \mathcal{D}_\mu \cup \mathcal{P}(\varphi) \cup \mathcal{S}(\sigma'', \mathcal{D})$, se existirem conjuntos σ' e σ'' tais que,

$\sigma'' \in (\sigma' \downarrow \vee((\psi - \varphi) \cup (\neg\varphi))$, com $\sigma' = \sigma$ se $\sigma \cup \varphi$ for consistente, e $\sigma' \in (\sigma \downarrow \vee(\neg\varphi))$, caso contrário.

Note-se que, ao encontrar as mudanças mínimas que levam a σ'', podem-se obter vários conjuntos válidos. Assim, a aplicação de uma mutação pode levar a diferentes mundos ou extensões.

2.2.2 O Sistema Semântico

O sistema semântico da Lógica das Mutações não é de particular importância para este trabalho, pelo que apenas serão focados, sucintamente, os seus aspectos mais relevantes.

De modo a modelar as diferentes situações que resultam da ocorrência da mudança, foi usado o conceito de *mundo possível*, tal como introduzido por Leibnitz e popularizado por Kripke para a semântica de sistemas formais [Kripke 1971]. Assim, dir-se-á que uma proposição, em Lógica das Mutações é verdadeira ou falsa em relação a um dado mundo possível ω_i.

Os vários mundos possíveis estão ligados entre si pela ocorrência de mudança, que pode ser representada por uma relação de acessibilidade entre os mundos.

Assim, um modelo na semântica da Lógica das Mutações é um triplo, $< W, R, V >$, em que W é o conjunto de mundos existentes (correspondendo às várias interpretações possíveis para um conjunto de $fbfs$, R é a relação de acessibilidade entre os mundos e V é a função de valoração que indica se uma dada fórmula atómica é ou não verdadeira em cada um dos mundos. A relação de acessibilidade entre mundos, R, é dada pelas mutações consideradas.

Existe um processo automático para determinar, a partir de um conjunto de fórmulas δ o conjunto de mundos possíveis (a partir de o subconjunto de fórmulas de LPO, δ_N) e de relações entre eles (a partir de o subconjunto de fórmulas que representam mutações, δ_μ).

Está-se, assim, perante uma semântica bem definida. É demonstrável que a Lógica das Mutações, apesar de ser *coerente* (*correct*, em Inglês), não é *completa*. Isto é, todos os conjuntos de fórmulas derivados pelo sistema sintáctico são indicados como satisfeitos pelo sistema semântico, mas há conjuntos de fórmulas indicados como satisfeitos pelo sistema semântico que não são deriváveis pelo sistema sintáctico. A não-completude deve-se ao facto de quando se calcula uma mudança mínima e se remove uma fórmula, nada garante a remoção de todas as suas consequências lógicas.

Adicionalmente, a Lógica das Mutações *não é decidível*, dado que na aplicação da regra da eliminação das mutações, deve-se determinar um subconjunto de um conjunto dado do qual não é possível derivar, em LPO, um determinado conjunto de fórmulas. Não existe nenhum processo para o fazer que seja decidível. Assim, também a Lógica das Mutações é indecidível.

2.3 Aplicações da Lógica das Mutações

Existem, a priori, quatro tipos diferentes de tarefas que um sistema baseado na Lógica das Mutações pode desempenhar: o planeamento linear, a simulação de execução de acções, a explicação e o treino. Cada uma destas tarefas é determinística e corresponde a um mecanismo de raciocínio diferente.

No caso do *planeamento linear*, são fornecidas ao sistema duas descrições (eventualmente incompletas) de situações em dois instantes distintos, bem como as descrições das acções permitidas. O sistema deve determinar, caso exista, uma sequência de designadores de acções que, quando aplicada à situação inicial, conduza à situação final. Os planos obtidos são *planos lineares* dado serem constituídos por uma sequência finita de designadores de acções. Mais do que fornecer um qualquer plano, o planeador deve obedecer a alguns critérios de preferência de modo a fornecer, de entre os vários planos possíveis, um que se destaque dos demais em termos de qualidade.

A *simulação* é um tipo de raciocínio em que é conhecida a descrição da situação no instante inicial e um plano linear susceptível de lhe ser aplicado, O sistema deve produzir cada uma das sucessivas descrições das situações resultantes da aplicação sucessiva das acções no plano. Pretende-se, em particular, obter uma descrição da situação final.

Um tipo de raciocínio recíproco da simulação é a *explicação*. Aqui, conhecemos a descrição da situação final e o plano que a ela conduz a partir de um qualquer estado inicial, e desejamos conhecer as descrições de todas as situações entre ambos, nomeadamente a do estado inicial.

Um plano linear pode ser visto como uma "acção composta". Efectivamente, quando aplicado à descrição de uma situação, dá origem a outra situação. O *treino* é o processo através do qual o planeador associa um plano já existente a um novo designador de acção, determinando as respectivas pré e pós-condições. Consegue-se, assim, evitar replanear para casos em que algum planeamento já foi efectuado no passado. Obtêm-se, deste modo, grandes ganhos de eficiência, dada a onerosidade computacional do processo de planeamento.

2.3.1 Aplicação da Lógica das Mutações ao Planeamento

Foi já visto como é possível escrever todas as fórmulas bem formadas da linguagem $\mathcal{L}$, bem como as fórmulas bem formadas suportadas. No entanto, para utilizar a lógica para a criação de um planeador, é necessário definir ainda um conjunto de estruturas acessórias.

O que foi apresentado até ao momento permite o raciocínio sobre mudança. Não existe, porém, nenhum meio de permitir representar vários instantes de tempo, o que é, sem dúvida de particular importância na criação de um planeador. Deve ser possível capturar as características fundamentais de cada *situação*. Outro tipo de informação necessário para a criação de um planeador é um meio de registar, de algum modo, as várias acções executadas em sucessão a partir de uma situação inicial, ou seja, de se obter o resultado no planeamento propriamente dito.

Assim, é definida uma nova linguagem, $\mathcal{L}^*$, semelhante a $\mathcal{L}$. No entanto, nesta nova linguagem, consideram-se *operadores de mutação*, e não mutações, são criadas novas estruturas, nomeadamente as *descrições* e *base de conhecimento*, e as regras de inferência são alteradas para lidarem com estas alterações, passando a designar-se por *regras de computação*.

Os Operadores de Mutação

Todas as $fbfs$ de $\mathcal{L}$ são, também, $fbfs$ de $\mathcal{L}^*$, excepto as mutações. Estas são substituídas pelos *operadores de mutação*, construídos como se segue:

Definição 2.9: Regra de criação operadores de mutação

Se $B_1, B_2, ..., B_n$ e $C_1, C_2, ..., C_n$ são conjuntos de fórmulas literais de $\mathcal{L}_{LPO}$ e $\mathbf{a}$ é um termo, então,

$\mu(B_1, B_2, ..., B_n; C_1, C_2, ..., C_n; \mathbf{a})$

é uma fbf de $\mathcal{L}^*_\mu$, denominada *operador de mutação*.

□

Foi introduzido um terceiro elemento no operador de mutação, denominado *designador de acção*, com o qual nos poderemos referir a uma dada acção executada mediante a aplicação de uma mutação.

As Novas Estruturas

Para as aplicações descritas, existem dois tipos de conhecimento com os quais deve haver uma preocupação: os conhecimentos *temporal* e *atemporal*. O conhecimento temporal está

associado a instantes de tempo particulares, e reporta-se, normalmente, às características do domínio que se vão alterando ao longo do tempo. O conhecimento atemporal, por outro lado, é imutável ao longo do tempo.

Por outro lado, importa distinguir entre *conhecimento primitivo* (adquirido por hipótese ou já inferido) e *conhecimento não-primitivo* (ainda não inferido, mas derivável a partir do conhecimento primitivo). A esta distinção está subjacente a dicotomia entre características *primárias* e *secundárias* do mundo modelado. Em princípio, a representação do mundo devia poder reduzir-se às primeiras.

Para representar as diferentes situações do mundo, é importante reter não só a caracterização do que é relevante no mundo em cada situação, como, também, a indicação do instante correspondente. A representação dos aspectos relevantes do mundo pode ser feita mediante um conjunto de fórmulas bem formadas que descrevem o mundo modelado admitindo que o conhecimento foi adquirido por hipótese ou observação directa (conhecimento primitivo). A representação do instante de tempo correspondente à situação pode ser feita recorrendo à sequência de acções aplicadas desde um instante inicial. Assume-se, claro, que a única de alterar a situação actual é a aplicação de uma acção. A esta sequência de acções é dado o nome de *designador de situação*.

Uma *descrição* é, pois, um par ordenado $[\Phi, s]$, em que Φ é um conjunto de fórmulas de $\mathcal{L}^*$ e um termo s designador de situação. Esta estrutura permite representar o conhecimento primitivo temporal.

Desta forma evita-se a representação explicita dos instantes de tempo através de uma variável presente nas fórmulas da linguagem, como no caso do calculo situacional. Esse conhecimento é colocado ao nível do planeador.

A representação do designador de situação da situação obtida a partir da situação s por aplicação da acção cujo designador é $\mathbf{a}$ é $do(\mathbf{a}, s)$.

Na posse da descrição é simples saber se uma dada característica está presente no mundo modelado no instante de tempo correspondente. Isto ocorrerá se a fórmula que representa essa característica pertencer ao conjunto de fórmulas que caracteriza a situação ou for derivável a partir deste.

A informação das dependências entre as fórmulas cuja existência decorre do raciocínio (adquiridas ao longo da utilização do sistema, por inferência) e as fórmulas na descrição é atemporal. Se este conhecimento for registado de alguma forma, torna-se simples conhecer as características derivadas que são válidas numa dada descrição, sem necessidade de voltar a fazer a inferência.

Assim, introduz-se o conceito de *base de conhecimento*, como o conjunto de todas as fórmulas suportadas cuja existência decorre do processo de raciocínio. Este conhecimento atemporal pode até ser mantido entre processos de planeamento (dentro do mesmo

domínio), bastando abandonar-se o temporal.

Finalmente, surge a noção de *espaço de conhecimento*. Este é um par ordenado que agrega a base de conhecimento, $\mathcal{KB}$ com o conjunto de todas as descrições já obtidas, $\mathcal{DS}$, denotando-se por $[\![\mathcal{KB}, \mathcal{DS}]\!]$

As Regras de Computação

As *regras de computação* ao dispor do planeador linear são equivalentes às regras de inferência da Lógica das Mutações, mas permitindo lidar com as novas estruturas e operadores.

Existem dois tipos de regras, de acordo com o tipo de raciocínio que efectuam: as *regras de computação sincrónicas* e as *regras de computação diacrónicas*.

As *regras de computação sincrónicas* são aquelas em que novas fórmulas são inferidas a partir das contidas numa descrição, sendo colocadas na base de conhecimento, constituindo novo conhecimento atemporal. Uma regra de computação sincrónica transforma o espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$ em $[\![\mathcal{KB}', \mathcal{DS}]\!]$, com $\mathcal{KB} \subseteq \mathcal{KB}'$.

As *regras de computação diacrónicas*, por outro lado, correspondem ao raciocínio sobre acções, sendo criadas novas descrições. A sua aplicação transforma o espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$ em $[\![\mathcal{KB}', \mathcal{DS}']\!]$, com $\mathcal{KB} \subseteq \mathcal{KB}'$ e $\mathcal{DS} \subseteq \mathcal{DS}''$.

As únicas regras de computação diacrónica são as da introdução e remoção do operador de mutação e as da introdução e eliminação de primitivas. Estas duas últimas são diacrónicas dado que, se estamos a alterar o conhecimento primitivo, isto reflecte decerto uma alteração observada directamente no mundo modelado, o que nos conduz a uma nova descrição.

A título de exemplo e de modo a permitir uma comparação, serão enunciadas em seguida as regras de computação equivalentes às regras de inferência apresentadas na secção 2.2.1.

Regra de Introdução de Primitivas

(IPri) A partir do espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$, em que $[\Phi, s] \in \mathcal{DS}$, de qualquer fórmula A, e de um termo $\mathbf{a}$, pode-se concluir,

$$[\![\mathcal{KB} \cup \{\langle A, \{A\}\rangle\}, \mathcal{DS} \cup \{[\Phi \cup \{A\}, do(\mathbf{a}, s)]\}]\!]$$

Esta regra de computação é diacrónica, dado que altera $\mathcal{DS}$, e $\mathbf{a}$ é o designador da acção de observação.

Regra de Eliminação de Primitivas

A partir do espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$, em que $[\Phi, s] \in \mathcal{DS}$, de qualquer fórmula $A \in \Phi$, e de um termo $\mathbf{a}$, pode-se concluir,

$$[\![\mathcal{KB}, \mathcal{DS} \cup \{[\Phi - \{A\}, do(\mathbf{a}, s)]\}]\!]$$

Esta regra de computação é diacrónica, dado que também altera $\mathcal{DS}$.

Note-se que ao remover A da descrição, se deixam automaticamente de acreditar em todas as fórmulas que se tivessem baseado em A para a sua derivação.

Regra de Introdução da Implicação

($\rightarrow$I) A partir do espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$, em que $\langle B, \xi \rangle \in \mathcal{KB}$ e de qualquer fórmula $A \in \xi$, pode-se inferir

$$[\![\mathcal{KB} \cup \{\langle A \rightarrow B, \{\xi - \{A\}\rangle\}, \mathcal{DS}]\!]$$

Regra de Modus Ponens

(MP) A partir do espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$, em que $\langle A, \xi_1 \rangle \in \mathcal{KB}$ e $\langle A \rightarrow B, \xi_2 \rangle \in \mathcal{KB}$, pode-se concluir

$$[\![\mathcal{KB} \cup \{\langle B, \xi_1 \cup \xi_2 \rangle\}, \mathcal{DS}]\!]$$

Regra de Introdução da Mutação

(μI) A partir do espaço de conhecimento $[\![\mathcal{KB}, \mathcal{DS}]\!]$ em que

$\langle \mu(\psi_1; \varphi_1; \mathbf{a}_1), \xi_1 \rangle \in \mathcal{KB}$ e $\langle \mu(\psi_2; \varphi_2; \mathbf{a}_2), \xi_2 \rangle \in \mathcal{KB}$, com

(i) $\psi_1 \cap (\psi_2 - \varphi_1) = 0$

(ii) $\neg\varphi_1 \cap \psi_2 = 0$

(iii) $(\psi_1 \cup (\psi_2 - \varphi_1))$ e $(\varphi_2 \cup (\varphi_1 - \psi_2))$ sejam conjuntos consistentes de fórmulas literais,

pode-se concluir,

$$[\![\mathcal{KB} \cup \{\langle \mu(\psi_1 \cup (\psi_2 - \varphi_1); \psi_2 \cup (\varphi_1 - \psi_2); \mathbf{a}_2 \circ \mathbf{a}_1), \xi_1 \cup \xi_2 \rangle\}, \mathcal{DS}]\!]$$

onde '$\circ$' representa o símbolo usual de composição de funções. A aplicação do novo operador de mutação a uma descrição $[\Phi, s] \in \mathcal{DS}$ gera a descrição $[\Phi'', do(\mathbf{a}_2, do(\mathbf{a}_1, s))] \in \mathcal{DS}$.

A sequência de designadores de acção $\ll \mathbf{a}_1, \mathbf{a}_2 \gg$ constitui um plano linear associado ao novo operador de mutação derivado a partir das mutações $\langle \mu(\psi_1; \varphi_1; \mathbf{a}_1), \xi_1 \rangle$ e $\langle \mu(\psi_2; \varphi_2; \mathbf{a}_2), \xi_2 \rangle$.

Regra de Eliminação da Mutação

(μE) A partir de $[\![\mathcal{KB}, \mathcal{DS}]\!]$ em que

(i) $[\Phi, s] \in \mathcal{DS}$ com $\Phi = \sigma \cup \sigma_\mu$, onde $\sigma = \Phi \cap \mathcal{L}_{LPO}$ e $\sigma_\mu = \Phi \cap \mathcal{L}_\mu$;

(ii) $\langle \mu(\psi; \varphi; \mathbf{a}), \xi \rangle \in \mathcal{KB}, \xi \in \Phi$;

(iii) $\psi \in \sigma$;

(iv) Se ω não for um conjunto unitário, então
$(A, B \in \psi) \wedge (A \neq B) \rightarrow \exists \xi_1 \xi_2 (\xi_1, \xi_2 \subseteq \sigma) \wedge (\langle A, \xi_1 \rangle, \langle B, \xi_2 \rangle \in \mathcal{KB}) \wedge (\xi_1 \cap \xi_2) = 0$;

pode-se concluir

$[\![\mathcal{KB} \cup \Xi, \mathcal{DS} \cup \Sigma]\!]$, em que

(i) $\Xi = \mathcal{P}(\varphi)$;

(ii) $\Sigma = \{[\Phi'_1, do(\mathbf{a}, s)]\} \cup ... \cup \{[\Phi'_r, do(\mathbf{a}, s)]\}$

onde, para $1 \leq i \leq r$,

$\Phi'_i = \sigma_\mu \cup \varphi \cup \sigma''_i$

$\sigma''_i \in (\sigma'_j \downarrow \vee((\psi - \varphi) \cup (\neg\varphi)))$ com $\sigma'_j = \sigma$ se $\sigma \cup \varphi$ for consistente e $\sigma'_j \in (\sigma \downarrow \vee(\neg\varphi))$, caso contrário.

2.4 Comparação dos Formalismos

Será, nas secções seguintes, feita uma comparação relativamente detalhada dos aspectos mais relevantes de cada um dos formalismos apresentados. De agora em diante, quando se encontrar uma referência à Lógica das Mutações, estar-se-á a falar do formalismo extendido para a aplicação ao planeamento (a linguagem $\mathcal{L}^*$), dado ser nessa vertente em particular que a Lógica das Mutações é de interesse para o presente trabalho.

2.4.1 Introdução

Tendo em vista a integração dos dois formalismos estudados, o OK-BDI e a Lógica das Mutações, torna-se necessário estudar, do ponto de vista das suas diversas componentes, as similaridades e diferenças existentes entre eles. Só assim se tornará possível restringir a nossa atenção aos factores em que, realmente, algum trabalho deve ser feito de forma a conseguir usar os dois formalismos de forma integrada com sucesso.

Ambos os formalismos possuem um poder expressivo não negligenciável, e ambos se referem explicitamente à ocorrência de mudanças. No entanto, como se verá, esses conceitos não coincidem nos dois casos.

De igual modo, tanto a Lógica das Mutações como o OK-BDI têm alguns pontos de contacto. Ambos, na sua génese, usaram ideias já existentes no SNePS (*Semantic Network Processing System*), um sistema de representação do conhecimento baseado em redes semânticas [Shapiro 1979, Shapiro e Rapaport 1987, Shapiro e The SNePS Implementation Group 1998]. Embora isto facilite o estabelecimento de paralelismos entre os dois formalismos, possibilita, também, que o mesmo conceito base, em SNePS, tenha evoluído em duas direcções similares mas no entanto distintas em cada um dos formalismos.

Assim, iremos, em cada uma das secções seguintes, descrever quais as principais diferenças encontradas entre os dois formalismos, a três níveis diferentes: *sintaxe e nível de expressividade*, *capacidades de inferência* e *semântica*. Na posse desse conhecimento poderemos, então, avançar na direcção de eliminar essas diferenças, conseguindo a integração desejada.

2.4.2 Sintaxe e Nível de Expressividade

Grande parte das diferenças entre os dois formalismos podem imediatamente ser encontradas a nível sintáctico. O que se evidencia de imediato é o facto de que, enquanto que o sistema OK-BDI define uma hierarquia de classes com as quais serão construídos os diversos elementos da linguagem, no caso da Lógica das Mutações estamos perante uma lógica no sentido clássico, onde existem os conceitos de predicado, termo e outros normalmente associados a esta forma de representação de conhecimento. De facto, dado tratar-se de uma extensão da LPO, todos os conceitos desta lógica se lhe podem aplicar com excepção das extensões que caracterizam a Lógica das Mutações.

As classes do OK-BDI, por outro lado, permitem a construção de, entre outros tipos de conhecimento, o equivalente a todas as proposições, termos e restantes elementos da LPO.

Efectivamente, é sabido que o OK-BDI que é um formalismo isomórfico ao SNePS [Kumar 1993]. Logo, para além ser possível representar todas as proposições da LPO, é também possível representar todas as $fbfs$ apenas representáveis em SNePS usando

as conectivas lógicas não-standard. Mais uma vez, esta é uma diferença apenas a nível sintáctico, e não de expressividade, dado que estas primitivas não-standard podem ser vistas como uma abreviatura de combinações das primitivas usuais.

2.4.3 Diferenças a Nível Sincrónico

A nível sincrónico ambos os formalismos têm uma expressividade compatível. O OK-BDI pode ser visto como tendo um nível de expressividade que engloba o da Lógica das Mutações. Isto deve-se ao facto de o OK-BDI ser extensível, mediante a sua estrutura hierárquica de classes e meta-classes. Este facto sugere, por um lado, que não ocorrerão grandes problemas, ao integrar os dois formalismos e, por outro, que se deve tentar englobar a Lógica das Mutações no OK-BDI e não vice-versa.

Os problemas aparecem quando na análise a representação das acções e a expressividade a nível diacrónico, como se verá em seguida.

2.4.4 Diferenças a Nível Diacrónico

É a este nível que as principais diferenças entre os formalismos se encontram. Com efeito, os formalismos em estudo tratam o problema da mudança e da sua representação de formas completamente distintas.

A Mudança no OK-BDI

No OK-BDI, embora se permita a representação de acções, essa representação possui características diferentes do encontrado noutros formalismos. Isto pode parecer peculiar, dado que o que motivou o desenvolvimento desse sistema foi o desejo de colocar o actor ao serviço do raciocinador e vice-versa. Embora isso tenha sido conseguido, o actor encontra-se situado no "mundo real". No OK-BDI as acções referem-se a acções reais, e não a *operadores*, que não são mais do que meras abstracções de acções sobre um modelo do mundo. Ou seja, embora o OK-BDI contemple, de forma integrada, a existência de acções, estas são efectivamente executadas por um agente actuador no mundo real. O sistema lidará imediatamente dos resultados dessas acções.

Por exemplo, se for efectuada uma acção de inspecção, o sistema irá imediatamente incluir na base de conhecimento os factos que possam advir dessa inspecção. No entanto não mantém nenhum registo de que se evoluiu no mundo para uma situação diferente da inicial. Efectivamente, não existe nenhuma parte do formalismo dedicada a manter um registo das dos vários instantes de tempo. Não se tem a informação sobre em que instantes são verdadeiras as várias proposições que constituem a base de conhecimento.

A Mudança na Lógica das Mutações

Na Lógica das Mutações, a forma de lidar com as mudanças é completamente diferente do encontrado no OK-BDI. Com efeito, a Lógica das Mutações é um formalismo completamente "incorpóreo", no sentido de que não prevê nenhuma interacção com um mundo exterior. Logo, as acções, como definidas no OK-BDI, não têm correspondente neste formalismo.

O que define a Lógica das Mutações é o facto de a mudança se encontrar reificada sob a forma de *operadores de mutação* (designados no futuro simplesmente como *mutações*). Uma mutação não é mais do que, em última análise, uma representação em lógica de um *operador*, no sentido clássico (do STRIPS, por exemplo), com as suas pré-condições e efeitos, com a grande diferença de se encontrar ao nível da linguagem e não num meta-nível.

As mutações podem ser usadas em qualquer momento, e fazer parte de qualquer fórmula. Como já foi descrito anteriormente, existem regras de computação próprias para lidar com elas, permitindo, nomeadamente, executar uma mutação (μI), e agrupar duas mutações numa única mutação resultante (μE). A Introdução de Mutações não é mais do que a criação de um macro-operador que engloba o funcionamento de duas mutações. A eliminação de uma mutação corresponde a aplicar o operador que lhe está subjacente, verificando-se as suas condições de aplicabilidade e calculando-se os seus efeitos.

Em virtude desta reificação da mudança, a Lógica das Mutações mantém um registo de qual a situação actual, incluindo toda a informação relevante, tal como a situação e a mutação que lhe deu origem, usando as estruturas já referidas, as *descrições* e a *base de conhecimento*. Como as $fbfs$ da Lógica das Mutações são $fbfs$ suportadas, mantendo-se um registo das dependências de cada uma delas, é possível manter na base de conhecimento simultaneamente os factos referentes a diferentes descrições, acreditando-se alternadamente em conjuntos diferentes deles de acordo com a descrição actual (o que vai de encontro ao paradigma dos "óculos mágicos" [Pinto-Ferreira 1991], em que o sistema só "vê" a cada momento o que é acreditado na descrição correspondente). Saber quais as $fbfs$ que são verdadeiras em cada descrição é trivial, bastando para tal consultar o seu registo de dependências.

O raciocínio diacrónico é, assim, não só possível, como simples de efectuar, dado existirem todos os mecanismos necessários para tal.

2.4.5 Análise Conjunta

Considerando tudo o que foi referido em relação aos dois formalismos, torna-se evidente qual a principal diferença entre eles em termos de expressividade: a forma em como é tratada a mudança.

O raciocínio sobre acções no sentido clássico supõe que o formalismo tem uma maneira de registar e distinguir o que se pode considerar como válido em cada situação, de forma a poder raciocinar sobre os efeitos e pré-condições de uma acção sem, de facto, a executar no mundo real. O OK-BDI é algo deficiente nesse ponto. Embora esse raciocínio seja possível (até certo ponto, dado que nem sempre podemos ter a certeza do resultado de uma acção executada no mundo real), não existem mecanismos que permitam lidar com vários instantes de tempo (alguns no passado e outros, eventualmente, no futuro) em simultâneo, pelo que o raciocinio sobre mudança é bastante difícil de conseguir, e nunca de forma directa e simples. Em suma, podemos afirmar que o OK-BDI não permite o raciocínio diacrónico (pelo menos de forma directa), ao passo que a a Lógica das Mutações o permite.

Apesar das suas limitações quanto ao raciocínio sobre mudança, o OK-BDI pode interactuar (de forma integrada) com o mundo através das acções e dos actuador que lhes estão associados. Na Lógica das Mutações, não é prevista qualquer interacção com o mundo real, encontrando-se, no entanto, a mudança reificada sob a forma das mutações (e, indirectamente, das descrições), existindo mecanismos e estruturas para suportar o raciocínio diacrónico.

Assim, a nível de representação e expressividade, o trabalho desenvolvido consiste em extender o formalismo OK-BDI (o pela sua sua natureza é possível e, á partida, mais simples do que extender a Lógica das Mutações) de forma a conseguir, também nele, efectuar raciocínio diacrónico, de forma integrada com o raciocínio sincrónico e a ligação com o actuador já nele existentes.

Para evitar futuras confusões, de agora em diante referir-nos-emos às acções no sentido do OK-BDI (i.e., que interagem directamente com o mundo real), como *acções*, e às que se reportam a um modelo interno do mundo, tal como ocorre na Lógica das Mutações, como *mutações*. Adicionalmente, o desencadear de uma acção será referido como *execução* dessa acção e o desencadear de uma mutação como *aplicação* da mesma.

2.4.6 Mecanismos de Inferência

A nível da inferência, os formalismos também divergem significativamente. Veremos, em seguida, como se processa a inferência em cada um deles e que pontos de contacto podem ser estabelecidos entre os dois métodos.

Inferência no OK-BDI

No OK-BDI, a inferência processa-se de uma forma diferente do que normalmente é encontrado nos diversos formalismos de representação de conhecimento. Não possuímos regras de inferência propriamente ditas, como é habito ocorrer nas lógicas e grande parte dos

restantes formalismos. Em vez disso, é introduzido o conceito de *transformador*, que permite modelar desde as regras de inferência clássicas, tais como o Modus-Ponens, como pré-condições de acções, ou efeitos das mesmas.

O funcionamento concreto de cada transformador é definido procedimentalmente, caso a caso, no conjunto de métodos que define o *motor racional* do OK-BDI

Inferência na Lógica das Mutações

Na Lógica das Mutações, é usado um sistema de dedução natural, em que cada conectiva lógica possui duas regras associadas: uma de *introdução* e outra de *eliminação*. As regras de inferência podem, como já se viu, ser classificadas em sincrónicas, se inferirem novo conhecimento atemporal, e diacrónicas, que correspondem ao raciocínio sobre mudança.

Análise Integrada

Mais uma vez, o formalismo OK-BDI possui expressividade suficiente para que lhe seja possível enquadrar, sem grandes problemas estruturais, novas formas de inferência.

Para a criação de novos tipos de inferência bastará, assim, definir novos transformadores com a semântica desejada. Essa semântica ser-lhes-à conferida procedimentalmente especializando os métodos necessários.

Como o OK-BDI possui já um sistema do tipo TMS responsável por manter as dependências entre as regras, não haverá problemas em integrar essa faceta da Lógica das Mutações (as $fbfs$suportadas).

Quanto às regras diacrónicas, a sua inclusão no OK-BDI irá implicar alterações mais profundas. De facto, dado que o sistema não possui, à partida, um modo de efectuar esse tipo de inferência, será necessário não só criar transformadores que reflictam o funcionamento dessas regras, como de alguma forma registar a situação em que cada forma pode é acreditada. Deve, assim, introduzir-se o conceito de *descrição* no sistema. Até certo ponto, este conceito já está presente, dada a existência de um TMS associado ao sistema, mantendo-se pois um *contexto* que, como será discutido adiante, poderá ser usado para os fins em causa. As regras referentes às mutações poderão, por exemplo, ser modeladas como novos transformadores crença-crença, que, para além de agir em conformidade com os efeitos das mutações, alteram o contexto actual para o contexto em que esses efeitos se verificam.

2.4.7 Semântica

Em termos semânticos, pouco há a referir. Embora a Lógica das Mutações possua uma semântica bem definida, o mesmo não se passa com o OK-BDI. A semântica das suas várias componentes é dada procedimentalmente, estando embebida no sistema, o que não permite o estudo formal da mesma.

Logo, não é possível que um sistema integrado tenha, por sua vez, uma semântica bem definida. O melhor que se conseguirá fazer será criar procedimentos que reflictam, para aspectos particulares, o funcionamento operacional de pontos concretos da Lógica das Mutações

Capítulo 3

Formalismo Integrado

Neste capítulo é descrito o modo através do qual é efectuada a integração da Lógica das Mutações no formalismo OK-BDI. Começarão por ser enumerados os aspectos sobre os quais a integração vai incidir, seguindo-se uma descrição da integração propriamente dita.

São aqui criadas as bases para permitir a integração de um sistema planeador no OK-BDI. Essa integração será descrita atempadamente em capítulos posteriores.

3.1 Introdução

A comparação feita no capítulo anterior dos dois formalismos a integrar permitiu identificar os principais pontos de divergência entre estes.

O primeiro desses pontos, sobre os quais este capítulo se irá debruçar, é *a capacidade de lidar com vários estados do mundo*. Na Lógica das Mutações é possível lidar com múltiplas descrições do mundo, resultantes da aplicação das várias mutações. Isto permite conhecer e inferir propriedades do mundo em diversos estados, o que é essencial para conseguir efectuar raciocínio diacrónico e planeamento.

No OK-BDI, por outro lado, não existem mecanismos integrados no sistema para manter de forma directa vários modelos internos do mundo em diferentes instantes de tempo. Pode considerar-se que o modelo do mundo que o agente tem em cada momento reflecte as condições do mundo nesse mesmo instante. Logo, impõe-se encontrar uma forma de alterar o OK-BDI de modo a que também ele passe a considerar, de forma simples, vários estados do mundo.

O segundo aspecto a ter em conta é a *forma de lidar com as acções*. Na Lógica das Mutações existem as mutações, que permitem alterar a representação interna do mundo de acordo com os efeitos da execução de uma dada acção. No OK-BDI encontramos acções

que, como já vimos, se reportam ao mundo real, e não a um modelo do mesmo.

Se queremos usar de algum modo o OK-BDI de forma análoga à Lógica das Mutações, implementando um sistema de planeamento sobre ele, torna-se, também, necessário criar os mecanismos através dos quais as mutações ou um seu equivalente possam ser expressas. Isto está relacionado com o que foi dito em relação à capacidade de representar modelos do mundo em diferentes instantes, na medida em que qualquer representação de acções se irá basear necessariamente nessa capacidade.

3.2 Representação dos Diferentes Instantes

A forma encontrada para conseguir representar diferentes instantes do mundo de forma simples no OK-BDI baseou-se numa característica já possuída pelo mesmo.

O espaço de crenças do agente no OK-BDI é modelado por um conjunto de proposições. Essas proposições podem classificadas como primitivas ou derivadas. As primeiras são as obtidas do mundo por observação directa, ou foram adicionadas pelo utilizador. As segundas são as que foram obtidas por inferência a partir de outras proposições.

Para ajudar a manter a origem e as dependências entre as várias proposições, o OK-BDI possui subjacente ao seu formalismo um sistema de manutenção de verdade, ou TMS. Esse sistema é, nomeadamente, do tipo ATMS (*Assumption-based Truth Maintenance System* [de Kleer 1986], ou sistema de manutenção de verdade baseado em suposições).

3.2.1 Os ATMS

Os sistemas ATMS e os TMS no geral devem o seu nome ao facto de que permitem detectar quando o sistema acredita num conjunto de proposições contraditórias, providenciando mecanismos para eliminar essas contradições e, assim, "manter a verdade" no sistema.

Num ATMS cada proposição na base de conhecimento é representada por um *nó*. É estabelecida uma *rede de dependências* entre esses nós. A cada nó são associados outros, de acordo com as proposições (representadas por esses outros nós) que lhe deram origem. Cada conjunto de proposições que dá origem a uma outra é designado como *justificação* da mesma. Dado que uma mesma proposição pode ser derivada de mais do que uma maneira, pode ter mais do que uma justificação associada (figura 3.1). Aos nós cuja crença depende apenas de si mesmos, correspondem as proposições que representam conhecimento primitivo, sendo designadas por *hipóteses* [de Kleer 1986].

Cada nó é rotulado com um conjunto de conjuntos. Os elementos de cada um desses conjuntos são as hipóteses que devem ser acreditadas para que a proposição representada por esse nó o seja. Estamos perante um conjunto de conjuntos dado que, como já vimos,

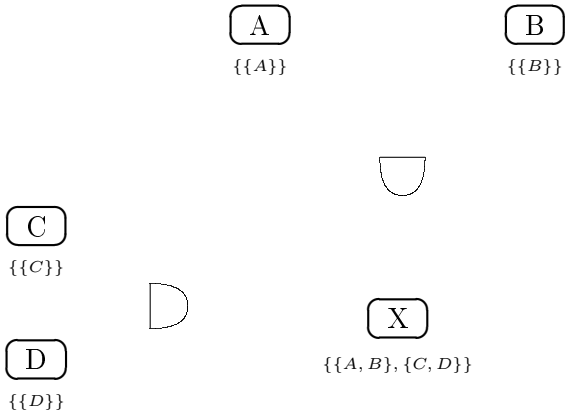


Figura 3.1: Exemplo de um ATMS

uma mesma proposição pode ser derivada de mais do que uma maneira. Os conjuntos dos rótulos têm origem nas diferentes justificações do nó, existindo métodos para os calcular e actualizar automaticamente. Um nó correspondente a uma hipótese, será rotulado por um conjunto singular, cujo elemento contém apenas o próprio nó. Na figura 3.1, A, B, C e D são hipóteses: não são justificadas por nenhum nó e o seu rótulo apenas as contém a si mesmas. O nó X, por outro lado, corresponde a uma proposição derivável de duas formas, a partir das proposições representadas por A e B, ou a partir das representadas por C e D. Logo, é justificado por dois conjuntos de nós, que nos indicam que X será acreditado se o forem, simultaneamente, A e B ou C e D.

Num sistema deste tipo, existe o conceito de *contexto*. Um contexto não é mais do que um conjunto de nós. Esses nós representam as proposições primitivas em que o agente acredita num dado momento. Saber se uma dada proposição é ou não acreditada num dado contexto é trivial. Basta, simplesmente, comparar o rótulo do nó a que está associada com o contexto em causa. Se o contexto for superconjunto de um qualquer conjunto de nós contido no rótulo de um nó, então estão reunidas as condições para que a proposição representada por esse nó seja acreditada.

Existem mecanismos para indicar ao sistema que conjuntos de proposições não podem ser acreditadas em simultâneo, por se estar na presença de uma contradição (por exemplo, acreditar numa proposição e na sua negação em simultâneo). Isto tornaria a base de conhecimento inconsistente, pelo que o TMS trata de remover a contradição, encontrando as premissas na base das fórmulas contraditórias e indicando quais os conjuntos mínimos de premissas que devem deixar de ser acreditados de modo a que deixe de ser possível derivar a contradição.

3.2.2 Os Contextos e os Estados do Mundo

A existência dos contextos e do contexto actual faz com que o conhecimento de um agente no OK-BDI seja modelado por um conjunto de *espaços de conhecimento*, sendo possível alternar entre os mesmos alterando o contexto actual. Os métodos `BELIEVE?` e `BELIEVE!` estão implementados de forma a apenas considerar as proposições presentes no espaço de conhecimento actual do sistema. Este parece, assim, ser um aspecto do OK-BDI no qual é possível basear, de forma natural, uma solução para o problema de conseguir representar diferentes instantes do mundo numa única representação.

O TMS do OK-BDI é encarado unicamente como uma forma de manter a verdade e evitar contradições numa descrição estática do mundo. O que em seguida será descrito cria as bases para a utilização desse TMS de forma muito mais abrangente, como mecanismo de manutenção simultânea de informação referente a diferentes instantes de tempo no sistema.

Antes que mais, deve notar-se que as instâncias da classe `Proposition Term`, em OK-BDI possuem já um campo relacionado com o TMS que está subjacente a este sistema. Esse campo é designado por `SUPPORT` e contém o conjunto de suporte da proposição. Corresponde, nada mais nada menos, do que ao rótulo do nó que representa a proposição no TMS. Isto torna directo o processo de saber se uma proposição deve ou não ser acreditada se um conjunto de proposições primitivas o forem.

Para o formalismo integrado, vamos usar esse facto para introduzir os conceitos de *descrição* e *situação* encontrados na Lógica das Mutações. Consideram-se, assim, definidas as classes `Description`, e `Situation` descritos em seguida. [1]:

Classe: `Description`

Superclasse(s): `Conceptual Entity Term`

Campos: Todas as instâncias desta classe têm os seguintes campos:

`CONTEXT`: Um conjunto de etiquetas (*labels*) que representam proposições primitivas.

`CURRENT-SITUATION`: Uma instância da classe `Situation`, indicando como se atingiu o contexto.

Métodos: Se *o* é uma instância desta classe, então:

`CONTEXT`(*o*): Devolve conjunto contido no campo `CONTEXT`.

`CURRENT-SITUATION`(*o*): Devolve conjunto contido no campo `CURRENT-SITUATION`.

Sintaxe: Apenas instâncias desta classe podem ser escritas. A sua representação escrita é o valor do campo `LABEL` seguido de ':'. A isto segue-se um par ordenado

[1] Os nomes das classes e dos campos criados no presente sistema serão em Inglês de fora a manter a coerência com os já existentes no sistema.

em que o primeiro elemento é o valor do campo `CONTEXT` e o segundo o valor de `CURRENT-SITUATION`. Por exemplo, se o for uma instância desta classe, a sua representação escrita será

$$<\texttt{LABEL}(o)>:\langle<\texttt{CONTEXT}(o)>, <\texttt{CURRENT-SITUATION}(o)>\rangle$$

Descrição: Esta classe serve para representar, a nível do formalismo OK-BDI, o conceito de descrição tal como ocorre na Lógica das Mutações. Este conceito terá, aqui, um significado muito semelhante ao encontrado nessa lógica.

De modo a obter uma expressividade semelhante à existente na Lógica das Mutações, a classe `Description` tem associadas as fórmulas primitivas que são verdade nessa mesma descrição. A partir dessas fórmulas, é possível saber quais as fórmulas derivadas que também são verdade. O valor do campo `CURRENT-SITUATION` é uma instância da classe `Situation`, que será definida em seguida. Tem o papel do *designador de situação* na Lógica das Mutações. Permitirá a um planeador que porventura use este formalismo saber que mutações foram executadas a partir de um *instante* ou *situação inicial* conduzindo, eventualmente, a esta descrição.

Classe: `Situation`

Superclasse(s): `Conceptual Entity Term`

Campos: Todas as instâncias desta classe têm os seguintes campos:

`INITIAL-CONTEXT`: Um conjunto de etiquetas (*labels*) de instâncias de `Proposition Term` que representam proposições primitivas que eram verdade na *situação inicial*.

`MUTATIONS`: O conteúdo deste campo é uma sequência de mutações, tais como definidas abaixo, indicando o caminho percorrido desde o estado inicial até à situação actual.

Métodos: Seja o uma instância desta classe e m um designador de mutação:

`INITIAL-CONTEXT(`o`)`: Devolve conjunto contido no campo `INITIAL-CONTEXT`.

`MUTATIONS(`o`)`: Devolve a sequência contido no campo `MUTATIONS`.

`ADD-MUTATION(`o,m`)`: Acrescenta uma nova mutação ao início da sequência guardada no campo `MUTATIONS`.

Sintaxe: Apenas instâncias desta classe podem ser escritas. A sua representação escrita é o valor do campo `LABEL` seguido de ':'. Então, para cada mutação contida em `MUTATIONS`, e pela ordem em que nele aparecem, aparecerá "do(" seguido desse designador de mutação, seguido de uma virgula. Ao último designador de mutação

segue-se o valor do campo `INITIAL-CONTEXT`, após o que se fecham todos os parêntesis por fechar. Por exemplo, se o for uma instância desta classe, e a sequência em `MUTATIONS` contiver n elementos, a sua representação escrita será

$$<\texttt{LABEL}(o)>:\text{do}(<\texttt{MUTATIONS}(o)>[0], \ldots, \text{do}(<\texttt{MUTATIONS}(o)>[n], <\texttt{INITIAL-CONTEXT}(o)>)) \ldots)$$

Descrição: Esta classe serve para representar, a nível do formalismo OK-BDI, o conceito de situação. Este irá ter um papel semelhante ao descritor de situação na Lógica das Mutações, permitindo conhecer que sequência de mutações deu origem a um dado estado.

Estas novas classes vão-nos permitir explicitar os vários instantes de tempo que o sistema poderá considerar. Não foi criada explicitamente nenhuma classe `Context` dado que se se recorrerá aos contextos do TMS já existente no sistema.

Para conseguir a representação das diferentes descrições do mundo, é ainda introduzido o conceito de *descrição actual*. A descrição actual será uma entidade global ao sistema, indicando a situação sobre a qual o sistema raciocina num dado momento. O valor do campo `CONTEXT` irá coincidir, a cada instante, com o valor do contexto actual do TMS do sistema. O contexto da situação actual é designado por *contexto actual*.

Se o sistema não se encontrar num processo de raciocínio diacrónico, ou seja, se não tiver sido aplicada nenhuma mutação, a descrição actual representará a situação existente no mundo. É nela que são efectuadas todas as alterações resultantes, por exemplo, das acções de inspecção e as inferências que dai possam advir. Todos os factos apreendidos por interacção com o mundo serão novas proposições primitivas do sistema. A descrição actual é alterada, sendo adicionadas ao seu contexto as novas proposições. Durante este tipo de raciocínio, a sequência de mutações contida no campo `MUTATIONS` da instância de `SITUATION` contida na situação actual será vazia.

Durante o raciocínio diacrónico, e de cada vez que for aplicada uma mutação, será criada uma nova descrição, que representa o hipotético estado do mundo após a execução da acção correspondente a essa mutação. Essa descrição passará a ser a descrição actual. A `SITUATION` é actualizada, mantendo-se no campo `INITIAL-CONTEXT` o contexto vigente antes da aplicação das mutações, e acrescentando-se a mutação recém-aplicada a `MUTATIONS`.

Quanto à manutenção dos conjuntos de origem das várias proposições, isto é já feito automaticamente pelo sistema, ao longo do processo de inferência. As proposições obtidas mediante inspecção do mundo (execução de acções correspondentes a sensores do agente) são introduzidas na base de conhecimento como hipóteses. O contexto actual é alterado pois dá-se uma passagem para uma nova situação, em que as novas proposições passam a ser acreditadas.

3.3 Representação das Mutações

A integração das mutações no OK-BDI vai ser feita de forma similar ao que é exemplificado por Kumar em [Kumar 1993] para os *quantificadores de omissão*, existentes em SNePSwD [Cravo e Martins 1993], usados para representar regras de omissão.

Nesse exemplo, Kumar refere que, para que o OK-BDI passe a considerar novos tipos de inferência, pode bastar com definir novos transformadores do tipo `Belief-Belief` (que "transformam" crenças noutras crenças) que reflictam a funcionalidade dessas novas regras de inferência.

Uma regra de omissão representada em SNePSwD toma a forma seguinte:

$$\nabla(x)[\alpha(x) \rightarrow \beta(x)]$$

onde ∇ é o quantificador por omissão e $\alpha(x)$ e $\beta(x)$ são proposições. A regra acima pode ser interpretada como representando que "quando algo é um α então, *tipicamente*, é um β".

Esta é uma nova forma de inferência que envolve duas proposições. Assim, para a representar, é criado um novo transformador, designado por `DefaultAntCq`, bem como os respectivos métodos `TRANSFORM?` e `TRANSFORM!`, para lhe dar a semântica adequada. Cada instância desse transformador corresponde a uma regra de omissão e a sua transformação representa a aplicação dessa regra.

De facto, como já foi referido no capítulo 2.1.3, os transformadores correspondem, no OK-BDI, às regras de inferência. As mutações não são mais do que um novo tipo de regra de inferência. Como tal, a integração das mutações, descrita em seguida, foi feita de forma similar ao proposto por Kumar, mediante a criação de um novo transformador.

3.3.1 Duas Alternativas de Representação

Um problema que desde logo surgiu na representação das mutações no OK-BDI foi o de qual seria a melhor forma de o fazer: representar cada mutação como um único transformador ou recorrendo a vários?

A primeira abordagem corresponde a criar um tipo de transformador que contenha em si mesmo as pré-condições da mutação e os seus efeitos. Assim, toda a informação necessária para aplicar essa mutação estará contida no transformador, podendo ser utilizada de imediato. Tendo em conta que os transformadores são proposições, ficamos, assim, com cada mutação representada por uma única proposição. Isto corresponde ao que encontramos na Lógica das Mutações.

A segunda abordagem tem origem no modo pelo qual as acções são representadas no OK-BDI. De facto, embora também as acções tenham pré-condições e efeitos, não existe um único transformador que contenha toda essa informação. Tal informação foi separada em vários transformadores de tipos diferentes. Assim, encontramos um transformador do tipo `PreconditionAct` para cada pré-condição e um transformador do tipo `ActEffect` para cada efeito. Os dados referentes a uma acção foram, assim, espalhados por diferentes proposições.

No presente trabalho foi usada uma abordagem híbrida. Por um lado, a primeira solução é a mais correcta do ponto de vista formal, ao respeitar-se a natureza da lógica das mutações. Por outro lado, para cada mutação existente no sistema deve existir uma acção física correspondente e vice-versa. Isto permitirá ao sistema usar os resultados obtidos recorrendo à utilização das mutações. A representação das acções físicas requer a especificação das suas pré-condições e efeitos, recorrendo a transformadores especializados, como já foi referido. Se o transformador que representa uma mutação contivesse também esses dados, surgiriam dois problemas. O primeiro seria a duplicação da informação referente às pré-condições e efeitos da mutação. O segundo, potencialmente mais sério, seria a não garantia da consistência entre aquilo que foi representado referente às acções físicas e o representado no referente às mutações. Poderiam surgir casos em que o sistema usava as mutações para, por exemplo, produzir um plano, que mais tarde se vem a revelar inválido dado as pré-condições e efeitos das mutações não serem idênticos aos das acções físicas propriamente ditas.

Assim, será criado um novo tipo de transformador, cada instância do qual representará uma mutação, mas nesse transformador não serão guardados os efeitos e pré-condições da mutação, recocorrendo-se aos transformadores `ActEffect` e `PreconditionAct` para os obter. Consegue-se assim uma representação mais próxima ao OK-BDI, sem deixar de ser fiel à filosofia subjacente à Lógica das Mutações em que um dos objectivos básicos é a reificação da mudança, sob a forma de proposições que a representam: as mutações. A separação de uma mutação em várias proposições é pouco natural tendo em vista essas preocupações. Assim, continua a ser possível referir a mutação como uma única proposição, mas obviam-se os problemas referidos de replicação da informação e coerência.

3.3.2 O Novo Transformador

Em consequência das considerações das secções anteriores, foi criado um novo transformador, `MutationPcEf`, que representa uma mutação.

Surgiu um problema na sua integração na hierarquia de classes do OK-BDI. Efectivamente, existem já pré-definidos no OK-BDI quatro classes distintas de transformadores: `Belief-Belief`, `Belief-Act`, `Act-Belief` e `Act-Act`. Os que correspondem a regras de inferência entre proposições são os `Belief-Belief` (os restantes envolvem acções). Ess-

es transformadores são definidos como transformando uma proposição noutra (daí o seu nome).

Por outro lado, a inferência na Lógica das Mutações caracteriza-se por gerar *conjuntos de fórmulas a partir de conjuntos de fórmulas*. Isto deve-se ao facto de, ao permitir o raciocínio diacrónico, ser possível indicar quais as alterações no mundo devido a uma dada mudança, podendo essas alterações ser mais do que uma.

O transformador necessário para a representação das mutações seria de um tipo que transforme *conjuntos de proposições em conjuntos de proposições*. Isto porque o transformador contém a informação sobre as pré-condições da mutação (tipicamente mais do que uma), que irão ser transformadas nos seus efeitos (também de número variável).

O próprio Kumar, ao exemplificar a representação dos quantificadores de omissão, está a considerar uma forma de inferência que envolve apenas duas fórmulas, podendo-se, assim, usar transformadores `Belief-Belief`. Nada é dito, no entanto, quanto à representação de regras de inferência no caso geral, em que conjuntos de proposições estão envolvidos.

É necessário dar ao significado dos transformadores `Belief-Belief` uma interpretação mais lata, em que estes representam transformações em que *apenas estão envolvidas proposições* (sejam estas proposições individuais ou conjuntos de de proposições). Só assim se pode usar correctamente esta classe como base para `MutationPcEf`. Isto não parece epistemologicamente correcto, pelo que outra solução teve que ser encontrada.

Aproveitando o facto de o OK-BDI ser facilmente extensível, foi definida uma quinta classe de transformador para capturar a semântica por nós desejada. Foi, assim, definida classe `BeliefSet-BeliefSet`, como se segue:

Classe: `BeliefSet-BeliefSet`

Superclasse(s): `Transformer, Proposition-Term, Conceptual-Entity-Term`

Campos: nenhum.

Métodos: nenhum.

Sintaxe: nenhuma

Descrição: Esta classe será a superclasse de todos os transformadores em que estejam envolvidos conjuntos de proposições, e não apenas proposições simples.

É de notar que, a nível da definição das classes e a nível implementacional, nada impediria à partida a utilização dos transformadores `Belief-Belief` para os fins em causa. A criação da nova classe foi feita tendo em vista unicamente uma maior correcção a nível do formalismo.

Esta nova classe de transformador seria a mais indicada para incorporar no OK-BDI, por exemplo, a *Lógica de Omissão de Reiter*[Reiter 1978, Reiter 1980] ou quaisquer outros formalismos em que existam processos de inferência que envolvam conjuntos de fórmulas.

Com a definição da classe `BeliefSet-BeliefSet`, é agora possível definir, para a representação das mutações, `MutationPcEf`.

Classe: `MutationPcEf`

Superclasse(s): `BeliefSet-BeliefSet`, `Transformer`, `Proposition Term`, `Conceptual Entity Term`

Campos: Todas as instâncias desta classe têm o seguinte campo:

`MUTATION`: A instância da classe `Individual Term` que representa o conceito da mutação, e que representa também o conceito da acção correspondente.

Métodos: Seja o uma instância desta classe e m um designador de mutação:

`PRECONDS`(o): Devolve conjunto de instâncias de `PropositionTerm` que são as pré-condições da mutação.

`EFFECTS`(o): Devolve conjunto de instâncias de `PropositionTerm` que são os efeitos da mutação.

`MUTATION`(o): Devolve conjunto contido no campo `MUTATION`.

Sintaxe: As instâncias desta classe podem ser escritas como descrito em seguida. Ao valor do campo `LABEL` seguem-se ':' e um triplo cujo primeiro elemento é o conjunto resultado do método `PRECONDS`, o segundo o conjunto resultante da invocação do método `EFFECTS` e o terceiro o nome do objecto em `MUTATION`:

$$<\texttt{LABEL}(o)>:\langle<\texttt{PRECONDS}(o)>,<\texttt{EFFECTS}(o)>,<\texttt{NAME(MUTATION}(o)\texttt{)}>\rangle$$

Descrição: Esta classe serve para representar, a nível do formalismo OK-BDI, as mutações.

Esta classe permite, assim, representar uma mutação em OK-BDI. Como foi já referido, deve existir uma acção física equivalente. A determinação de qual é essa acção é feita recorrendo ao valor do campo `Mutation` do transformador. O `Individual Term` nele contido, e que idêntifica a mutação, deve ser idêntico ao que caracteriza a acção física correspondente. Isto vem de encontro ao principio da Unicidade de Referência, que dita que existe uma correspondência unívoca entre cada entidade no sistema e o conceito intensional que representa. De facto, tanto a mutação como a acção correspondente representam, no fundo, facetas diferentes do mesmo conceito, pelo que é justo que sejam identificadas pela mesma instância de `Individual Term`.

Como se pode constatar, não é explicitamente guardada no transformador a informação referente às pré-condições e efeitos da mutação. No entanto, dado ser importante conhecer, a qualquer momento, essa informação, estão definidos dois métodos, `PRECONDS` e `EFFECTS` que a permitem obter. Esses métodos vão inspeccionar a base de conhecimento em busca dos transformadores `PreconditionAct` e `ActEffect` referentes à acção física correspondente à mutação em causa. É então usado um simples processo de conversão, descrito em seguida.

Considerem-se os conjuntos P o conjunto formado pelas proposições p contidas nas instâncias de `PreconditionAct` da forma `PreconditionAct(`p`,`a`)`, em que a se refere à acção física equivalente à mutação em causa. Analogamente, considere-se o conjunto E, formado pelas proposições q contidas nas instâncias de `ActEffect` da forma `PreconditionAct(`a`,`q`)`, em que q se refere à acção física equivalente à mutação em causa. Considerem-se, adicionalmente, os conjuntos E^+ e E^-, correspondendo, respectivamente, aos literais positivos e negativos de E

1. O conjunto de pré-condições da mutação é o conjunto P.
2. O conjunto dos efeitos da mutação é dado por todos os elementos de $(P - \neg E) \cup E^+$.

Com esta classe recém-definida, é possível representar todas as mutações no OK-BDI. Para demonstrar a sua utilização considere-se, por exemplo a seguinte mutação, apresentada por Pinto-Ferreira [Pinto-Ferreira 1991] na adaptação do *Problema de Yale* (*Yale Shooting Problem*) [Hanks e McDermott 1986] para a Lógica das Mutações:

$$\mu(\{loaded, alive\}; \{dead\}; shoot)$$

Esta mutação representa que, se à priori, a arma estiver carregada e a pessoa viva, então após aplicar a mutação, designada por `shoot`, essas condições deixam de se verificar e a pessoa passa a estar morta.

No formalismo OK-BDI, esta mutação será representada por uma instância da classe `MutationPcEf`. O campo `MUTATION` dessa instância irá conter a instância de `Individual Term` com o nome `shoot`. Será, simultâneamente, criada a acção física correspondente, mediante a criação de dois trasnformadores `PreconditionAct` para as pré-condições e de um transformador `ActEffect` para o efeito.

Alguns Problemas

A representação escolhida não é completamente isenta de problemas. Existem duas considerações importantes a tecer antes de prosseguir nesta dissertação.

Em primeiro lugar, esta representação não é completamente fiel às mutações, como definidas originalmente na Lógica das Mutações. Isto prende-se com o aspecto da *natureza* das proposições que pertencem aos conjuntos contidos nos campos `PRECONDS` e `EFFECTS`. Na Lógica das Mutações, é estabelecido que essas proposições devem ser fórmulas literais (fórmulas atómicas ou a negação de fórmulas atómicas). Na definição de `MutationPcEf` apresentada, o tipo dos elementos desses conjuntos é apenas restringido a ser uma instância de `Proposition Term`, ou seja, qualquer proposição, consista ela numa fórmula literal ou não.

Não existe na hierarquia OK-BDI forma de distinguir entre as fórmulas literais e as restantes. A hierarquia apenas permite destingir, separadamente, as instâncias de `Individual Term` (as constantes da linguagem), de `Variable Term` (as variáveis) e de `Proposition Term`, que engloba todas as proposições. Embora as várias instâncias de `Proposition Term` sejam fórmulas atómicas, isso não incluiria um outro tipo de fórmulas literais: as fórmulas atómicas negadas. Além disso, por polimorfismo de inclusão, todas as instâncias das sub-classes de `Proposition Term`, usadas para representar fórmulas não-atómicas, podem ser consideradas como pertencentes a essa classe.

Este problema só poderá ser eventualmente resolvido ao nível da implementação do sistema, não se permitindo a criação de mutações com fórmulas que não sejam literais.

Outro problema, cuja resolução também passa por alterações ao nível implementacional é o de garantir a existência de uma acção física correspondente a cada mutação, e vice-versa.

3.4 Modificações ao Motor Racional

Na secção acima, foi introduzido um novo transformador, `MutationPcEf`, que permite a representação de mutações em OK-BDI. A descrição feita até agora indica qual o funcionamento desse transformador a nível sintáctico. Para que ele tenha, de facto, o funcionamento desejado, é necessário alterar o motor racional do OK-BDI, para que este consiga lidar com o novo transformador.

A introdução do transformador pode dar lugar a dois tipos de alterações: *alterações para conseguir o funcionamento correcto da nova classe* e *alterações ao nível das classes já existentes*, para lidar, se necessário, com o novo transformador.

Essas alterações são efectuadas modificando os métodos que compõem o motor racional. Esses métodos são `BELIEVE!` e `BELIEVE?`, para `Proposition Term` e as suas subclasses, `TRANSFORM!` e `TRANSFORM?`, para os transformadores e `INTEND` para as acções. Esses métodos devem, se necessário, ser especializados na nova classe para reflectir o seu funcionamento.

3.4.1 Especialização dos Métodos de `MutationPcEf`

A nova classe `MutationPcEf`, ao tratar-se de um transformador, é também uma proposição. Em princípio, seria necessário alterar os métodos `BELIEVE!` e `BELIEVE?`. Nada indica, porém, que as mutações devam ser tratadas de forma especial em relação às outras sub-classes de `Proposition Term`. Para essas sub-classes, os métodos `BELIEVE` funcionam verificando a existência de transformadores `AntCq`, `DoIf` e `WhenDo` que possam ser activados em relação à proposição considerada. Esse mesmo comportamento é o desejado para o novo transformador, pelo que não existe a necessidade de especializar esses métodos. No entanto, como se verá adiante, serão por outros motivos necessárias algumas alterações à definição do método na classe `Proposition Term`.

Dado que os métodos `BELIEVE!` e `BELIEVE?` não necessitam de ser especializados, basta alterar os métodos `TRANSFORM!` e `TRANSFORM?` para a nova classe. Esses métodos definem o modo de funcionamento de um transformador em particular. Assim, é necessário um especial cuidado na sua implementação para conseguir a semântica desejada. O método `TRANSFORM!` permite efectuar inferência progressiva com o transformador. Reciprocamente, é `TRANSFORM?` que despoleta a inferência regressiva.

Para a implementação de um planeador, é necessário implementar a aplicação das mutações, equivalente à regra da eliminação da mutação, na Lógica das Mutações. Isto irá corresponder à inferência progressiva.

A existência da inferência regressiva implica a existência de uma regra de "regressão da mutação" que, a partir de uma mutação e da situação actual permite obter a situação que deu origem à actual por aplicação dessa mutação. Isto implica a existência de operadores de mutação inversos para cada uma das mutações definidas, ou de um processo sistemático que, a partir de uma descrição e de uma mutação, permita calcular a situação que poderia ter estado na origem da descrição actual por aplicação da mutação dada. As regras inversas não existem na Lógica das Mutações. Por outro lado, não é possível construir um processo que permita "regredir no tempo" como descrito. Nunca é possível saber, por exemplo, se na situação original existia ou não a negação de um dado efeito da mutação aplicada, dado que essa aplicação a retiraria caso existisse.

Pelas razões apresentadas, considera-se que o transformador `MutationPcEf` só pode ser aplicado em inferência progressiva, correspondendo à passagem para uma descrição futura.

O método `TRANSFORM!` para a classe `MutationPcEf` pode, assim, se definido como se segue:

Method
TRANSFORM!(t : MutationPcEf; σ: Substitution := NIL)
is
$P \leftarrow PRECONDS(t)$

foreach $p \in P$ *loop*
if not BELIEVE?$(p\sigma)$ *then*
return
endif
end loop

$S \leftarrow$ *SUPPORT(t)*
if a mutação foi derivada por agregação de outras mutações then
if not ENSURE-INDEPENDENCE(t) then
return
endif
endif

SIGMA2 = COMPUTE-SIGMA2(t)
Criar uma instância de Description para cada um dos elementos de SIGMA2
Colocar todas as instâncias criadas na sequência RES
DESCRIÇÃO-ACTUAL $\leftarrow$ *FIRST(RES)*

return RES
end TRANSFORM!

O processo implementado pelo método acima descrito deriva directamente da regra de inferência da Eliminação das Mutações descrita na secção 2.2.1.

Em primeiro lugar, é necessário verificar se a mutação é aplicável. Isto ocorre se as duas condições seguintes se verificarem:

- As pré-condições da mutação são demonstráveis na descrição actual.
- Se a mutação em causa for uma fórmula derivada por agregação de duas outras mutações, deve garantir-se que a remoção de uma fórmula presente no conjunto de suporte e que corresponde a pré-condições da mutação não implica a remoção automática de outra fórmula nas mesmas condições, dado que isto iria contrariar os pressupostos que permitiram a criação da mutação. A verificação de se a mutação é desse tipo pode ser efectuada verificando se a cardinalidade de algum elemento do conjunto de suporte da fórmula (designado por `S` acima) é maior do que 1 e se pelo menos dois dos seus elementos correspondem a outras mutações, dado que uma mutação pode ser uma fórmula derivada sem que tenha necessariamente sido criada por agregação de outras mutações.

Logo, o método implementado começa por verificar estas duas condições. A primeira é feita recorrendo ao método `BELIEVE?`, que indicará se é possível ou não acreditar em cada uma das pré-condições (após aplicada a substituição adequada) na descrição actual.

A segunda condição é implementada recorrendo a um método auxiliar que, dada uma mutação, a verifica: `ENSURE-INDEPENDENCE`. A definição do funcionamento desse método pode ser encontrada abaixo. É no entanto de notar que no formalismo aqui apresentado se considera que cada proposição tem como suporte um conjunto de conjuntos, correspondentes a diferentes modos de a derivar. Na Lógica das Mutações, embora a mesma proposição possa ser derivada de várias formas, correspondentes a vários conjuntos de suporte, assume-se que cada um deles irá dar origem a uma fbf suportada diferente. Assim, enquanto que os algoritmos descritos para a Lógica das Mutações lidam apenas com um conjunto (o de uma instância em particular da proposição considerada), aqui deve ser considerado o conjunto de conjuntos e aplicar a cada um deles as verificações indicadas na Lógica das Mutações.

```
Method
    ENSURE-INDEPENDENCE(t : MutationPcEf)
is
        P ← PRECONDS(t)
        ALREADY-CHECKED ← {}
        foreach p ∈ P loop
            ALREADY-CHECKED ← ALREADY-CHECKED + p
            foreach j ∈ (P−ALREADY-CHECKED) loop
                S1 ← SUPPORT(p)
                S2 ← SUPPORT(j)
                if S1∪S2 não é vazio then
                  return FALSE
                endif
            end loop
        end loop

        return TRUE
end ENSURE-INDEPENDENCE
```

Garante-se, assim, que os conjuntos de suporte de todas as pré-condições são disjuntos. Isto implica a independência entre as pré-condições e garante o critério a atingir.

Uma vez verificadas as condições para a aplicação da regra, os seus efeitos podem ser calculados. Como efeito da remoção da mutação, são criadas novas descrições em tudo idênticas à original, excepto em que são introduzidas na base de conhecimento as pós-condições da mesma, as pré-condições da mutação que não aparecem nas pós-condições da mesma são eliminadas, e são eliminadas as negações das pós-condições introduzidas. A aplicação de uma mutação pode conduzir a mais do que uma descrição porque, como já foi referido, pode haver mais do que um conjunto de mudanças mínimas para remover as negações dos efeitos da mutação e as pré-condições da mesma que não são efeitos.

Isto passa pela criação de três conjuntos intermédios, designados por σ, σ' e σ''. A

definição desses conjuntos pode ser encontrada, em termos lógicos em [Pinto-Ferreira 1991, Pinto-Ferreira e Martins 1992]. Essa definição, porém, não facilita a implementação da criação desses conjuntos. Como tal, foi desenvolvido por Larsen e Martins um algoritmo que permite a operacionalização da construção de novas descrições por aplicação de uma mutação [Larsen e Martins 1992]. Foi efectuada uma adaptação desse algoritmo para o domínio deste trabalho, descrita abaixo.

Esse algoritmo tem como ponto de partida o conjunto σ, das fórmulas primitivas acreditadas na descrição actual. Em seguida, procede da seguinte forma:

1. Calcular o conjunto σ, que contém as fórmulas primitivas acreditadas na descrição actual (ou seja, as fórmulas no contexto actual).
2. Reunir todos os conjuntos de suporte para as fórmulas que são as negações de pós-condições da mutação. O conjunto resultante é designado por *conjunto de suporte das pós-condições negadas.*
3. Reunir os conjuntos de suporte de todas as pré-condições.
4. Reunir os conjuntos de suporte para as pré-condições que não são pós-condições (que serão removidas por aplicação da mutação).
5. Calcular σ', como se segue:
   (a) Construir um conjunto de conjuntos de nós. Cada um desses conjuntos de nós deve conter um nó de cada conjunto de suporte no conjunto de suporte das pós-condições negadas. Isto confere-lhe a propriedade de que, se as proposições que contem forem retiradas do contexto da situação actual, é bloqueada a derivação da negação da negação de qualquer pós-condição. Logo, este conjunto de conjuntos é designado por *conjunto de bloqueio.*
   (b) De modo a efectuar uma mudança mínima na descrição actual, calcula-se o *conjunto de bloqueio mínimo* a partir do conjunto de bloqueio, isolando os seus elementos de menor cardinalidade.
   (c) Calcula-se a diferença entre σ e cada conjunto do conjunto de bloqueio mínimo, o que dá origem a um conjunto de conjuntos, σ'. Esse conjunto é em tudo idêntico ao contexto actual, excepto que foi bloqueada a derivação da negação das pós-condições da mutação a aplicar.
6. Já foram bloqueadas as negações das pós-condições. É agora necessário bloquear as pré-condições que não vão transitar para a situação seguinte. Para tal, σ'' é calculado a partir de σ'. Considera-se o conjunto de de suportes para as pré-condições que não são pós-condições. Então, para cada conjunto de nós γ em σ':
   (a) Eliminar os conjuntos de suporte que não são sub-conjuntos de γ, dado que isto indica que a derivação da fórmula correspondente já foi bloqueada. O conjunto resultante é designado por *conjunto de suportes revisto*

(b) Construir um conjunto de conjuntos de nós. Cada um desses conjuntos de nós deve conter um nó de cada conjunto no conjunto de suportes revisto. Isto confere-lhe a propriedade de que, se as proposições que contem forem retiradas de γ o contexto da situação actual, é bloqueada a derivação da pré-condições indesejadas, bem como a negação da negação de qualquer pós-condição (o que já tinha sido obtido em σ'. Este é o *conjunto de bloqueio* para o cálculo de σ''.

(c) Isolar os conjuntos no conjunto de bloqueio com a menor cardinalidade, de modo a obter as mudanças mínimas. Este é o o *conjunto de bloqueio mínimo* para o cálculo de σ''.

(d) Construir um conjunto de conjuntos de nós em que cada um desses conjuntos é a diferença entre γ e cada um dos elementos do conjunto de bloqueio mínimo.

O conjunto com todos os conjuntos obtidos na alínea 6d para todos os γ possíveis é o conjunto σ''.

7. São calculadas as várias descrições resultantes da aplicação da mutação juntando a cada conjunto de σ'' as pós-condições da mesma. Cada um dos conjuntos resultantes é o contexto de uma nova situação.

Para melhor compreender o algoritmo acima descrito, considere-se o seguinte exemplo (adaptado de [Larsen e Martins 1992]).

A descrição inicial é contém as seguintes $fbfs$ primitivas:

$\mu(\{P,Q\};\{R,S\};accao)$	$\langle 1,\{\{1\}\}\rangle$
$(A \vee B) \rightarrow P$	$\langle 2,\{\{2\}\}\rangle$
$(B \vee C) \rightarrow Q$	$\langle 3,\{\{3\}\}\rangle$
$(C \vee D) \rightarrow \neg R$	$\langle 4,\{\{4\}\}\rangle$
$E \rightarrow \neg S$	$\langle 5,\{\{5\}\}\rangle$
A	$\langle 5,\{\{5\}\}\rangle$
B	$\langle 6,\{\{6\}\}\rangle$
C	$\langle 7,\{\{7\}\}\rangle$
D	$\langle 8,\{\{8\}\}\rangle$

A partir destas $fbfs$ primitivas, é possível derivar as seguintes $fbfs$ derivadas que se seguem, completando assim a definição extensiva da descrição:

P	$\langle 10,\{\{2,6\},\{2,7\}\}\rangle$
Q	$\langle 11,\{\{3,7\},\{3,8\}\}\rangle$
$\neg R$	$\langle 12,\{\{4,8\},\{4,9\}\}\rangle$
$\neg S$	$\langle 13,\{\{5,9\}\}\rangle$

Supondo que se vai aplicar a mutação *acção*, o algoritmo decorre da seguinte forma:

1. Calcula-se σ:

$$\{1,2,3,4,5,6,7,8,9\}$$

2. Calcula-se o conjunto de suporte das pós-condições negadas:

$$\{\{4,8\},\{4,9\},\{5,9\}\}$$

3. Calcula-se o conjunto de suportes das pré-condições:

$$\{\{2,6\},\{2,7\},\{3,7\},\{3,8\}\}$$

4. Calcula-se o conjunto de suportes para as pré-condições que não são pós-condições. Neste caso, esse conjunto coincide com o anterior.

5. Calcula-se σ':

   (a) Determina-se o conjunto de bloqueio:

   $$\{\{4,5\},\{4,9\},\{4,5,9\},\{8,4,5\},\{8,4,9\},\{8,5,9\},\{8,9\}\}$$

   (b) Encontra-se o conjunto de bloqueio mínimo, isolando do conjunto de bloqueio os elementos de menor cardinalidade (neste caso, os que apenas têm dois elementos):

   $$\{\{4,5\},\{4,9\},\{8,9\}\}$$

   (c) Calcula-se a diferença entre σ e cada um dos elementos do conjunto de bloqueio mínimo:

   $$\{\{1,2,3,6,7,8,9\},\{1,2,3,5,6,7,8\},\{1,2,3,4,5,6,7\}\}$$

   O conjunto obtido é σ'. Note-se como é impossível, a partir de cada um dos seus elementos, derivar a negação de qualquer pós-condição (as $fbfs$ 12 e 13).

6. Calcula-se σ''. Para tal considera-se o conjunto de suportes para as pré-condições que não são pós-condições, calculado em 3. Para cada elemento γ de σ', são efectuados os seguintes passos, exemplificados para $\{1,2,3,4,5,6,7\}$:

   (a) Eliminam-se os conjuntos de suporte que não são subconjuntos de γ, obtendo-se:

   $$\{\{2,6\},\{2,7\},\{3,7\}\}$$

   (b) Calcula-se o conjunto de bloqueio:

   $$\{\{2,3\},\{2,7\},\{2,3,7\},\{2,3,6\},\{2,6,7\},\{3,6,7\},\{6,7\}\}$$

(c) Encontra-se o conjunto de bloqueio mínimo, isolando do conjunto de bloqueio os elementos de menor cardinalidade (neste caso, os que apenas têm dois elementos):

$$\{\{2,3\},\{2,7\},\{6,7\}\}$$

(d) Efectua-se a diferença entre γ e cada um dos conjuntos no conjunto de bloqueio mínimo, obtendo-se:

$$\{\{1,4,5,6,7\},\{1,3,4,5,6\},\{1,2,3,4,5\}\}$$

Repetindo o mesmo processo para todos os elementos de σ', obtem-se σ'', cujo valor é:

$$\{\{1,4,5,6,7\},\{1,3,4,5,6\},\{1,2,3,4,5\},\{1,6,7,8,9\},\{1,5,6,7,8\}\}$$

7. Cada um dos novos contextos será um dos conjuntos de σ'' ao qual serão adicionadas as $fbfs$ que constituem as pós-condições da mutação:

$$\begin{array}{ll} R & \langle 14,\{\{1,2,6\},\{1,2,7\}\}\rangle \\ S & \langle 15,\{\{1,3,7\},\{1,3,8\}\}\rangle \end{array}$$

Assim, os conjuntos de fórmulas primitivas que dão origem aos novos contextos são: $\{1,4,5,6,7,14,15\}$, $\{1,3,4,5,6,14,15\}$, $\{1,2,3,4,5,14,15\}$, $\{1,6,7,8,9,14,15\}$ ou $\{1,5,6,7,8,14,15\}$.

Este algoritmo não garante a mudança mínima, porque a composição de dois conjuntos mínimos (σ' e σ'') não é, necessariamente, um conjunto mínimo. Consegue, no entanto, bons resultados na maior parte dos casos.

O algoritmo acima é implementado pelo método `COMPUTE-SIGMA2`, definido em seguida. Esse método segue, a cada momento, os diversos passos do algoritmo. Começa por consultar o contexto actual, procurando as $fbfs$ que não sejam mutações. É, assim, construído o conjunto σ, armazenado na variável `SIGMA`. Um raciocínio similar aplica-se para determinar o conjunto de suportes das pós-condições negadas (na variável `SUPPOSNEG` e das pré-condições que não são pós-condições (`SUPPRECOND`).

A construção de σ' e σ'' (guardados, no método, nas variáveis `SIGMA1` e `SIGMA2`, respectivamente), segue um processo similar, se atentarmos ao indicado acima na descrição do algoritmo. Este processo é efectuado por uma função auxiliar, `FIND-SIGMA`.

Method
COMPUTE-SIGMA2(t : MutationPcEf; σ: Substitution := NIL)
is
SIGMA ← *{}*

```
        foreach c ∈ CURRENT-CONTEXT loop
            if not IsA(c,MutationPcEf)
              SIGMA ← SIGMA ∪ {c}
            endif
        endloop

        SUPPOSNEG ← {}
        foreach e ∈ EFFECTS(t) loop
            if BELIEVE?((¬e)σ) then
              POSNEG ← POSNEG ∪ SUPPORT((¬e)σ)
            endif
        end loop

        SUPPRECOND ← {}
        foreach p ∈ PRECONDS(t) loop
            if p ∉ EFFECTS(t) then
              SUPPRECOND ← SUPPRECOND ∪ SUPPORT(p)
            endif
        end loop

        SIGMA1 ← FIND-SIGMA(SIGMA,SUPPOSNEG)
        SIGMA2 ← {}
        foreach g ∈ SIGMA1 loop
            TEMP-SET ← {}
            foreach n ∈ SUPPRECOND loop
                if n ∈ g then
                  TEMP-SET ← TEMP-SET ∪ {n}
                endif
            end loop
        SIGMA2 ← SIGMA2 ∪ FIND-SIGMA(g,TMP-SET)
    end loop

    return SIGMA2
end COMPUTE-SIGMA2

Method
    FIND-SIGMA(REFERENCE-SET, SUPPORT-SET : conjuntos de etiquetas de nós)
is
    Colocar em BLOCKING-SET todos os conjuntos que se podem obter
    contendo um elemento de cada conjunto em SUPPORT-SET

    Encontrar a cardinalidade mínima dos elementos de BLOCKING-SET
    e remover os que não a tiverem

    RESULT ← {}
    foreach n ∈ BLOCKING-SET loop
        RESULT ← RESULT ∪ {REFERENCE-SET − n}
```

end loop

return RESULT
end FIND-SIGMA

Na posse do conjunto `SIGMA2`, o método `TRANSFORM!` pode criar os novos contextos, concluindo, assim, a aplicação da mutação. Surge, agora, um aspecto controverso. Cada elemento de `SIGMA2` irá corresponder a uma nova descrição. Para que a aplicação da mutação se considere completa, o sistema deve transitar para uma dessas novas contextos. Não é possível, no entanto, decidir, à priori, qual das descrições deve ser escolhida. Essa escolha é, normalmente, feita recorrendo à intervenção externa do utilizador. Isso, no entanto, é de evitar principalmente se se pretender que o raciocínio diacrónico incorporado no sistema possa servir como base a processos de funcionamento automático como o planeamento de acções.

Numa primeira abordagem, é possível considerar duas abordagens diferentes para este problema: a abordagem *céptica* ou a abordagem *crédula*. Tanto num caso como no outro garante-se que apenas é gerada uma descrição para a aplicação de cada mutação. Na abordagem céptica a descrição gerada contém as $fbfs$ que aparecem em *todos* os elementos de `SIGMA2`. Na abordagem crédula essa descrição contém as $fbfs$ que aparecem em *pelo menos um* dos elementos de `SIGMA2`). Nenhuma dessas abordagens é satisfatória, neste caso. Podem existir aplicações do raciocínio diacrónico em que importa conhecer todas as descrições alternativas. Essa informação deve, sempre que possível, ser mantida. Posteriormente, como o OK-BDI integra também um actuador, será possível ir verificar directamente no mundo qual a extensão a considerar.

Assim, foi seguida uma terceira via: é escolhida de entre as várias descrições alternativas uma que passará a ser a descrição actual. No entanto, a informação sobre as restantes não se perde, sendo devolvida pelo método uma sequência com todas as descrições a que a mutação pode dar origem. Essa informação pode ser aproveitada ou desprezada, de acordo com a aplicação que esteja a invocar a mutação. Numa utilização interactiva do sistema, pode ser dada ao utilizador a possibilidade de escolher entre as várias alternativas possíveis.

3.4.2 A Introdução de Mutações

Existe uma regra de inferência na Lógica das Mutações respeitante às mutações que ainda não foi aqui considerada: a regra da Introdução de Mutações. Essa regra permite criar uma nova mutação a partir de duas outras, correspondendo à aplicação em sequência dessas mutações. Na abordagem seguida neste trabalho, assume-se que a cada mutação existente no sistema corresponde uma acção física. Isto ocorre, como já foi atrás referido, para permitir que o processo de planeamento possa gerar planos utilizáveis pelo actuador.

Torna-se, assim, necessário que o processo de criação de novas mutações por agregação de mutações já existentes crie, também, acções físicas que correspondam à mutação recém-criada. A criação de novas acções físicas primitivas não é possível, mas é possível agregar, com acções de controlo, diversas acções primitivas. Assim, sempre que for derivado um novo plano, se as pré-condições para a Introdução de novas mutações forem pré-satisfeitas, será possível criar as acções físicas correspondentes.

Para implementar o comportamento da regra da Introdução de Mutações, é definido o método `AGGREGATE`, como se segue:

Method
AGGREGATE(t1 : MutationPcEf; t2 : MutationPcEf)
is
if $((PRECONDS(t1) \cap (PRECONDS(t2) - EFFECTS(t1)) = \{\}$ *e*
$\{\neg x | x \in EFFECTS(t1)\} \cap PRECONDS(t2) = \{\}$ *e*
$((PRECONDS(t1) \cup (PRECONDS(t2)\text{-}EFFECTS(t1)))$ *e*
$(EFFECTS(t2) \cup (EFFECTS(t1)\text{-}PRECONDS(t2)))$
são conjuntos consistentes de fórmulas literais)))
then
Criar uma nova instância de `Individual Term`*, na, para representar a nova acção, tal que:*
NAME(na)=NAME(MUTATION(t1))+'-'+NAME(MUTATION(t2))
Criar uma nova instância a de PlanAct tal que
ACT(a)=na
PLAN(a)=SEQUENCE aplicado às acções correspondentes a t1 e t2
$PrC = PRECONDS(t1) \cup (PRECONDS(t2) - EFFECTS(t1))$
$EffMut = (PRECONDS(t2) \cup (PRECONDS(t1) - EFFECTS(t2)))$
$Eff = (EffMut - PrC) \cup \neg(PrC - EffMut)$,
Para cada um dos elementos de PrC criar uma nova instância de `PreconditionAct` *e fazer BELIEVE!(t) dessa instância.*
Para cada um dos elementos de Eff, criar uma nova instância de `ActEffect` *e fazer BELIEVE!(t) dessa instância.*
Criar uma nova instância t de MutationPcEf tal que
MUTATION(t)=na
Efectuar os registos de dependências necessários no TMS
Invocar o método BELIEVE!(t)
endif
end AGGREGATE!

Este método, como o nome indica, efectua a agregação de duas mutações. Começa por verificar as condições necessárias para a aplicação da regra de inferência e se estas forem satisfeitas, cria e introduz na base de conhecimento uma nova mutação de acordo com o ditado por essa regra.

Adicionalmente, pelos motivos apresentados, é criada uma nova acção física composta

que corresponde à execução sequencial das acções que correspondem às mutações agregadas. O nome da nova mutação e acção será, por convenção, criado a partir dos nomes das mutações envolvidas na agregação. A mesma instância de `Individual Term` é usada para representar o conceito inerente à nova acção e à nova mutação. É efectuado, também, o registo no TMS do OK-BDI que indica que a nova mutação e acção depende das duas que foram agregadas para lhes dar origem.

São, ainda, criadas instâncias de `PrecondictionAct` e `ActEffect` para as pré-condições e efeitos da nova acção. Estes transformadores são necessários para futuras referências à nova mutação.

O método `AGGREGATE` terá particular importância em aplicações dos transformadores `MutationPcEf` como o planeamento, tal como se verá no capítulo 4.

3.4.3 Alterações Aos Métodos Já Existentes

Em `Proposition Term`, `BELIEVE!` é responsável por assercionar uma proposição e efectuar inferência progressiva na base de conhecimento recém-alterada, verificando-se se a nova proposição existe como antecedente de alguma implicação (representada pelo transformador `AntCq`) ou se desencadeia uma regra reactiva (`WhenDo`). O método `BELIEVE?` tenta descobrir se a proposição é ou não verdadeira efectuando inferência regressiva. Verifica se a proposição é o consequente de alguma implicação (`AntCq`), tentando nesse caso provar os seus antecedentes, ou se é possível consultar o mundo para determinar a sua validade (`DoIf`).

Este comportamento levanta um *problema de sincronismo*, entre o mundo real e o estado interno do sistema. Quando o OK-BDI não possuia forma de efectuar raciocínio diacrónico, o seu estado interno reportava-se, sempre, ao estado actual do mundo. Assim, é legítimo, para determinar a validade de uma proposição, fazer uma inspecção directa do mundo, recorrendo a um transformador `DoIf`. De igual modo, é plausível desencadear efeitos reactivos no mundo devidos ao acrescentar de uma dada proposição à base de conhecimento, recorrendo a um transformador `WhenDo`.

Quando em presença do raciocínio diacrónico, porém, um tal comportamento já não é válido. Considere-se o caso em que, por algum motivo, foi efectuado o raciocínio diacrónico mediante a aplicação de uma mutação, sob a forma de um transformador `MutationPcEf`. Após essa aplicação, embora o estado do mundo real se ter mantido inalterado, o estado interno do sistema reporta-se, agora, a uma nova situação que representa o estado em que o mundo se encontraria após executar a acção correspondente à mutação considerada. Os dois estados, o interno e o externo, encontram-se *dessincronizados*.

Quando, nessa situação, o sistema quiser verificar se uma dada proposição é verdadeira, utilizando o método `BELIEVE?`, continua a ser válido usar os transformadores `AntCq`, dado

que o conhecimento neles contido não depende do estado do mundo. No entanto, a consulta ao mundo real recorrendo a `WhenDo` perde o sentido, dado que isso só permitiria conhecer propriedades do mundo num instante que, no estado interno do sistema, se encontra no passado. Um raciocínio análogo pode ser elaborado quanto à utilização dos transformadores `DoIf` na inferência progressiva.

Um exemplo que torna patente o problema enunciado seria o caso em que, da base de conhecimento do sistema, fazem parte as seguintes proposições:

$Plano(A)$	$\langle 1,\{\{1\}\}\rangle$
$Plano(A) \rightarrow Suporte(A)$	$\langle 2,\{\{2\}\}\rangle$
$Cor(A, Azul) \rightarrow Vidro(A)$	$\langle 3,\{\{3\}\}\rangle$
$DoIf(Olhar(A), Cor(A, ?cor))$	$\langle 4,\{\{4\}\}\rangle$
$\mu(\{Cor(A, Azul)\}, \{Cor(A, Vermelho)\}, Pintar(A))$	$\langle 5,\{\{5\}\}\rangle$

O contexto actual, nesta situação, seria $\{1, 2, 3, 4, 5, 6\}$. A base de conhecimento pode agora ser usada para responder várias questões. Por exemplo, `BELIEVE?(Suporte(A))` dirá se é ou não possível acreditar na proposição `Suporte(A)` no contexto actua. Para tal, verifica se essa proposição ocorre como o consequente de alguma regra ou se existe algum transformador `DoIf` que a mencione, e que sejam acreditados no contexto actual. É descoberta uma regra de inferência (regra 2), pelo que o sistema tenta então provar o antecedente da mesma, o que é trivialmente conseguido.

De forma análoga, é possível perguntar uma questão como `BELIEVE?(Vidro(A))`. O mesmo raciocínio é seguido, encontrando-se a regra 3. Para provar o antecedente desta regra, é descoberto um transformador `DoIf`. Donde, a acção é executada e, assumindo que, no mundo real, o objecto A era, de facto, azul, o antecedente da regra 3 encontra-se demonstrado e pode-se concluir que o objecto A é de vidro.

Se, agora, se decidir aplicar a mutação (fórmula 5), o estado interno do sistema passa para uma situação posterior, em que a pré-condição da mutação é removida do contexto e é adicionada uma nova fórmula à base de conhecimento:

$$Cor(A, Vermelho) \qquad \langle 6,\{\{6\}\}\rangle$$

A mutação é aplicável dado que é possível, como indicado acima, demonstrar a verdade da pré-condição da mutação, recorrendo a uma inspecção directa do mundo. O novo contexto é, assim, $\{1, 2, 3, 4, 5, 6\}$. O sistema entrou numa fase de raciocínio diacrónico, em que simula o que ocorreria se fosse executada a acção de pintar o objecto A de Vermelho. O que acontece agora em relação às perguntas feitas anteriormente à base de conhecimento? A primeira, `BELIEVE?(Suporte(A))`, decorre da mesma forma que foi anteriormente descrita, dado que todas as fórmulas envolvidas continuam a ser verdade no novo contexto. A

segunda, por outro lado, embora todas as fórmulas envolvidas continuem a ser verdade no novo contexto, já não deve decorrer do mesmo modo.

Efectivamente, assumindo que o processo de raciocínio era idêntico ao anteriormente descrito, para responder à questão `BELIEVE?(Vidro(A))`, o sistema encontra a regra 3, tentando, depois, provar o seu antecedente, `Cor(A,Azul)`. Esse facto não é conhecido à priori, pelo que o transformador da fórmula 4 vai ser usado para determinar qual a cor do objecto A por inspecção directa do mundo real. Mas, no mundo real, a cor do objecto continua a ser Azul, pois apenas na simulação ela passou a ser Vermelha! Donde, continua a ser possível provar a proposição `Cor(A,Azul)` que, claramente, não é o desejado, dado que na simulação em curso a sua cor já não é essa.

É possível constatar que apenas se deve poder interagir directamente com o mundo real em situações em que não se esteja a efectuar raciocínio diacrónico, sob pena de se obterem conclusões erradas. Importa, pois, distinguir entre o raciocínio feito quando a representação interna do mundo está sincronizada com a situação real do mesmo, e o efectuado quando isto não acontece.

Adicionalmente, ao efectuar a mudança para um instante seguinte, é alterado o contexto actual, que passa a conter as proposições primitivas da nova situação. Enquanto o sistema se mantiver a efectuar raciocínio diacrónico, esse contexto vai evoluindo de acordo com as mutações aplicadas. No entanto, assim que esse tipo de raciocínio termina (com o fim da execução de um algoritmo de planeamento, por exemplo), o contexto actual a considerar deve ser o contexto original de onde se partiu no início do raciocínio diacrónico. No exemplo apresentado, após terminar a simulação da execução do operador `Pintar` e se terem tirado as conclusões que com ela se pretendiam obter, o contexto actual deve voltar a reflectir as condições existentes no mundo. A aplicação do raciocínio diacrónico é, assim, encarável como uma "incursão no futuro", regressando-se ao momento presente após ter terminado.

Em suma, é necessário não só alterar os métodos relevantes para reflectir a diferença entre os dois tipos de raciocínio (em sincronismo ou não com o mundo real), como também providenciar mecanismos que permitam a manutenção do contexto original e retornar a ele quando necessário.

3.4.4 Três Formas de Efectuar a Inferência

Existem, à priori, três formas de evitar usar os transformadores `DoIf` e `WhenDo` na inferência quando em raciocínio diacrónico. A primeira consiste, simplesmente, em, quando se aplica uma mutação que despoleta esse tipo de raciocínio, remover do contexto esses transformadores. Dessa forma, quando o algoritmo tentar efectuar inferências como as que se desejam evitar, não o conseguirá pois não encontrará nenhum transformador que consiga aplicar. No exemplo anterior, isto corresponde a considerar como contexto decorrente da

aplicação da mutação o contexto $\{1, 2, 3, 5, 6\}$.

A segunda abordagem ao problema consiste em alterar os métodos `TRANSFORM!` e `TRANSFORM?` para os transformadores `WhenDo` e `DoIf`, respectivamente, de forma a que apenas desempenhem o seu funcionamento normal se a base de conhecimento estiver sincronizada com o mundo exterior, abstendo-se de o fazer caso contrário.

A terceira e última hipótese reside em se alterar os métodos `BELIEVE?` e `BELIEVE!` da classe `Proposition Term`, de modo a que não procurem transformadores dos tipos em causa se o sistema não estiver sincronizado.

Algumas considerações podem ser colocadas. A primeira solução, a ser possível, introduziria uma ineficiência adicional Obrigaria a isolar, na altura da aplicação da mutação, todos os transformadores dos tipos em causa, para os retirar do contexto obtido para a nova situação. No entanto, mesmo que fosse retirados todos os transformadores existentes no mundo, seria, ainda, necessário um mecanismo para evitar a derivação de novos transformadores. Tem ainda a desvantagem de restringir desnecessariamente o acesso à informação representada no sistema,

A segunda abordagem limita a aplicação dos transformadores em qualquer situação, desde que se verifique o dessincronismo. Mas, uma qualquer aplicação implementada sobre o OK-BDI extendido pode desejar conhecer propriedades sobre o mundo num instante do futuro, desde que tenha plena consciência do facto. O que se pretende é evitar a utilização automática e errónea dos transformadores, e não qualquer uso que deles se deseje fazer. Por exemplo, dependendo do domínio, pode existir conhecimento à priori de que certas características dos objectos no mundo são imutáveis. Isto permitiria a escrita de regras que podem fazer inspecção directa do mundo a qualquer instante para conhecer essas características.

Assim, a solução escolhida foi a terceira, em que apenas é restringido o uso dos transformadores nas situações em que esse uso se revelaria verdadeiramente prejudicial à qualidade das conclusões obtidas.

Para estabelecer a distinção entre o sistema estar ou não a funcionar de modo síncrono, é usada a *descrição actual*. Com efeito, essa descrição possui dois campos. No campo `CONTEXT` estará armazenado, a cada momento, o contexto actual do sistema, quando em raciocínio sincrónico. De cada vez que o sistema agir sobre o mundo, conduzindo a uma nova situação *que ainda está sincronizada com o mesmo*, o contexto actual será actualizado, bem como o valor do campo correspondente na descrição actual. Nesses casos, o campo `CURRENT-SITUATION` dessa descrição conterá uma instância de `SITUATION` que indica que tem como `INITIAL-CONTEXT` o contexto actual e em `MUTATIONS` uma sequência vazia. Quando se aplicar uma mutação, o contexto actual bem como o campo correspondente na *descrição actual* é actualizado conforme os resultados da mutação, e no campo `SITUATION` começam a ser registadas as várias mutações aplicadas.

Assim, é possível saber se o sistema está ou não sincronizado com o mundo verificando se a sequência de mutações na `SITUATION` da descrição actual é ou não vazia. Se o for, não foi aplicada nenhuma mutação para se chegar à descrição actual e o sistema está sincronizado. Caso contrário, foram aplicadas mutações e não é permitido usar os transformadores `DoIf` e `WhenDo` nas condições supra-citadas.

3.4.5 Recuperação do Contexto Inicial

Para efectuar a recuperação do contexto inicial (sincronizado com o mundo) após a aplicação de mutações, devem ser respondidas duas questões: *onde* está armazenada a informação sobre o contexto inicial e *quando* reverter a esse contexto.

A primeira questão é de fácil resposta. O contexto inicial está armazenado no campo `INITIAL-CONTEXT` da instância da classe `SITUATION` contida na descrição actual. Reverter para esse contexto consiste, apenas, em voltar à descrição por ele caracterizada, e em que não foram aplicadas quaisquer mutações, ou seja, cujo conteúdo do campo `SITUATION` é a sequência vazia. Essa descrição será a instância de `Description` que era a descrição actual antes do início do raciocínio diacrónico. Consegue-se, assim, que o estado interno do sistema volte a ser o mesmo que estava presente antes da aplicação de qualquer mutação. Este processo define um novo método, `SYNCHRONIZE`, encarregado do funcionamento acima descrito.

A segunda questão não é de resposta tão directa. Quando é executada alguma acção no mundo, os efeitos da execução dessa acção pertencem ao contexto inicial, sincronizado com o mundo, e não ao contexto actual que, como já se viu, se refere a uma hipotética situação futura. Assim, não parecem existir quaisquer dúvidas no que se refere a reverter à situação inicial (ou seja, terminar o raciocínio diacrónico) quando se deseja executar uma acção. Para conseguir este comportamento, devem ser alterados o método `INTEND` da classe `Act Term`. Esse é o método invocado de cada vez que uma acção física vai se executada. Nele deve-se verificar, antes que mais, se o sistema está ou não em raciocínio diacrónico. Se o estiver, volta-se a sincronizá-lo invocando `SYNCHRONIZE` antes de executar a acção em causa.

A execução de acções não é a única situação em que pode interessar reverter à descrição inicial. Uma aplicação pode utilizar o raciocínio diacrónico para um qualquer fim e esse processo pode não terminar com a execução de uma acção. Deve, pois, existir uma forma de, explicitamente, reverter à situação original. Assim, permite-se a invocação directa de `SINCHRONIZE` por parte do utilizador do sistema.

Capítulo 4

O Planeador Integrado

No capítulo anterior, foi efectuada a integração da Lógica das Mutações no formalismo OK-BDI. Isso fornece a esse formalismo a capacidade de efectuar raciocínio diacrónico. Esta nova capacidade irá permitir a utilização do OK-BDI para um leque mais alargado de aplicações, entre as quais se encontra-se o *planeamento de acções*.

Para ilustrar as potencialidades do sistema são descritos em seguida dois planeadores simples implementados sobre o formalismo OK-BDI extendido com a Lógica das Mutações, planeadores esses que poderão ser usados livremente por outras partes do próprio formalismo que deles necessitem.

4.1 Introdução

O planeamento de acções e o raciocínio sobre mudança são duas áreas da Inteligência Artificial fortemente relacionadas entre si. O planeamento de acções centra-se no estudo dos processos que permitem estabelecer uma sequência de acções que conduzem de um estado do mundo a outro em que se verificam certas condições. O raciocínio sobre mudança preocupa-se mais com o estudo das mudanças propriamente ditas, a nível da sua natureza e propriedades formais. A falta de semântica formal para o OK-BDI não o torna adequado para o raciocínio sobre mudança. No entanto, está numa posição privilegiada para o planeamento de acções.

Um planeador não é mais do que um algoritmo que deve estabelecer uma sequência de acções a executar no mundo que o levem de um qualquer estado inicial a um estado final que satisfaz certas condições pré-determinadas. O conjunto de proposições que devem ser verificadas no estado final será designado doravante por *conjunto objectivo* e representado por Ω. Isto assume que existe uma representação do mundo e das várias acções que nele podem ser desempenhadas, na posse da qual o algoritmo consegue desenvolver o plano. É, pois, uma aplicação natural para aplicar um formalismo que permita o raciocínio

diacrónico, tal como o apresentado neste trabalho.

O OK-BDI apresenta grandes potencialidades como base para um planeador dado que, ao englobar um sistema capaz de agir e inspeccionar o mundo real, permite a criação e verificação dos planos de forma integrada. A integração de um planeador no OK-BDI vem não só aumentar ainda mais as suas capacidades como também suprir uma lacuna nesse sistema, dado que não está previsto, à partida, nenhum processo de planeamento para o mesmo.

4.2 As Estratégias de Controlo

A existência de um formalismo capaz de raciocínio diacrónico permite, como já foi referido, a implementação de um planeador. O formalismo indica como transitar de uma situação para outra, sendo o planeador o algoritmo que indica por que ordem essas transições serão efectuadas até se atinja uma situação em que é possível satisfazer todas as condições do conjunto objectivo Ω. Os elementos de Ω devem ser fórmulas literais, dado que as mutações só permitem que fórmulas literais sejam especificadas como seus efeitos.

A implementação de um planeador passa, em grande parte, pela escolha de uma estratégia de controlo adequada. Essa estratégia indica de que forma vão ser efectuadas as diferentes mudanças possíveis no mundo em busca de uma sequência coloque o mundo numa situação que satisfaça as condições a atingir. Desta estratégia dependerão, em grande parte, a qualidade dos planos obtidos e a eficiência do processo de planeamento propriamente dito.

A maioria dos sistemas mais antigos, como o `STRIPS` [Fikes e Nilsson 1971], o `HACKER` [Sussman 1975] ou o `NOAH` [Sacerdoti 1977] focam a sua atenção em assuntos como a representação da mudança (normalmente sob a forma de operadores e a sua aplicação), a representação do estado do mundo e a estrutura dos planos criados. Mais recentemente, tem-se vindo a registar uma preocupação crescente com a estratégia de controlo subjacente ao processo de planeamento. Um exemplo bem claro disso é o `UCPOP` [Penberty e Weld 1992], em que a estratégia de controlo é parametrizável pelo utilizador de acordo com o domínio em causa.

No caso concreto do presente trabalho, o processo de planeamento estará fortemente ligado ao raciocínio, dado que, ao contrário de outros formalismos, a representação da mudança se encontra ao mesmo nível que a representação dos diversos factos sobre o mundo, e não a um meta-nível, em que a mudança é representada por um conjunto de operadores que não partilham da mesma linguagem que a descrição do mundo. Na lógica das mutações (e, consequentemente, neste trabalho), o planeamento pode ser encarado de duas formas diferentes, *orientado às proposições* e *orientado às descrições*. Cada uma dessas formas de encarar o planeamento, descritas em seguida, está na base de diferentes

estratégias de controlo.

4.2.1 Planeamento Orientado às Proposições

Segundo esta abordagem, as mutações são encaradas como mais um tipo de regras de inferência que não merecem tratamento especial em relação às restantes. As conclusões obtidas pela aplicação de uma regra de mutação transportam a base de conhecimento do sistema para um instante posterior ao original, mas, de resto, a sua aplicação faz-se sem quaisquer preocupações adicionais.

O acto de planear pode ser encarado como uma tentativa de efectuar uma prova das proposições no conjunto objectivo aplicando mutações se isso se mostrar necessário para as atingir. Não se estabelece, assim, diferença entre o raciocínio sincrónico e o raciocínio diacrónico.

Se for possível efectuar uma prova desse tipo, então é necessário verificar quais as mutações envolvidas e ordená-las convenientemente. A sequência obtida será o plano desejado. Se não for possível efectuar a prova, isso significa que não existe nenhuma sequência de transições capaz de conduzir à descrição desejada, pelo que não é possível encontrar um plano.

4.2.2 Planeamento Orientado às Descrições

No planeamento orientado às descrições, as mutações têm um tratamento de excepção em relação às restantes regras de inferência. São vistas como operadores que permitem a transição entre instantes de tempo diferentes. Esses instantes de tempo, na Lógica das Mutações, são representados pelas descrições, daí o nome desta abordagem ao planeamento.

Existe uma diferença bem definida entre o raciocínio sincrónico e o raciocínio diacrónico. O planeamento é efectuado em duas vertentes distintas. A primeira, a sincrónica, consiste em tentar provar as proposições que representam as condições do conjunto objectivo numa dada descrição sem usar operadores de mutação. Se isso for possível, então a sequência de mutações que deu origem, a partir da descrição inicial, a essa descrição, é o plano desejado. Caso contrário, é efectuado raciocínio diacrónico, mediante a aplicação de uma mutação, que conduz a uma nova situação em que se espera que as condições a verificar sejam deriváveis.

4.2.3 Abordagem considerada

Neste trabalho, a abordagem considerada foi a do *planeamento orientado às descrições*. As principais considerações que se colocaram ao efectuar essa escolha foram preocupações ao

nível da eficiência. A utilização de uma estratégia de planeamento orientada às proposições no OK-BDI implica que também as mutações devem poder ser usadas livremente durante o processo de inferência para verificar a presença de uma determinada proposição. Seria assim necessário alterar os métodos `BELIEVE!` e `BELIEVE?` para todos os tipos de proposições, de modo a que, para além de efectuarem inferência recorrendo aos transformadores `AntCq`, `DoWhen` e `DoIf`, tentassem, também, determinar a validade ou não de uma dada proposição recorrendo a transformadores `MutationPcEf`. Isto tornaria possível a utilização das mutações como se de quaisquer outras regras de inferência se tratassem, permitindo eventualmente atingir planos para resolver determinados problemas.

Esta não é, porém, uma abordagem adequada. Efectivamente, o OK-BDI não é um sistema dedicado exclusivamente ao planeamento, mas a um leque mais vasto de aplicações para as quais, à priori, em nada interessa o raciocínio diacrónico e a construção de planos. Para além disso, todo um leque de problemas relacionados com o sincrinísmo com o mundo real se levanta. Donde, a integração do planeamento no OK-BDI da forma apresentada não é a mais adequada visto que torna o formalismo pouco adequado para outras aplicações.

Adicionalmente, a abordagem orientada às proposições pode, à partida, dar origem a uma ramificação bastante grande na inferência, o que dificultaria o processo de planeamento, tornando-o bastante ineficiente. Considerar como separados o raciocínio sincrónico e diacrónico diminui grandemente as dimensões do espaço de estados a considerar. Os processos de raciocínio são de natureza exponencial com o número de proposições envolvidas. Logo, não se considerando em simultâneo a informação referente a todos os instantes possíveis do mundo (no planeamento orientado às proposições), mas sim apenas o referente a um dado instante de cada vez, consegue-se, à partida, tornar o processo de planeamento mais eficiente.

4.2.4 Duas Direcções de Planeamento

Uma vez escolhida a estratégia de global do planeamento, o *planeamento orientado às descrições*, é necessário definir como vai ser efectuado o processo de planeamento propriamente dito.

As várias mutações existentes no sistema definem um grafo cujos nós são as descrições que é possível atingir por aplicação sucessiva de mutações. Isto encontra-se exemplificado na figura 4.1. Nessa figura é possível constatar como duas mutações são aplicáveis em D_0, conduzindo, respectivamente a D_{1a} e D_{1b}. Em D_{1b} duas outras mutações são aplicáveis, gerando D_{2a} e D_{2b}. Em D_{1a}, a aplicação da mutação μ_{2a} conduz, tal como a aplicação de μ_{2b} em D_{1b}, a D_{2a}. O grafo assim criado é designado por *espaço de descrições*. O planeamento pode ser visto como uma procura neste espaço de um caminho entre a situação inicial e uma que satisfaça os constrangimentos do problema. O plano final consistirá nas mutações que definem o caminho entre essas duas situações.

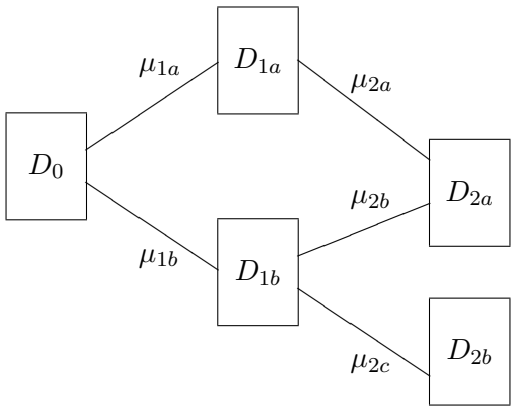


Figura 4.1: Exemplo de um grafo de situações

Existem várias formas diferentes de percorrer o grafo apresentado em busca de um plano. Para o presente sistema, duas estratégias principais se destacam: uma estratégia *progressiva* e uma estratégia *regressiva* (designadas por progressiva/regressiva e regressiva/regressiva em [Pinto-Ferreira 1991]).

Estratégia de Controlo Progressiva

Esta é a estratégia de controlo que mais se assemelha às técnicas de procura clássicas, como o A^*. A partir da descrição inicial, o planeador tenta encontrar uma sequência de mutações cuja aplicação conduza a uma descrição em que se verificam as proposições na especificação do objectivo.

Esta estratégia é iniciada tentando verificar sincronicamente (sem recorrer a mutações), por encadeamento regressivo, se as proposições que definem as condições do estado final se verificam na descrição inicial. Em caso afirmativo, o problema tem uma solução trivial. Caso contrário, é necessário obter os operadores de mutação aplicáveis nessa descrição. Para tal, é necessário verificar se as suas pré-condições são deriváveis na descrição. Na posse desses operadores de mutação, escolhe-se, de entre eles, o mais promissor, segundo algum critério, tendo em vista o objectivo a atingir. A aplicação dessa mutação conduz a uma nova descrição, em que o processo se repete. É feito um retrocesso na procura sempre que necessário para tentar novos operadores de mutação. Quando se encontrar uma descrição em que é possível derivar as proposições que especificam as condições do objectivo, o plano é a sequência de mutações que levou a essa descrição. Se não for possível encontrar uma descrição nessas condições, o processo de planeamento falha.

Estratégia de Controlo Regressiva

A estratégia de controlo regressiva parte, como o seu nome indica, das condições que definem o estado final. O seu modo de funcionamento consiste em tentar satisfazer todas essas condições na situação inicial. Se todas o forem, o problema tem uma solução trivial. Caso contrário, é necessário satisfazer as que não o foram directamente. Assim, verificam-se quais as regras de inferência (de entre as quais se destacam as mutações) cujos efeitos o façam. Tentam-se então aplicar essas regras. Como, de um modo geral, nem todas as pré-condições para essa aplicação são satisfeitas torna-se necessário satisfazê-las antes da sua aplicação. Essas pré-condições tomam agora o papel antes ocupado pelas condições do objectivo num processo recursivo.

Esta estratégia é reminiscente da *Means-Ends-Analysis* encontrada em sistemas clássicos de planeamento como o `GPS` [Newell e Simon 1961, Ernst e Newell 1969], o `STRIPS` [Fikes e Nilsson 1971]. Mais recentemente, ainda é possível encontrá-la como subjacente a sistemas como o `Prodigy` [Veloso et al. 1995] ou o `UCPOP` [Penberty e Weld 1992].

Cada uma destas duas estratégias de controlo pode ser usada nas mesmas situações. Diferem pelo facto de a sua ramificação depender, primariamente, do número de mutações aplicáveis e do número de condições a atingir no objectivo, para as estratégias progressiva e regressiva, respectivamente. Assim, a sua utilização deve ser escolhida criteriosamente de acordo com o problema em causa.

Em seguida serão descritos dois planeadores, cada um dos quais referentes a uma das estratégias de controlo acima.

4.3 O Planeador Progressivo

Este planeador incorpora uma estratégia de controlo progressiva. Parte da descrição inicial e, através da aplicação de mutações, tenta chegar a uma situação em que as condições a atingir são satisfeitas.

O planeador recebe um conjunto objectivo, Ω, de proposições que devem ser satisfeitas na descrição a atingir. O algoritmo básico é o seguinte:

1. Verificar se, na descrição inicial, D_0, o conjunto objectivo Ω é satisfeito, usando inferência regressiva (métodos `BELIEVE?`). Se for, o problema tem uma solução trivial. Caso contrário, efectuam-se os passos seguintes.

2. Obter os transformadores `MutationPcEf` aplicáveis em D_0, conduzindo a novas descrições, eventualmente mais próximas da descrição onde Ω será satisfeito.

3. Ordenar as mutações encontradas segundo um critério que indique quais as mais promissoras em relação ao objectivo a atingir.

4. Seleccionar a mutação que se revelou mais promissora no passo 3 e aplicar essa mutação, conduzindo a uma nova descrição D_1, representada por uma nova instância de `Description` que passa a ser a descrição actual. Nessa descrição, para além de se actualizar o contexto, é também alterado o campo `CURRENT-SITUATION`, que passa a conter uma instância de `SITUATION` em que é indicada, no campo `MUTATIONS`, a aplicação do transformador.

5. Verificar, usando inferência regressiva, se Ω é satisfeito em D_1. Em caso afirmativo, a sequência de mutações contida no campo `MUTATIONS` da descrição actual é o plano a atingir. Devolve-se esse plano e, dado que terminou o raciocínio diacrónico, executa-se o método `SYNCHRONIZE` para voltar a sincronizar o sistema com a situação existente no mundo.

6. Caso Ω não seja satisfeito em D_1, são encontradas as mutações aplicáveis nessa descrição, classificam-se essas mutações de acordo com o critério anteriormente usado e juntam-se ordenadamente à sequência de mutações por aplicar. Volta-se, então, ao passo 4 até que se atinja uma solução.

7. Se todas as mutações possíveis tiverem sido aplicadas sem que se tenha chegado a uma descrição em que Ω se verifique, o processo de planeamento falha.

Este algoritmo é bastante simples e, numa primeira aproximação, parece resolver o problema em questão. Faltaria apenas definir o critério de valoração das mutações a aplicar. No entanto, existem alguns factores que não tem em conta e que são determinantes para o funcionamento correcto de um planeador para o sistema em causa. Adicionalmente, a elevada expressividade do OK-BDI no que diz respeito ao agrupamento de acções usando acções de controlo sugere que os planos gerados devem, se possível, tomar algum partido essa expressividade

4.3.1 O Problema das Descrições Alternativas

A principal razão pela qual o algoritmo acima descrito falha no caso geral encontra-se no passo 5. Com efeito, é referido que, ao aplicar uma mutação em D_0, se transporta o sistema para uma nova descrição D_1, em que é necessário verificar se são atingidas as condições em Ω. No entanto, não há garantias de que a aplicação da mutação resulte em apenas uma descrição.

A aplicação das mutações obriga a calcular a mudança mínima da descrição de partida de forma a que deixe de ser possível, com a nova descrição, acreditar na negação dos efeitos da mutação e das pré-condições que não são simultaneamente efeitos. Como já foi visto

no capítulo anterior, pode existir mais do que uma mudança mínima que cumpre os pré-requisitos estabelecidos. Donde, a aplicação de uma mutação pode, por vezes, conduzir a várias descrições alternativas. Para além de não ser gerada uma única descrição, não existe nada, à priori, que permita escolher de entre as descrições geradas qual a correcta.

Considere-se, por exemplo, uma descrição em que $fbfs$ suportadas primitivas são as seguintes:

A	$\langle 1,\{\{1\}\}\rangle$
B	$\langle 2,\{\{2\}\}\rangle$
$\forall x\ ((A \wedge B) \rightarrow C)$	$\langle 3,\{\{3\}\}\rangle$
$\mu(\{C\};\{D\};\mu_1)$	$\langle 4,\{\{4\}\}\rangle$
$\mu(\{A\};\{E\};\mu_2)$	$\langle 5,\{\{5\}\}\rangle$
$\mu(\{B\};\{E\};\mu_3)$	$\langle 6,\{\{6\}\}\rangle$

Na descrição acima é ainda possível inferir a seguinte fórmula a partir de 1, 2 e 3:

$$C \qquad \langle 7,\{\{1,2,3\}\}\rangle$$

Para o demonstrar o problema das descrições alternativas, consideremos que se deseja encontrar um plano que, a partir da descrição apresentada, consegue chegar a uma descrição em que se verificam simultaneamente D e E. Assim, temos que a descrição inicial D_0 é a apresentada acima, e o conjunto objectivo, Ω é igual a $\{D, E\}$.

Existem, evidentemente, vários planos possíveis para atingir uma descrição que satisfaça Ω. Imaginemos que o planeador vai explorar em primeiro lugar a aplicação da mutação μ_1. A aplicação dessa mutação dá origem a três descrições alternativas. As $fbfs$ suportadas acreditadas em cada uma delas são:

$$D_{1a} = \{1, 2, 4, 5, 6, 8\}$$
$$D_{1b} = \{2, 3, 4, 5, 6, 8\}$$
$$D_{1c} = \{1, 3, 4, 5, 6, 8\}$$

$$D \qquad \langle 8,\{\{8\}\}\rangle$$

É, agora, de recordar que, embora o formalismo dê origem a três descrições alternativas, quando a acção correspondente à mutação μ_1 for, de facto, executada no mundo real, terá um resultado perfeitamente definido, correspondente, eventualmente, a uma das três descrições. No entanto, o formalismo não tem forma de saber qual das descrições irá corresponder à verdade. Aliás, pode ser impossível sabe-lo antes de executar a acção, devido à natureza da mesma.

Um dos objectivos de Ω, D, foi atingido. Falta, no entanto, atingir E. Para tal, outra mutação deve ser aplicada. O que se passa agora é que, de acordo com a descrição considerada como resultante de μ_1, *uma mutação diferente pode ser aplicada*. Se, por exemplo, se assumir que se obteve a descrição D_{1b}, pode ser aplicada μ_3. Se, por outro lado, se assumir a descrição D_{1c}, apenas pode ser aplicada μ_2. Em D_{1a} ambas as mutações poderiam ser aplicadas.

A melhor forma de resolver este problema reside na utilização do elevado poder expressivo do OK-BDI, com a construção de *planos condicionais*. A maioria dos planeadores tradicionais assume um conhecimento perfeito sobre as acções e os seus efeitos no mundo. Não existe latitude para lidar com incertezas no mesmo. No entanto, nem sempre isso adequado, como no caso presente. A construção de planos condicionais é uma área de estudo relativamente recente e (também chamados por vezes de *planos contingentes*) vem minimizar os problemas que podem surgir devido a incertezas no resultado das acções. Um exemplo de um sistema que cria planos capazes de lidar com contingências é o `Cassandra`, desenvolvido por Pryor e Collins tendo como base o `UCPOP` [Pryor e Collins 1996].

4.3.2 Os Planos Condicionais

A necessidade de introduzir planos condicionais surge em virtude do problema das descrições alternativas, em que não é possível decidir qual das descrições obtidas por aplicação de uma mutação deve ser considerada. A solução consiste em considerar *todas* as descrições alternativas, construindo planos condicionais que estabeleçam uma alternativa para cada descrição possível. A escolha entre as diversas alternativas será efectuada apenas na altura da execução do plano, em que se saberá qual o resultado efectivo da aplicação da mutação que no processo de planeamento deu origem a descrições alternativas.

As múltiplas alternativas podem ser expressas, no OK-BDI, mediante acções de controlo `IF`. Essas acções de controlo permite especificar um conjunto de `GUARDED-ACTS`, sendo cada um deles constituído por uma condição e uma acção. A acção a executar será aquela cuja condição correspondente for verdadeira [Kumar 1993].

Para a criação dos planos condicionais é necessário, antes que mais, reconhecer que existem várias $fbfs$ primitivas comuns às várias descrições alternativas. Considerem-se, assim, os seguintes conceitos:

Θ: conjunto de $fbfs$ primitivas comuns às várias descrições alternativas.

Δ: conjunto em que cada um dos seus elementos é um conjunto com as $fbfs$ primitivas que são acreditadas em cada descrição, para além das contidas em Θ.

$\Lambda(D, \mu)$: uma função que devolve um par $[\![\Theta, \Delta]\!]$ representando as várias descrições possíveis que se podem obter a partir de D por aplicação de μ.

As fórmulas primitivas das várias descrições são obtidas por união de Θ com cada um dos elementos de Δ. Para o domínio acima descrito, a aplicação de μ_1, conduz a três descrições alternativas. Assim, $\Lambda(D_0, \mu_1) = [\![\{4,5,6,8\}, \{\{1,2\},\{1,3\},\{2,3\}\}]\!]$.

Na posse da informação fornecida pela função Λ, é possível elaborar vários planos alternativos, mediante chamadas recursivas ao planeador para cada uma das descrições. Os testes que permitem efectuar a escolha entre esses vários planos são a verificação da satisfação das proposições em cada um dos elementos de Δ no mundo.

Em suma, após a aplicação de cada mutação o planeamento faz-se não num mas em vários ramos paralelos, correspondentes às várias descrições obtidas. Em cada ramo para o qual seja possível encontrar um plano que conduza à descrição em que são satisfeitas as condições em Ω, é criada uma nova condição, a conjunção de todos as fórmulas contidas no elemento de Δ correspondente a esse ramo, que será colocada dentro de uma acção de controlo `IF` no local oportuno do plano.

Pode acontecer que para alguns dos ramos não seja possível encontrar um plano. Nesse caso, a condição referente a esse ramo não aparece no plano final. Se nenhuma dos restantes ramos puder ser seguido na altura da execução do plano, este falha.

Se não for possível encontrar um plano para nenhum dos ramos, então considera-se que a aplicação da mutação que deu origem às várias descrições não pode conduzir a um plano e deve ser efectuado um retrocesso, testando-se outras possibilidades.

No caso do exemplo acima, o plano gerado poderia ser o seguinte (a mutação a aplicar quando se verifica $A \wedge B$ poderia ser μ_3, dado que tanto essa mutação como μ_2 são aplicáveis):

$$\texttt{SEQUENCE(}\mu_1\texttt{,IF(}[(A \wedge B) \rightsquigarrow \mu_2][(A \wedge C) \rightsquigarrow \mu_3][(B \wedge C) \rightsquigarrow \mu_3]\texttt{))}$$

4.3.3 Escolha da Próxima mutação

Outro dos problemas encontrados no processo de planeamento é a escolha de qual a próxima mutação a aplicar, ou seja, de qual o ramo do espaço de descrições a considerar em seguida. Nem todos os ramos são igualmente promissores na pesquisa de um plano. Alguns podem mesmo conduzir a becos sem saída, redundando a sua exploração num desperdício de recursos computacionais. Adicionalmente, como apenas se procura um plano, o primeiro a ser encontrado termina o processo de planeamento. Se se conseguir descobrir um plano pelo ramo de menor custo computacional, o processo torna-se muito mais eficiente. Assim, importa escolher correctamente entre as várias possibilidades.

Considere-se o exemplo representado graficamente na figura 4.2. Há duas mutações aplicáveis em D_0. Uma delas, μ_1 resulta uma única descrição, e a outra, μ_2 em duas

descrições alternativas.

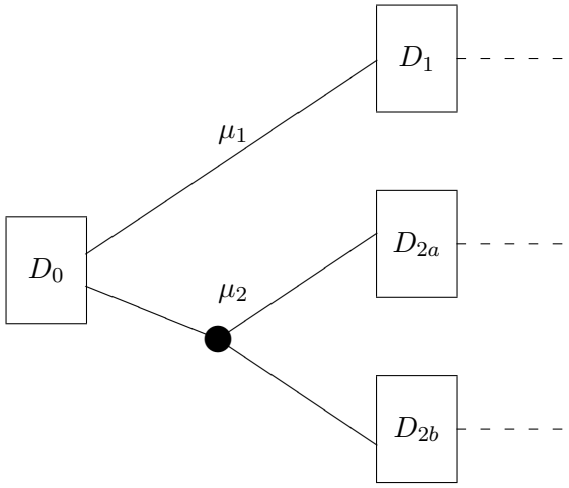


Figura 4.2: Dois caminhos a seguir

O planeador tem, pois necessidade de escolher qual o ramo que irá explorar em primeiro lugar: o resultante da aplicação de μ_1 ou o resultante da aplicação de μ_2. Para tal, dois factores são determinantes: a *redução da distância em relação a* Ω e a *redução da ramificação do planeamento*.

Redução da Distância em Relação ao Objectivo

O processo de planeamento procura, em última análise, chegar a uma descrição em que todas as condições em Ω são satisfeitas. Assim, interessa aplicar mutações que tornem satisfeitas o maior número dessas condições, conduzindo, assim, à descrição pretendida. Por outras palavras, as mutações relevantes são aquelas que diminuem a distância entre a descrição actual e a descrição final.

Na escolha do ramo a seguir, o planeador deve considerar, primeiro, as mutações que, a partir da descrição em que forem aplicadas, mais reduzem essa distância. Para determinar a distância para cada caso é necessária informação não só das mutações a aplicar, como também das descrições em que vão ser aplicadas. Para tal, deve calcular-se a distância entre Ω e essas descrições e em seguida analisar-se os efeitos da mutação, verificando se algum desses efeitos altera essa distância.

O calculo do mérito de uma mutação levanta, no entanto, um problema adicional quando na presença de descrições alternativas. Nesse caso, pode não ser evidente qual a distância entre as descrições obtidas e o conjunto objectivo. É possível saber os efeitos directos de uma uma mutação (a adição dos seus efeitos e a remoção da negação dos mesmos e das pré-condições que não são efeitos), mas os efeitos indirectos, resultantes do processo de determinação dos conjuntos mínimos que está na génese da obtenção das

descrições alternativas, não são fáceis de conhecer à priori, sem efectuar um raciocínio de complexidade semelhante ao da aplicação da mutação propriamente dita. Isto iria originar uma grande redundância e uma consequente degradação da eficiência do planeamento.

É, pois, mais sensato fazer a escolha da próxima descrição a explorar analisando todas várias descrições ainda inexploradas e não as mutações aplicáveis a cada uma delas. Quando uma dada descrição é escolhida para exploração, aplicam-se todas as mutações possíveis, dando origem a novas descrições. Essas descrições são classificadas quanto à sua distância ao objectivo e consideradas no passo seguinte em conjunto com as restantes descrições ainda não exploradas.

O conceito de distância entre conjuntos de fórmulas (no presente caso, descrições) tem vindo a ser usado ao longo da história do planeamento pelos mais diversos sistemas, como o `STRIPS` [Fikes e Nilsson 1971], Prodigy [Veloso et al. 1995], entre outros. A maneira ideal de calcular a distância entre uma descrição e um conjunto objectivo seria obter o tamanho da menor sequência de mutações que conduz a uma descrição em que as condições objectivo sejam satisfeitas. Isto, no entanto, é um processo difícil de desempenhar, com uma complexidade comparável à do processo de planeamento propriamente dito.

Uma forma mais simples para medir a distância entre uma descrição D e um conjunto de fórmulas Ω consiste em verificar quais as fórmulas em Ω que não são satisfeitas em D. Não é um valor real da distância. Por exemplo, algumas mutações podem satisfazer duas condições em simultâneo. Apesar disso permite guiar o processo de planeamento de forma satisfatória.

Redução da Ramificação do Planeamento

Outro dos problemas relevantes para a eficiência do planeador é a ramificação podo espaço de situações. Esta pode ser ser encarada segundo duas vertentes: a *ramificação inerente ao domínio* e a *ramificação devida às descrições alternativas*.

A ramificação inerente ao domínio traduz-se pelo número de mutações aplicáveis a cada descrição. Se o domínio tiver alguma complexidade, esse número será relativamente grande, colocando muitas alternativas ao dispor do planeador. Esta ramificação, embora seja um factor a considerar, não é a mais preocupante. Efectivamente, o planeador apenas irá, se tudo correr bem, percorrer alguns dos ramos possíveis. De qualquer forma, esta ramificação é inerente ao domínio e apenas pode ser controlada até certo ponto, recorrendo a escolhas mais informadas.

A ramificação devida às descrições alternativas, por outro lado, não depende apenas do domínio mas também do formalismo em que este se encontra representado no OK-BDI. Como é necessário considerar todos os ramos alternativos para dar o processo de planeamento como terminado, pode incorrer-se numa ineficiência maior do que a expectável. Importa,

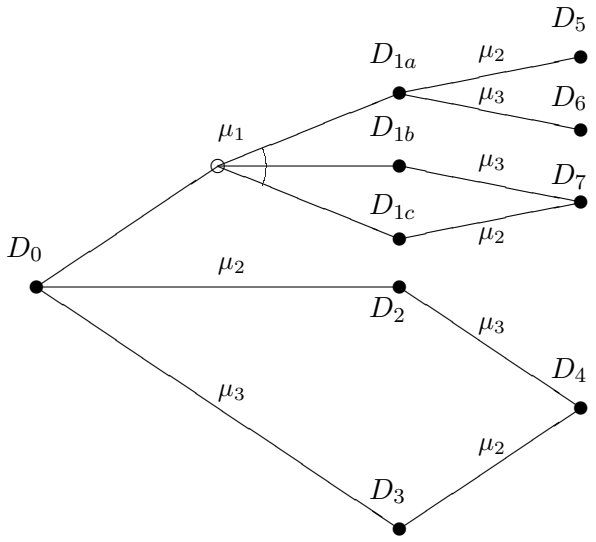


Figura 4.3: Espaço de Situações

pois, tomar medidas para tornar o planeamento mais eficiente.

Uma dessas medidas é uma alteração ao critério de escolha do ramo seguinte do espaço de descrições a explorar. Na secção anterior, apresentou-se um critério baseado na distância entre descrições. Outro critério que pode ser considerado é dar relevância a ramos com menor número de descrições alternativas.

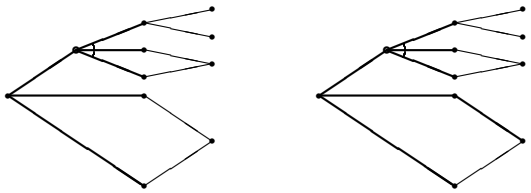
Existem, assim dois critérios diferentes mas complementares a considerar. O primeiro considera os vários ramos alternativos e escolhe o com menor ramificação. O segundo escolhe, no ramo isolado pelo anterior, qual o estado a explorar em seguida, de acordo com o critério da distância.

O exemplo na secção 4.3.1 pode ser esquematicamente representado pelo grafo na figura 4.3. Nessa figura, as descrições envolvidas são caracterizadas pelos seguintes conjuntos de fórmulas primitivas:

D_0	$\{1,2,3,4,5,6\}$	D_3	$\{1,3,4,5,6,9\}$
D_{1a}	$\{1,2,4,5,6,8\}$	D_4	$\{3,4,5,6,9\}$
D_{1b}	$\{2,3,4,5,6,8\}$	D_5	$\{2,4,5,6,8,9\}$
D_{1c}	$\{1,3,4,5,6,8\}$	D_6	$\{1,4,5,6,8,9\}$
D_2	$\{2,3,4,5,6,9\}$	D_7	$\{3,4,5,6,8,9\}$

D	$\langle 8,\{\{8\}\}\rangle$
E	$\langle 9,\{\{9\}\}\rangle$

As descrições solução, que satisfazem todas as condições em Ω, são D_5, D_6 e D_7. Como



(a) (b)

Figura 4.4: Várias alternativas para o planeamento

se vê, a partir da descrição D_0 será, à partida, possível encontrar um plano por três vias diferentes, que correspondem, respectivamente, a aplicar em primeiro lugar μ_1, μ_2 e μ_3 (figura 4.4(a)). O planeador deve, então, escolher o caminho mais promissor. Assim, numa primeira fase, deve escolher a alternativa com menor ramificação. As duas alternativas nessas condições são as resultantes da aplicação de μ_2 e μ_3. De ente elas, o planeador pode, então, escolher a que mais reduza a distância em relação aos objectivos a atingir. Neste caso, tanto μ_2 como μ_3 conduzem a uma descrição à mesma distância do objectivo (1), pelo que se pode assumir que o planeador escolhe qualquer uma delas, por exemplo, μ_2. Como a descrição a que essa mutação conduz, D_4, não é solução, o planeador tem agora três alternativas possíveis: aplicar μ_1 em D_0, aplicar μ_3 em D_0, ou aplicar μ_3 em D_2 (figura 4.4(b)).

Eventualmente, o planeador deverá explorar a alternativa referente à aplicação de μ_1 em D_0, dado que nenhum dos outros ramos conduz à solução. Quando é escolhida essa possibilidade, o planeador utiliza o segundo critério, para verificar quais das descrições consideradas (D_{1a}, D_{1b} e D_{1c}) se encontra mais perto da solução (neste caso é indiferente), escolhendo-se essa descrição para explorar em seguida.

Um ramo conjuntivo é uma solução se pelo menos um dos ramos o for. No entanto, um plano ideal deveria apresentar uma solução para cada uma das alternativas. Assim, enquanto não se explorarem todos os caminhos alternativos para um ramo conjuntivo (pela ordem estabelecida pelo critério da distância), o processo de planeamento prossegue.

Pode dar-se o caso de a derivação de um plano para todas as alternativas possíveis não ser plausível dado o esforço computacional envolvido. Assim, uma optimização possível é a de, assim que for encontrado um plano para pelo menos uma das alternativas de um plano condicional, sempre que for excedido um dado limite de recursos temporais ou computacionais, desistir de tentar encontrar planos para as outras alternativas.

Nesse caso, o planeador deve apresentar para essas alternativas um plano fictício consistindo numa indicação de que deve ser efectuado planeamento adicional para esse ramo. Isto pode ser conseguido recorrendo a acções de controlo do OK-BDI. A acção de controlo

relevante para esta situação é `ACHIEVE`. Esta acção tem como argumento uma proposição. Quando executada, desencadeia o processo necessário para tornar verdade essa proposição. Assim, é introduzida uma instância de `ACHIEVE` aplicada a uma conjunção dos elementos de Ω ainda não satisfeitos. Isto implica algumas alterações na definição do método `INTEND` para esta classe, que serão descritas na secção 4.7.

Está-se assim, na prática, a relegar para o futuro o planeamento necessário para esse ramo. Eventualmente, dependendo de como a execução do plano vier a decorrer, se for escolhido o ramo ainda não planeado o planeador será se necessário invocado pela acção de controlo `ACHIEVE`. Pode mesmo dar-se o caso de o plano gerado originalmente bastar para as necessidades encontradas, não sendo necessário efectuar o planeamento para os ramos não considerados originalmente.

4.3.4 O Planeamento como Procura

Considerando o que foi dito nas secções anteriores sobre as possíveis estratégias de controlo para o processo de planeamento, decorre de forma directa que o mesmo, na realidade não é mais do que a aplicação de um algoritmo de procura ao grafo de descrições.

Um *algoritmo de procura* não é mais do que um algoritmo que, num dado espaço de estados (topologicamente um grafo em que os nós são os estados e os arcos os diferentes operadores que permitem as transições entre esses estados), a partir de um estado inicial, tenta encontrar uma sequência de operadores que conduza a um estado final, caracterizado por satisfazer um conjunto de condições pré-definidas. Os algoritmos de procura são, normalmente, classificados como *cegos* ou *informados*. Os primeiros efectuam uma travessia sistemática do espaço de estados em procura da solução. Os segundos utilizam alguma informação do domínio em causa para guiar essa procura pelos caminhos mais promissores em direcção à solução. Essa informação está presente sob a forma de uma função aplicável a cada estado, designada por *heurística*, que dá uma avaliação informal da qualidade de cada um.

A base de grande parte dos algoritmos de procura é o algoritmo designado por *Melhor-Primeiro** (*Best-First**, ou `BF*` na literatura Anglo-Saxónica). Esse algoritmo baseia-se em duas lista de nós: a lista de nós `ABERTOS`, que contém os estados ainda não expandidos (cujos sucessores ainda não foram gerados) ordenados de acordo com a função heurística (a princípio apenas o estado inicial), e a lista de nós `FECHADOS`, que já foram anteriormente expandidos. A cada momento, o algoritmo selecciona o estado que é o primeiro elemento de `ABERTOS` (o que a heurística indica estar mais próximo da solução), retira-o dessa lista e coloca-o em `FECHADOS`. Verifica então se esse estado é ou não a solução pretendida. Caso o seja, o algoritmo termina com sucesso. Caso contrário são aplicados todos os operadores válidos a esse estado, num processo designado por *geração dos seus sucessores*. Esse processo dá origem a um conjunto de novos estados que, se ainda não estiverem

presentes em `FECHADOS`, ou se estando presentes, têm um valor da heurística melhor que o do estado contido nessa lista, são colocados ordenadamente em `ABERTOS` de acordo com o valor da heurística. O verificação da presença em `FECHADOS` serve para evitar a criação de ciclos no processo de procura. O algoritmo prossegue até se encontrar uma solução ou a lista de abertos ficar vazia, caso em que falha.

Existem variantes deste algoritmo como, por exemplo, o `A*` [Hart et al. 1968] e as mais diversas estratégias híbridas. Para mais informação sobre estes algoritmos pode ser obtida consultando [Pearl 1985].

A semelhança do algoritmo descrito com a estratégia de controlo progressiva para o planeamento que tem vindo a ser descrita é enorme. No caso do planeador, o espaço de estados é designado por espaço de descrições, dado serem as descrições os estados do problema. Os vários operadores surgem sobre a forma de mutações. Tem-se, também um estado inicial (a descrição inicial) e as condições que definem o estado final (o conjunto Ω). Finalmente, existe, até, uma função heurística que determina a qualidade de um estado: a distância de uma descrição à descrição objectivo. Efectivamente, não obstante algumas diferenças notacionais, o planeamento pode-se reduzir, muitas vezes, à procura. Vários estudos têm sido desenvolvidos sobre esta área [Minton et al. 1994, Chapman 1987, Barrett e Weld 1994].

A constatação da equivalência entre planeamento e procura permite simplificar o desenvolvimento do planeador pois torna possível aproveitar os resultados já existentes para a procura, uma área em que existe um grande número de estudos teóricos, ao contrário do planeamento que, muitas vezes, se centra apenas numa vertente prática. O algoritmo apresentada abaixo teve como fonte de inspiração o algoritmo de procura `GBF*` (*Generalized Best First*, ou Melhor-Primeiro Generalizado). Esta versão do Melhor-Primeiro é similar à original mas permite abordar um leque mais alargado de problemas.

O algoritmo Melhor-Primeiro assume que o espaço de estados é um grafo `Or`, ou seja, que os vários arcos do grafo definem caminhos *alternativos* para a solução. Encarando o estado inicial como indicando um problema a resolver (encontrar um caminho para o estado final), basta resolver um dos vários sub-problemas (chegar de um dos sucessores do estado inicial ao estado final para o algoritmo terminar com sucesso. Existe um outro tipo de domínios que não é expressável em termos de um grafo *Or*. Nesses domínios, existem problemas no espaço de estados que só se consideram resolvidos quando *uma conjunção de sub-problemas o esteja*. Nestas situações, o espaço de estados toma a forma de um grafo *And/Or*.

O *GBF** é similar ao do *BF** mas possui duas funções de avaliação heurística e não apenas uma. Funciona considerando a cada passo o grafo que já foi gerado até ao momento. Identifica então os vários sub-grafos desse grafo que podem ser uma solução do problema em causa, designados por *bases-solução*. Um sub-grafo é uma base-solução se contiver o

estado inicial, para todos os nós conjuntivos (*And*) já explorados contém todos os seus sucessores e para todos os nós disjuntivos (*Or*) já explorados contém exactamente um dos seus sucessores. Adicionalmente, os estados presentes no grafo podem ainda conduzir a um estado final (ou seja, não devem existir estados já explorados que não têm sucessores).

As bases-solução são ordenadas de acordo com a primeira função heurística, isolando-se a que se mostre mais promissora tendo em vista a solução a atingir. De entre os estados ainda não explorados dessa base solução, escolhe de acordo com a segunda função de avaliação o que deve ser explorado em seguida. Os seus sucessores são acrescentados à lista de abertos e o processo prossegue com a escolha de uma nova base-solução. Para determinar se foi encontrada uma solução, existe um processo de rotulagem que rotula cada nó como *resolvido*, se é um nó solução, um nó conjuntivo em que todos os seus sucessores estão rotulados como resolvidos ou um nó disjuntivo em que pelo menos um dos seus sucessores está rotulado como resolvido. Caso contrário, um nó é rotulado *não-resolvido*. Quando o estado inicial for rotulado como resolvido, o algoritmo encontrou uma solução para o problema colocado.

É esta a situação presente no planeador criado devido ao problema das descrições alternativas. Cada mutação aplicável a uma descrição dá origem a um caminho alternativo para se chegar à solução. De cada vez que a aplicação de uma mutação dá origem a descrições alternativas, isso equivale a um ramo conjuntivo, dado que é necessário procurar um plano para cada uma das descrições alternativas antes de dar o planeamento por terminado. A nível formal, pode considerar-se a existência de um nó conjuntivo, não correspondente a nenhuma descrição, do qual derivam os nós correspondentes às várias descrições alternativas criadas pela aplicação da mutação. Na práctica, a implementação do algoritmo pode evitar a criação explícita desses nós. A função de avaliação que permite escolher a base-solução é o critério de escolha do ramo com menor ramificação. A função que permite escolher, dentro da base-solução, o próximo estado a explorar, é o critério da distância ao objectivo. O planeador só não corresponde perfeitamente à procura num grafo *And/Or* porque não é necessário que todas as alternativas de um ramo conjuntivo dêem origem a um plano, bastando que pelo menos uma o faça (o `GBF*` obriga a que todos os sub-problemas de um ramo conjuntivo conduzam à solução).

4.3.5 O Algoritmo

Tendo em conta as considerações tecidas nas secções anteriores, é agora possível apresentar o algoritmo final do planeador progressivo.

O planeador vai tentar encontrar um plano que conduza de uma descrição D a uma outra em que sejam satisfeitas as condições no conjunto Ω. Consideram-se duas listas de descrições: `DESCRIÇÕES-POR-EXPLORAR` e `DESCRIÇÕES-EXPLORADAS`, correspondentes às listas de `ABERTOS` e `FECHADOS` dos algoritmos de procura tradicionais. O algoritmo apre-

sentado é muito similar ao `GBF*`, excepto no que diz respeito à função de rotulagem dos estados, que foi adaptada para se obter o comportamento acima discutido.

O algoritmo abaixo dá origem a um novo método, `PLAN!`, que é invocado tendo como argumentos uma descrição e um conjunto de fórmulas a verificar na descrição final, e retorna um plano, caso o encontre.

1. Colocar D em `DESCRIÇÕES-POR-EXPLORAR`.
2. Encontrar todas as bases-solução do grafo construído até ao momento (englobando todas as descrições em `DESCRIÇÕES-POR-EXPLORAR` e `DESCRIÇÕES-EXPLORADAS`.
3. De entre as bases-solução identificadas no passo anterior, escolher a de menor ramificação, designada por G_0. A ramificação de uma base-solução pode ser obtida multiplicando os factores de ramificação individuais de cada ramo conjuntivo nela contida.
4. Se o algoritmo de rotulação indica que a descrição inicial está rotulada como *resolvida*, o algoritmo termina recuperando-se o plano associado a esse rótulo (ver algoritmo de rotulação das descrições abaixo). Antes do seu término, é executado o método `SYNCHRONIZE` indicando que terminou o raciocínio diacrónico.
5. Escolhe-se, de entre as descrições de G_0 ainda em `DESCRIÇÕES-POR-EXPLORAR` aquela que estiver mais próxima da solução, retirando-a dessa lista e colocando-a em `DESCRIÇÕES-EXPLORADAS`.
6. Se todas as condições em Ω são satisfeitas nessa descrição, rotula-se essa descrição como `resolvida` e propaga-se esse rótulo como definido no algoritmo abaixo.
7. Encontram-se as mutações aplicáveis em D. Aplicam-se essas mutações, obtendo-se de cada uma delas um *conjunto de descrições*. Esse conjunto será singular se a mutação correspondente não der origem a descrições alternativas. Caso contrário, é introduzido um novo ramo conjuntivo no grafo. Todas as descrições geradas são colocadas em `DESCRIÇÕES-POR-EXPLORAR`. Se não existirem mutações aplicáveis, deve-se rotular a descrição como `não-resolvida` e propagar esse rótulo, se necessário.

Para o algoritmo de planeamento acima funcionar correctamente, deve ser definido o algoritmo de rotulagem das descrições. Assim, serão consideradas três etiquetas possíveis para um estado: *indeterminado*, *não-resolvida* e *resolvida*. O primeiro rótulo, `indeterminado` é o valor por omissão para todas as descrições quando são criadas. Indica que ainda não se sabe se é possível atingir uma solução que inclua a descrição assim rotulada. O rótulo `não-resolvida` é aplicado às descrições que já se sabe não conduzirem a uma solução. Um exemplo de uma descrição nesta situação é um estado terminal (sem sucessores) que não satisfaz as condições em Ω. Finalmente, o rótulo `resolvida` indica

que uma descrição se encontra no caminho de uma solução. Por se tratar de um algoritmo de planeamento, assume-se ainda que associado ao rótulo de `resolvida` está o plano que conduz de essa descrição à que satisfaz as condições em Ω[1].

Posto isto, o algoritmo de propagação dos rótulos é o seguinte:

1. Uma descrição terminal é rotulada como `resolvida` se satisfizer todas as condições em Ω, e `não-resolvida` caso contrário. Caso seja rotulada como `resolvida`, o plano associado a esse rótulo é o plano vazio (não é necessário aplicar nenhuma mutação para chegar a uma descrição que satisfaça as condições em Ω).

2. Uma descrição disjuntiva (que corresponde a uma de várias formas alternativas para se atingir uma solução, tendo sido a única descrição resultante da aplicação de uma mutação) para a qual pelo menos um dos seus sucessores está rotulado como `resolvida` é, também, rotulada como `resolvida`, acrescentando-se ao início do plano associado ao rótulo a acção correspondente à mutação cuja aplicação conduziu à descrição com esse rótulo. Para tal, usam-se acções de controlo `SEQUENCE`. No caso trivial em que o segundo elemento da sequência seria o plano vazio, o plano associado a esta descrição é composto apenas por essa mutação. Se todos os seus sucessores estão rotulados com `não-resolvida`, então também a ela é associado esse rótulo.

3. Uma nó conjuntivo (que tem como descendentes as várias resultantes da aplicação da mutação que está na sua origem) é rotulado como *resolvida* quando todos os seus sucessores tenham um rótulo diferente de `indeterminada` e pelo menos um deles tenha o rótulo *resolvida*. O plano associado ao novo rótulo consistirá em utilizar uma acção de controlo `IF` com tantas condições quantos os sucessores da descrição rotulados como *resolvida*. As condições serão dadas pelos vários elementos do conjunto Δ obtido pela função Λ para a mutação envolvida, como descrito na secção 4.3.2. Se todos os sucessores de uma descrição conjuntiva estiverem rotulados como `não-resolvida`, então também a ela é associado esse rótulo.

Sempre que ao rotular um nó conjuntivo se encontrou um plano para pelo menos um dos seus sucessores, se um dado limite para o tempo ou recursos computacionais pré-estabelecido for excedido, pode considerar-se que todos os outros sucessores estão, também, rotulados como `resolvida` e substituir o plano que eventualmente pudesse vir a ser derivado para esses ramos por uma acção de controlo `ACHIEVE` com a conjunção das condições em Ω ainda não satisfeitas.

Como algoritmo de rotulação acima, garante-se que quando a descrição inicial está rotulada como `resolvida`, tem associado um plano, na forma usada pelo OK-BDI, para o

[1] Em termos implementacionais, o rótulo pode ser o plano propriamente dito. Nesse caso, para determinar o rótulo de uma descrição verifica-se se é `indeterminada` ou `não-resolvida`. Se for qualquer outra coisa (um plano), então considera-se que essa descrição está rotulada como `resolvida`.

objectivo a atingir. Idealmente, não seria necessário fazer esta associação dado que cada descrição contém a indicação das mutações aplicadas para chegar até ela. A consulta dessa informação numa descrição que satisfaça as condições de Ω seria suficiente para determinar um plano válido. No entanto, os planos gerados pelo algoritmo são planos condicionais. A informação guardada nas descrições não é suficientemente expressiva para representar esses planos, pelo que foi necessário recorrer a uma estrutura de dados auxiliar para o fazer.

4.4 O Planeador Regressivo

Uma segunda estratégia de controlo possível para o planeamento é, como já foi referido, o planeamento regressivo. Este tipo de planeamento pode ser dividido em duas fases complementares mas distintas, a da *determinação das regras envolvidas* e a da *propagação dos efeitos dessas regras*.

4.4.1 Determinação das Regras Envolvidas

Esta é a primeira fase do processo de planeamento. Nela, é implicitamente gerado um grafo de dependências entre as várias regras de inferência presentes no sistema, de acordo com as proposições que permitem satisfazer.

Ao planeador é fornecido um conjunto objectivo Ω e deve criar um plano que conduza da descrição actual D_0 a uma em que as condições Ω sejam satisfeitas. O planeador começa por verificar quais dessas condições são satisfeitas em D_0. Se todas o forem, o problema tem uma solução trivial. Caso contrário, é necessário planear.

A verificação da satisfação das condições em Ω é feita recorrendo aos métodos `BELIEVE?`, ou seja, recorrendo a inferência regressiva. Se existir alguma forma de derivar essas condições, mesmo que elas não ocorram explicitamente na base de conhecimento, serão consideradas como satisfeitas. Se não for possível derivá-las, podem ter ocorrido duas situações distintas. Ou não existem regras de dedução (sob a forma de transformadores `AntCq`) cuja aplicação conduza a essa derivação, ou existem regras que permitiriam derivar essas condições mas não podem ser aplicadas porque algumas das suas pré-condições também não são satisfeitas na descrição inicial.

O planeador identifica então quais as regras de inferência, sob a forma de transformadores `AntCq` e transformadores `MutationPcEf` cuja aplicação pode satisfazer as condições ainda não satisfeitas de Ω. Essas regras podem ser aplicáveis de imediato, ou podem, como já se referiu, ter algumas das suas pré-condições por satisfazer. Para as pré-condições não satisfeitas, é necessário repetir o processo, de forma recursiva, verificando-se quais as regras de inferência que poderiam, eventualmente, satisfazer as pré-condições não satisfeitas das outras regras.

É de notar que este processo de determinação das dependências entre as regras de inferência não implica a aplicação de nenhuma mutação, pelo que a verificação da satisfação das pré-condições das regras é sempre efectuada na descrição inicial D_0 recorrendo à inspecção directa do mundo (mediante a utilização dos transformadores `DoIf`), se necessário.

4.4.2 Propagação dos Efeitos das Regras

Se o problema tiver solução, chega-se, eventualmente, a uma situação em que todas as pré-condições de algumas regras de inferência consideradas são verdadeiras. Nessa altura, todas as regras que puderem ser aplicadas sê-lo-ão. Não existe necessidade de aplicar explicitamente os transformadores `AntCq` pois o próprio OK-BDI o fará automaticamente quando houver necessidade de determinar se os seus efeitos são ou não deriváveis. Isto acontece pois essa verificação é feita recorrendo aos métodos `BELIEVE?`, que efectuam todos os passos de inferência sincrónica necessários. São apenas aplicadas as várias mutações cujos pré-requisitos se encontram satisfeitos, o que dá origem a novas descrições. Nessas novas descrições, outras mutações poderão ser aplicáveis, repetindo-se o processo.

Eventualmente atinge-se uma situação em que ou são satisfeitas todas as condições em Ω ou não é possível aplicar mais mutações. No primeiro caso, atingiu-se uma descrição que satisfaz objectivo a partir da descrição inicial, pelo que a sequência de mutações que conduziu a essa descrição é um plano nas condições desejadas. Se não for possível aplicar mais mutações sem se ter atingido uma descrição em que todas as condições de Ω são satisfeitas, o planeamento falha para o caminho seguido, sendo necessário explorar outras alternativas caso existam. Se todas as alternativas tiverem sido esgotadas, não é possível encontrar o plano desejado e o processo de planeamento falha.

4.4.3 As Descrições Alternativas

No processo de planeamento regressivo também é necessário lidar com as descrições alternativas que a aplicação de uma mutação pode originar. A solução para esse problema é, no entanto, mais simples do que na estratégia de controlo progressiva.

Na primeira parte do processo de planeamento, são escolhidas as mutações cujos efeitos podem satisfazer as condições ainda não satisfeitas de Ω, ou as pré-condições não satisfeitas de outras regras. Isso é conseguido considerando os efeitos explícitos das regras de inferência contidos nos campos `CQ` e `EFFECTS` dos transformadores `AntCq` e `MutationPcEf`, respectivamente. Esses efeitos são obtidos garantidamente nas várias descrições a que eventualmente a aplicação de uma regra de inferência possa dar origem. No caso da aplicação de uma mutação, todas as descrições alternativas geradas satisfarão, obrigatoriamente, esses efeitos. Logo, não interessa nesta fase considerar as várias descrições, mas apenas as proposições Θ (ver secção 4.3.2), comuns a todas elas.

Na segunda parte, em que os efeitos das regras consideradas são propagados, as mutações são aplicadas, podendo, então, surgir as descrições alternativas. Nessa fase o algoritmo limita-se a aplicar as várias mutações pré-calculadas. Donde, lidar com as descrições alternativas consiste simplesmente em prosseguir o processo de aplicação das mutações como normalmente, tendo-se o cuidado de o despoletar em todas as descrições alternativas que possam surgir. São então colocando os sub-planos encontrados para cada alternativa como parâmetros de uma acção de controlo `IF` introduzida oportunamente.

4.4.4 Considerações a Nível de Eficiência

A bem da eficiência do processo de planeamento, não é necessário dividir explicitamente o planeamento nas duas fases como descritas, sendo possível implementá-lo de forma a que sejam realizadas, tanto quanto possível, em simultâneo de modo a aumentar a sua eficiência.

Para além disso, numa tentativa de diminuir a ramificação do processo de planeamento e o tamanho do plano gerado, quando se determinam, para as pré-condições de uma dada regra, quais as regras que permitem satisfazê-las, ordenam-se essas regras de forma a preferir as que permitam satisfazer simultaneamente um maior número de condições. Após essa ordenação, as várias regras encontradas vão ser exploradas em sequência. Evidentemente que se uma delas permite atingir um plano, torna-se desnecessário considerar as restantes.

Na fase de propagação dos efeitos das regras, de forma análoga, quando se verifica que existem várias mutações aplicáveis, aplica-se uma delas e segue-se uma estratégia em de procura em profundidade. Isto consiste em tentar aplicar mutações às novas descrições recém geradas até que tal não seja possível ou se encontre uma solução. Apenas é necessário retroceder e aplicar uma das outras mutações se o ramo explorado não conduzir a um plano.

4.4.5 O Algoritmo

O algoritmo do planeador regressivo, tendo em conta as considerações acabadas de tecer, está descrito abaixo. Dá origem a um novo método, `PLAN?`, que é invocado tendo como argumentos uma descrição inicial D_0 e um conjunto de proposições Ω. Retorna um plano que conduza de D_0 a uma descrição que satisfaz as condições em Ω, caso o encontre.

1. Verificar quais as fórmulas de Ω satisfeitas em D_0, usando os métodos `BELIEVE?`. Se todas o forem, o problema tem uma solução trivial. Caso contrário, colocam-se num conjunto CNS as condições ainda não satisfeitas.

2. Procurar os transformadores `MutationPcEf` que contenham no campo `EFFECTS` pelo menos uma das condições por satisfazer. Essas são as regras de inferência que vão

permitir satisfazer as condições que ainda não o foram. Associa-se a cada elemento de CNS um conjunto com as regras de inferência encontradas para ele.

3. Se, para algum dos elementos de CNS não for possível encontrar um transformador `MutationPcEf` que permita eventualmente satisfazê-lo, o processo de planeamento falha para este ramo.

4. Ordenam-se as regras de inferência encontradas acima de acordo com o critério descrito. Para tal, verifica-se em quantos elementos de CNS ocorre cada regra.

5. Isolar de entre todos os transformadores `MutationPcEf` na base de conhecimento os que são aplicáveis. Escolhe-se o mais promissor de acordo com a ordenação acima e aplica-se esse transformador. Acrescenta-se a acção correspondente à mutação aplicada ao final do plano guardado em `PLANO-ACTUAL` recorrendo uma acção de controlo `SEQUENCE` e retiram-se de CNS as condições satisfeitas pela à regra aplicada. Existem agora dois casos a considerar:

   (a) Se a mutação aplicada *deu origem a apenas uma descrição*, verifica-se se a descrição actual satisfaz as condições em Ω. No caso afirmativo, `PLANO-ACTUAL` contém um plano que consiste nas mutações aplicadas para chegar a essa descrição. Caso contrário, invoca-se o algoritmo recursivamente neste passo (de modo a verificar que outras regras são ainda aplicáveis).

   (b) Se a regra aplicada *deu origem a descrições alternativas*, é necessário verificar quais dessas descrições satisfazem as proposições em Ω. Para as que o não fizerem, invoca-se sucessivamente o planeador recursivamente neste passo do algoritmo. Se nenhuma das descrições alternativas conduz a um plano, este ramo do planeamento falha. Se, por outro lado, pelo menos uma das alternativas conduz a uma solução, é criado um plano condicional, colocando-se em `PLANO-ACTUAL` uma acção de controlo `IF`, com uma condição para cada descrição alternativa que conduza a uma solução do problema, associada ao plano para ela derivado.

6. Caso a regra escolhida não conduza a uma solução, colocam-se em `PLANO-ACTUAL` e `CNS` os seus valores antes da aplicação da regra e aplica-se outra regra. Isto é feito até se atingir a solução ou não haver mais regras aplicáveis.

7. Quando não for possível aplicar mais regras, e não se obteve um plano, colocam-se em `PLANO-ACTUAL` e `CNS` os seus valores originais. Escolhe-se, então, pela ordem determinada no passo 4, a regra de inferência seguinte e verificam-se quais as suas pré-condições não satisfeitas em D_0. Colocam-se essas pré-condições em CNS. Condições que já sejam referidas em CNS não são duplicadas. Se isso implicar que não são feitas alterações a CNS (o que corresponde dependências circulares entre as mutações), este ramo do planeamento falha. Caso contrário, invoca-se recursivamente o planeador no passo 2

8. Se o planeador falhar para a opção feita, deve escolher-se a regra de inferência seguinte. Se se esgotarem todas as possibilidades, processo de planeamento falha.

O algoritmo acima implementa de forma integrada as duas fases do planeamento regressivo. Os passos 5 e 6 correspondem à propagação dos efeitos das regras, e os restantes à determinação das regras envolvidas. Foi usada uma variável adicional, `PLANO-ACTUAL`, que contém a cada momento o plano que conduz à descrição em consideração. Normalmente, isto não seria necessário, dado que, quando se atinge uma descrição em que são satisfeitas todas as condições em Ω, o campo `MUTATIONS` da situação em `CURRENT-SITUATION` contém a sequência de mutações aplicadas para atingir essa descrição. No entanto, como vão ser gerados planos condicionais, não é possível saber por esse meio os planos para os vários ramos alternativos do planeamento. Logo, é necessário recorrer a uma estrutura de dados auxiliar para registar essa informação.

É de notar que este planeador pode não encontrar um plano com a menor ramificação nem com o maior número de ramos planeados. Garante apenas que encontra um plano caso ele exista.

4.5 Algumas Considerações sobre os Algoritmos

Foram apresentados nas secções anteriores dois algoritmos de planeamento, baseados em duas estratégias de controlo diferentes. Para se ter uma ideia mais concreta da sua utilidade e para ser possível classificá-los em relação a outros planeadores, importa caracterizá-los quanto a vários aspectos. Nas secções seguintes, os planeadores serão classificados quanto à sua *solidez*, *completude* e *sistematicidade*.

Convém antes que mais notar que os planeadores apresentados se baseiam em várias suposições. Supõe-se que os algoritmos têm conhecimento perfeito sobre as condições iniciais do problema e que as únicas alterações que podem ocorrer no mundo se devem às acções do agente. Estas suposições são feitas na grande maioria dos sistemas de planeamento, como o `STRIPS` [Fikes e Nilsson 1971], o `TWEAK` [Chapman 1987] ou o `UCPOP` [Penberty e Weld 1992]. Uma outra suposição é feita: assume-se que não foram cometidas incorrecções na representação das mutações e que a correspondência entre as mutações representadas no sistema e as respectivas acções físicas está correcta. É à luz destas suposições que os novos planeadores serão caracterizados. Aliás, o OK-BDI está numa posição privilegiada em relação a outros sistemas para aceitas essas suposições, dado que, ao se encontrar fisicamente localizado, pode detectar e facilmente recuperar de casos em que essas suposições falhem.

4.5.1 Solidez

Um planeador diz-se *sólido* se garante que todos os planos por si gerados são sólidos, ou seja, atingem os objectivos estabelecidos.

Os planeadores apresentados baseiam-se na aplicação das regras de inferência correspondentes directamente às acções físicas executáveis pelo agente. Deveriam, pois, ser sólidos. Isto pode não se verificar porque os algoritmos não sabem à priori qual o resultado da aplicação das diversas mutações, o que está na génese do problema das descrições alternativas. Se todas as mutações derem origem a apenas uma descrição (algo que, de resto, também é assumido na maior parte dos planeadores), os algoritmos são sólidos.

No caso geral, para que um plano gerado pelos algoritmos seja sólido, é necessário que, para todas as descrições alternativas resultantes da aplicação de uma mutação nesse plano, exista um sub-plano que conduz à solução. Se existirem alternativas não satisfeitas, o plano pode falhar na altura da sua execução, caso essa alternativa vier a ser escolhida. Assim, os algoritmos não são sólidos. Para que o fossem, seria necessário fortalecer a regra que considera que um dado ramo conjuntivo do espaço de descrições conduz a um plano se apenas algumas das várias alternativas o fizer. Deveria-se obrigar a que *todas* as alternativas conduzam à solução. Isto não seria correcto dado que algumas das descrições inferidas podem ser meros artifícios gerados pela lógica sem nenhuma correspondência no mundo físico.

4.5.2 Completude

Um planeador é *completo* se para todos os problemas para os quais existe uma solução, essa solução é encontrada. Dito de outra forma, se o planeador não consegue encontrar um plano para um dado problema, então ele não existe.

Os planeadores apresentados são completos. O planeador progressivo explora, em última análise, todo o espaço de descrições em busca de uma solução. Isto decorre directamente do facto de ser baseado num algoritmo de procura, o `GBF*`. O planeador regressivo também é completo. Na sua primeira fase, estabelece um grafo que regista *todas* as dependências entre as regras de inferência presentes na base de conhecimento. Assim, se na segunda fase, ao explorar esse grafo não encontrar uma solução, é garantido que não existe nenhuma sequência de aplicações de regras de inferência que conduza a essa solução.

4.5.3 Sistematicidade

Um planeador *sistemático* é aquele que nunca considera ao longo do seu funcionamento o mesmo plano potencial mais do que uma vez. Para o planeador progressivo, isso é verdade. Dado que se baseia no algoritmo `GBF*`, é sabido que não são visitados estados repetidos.

Donde, esse planeador é sistemático. Algo similar se passa no que diz respeito ao planeador regressivo. Embora quando é feita a propagação dos efeitos das regras nada impeça que se cheguem a situações já antes atingidas, a sequência de mutações considerada para as atingir será, necessariamente diferente. Donde, o planeador regressivo também é sistemático.

4.6 Criação de Novas Mutações

A criação de um plano fornece uma indicação da existência de mutações que, quando executadas sequencialmente, conduzem a descrições em que se verificam algumas propriedades. Dentro de um mesmo domínio, ocorrem, em problemas de planeamento diferentes, situações análogas. A detecção dessas situações pode facilitar futuros processos de planeamento, tornando-os mais rápidos e eficientes.

O modo para conseguir, no formalismo apresentado, capturar essa informação é a *criação de novas mutações*. Isto equivale à criação de macro-operadores na procura. A criação de macro-operadores e a reutilização de planos em geral tem sido amplamente estudada em inúmeros trabalhos, desde o `ABSTRIPS` e o `STRIPS com Macro-Operadores` ao `Prodigy` [Veloso et al. 1995] e ao planeador descrito em [McAllester e Rosenblitt 1991].

A utilização destes macro-operadores trás diversas vantagens como, por exemplo, a diminuição da profundidade a que se encontra a solução procurada. Isto permitindo encontrá-la mais facilmente e diminui a complexidade espacial do algoritmo. Adicionalmente, espera-se que os novos operadores desempenhem um papel importante no sentido de guiar o processo de planeamento pelos caminhos mais relevantes para o domínio específico a tratar.

Para efectuar a agregação de duas mutações é usado o método `AGGREGATE`, definido no capítulo 3. De cada vez que é derivado um plano, esse método pode ser aplicado sucessivamente às várias mutações que constituem esse plano. Cria-se, assim, uma nova mutação (e a correspondente acção), para as várias etapas do plano, uma das quais representa todo o plano com uma única acção.

Este processo não é, no entanto, tão imediato como o descrito no parágrafo anterior. Os planos gerados pelos planeadores apresentados podem conter ramos alternativos. Não é possível nesses casos criar uma mutação que reflicta esse comportamento, dado que a regra de inferência usada apenas consegue representar a sequenciação de mutações. Nestes casos, deve ser criada uma mutação diferente, por agregação sucessiva, para cada um dos sub-planos lineares obtidos. Isto corresponde a criar uma nova mutação para cada um dos ramos condicionais do plano. A partir do plano representado abaixo, por exemplo, poderiam ser criadas três novas mutações, correspondentes, respectivamente, à agregação de μ_1 e μ_2, de μ_3 e μ_4 e de μ_5 e μ_6.

SEQUENCE(μ_1,SEQUENCE(μ_2,IF([$P \rightsquigarrow$ SEQUENCE(μ_3,μ_4)][$Q \rightsquigarrow$ SEQUENCE(μ_5,μ_6)])))

4.7 Integração com os Métodos Existentes

Nas secções anteriores foram descritos dois planeadores que podem ser usados nas mais variadas aplicações. Resta, agora, encontrar a forma de os integrar com o OK-BDI.

Essa integração é relativamente simples. É necessário, em primeiro lugar, identificar em que situações necessita o OK-BDI de planear. Depois, basta invocar `PLAN!` ou `PLAN?` nessas situações e utilizar os resultados por eles devolvidos.

Para além de pedidos explícitos de planeamento por parte do utilizador, o OK-BDI pode necessitar planear em duas situações distintas. A primeira ocorre no método `INTEND` para as instâncias de `Act Term`. Esse método resulta na execução de uma dada acção física sobre o mundo real. Se a acção em causa for primitiva, é executada imediatamente. Se for complexa, é necessário encontrar uma decomposição dessa acção em acções primitivas, que podem ser executadas. Essas decomposições são armazenadas, no OK-BDI, nos transformadores `PlanAct`. Pode, no entanto, dar-se o caso de não existir nenhuma decomposição conhecida para um dada acção complexa embora se conheçam, mediante transformadores `ActEffect`, quais devem ser os seus efeitos. Nesse, caso, deve encontrar-se um plano que os atinja. Isto pode ser feito recorrendo a um dos planeadores aqui apresentados, alterando-se o método `INTEND` para a classe `Act Term` de modo a que o invoque. O planeador será invocado tendo como argumentos a descrição actual e o conjunto de proposições que representam os os efeitos que a acção complexa deve atingir. Se um plano for atingido, deve ser armazenado na base de conhecimento sob a forma de um transformador `PlanAct` referente à acção complexa em causa.

O segundo local em que o planeamento pode ser necessário é, mais uma vez, no método `INTEND`, mas da classe `ACHIEVE`. Esta classe é uma sub-classe das acções de controlo (`Control Act`). As suas instâncias são caracterizadas por uma proposição e a sua execução leva o OK-BDI a procurar uma forma de tornar verdade essa proposição. Isto leva-o a procurar um plano para o efeito, recorrendo à informação guardada nos transformadores `PlanGoal`. Se não existir nenhum transformador `PlanGoal` referente à proposição a atingir é necessário invocar um planeador que crie um plano nessas condições. Assim, o método `INTEND` para a classe `ACHIEVE` deve ser alterado de modo a que um dos planeadores seja invocado tendo como argumentos a descrição actual e a proposição a atingir. Se um plano for encontrado, deve ser armazenado na base de conhecimento mediante a utilização de um transformador `PlanGoal`.

É necessário analisar esta proposição e verificar se se trata de uma conjunção. Nesse caso, cada um dos operandos dessa conjunção dá origem a um elemento diferente do conjunto passado como segundo argumento para o planeador. O processo para encontrar uma

situação que satisfaça varias proposições simultaneamente é equivalente ao usado para encontrar uma situação em que se verifica a uma conjunção dessas mesmas proposições. No entanto, como as mutações só contemplam fórmulas literais, é necessário efectuar esta decomposição. Isto não garante que todas as fórmulas obtidas são literais, mas pode, potencialmente, resolver alguns casos. Em particular, quando um dos planeadores delega para o futuro o planeamento de um ramo conjuntivo, cria uma acção `ACHIEVE` com a conjunção dos objectivos por atingir, que deve ser decomposta da forma descrita na altura de voltar a planear.

4.8 Exemplo de Aplicação dos Planeadores

Para exemplificar o funcionamento dos planeadores e ajudar à compreensão dos mesmos, é aqui apresentado um exemplo detalhado de como podem ser aplicados a um dado problema. O mesmo problema será submetido aos dois processos de planeamento, de modo a facilitar uma comparação entre eles.

4.8.1 O Domínio

Está muito calor e o Sr. Silva, de férias num hotel no Algarve, decidiu ir dar um mergulho para se referescar. Pode fazê-lo na praia ou na piscina de hotel. O Sr. Silva tem, ainda, uma Nota e uma Moeda. Com esse dinheiro pode comprar um protector solar. A entrada na piscina apenas pode ser paga com a nota. O toldo na praia pode ser pago com a moeda.

Estas afirmações podem ser representadas, na Lógica das Mutações (e, consequentemente, no formalismo integrado), pelas proposições abaixo. Os predicados nelas encontrados têm o seguinte significado:

Moeda:	O Sr. Silva tem a moeda.
Nota:	O Sr. Silva tem a nota.
Dinheiro:	O Sr. Silva tem dinheiro.
Protector:	O Sr. Silva tem o protector solar.
Hotel:	O Sr. Silva está no hotel.
EntradaPiscina:	O Sr. Silva está na entrada da piscina.
Piscina:	O Sr. Silva está na piscina.
Praia:	O Sr. Silva está na praia.
Agua:	O Sr. Silva está na agua.

$Moeda$	$\langle 1,\{\{1\}\}\rangle$
$Nota$	$\langle 2,\{\{2\}\}\rangle$
$(Nota \wedge Moeda) \rightarrow Dinheiro$	$\langle 3,\{\{3\}\}\rangle$
$Hotel$	$\langle 4,\{\{4\}\}\rangle$
$Hotel \rightarrow EntradaPiscina$	$\langle 5,\{\{5\}\}\rangle$
$\mu_1(\{Hotel, Dinheiro\}; \{Protector\}; comprar - protector)$	$\langle 6,\{\{6\}\}\rangle$
$\mu_2(\{Nota, EntradaPiscina\}; \{Piscina\}; ir - piscina)$	$\langle 7,\{\{7\}\}\rangle$
$\mu_3(\{Piscina\}; \{Agua\}; mergulho - pscina)$	$\langle 8,\{\{8\}\}\rangle$
$\mu_4(\{Moeda, Protector\}; \{Praia\}; ir - praia)$	$\langle 9,\{\{9\}\}\rangle$
$\mu_5(\{Praia\}; \{Agua\}; mergulho - praia)$	$\langle 10,\{\{10\}\}\rangle$

Podemos ainda considerar, as fórmulas seguintes, não pertenentes à descrição inicial, mas que irão aparecer noutras descrições do espaço de descrições do problema:

$fbf_{11} = Protector$
$fbf_{12} = Praia$
$fbf_{13} = Piscina$
$fbf_{14} = Agua$

O espaço de descrições a que as $fbfs$ anteriores dão origem está representado na figura 4.5. O conjunto objectivo Ω é igual a $\{Agua\}$. Nas secções seguintes, será exemplificado o funcionamento dos dois planeadores para este caso.

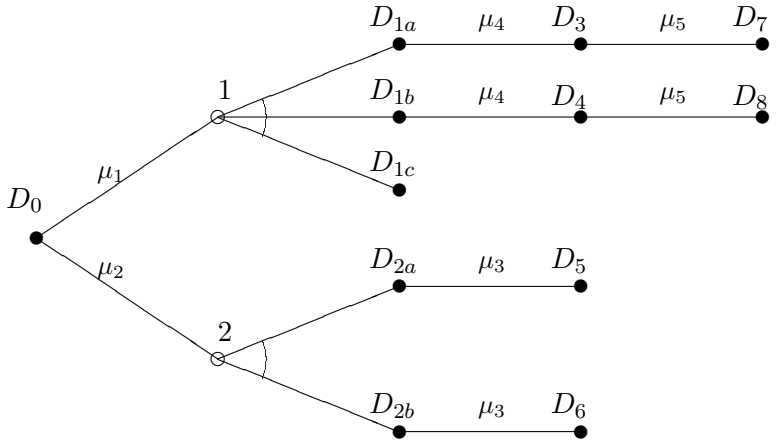
4.8.2 O Planeador Progressivo

O planeador progressivo começa por inicializar `DESCRIÇÔES-EXPLORADAS` como sendo a lista vazia, e `DESCRIÇÕES-POR-EXPLORAR` como contendo o grafo composto apenas por D_0, estando essa descrição rotulada como `indeterminada`. Neste caso, apenas existe uma base solução, o grafo que apenas contém D_0, pelo que será essa a escolhida para expansão.

Escolhe-se o único nó não expandido da base solução. Esse nó é retirado da lista em `DESCRIÇÕES-POR-EXPLORAR` e colocado em `DESCRIÇÔES-EXPLORADAS`. É, em seguida, expandido, o que dá origem às descrições D_{1a}, D_{1b} e D_{1c}, por aplicação da mutação `comprar-protector`, e a D_{2a} e D_{2b}, se a mutação aplicada for `ir-piscina`. São, pois, dois nós conjuntivos, dado existirem vários resultados possíveis para a aplicação da mutação. Os novos nós, rotulados como `indeterminado`, são colocados em `DESCRIÇÕES-POR-EXPLORAR`.

Para o passo seguinte, existem duas bases-solução de entre as quais escolher (figuras 4.6(a) e 4.6(b)). A escolha deve ser feita recorrendo ao critério da ramificação. a primeira tem uma ramificação de 3, e a segunda de 2, pelo que será essa a escolhida.

Dessa entre os nós dessa base-solução, dois ainda se encontra por explorar, D_{2a} e D_{2b}. Deve ser escolhido de entre eles o que mais se aproxime de Ω. Como ambos



$D_0 = \{1,2,3,4,5,6,7,8,9,10\}$
$D_{1a} = \{1,2,5,6,8,9,10,11\}$
$D_{1b} = \{1,3,5,6,8,9,10,11\}$
$D_{1c} = \{2,3,5,6,8,9,10,11\}$
$D_{2a} = \{1,3,4,6,7,8,9,10,13\}$
$D_{2b} = \{1,3,5,6,7,8,9,10,13\}$
$D_3 = \{2,5,6,7,8,9,10,12\}$
$D_4 = \{3,5,6,7,8,9,10,12\}$
$D_5 = \{1,3,4,6,7,8,9,10,14\}$
$D_6 = \{1,3,5,6,7,8,9,10,14\}$
$D_7 = \{2,5,6,7,8,9,10,14\}$
$D_8 = \{3,5,6,7,8,9,10,14\}$

Figura 4.5: Espaço de Situações

se encontram à mesma distância (falta uma fbf), a escolha é arbitrária. Assumindo que foi escolhida D_{2a}, essa descrição é retirada de `DESCRIÇÕES-POR-EXPLORAR` e colocada em `DESCRIÇÔES-EXPLORADAS`. A sua expansão gera uma nova descrição, D_5, rotuladas como `resolvida` dado que contém o conjunto objectivo Ω. Associado a este rótulo encontra-se o plano vazio. O algoritmo de propagação dos rótulos atribui, assim, a D_{2a} o rótulo `resolvida`, tendo associado o plano μ_3). Em seguida, D_5 é colocada em `DESCRIÇÕES-POR-EXPLORAR`.

Mais uma vez é escolhida uma base solução (figura 4.6(c)). Dado que a descrição inicial ainda está rotulada como `indeterminada` É, pois, necessário expandir um novo nó. O único nó na base-solução ainda em `DESCRIÇÕES-POR-EXPLORAR` é D_{2b}, pelo que é esse o escolhido, dando origem a D_6. Também essa descrição é rotulada como `resolvida`. Isto faz com que a D_{2b} também seja associado esse rótulo, com o plano associado de μ_3, o que conduz ao plano `IF(`$[Hotel \rightsquigarrow \mu_3][True \rightsquigarrow \mu_3]$`)` no nó 2. Continuando a propagação dos rótulos, também D_0 é rotulado como `resolvida`, sendo-lhe associado o plano `SEQUENCE(`μ_2`,IF(`$[Hotel \rightsquigarrow \mu_3][True \rightsquigarrow \mu_3]$`))`. É este o plano final, devolvido pelo algoritmo na altura da escolha da próxima base-solução (figura 4.6(d)) .

É de notar como, neste caso, apesar de as duas alternativas existentes na solução apresentada conduzirem à aplicação da mesma mutação, é vantajosa a criação de planos

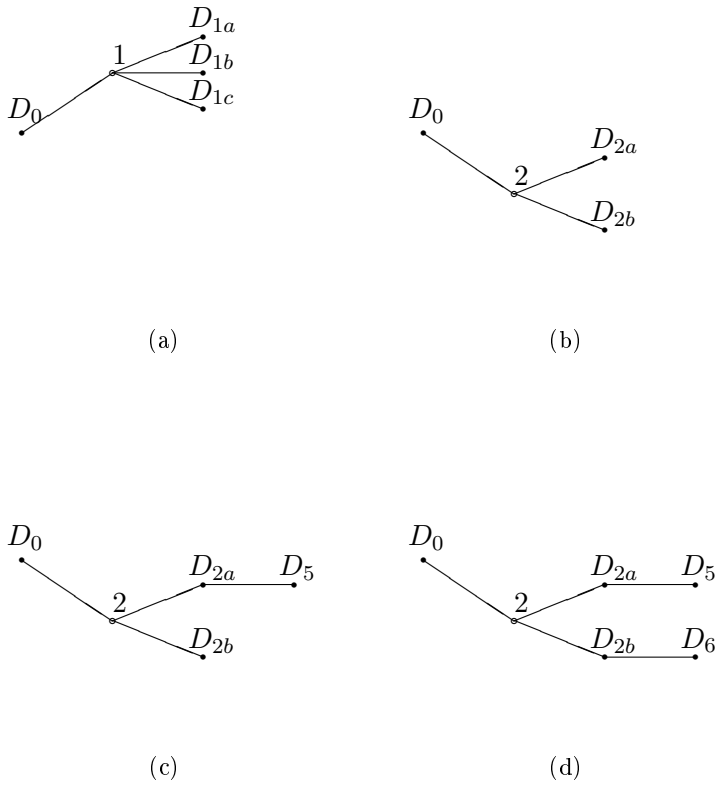


Figura 4.6: Bases Solução

alternativos, dado que uma das alternativas (em que ao comprar o bilhete para a piscina deixamos de estar no hotel!) é um mero artifício do formalismo. Ao executar o plano, o agente verifica que, efectivamente ainda se encontra no hotel, e age correctamente.

4.8.3 O Planeador Regressivo

Vamos agora ver como um plano similar pode ser obtido recorrendo ao planeador regressivo. Inicialmente, o conjunto `CNS` está vazio e `PLANO-ACTUAL` não contém qualquer plano. De igual modo, nunhum dos elementos de Ω é satisfeito em D_0. Coloca-se, pois, em `CNS` o elemento de Ω não satisfeito, ao qual são associadas as mutações que o podem vir a fazer. Ficamos, pois, com:

$$\texttt{CNS}= \{\langle Agua, \{\mu_3, \mu_5\}\rangle\}$$

Nenhuma das mutações referidas em `CNS` é aplicável em D_0, pelo que vamos escolher uma delas e verificar quais das suas pré-condições não são satisfeitas nessa descrição. Deve ser escolhida a mutação que ocorra em mais elementos de `CNS`. Neste caso podemos escolher arbitrariamente uma delas, por exemplo μ_3. Obtemos, assim:

$$\texttt{CNS}= \{\langle Agua, \{\mu_3, \mu_5\}\rangle, \langle Piscina, \{\mu_2\}\rangle\}$$

Mais uma vez, verificamos se alguma das mutações referida em `CNS` é aplicável em D_0. Verificamos que μ_2 o é. A primeira pré-condição, $Nota$, ocorre explicitamente na descrição. A segunda, $EntradaPiscina$ não ocorre explicitamente mas pode ser derivada a partir de fbf_5. Como a verificação da satisfação das pré-condições é feita recorrendo ao método $BELIEVE?$, essa regra será aplicada, obtendo-se o resultado pretendido.

Assim, o algoritmo começa a fazer a propagação dos efeitos das mutações. Começa por aplicar a mutação μ_2. As variáveis `CNS` e `PLANO-ACTUAL` ficam com os valores

$$\texttt{CNS}= \{\langle Agua, \{\mu_3, \mu_5\}\rangle\}$$
$$\texttt{PLANO-ACTUAL} = \mu_2$$

A aplicação de μ_2 deu origem a duas descrições alternativas, D_{2a} e D_{2b}. Logo, deve verificar-se se é possível satisfazer as condições de Ω em ambas. Considerando, em primeiro lugar a descrição D_{2a}, verificamos que há condições de Ω não satisfeitas. Porém, é possível aplicar μ_3. Isto conduz a D_5, com:

$$\texttt{CNS}= \{\}$$
$$\texttt{PLANO-ACTUAL} = \mu_3$$

Em D_5, todas as condições em Ω são satisfeitas, pelo que o planeamento termina para este ramo, com sucesso.

Falta, ainda, verificar D_{2b}. Também nesta descrição μ_3 é aplicável, conduzindo à descrição D_6, em que todas as condições em Ω são satisfeitas, com os valores:

$$\texttt{CNS}= \{\}$$
$$\texttt{PLANO-ACTUAL} = \mu_3$$

Uma vez verificadas as duas alternativas com origem na aplicação da mutação μ_2, verifica-se que pelo menos uma delas conduz a uma solução, pelo que pode ser criado um plano condicional. Assim, o plano obtido a partir das duas alternativas geradas pela aplicação de μ_2 é `IF(`$[Hotel \rightsquigarrow \mu_3][True \rightsquigarrow \mu_3]$`)`. Como já tinha sido aplicada μ_2 propriamente dita, o plano final fica `SEQUENCE(`μ_2`,IF(`$[Hotel \rightsquigarrow \mu_3][True \rightsquigarrow \mu_3]$`))`, como esperado.

4.9 Conclusões

Os dois planeadores aqui apresentados demonstram de forma inequivoca como o novo formalismo pode ser usado para a criação das mais diversas aplicações, das quais o planea-

mento é um mero exemplo.

A facilidade em efectuar raciocínio diacrónico, aliada ao motor racional do OK-BDI tornou simples a implementação dos algoritmos, permitindo que se abstraiam de certos detalhes. Um exemplo claro disso encontra-se no planeador regressivo, em que não é necessário considerar explicitamente os transformadores `AntCq`, dado que o motor racional o faz automaticamente quando necessário (na invocação do método `BELIEVE?`.

Isto prende-se directamente com o facto do formalismo possuir grande expressividade. Nos exemplos apresentados, a maior vantagem desta espressividade encontra-se na possibilidade de criação de planos condicionais. Considerar estes planos não só aumenta a utilidade dos planeadores, como minimiza o problema das descrições alternativas existente na Lógica das Mutações.

Capítulo 5

Conclusões

Neste capítulo são apresentados os principais resultados obtidos ao longo do desenvolvimento desta tese, bem como as principais contribuições dela resultantes. São ainda indicadas formas através das quais o estudo desenvolvido pode ser continuado, indicando potênciais trabalhos futuros nesta área.

5.1 Resultados Obtidos

A motivação para esta tese foi a de obter um formalismo que integre as principais características da Lógica das Mutações e do OK-BDI. Ambos possuem características que podem reconhecidamente ser de uma utilidade inestimável para qualquer sistema capaz de demonstrar um comportamento inteligente baseado em raciocínio do senso comum. O formalismo integrado deveria, pois, ser capaz de raciocinar sobre mudança e planear de forma a satisfazer as mais diversas necessidades, mas estando ao mesmo tempo situado no mundo, podendo agir sobre ele sempre que necessário de forma simples e integrada.

Para atingir o objectivo proposto, tornou-se imperativa uma análise dos dois formalismos a integrar, de modo a evidenciar que aspectos merecem uma atenção mais cuidada na altura da integração. Nessa análise, foi possível constatar que tanto na Lógica das Mutações como no OK-BDI existe uma forma de representar acções, embora o conceito de "acção" difira nos dois casos.

Na Lógica das Mutações as acções, sob a forma de mutações, reificam a mudança. Os seus efeitos reportam-se apenas a um modelo do mundo, representado pelas diversas proposições na base de conhecimento, modelo esse capaz de manter informação relativa a diferentes instantes de tempo. No OK-BDI, as acções consideradas são acções que se reportam a um mundo real. No entanto, as crenças do agente modelado pelo OK-BDI apenas se reportam ao instante actual, não existindo nenhuma forma de se raciocinar, de forma natural, sobre quais as características do mundo nalgum instante que não esse.

No entanto, o mecanismo subjacente à mudança nos dois formalismos é similar. Esta constatação permitiu chegar à principal conclusão desta análise:

> Tanto na Lógica das Mutações como no OK-BDI existe um modo de representar a mudança, sob a forma de mutações e acções físicas, respectivamente. Esse conceito, embora não seja idêntico nos dois casos, é análogo, uma vez que as mutações e as acções podem ser encaradas como diferentes pontos de vista sobre a mesma mudança. As mutações permitem ao agente pensar sobre os efeitos da mudança e as acções permitem ao agente efectivar esses efeitos, mas a mudança em causa é a mesma.

Posto isto, concluimos que deve existir uma correspondência unívoca entre as mutações e as acções físicas presentes no sistema. Como o OK-BDI se revelou bastante flexível e expansível, foi tomada a decisão de fazer a integração enquadrando as facetas relevantes da Lógica das Mutações no OK-BDI e não vice-versa. O principal aspecto a considerar, nessa integração, foi o modo de fornecer ao OK-BDI a capacidade de efectuar raciocínio diacrónico. Para tal, foi necessário encontrar um mecanismo que lhe permita manter na base de conhecimento informação relativa a diferentes instantes de tempo. Isso foi conseguido recorrendo ao sistema de manutenção de verdade subjacente ao OK-BDI, utilizando-o de modo a satisfazer as novas necessidades. O segundo aspecto em que foi necessário alterar o OK-BDI foi a criação de um novo transformador que emula o funcionamento das mutações.

No processo de integração do raciocínio diacrónico no OK-BDI surgiu um problema que se mostrou especialmente relevante. Dado o OK-BDI se encontrar situado no mundo real, é como que obrigado a lidar em simultâneo com dois mundos diferentes: um interno, resultante das suas crenças, que modela o mundo real, e o mundo real propriamente dito, que tem um estado bem definido e acessível ao agente.

É necessário ter especial cuidado com a informação que pode ser usada a cada momento no processo de inferência. Se o agente estiver apenas a raciocinar sobre o momento presente, nada lhe impede que observe directamente o mundo e use o resultado dessa observação como informação adicional para o seu processo de raciocínio. Se, por outro lado, estiver a efectuar raciocínio diacrónico, pode estar a considerar o estado do mundo num instante de tempo que não o actual. Nesse caso, o resultado de uma eventual inspecção do estado do mundo real não deve ser usado na inferência, dado não existirem garantias de que as propriedades que o mundo possui actualmente ainda se mantenham no instante a que o raciocínio do agente se está a referir. Isto levanta alguns problemas de sincronísmo que obrigaram à alteração do motor racional do OK-BDI de forma a evitar a sua presença. O segundo resultado principal deste trabalho encontra-se enunciado em seguida:

> Ao incorporar mecanismos de raciocínio diacrónico a um sistema situado no mundo, é necessário um especial cuidado de modo a garantir que o agente

> tem presente a cada momento as diferenças entre o estado actual do mundo e o estado actual do seu modelo mental do mesmo, que podem nem sempre reportar-se ao mesmo instante de tempo.

Na posse de um formalismo capaz de efectuar de forma integrada o raciocínio diacrónico e acções sobre o mundo, procedeu-se ao estudo da forma através da qual é possível efectuar planeamento de acções usando as capacidades desse formalismo, dado ser essa uma característica importante de qualquer sistema que manifeste um raciocínio de senso comum. Para tal, é importante reconhecer a importância da estratégia de controlo usada. Na posse de um sistema capaz de efectuar raciocínio diacrónico, o planeamento reduz-se ao raciocínio. O que mais importa é o modo de conduzir esse raciocínio de modo a obter as conclusões desejadas.

O maior problema encontrado na criação dos planeadores foi, sem dúvida alguma, o problema das descrições alternativas. Este problema não é mais do que a manifestação no formalismo integrado do problema clássico das múltiplas extensões. Usualmente, não existe forma de o resolver dado que o planeador não possui a informação suficiente à priori sobre os resultados mais prováveis (ou mesmo possíveis) das várias acções. Normalmente, é pedido o auxílio externo do utilizador para indicar qual das extensões é a correcta.

A intervenção externa num sistema que se requer automático não é, de forma alguma, a solução ideal. Donde, foram estudadas as formas pelas quais o facto de o planeador se encontrar situado no mundo, ao contrário do que ocorre nos sistemas clássicos, pode auxiliar a resolver esse problema.

Verifica-se que, dado existir a possibilidade de inspeccionar o mundo para descobrir exactamente qual o seu estado minimiza o problema ao ponto de o tornar de simples resolução. Efectivamente, a escolha de qual das descrições alternativas obtidas pela aplicação de uma mutação é a correcta pode ser feita automaticamente, se o mundo for inspeccionado em busca das propriedades que distinguem as várias alternativas. Aquelas que se verificarem permitem a escolha da descrição correcta. Esta verificação continua a não poder ser feita durante o processo de planeamento propriamente dito dados os problemas de sincronísmo já referidos. O terceiro resultado a salientar desta tese é, pois:

> O problema das múltiplas extensões reveste-se de grande relevância em sistemas completamente alheios ao mundo que modelam. A partir do momento em que é dada ao sistema a possibilidade de interagir com esse mundo, o problema torna-se de fácil solução.

Assim, os planeadores desenvolvidos permitem a geração de planos condicionais ou contingentes. O plano prevê as diversas possibilidades que possam vir a ocorrer como resultado de uma acção, introduzindo ramos condicionais no plano final para que, de acordo

com o que na realidade se vier a verificar no mundo, um plano alternativo adequado venha a ser seguido.

5.2 Contribuições

As principais contribuições resultantes da investigação que está na génese desta tese são:

1. O estudo das características essenciais de dois sistemas de tipos diferentes: a Lógica das Mutações preocupada com o raciocínio sobre mudança mas completamente alheia ao mundo, e o OK-BDI, imerso no mundo mas incapaz de raciocinar sobre diferentes instantes de tempo.

2. O desenvolvimento de um formalismo que integra não só o raciocínio clássico e a acção, mas também o raciocínio sobre diferentes instantes de tempo.

3. A análise de problemas que podem surgir no processo de planeamento usando formalismos como a Lógica das Mutações, em especial o problema das descrições alternativas, indicando-se como o formalismo integrado fornece mecanismos para os solucionar, mediante a criação de planos condicionais.

4. A criação de um planeador de acções com base no formalismo integrado, com o intuito de ilustrar a sua utilidade. Esse planeador, ao estar inserido no mundo para o qual planeia, consegue, em maior ou menor grau, gerar planos de qualidade superior a sistemas análogos incorpóreos.

5.3 Trabalho Futuro

Existem duas vertentes da investigação apresentada nesta tese para as quais é possível desenvolver novos trabalhos. A primeira está relacionada com a representação do conhecimento em geral e das mutações em particular no OK-BDI. A segunda está relacionada com as aplicações desenvolvidas a partir do formalismo, como o planeamento de acções.

No que diz respeito à representação do conhecimento existem decerto, dependendo de domínio para domínio, um grande número de proposições representadas no sistema cuja veracidade nunca se irá alterar ao longo do raciocínio diacrónico. Por exemplo, a informação referente ao mapa de uma casa em que um *robot* se desloca nunca se irá alterar se as únicas acções permitidas ao agente forem, por exemplo, movimentos de um local para outro. Não é, actualmente, possível separar esse conhecimento do conhecimento mutável ao longo dos varios instantes de tempo, como a posição do *robot*. Se fosse possível estabelecer a distinção entre os dois tipos de conhecimento, vários dos problemas existentes poderiam ser resolvidos ou minimizados. Um caso concreto disso ocorre quando se considera o problema

do sincronísmo. Esse problema limita de alguma forma o desempenho do sistema, dado que se restringe o sistema a apenas poder inspeccionar o mundo quando em raciocínio sincrónico. Não haveria nenhum problema em que o mundo fosse consultado no referente às características imutáveis do mesmo, se fosse possível distingui-las das demais. Se tal fosse possível, apenas se limitaria o sistema nos casos em que verdadeiramente essa limitação se justifica, o que não acontece no presente. Outra situação em que a distinção entre o conhecimento mutável e imutável se poderia revelar de grande utilidade é na escolha entre as descrições alternativas resultantes da aplicação de algumas mutações. Nalgumas das descrições obtidas, pode ocorrer que é alterada informação designada como imutável. Essas descrições podem ser eliminadas, minimizando-se o problema das descrições alternativas.

Quanto à estrutura das mutações, actualmente as suas pré-condições e efeitos apenas podem ser fórmulas literais. O OK-BDI não permite garantir, a nível formal, essa restrição. Seria interessante efectuar um estudo sobre as implicações da utilização de fórmulas não literais nas mutações.

Já no respeitante à utilização do formalismo, o facto de o sistema estar localizado no mundo permite, decerto, os mais variados tipos de aplicações. Apenas uma delas, o planeamento, foi estudada, sendo interessante o desenvolvimento de novos tipos de aplicações.

No relativo ao planeamento, os planeadores apresentados são planeadores lineares. Existe um consenso mais ou menos generalizado de que os planeadores não-lineares ou de ordem parcial são mais eficientes que os planeadores lineares. A criação de planeadores não-lineares fiéis à filosofia subjacente aos planeadores apresentados neste trabalho é, sem dúvida, um tópico de grande interesse. Um estudo da complexidade dos algoritmos apresentados seria, também de um inestimável valor para a sua depuração e caracterização.

De igual modo, planos de estruturas muito mais complexas podem ser construídos recorrendo às acções de controlo do OK-BDI. Uma análise dos planos gerados também seria útil, tendo como objectivo a sua optimização (para o caso de ramos condicionais idênticos, por exemplo).

Finalmente, os planeadores criados apresentam soluções rudimentares para casos em que vários ramos de planeamento podem surgir. Essas soluções podem ser levadas mais longe, extendendo-se os algoritmos para a criação de planos condicionais em que a incerteza não redunda apenas da natureza do formalismo usado, mas do próprio domínio. A alteração da estrutura das mutações de forma a permitir efeitos dependentes do contexto em que são aplicadas (como existe em sistemas como o `UCPOP`) permitiria a inclusão dessa informação. Os algoritmos de planeamento poderiam em seguida ser alterados de forma simples de modo a reflectir mais essa fonte de incerteza.

Bibliografia

[Anderson e Belnap 1975] A. Anderson e N. Belnap. *Entailment: The Logic of Relevance and Necessity*, volume 1. Princeton University Press, 1975.

[Barrett e Weld 1994] Anthony Barrett e Daniel S. Weld. Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence*, 67:71–112, 1994.

[Brewka e Hertzberg 1993] Gerhard Brewka e Joachim Hertzberg. How to do things with worlds: on formalizing actions and plans. *Journal of Logic and Computation*, 3(5):517–532, 1993.

[Brooks 1991] Rodney Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3), 1991.

[Chapman 1987] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987.

[Cravo e Martins 1993] M. R. Cravo e João P. Martins. SNePSwD: A newcomer to the SNePS family. *Journal of Experimental and Theoretical Artificial Intelligence*, (5):135–148, 1993.

[de Kleer 1986] J. de Kleer. An assumption-based tms. *Artificial Intelligence 28*, (2):127–162, 1986.

[Drabble e Tate 1995a] Brian Drabble e Austin Tate. O-Plan: A Situated Planning Agent. In *3rd European Workshop on Planning Systems*. 1995a.

[Drabble e Tate 1995b] Brian Drabble e Austin Tate. O-plan: A situated planning agent. In *3rd European Workshop on Planning Systems*. 1995b.

[Ernst e Newell 1969] G.W. Ernst e A. Newell. *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York, 1969.

[Fikes e Nilsson 1971] R. E. Fikes e N. J. Nilsson. Strips: a new approact to the application of theorem proving to problem solving. *Artificial Intelligence*, (2):189–208, 1971.

[Ginsberg e Smith 1988] Mathew L. Ginsberg e David E. Smith. Reasoning about Action I: A Possible Worlds Approach. *Artificial Intelligence*, 35:165–195, 1988.

[Grosse et al. 1996] Gerd Grosse, Steffen Hölldobler e Josef Schneeberger. Linear Deductive Planning. *Journal of Logic and Computation*, 6(2):233–262, 1996.

[Hanks e McDermott 1986] S. Hanks e D. McDermott. Default reasoning, default logics and the frame problem. In *Proceedings Fifth National Conference on Artificial Intellgence*, pages 328–333. Morgam Kaufmann Publishers Inc., Los Altos, CA, 1986.

[Hanks e Weld 1995] Steve Hanks e Daniel S. Weld. A Domain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.

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

[Hertzberg e Thiébaux 1994] Joachim Hertzberg e Sylvie Thiébaux. Turning an Action Formalism Into a Planner – A Case Study. *Journal of Logic and Computation*, 4(5):617–654, 1994.

[Janlert 1987] E. Janlert. Modeling change - the frame problem. In Pylyshyn, editor, *The Robot's Dilemma*, pages 401–405. Abex Publishing Corporation, 1987.

[Kambhampati 1994] Subbarao Kambhampati. Designs Tradeoffs in Partial Order (Plan Space) Planning. In *Proceedings of the Second International Conference on AI Planning Systems (AISP–94)*, pages 92–97. 1994.

[Kautz 1986] Henry Kautz. The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 401–405. 1986.

[Kripke 1971] S. Kripke. Semantical considerations on modal logic. In Linsky, editor, *Reference and Modality*, pages 63–72. Oxford University Press, 1971.

[Kumar 1989] Deepak Kumar. An Integrated Model of Acting and Inference. In *Current Trends in SNePS – Semantic Network Processing System: Proceedings of the First Annual Workshop*. 1989.

[Kumar 1993] Deepak Kumar. *From Beliefs and Goals to Intentions and Actions: An Amalgamated Model of Inference and Acting*. Ph.D. thesis, Department of Computer Science of State University of New York at Buffalo, 1993.

[Kumar et al. 1988] Deepak Kumar, Syed S. Ali e Stuart C. Shapiro. Discussing, Using and Recognizing Plans in SNePS Preliminary Report - SNACTor: An Acting System. In *Proceedings of thew Seventh Biennial Convention of South East Asia Regional Computer Conferedation*. Tata McGraw-Hill Publishing Company Limited, New Delhi - India, 1988.

[Kumar e Shapiro 1993] Deepak Kumar e Stuart C. Shapiro. Deductive efficiency, belief revision and acting. *Journal of Experimental Theoretical Artificial Intelligence (JETAI)*, 5(2&3):167–177, 1993.

[Kumar e Shapiro 1994] Deepak Kumar e Stuart C. Shapiro. The OK-BDI architecture. *International Journal of Artificial Intelligence Tools (IJAIT)*, 3(3):349–366, 1994.

[Larsen e Martins 1992] Morten N. Larsen e João P. Martins. An implementation of a planner using mutation logic. Technical report, Instituto Superior Técnico, 1992.

[Larsen et al. 1992] Morten N. Larsen, Carlos Pinto-Ferreira e João P. Martins. A planner using mutation logic. Research carried out at the Artificial Inteligence Group of Instituto Superior Técnico.

[Lin e Reiter 1997] Fangzhen Lin e Ray Reiter. How to progress a database. *Artificial Intelligence*, (92):131–167, 1997.

[McAllester e Rosenblitt 1991] D. McAllester e D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639. 1991.

[McCarthy 1958] J. McCarthy. Programs with common sense. In *Mechanisation of Thought Processes, Proceedings of the Symposioum of the National Physics Laboratory*, pages 77–84. London, UK (Her Majesty's Stationary Office, 1958), 1958.

[McCarthy 1980] J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1):27–39, 1980.

[McCarthy 1986] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[McDermott 1994] Drew McDermott. The current state of AI planning research. In *International Conference on Industrial and Engineering Aplications of AI and Expert Systems*. 1994.

[McDermott 1996] Drew McDermott. A Heuristice Estimator for Means-Ends Analisys in Planning. In *Proceedings if the Third International Conference on AI Planning Systems*. 1996.

[McDermott e Hendler 1995] Drew McDermott e James Hendler. Planning: What it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence*, (76):1–16, 1995.

[Minton et al. 1994] Steven Minton, John Bresina e Mark Drummond. Total-Order and Partial-Order Planning: A Comparative Analysis. *Journal of Artificial Intelligence Research*, 2:227–262, 1994.

[Moore 1988] R. Moore. Autoepistemic logic. In Smets, Mamdani, Dubois e Prade, editors, *Non-Standard Logics for Automated Resoning*, pages 105–136. Academic Press, 1988.

[Newell e Simon 1961] A. Newell e H. Simon. Gps: a program that simulates human thought. In *Lernende Automaten*, pages 279–293. R. Oldenbourg KG, 1961. Reprinted in Feigenbaum and Feldman 1963.

[Newell e Simon 1976] A. Newell e H. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

[Pearl 1985] Judea Pearl. *Heuristics-Intelligent Search Strategies for the Computer Problem Solving*. Artificial Intelligence. Addison-Wesley, Reading, Massachussets, 1985.

[Pednault 1994] Edwin P. D. Pednault. ADL and the State-Transition Model of Action. *Journal of Logic and Computation*, 4(5):465–512, 1994.

[Penberty e Weld 1992] J. Scott Penberty e Daniel S. Weld. UCPOP, a sound, complete, partial order planner for ADL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. 1992.

[Pinto-Ferreira e Martins 1990] Carlos Pinto-Ferreira e João P. Martins. A Formal System for Reasoning about Change. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 503–508. Pitman Publishing, London, 1990.

[Pinto-Ferreira e Martins 1992] Carlos Pinto-Ferreira e João P. Martins. The strict assumption – a propositional approach to change. In *Spring Symposium Series Working Notes*, pages 123–128. AAAI, 1992.

[Pinto-Ferreira 1991] Carlos Alberto Pinto-Ferreira. *A Lógica de Mutações*. Ph.D. thesis, Instituto Superior Técnico – Universidade Técnica de Lisboa, 1991.

[Pryor e Collins 1996] Louise Pryor e Gregg Collins. Planning for Contigencies: A Decision-based Approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996.

[Reiter 1978] R. Reiter. On reasoning by default. In *Proceedings Second Symposium on Theoretical Issues in Natural Language Processing*, pages 25–27. 1978.

[Reiter 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 1(13):81–132, 1980.

[Sacerdoti 1977] E. Sacerdoti. *A Structure for Plans and Behavior*. American Elsevier, 1977.

[Sandewall e Shoham 1994] Eric Sandewall e Yoav Shoham. *Non-monotonic Temporal Reasoning*, volume 4. Oxford University Press, 1994.

[Sandewall 1989] Erik Sandewall. Filter Preferential Entailment for the Logic of Action in Almost Continuous Worlds. In *Proceedings of the Eleventh International Joint Conference in Artificial Intelligence*, pages 894–899. 1989.

[Sandewall 1996] Erik Sandewall. Comparative assessment of update methods using static domain constraints. In *Proceedings of the International conference on Knowledge Representation and Reasoning*. 1996.

[Shapiro 1979] Stuart C. Shapiro. The SNePS Semantic Network Processing System. In Findler, editor, *Associative Networks: The Representation an dUse of Knowledge by Computers*, pages 179–203. Academic Press, New York, 1979.

[Shapiro e Rapaport 1987] Stuart C. Shapiro e W. Rapaport. SNePS considered as a fully intentional propositional semantic network. In McCalla e Cercone, editors, *The Knowledge-Frontier*, pages 262–315. Springer-Verlag, 1987.

[Shapiro e The SNePS Implementation Group 1998] Stuart C. Shapiro e The SNePS Implementation Group. *SNePS 2.4 User's Manual*. Department of Computer Science, SUNY at Buffalo, Buffalo, NY 14260, 1998.

[Shoham 1988] Yoav Shoham. Chronological Ignorance: Experiments in Nonmonotonic Temporal Reasoning. *Artificial Intelligence*, 36:279–331, 1988.

[Sussman 1975] G. J. Sussman. *A Computer Model of Skill Aquisition*. American Elsevier, 1975.

[Veloso et al. 1995] Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink e Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental Theoretical Artificial Intelligence*, 7:81–120, 1995.

[Weld 1994] Daniel S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, Winter:27–61, 1994.

[Wilkins et al. 1995] David Wilkins, Karen L. Myers, John D. Lowrance e Leonard P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental Theoretical Artificial Intelligence*, 7:121–152, 1995.