FAUSTO NOVAES CHIAPPIN VIZONI

PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL NA LÓGICA DE AÇÕES E PLANOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Alexandre Castilho

CURITIBA

2008

Agradecimentos

Primeiramente quero agradecer a minha família pelo apoio, compreensão, paciência e pela grande ajuda que me deram durante toda a minha vida. Aos meus pais que mesmo longe me incentivaram. A minha pequena irmã que não sabendo o motivo pelo qual eu estava longe e apenas visitava-a poucas vezes, sorria todas as vezes que me encontrava.

Aos meus professores e amigos da faculdade que acompanharam essa jornada. Especialmente ao Marcos Castilho que me orientou de forma esplêndida, que em poucos minutos de conversa transmitia muito conhecimento. Aos professores Fabiano e Luis Allan pelas dicas e informações que sem elas demoraria muito mais tempo para desenvolver esta pesquisa. Aos meus amigos do mestrado Nacib, Juliana e Franciele que juntos conseguimos superar vários desafios.

Aos meus amigos de longa data, Murilo, João e Hugo, que em nada contribuiram para o meu trabalho diretamente, mas que me acompanharam durante esse tempo.

Eu não posso deixar de agradecer e também dedicar este trabalho a minha namorada Sayuri, que mesmo não entendendo nada sobre o assunto corrigiu todo o texto com muita paciência e dedicação, várias e várias vezes. Também entendeu de forma amável o motivo pelo qual tive que mudar de cidade, aguentou mais as saudades do que eu.

Um sincero agradecimento. Muito obrigado a todos.

SUMÁRIO

LISTA DE FIGURAS

Resumo

Neste trabalho, buscamos mostrar que é possível executar a tarefa de planejamento de um problema que utiliza a lógica como forma de representação. A lógica é a forma mais normal de se representar um cenário, porém ela possui características que atrapalham o desenvolvimento de um planejador prático. Apresentamos um planejador baseado na lógica de ações e planos que possui a solução para o problema da persistência e o da ramificação. Utilizamos o método de *tableaux* semânticos para a lógica modal com regras de propagação e regras estruturais. Apresentamos resultados obtidos a partir de cenários da área de planejamento e mostramos o desempenho do planejador implementado.

Palavras-chave: Raciocínio sobre ações, lógica modal, lógica de ações e planos, tableau modal, planejamento.

Abstract

In this work we show that is possible to solve the plan generation task of a problem that uses a logic-based approaches to knowledge representation. Logic is the normal way to represent a scenario, but it has some problems that hinder the development of a practical planner. We present a planner based on the *logic of action and plans* which has a solution to the frame and ramification problem. We use semantic tableau for modal logic with propagation rules and structural rules. We present the results obtained for scenarios of the planning area and show the performance of our implemented planner

Keywords: Reasoning about actions, modal logics, logic of action and plans, modal tableau, plan generation.

CAPÍTULO 1

INTRODUÇÃO

Uma das áreas do estudo da Inteligência Artificial é a de raciocínio sobre ações. Nesse contexto, uma máquina para ser considerada inteligente, precisaria ser capaz de raciocinar sobre atos e suas conseqüências, levando em conta as condições para realizá-lo.

Uma das tarefas que essa máquina deve ser capaz de resolver é a de planejamento, que é encontrar uma seqüência de ações que transforma um dado estado inicial em um estado desejado.

Para representar um problema devemos encontrar um modo de formalizá-lo. Uma das principais ferramentas de base utilizada para isso é a lógica matemática. Os principais argumentos pró e contra o seu uso foram reunidos por Shanahan em [49]. Toda forma de representação pode ser reduzida à lógica formal [49], isso reforça a idéia de que a lógica é uma boa forma de representar conhecimento.

Porém, alguns problemas são inerentes ao seu uso: os problemas da persistência, da ramificação e da qualificação, explicados no capítulo 2. Devido a esses problemas, Fikes e Nilsson construíram um sistema baseado em regras que foi chamado de STRIPS [14].

O STRIPS, em 1971, surgiu como um planejador que solucionava problemas baseado em uma representação simples que realizava busca em espaço de estados, objetivando evitar os problemas da lógica. Mas a forma de representação ganhou mais fama que o planejador. Enquanto a comunidade de raciocínio sobre ações se preocupava em solucionar os problemas da representação em lógica, as pesquisas em planejamento buscavam melhorar o desempenho do processo, principalmente relativo ao tempo e à quantidade de problemas. Os algoritmos propostos depois do *STRIPS* enfrentaram o mesmo problema que este. A explosão combinatória do espaço de estados que podia ser criado inviabilizava um método prático ao tratar cenários complexos. Isto resultou em um período de

aproximadamente 20 anos sem resultados significativos.

Apenas em 1992, Kautz e Selman [29] propuseram uma tradução do problema em *STRIPS* para um problema de satisfatibilidade (SAT) e utilizou os avanços dos métodos dessa área para solucioná-lo. O resultado obtido pelo *Satplan* mostrou que é possível resolver problemas de planejamento em um tempo bem menor quando comparado aos anteriores. Três anos depois, surgiu o *Graphplan* [2], apresentado por Blum e Furst, cuja principal contribuição foi reduzir a representação do espaço de estados. Esses dois planejadores incentivaram o surgimento da competição entre planejadores [41], em 1998. Com isso, a comunidade da área de planejamento se voltou à busca por métodos que solucionam maior quantidade de problemas em menor tempo.

Todos os algoritmos recentes são baseados em *STRIPS*, não encontramos qualquer solução eficiente efetivamente implementada para planejamento baseado em lógica após o artigo original de Green [23].

O método desenvolvido por Green consistia em provar um teorema em lógica de primeira ordem pelo procedimento da resolução, para então recuperar o caminho feito pelo provador, gerando desse modo o plano. Naquela época não existia uma solução adequada para os problemas da representação em lógica, o que limitava o número de cenários que esse planejador conseguia solucionar.

Atualmente existem diversas formas de representação em lógica com soluções para esses problemas. Uma delas é chamada de $\mathcal{LAP}_{\rightsquigarrow}$ (Lógica de Ações e Planos). Formalmente desenvolvida por *Castilho et al* em [5], ela é baseada em lógica modal, é completa, adequada e possui uma solução simples para os problemas da persistência e da ramificação.

Além de possuir um grande poder de expressão, $\mathcal{LAP}_{\rightsquigarrow}$ conta com um provador de teoremas correto e completo. O método de tableau modal baseado em grafo acíclico orientado à raiz com regras de propagação e regras estruturais foi desenvolvido por *Castilho et al* [6] para provar teoremas em lógica modal. Podemos também utilizá-lo para $\mathcal{LAP}_{\rightsquigarrow}$ através de pequena adaptação (ver capítulo 5). Assim, temos uma representação em lógica e um provador de teoremas para ela.

O objetivo do presente trabalho é mostrar que podemos utilizar $\mathcal{LAP}_{\rightsquigarrow}$ como forma de representação de cenário para problemas de planejamento e também mostrar uma forma de resolvê-los efetivamente através da implementação do método de tableau. A motivação deste estudo é que queremos integrar as pesquisas em planejamento com as soluções para os problemas na área de raciocínio sobre ações.

O trabalho está organizado da seguinte maneira: Primeiro, mostraremos no capítulo 2 a área em que nos encontramos dentro da inteligência artificial. Nele poderemos conhecer os problemas que envolvem uma representação do conhecimento que utiliza a lógica. No capítulo seguinte introduziremos os conceitos da lógica modal necessários para entender os capítulos seguintes. No capítulo 4 falaremos sobre o método de tableau modal desenvolvido para provar teoremas naquela lógica. A lógica $\mathcal{LAP}_{\rightsquigarrow}$ será apresentada no capítulo 5, no qual mostraremos também o método de tableau adotado neste trabalho.

Formalizada toda a matéria que precisamos para o presente trabalho entraremos no método que desenvolvemos para procurar planos em $\mathcal{LAP}_{\rightsquigarrow}$, utilizando o tableau modal como base. Explicaremos esse processo no capítulo 6, mostrando os aspectos teóricos e a nossa implementação, assim como alguns resultados de testes que fizemos utilizando cenários da competição de planejadores [41].

CAPÍTULO 2

RACIOCÍNIO SOBRE AÇÕES

Raciocinar é usar a razão para conhecer, julgar, pensar, refletir e considerar. De acordo com o dicionário Aurélio, esses verbos são sinônimos e relacionados à inteligência. O fato de pensar sobre um ato ou julgar a sua conseqüência ou ainda conhecer o estado anterior a uma ação é trabalho relacionado ao estudo do *raciocínio sobre ações*, uma das áreas da Inteligência Artificial.

Vamos imaginar um mundo que representa uma mesa de bilhar. As 15 bolas de bilhar estão paradas no centro, ou seja, estão estáticas porque não há nenhuma força externa que as façam mudar de estado (sem considerarmos as forças da natureza, para que esse exemplo seja mais objetivo, podemos descartá-las). Agora será lançada a bola branca em direção as 15 bolas estáticas, uma ação realizada por um agente que agiu sobre outros agentes, antes estáticos, que também podem agir sobre outros agentes e assim por diante. Agora é fácil imaginar que o estado desse mundo mudou, ou seja, trazendo para a nossa mesa de bilhar, vemos que as bolas mudaram de posições.

O fato é que através de uma ação mudamos o estado de um mundo ou, também, podemos não mudá-lo, se, no caso, a bola branca não tivesse interagido com as outras bolas e voltado para a sua posição original.

As três tarefas que um agente, dito inteligente, deve ser capaz de realizar em raciocínio sobrea ações são: a predição, explicação e planejamento.

A **predição** consiste em saber se uma situação é verdadeira após a execução de uma determinada seqüência de ações ("prever o futuro"). Como exemplo: estamos em uma sala escura (*estado inicial*) e queremos saber em que estado o mundo estará se apertarmos o interruptor, nesse caso a tarefa da predição é inferir o resultado desse ato. Uma possível resposta seria que a sala se iluminará se o interruptor, no caso, tem a função de acender a luz.

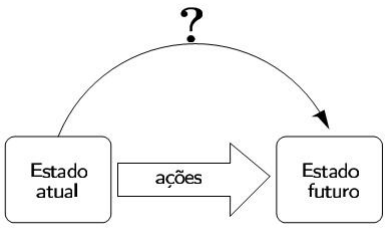


Figura 2.1: Predição: Raciocinar sobre o futuro

A **explicação**, também conhecida como **regressão**, é a tarefa de se determinar o estado do mundo *antes* da execução de uma certa seqüência de ações. Tendo o estado final e a seqüência de ações que nele resultou, o objetivo é achar o estado que era verdadeiro na situação inicial ("revelar o passado"). Por exemplo: a sala está iluminada, e o interruptor acaba de ser pressionado, sabendo que a ação de ligar o interruptor acende a luz concluímos então que a sala estava escura.

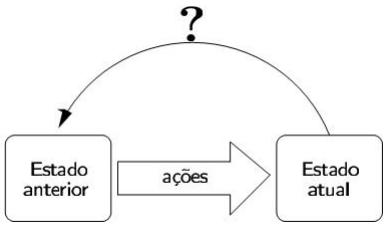


Figura 2.2: Explicação: Raciocinar sobre o passado

O **planejamento** é a tarefa de descobrir se existe uma seqüência de ações que nos leva do primeiro estado ao segundo e quais são essas ações. Em outras palavras, podemos dizer que o planejamento tem a finalidade de determinar as ações necessárias para atingir um determinado objetivo. Seguindo o exemplo, supomos que estamos numa sala escura e queremos iluminá-la, quais passos podemos fazer para atingir essa meta? Podemos acender a luz ou abrir uma janela e etc.

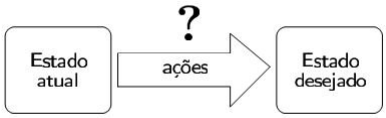


Figura 2.3: Planejamento: Os passos para atingir um objetivo

O uso da lógica matemática para a representação do conhecimento e a realização de inferências parece ser a maneira mais natural para se formalizar o conhecimento e o processo de raciocínio sobre ações [49], porém 3 problemas são inerentes ao uso da lógica como formalismo de base: o *problema da persistência*; *o problema da ramificação* e o *problema da qualificação*.

2.1 Problema da persistência

O problema da persistência foi apontado por John McCarthy e Patrick J. Hayes em [39] e consiste no fato de que ao representar um mundo dinâmico usando a lógica formal, precisamos, além de representar todos os objetos que mudam em função da execução das ações consideradas, também explicitar todos os objetos que não mudam pela ocorrência dessas ações.

Por mais evidente que isso seja para nós humanos, em lógica não há como fazer inferência sobre os fatos inalterados tomando por base apenas as informações das coisas que alteraram de estado decorrente da execução de uma ação. Assim, verifica-se a necessidade de inclusão de expressões que garantam as não mudanças de estado dos objetos, chamadas de *axiomas de persistência*[1]. Devemos notar que os objetos mudam ou não de estado para cada ação executada. Tomando uma ação que não modifica nenhum estado em um mundo, devemos então explicitar que a execução dessa ação não altera nenhum fato.

O grande problema é que a lista de todas as não mudanças pode ser extremamente longa. Isso significa que em domínios mais complexos, a quantidade de informações a serem codificadas pode incapacitar o processo de inferência.

Segundo o trabalho de McCarthy e Hayes [39], para um domínio contendo n objetos e m ações, o número de expressões que devemos fazer para dizer as não mudanças do mundo gira em torno de $n \times m$. A maioria dos cenários-exemplo encontrados na literatura apóia essa conjectura. Esse valor evidencia a quantidade de axiomas de per-

[1]No original, em inglês, essas expressões são conhecidas como *frame axioms*, isto é relacionado com quadros de filmes, onde os objetos não mudam muito de um quadro para outro dando a idéia de "inércia" ou "persistência", por causa disso muitos trabalhos em português adotam a tradução "axiomas de quadro"

sistência que devemos citar para demonstrar um cenário, o que pode se tornar um grande empecilho com um número considerável de objetos.

Uma maneira interessante de se tentar evitar descrever todos os axiomas de persistência necessários para um dado domínio é considerar apenas as leis de efeito das ações e supor que todo o restante permanece inalterado. Essa técnica constitui a chamada *hipótese da inércia*. Ela supõe que uma ação não altera o estado de um objeto a não ser que isso seja explícito. O problema é como integrar isto formalmente na lógica adotada.

A hipótese da inércia foi utilizada pela primeira vez por McCarthy dentro do contexto da *circunscrição* [38]. A circunscrição consiste numa forma de raciocínio não-monotônico baseada em técnicas de *minimização de modelos*. Ao invés de escrever todos os axiomas de persistência para um dado domínio, devemos adicionar às leis de execução (efeito) das ações uma fórmula que codifica a idéia de que a ação não afeta um objeto em uma determinada situação. O princípio de ação, objeto e situação, assim como a fórmula de circunscrição podem ser vistos em [39, 38, 49]. McCarthy criou um formalismo para expressar um cenário em raciocínio sobre ações, chamado de *cálculo de situações* [38] e desde o seu surgimento, em 1969, diferentes variantes vêm sendo amplamente utilizadas.

A circunscrição causou um grande impacto nas pesquisas da área, pois possibilitou reduzir a uma única fórmula todo o conjunto de enunciados a respeito das não-mudanças do mundo, porém a utilização da circunscrição tem o inconveniente de requerer o enunciado e conseqüêntemente a prova de teoremas de fórmulas em lógica de segunda ordem. Dessa forma, além de aumentar a complexidade das soluções do problema da persistência, ainda dificulta a implementação de provadores de teoremas circunscritivos, uma vez que a lógica de segunda ordem é indecidível, além de cara e incompleta do ponto de vista computacional.

2.2 Problema da ramificação

O problema da ramificação foi identificado por Joseph J. Finger em 1987 [15]. Surge da necessidade de descrever todas as consequências, tanto os efeitos diretos como os indiretos, da execução de uma ação. A dificuldade aqui encontrada é explicitar todos

as coisas que mudam no decorrer do tempo e da execução das ações.

Um exemplo que pode mostrar bem esse problema é o modo como representamos o seguinte cenário: imagine uma televisão sobre uma mesa. Os dois estão inertes até que alguém move a televisão de lugar, nesse caso a televisão muda de estado, mas a mesa não. Então devemos dizer que ao executar a ação de *mover a televisão* a mesa não sofre alteração de estado, mas e quando movemos a mesa? Como a televisão está sobre a mesa os dois sofrem a ação de *mover*, mas a televisão sofre a ação devido ao efeito indireto de mover a mesa.

A solução é achar um modo de representar o fato de que *mover* a mesa também provoca o movimento da televisão mas, ao *mover* esta, aquela não sofre alteração. De fato a dificuldade para se tratar o problema da ramificação surgiu por causa da maneira como ele era tratado. Um consenso na comunidade de raciocínio sobre ações sempre foi que a solução para esse problema consistia em um conjunto de axiomas de restrição de domínio e os efeito indiretos poderiam ser inferidos por meio delas.

Porém surgiram contra-exemplos que mostraram que a simples utilização de leis de domínio não é suficiente para se obter todas as conseqüências de determinadas ações [35]. Assim, é necessário enriquecer o formalismo para aumentar o seu poder de expressão.

Uma forma de atacar o problema da ramificação consiste na formalização de que as mudanças devem ser explicitamente causadas (*abordagens causais*) [8]. Com isso, o objetivo passou a ser procurar maneiras de se expressar uma noção de causalidade envolvendo efeitos de ação em lógica formal. A forma de tratar essa noção de causa passou a ser o centro das pesquisas da comunidade desde então.

2.3 Problema da qualificação

O problema da qualificação, apontado em 1977 [37] por McCarthy, é determinar todas as pré-condições necessárias para que uma dada ação possa ser executada com sucesso. Porém o número de condições que precisam ser satisfeitas para cada ação pode ser imensa, dificultando a descrição do domínio e a criação de cenários.

Um exemplo conhecido do problema da qualificação é o que se segue: Uma pré-condição para se ligar um carro é ter a chave ligada na ignição, mas também existem muitos outros. Assim, deve ter gasolina no tanque, a bateria tem que estar conectada e carregada e etc. Além de existir pré-condições inimagináveis normalmente como o carro tem que ter um motor, um caburador e não ter uma batata no escapamento.

Dentre os três problemas em raciocínio sobre ações, o da qualificação é o de maior complexidade pelo fato de ainda não estar bem definido o que são efetivamente todas as pré-condições relevantes de uma ação e quais regras de pré-condições são necessárias para expressar um cenário.

Uma sugestão é que toda a qualificação necessária para uma ação ser executada deve estar agrupada em sua pré-condição. Assumimos que nenhuma ação pode ser executada de início para cada situação[2]. A vantagem desse método é que o sistema não precisa das qualificações. Para que uma ação possa ocorrer devemos explicitar o fato [19].

Porém ainda assim temos vários problemas com relação à qualificação das ações. Imagine o cenário do *mundo dos blocos*[3] com apenas duas ações; *empilhar* e *desempilhar*. Para *empilhar* um bloco sobre outro precisamos que os dois blocos estejam livres, ou seja, que não possuam nenhum blocos sobre eles. É necessário explicitar uma regra que diz que só pode empilhar um bloco sobre outro se os dois estiverem livres. Porém nada falamos sobre as outras qualificações, como por exemplo, um bloco poderia estar pregado na mesa o que imperia este ser empilhado sobre outro.

Outro problema é distinguir legítimas qualificações de uma ação das possíveis ramificações delas. Utilizando do exemplo do mundo dos blocos e adicionando uma nova ação, chamaremos de *pregar*, com o objetivo de pregar um bloco na mesa. Se executarmos essa ação em um bloco, prendendo-o na mesa, este, então, perde a possibilidade de ser empilhado sobre outro. O efeito causado foi uma ramificação da ação *pregar*.

Uma solução para esse poblema, sugerida por Matthew L. Ginsberg e David E. Smith [19] é separar as consequências de cada ação em cada situação. Uma situação é

[2]Uma situação, nesse caso, se refere ao estado inicial ou ao estado resultante do efeito de uma ação.

[3]O mundo dos blocos, ou *BlockWorld*, consiste de um cenário clássico em inteligência artificial onde temos um conjunto de blocos colocados sobre uma mesa e que podem ser empilhados uns sobre os outros. Temos também uma garra que pode manipular esses blocos.

o estado (valores) dos objetos do cenário em um dado momento. Com isso se conseguiria qualificar as ramificações.

2.4 Considerações

No final da década de 60, a solução para o problema da persistência consistia basicamente no formalismo da circunscrição. Porém surgiram contra-exemplos que mostram a dificuldade em modelar um cenário nesse formalismo, como é o caso do cenário do tiro em Yale [24] que consiste em uma situação que envolve uma ação sem efeito. A utilização da política de circunscrição nesse exemplo não produziu o resultado esperado [24].

Ao longo da década de 90, os pesquisadores se lançaram em uma busca intensa para melhorar os métodos de circunscrição e ao mesmo tempo, vários outros formalismos foram apresentados como substitutos ideais para o cálculo de situações. Dentre os mais importantes estão a linguagem $\mathcal{A}$ [18], o formalismo de *Features and Fluents* [48], o *cálculo de fluentes* [53] e o *cálculo de eventos* [49]. Esses foram adaptados para suportar técnicas de minimização[4] similares às da circunscrição ou até mesmo ela própria.

Durante essa época, pode-se perceber uma separação entre os pesquisadores que procuraram ainda insistir em uma abordagem não-monotônica e aqueles que seguiram uma alternativa monotônica para tentar solucionar os problemas da persistência e da ramificação.

A diferença entre as duas abordagens consiste que em uma lógica monotônica podemos afirmar que uma hipótese tida como conseqüência lógica de um conjunto de axiomas e regras de inferência continuará a ser válida mesmo depois de adicionar qualquer novo axioma. Ao contrário das lógicas não monotônicas, onde inferimos fatos que eventualmente podemos descobrir não serem mais válidos depois da adição de novas informações.

Estes problemas levaram a comunidade de planejamento a abandonar a lógica por outro formalismo. Um exemplo é o STRIPS [14] que realiza busca em espaço de esta-

[4]Em geral, o objetivo dessas técnicas é reduzir ao máximo o conjunto de efeitos causados por uma ação, um outro exemplo de técnicas de minimização é a "suposição do mundo fechado" que é a presunção de que aquilo que não é conhecido como verdadeiro é falso.

dos. Em 1971 esse formalismo ganhou fama por apresentar uma forma de representação simples e independente do planejador, entretanto ele não é uma lógica e consequentemente não busca solucionar os problemas apresentados nesse capítulo.

No presente trabalho queremos resgatar o uso da lógica para a tarefa de planejamento. Usaremos como base $\mathcal{LAP}_{\rightsquigarrow}$, que mostraremos no capítulo 5. Antes precisamos comentar sobre a lógica modal (capítulo 3) e tableau modal (capítulo 4), o primeiro é a lógica em que $\mathcal{LAP}_{\rightsquigarrow}$ é baseada e o segundo é um provador de teoremas para ela.

CAPÍTULO 3

LÓGICA MODAL

Nesse capítulo introduzimos brevemente a lógica modal por ser a base do formalismo utilizado no presente trabalho. Primeiro precisamos entender a sintaxe e semântica dessa lógica para nos aprofundarmos na linguagem de ações e planos ($\mathcal{LAP}$). Textos mais completos sobre lógica modal são [9] e [43].

A lógica modal mais próxima da que se conhece hoje surgiu no começo do século 20 com Clarence Irving Lewis. Ele criou vários sistemas axiomáticos com operadores modais [33] conhecidos como $S1$, $S2$, $S3$, $S4$, $S5$. Ele foi o primeiro a propor um sistema simbólico que estendia a lógica clássica com operadores modais.

Lewis observou que o significado algébrico, como usado no Princípio Matemático de Russel e Whitehead, conduz para o chamado "paradoxos da implicação material". Uma proposição falsa implica em qualquer proposição enquanto que uma proposição verdadeira é inferida de qualquer valor de proposição, pela regra da implicação [21].

Veja as formas simbólicas abaixo:

$$\neg\alpha \rightarrow (\alpha \rightarrow \beta)$$

$$\alpha \rightarrow (\beta \rightarrow \alpha)$$

O que Lewis queria era poder inferir um valor real de β da fórmula α implica β. Tendo a fórmula "não α ou β" como sinônimo, ele distinguiu o significado da disjunção de dois modos. *Extensional dijunction* que é o significado usual do operador "ou" e *Intensional dijunction*, que diz que pelo menos uma das proposições seja "necessariamente" verdade.

Essa última originou a implicação de Lewis, que ele apelidou de "Implicação

estrita"[1], que significa "é impossível (ou logicamente inconcebível) que α seja verdade e β seja falso".

No início Lewis utilizou um símbolo modal chamado "Impossível" para definir a implicação estrita. Posteriormente se tornou o sistema $S3$, onde transformou o símbolo "Impossível que" para o hoje conhecido $\Diamond$ (possibilidade) [34].

O sistema de Lewis contém toda verdade funcional tautológica como teorema, mas isso requer uma extensa análise para demonstrá-la [11]. Tal esforço seria desnecessário se o sistema fosse definido como uma extensão direta da base do cálculo proposicional.

Kurt Gödel, em 1933, formalizou asseções de *provabilidade* usando um conectivo proposicional chamado de B. Onde $B\alpha$, em lógica proposicional, significa que "α é *provável*". Ele definiu, em adição aos axiomas e regras do cálculo proposicional básico, os axiomas:

$$Bp \rightarrow p$$

$$Bp \rightarrow (B(p \rightarrow q) \rightarrow Bq)$$

$$Bp \rightarrow BBp$$

e a regra de inferência: *de* α *inferir* $B\alpha$. Esse sistema de Gödel era parecido com o sistema $S4$ de Lewis quando substituído $B\alpha$ por $\Box\alpha$.

Ele ainda transpôs vários teoremas do cálculo proposicional para o seu sistema [20], o que posteriormente contribuiu para o desenvolvimento da semântica de Kripke. Atualmente a lógica modal é apresentada no estilo axiomático de Gödel.

A notação de *lógica* refere para qualquer conjunto Λ de fórmulas que inclui todas funções tautológicas e é fechado sobre a regra de substituição uniforme de variáveis e desprendimento da implicação material[2] [21].

As fórmulas que pertencem a Λ são Λ-*teoremas* e também são ditas Λ-*prováveis*.

[1] no original "Strict Implication"

[2] **Paradoxo da implicação material**. Paradoxo que decorre da definição de implicação material pela qual uma proposição falsa implica proposições verdadeiras ou proposições falsas e proposições verdadeiras podem ser implicadas por proposições verdadeiras ou proposições falsas.

Uma lógica modal é chamada de *normal* se possui o segundo axioma de Gödel.

$$\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$$

Onde $\Box$ substituiu B, e tem a regra da *necessidade*: *de* α *inferir* $\Box\alpha$.

Outros autores como Feys, Carnap e Lemmon também desenvolveram estudos sobre o tema. Lemmon trabalhou com o formalismo da lógica modal *não normal* [32]. Essa época é conhecida como a era sintática, porque ainda não existia uma semântica adequada para a lógica modal que surgiu em 1959 com os trabalho de Saul Kripke.

Na próxima seção falaremos sobre a notação da linguagem da lógica modal depois introduziremos a semântica de Kripke e a relação de acessibilidade. Mostraremos também alguns axiomas mais comuns dessa lógica, quando falaremos dos *sistemas* da lógica modal. Por fim discutiremos sobre a lógica multimodal que é fundamental para o entendimento da linguagem de ação e planos $\mathcal{LAP}$ que será vista no capítulo 5.

3.1 Definição da linguagem

Nós mostraremos a lógica modal no contexto da linguagem da *necessidade* e da *possibilidade*. Porém, os operadores modais podem possuir outros conceitos, dentre eles estão:

- A crença, conhecida como a lógica doxástica, utiliza o símbolo Bx para dizer que "x *acredita que*".

- O tempo, que é o caso da lógica temporal. Temos dois operadores básicos: o G para representar o futuro e o H, o passado. Lemos G como "***sempre*** *será o caso que*" e o operador H como "***sempre*** *foi o caso que*".

  Temos ainda um operador F, FA, sendo A uma fórmula lógica qualquer, é equivalente a dizer que $\neg G \neg A$. Lemos o operador F como "*será o caso que*".

- A obrigação, lógica deôntica. Temos um operador O que representa "*é obrigado que*". Também temos dois operadores, P e F que representam "*é permitido que*" e

"*é proibido que*"[3] respectivamente, que são derivados do operador O. Utilizando de uma fórmula lógica A qualquer, temos $PA = \neg O \neg A$ e $FA = O \neg A$.

A lógica modal é constituída por um conjunto de *sentenças atômicas*, que são as fórmulas lógicas mais simples:

$$A_0, A_1, A_2...$$

e por fórmulas não atômicas, que são formadas por um ou mais *operadores sintáticos*:

$$\top, \bot, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \Box, \Diamond$$

$\top$ e $\bot$ são constantes (verdadeiro/falso); $\neg$, $\Box$ e $\Diamond$ são operadores unários e $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$ são operadores binários.

O conjunto de sentenças pode ser definido como a seguir:

Definição 3.1.1 *(Sentenças)*

1. *A_n é uma sentença, para n = 0, 1, 2, ...*
2. *$\top$ e $\bot$ são sentenças.*
3. *$\neg A$ é uma sentença sse A for uma sentença.*
4. *$A \wedge B$ é uma sentença sse A for uma sentença e B for uma sentença.*
5. *$A \vee B$ é uma sentença sse A for uma sentença ou B for uma sentença.*
6. *$A \rightarrow B$ e $A \leftrightarrow B$ são sentenças sse A for uma sentença e B for uma sentença.*
7. *$\Box A$ e $\Diamond A$ são sentenças sse A for uma sentença.*

A sentença $\top$ é chamada de *verum* ou constante verdadeira, porque ela é sempre verdade, ao contrário de $\bot$ chamada de *falso* ou constante falsa, por representar o valor falso. As sentenças da disjunção ($\vee$), da condição ($\rightarrow$) e da bicondição ($\leftrightarrow$) podem ser abreviadas pelos operadores de conjunção ($\wedge$) e negação ($\neg$).

[3] o operador F vem do inglês *forbidden*

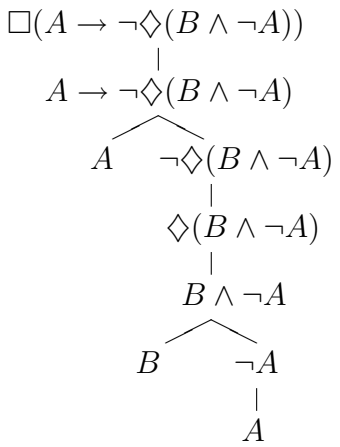
A tabela abaixo mostra a abreviação dessas sentenças:

Tabela 3.1.2 *Abreviações*

Sentenças	Abreviações
$A \vee B$	$\neg(\neg A \wedge \neg B)$
$A \rightarrow B$	$\neg(A \wedge \neg B)$
$A \leftrightarrow B$	$\neg(A \wedge \neg B) \wedge \neg(B \wedge \neg A)$

A sentença $\Diamond A$, "*é possível* A", pode ser abrevidado por $\neg\Box\neg A$.

Antes de falarmos sobre a semântica da lógica modal, vamos apenas comentar sobre as sub-fórmulas, também chamadas de sub-sentenças. Vamos utilizar, por exemplo, a fórmula $\Box(A \rightarrow \neg\Diamond(B \wedge \neg A))$ para mostrar a construção de uma árvore com as respectivas sub-fórmulas daquela.



O objetivo de demonstrar a árvore de sub-sentenças é porque queremos demonstrar como uma fórmula é descrita no nosso algoritmo. Veremos no capítulo 6 como as fórmulas são organizadas utilizando esse princípio.

3.2 Semântica da lógica modal

Ao trabalhar com o conceito de necessidade e possibilidade, utilizamos um objeto abstrato de análise chamado *mundo* para representar um conjunto de fórmulas lógicas.

A lógica modal possui os operadores $\Box$ para representar a *necessidade* e $\Diamond$ a *possibilidade*. *A é necessário* significa dizer que A é verdade em todos os *mundos possíveis*

e dizer que A *é possível* significa que A é verdade em algum *mundo possível*.

Estamos agora trabalhando com diversos mundos, onde uma fórmula pode possuir valores diferenciados em cada mundo e ainda assim o cenário ser válido. Por exemplo a figura 3.1, que possui a fórmula $\neg B$ em w_1 e B em w_2.

Vamos dar um exemplo para entender melhor esse conceito:

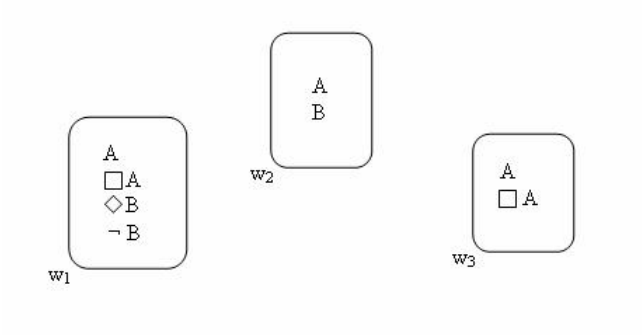


Figura 3.1: Exemplos de mundos possíveis

Temos três mundos possíveis nesse exemplo, w_1, w_2 e w_3. As fórmulas A e $\neg B$ estão presente no mundo w_1. As proposições A e B em w_2 e, em w_3 temos A.

Observe que no mundo w_1 e w_3 temos a fórmula $\Box A$, ou seja, a proposição A é necessária. A tem que ser verdade em todos os mundos possíveis para $\Box A$ ser válida. Como temos três mundos possíveis e A está presente em todos eles então $\Box A$ é verdadeira.

Com a fórmula $\diamondsuit B$ precisamos apenas encontrar um mundo possível em que B seja verdadeiro para que essa fórmula seja verdadeira. No caso temos o mundo w_2. Repare que se tivéssemos $\Box B$ em algum mundo, esta fórmula não seria válida, pois não temos B como verdade no mundo w_1 nem no mundo w_3.

Nos vemos, então, diante de vários mundos possíveis, incluindo o mundo corrente, onde as fórmulas possuem valores verdadeiros ou falsos. Desse modo, precisamos delimitar e organizar para refletir um cenário desejado. Isso é feito através de um *modelo*.

Definição 3.2.1 ***Modelo*** *Um modelo M é um par (W, V) onde W é um conjunto não vazio de mundos possíveis e V é uma função de valoração tal que V: $Formula \rightarrow P(w)$, onde $Formula$ é uma sentença lógica que pertence ao conjunto FLM e $P(w)$ é o subconjunto de mundos possíveis de W onde $Formula$ é verdadeiro. $w \in W$.*

Vamos mostrar um exemplo para a definição anterior utilizando da figura 3.1:

Exemplo 3.2.2 *Temos* $W = \{w_1, w_2, w_3\}$ *e o conjunto de fórmulas lógicas, chamaremos de* FLM*, como* $FLM = \{A, B, \neg B, \Box A, \Diamond B\}$*; E temos a seguinte valoração:* $V(A) = \{w_1, w_2, w_3\}$*,* $V(B) = \{w_2\}$*,* $V(\neg B) = \{w_1\}$*,* $V(\Box A) = \{w_1, w_3\}$ *e* $V(\Diamond B) = \{w_1\}$*.*

Para representar se uma sentença é consequência semântica de um conjunto de premissas de um modelo M usamos o símbolo $\models^M A$, isso significa que não existe uma interpretação que faz com que todos os membros de M sejam verdadeiros e A seja falso. Quando a valoração de uma fórmula A for verdadeiro em um mundo w pertencente ao modelo M expressamos como $\models^M_w A$.

Vamos, agora, definir as condições de verdade utilizando a noção de modalidade.

Definição 3.2.3 *Seja* w *um mundo pertencente ao modelo* M *e* V *uma função de valoração.*

1. $\models^M_w A$ *sse* $w \in V(A)$
2. $\models^M_w \top$
3. $\models^M_w \neg A$ *sse não é verdade que* $\models^M_w A$
4. $\models^M_w A \wedge B$ *sse* $\models^M_w A$ *e* $\models^M_w B$
5. $\models^M_w \Box A$ *sse para todo* $w' \in M$, $\models^M_{w'} A$*. Por exemplo, na figura 3.1,* $\models^M_{w_2} \Box A$ *mas* $\Box A \notin w_2$*.*
6. *não é verdade que* $\models^M_w \bot$
7. $\models^M_w A \vee B$ *sse ou* $\models^M_w A$ *ou* $\models^M_w B$
8. $\models^M_w A \rightarrow B$ *sse se* $\models^M_w A$ *então* $\models^M_w B$
9. $\models^M_w A \leftrightarrow B$*, sse (se* $\models^M_w A$ *sse* $\models^M_w B$*)*
10. $\models^M_w \Diamond A$ *sse para algum* $w' \in M$, $\models^M_{w'} A$

Quando acrescentamos uma valoração (verdadeiro/falso) para os membros de um modelo M e, se $A \in M$ e A for verdadeiro em todos os mundos pertencente ao modelo, dizemos que a fórmula A é uma *conseqüência semântica* do conjunto de fórmulas do modelo M e representamos por $\models^M A$, se esta fórmula for verdade em todos os modelos então representamos por $\models A$ e podemos afirmar que A é uma tautologia.

Definição 3.2.4 $\models^M A$ *sse para todo mundo* $w \in M$, $\models^M_w A$

Definição 3.2.5 $\models A$ *sse para todo modelo* M *e todo mundo* $w \in M$, $\models^M_w A$

O nosso objetivo é poder distinguir as sentenças que são válidas das que não são, com a noção de validade ficou mais fácil fazer essa verificação. Vamos dar um exemplo de duas fórmulas lógicas $\Box A \rightarrow A$ e $A \rightarrow \Box A$.

Exemplo 3.2.6 *Da fórmula* $\Box A \rightarrow A$ *podemos concluir pela definição 3.2.3(8), se* $\Box A$ *então A, e por 3.2.3(5) se A for verdadeira em todo mundo possível do modelo então* $\Box A$ *é verdade. O que torna a sentença* $\Box A \rightarrow A$ *válida, ou podemos dizer que* $\models \Box A \rightarrow A$.

Exemplo 3.2.7 *A fórmula* $A \rightarrow \Box A$ *já não podemos dizer que é válida para todos os modelos, ou que* $\not\models A \rightarrow \Box A$. *Pois se temos em um modelo um conjunto de mundos* $W = \{w_1, w_2\}$ *onde* $V(A) = w_1$ *e* $V(\Box A) = w_1$, *podemos afirmar que a fórmula A é verdade no mundo* w_1 *mas não em* w_2 *o que torna* $\Box A$ *inválida.* $\models^M_{w_1} \Box A$ *deve implicar que* $\models^M_{w_1} A$ *e* $\models^M_{w_2} A$.

A seguir nós explicaremos a *relação de acessibilidade* proposta por Saul Kripke em 1959. O que tornou a lógica modal mais interessante porque agora vamos representar relações entre os mundos pertencente a um modelo.

3.2.1 Relações de acessibilidade

A semântica relacional, também conhecida como a semântica de Kripke, foi sugerida por Saul Kripke em 1959 inicialmente para a lógica modal, mas depois foi adaptada para outras lógicas não clássicas.

A descoberta da semântica de Kripke foi um marco nas lógicas não clássicas, porque antes não existia um *modelo teórico*[4] para para essas lógicas.

Em 1959, Kripke aprimorou a idéia inicial de Leibniz propondo que o conceito inicial de "possivelmente verdade" fosse modificado "verdade em algum mundo possível" ou "verdade em algum mundo acessível". Kripke propôs uma generalização de sua semântica que representa a relação entre os mundos possíveis. Ficou conhecida como a "*relação de acessibilidade*", traduziu os "mundos possíveis" como sendo todos os mundos "acessíveis" a partir do mundo atual [31].

A *relação de acessibilidade* é uma estrutura que relaciona os mundos entre si, é uma relação binária incluída na definição de modelo da semântica da lógica modal.

Definição 3.2.8 *$M = (W, R, V)$ é um modelo sse:*

1. *W é um conjunto de mundo possíveis*
2. *R é uma relação binária em W. $R \subseteq (W \times W)$*
3. *V é a função de valoração, tal que V: $Formula \rightarrow P(w)$, onde $Formula$ é uma sentença e $P(w)$ é o sub-conjunto de mundos possíveis de W onde $Formula$ é verdadeiro.*

Podemos interpretar $\alpha R \beta$, sendo α e β dois mundos possíveis, como β é acessível, ou alcançável por α. Devido à relação de acessibilidade teremos que rever o conceito de validade dada na definição 3.2.3 que envolvem os operadores modais (as fórmulas não modais continuam sem alteração).

Definição 3.2.9 *Seja w um mundo no modelo $M = (W, R, V)$*

- $\models^M_w \Box A$ *sse para cada* $w' \in W$ *tal que* wRw', $\models^M_{w'} A$
- $\models^M_w \Diamond A$ *sse para algum* $w' \in W$ *tal que* wRw', $\models^M_{w'} A$

[4]Um modelo teórico é o estudo de estruturas matemáticas como grupos e modelos de um conjunto de teorias usando artifícios da lógica matemática.

Daremos um exemplo para entender melhor esse conceito, ele é parecido com o exemplo da figura 3.1, mas agora relacionamos os mundos de forma que: Seja um modelo $M = (W, R, V)$, onde $W = \{w_1, w_2, w_3\}$; $V(A) = \{w_1, w_2, w_3\}$, $V(\Box A) = \{w_1, w_3\}$, $V(\Diamond B) = \{w_1\}$, $V(\neg B) = \{w_1\}$ e $V(B) = \{w_2\}$; E o conjunto da relação de acessibilidade é $w_1 R w_2$, $w_1 R w_3$ e $w_3 R w_1$. Veja a figura a seguir.

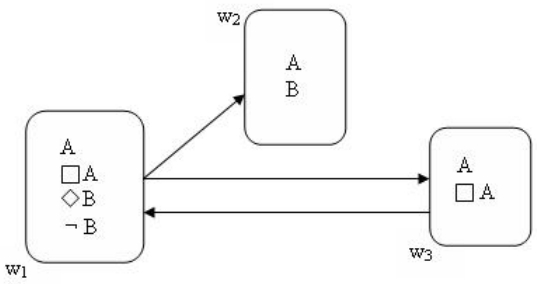


Figura 3.2: Exemplos de mundos possíveis com suas respectivas relações de acessibilidade

A fórmula $\Box A$ é válida no mundo w_1 pois em todos os mundos relacionados a w_1, que são w_2 e w_3, a fórmula A é verdade. Assim como no w_3 é válida a fórmula $\Box A$ pois A é verdade em w_1 que é acessível por w_3.

Podemos observar que não existe a relação $w_1 R w_1$, isso significa que mesmo se w_1 não possuísse a fórmula A, $\Box A$ no mundo w_1 continuaria sendo válida, porém no mundo w_3 a fórmula $\Box A$ perderia sua validade.

$\Diamond B$ é válida em w_1 pois existe um mundo possível que é acessível por w_1, nesse caso o mundo w_2, onde B é verdade.

Com os conceitos de mundos possíveis e da relação de acessibilidade é fácil entender os axiomas da lógica modal.

3.3 Sistemas da lógica modal e suas características

Nessa seção comentaremos sobre a classe da lógica modal conhecida como *normal*. É dito *normal* o sistema (as vezes chamado só de lógica) desenvolvido para a lógica modal que usa o axioma K (K em homenagem a Kripke) como base.

A lógica modal normal mais fraca é simplesmente o cálculo proposicional mais o operador $\Box$, a *regra da necessidade* e o axioma K, como veremos na definição 3.3.1.

Definição 3.3.1 *Lógica modal normal é um conjunto L de fórmulas modais tal que L contém:*

- *Todas as tautologias proposicionais;*
- *O axioma K:* $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$*;*

  E é fechado sobre as regras:

- *Da necessidade (chamada de regra N ou RN): Se p é um teorema então* $\Box p$ *também é. Seja A uma fórmula lógica e A é uma tautologia então* $\Box A$ *é válida.*
- *A regra modus ponens ou MP: Se* $A \rightarrow B$*, A é um teorema válido então inferimos B.*

A regra da necessidade diz que se algo pode ser provado usando apenas as regras lógicas simples então ele é necessário. Se $\models A$ então $\models \Box A$. Não confunda a RN com o exemplo 3.2.7, onde $\not\models A \rightarrow \Box A$.

Um sistema modal é formado por um conjunto de axiomas da lógica modal, por exemplo, o sistema K é formado apenas pelo axioma K e os da lógica proposicional.

O conceito de necessidade que é consistente com as regras do sistema K é dado por duas características: O que pode ser provado sem pressupostos é necessário (regra RN) e o que é necessariamente verdade também é necessário (axioma K) [5].

Em lógica modal normal nomeamos os sistemas da forma $KA_1A_2...A_n$, onde K é o axioma que define o sistema como normal e A_x $(1 \leq x \leq n)$ são outros axiomas da lógica modal. Porém existem alguns sistemas que são conhecidos na literatura com outros nomes, é o caso de $S5$, representado pelos axiomas K, T e 5 e o sistema $S4$ formado por K, T e 4.

[5] A fórmula do axioma K, $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$, expressa o princípio da distribuidade da necessidade. Isso significa que se uma condição e seu antecedente são ambos necessários então o conseqüente também é.

Para a validade do axioma K suponha que w é um mundo possível no modelo M tal que ambos $\Box(A \rightarrow B)$ e $\Box A$ sejam válidos em w. Então para todo mundo possível $w' \in M$, ambos $A \rightarrow B$ e A são válidos em w' disso segue que para cada mundo possível $w' \in M$ temos $\models_{w'} B$. Assim, se B é válido em todo mundo possível $w' \in M$ então concluímos que $\models_{w'} \Box B$.

Existem vários axiomas que refletem algumas características peculiares, veremos alguns deles. Principalmente os axiomas que compõe a linguagem de ação e planos (veja o capítulo 5).

Um desses axiomas é o T $(\Box A \rightarrow A)$, que exprime a idéia que se algo é necessário então ele deve ser válido em todos os mundos possíveis inclusive no próprio mundo. Esse axioma reflete a característica de reflexividade.

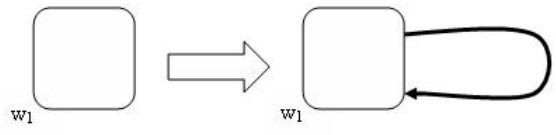


Figura 3.3: Propriedade de reflexividade

O axioma K sozinho, não possui essa característica. Se adicionarmos a propriedade da reflexividade à K, resultando em KT, podemos observar que em todos esses modelos a fórmula $\Box A \rightarrow A$ será válida.

Exemplo 3.3.2 *Seja o modelo $M = (W, R, V)$ onde $W = \{w_1, w_2, ..., w_n\}$ e $w_x R w_x$ para todo $1 \leqslant x \leqslant n$ ou seja, M é um modelo onde as relações de acessibilidade são apenas reflexivas.*

Vamos abreviar a fórmula $\Box A \rightarrow A$ para $\neg(\Box A \wedge \neg A)$ e provaremos por contradição que essa fórmula é válida em todos os modelos onde exista a propriedade de reflexividade.

$\not\models \neg(\Box A \wedge \neg A)$

$\neg \models \neg(\Box A \wedge \neg A)$

$\neg(\Box A \wedge \neg A)$ não é válido o que implica que $(\Box A \wedge \neg A)$ é satisfatível então existe uma valoração onde $(\Box A \wedge \neg A)$ seja verdade.

se $\Box A$ é verdade para todo $w' \in W$ tal que wRw', então concluimos que A é verdade

em todo $w' \in W$ tal que wRw'. Mas como $w' = w$ e temos uma conjunção com $\neg A$ em w então concluímos que não existe um modelo que possua a característica da reflexividade onde a fórmula $\Box A \rightarrow A$ não seja válida.

O próximo é chamado de 4 ($\Box A \rightarrow \Box\Box A$), que representa a característica da transitividade. Esse axioma diz que se algo é necessário então é necessariamente necessário.

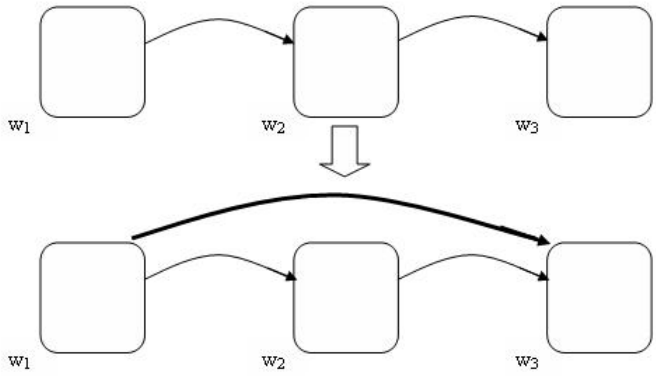


Figura 3.4: Propriedade de transitividade

Exemplo 3.3.3 *Suponha que temos um modelo $M = (W, R, V)$ onde $W = \{w_1, w_2, w_3\}$ e se w_1Rw_2 e w_2Rw_3 então pela transitividade teremos a relação w_1Rw_3.*

Temos a fórmula $\Box A \rightarrow \Box\Box A$ e $\models^M_{w_1} \Box A$ então $\models^M_{w'} A$ para todo $w' \in M$ onde w_1Rw'. Desse modo temos que $\models^M_{w_2} \Box A$ seja verdade devido à relação w_2Rw_3. Também temos que $\models^M_{w_3} \Box A$[6].

Agora é só observar a fórmula do axioma K, $\Box(A \rightarrow \Box A) \rightarrow (\Box A \rightarrow \Box\Box A)$, para $\models^M_{w_1} \Box(A \rightarrow \Box A)$ ser válida é necessário que $A \rightarrow \Box A$ seja válido em todos os mundos acessíveis de w_1, como vimos que A e $\Box A$ é válido em w_2 e em w_3 então concluímos que $\Box A \rightarrow \Box\Box A$ é um teorema.

Em $K4$ provamos que $\Box\Box A$ é equivalente a sentença $\Box A$. Como resultado qualquer seqüência de $\Box$ pode ser substituída por um único operador de necessidade.

[6]O mundo w_3 não possui nenhuma relação de acessibilidade, mas pela definição 3.2.9 de $\models^M_w \Box A$, onde A deve ser válida em todos os mundos que se relacionam a w, nesse caso nenhum. A condição é satisfeita. O contrário de $\not\models^M_w \Diamond A$, pois precisamos de pelo menos um mundo onde A seja válido

Se unirmos os axiomas T e 4, juntamente com o K, teremos o sistema conhecido como $S4$ que possui as características de reflexividade e transitividade.

O axioma 5, $(\Diamond A \rightarrow \Box\Diamond A)$, representa o fato do que é "possível é necessariamente possível". Para ver porque o axioma 5 é válido, suponha que temos $\models^M_w \Diamond A$ para um mundo possível w no modelo M. Isso significa que temos um mundo possível w' tal que $\models^M_{w'} A$. Possuindo um mundo possível onde A é válido, então para qualquer mundo relacionado a w' $\Diamond A$ é válido, ou seja, $\models^M_w \Box\Diamond A$.

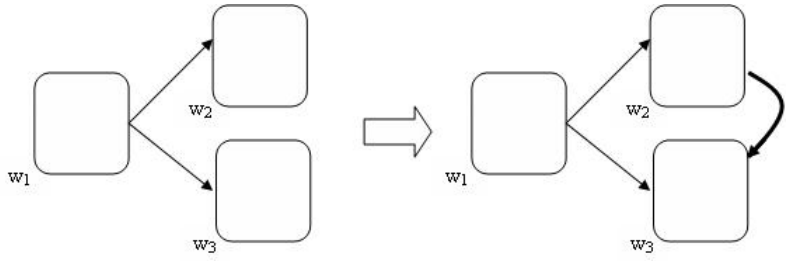


Figura 3.5: Propriedade euclidiana

O axioma 5 reflete a característica *euclidiana* (para todo w, w' e w'' em M se wRw' e wRw'' então $w'Rw''$). A sua união com o sistema KT é conhecida por $S5$ e o conjunto de suas características forma uma relação de equivalência.

O axioma B (em homenagem ao lógico Brouwer) reflete a característica de simetria, $A \rightarrow \Box\Diamond A$. Se possuirmos dois mundos w e w' em M e a relação wRw' então, por essa característica, teremos $w'Rw$.

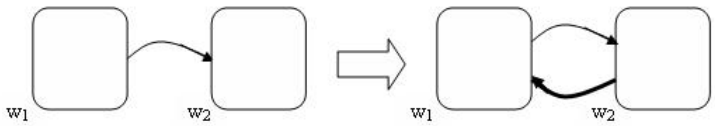


Figura 3.6: Propriedade de simetria

A lógica modal possui diversos outros sistemas, dentre eles ainda temos o axioma D com a característica da *serialidade*, C com a *confluência*, De com a *densidade*, etc. Além da combinação entre eles. Mais informações sobre a axiomática e sistemas da lógica modal podem ser encontradas em [9] e [43].

Para nós, o importante é conhecer o sistema $S4$ ($KT4$) que é a base para a

linguagem de ações e planos. Isso, porque precisamos das características de reflexividade e transitividade, além dessas é necessário o conhecimento da lógica multimodal, que será apresentada na próxima seção.

Nessa seção nós misturamos a axiomática com a semântica da lógica modal para aproveitar o contexto e mostrar as características de cada axioma. A seguir falaremos sobre necessidade e possibilidade sobre o ponto de vista axiomático e posteriormente comentaremos sobre a completude e adequação da lógica modal.

3.3.1 Dedução de teoremas em sistema modal normal

Chamamos de *teorema* qualquer sentença que pode ser provada com um conjunto de axiomas e regras de inferências. Axiomas são automaticamente teoremas.

Quando uma sentença A é dedutível de um conjunto de fórmulas lógicas (Γ) em um sistema através de regras de inferências diremos que A é um *teorema* de Γ para aquele sistema.

Denotaremos por $\Gamma \vdash_{\Sigma} A$ como um teorema A de um conjunto de sentenças Γ pelo sistema ou conjuntos de sistemas modais Σ.

Usaremos as regras de inferências RN (regra da necessitação), MP (modus ponens) e as fórmulas tautológicas. Os axiomas serão os K, T e 4 (sistema $S4$) que são os mais importantes no presente trabalho. Utilizaremos também da abreviação de $\Diamond$ ($\Diamond A \leftrightarrow \neg\Box\neg A$)

Exemplo 3.3.4 *Vamos, como exemplo, provar que a fórmula* $\Box A \leftrightarrow \neg\Diamond\neg A$ *é um teorema de* $S4$.

1. *Utilizando a abreviação de* $\Diamond$ *temos:* $\Diamond\neg A \leftrightarrow \neg\Box\neg\neg A$
2. *Sabendo que* $A \leftrightarrow \neg\neg A$
3. *Assim, por 1 e 2 obteremos* $\Diamond\neg A \leftrightarrow \neg\Box A$
4. *Pela regra tautológica, onde* $(A \leftrightarrow B) \leftrightarrow (\neg A \leftrightarrow \neg B)$. *A fórmula 3 resulta em* $\neg\Diamond\neg A \leftrightarrow \neg\neg\Box A$.

5. *Desse modo obtemos* $\neg\Diamond\neg A \leftrightarrow \Box A$ *ou* $\Box A \leftrightarrow \neg\Diamond\neg A$.

Está provado que $\Box A \leftrightarrow \neg\Diamond\neg A$ é um teorema de $S4$. Nessa prova não utilizamos os axiomas K, T nem o 4. Como a lógica modal é monotônica podemos deduzir que essa fórmula também é um teorema do sistema K, do KT, etc.

Exemplo 3.3.5 *No exemplo seguinte nós provaremos que a fórmula* $\Diamond\Diamond A \rightarrow \Diamond A$ *é um teorema de* $S4$ *para isso, iniciaremos com a fórmula do axioma* 4.

1. *Com a fórmula do axioma* 4 *temos:* $\Box\neg A \rightarrow \Box\Box\neg A$

2. *Pela contrapositiva aplicada em 1 nós temos:* $\neg\Box\Box\neg A \rightarrow \neg\Box\neg A$

3. *Por 2 e pela abreviação de* $\Diamond$ *obtemos:* $\Diamond\neg\Box\neg A \rightarrow \Diamond A$

4. *Novamente aplicando a abreviação de* $\Diamond$ *em 3 temos como resultado:* $\Diamond\Diamond A \rightarrow \Diamond A$

Está provado que $\vdash_{S4} \Diamond\Diamond A \rightarrow \Diamond A$, na literatura essa fórmula é conhecida como $4\Diamond$, por representar o inverso do axioma 4.

3.4 Lógicas multimodais

Até agora mostramos apenas as lógicas monomodais onde a definição do sistema é apresentada em função do seu conjunto de axiomas. Agora vamos falar sobre os operadores modais e suas características para diferentes "representações".

Antes tínhamos um operador de necessidade ($\Box$) e possibilidade ($\Diamond$), assim as características da lógica modal eram atribuidas a esses operadores. Para individualizar os operadores, indexamos os, ou seja, ao invés de um único operador de necessidade e possibilidade, agora temos $[i], [j], [q]...$ e $\langle i\rangle, \langle j\rangle, \langle q\rangle...$, onde cada um pode ter características diferentes.

Desse modo, as relações de acessibilidade também possuem índices, ou seja uma relação $wR_\alpha w'$ significa que o mundo w' é apenas acessível por w pelo índice α.

As definições de verdade novamente mudaram, agora precisamos dos índices das relações de acessibilidade entre os mundos.

Definição 3.4.1 *Seja w um mundo no modelo $M = (W, R, V)$*

- $\models^M_w [i]A$ *sse para cada* $w' \in W$ *tal que* wR_iw', $\models^M_{w'} A$
- $\models^M_w \langle i\rangle A$ *sse para algum* $w' \in W$ *tal que* wR_iw', $\models^M_{w'} A$

As lógicas multimodais são adequadas para representação de combinação de noções de conhecimento, podemos expressar, por exemplo, ações ou indivíduos etc.

Exemplo 3.4.2 $[i][j]A$

Pode significar que *um agente "i" sabe do conhecimento "A" do agente "j"*, ou *podemos executar uma ação "i" e depois "j" para obtermos "A"*.

Pode ser o caso de termos que expressar a interação entre os operadores, para isso é necessário acrescentar um *axioma de interação*, $I_{i,j} : [i]A \rightarrow [j]A$. Em uma interpretação em termos de conhecimento pode ser *o que o agente "i" conhece, o agente "j" também.*

Vamos dar um exemplo para entender melhor as regras da relação de acessibilidade da lógica multimodal.

Exemplo 3.4.3 *Seja M um modelo onde temos $W = \{w_1, w_2, w_3\}$ e $w_1R_iw_2$, $w_2R_iw_3$, $w_2R_jw_2$ e $w_2R_jw_3$, onde o índice "i" possui a característica da transitividade e o índice "j" da reflexividade. Veja a figura:*

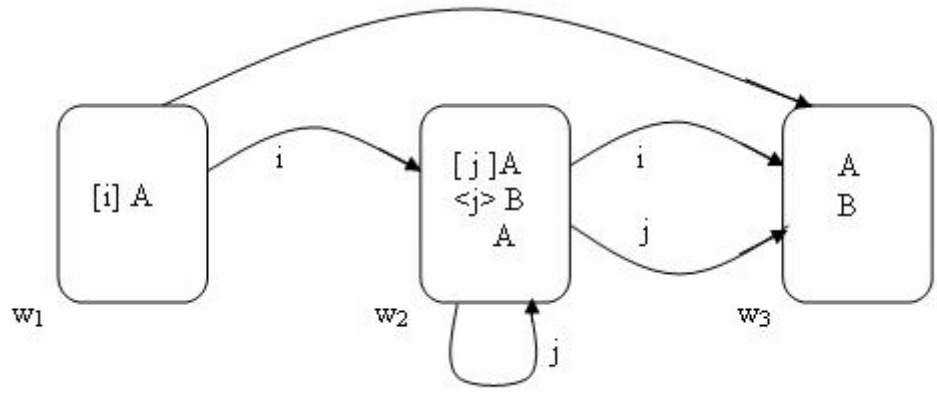


Figura 3.7: Exemplo de lógica multimodal

Como o índice "i" possui a propriedade da transitividade existe uma relação entre $w_1 R_i w_3$ *e o índice "j" a da reflexividade possui a relação* $w_2 R_j w_2$*. Por exemplo, a fórmula* $[i]A$*, para que ela seja válida em* w_1*,* A *deve ser válido em* w_2 *e* w_3*. Já a fórmula* $\langle j \rangle B$ *deve existir um mundo relacionado pelo índice "j" com* w_2 *onde* B *seja verdadeiro, que é o mundo* w_3*.*

Nesse capítulo demos uma introdução sobre lógica modal, com suas relações de acessibilidade, suas características e as lógicas multimodais. O objetivo é dar um conhecimento prévio ao leitor para entender a linguagem de ações e planos $\mathcal{LAP}$, que será apresentada no capítulo 5.

3.5 Completude e adequação

A lógica modal normal é *completa* porque toda sentença válida em um modelo semântico é um teorema em um conjunto de sistemas axiomáticos. Todas as fórmulas que são conseqüência semântica em um modelo, ou conjunto de modelos (representado por C) é um teorema em Σ, ou seja:

$$\models_C A \rightarrow \vdash_\Sigma A$$

Ela também é *adequada* (*sound*) porque todo teorema é uma sentença válida. Todo teorema provado pelo conjunto de sistemas axiomáticos pode ser provado como uma conseqüências da semântica de um modelo. Desse modo também é certo afirmar que:

$$\vdash_\Sigma A \rightarrow \models_C A$$

A prova da *completude* e *adequação* da lógica modal pode ser encontrada nos livros de Brian Chellas [9] e Sally Popkorn [43].

3.6 Considerações

A lógica modal é bastante utilizada na análise semântica, visto que as representações dos conectivos modais permitem expressar advérbios, o que a lógica clássica não pode representar.

Ela é um formalismo que pode ser modelado para lógica de predicados de primeira ordem. Na verdade, a lógica modal é um subconjunto bem comportado desta. Podemos substituir os quantificadores pelos operadores modais($\forall$ por $\Box$ e $\exists$ por $\Diamond$). Enquanto que o contrário não é verdade.

Uma das vantagens é um sistema de provas de teoremas simples, baseado em tableau. No próximo capítulo mostraremos este método, no seguinte demonstraremos que a lógica modal é adequada para representar inferências de cenários de raciocínio sobre ação e inclusive problemas de planejamento.

CAPÍTULO 4

MÉTODOS DE TABLEAUX MODAIS

Os sistemas de tableaux[1] consistem em um processo de *refutação*. O objetivo é encontrarmos inconsistência no conjunto resultante.

Chamaremos de fórmula objetivo a fórmula que queremos provar. A negação dessa fórmula mais as fórmulas válidas são decompostas em subfórmulas criando assim uma árvore de tableau. É então efetuada uma busca, segundo certos critérios, até que sejam encontradas contradições em todos os ramos dessa árvore. Concluindo a verificação, caso sejam encontradas inconsistências, provamos que a fórmula objetivo é um teorema da teoria considerada. Assim é dito que o tableau é fechado para essa teoria, caso contrário, o tableau é aberto.

No ano de 1936, Gerhard Gentzen [27] criou um método dedutivo de teoremas chamado de cálculo de seqüentes ou sistema de seqüentes. Gentzen foi o primeiro a separar as propriedades estruturais dedutivas da lógica das regras dos conectivos e quantificadores. Esse método utiliza axiomas que manipulam construções chamadas de *seqüentes*[2], ao invés de fórmulas.

Haskell B. Curry [10] foi o primeiro a utilizar o sistema de Gentzen para a lógica modal, depois vieram os trabalhos de Masao Ohnishi e Kazuo Matsumoto [42]. Kanger [28] utilizou pela primeira vez dispositivos "extra-lógica" para obter o sistema de Gentzen. Ele foi o percursor do que hoje é conhecido como sistema de tableaux indexados[3]. Em um sistema de tableaux indexados possuimos objetos que são referenciados por índices com a função de representar estruturas semânticas, no caso de lógica modal, podemos representar os mundos possíveis.

As fórmulas de um sistema de tableau indexado são expressões da forma $A : i$,

[1] Tableau em francês significa quadro e tableaux é o seu plural na lingua francesa

[2] Um seqüente pode ser interpretado como uma regra da forma $\Gamma \Rightarrow \triangle$, onde Γ e $\triangle$ são conjuntos finitos de fórmulas de uma linguagem L.

[3] *labelled tableau system* no original

onde A é uma fórmula lógica e i é um índice. Dessa forma, podemos dizer que A é válida em um mundo possível denotado pelo índice i.

Esses primeiros trabalhos ocorreram antes da semântica de Kripke, por isso essa linha é marcada pela sintática de seqüentes e pelos teoremas desenvolvidos por Gentzen.

A linha semântica, surgiu com o trabalho de E. W. Beth [1] e K. J. J. Hintikka nos anos 50. Beth introduziu o termo *tableau semântico* pela primeira vez, e pensou nos tableaux como métodos que tratam de encontrar contra-exemplos de forma sistemática. A representação que usou separava as fórmulas negadas e as não negadas distribuindo-as em duas colunas.

Os estudos de K. J. J. Hintikka [25] coincidiram, na época, com os de Beth na linha semântica. Assim, em 1955 ele pesquisou os métodos de tableau como mecanismos por refutação que fazem busca de contra-exemplos. A metodologia utilizada no trabalho de Hintikka para demonstrar a completude separa as propriedades abstratas de satisfabilidade, dos detalhes específicos do processo de construção dos tableaux. Esta idéia dura até o nosso tempo, porque desse modo é simples fazermos a prova de completude do método.

A representação usada por Hintikka é de uma árvore cujos nós são conjuntos de fórmulas. Porém esta estrutura é redundante, pois as fórmulas podem se repetir desnecessariamente. Raymond Smullyan [52] apresentou uma notação uniforme para os trabalhos de Beth e Hintikka. Ele desenvolveu as árvores de tableau com nós etiquetados por fórmulas e também as regras de expansão. Apresentou o termo conhecido como *tableau analítico* para expressar a característica principal semelhante a todos estes métodos, que é o princípio das subfórmulas. Nessa época ainda não existia um tableau para a lógica modal porque não havia uma semântica adequada para essa lógica.

As duas histórias começam a se encontrar mais tarde, nos anos 60, quando perceberam que o sistema de tableau semântico clássico e o sistema clássico de Gentzen eram essencialmente a mesma coisa. J. Jay Zeman [56] foi o primeiro a trabalhar com as duas simultaneamente. Até os anos 80 as duas linhas andaram juntas como se fossem dois lados da mesma moeda, mas recentemente os trabalhos semânticos se sobressaíram

na área prática, enquanto o sintático ficou no campo da teoria.

Há, também, os trabalhos de Rautenberg [44], Rajeev Goré [22] e Fitting [16]. As pesquisas de Fitting foram fundadas sobre as descobertas de Smullyan enquanto Goré seguiu a linha de Hintikka e Rautenberg. Nessa época os autores já possuíam o conhecimento do modelo de Kripke, tanto que os seus trabalhos usavam as regras de propagação de fórmulas em uma árvore que simulava essas propriedades.

As regras do sistema de tableau modal empregadas por Goré são definidas para cada sistema da lógica modal, por exemplo, para o axioma K existe uma regra que o define, assim como para T e 4, porém para o sistema $S4$ ($KT4$), que é a união desses três sistemas, existe uma regra independente das regras de K, T e 4, tornando o seu método não modular. *Castilho et al* [6] criaram um sistema genérico para o método de tableau modal, simplificando e diminuindo a quantidade de regras existente.

Goré seguiu a linha dos trabalhos de Hintikka e adaptou as regras de tableau para propagarem as fórmulas em uma árvore que simula as propriedades do modelo de mundos possíveis de Kripke. Assim como Fabio Massacci [36], Goré também utilizou a idéia de que as fórmulas devem ser carregadas de um nó do tableau para o seu filho (*top-down*). Essa notação é mais fácil de ser usada porque desse modo separamos as fórmulas que sofreram a ação das regras das que foram geradas por ela. *Castilho et al* utilizam uma notação diferente, onde a propagação das fórmulas não precisa ser *top-down* e que a estrutura também não precisa ser uma árvore.

4.1 Noções preliminares

Os sistemas de tableaux, independente da lógica utilizada, consistem em um processo de *refutação*, ou seja, a fórmula que queremos provar, chamaremos de fórmula objetiva, é negada e inserida no conjunto original, passando por regras definidas pelo sistema, sendo o objetivo é encontrar contradições ou inconsistência. Essas fórmulas são decompostas em subfórmulas criando assim a árvore de tableau. É, então, efetuada uma busca, segundo certos critérios, até que sejam encontradas contradições em todos os ramos da árvore. Concluindo a verificação, caso todas as folhas sejam inconsistentes, então é

provada que a fórmula objetiva é um teorema da teoria considerada.

A entrada de um sistema de tableau deve ser o conjunto de fórmulas original adicionada da negação da fórmula objetiva, isso porque logicamente se um conjunto de fórmulas $Y = \{A_1 \wedge A_2 \wedge ... \wedge A_k\}$ unido com uma fórmula A, $Y \cup A$, for verdadeiro então, $Y \cup \neg A$ não o é, se $A \in \{A_1, A_2, ...A_k\}$ então $\neg A \notin \{A_1, A_2, ...A_k\}$. Temos então o objetivo de achar contradições nos ramos da árvore criada pelo sistema.

O conjunto de fórmulas original mais a negação da fórmula que se deseja provar é a raiz da árvore, os ramos desta são criados através de regras pré-estabelecidas, chamadas de *regras de tableau*. Uma contradição em um ramo quer dizer que nesse nó existe uma fórmula A e sua negação $\neg A$ ou a constante *falso* ($\perp$). Caso isso ocorra então esse nó é chamado de *fechado*, nesse caso o sistema deve parar o processo nesse ramo e seguir por outro caminho, caso contrário o nó é dito *aberto*. O processo continua até que todos os ramos da árvore estejam *fechados* ou que não hajam mais regras aplicáveis.

O que difere bastante entre os métodos é como as regras são aplicadas. Elas são executadas no conjunto de fórmulas e o seu resultado é a criação de um ou mais nós na árvore de tableau, com as devidas modificações nas fórmulas.

O sistema de tableau para a lógica modal, conhecido como método de tableau modal, pode ter em seu sistema mais de um tableau, ou *sub-tableau*, onde cada sub-tableau representa um mundo possível. Ao se encontrar uma fórmula do tipo $\Diamond A$ em um nó da árvore, cria-se então um sub-tableau com a fórmula A para verificar as informações sobre esse mundo. Se os dados deste novo mundo forem contraditórios então as informações contidas no mundo atual, a respeito desse novo mundo possível, também são contraditórias. Isso quer dizer que se este sub-tableau for fechado então o nó que o criou também é fechado.

4.2 Tableau com grafo acíclico orientado à raiz (RDAG), regras de propagação e regras estruturais

O problema do método de Goré é que suas árvores de provas são restritivas e difíceis de automatizar, porque a propagação das fórmulas desse método é somente *top-down*; desse modo é difícil projetar um método de tableau para uma lógica modal que é baseada, por exemplo, nos axiomas de *densidade* ($\Diamond p \rightarrow \Diamond\Diamond p$) ou nos axiomas de *confluência* ($\Diamond\Box p \rightarrow \Box\Diamond p$). Outro problema foi que Goré criou um conjunto de tableau para cada sistema modal, por exemplo, para o sistema K existe um tableau e para o sistema $S4$(formado pelos axiomas K, T e 4) existe outro.

Uma solução para esse e outros problemas foi apresentada por *Castilho et al* [6] que modificaram a estrutura que o método de tableau cria, ao invés de uma árvore *top-down* utilizou um grafo acíclico dirigido com um nó raiz chamado de **RDAG**(grafo acíclico orientado à raiz). Agora a propagação das fórmulas não precisa mais ser exclusivamente *top-down* e a estrutura não precisa ser uma árvore, o que permitiu projetar regras para relações de acessibilidade que utilizam axiomas mais complexos, como *densidade* e *confluência*.

Nosso trabalho utilizou como base o modelo de *Castilho et al* devido a modularidade do método e pela facilidade de automatização, além de permitir a criação de um provador genérico para a lógica modal. Uma referência é o provador GTTP [47][4] que é a implementação do método sugerido por *Castilho et al*.

As regras de tableau para esse método foram divididas em três tipos, as *Regras Clássicas em conjuto com a regra da possibilidade ($\Diamond$)*, as *Regras Estruturais* e as *Regras de Propagação*.

- **Regras de propagação** - Corresponde aos axiomas *T*, *4*, *B* e *5* da lógica modal (são as propriedades de reflexividade, transitividade, simetria e euclidiana, respectivamente). As regras desse tipo funcionam do seguinte modo: deve-se propagar uma fórmula (a mesma ou outra fórmula gerada pela regra aplicada) do nó atual para

[4] *Generic Tableaux Theorem Prover*

um outro nó relacionado a este.

- **Regras estruturais** - As regras desse tipo só podem ser aplicadas quando existe um determinado padrão de relação entre os nós. Os axiomas correspondentes a esse tipo são, *D*, *De* e *C* (são respectivamente as propriedades de serialidade, densidade e confluência). A execução das regras desse tipo adicionam novos mundos (sub-tableaux) à árvore, esses sub-tableaux são criados vazios, ou seja, nenhuma fórmula é propagada a ele, a diferença entre as regras desse tipo são o modo como se relacionam. Será mostrado nas próximas seções.

As regras para os conectivos clássicos e para a regra $\Diamond$ são comuns a todos os sistemas modais. A regra do tableau que trata as fórmulas da *possibilidade* tem como função criar um novo mundo sempre que encontrar o operador $\Diamond$(ou $\neg\Box$). O próximo passo, então, é passar as fórmulas que são afetadas por esse operador para o novo mundo criado por ele.

O tableau também é uma estrutura cujos nós são etiquetados por um conjunto de fórmulas, porém para *Castilho et al* a estrutura do tableau é um grafo acíclico diferente do de Goré que é uma árvore.

A prova do método de *Castilho et al* é feita em dois passos: a construção do modelo nessa estrutura e verificação se ele satisfaz as fórmulas dos nós (chamado de *Lemma Fundamental* [6]).

No primeiro passo adicionamos novos arcos a estrutura, de acordo com a propriedade particular da relação de acessibilidade da lógica em questão. Devemos caracterizar quando dois nós são relacionados. Para isso podemos dizer que dados dois nós x e y, o segundo será acessível por x se existir um $n \geq 0$ e $x_0, ..., x_i, x_{i+1}, ..., x_n$ onde x_0 é x e x_n é y e x_{i+1} é um filho de x_i.

Através da caracterização do fechamento do grafo inicial com propriedades adicionais da lógica modal, podemos confirmar a correta propagação das fórmulas. Nesse sistema de tableau apenas as subfórmulas das fórmulas iniciais são propagadas.

4.3 Notação

O método de tableau que estamos prestes a mostrar é baseado em um **RDAG** (Grafo acíclico dirigido com um nó raiz) tendo algumas propriedades adicionais. Nesta seção, apresentamos as convenções e notação utilizadas nesse método.

Para minimizar o número de regras e facilitar a explicação, trabalharemos com a notação conjuntiva e utilizaremos apenas os operadores $\neg$ e $\wedge$ para as regras da lógica clássica. Assim uma fórmula do tipo $A \vee B$ equivale à $\neg(\neg A \wedge \neg B)$, a fórmula $A \rightarrow B$ é $\neg(A \wedge \neg B)$ e a fórmula $A \leftrightarrow B$ é igual à $\neg(A \wedge \neg B) \wedge \neg(\neg A \wedge B)$. Para a lógica modal utilizaremos tanto o $\Diamond A$ como $\Box A$. $\Diamond A$ pode ser visto como $\neg\Box\neg A$, ou seja, $\Diamond A \leftrightarrow \neg\Box\neg A$, mas utilizaremos a notação simplificada ($\Diamond$) para facilitar o entendimento.

Definição 4.3.1 ***Notação***

1. $\perp$ *denota a constante proposicional* falso *e* $\emptyset$ *denota o conjunto vazio;*
2. p, q *denotam proposições atômicas;*
3. A, B, A_i *e* B_i *denotam fórmulas bem formadas;*
4. ρ *denota o conjunto das propriedades da relação de acessibilidade da lógica modal em questão;*
5. $S, S_0, S_1, ...$ *denotam nós da árvore;*

Seja ρ o conjunto destas propriedades, definimos:

Definição 4.3.2 *Um ρ-RDAG é uma tripla ($\mathcal{N}$, Σ,* FOR*) onde:*

- $\mathcal{N}$ *é o conjunto de nós.*
- Σ *é o conjunto de arestas.*
- *($\mathcal{N}$, Σ) é um grafo acíclico orientado (DAG) com um nó distinto chamado raiz que pode acessar todos os nós em um fecho transitivo de Σ;*
- *($\mathcal{N}$, Σ) satisfaz todas as propriedades de ρ;*

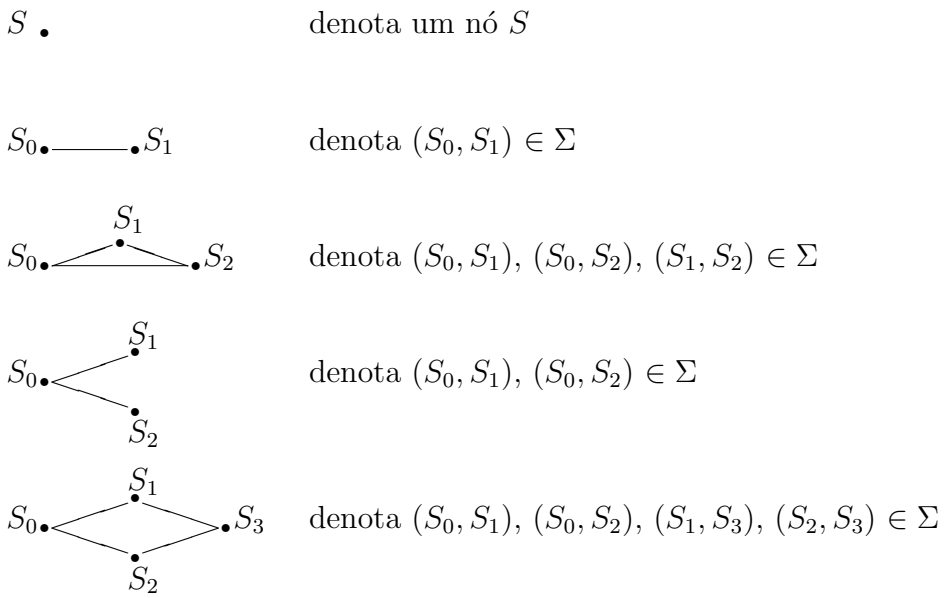
- FOR *é uma função que associa informações adicionais a cada um dos nós: se* x *é um nó,* FOR(x) *é um conjunto de fórmulas pertencentes a* x*.*

Definição 4.3.3 *Dado um conjunto* ρ *das propriedades da relação de acessibilidade de uma determinada lógica modal, um* ρ*-modelo é um modelo de* Kripke *cuja relação de acessibilidade satisfaz* ρ*. Uma fórmula é* ρ*-válida se, e somente se, é válida na classe de todos os* ρ*-modelos e será representada por* $\models_\rho P$*. Assim,* P *é um teorema de um sistema representado por um conjunto de propriedades se e somente se é* ρ*-válida.*

Definição 4.3.4 *Um* RGRAPH *é um grafo que possui um nó raiz, e um* ρ-RGRAPH *é um* RGRAPH *que satisfaz todas as propriedades de* ρ*.*

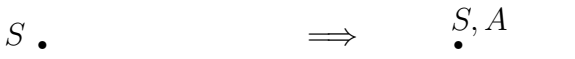
A representação $\Sigma(x)$ indica o conjunto dos nós acessíveis a partir de um nó x. Se existe um caminho de x até y que possui comprimento n este será representado por Σ^n, $\Sigma(x) = \{y \in \mathcal{N} : (\S, \dagger) \in \pm\}$. A relação diagonal $\{(x, x) : x \in \mathcal{N}\}$ pode ser representada por I ou por Σ^0.

Para facilitar o entendimento usaremos um diagrama para representar o **RDAG**. As arestas são orientadas da esquerda para a direita, enquanto os pontos são os nós do grafo.

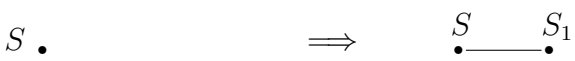


O sistema de tableau também consiste em reescrever a estrutura quando uma regra é aplicada a um nó. Antes de apresentar as regras vamos mostrar algumas transformações que geralmente ocorre na árvore pela aplicação daquelas.

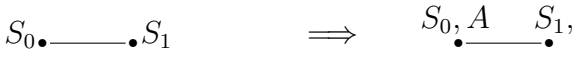
Do lado esquerdo da flecha dupla está o nó antes da aplicação da regra, também chamada de *pré-condição*; do lado direito da flecha está o nó resultante da execução da regra, *pós-condição*. O significado de cada regra é dado abaixo de sua diagramação. Para simplificar, usaremos S, A para representar $S \cup \{A\}$.



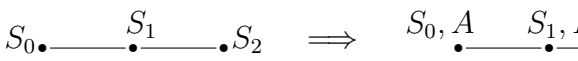
adiciona a fórmula A ao nó S, resultando em $S \cup \{A\}$



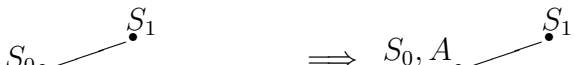
adiciona o novo nó S_1 como sucessor do nó S



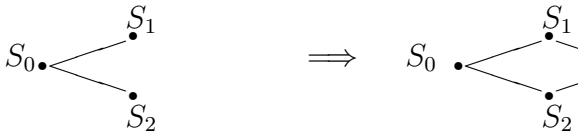
adiciona a fórmula A ao nó S_0 e a fórmula B ao nó S_1



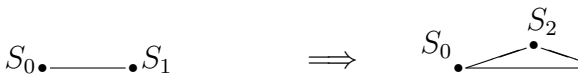
adiciona a fórmula A ao nó S_0, B ao nó S_1 e C ao nó S_2



adiciona a fórmula A ao nó S_0, B ao nó S_1 e C ao nó S_2

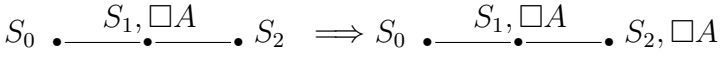


cria um novo nó S_3 e o define como o mesmo sucessor dos nós S_1 e S_2



adiciona o novo nó S_2 entre os nós S_0 e S_1

Esse modo de representar as regras possibilitou uma visualização melhor de sua semântica, assim podemos levar em conta, inclusive, as restrições de aplicação de alguma regra. Por exemplo:



pode ser lida da forma, "se S_1 é um nó relacionado com um nó antecessor e um outro sucessor, assim, se o nó S_1 possuir uma fórmula $\Box A$ então deve-se propagar esta fórmula para o nó sucessor (no caso o nó S_2)".

4.4 Regras

Nessa seção mostraremos as regras de tableau definidas por *Castilho et al.* As regras são definidas da forma apresentada na seção anterior.

O nome da regra é indicado no início. A flecha dupla separa as *pré-condições* das *pós-condições* (resultado da aplicação da regra).

As regras dos grupos das *regras de propagação* e das *regras estruturais*, também chamados de "Grupo 1"e "Grupo 2"respectivamente, simulam os axiomas de mesmo nome das relações de acessibilidade do modelo de Kripke (veja 3.2.1), vamos a elas.

- Regras clássicas e regra $\Diamond$:

  - Regra $\bot$: $A, \underset{\bullet}{\neg A}, S \quad \Rightarrow \quad A, \underset{\bullet}{\neg A}, \bot, S$

  - Regra $\neg$: $\underset{\bullet}{\neg\neg A}, S \quad \Rightarrow \quad \underset{\bullet}{\neg\neg A}, A, S$

  - Regra $\wedge$: $\underset{\bullet}{A \wedge B}, S \quad \Rightarrow \quad \underset{\bullet}{A \wedge B}, A, B, S$

  - Regra $\vee$: $\underset{\bullet}{\neg(A \wedge B)}, S \quad \Rightarrow \quad \underset{\bullet}{\neg(A \wedge B)}, C, S$

  onde C é $\neg A$ ou $\neg B$

  - Regra $\Diamond$: $\underset{\bullet}{\Diamond A}, S \quad \Rightarrow \; \Diamond A, S \; \bullet\!\!\_\_\_\!\!\bullet A$

  A regra da *possibilidade* ($\Diamond$) cria um mundo novo na estrutura e passa a sua sub-fórmula para ele. Após nomearmos esse novo mundo ele fica disponível para ser acessível pelas outras regras aqui definidas.

- Regras de propagação:

  - Regra K: $\Box A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet S_1 \Rightarrow \Box A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet A, S_1$

  - Regra T: $\overset{\Box A, S}{\bullet} \Rightarrow \overset{\Box A, A, S}{\bullet}$

  - Regra 4: $\Box A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet S_1 \Rightarrow \Box A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet \Box A, S_1$

  - Regra B: $S \bullet\!\!-\!\!-\!\!\overset{\Box A, S_1}{\bullet} \Rightarrow A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet \Box A, S_1$

  - Regra $5_{\rightarrow}$: $S \overset{\Box A, S_1}{\underset{S_2}{\bullet}} \Rightarrow S \overset{\Box A, S_1}{\underset{\Box A, S_2}{\bullet}}$

  - Regra $5_{\uparrow}$: $S \bullet\!\!-\!\!-\!\!\overset{\Box A, S_1}{\bullet} \Rightarrow \Box A, S \bullet\!\!-\!\!-\!\!-\!\!\bullet \Box A, S_1$

  - Regra $5_{\downarrow}$: $S \bullet\!\!-\!\!-\!\!\overset{\Box A, S_1}{\bullet}\!\!-\!\!-\!\!\bullet S_2 \Rightarrow S \bullet\!\!-\!\!-\!\!\overset{\Box A, S_1}{\bullet}\!\!-\!\!-\!\!\bullet \Box A, S_2$

- Regras estruturais:

  - Regra D: $\overset{S}{\bullet} \Rightarrow S \bullet\!\!-\!\!-\!\!-\!\!\bullet \emptyset$

  - Regra $C0$: $S \overset{S_1}{\underset{S_2}{\bullet}} \Rightarrow S \overset{S_1}{\underset{S_2}{\bullet}} \emptyset$

  - Regra $C1$: $S \bullet\!\!-\!\!-\!\!-\!\!\overset{S_1}{\bullet} \Rightarrow S \bullet\!\!-\!\!-\!\!\overset{S_1}{\bullet}\!\!-\!\!-\!\!\bullet \emptyset$

  - Regra De: $S \bullet\!\!-\!\!-\!\!-\!\!\overset{S_1}{\bullet} \Rightarrow S \overset{\emptyset}{\bullet} S_1$

Para utilizarmos o sistema de tableau devemos associar um conjunto de regras a ele. Todos os sistemas desse método de tableau devem conter as regras clássicas, a

regra da *possibilidade* ($\Diamond$) e a regra K(pois nesse trabalho só consideraremos as lógicas modais normais). Assim, utilizando das outras regras, que são escolhidas conforme as propriedades da relação de acessibilidade da lógica modal que queremos, podemos provar qualquer fórmula na lógica modal normal.

A regra $\Diamond$ cria um novo mundo e propaga a sua sub-fórmula para ele, ou seja, quando temos $\Diamond A$ em um mundo S_1 então devemos criar um mundo S_2 e propagar a fórmula A para ele.

A regra K mantém todo o sistema organizado. Essa regra diz que, quando temos um mundo S_1 relacionado a um outro mundo S_2 e se em S_1 existir uma fórmula do tipo $\Box A$ então devemos propagar a fórmula A para o mundo S_2.

A tabela abaixo ilustra as propriedades relacionais e suas respectivas regras.

Tabela 4.4.1

	Relação de acessibilidade	**Regras**	
Grupo 1	Reflexiva Simetria Transitiva Euclidiana	T B 4 $5_{\uparrow}$, $5_{\downarrow}$, $5_{\rightarrow}$	Regras de Propagação
Grupo 2	Serial Densidade Confluência	D De $C0$, $C1$	Regras Estruturais

Para representar a propriedade da reflexividade, cujo conceito é ter um mundo que é acessível a partir de si mesmo, foi criada a regra T. Essa regra simula o axioma T, tem como função propagar a fórmula do operador $\Box$ no mundo atual. Para cada $\Box A$ no mundo S, a fórmula A deve ser inserida no próprio mundo S.

A regra B representa a propriedade de simetria. Sejam dois mundos possíveis S_1 e S_2, S_2 deve ser acessível a partir de S_1 e conter a fórmula do tipo $\Box A$, desse modo os pré- requisitos para se executar a regra B são satisfeitos. O efeito dessa regra é propagar a fórmula A para o mundo S_1.

Para aplicar a regra 4 precisamos de dois mundos S_1 e S_2, onde S_2 seja acessível a partir de S_1 e que S_1 contenha uma fórmula do tipo $\Box A$. A aplicação dessa regra consiste em propagar a fórmula $\Box A$ para o mundo S_2 simulando assim o axioma 4 que representa a propriedade da transitividade.

A propriedade euclidiana é representada pelo axioma 5, no sistema de tableau aqui proposto essa propriedade é satisfeita pelas regras $5_{\uparrow}$, $5_{\rightarrow}$ e $5_{\downarrow}$. A primeira regra é aplicada quando temos um mundo S_1 que tenha um antecessor S_0, assim como a propriedade de simetria. Se S_1 possuir uma fórmula do tipo $\Box A$ então devemos propagá-la para o mundo S_0.

As outras duas regras que representam a propriedade euclidiana precisam de um terceiro mundo, chamaremos ele de S_2. Para representar a aplicação dessas regras vamos ter o seguinte cenário: S_1 tem como antecessor S_0 e possui uma fórmula do tipo $\Box A$. Para a regra $5_{\rightarrow}$ se S_0, além do sucessor S_1 também tiver um sucessor S_2, podemos dizer que S_1 é irmão de S_2, propagaremos a fórmula $\Box A$ para o mundo S_2. Na regra $5_{\downarrow}$ é necessário que S_2 seja sucessor do mundo S_1 para propagarmos a fórmula $\Box A$ para S_2.

As regras estruturais, cujas as propriedades são serial (regra D), densidade (regra De) e confluência (regra $C0$ e $C1$), têm o objetivo de criar novos mundos e fornecer assim mais mundos possíveis para o modelo. Essas regras não propagam fórmulas.

A regra D, que simula o axioma D, criar um mundo vazio que seja sucessor de um mundo S. A regra De, densidade, deve ter como condição para ser executada a existência de dois mundos, S_1 e S_2 e que S_1 seja relacionado com S_2, essa regra cria um novo mundo que seja acessível tanto por S_1 como por S_2.

Por fim temos regra que aplica a propriedade da confluência, $C0$ e $C1$, elas simulam o axioma C. Essas regras devem ter um mundo S_1 acessível por um mundo S_0. Para aplicar a regra $C0$ deve, ainda, existir um mundo S_2 irmão de S_1, ou seja, que S_2, assim como S_1, seja sucessor de S_0. O resultado é um novo mundo acessível através dos mundos S_1 e S_2. A regra $C1$ apenas precisa da primeira condição do axioma C, criamos um novo mundo acessível por S_1.

Chamaremos as regras do grupo 1 (propagação) por ρ_1 e as do grupo 2 (estru-

turais) por ρ_2. Todo sistema de tableau modal deve possuir as regras clássicas, a regra $\Diamond$ e a regra K, essas regras são comuns a todo cálculo de tableau. Mais alguma ou nenhuma regra de $\rho_1 \cup \rho_2$.

Definição 4.4.2 *Um ($\rho_1 \cup \rho_2$)-tableau para a fórmula A é uma seqüência finita $\Upsilon_0, ..., \Upsilon_i, \Upsilon_{i+1}$ onde:*

- Υ_0 *é um RDAG que consiste de apenas um nó cujo o conjunto de fórmulas associado é* $\{A\}$*;*
- Υ_{i+1} *é obtido de* Υ_i *pela aplicação de alguma regra clássica, ou pela regra* $\Diamond$ *ou pela regra* K *ou por qualquer uma das regras de* $\rho_1 \cup \rho_2$*;*

Terminamos a execução em um nó Υ_i quando este é *fechado* ou quando todas as regras foram aplicadas nele. Se não há mais regras e o nó não for fechado então ele é *aberto*.

4.5 Considerações

O método de tableau modal, mostrado nesse capítulo, é completo e adequado e a sua prova pode ser encontrada em [6], também será utilizado para provar teoremas na lógica de ações e planos.

No próximo capítulo falaremos sobre essa lógica de representação e as regras para o método de tableau para solucionar o problema da persistência.

CAPÍTULO 5

$\mathcal{LAP}_{\rightsquigarrow}$: DEFINIÇÃO E PROVA DE TEOREMAS

$\mathcal{LAP}_{\rightsquigarrow}$(lógica de ações e planos), sugerido por *Castilho et al* em [5], é uma abordagem monotônica fundamentada em lógica modal que permite raciocinar sobre ações em lógica e que trata convenientemente o problema da persistência e da ramificação.

A idéia desse formalismo é definir uma linguagem lógica de base fundamentada em lógica multimodal, que tenha como objetivo ser o mais simples possível para possibilitar a implementação de um provador automático de soluções de problemas.

Ao contrário do cálculo de situações [39], que é baseado em uma lógica de primeira ordem, essa abordagem utiliza os fundamentos da lógica modal. Mais precisamente $\mathcal{LAP}_{\rightsquigarrow}$ é uma lógica multimodal proposicional. Ela é mais simples que a PDL[1] (Lógica proposicional dinâmica), porém demonstra possuir um poder de representação suficiente para expressar todos os problemas que foram investigados em raciocínio sobre ações até aqui.

A lógica de ações e planos($\mathcal{LAP}_{\rightsquigarrow}$) contém um operador modal ($[\alpha]$) para cada ação atômica α e um operador suplementar ($\Box$) que é usado para representar as leis de domínios, ou seja, todas as fórmulas que são verdadeiras em qualquer mundo. Basicamente a lógica de cada operador $[\alpha]$ corresponde a um axioma da lógica modal K, enquanto que as lógicas do $\Box$ obedecem ao sistema $S4$. $\Box$ interage com cada ação atômica $[\alpha]$ da forma $\Box A \rightarrow [\alpha]A$.

Na próxima seção veremos as notações da linguagem e apresentaremos formalmente a lógica básica de $\mathcal{LAP}_{\rightsquigarrow}$, também introduziremos o conceito de *relação de dependência* ($\rightsquigarrow$). Por fim mostraremos o método de prova de teoremas, descrito no capítulo anterior, para essa lógica.

[1]Do original em inglês, *Proposicional dynamic logic*, PDL é um sistema formal para racionalizar sobre programas. Podemos descreve-lo como a união de três ingredientes, lógica de predicados de primeira ordem, lógica modal e algebra de eventos regulares. A idéia de PDL surgiu dos trabalhos de Engeler [13], Hoare [26] e Yanov [55].

5.1 Linguagem e notação

Seja $ACT = \{\alpha, \beta, ...\}$ o conjunto de *ações atômicas* como "*carregar*"e "*atirar*", e seja $ATM = \{P, Q, ...\}$ o conjunto de *fórmulas atômicas* ou apenas *átomos* como "*Carregado*"e "*Vivo*". Ambos conjuntos podem ser infinitos. O conjunto $LIT = \{L, L_1, L_2, ...\}$ são os *literais* e são formados por todos os átomos e suas negações, ou seja se $A \in ATM$ então $A \in LIT$ e $\neg A \in LIT$.

O conjunto de *fórmulas* ou sentenças, FOR, é definido como o menor conjunto possível que:

- $ATM \subseteq FOR$;
- $\top, \bot \in FOR$, onde $\top$ representa a constante verdade e $\bot$ é igual a constante falsa, $\top = \neg\bot$;
- Se $A, B \in FOR$ e $\alpha \in ACT$ então $\neg A, A \wedge B, \Box A, [\alpha]A, [\alpha](A \wedge B) \in FOR$;

Usaremos o operador de possiblidade em conjunto com uma fórmula $\Diamond A$ como abreviatura para $\neg\Box\neg A$, assim como $\langle\alpha\rangle A$ representa $\neg[\alpha]\neg A$.

Uma fórmula do tipo $[\alpha]A$ é lida como "A é verdadeiro depois da execução da ação α", $\Box A$ é lida como "a fórmula A é sempre verdade", $[\alpha]\bot$ significa que α não é executável e uma fórmula sem um operador modal é uma fórmula *clássica*. A fórmula $\langle\alpha\rangle\top$ significa que α é executável e $\Diamond A$ será usada para tarefa de planificação, a leitura dessa fórmula é "existe uma seqüência de ações que torna A verdadeira", detalharemos melhor esse conceito no capítulo 6.

Agora ilustraremos a linguagem $\mathcal{LAP}_{\rightsquigarrow}$ e utilizaremos o cenário clássico do tiro em Yale [24] para exemplificá-la(Falaremos mais sobre esse exemplo na seção 5.1.2). Vamos descrever essa lógica em: *base de conhecimento*, *objetivos*, *leis de domínio* e *relação de dependência*.

- A *base de conhecimento* (BC) representa o estado atual do nosso cenário. BC consiste num conjunto finito de fórmulas que não contenham o operador modal $\Box$.

Exemplo 5.1.1 *Um exemplo da base de conhecimento é:*

$$BC = \{Caminhando, Vivo, [atirar]\neg Vivo\}$$

As proposições atômicas *Vivo* e *Caminhando*; E a fórmula da ação *atirar*.

- Os *objetivos* são fórmulas que derivam da base de conhecimento onde as leis de domínio são verificadas. Particularmente eles são os teoremas que queremos provar, com fórmulas do tipo $\Diamond A$, onde A é clássica. Esses tipos de fórmulas aparecem quando nós queremos provar que existe um plano que nos leva ao estado onde A é verdade. Por exemplo: $\Diamond Morto$, $\Diamond(Vivo \wedge \neg Carregada)$.

- As *leis de domínio*, denotado por $LEIS$, são um conjunto finito de fórmulas da forma $\Box A$, que geralmente representam as leis sobre o mundo. Podemos dividir, primeiramente, as leis de domínio em duas categorias, *as leis estáticas* e as *as leis dinâmicas*.

- A *relação de dependência* é o método utilizado para solucionar o problema da persistência nessa lógica. Consiste em retirar do modelo representativo as fórmulas chamadas de *axiomas de persistência*[2].

O conjunto dos pares (α, L), onde L independe de α é chamado de *relação de independência* entre ações e literais. Representamos pela forma $\alpha \not\rightsquigarrow L$.

Então podemos trocar todos os axiomas de persistência pelos pares que representam as relações de independência, entretanto isso será um trabalho considerável imaginando um domíno grande e complexo.

A hipótese da inércia diz que "quase todos"os literais L são independentes de uma dada ação α, então é razoável pensar que o complemento da relação de independência é muito menor. O complemento é chamado de *relação de dependência* e é representado pelo símbolo $\rightsquigarrow$.

[2]Em inglês são chamadas de *frame axioms*. São fórmulas do tipo $\Box(L \rightarrow [\alpha]L)$, significa que α não altera o valor de L de verdadeiro para falso. Nesse caso dizemos que α *não causa* $\neg L$, ou que $\neg L$ *é independente de* α.

A solução do problema de persistência está no fato que ao invés de escrever todos os axiomas de persistência, colocamos apenas os pares de literais e ações que representam as relações de dependência.

Uma fórmula do tipo $\Box A$ é classificada como lei estática (ou restrição de domínio), caso a sub-fórmula A não possua o operador modal de ação ($[\alpha]$), para alguma $\alpha \in ACT$, ou seja, A é uma fórmula clássica. Um exemplo é a seguinte restrição:

$$\Box(Caminhando \rightarrow Vivo) \qquad ou \qquad \Box(Vivo \leftrightarrow \neg Morto)$$

Observamos que com o operador modal $\Box$, pelo sistema $S4$, estamos dizendo que em todos os mundos possíveis desse cenário a restrição $(Caminhando \rightarrow Vivo)$ deve ser obedecida. Essa lei serve para dizer que as fórmulas são sempre verdadeiras independente da ação executada e devem ser respeitadas.

Para a fórmula $\Box A$ ser classificada como lei dinâmica, esta deve possuir um operador modal que indica uma ação ($[\alpha]$) e seus respectivos efeitos.

Exemplo 5.1.2

$$\Box[carregar]Carregada$$

Significa que a qualquer momento (mundo), quando se executar a ação *carregar* então a arma estará $Carregada$. Após a execução da ação *carregar* a fórmula clássica $Carregada$ é verdadeira.

As leis dinâmicas podem ainda ser classificadas em várias categorias, dependendo de sua forma, função ou significado. As principais são as *leis de efeito das ações*, *leis de inexecutabilidade* e *leis de executabilidade*.

- As *leis de efeito das ações* são fórmulas do tipo $\Box(B \rightarrow [\alpha]C)$, onde B é uma *pré-condição* para executar a ação α e que resulta em um estado C (*pós-condição*).

  Exemplo 5.1.3 *Um exemplo de leis de efeito da ação é:*

  $$\Box(Carregado \rightarrow [atirar]Morto)$$

que diz que, "se uma arma estiver carregada então após atirar, o peru estará morto".

Uma particularidade dessas leis, são os *axiomas de persistência* que estão representadas indiretamente em função da relação de dependência.

As fórmulas do tipo $\Box((B \wedge A) \rightarrow [\alpha]A)$ chamados de *axiomas de persistência condicional*. Assim como nos axiomas de persistência, o resultado da ação não altera o estado do sistema. A diferença está na condição necessária para executar a ação.

Exemplo 5.1.4 $\Box((\neg Carregada \wedge Vivo) \rightarrow [atirar]Vivo)$

que significa que "se a arma não estiver carregada e o peru estiver vivo, então após a execução do ato de *atirar* o peru continuará vivo".

- As *Leis de inexecutabilidade* são da forma de $\Box(A \rightarrow [\alpha]\bot)$. Elas asseguram que uma ação α não pode ser executada no contexto A, ou as fórmulas do tipo $\Box[\alpha]\bot$ que significa que uma ação não pode ser executada independente do estado do sistema, as ações, dessa última forma, nunca poderão ser executadas.

  Exemplo 5.1.5 $\Box(\neg Tem\_Arma \rightarrow [atirar]\bot)$

  significa que "se a pessoa não tiver uma arma então ela não pode atirar".

- *Leis de executabilidade* garantem a possibilidade de executar uma ação, $\Box(A \rightarrow \langle\alpha\rangle\top)$ ou $\Box\langle\alpha\rangle\top$

  Exemplo 5.1.6 *Uma fórmula que diz o contrário das leis de inexecutabilidade* $\Box(Tem\_Arma \rightarrow \langle atirar\rangle\top)$

  Esse exemplo diz que "em qualquer mundo se tiver uma arma então pode atirar".

Vários outros tipos de leis ainda podem ser escritas nessa linguagem, geralmente quando indicam uma situação, como por exemplo, $\Box(\neg Vivo \rightarrow \Box\neg Vivo)$, que diz

que nenhuma ação pode mudar o estado da pré-condição, nesse caso, não podemos trazer um peru que morreu de volta para vida. Essas fórmulas são chamadas de *proposições imutáveis.*

Outro caso é quando não podemos mudar a ordem de execução de uma seqüência de ações.

Exemplo 5.1.7 $\Box(Carregado \rightarrow [atirar]\Box[fazer\_andar]\bot)$

expressa que a ação $fazer\_andar$ nunca pode ser executada depois de $atirar$ no estado do mundo em que a arma estiver $Carregada$, pois o peru morreu.

Para a lógica de ações e planos é definida uma semântica baseada na *semântica dos mundos possíveis* de Kripke (veja a seção 3.2.1). Vamos, agora, formalizar a semântica de $\mathcal{LAP}_{\rightsquigarrow}$.

Definição 5.1.8 *Definimos um modelo para $\mathcal{LAP}_{\rightsquigarrow}$, chamado de $\mathcal{LAP}_{\rightsquigarrow}-$modelo, como uma quádrupla da forma $\mu = \langle W, \{R_\alpha : \alpha \in ACT\}, R_\Box, \tau\rangle$, onde W é o conjunto de estados(mundos) possíveis, $R_\Box$ e cada R_α são as relações binárias em W, chamadas de relações de acessibilidade e $\tau : ATM \rightarrow 2^W$ é uma interpretação de átomos, também chamada de valoração*[3]*. Precisamos que $R_\Box$ seja reflexiva e transitiva*[4] *e $R_\alpha \subseteq R_\Box$, para cada ação $\alpha \in ACT$.*

Definição 5.1.9 *Uma* relação de dependência *é uma relação binária entre ações e literais $\rightsquigarrow \subseteq ACT \times LIT$. O complemento de $\rightsquigarrow$ é denotado por $\not\rightsquigarrow$ e é chamado de* relação de independência. *Um $\mathcal{LAP}_{\rightsquigarrow}-$ modelo $\mu = \langle W, \{R_\alpha : \alpha \in ACT\}, R_\Box, \tau\rangle$, caso exista $wR_\alpha w'$ então teremos para todo α:*

- *$\alpha \not\rightsquigarrow P$ e $w \notin \tau(P)$ implica que $w' \notin \tau(P)$;*
- *$\alpha \not\rightsquigarrow \neg P$ e $w \in \tau(P)$ implica que $w' \in \tau(P)$.*

As condições de verdade são:

[3]Na seção 3.1 demos uma noção sobre valoração. Para mais detalhes sobre esse assunto em lógica modal, pode-se ver [9].

[4]Reflexiva, assim como a transitiva, é uma característica da lógica modal que corresponde aos axiomas T e 4 respectivamente, ver seção 3.3

- $\models^{\mu}_{w} \top$; Que significa que $\top$ é sempre válido;
- $\not\models^{\mu}_{w} \bot$; Que significa que $\bot$ nunca é válido;
- $\models^{\mu}_{w} P$ sse $w \in \tau(P)$, para $P \in ATM$; Se P for uma fórmula atômica e se w pertencer à valoração de P então P é válido em w no modelo μ;
- $\models^{\mu}_{w} \neg A$ sse $\not\models^{\mu}_{w} A$; A negação de A é válida em w no modelo μ somente se A não for.
- $\models^{\mu}_{w} A \wedge B$ sse $\models^{\mu}_{w} A$ e $\models^{\mu}_{w} B$; A fórmula $A \wedge B$ é válida somente se as fórmulas A e B também forem válidas.
- $\models^{\mu}_{w} [\alpha]A$ sse para todo $w' \in W : wR_{\alpha}w'$ implica que $\models^{\mu}_{w'} A$; Isso significa que para $[\alpha]A$ ser válida em w, a fórmula A deve ser válida em todo w' que possui um relacionamento de arco[5] α com w.
- $\models^{\mu}_{w} \Box A$ sse para todo $w' \in W : wR_{\Box}w'$ implica que $\models^{\mu}_{w'} A$; Isso significa que para $\Box A$ ser válida em w, a fórmula A deve ser válida em todo w' que possui um relacionamento com w.

Uma fórmula A é válida no modelo $\mu = \langle W, \{R_{\alpha} : \alpha \in ACT\}, R_{\Box}, \tau\rangle$ se $\models^{\mu}_{w} A$ para todo $w \in W$. A é um $\mathcal{LAP}_{\rightsquigarrow}$-válido (denotado por $\models_{\mathcal{LAP}_{\rightsquigarrow}} A$) se A é verdade em todo $\mathcal{LAP}_{\rightsquigarrow}$-modelo. A é satisfatível se $\models^{\mu}_{w} A$ para algum $\mu = \langle W, \{R_{\alpha} : \alpha \in ACT\}, R_{\Box}, \tau\rangle$ e $w \in W$.

A relação $\alpha \rightsquigarrow L$ significa que "α *pode causar* L", isso expressa que após a execução da ação α o valor de L *pode mudar* de $falso$ para *verdadeiro*. Percebemos que a relação de dependência não *causa* mudança, mas *permite* a alteração do valor de verdade do literal.

Vamos supor que temos um cenário onde P não é verdadeiro em w e que temos $\alpha \rightsquigarrow P$. Então sabemos que a execução da ação α pode tornar P verdadeiro ou a

[5] um arco é "ligação"de um mundo w a outro mundo w' ou a ele mesmo. Um arco nomeado como α só pode ser acessível por axiomas que possuem esse índice, para mais detalhes veja lógica multimodal em [9]

pode, também, levar a uma situação onde P continua não sendo válida. Agora supomos que temos $\alpha \not\rightsquigarrow P$, então temos certeza que a execução da ação α nunca causará uma situação onde P será verdadeiro. Para toda conseqüência de α, o literal P continuará a ser falso.

5.1.1 Axiomatização

A axiomatização do $\mathcal{LAP}_{\rightsquigarrow}$ é feita seguindo os seguintes axiomas da lógica modal e as regras de inferência abaixo:

- PL : são todas as tautologias da Lógica Proposicional Clássica;
- $K([\alpha])$ para o índice α.
- Os axiomas K, T e 4 para $\Box$;
- $I(\Box, [\alpha])$: $\Box A \rightarrow [\alpha]A$, é a relação entre os operadores $\Box$ e $[\alpha]$, isso significa que a fórmula A é válida para todos os arcos da ação α;
- As regras *Modus Ponens* (MP) e da necessitação (RN);
- Persistência $([\alpha])$: $\neg L \rightarrow [\alpha]\neg L$, se $\alpha \not\rightsquigarrow L$;

A partir de RN e $I(\Box, [\alpha])$ podemos derivar a inferência da regra RN para $[\alpha]$: de A infere $[\alpha]A$. Assim como, de 4 e $I(\Box, [\alpha])$ podemos provar $\Box A \rightarrow [\alpha_1][\alpha_2]...[\alpha_n]A$, para todo $n \geq 1$.

5.1.2 Tiro em Yale em $\mathcal{LAP}_{\rightsquigarrow}$

Vamos demonstrar o cenário do tiro em Yale [24] na lógica de ações e Planos. Esse cenário é muito utilizado em trabalhos envolvendo o problema da persistência porque não conseguimos inferir o resultado desejado quando utilizamos uma ação sem efeito.

Steve Hanks e Drew McDermott trabalhavam na universidade de Yale quando propuseram esse problema. Trata de um cenário na qual temos uma vítima, geralmente

um peru[6]. Este pode estar vivo(representado por $Vivo$) ou morto($\neg Vivo$). Também temos uma arma que pode estar carregada ou não, representada respectivamente por $Carregada$ e $\neg Carregada$.

Temos, também, três ações, o ato de carregar a arma que chamaremos de *carregar*, de disparar a arma, chamaremos de *atirar* e esperar que receberá o nome de *esperar*.

A ação *carregar* tem o objetivo de carregar a arma, o seu efeito é tornar $Carregada$ verdadeiro. A ação *atirar* matará o peru se a arma estiver carregada, percebemos que esta ação precisa de uma pré-condição para ser executada, $Carregada$ tem que ser verdadeiro. O efeito dessa última tornará o valor de $Vivo$ para falso($\neg Vivo$) e descarregará a arma($\neg Carregada$).

Por fim a ação *esperar* tem o objetivo de esperar um instante, não causa nenhuma alteração nos valores dos literais. É uma ação sem efeito nem pré-condição.

Considere as ações *esperar*, *carregar* e *atirar* e os átomos de $Carregado$ e $Vivo$. O cenário é montado conforme segue:

- Leis de domínios ($LEIS$):

  1. $\Box\langle esperar\rangle\top$,
  2. $\Box\langle carregar\rangle\top$,
  3. $\Box\langle atirar\rangle\top$,
  4. $\Box[carregar]Carregado$,
  5. $\Box[atirar]\neg Carregado$,
  6. $\Box(Carregado \rightarrow [atirar]\neg Vivo)$,
  7. $\Box((\neg Carregado \wedge Vivo) \rightarrow [atirar]Vivo))$

- Base de conhecimento (BC):

  1. $\neg Carregado$,

[6]Originalmente a vítima se chamava Fred, depois alterada para um peru.

2. $Vivo$

- Relação de dependência ($\rightsquigarrow$):
  - $carregar \rightsquigarrow Carregado$,
  - $atirar \rightsquigarrow \neg Carregado$,
  - $atirar \rightsquigarrow \neg Vivo$

Note que existem leis de executabilidade $LEIS$ (1), $LEIS$ (2) e $LEIS$ (3) para expressar que as ações *esperar*, *carregar* e *atirar* são sempre executáveis; Leis de efeito de ações: LAW (4), LAW (5) e LAW (6); E também o axioma de persistência condicional LAW (7).

As relações de dependência substituem os axiomas de persistência, minimizando a quantidade de fórmulas para expressá-las. Por exemplo: a ação *esperar* não altera nenhum valor dos literais, então $Vivo$, $\neg Vivo$, $Carregado$ e $\neg Carregado$ são independentes dessa ação. Deveríamos ter, então, quatro axiomas de persistência que descreveriam essa independência($\Box(Carregado \rightarrow [esperar]Carregado)$, $\Box(\neg Carregado \rightarrow [esperar]\neg Carregado)$, $\Box(Vivo \rightarrow [esperar]Vivo)$ e $\Box(\neg Vivo \rightarrow [esperar]\neg Vivo)$). O complemento da relação de independência dessa ação é vazio, por isso não temos relações de dependência que envolvam *esperar*.

Desse modo conseguimos solucionar o problema da persistência[7], minimizando a quantidade de fórmulas nescessárias para expressar a independência entre as ações e os literais.

5.2 Método de tableau para $\mathcal{LAP}_{\rightsquigarrow}$

Um sistema de tableau, como foi apresentado no capítulo 4, é um sistema baseado em refutação que constrói uma árvore através de regras aplicadas no conjunto de fórmulas até encontrar uma contradição ou não conseguir aplicar mais regras nas fórmulas. Quando encontramos uma contradição em todas as folhas da árvore de tableau dizemos

[7] O axioma de persistência condicional não possui uma independência entre ação e um único literal por essa razão deveremos mantê-lo no cenário.

que este tableau é fechado e significa que conseguimos provar que o nosso cenário está correto.

O sistema de tableau faz uso da lógica multimodal onde cada índice é uma ação, tendo ainda o operador $\Box$ que indica que a sua fórmula sempre é verdadeira independente da situação, pelo sistema $S4$. O método fará uso das regras descritas na axiomatização da $\mathcal{LAP}_{\rightsquigarrow}$, visto na seção 5.1.1, dessas, falta apenas expressar as regras para a relação de dependêndia.

Se temos um P em um nó do tableau e se $\alpha \not\rightsquigarrow \neg P$, então colocamos o átomo P em todos os mundos acessíveis daquele nó pela ação α, se temos uma $wR_{\alpha}w'$ então P é propagado para w'. Essa é a regra SF[8]. Isso ainda não é o suficiente.

Supomos agora que temos $\alpha \not\rightsquigarrow P$ e $\beta \not\rightsquigarrow \neg P$, para algum átomo P e ações α e β. Então a fórmula $\langle\alpha\rangle P \wedge \langle\beta\rangle\neg P$ é $\mathcal{LAP}_{\rightsquigarrow}$ insatisfatível, mas a regra SF não é suficiente para fazer o tableau ser fechado. Uma solução foi adicionar uma nova regra chamada SB[9], ela permite propagar o átomo do novo mundo para o mundo que o criou.

- Regra SF: $P, S \bullet\!\!\underline{\quad\alpha\quad}\!\!\bullet\ S_1 \stackrel{\alpha}{\Longrightarrow} P, S \bullet\!\!\underline{\quad\alpha\quad}\!\!\bullet\ P, S_1$

- Regra SB: $S \bullet\!\!\underline{\quad\alpha\quad}\!\!\bullet\ P, S_1 \stackrel{\alpha}{\Longrightarrow} P, S \bullet\!\!\underline{\quad\alpha\quad}\!\!\bullet\ P, S_1$

O método de tableau para $\mathcal{LAP}_{\rightsquigarrow}$ é definido pelo uso do sistema de $S4$ (usando as regras K, T e 4) para o operador $\Box$ e da regra K para $[\alpha]$, que podem ser vistas no capítulo anterior. A relação de dependência é executada pelas regras SF e SB.

Seja Σ (esqueleto de tableau) uma relação ternária onde $\Sigma \subseteq (ACT \cup \Box) \times \mathbb{N} \times \mathbb{N}$. Os elementos ($\alpha$, n, n') dela são escritos como: $n \stackrel{\alpha}{\rightarrow} n'$, onde n é um inteiro chamado de índice, um índice é um mundo possível. E seja S o conjunto de fórmulas então $\langle S, \Sigma\rangle$ é a árvore de tableau e as regras da relação de dependência são:

- Regra SF: Se S contém (n, L), onde L é um literal, e $n \stackrel{\alpha}{\rightarrow} n' \in \Sigma$, e $\alpha \not\rightsquigarrow \neg L$, então adiciona (n', L) a $\langle S, \Sigma\rangle$;

[8]No original em inglês é chamada de *stepforward - rule*
[9]No original chamado de *stepback - rule*

- Regra SB: Se S contém (n', L), onde L é um literal, e $n \xrightarrow{\alpha} n' \in \Sigma$, e $\alpha \not\rightarrowtail L$, então adiciona (n, L) a $\langle S, \Sigma \rangle$;

5.3 Considerações

Finalmente temos todo o processo formalizado, agora, já temos um provador automático que é simples, modular e que, principalmente, mostra a árvore resultante do sistema de tableau para o cenário de entrada [47].

Com essas regras queremos executar a tarefa de planejamento. O provador automático como descrito nessa seção, precisa da seqüência de ações. Mas e se não tivermos essa seqüência? O presente trabalho tem como função mostrar um método de procurar a seqüência de ações que nos leva ao objetivo descrito, realizando assim a tarefa de planejamento. O próximo capítulo é dedicado a isto.

CAPÍTULO 6

PLANEJAMENTO UTILIZANDO A LINGUAGEM DE AÇÕES E PLANOS

Vamos mostrar nesse capítulo como encontrar um plano para problemas de planejamento em $\mathcal{LAP}_{\rightsquigarrow}$ utilizando o método de tableau descrito no capítulo 4, porém com algumas modificações. Primeiro falaremos sobre a teoria que envolve o método, depois demostraremos o exemplo do tiro de Yale e por fim daremos atenção à implementação do planejador.

Em assuntos relacionados a planejamento sempre encontraremos um agente e ações que este poderá executar para sintetizar um curso daquelas para alcançar o seu objetivo.

Para *planejar* precisamos de uma linguagem para representar o cenário, também são necessários as ações e o estado final, que é o objetivo do agente. Neste capítulo mostraremos como modelar problemas de planejamento em lógica. Note que, sendo uma das tarefas da área de raciocínio sobre ações, devemos considerar solução dentro desse contexto, discutidos no capítulo 2.

As primeiras abordagens em planejamento surgiram na década de 60 e tinham a lógica de primeira ordem como linguagem representativa. Cordell Green, em 1969, criou o método conhecido como "explicação de respostas" [23].

Esse método consistia em provar automaticamente um teorema pelo procedimento da resolução[1] e depois recuperar os passos para a obtenção da prova. Com isto podemos retirar a seqüência de ações usada no processo. No geral, a ordem da seqüência dependia da estratégia de prova adotada, já que a estratégia determinava quais cláusulas eram resolvidas ou fatoradas.

[1] A adaptação deste algoritmo para a lógica de primeira ordem está na base do mecanismo de inferência da linguagem Prolog. Informações sobre prova de teorema por resolução podem ser vistas em [45], [54] e [46]

O método de Green, por utilizar uma lógica de primeira ordem, estava sujeito ao problema de persistência e naquela época a solução para tal problema ainda implicava em métodos de prova ineficientes e incompletos. Ele precisava explicitar através de um axioma especial as relações que não eram afetadas por um operador. Isso gera uma quantidade de fórmulas extremamente grande, necessária para provar que todas as proposições que não são afetadas por um operador permanecem inalteradas.

Foi pensando nos problemas inerentes à lógica que, em 1971, surgiu o sistema *STRIPS*[2] proposto por Fikes e Nilsson [14]. Esta solução agregava uma linguagem de representação simples do cenário e um algoritmo de busca de espaço de estados com a função de encontrar em um deles o estado objetivo do cenário.

O algoritmo cria uma árvore onde os nós representam os estados do cenário e os arcos as ações. O arco que chega em um nó é a ação que criou esse estado enquanto os arcos que saem de um nó são todas as ações possíveis que podem ser executadas nesse estado. A simplicidade de STRIPS atraiu a comunidade de planejamento que até hoje o utiliza como base para pesquisas nessa área.

Foi criada em 1998 uma competição de planejadores [41], devido a diversas pesquisas na área desde as inovações mostradas pelos planejadores *Satplan* [29] e *Graphplan* [2]. Essa competição é realizada a cada dois anos desde de seu surgimento. O objetivo é comparar os planejadores vencendo quem conseguir resolver mais problemas em menos tempo. Para que fosse realizada era necessário ter uma linguagem única para descrever os problemas. Assim, surgiu a linguagem de definição de domínios de planejamento (*PDDL* [3] [40]).

A competição de planejadores (*IPC*) vem evoluindo a complexidade dos problemas abordados. Introduzindo problemas como restrições temporais (PDDL 2.1) que permitem modelar problemas do mundo real que envolvam tempo e restrições de recursos [17]; modelar fatos que se tornam verdadeiros ou falsos em um determinado tempo independente das ações (PDDL 2.2) [12]; entre outros.

[2]STRIPS significa Stanford Research Institute Problem Solver.

[3]Do inglês "Planning Domain Definition Language". É uma linguagem baseada em *STRIPS* e *ADL* (Linguagem de descrição de ações).

A competição foi uma forma de incentivar as pesquisas na área de planejamento, com isso surgiram diversos planejadores que utilizam os mais variados métodos, entre eles estão os algoritmos baseados em *Satisfação de Restrições* [29][30] *Busca Heurística* [4], *Redes de Petri* [50][51] e *Programação Inteira* [3] *Satisfabilidade* [29], dentre outros.

Porém, os planejadores utilizam como base de formalismo de representação o STRIPS, que é baseado em regras. Desse modo, problemas como persistência e ramificação não existem, por outro lado perde-se todo o poder de representação da lógica. Foi pensando nisso que desenvolvemos esse trabalho.

Planejamento em $\mathcal{LAP}_{\rightsquigarrow}$ é, basicamente, achar uma seqüência de ações usada para se fechar o tableau e capturar da árvore criada as regras de mudança de mundos utilizados no processo. Ao contrário do provador vamos obter mais informações da árvore gerada pelo tableau do que apenas "sim" ou "não".

Para provar um teorema no método de tableau, precisamos encontrar uma inconsistência em G (chamaremos desse modo a fórmula que queremos provar (*objetivo*)). Para saber se essa fórmula pertence a um dado cenário, o tableau deve ser fechado para $(BC \wedge LEIS) \rightarrow G$, onde BC é a nossa base de conhecimento e $LEIS$ são as leis de domínio. Também podemos tentar descobrir se a fórmula é válida depois da execução de uma ação(α). Para isso precisamos provar que $[\alpha]G$ é válida no cenário.

A nossa tarefa é justamente encontrar uma sequência de ações $[\alpha_0][\alpha_1]...[\alpha_n]$ que torne G válida no conjunto de fórmulas $BC \wedge LEIS$, para isso o tableau precisa ser fechado para $(BC \wedge LEIS) \rightarrow [\alpha_0][\alpha_1]...[\alpha_n]G$, obtendo assim o plano.

6.1 Aspectos teóricos do gerador de planos

A regra da possibilidade ($\Diamond$) é a responsável por criar novos sub-tableaux e possibilitar a mudança de mundos. O fato é que o plano desejado é uma seqüência finita de operadores $\langle\alpha\rangle$, onde α é uma ação. Nessa seção vamos justificar essa afirmação.

Um sub-tableau, também chamaremos de "mundo", é um conjunto de informações que representa um estado estático do nosso cenário ou sistema. As informações são fórmulas lógicas ou literais que não alteram seus valores em um estado. Quando ocorre

uma mudança de estado é construido um novo mundo para representá-lo. Essa mudança só ocorre pelo efeito de uma ação.

Supomos que temos G como nosso objetivo, e que queremos provar $(BC \wedge LEIS) \rightarrow \Diamond G$. Pelo propósito do operador $\Diamond$, sabemos que existe um plano para conseguir G. Vamos construir uma seqüência de ações (que chamaremos de π) onde $\vdash_{\mathcal{LAP}^{\rightsquigarrow}} (BC \wedge LEIS) \rightarrow \langle\pi\rangle G$.

Se toda transição do tableau somente ocorrer através do operador $\neg[\alpha]$ (e não por $\neg\Box$), então um plano pode ser achado pela composição de todas as ações que tem sido aplicadas desde o nó inicial até algum nó fechado.

Seja $\{\langle S^1, \Sigma^1\rangle, \langle S^2, \Sigma^2\rangle, ..., \langle S^n, \Sigma^n\rangle\}$ um conjunto de sub-tableaux e $\langle S^0, \Sigma^0\rangle$ o tableau inicial. Lembrando que toda construção de tableau é finito, queremos achar os sub-tableaux que são fechados para $BC \wedge LEIS \wedge \Box\neg G$ a partir do tableau inicial. Primeiro, retiramos todas as árvores que pode ser fechadas sem envolver a subfórmula $\Box\neg G$. Para cada árvore remanescente $\{\langle S^{i_1}, \Sigma^{i_1}\rangle, ..., \langle S^{i_n}, \Sigma^{i_n}\rangle\}$ nós aplicamos a fórmula $\Box\neg G$ na raíz e esperamos fechar os sub-tableaux remanescentes. Extraímos o caminho existente do tableau inicial até um sub-tableau fechado por $\neg G$. Este é o plano associado a essa árvore. Então o plano associado com o tableau é uma escolha não determinística $\pi_{i_1} \cup ... \cup \pi_{i_m}$ de todos os planos.

Mas, podemos construir um plano determinístico através de uma construção se-então-senão. Vamos construir uma árvore de tableau apenas da base de conhecimento, o tableau construído está correto porque independe da escolha da ação, se ele for fechado depois da aplicação da fórmula $\Box\neg G$ então temos um plano que não contém ação alguma. Caso contrário vamos escolher uma ação de $LEIS$ que aplicada na árvore do tableau resulta em apenas um nó aberto.

Esse fato é constatado porque a fórmula da *lei de efeito da ação* $(\Box(A \rightarrow [\alpha]B))$ ou a *lei de executabilidade da ação* $(\Box(A \rightarrow \langle\alpha\rangle\top))$, precisam que a sua pré-condição (A) seja satisfeita. Se a pré-condição for satisfeita então a ação é uma possível candidata para criar um novo mundo, senão devemos escolher outra ação.

Se a pré-condição de uma fórmula pertencentes a *lei de inexecutabilidade*

$(\Box(A \rightarrow [\alpha]\bot))$ for satisfeita devemos ignorar essa ação nesse momento porque esta não será executada agora, já que cria um mundo com o nó raiz inconsistente sem a aplicação de $\neg G$. Essa lei fecha o tableau sem que o nosso objetivo tenha sido satisfeito.

Quando uma ação é escolhida nós aplicamos a fórmula $\langle\alpha\rangle\top$, sendo α a ação candidata. Criando, dessa forma, um sub-tableau com arco α. O próximo passo é aplicar as regras K e 4 nas fórmulas pertencente a base de conhecimento. Também aplicaremos a regra K as fórmulas que pertencem à ação escolhida.

Para completar, precisamos aplicar a regra da relação de dependência. Para, assim, passar para o próximo mundo os literais que independem da ação escolhida.

Agora, estamos em um novo estado do cenário. Aplicamos a fórmula $\Box\neg G$ nesse mundo esperando contradizê-lo. Se, dessa forma, fecharmos esse sub-tableau, então recuperamos o caminho até o mundo inicial utilizando os arcos nomeados com as ações para obtermos o plano.

Mas como vamos saber se a ação escolhida é a certa ou se é o momento certo para ser executada? Não temos uma resposta para isso. Desconhecemos um artifício que nos dá certeza que devemos executar uma ação em um determinado momento. Fazemos tentativas até encontrarmos o plano ou pararmos o algoritmo.

Uma técnica que podemos utilizar é a "busca em largura" (breadth-first search), ou seja, em cada momento escolher todos as possíveis ações candidatas e criar um sub-tableau para cada uma delas, e continuar com esse processo até encontrarmos em algum ramo um tableau fechado pela aplicação de $\Box\neg G$.

Desse modo verificamos todas as possibilidades e em algum momento, se existir um plano, o encontraremos. Porém acabamos enfrentando um problema, devemos manter todo os ramos do tableaux na memória o que poderia ser muito extenso e impossibilitaria o uso computacional.

Outro método conhecido e que não causa problema de espaço computacional é a "busca em profundidade" (depth-first search). Criaríamos apenas um tableau em cada nível. Um nível é o conjunto de arco que conecta um tableau aos sub-tableaux resultante daquele. Mas, com essa técnica, não teremos certeza que encontraremos um sub-tableau

fechado pela aplicação de $\Box\neg G$.

Isso porque podemos retornar em algum momento a um estado alcançado anteriormente, gerando, dessa forma, um laço na busca pelo plano. Essa "repetição" do estado também pode ocorrer na "busca em largura", porém, nessa estratégia, ocorreria apenas uma perda de recurso, mas ainda seria possível encontrar o plano. O laço criado na estratégia de "busca em profundidade" nos impede de encontrar o estado que contém a fórmula objetiva pois encontraríamos em algum momento, nesse ramo, novamente o mesmo estado.

Uma solução é percorrer o caminho de cada sub-tableau até o primeiro tableau e comparar os estados deles entre si. Se encontrarmos um laço, a ação que o resultou é descartada nesse momento. Escolhemos a próxima ação candidata.

Escolhemos a técnica chamada de "busca em profundidade iterativa". Nesse método fazemos uma "busca em profundidade" com limite de alcance. Quando terminamos todas as possibilidades e não encontramos uma resposta para o nosso problema, o limite é incrementado e então reiniciamos o processo. Desse modo há uma perda de tempo mas em compensação economizamos espaço e temos a certeza que, se existir, encontraremos o plano.

Com esse método nós encontraremos o plano ótimo, ou seja, a menor seqüência de ações necessária para se alcançar o objetivo. Assim, o nosso plano é determinado com sendo o primeiro plano encontrado com o menor número de ações aplicadas. Nada impede que possamos continuar o processo e encontrar outras seqüências que também sejam válidas.

Outro problema que ocorre na geração de plano é quando possuímos mais de um nó aberto em um tableau, possivelmente pela aplicação da regra da disjunção($\vee$). Precisamos criar mais de um sub-tableau, um para cada nó aberto. O sub-tableau é ligado diretamente ao nó que o criou, recebendo os valores das regras aplicadas sobre este. É necessário criar um controle em cada nível, pois a ação para se criar novas árvores de tableau é a mesma.

Assim, se um sub-tableau for aberto, a fórmula escolhida está errada para

todos os sub-tableaux desse nível e devemos descartar a ação nesse momento. Outra condição necessária é que a pré-condição da ação deve ser satisfeita[4] em todo nó aberto e em todo sub-tableaux.

Devido a regra da disjunção, pode ocorrer que cada sub-tableaux no mesmo nível contenha literais e leis diferentes uns dos outros, se esse for o caso o estado do sistema é formado pela disjunção dos sub-tableaux desse nível. Esse controle é complexo com o tamanho da árvore criado pelos tableaux. Uma solução é adiar ao máximo a regra da disjunção, podemos inclusive criar um novo sub-tableau sem que tenhamos aplicado a regra "$\vee$" na base de conhecimento ou leis do tableau anterior.

Outra solução é ignorar os nós abertos e se preocupar apenas com um ramo até que este esteja fechado, então voltamos ao nó aberto e aplicamos a mesma seqüência de ações que fechou aquele ramo. Se este não for fechado então a seqüência é inválida.

6.1.1 Exemplo do gerador de planos no cenário do Tiro em Yale

Vamos mostrar nessa subseção o gerador de planos para $\mathcal{LAP}_{\rightsquigarrow}$ aplicado no cenário que conhecemos como o "Tiro em Yale".

Utilizaremos a descrição mostrada na seção 5.1.2, onde temos a base de conhecimento, as leis de domínios e as relações de dependência. O nosso objetivo é matar o peru, então a fórmula *objetiva* é $\neg Vivo$.

Queremos encontrar uma seqüência de ação que transforme o estado inicial em um estado onde $\neg Vivo$ seja válido. Vamos iniciar o processo construindo o estado inicial formado pela base de conhecimento ($\neg Carregado$ e $Vivo$), pelas $LEIS$ e, também, da negação da fórmula *objetiva*$(\Box\neg(\neg Vivo))$[5].

Aplicando as regras clássicas do tableau na base de conhecimento obtemos apenas um nó aberto, nesse caso não teremos o problema da disjunção. A aplicação da regra T na fórmula *objetivo* não causa inconsistência, então é necessário fechar o tableau

[4]Não é necessário que seja a mesma fórmula da ação, por exemplo: podemos possuir duas fórmulas de uma mesma ação, $\Box(A \rightarrow [\alpha]B)$ e $\Box(C \rightarrow [\alpha]D)$. Como são a mesma ação($\alpha$), uma fórmula pode ser satisfeita em um nó aberto e a segunda fórmula em outro.

[5]Lembramos que o tableau é um método por refutação

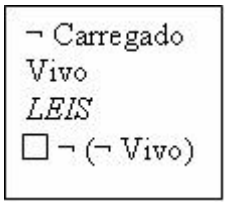


Figura 6.1: Estado inicial do sistema

em um outro mundo.

O próximo passo é separar as ações candidatas que deverão criar um sub-tableau. Nesse exemplo, temos três: *esperar*, *carregar* e *atirar*. Todas podem ser executadas devido a lei de executabilidade($\Box\langle esperar\rangle\top$, $\Box\langle carregar\rangle\top$ e $\Box\langle atirar\rangle\top$).

Devemos escolher uma e usaremos como critério para isso, a ordem. A primeira tentativa será a ação *esperar*. Separamos, então, as fórmulas dessa ação e a relação de dependência e criamos um novo mundo acessível pelo arco intitulado por *esperar*.

O exemplo não possui nenhuma fórmula com essa ação nem relações de dependência, então todos os literais desse estado independem da ação o que resulta em um novo estado igual ao anterior. Dessa forma, teremos um laço que indica que podemos ignorar essa ação nesse momento.

A próxima tentativa será com a ação *carregar*. Separamos as suas fórmulas e relação de dependência. A única fórmula($\Box[carregar]Carregado$) pode ser executada visto que não possui pré-condição. Aplicando as regras K e SF da relação de dependência teremos o seguinte estado $Carregado$ e $Vivo$.

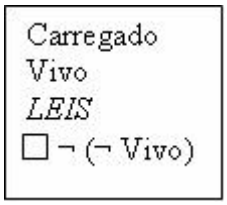


Figura 6.2: Representação do sub-tableau formado pela ação *carregar*.

A aplicação da fórmula *objetiva* ainda não fecha o tableau. Devemos, então, continuar com o processo. O algoritmo, nesse ponto, faria uma comparação do nível do ramo da ávore com o limite de profundidade, se for maior ou igual descartaríamos essa ação e retornaríamos ao tableau anterior, utilizando a próxima ação para pesquisa. Como

isso é um exemplo vamos continuar desse ponto.

Devemos, novamente, selecionar as ações candidatas, contudo todas podem ser executadas devido às leis de executabilidade. Pelo mesmo critério escolhemos *esperar*. Como essa ação não tem nenhuma fórmula e nenhuma relação de dependência ela também criará um laço nesse momento. Descartando-a, passaremos para próxima.

A ação *carregar* pode ser executada novamente, já que sua fórmula não possui pré-condição. Porém, agora, essa ação causa um laço, porque cria novamente o estado $Carregado$ e $Vivo$.

Sobra, apenas, a ação *atirar*. Nesse estado, podemos executar duas fórmulas, $\Box([atirar]\neg Carregado)$ e $\Box(Carregado \rightarrow [atirar]\neg Vivo)$. A segunda fórmula pode ser executada porque $Carregado$ é válido nesse estado. As regras $K[atirar]$ e SF resultam no novo mundo acessível pelo arco *atirar* formado pelo estado $\neg Vivo$ e $\neg Carregado$.

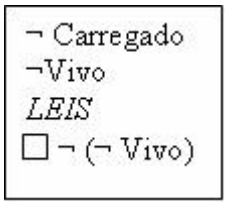


Figura 6.3: Representação do sub-tableau formado pela ação *atirar*.

Aplicando a fórmula $\Box\neg(\neg Vivo)$ encontramos inconsistência nesse estado o que indica que fechamos esse sub-tableau. Como essa árvore possui apenas uma ramificação conseqüentemente fechamos o tableau inicial. Os arcos que conectam um sub-tableau ao outro formam o nosso plano. Recuperando esse caminho temos *carregar* e *atirar*. Precisamos, então, executar primeiro *carregar* e depois a ação *atirar* para conseguir chegar ao nosso objetivo, que era matar o peru. Podemos ver o plano e os estados criados por cada uma das ações na figura 6.4.

6.2 Implementação do algoritmo de geração de planos

A busca pela seqüência de ações que leva ao objetivo do cenário utiliza os princípios do método de Tableau para lógica modal como visto no capítulo 4.

Figura 6.4: Plano formado pela seqüência de ações, *carregar* e *atirar*

Numa visão mais ampla, construimos o plano de "cima" para "baixo" (*Top-Down*), aplicando as ações nos tableaux. Se a pré-condição da ação for respeitada pelo estado do mundo. Então é criado um novo tableau com os efeitos dessa ação adicionada as proposições que não fazem parte da relação de dependência. É também, passado as fórmulas que contém o operador $\Box$ pela regra 4. Relacionamos, então, o novo mundo com o seu antecessor direto, essa relação deve ser marcada com o nome da ação que a criou(usamos a regra $\langle\alpha\rangle$ para criar o novo mundo). O processamento é passado para o novo mundo e repetiremos o laço até que algum sub-tableau criado contenha a fórmula *objetiva* do cenário.

A implementação do algoritmo contém quatro arquivos de organização do algoritmo. São eles: *Fórmula*, *Nó*, *Mundo* e *Ação*.

- O objeto *Fórmula* contém as fórmulas lógicas, é organizado pelo princípio das sub-fórmulas, comentadas na seção 3.1. Esse objeto possui uma propriedade *nome*, que é a semântica da fórmula; um *tipo* que representa o tipo da fórmula, dentre esses tipos estão os operadores lógicos, $\bot$, $\top$ ou uma proposição; Possui, também, dois ponteiros para conectar entre as suas sub-fórmulas.

- O objeto *Nó* representa um nó de um tableau. Cada objeto possui um conjunto de fórmulas lógicas; dois ponteiros para conectar entre os nós e um ponteiro para ligar ao objeto *Mundo*, se este nó criar um sub-tableau. Um novo objeto *Nó* é formado pela aplicação das regras de tableau em uma fórmula.

- O objeto *Mundo* representa o estado do sistema a cada execução de uma ação, um mundo contém um nó raiz, um nome e o arco que o conecta a um nó do tableau anterior.

- O objeto *Ação* contém todas as fórmulas de uma determinada ação e relação de dependência.

Nas próximas subseções falaremos sobre as rotinas responsáveis por encontrar o plano. Primeiro comentaremos sobre o arquivo de entrada que contém toda a informação do cenário. Depois sobra a classe que utilizamos como um banco de dados, fazemos o pré-processamento para melhorar o desempenho e armazenamos as informações em um objeto chamado *Base*. Por último, porém a mais importante, falaremos da classe responsável pelo método de tableau, executando suas regras e controlando o fluxo da escolha das ações.

6.2.1 Arquivo de entrada

O cenário de entrada está na linguagem $\mathcal{LAP}_{\rightsquigarrow}$[5] que tem como base a lógica modal. É composto de: uma fórmula conjuntiva que corresponde ao estado inicial e das leis de domínio; A fórmula conjuntiva do nosso objetivo; A relações de dependência entre as ações e literais.

No capítulo 5 nós falamos sobre os tipos de fórmulas que podemos encontrar na lógica de ações e planos. As leis de domínios, as leis de efeito de ação, leis de executabilidade, de inexecutabilidade, os objetivos e a base de conhecimento, formada especialmente por literais.

As proposições possuem a primeira letra maiúscula enquanto as ações contém a sua inicial minúscula, mas a principal diferença é que as ações só são encontradas entre colchetes ([ ]), cujo símbolo representa o operador de necessidade na lógica modal. Estes podem ser vazios, sem ação, para representar leis que são válidas em todos os mundos ($\Box$ é $S4$).

Passamos o arquivo em um analisador léxico que também transforma as

fórmulas do cenário, inclusive a fórmula objetiva, em objetos da classe *Fórmula*. O próximo passo é realizado pelo objeto *Base* que processa as fórmulas para facilitar o uso destas, para, então, criar o tableau inicial.

6.2.2 Criando a Base

Os objetos da classe *Fórmula* criadas no analisador léxico são armazenadas numa classe chamada *Base*. Um dos pré-processamentos é separar os literais da base de conhecimento das leis de domínio. Outro é separar e armazenar de forma organizada as ações. Por último montamos a fórmula *objetiva*, negando-a e acrescentando o operador $\Box$.

Na *Base* temos 2 listas que farão o suporte para o processamento da busca pelo plano, estão organizadas da seguinte forma:

I. Uma lista armazena as leis de domínios que não contém ação. Isso facilita para verificar restrições em um estado. Por exemplo, se temos uma restrição da forma $\Box(A \rightarrow B)$ podemos verificar se em todo estado criado essa restrição é satisfeita, caso contrário podemos ignorar o mundo que contém esse estado e continuar o processo com outra ação.

II. A lista seguinte é responsável por guardar as ações. A lista armazena um conjunto de objetos da classe *Ação*. Nessa lista estão armazenadas todas as ações do sistema, o objetivo é diminuir o tempo que precisamos para encontrar as ações.

Como as leis de domínios, as ações e a fórmula *objetiva* são formadas pelo operador $\Box$ e são redigidas pelo sistema $S4$ nós as aplicamos em todos os mundos possíveis, por essa razão nós as manteremos nesse objeto e as executamos em todos os mundos criados.

Criada a *Base*, o próximo passo é construir o mundo inicial com os literais da base de conhecimento. O processo é, então, passado para a última classe que trata do método de tableau para a busca do plano.

6.2.3 A classe Planning

A classe *Planning* contém o algoritmo responsável por procurar o plano. Interage com o objeto *Base* na aplicação das leis de domínios e na escolha da ação.

Utilizamos uma função recursiva responsável por gerar sub-tableaux e conseqüentemente, novos estados. Passamos como parâmetro o mundo atual, se o estado contém a fórmula *objetiva* ele será o nosso estado final, caso contrário devemos continuar com o processo a partir dele, isso se esse estado não gerar um laço no caminho até o estado inicial. A seguir, mostramos as verificações que ocorrem antes de escolher a ação que criará o próximo mundo. São elas:

I. Verifica se o estado do mundo já existe no ramo, ou seja, se existe um laço.

II. Verifica se a fórmula *objetiva* cria inconsistência no mundo, caso isso seja verdadeiro terminamos o processo e mostramos as ações que criarem cada um dos mundos pertencentes do mundo atual até o estado inicial. Gerando, assim, o plano.

III. Verifica se o processo não ultrapassou o limite de profundidade[6].

IV. Aplicamos as restrições do cenário nesse estado, elas são fórmulas pertencentes às leis de domínio mas que não contém ações. Devem ser satisfeitas no estado para que possamos continuar com o processo a partir desse mundo.

Se as três primeiras verificações resultarem em resposta negativa e a última for positiva então nos encontramos em um mundo intermediário, a próxima etapa é encontrar uma ação que possa ser executada nesse estado. Para cada ação existente verificamos se a sua pré-condição é satisfeita no mundo atual. Aplicamos as regras de tableau na fórmula da ação, se o resultado for apenas um nó aberto então encontramos uma ação candidata para criar o próximo mundo. Devemos verificar, também, se existem outras fórmulas da mesma ação que possam ser executadas. O novo estado será composto pela conjunção dos efeitos dessas fórmulas e dos literais que independem dessa ação.

[6]O tamanho da profundidade também pode ser visto como o tamanho do ramo da árvore.

Conectamos o novo mundo com o nó criador e relacionamos esse arco com o nome da ação que o criou. O processo se repete até que encontramos um estado inconsistente pela negação da fórmula *objetiva* ou que todas as possibilidades tenham sido aplicadas e no caso não exista um plano que satisfaça o cenário. Outro critério de parada é colocar um limite de profundidade ou um limite de tempo.

O algoritmo responsável por achar o plano em lógica de ação e planos pode ser visto a seguir:

Algoritmo 6.2.1

```
ConstruirPlano(Mundo* mundo, inteiro profundidade)
{
   TerminouTempo();      //Verifica se ultrapassou um limite pré escolhido de tempo.
   se (mundo→AchouLaço()) retorna;      //Verifica é um laço.
   se (!mundo→AchaRestrições(base)) retorna;      //Verifica se as restrições das leis de domínio são satisfeitas.
   AchaPlano(mundo);      //Se o mundo for fechado pela aplicação da negação da fórmula objetiva, então essa rotina termina o programa criando, como saída, um grafo que mostra o plano.
   se (profundidade > limite) retorna;      //Verifica se o tamanho do ramo é maior que o limite atual
   it_ação = base→pega_ação().inicio();
   enquanto ( it_ação != base→pega_ação().fim())      //Procura as ações
   {
      No *no = Vazio;
      no = AchaFormula(it_ação, mundo)      //Procura as fórmulas da ação que são satisfeitas no mundo, retornando o único nó aberto.
      se (no != Vazio)
      {
         Formula *novoestado = Vazio;
         novoestado = CriaNovoEstado(it_ação, no);      //Cria uma fórmula conjuntiva com os efeitos da ação e os literais pertencente á "no" que não depende da ação(relação de dependência).
         Mundo* novomundo = novo Mundo();
         novomundo→CriaEstado(novoestado);      //Cria um novo estado
         novomundo→adiciona_ação(&(*it_ação));      //Armazena o nome da ação que o criou
         novomundo→adiciona_arco(&(*no));      //Se relaciona com o nó que o criou
         ConstuirPlano(novomundo, profundidade+1);      //Recursão
      }
      ++it_ação;
   }
```

Podemos analisar o nosso algoritmo em função do número de ações, chamaremos de n. No pior caso executamos todas as possibilidades e não encontramos um plano. Do estado inicial podemos criar n novos mundos e no nível seguinte teremos n^2 mundos, no próximo n^3 e sucessivamente, desse modo, é facil perceber que a complexidade do algoritmo dado é exponencial. A busca de laços e verificação de restrições pode diminuir o tempo.

6.3 Testes e resultados

Existem poucos cenários na literatura de raciocínio sobre ações, assim nós transformamos alguns problemas da competição de planejamento de 2000 e 2002 para a lógica de ações e planos. Queremos, com isto, fazer testes e medir o desempenho do planejador.

Fizemos uma rotina para transformar os domínios em PDDL para $\mathcal{LAP}_{\rightsquigarrow}$. Foi preciso instânciar os parâmetros dos predicados e das ações além de adicionar os literais negativos, já que o formalismo de PDDL trabalha com a hipótese do mundo fechado. Criamos também as relações de dependência de cada ação.

Antes de mostrar os resultados, vamos comentar como eles estão organizados. Fizemos uma tabela mostrando o problema onde o primeiro número do nome indica a quantidade de blocos; o primeiro marcador é o tempo gasto para realizar a tarefa de planejamento, colocamos um limite de 3.600 segundos que equivale a 1 hora; o próximo é a quantidade de mundos que o algoritmo gera durante o processo; o "Nível" significa o limite da "profundidade iterativa" que alcançamos, quando o plano é encontrado esse valor também indica o número mínimo de ações para achá-lo; e por fim, em "Plano" um "X" indica que um plano foi encontrado.

Vamos apresentar os domínios utilizados como testes e os dados que obtivemos de cada um. Começaremos com o domínio do mundo de blocos, conhecido como *BlocksWorld*. Consiste em um número arbitrário de blocos empilhados sobre uma superfície, onde cada bloco pode ter no máximo um outro bloco sobre si e também estar sobre um único bloco ou sobre a superfície. Uma garra captura um bloco que não esteja

obstruído e o empilha sobre outro bloco ou sobre a mesa.

Problema	**Tempo**	**Mundos**	**Nível**	**Plano**
probBLOCKS-4-0	0,14000000059604644775	638	5	X
probBLOCKS-4-1	0,10000000149011611938	478	9	X
probBLOCKS-4-2	0,090000003576278686523	439	5	X
probBLOCKS-5-1	4,0399999618530273438	11.145	9	X
probBLOCKS-5-2	19,809999465942382812	53.079	15	X
probBLOCKS-6-0	31,879999160766601562	57.551	11	X
probBLOCKS-6-1	193,4499969482421875	353.187	9	X
probBLOCKS-6-2	3.601	$>$ 5 milhões	19	
probBLOCKS-7-0	3.601	$>$ 4 milhões	18	
probBLOCKS-7-1	3.601	$>$ 4 milhões	15	
probBLOCKS-7-2	3.601	$>$ 4 milhões	15	
probBLOCKS-8-0	3.601	$>$ 3 milhões	11	
probBLOCKS-8-1	3.601	$>$ 3 milhões	11	
probBLOCKS-8-0	3.601	$>$ 3 milhões	11	
probBLOCKS-9-0	3.601	$>$ 2 milhões	15	
probBLOCKS-9-1	3.601	$>$ 2 milhões	17	
probBLOCKS-9-2	3.601	$>$ 2 milhões	15	
probBLOCKS-10-0	3.601	$>$ 1 milhão	15	
probBLOCKS-10-1	3.601	$>$ 1 milhão	13	
probBLOCKS-10-2	3.601	$>$ 1 milhão	15	
probBLOCKS-11-0	3.601	$>$ 1 milhão	11	
probBLOCKS-11-1	3.601	$>$ 1 milhão	9	
probBLOCKS-11-2	3.601	$>$ 1 milhão	13	
probBLOCKS-12-0	3.601	924.554	12	
probBLOCKS-12-1	3.601	982.980	13	

probBLOCKS-13-0	3.601	735.766	11	
probBLOCKS-13-1	3.601	710.799	13	
probBLOCKS-14-0	3.601	629.019	11	
probBLOCKS-14-1	3.601	611.440	8	
probBLOCKS-15-0	3.601	538.195	8	
probBLOCKS-15-1	3.601	431.304	13	
probBLOCKS-16-1	3.601	357.854	10	
probBLOCKS-16-2	3.601	340.506	13	
probBLOCKS-17-0	3.601	362.668	7	

O próximo domínio que trabalhamos foi o chamado *Elevator*, ele simula a operação de um elevador em um edifício com uma determinada quantidade de pessoas e de andares. Cada passageiro tem um andar de origem e de destino. As ações consistem na entrada e saída de pessoas no elevador e na movimentação do elevador para cima e para baixo, um andar por ação. Os resultados foram:

Problema	**Tempo**	**Mundos**	**Nível**	**Plano**
s1-0	0,0	14	3	X
s1-1	0,0	10	2	X
s1-2	0,0	14	3	X
s1-3	0,0	14	3	X
s1-4	0,0	14	3	X
s2-0	0,43999999761581420898	1.645	6	X
s2-1	0,25999999046325683594	997	6	X
s2-2	0,25	968	6	X
s2-3	0,31999999284744262695	1.291	6	X
s2-4	0,23999999463558197021	922	6	X
s3-0	2.972,35009765625	475.465	9	X

s3-3	2.842,219970703125	442.767	9	X
s3-4	2.962,489990234375	500.738	9	X

O domínio *Logistics* simula o problema de transportar cargas entre localidades. Podem existir várias localidades dentro de uma cidade que também pode ter uma quantidade variada no problema. Existem caminhões que transportam as cargas entre as localidades da mesma cidade e aviões que podem se movimentar entre aeroportos, que é classificada como uma localidade especial, independente da cidade.

Problema	**Tempo**	**Mundos**	**Nível**	**Plano**
probLOGISTICS-4-0	3.601	$>$ 6 milhões	7	
probLOGISTICS-4-1	3.601	$>$ 6 milhões	7	
probLOGISTICS-4-2	3.601	$>$ 5 milhões	7	
probLOGISTICS-5-0	3.601	$>$ 6 milhões	7	
probLOGISTICS-5-1	3.601	$>$ 5 milhões	7	
probLOGISTICS-5-2	1929,4300537109375	$>$ 3 milhões	7	X
probLOGISTICS-6-0	3.601	$>$ 6 milhões	7	
probLOGISTICS-6-1	3.601	$>$ 5 milhões	7	
probLOGISTICS-6-2	3.601	$>$ 6 milhões	7	
probLOGISTICS-6-3	3.601	$>$ 5 milhões	7	
probLOGISTICS-7-0	3.601	$>$ 1 milhão	5	
probLOGISTICS-7-1	3.601	$>$ 1 milhão	5	
probLOGISTICS-8-0	3.601	$>$ 1 milhão	5	
probLOGISTICS-8-1	3.601	$>$ 1 milhão	5	
probLOGISTICS-9-0	3.601	$>$ 1 milhão	5	
probLOGISTICS-9-1	3.601	$>$ 1 milhão	5	
probLOGISTICS-10-0	3.602	639.332	4	
probLOGISTICS-10-1	3.602	643.683	4	

Outro teste é o domínio do *DriverLog*. É similar ao *Logistics* mas com uma complicação, os caminhões requerem motoristas que podem estar em uma localidade diferente daquele, devendo caminhar entre as localidades para encontrá-lo e dirigí-lo.

Problema	**Tempo**	**Mundos**	**Nível**	**Plano**
driverlog1	149,7899932861328125	141.574	6	X
driverlog2	3.601	$>$ 1 milhão	7	
driverlog3	3.601	$>$ 1 milhão	7	
driverlog4	3.601	$>$ 1 milhão	6	
driverlog5	3.601	$>$ 1 milhão	6	
driverlog6	3.601	$>$ 2 milhões	5	
driverlog7	3.601	$>$ 1 milhão	5	
driverlog8	3.601	$>$ 1 milhão	5	
driverlog9	3.601	234.903	6	
driverlog10	3.604	97.365	4	
driverlog11	3.609	48.953	5	
driverlog12	3.653	9.498	4	

A quantidade de estados criados é um fator que prejudica o desempenho, porém não é o único. Podemos ter um número pequeno de mundos criados e ainda não alcançar um nível de profundidade alta, isso porque o número de literais e ações é muito grande e se gasta muito tempo para encontrar uma ação cuja pré-condição seja satisfeita no estado. Os testes de *Logistics* e *DriverLog* demonstram isso.

Fazemos uma busca linear na lista de ações verificando se o estado satisfaz a pré-condição. Essa verificação é cara quando o número de literais existentes em um estado for grande.

O laço, ou estado que já tenha sido alcançado durante um percurso, também é outro fator que prejudica o desempenho do algoritmo. Para cada novo mundo devemos percorrer o ramo deste até o mundo inicial com o intuito de procurar um estado igual. Se não encontrarmos perdemos tempo comparando. Quanto maior o nível de profundidade

maior o número de comparações, porém com maior probabilidade de se encontrar um laço.

6.4 Estratégia *Top-Down* vs *Bottom-Up*

As primeiras pesquisas foram voltadas para a estratégia *Bottom-Up*, isso quer dizer que começávamos a nossa busca pelo estado final, ou seja, utilizávamos a fórmula *objetiva* como estado e a busca era voltada para encontrar o estado inicial.

Dessa forma, nós não tínhamos o estado completo, só possuíamos os literais que faziam parte da fórmula *objetiva*, uma vantagem é que trabalhávamos com menos informação, mas a desvantagem era que para contradizer esse estado apenas era necessário um literal do efeito da ação.

Apesar de trabalhar com menos informações e, desse modo, menos tempo de busca e comparação, criávamos um quantidade maior de mundos. O que nos levou a considerar que essa estratégia se tornaria mais dispendiosa. Isso porque não possuíamos o estado anterior. Se o literal não pertencia a relação de dependência e o seu complemento estava contido no estado atual então podíamos satisfazer o estado atual com uma quantidade maior de ações.

Por essa razão escolhemos a estratégia *Top-Down*, onde começamos a busca pelo estado inicial até encontrar um estado onde a fórmula *objetiva* o contradiz. A desvantagem é que com maior número de informações, já que trabalhamos com o estado completo, maior o tempo de busca. Porém com menor quantidade de mundos.

Outra diferença relevante entre as duas estratégias é que na *Bottom-Up* o estado deveria ser satisfeito pelo efeito da ação e pelos literais que não pertencem à relação de dependência e o novo estado era formado com os literais da sua pré-condição e, também, incluía os literais independentes da ação. Enquanto na *Top-Down* é o contrário, usamos a pré-condição para satisfazer o estado e formamos o novo com os literais do efeito da ação e da relação de independência.

6.5 Considerações

Nesse trabalho adaptamos o método de tableau para procurar a seqüência de ações realizando a tarefa de planejamento. Podemos, também, orientar o algoritmo com uma seqüência específica de ações, desse modo, realizamos a tarefa da predição, cujo o objetivo é o estado final.

A tarefa da explicação pode ser conseguida utilizando a estratégia *Bottom-Up*. Com o estado final e uma seqüência específica podemos alcançar o estado inicial. Realizando todas as tarefas da área de raciocínio sobre ações.

Nesse capítulo mostramos um processo de se descobrir um plano para $\mathcal{LAP}_{\rightsquigarrow}$ que inclui as soluções para os problemas de persistência e ramificação. Para isso, foi preciso criar um controle no método de tableau que cria uma seqüência deterministica de ações.

CAPÍTULO 7

CONCLUSÃO

Conseguimos mostrar que é possível desenvolver um planejador para uma lógica proposicional que possua soluções para os problemas da persistência e ramificação. O presente trabalho resgatou as pesquisas na área de planejamento que foram ignoradas desde 1971, com o surgimento de STRIPS [14].

Devido a facilidade em se trabalhar com a representação STRIPS, que é um formalismo baseado em regras, as pesquisas em planejamentos apenas se preocuparam com soluções em tempo de relógio. Nossa meta foi utilizar uma representação baseada em lógica que possui a característica de ser completa e adequada além de ser monotônica. A última pesquisa conhecida foi descrita por Green em 1969 [23].

O nosso trabalho foi facilitado pela simplicidade de expressão da lógica de ações e planos. Outro fator que contribuiu para o resultado foi o provador automático de teorema que utiliza o método de tableau para lógica modal [6], descrito no capítulo 4, e a implementação do mesmo [47].

Utilizamos o método de tableau para construir um planejador simples e prático para $\mathcal{LAP}_{\rightsquigarrow}$. Diferente do provador de teoremas, precisamos além das regras que simulam os axiomas da lógica modal, controlar de forma organizada o método de tableau.

Comentamos que para resolver o problema é preciso supor que exista um operador $\langle\alpha\rangle$ para que um novo mundo seja criado. As regras do método de tableau farão o transporte das fórmulas para esse novo mundo.

Quanto ao problema da persistência inerente à lógica [39], $\mathcal{LAP}_{\rightsquigarrow}$ possui uma solução chamada de relação de dependência [7], essa solução também está presente no provador de teoremas [6] e conseqüentemente no nosso trabalho.

Também falamos dos problemas que encontramos no processo e o modo como os solucionamos e de como a explosão de possibilidades de mundos criados prejudicaria

a criação de um planejador prático. Outro incomodo é se trabalhar com o operador disjuntivo, que criaria diferentes sub-tableaux que possuem o mesmo antecessor, para isso é preciso verificar em cada nó folha do tableau se a mesma ação pode ser aplicada.

Concluímos nosso trabalho mostrando os resultados que obtivemos de alguns testes feitos a partir de problemas inicialmente descritos em PDDL, que transformamos em representação $\mathcal{LAP}_{\rightsquigarrow}$. Justificamos o nosso trabalho com o propósito de mostrar que podemos usar representação em lógica para problemas de planejamento.

Trabalhos futuros podem vir a melhorar o método para solucionar o problema do tempo e tornar o planejador competitivo com os atuais, porém em uma representação melhor que o PDDL. Uma proposta é utilizar uma busca heurística para guiar as escolhas das ações. Outra, é encontrar um modo de determinar a quantidade mínima de ações para gerar o plano diminuindo o tempo gasto pela iteração do limitador da "busca em profundidade iterativa". Dentre outras possibilidades de diminuir o consumo de tempo.

Também pode-se utilizar de outros métodos para procurar planos em $\mathcal{LAP}_{\rightsquigarrow}$ que diminuam a quantidade de estados gerados, porém, os mesmos, devem possuir mecanismos para se trabalhar com a relação de dependência.

Outro interessante trabalho é aumentar o poder de representação da lógica de ações e planos, como incrementar o uso de tempo e de operadores matemáticos. Dessa forma podemos substituir o uso do formalismo PDDL como forma de representação de problemas por uma lógica simples, completa, adequada e monotônica.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] E. W. Beth. On padoa's method in the theory of definition. *Inda. Math.*, 15:330–339, 1953.

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

[3] Alexander Bockmayr e Yannis Dimopoulos. Mixed integer programming models for planning problems. Jeremy Frank e Mihaela Sabin, editors, *Proceedings of the Workshop on Constraint Problem Reformulation (CP-98)*, páginas 1–6, Pisa, Italy, 1998. NASA Ames Research Center.

[4] Blai Bonet e Hector Geffner. Planning as heuristic search: New results. *ECP '99: Proceedings of the 5th European Conference on Planning*, páginas 360–372, London, UK, 2000. Springer-Verlag.

[5] Marcos A. Castilho. *Modèles logiques pour le raisonnement sur les actions*. Tese de Doutorado, Université Paul Sabatier, ToulousEine Interpretation des intuitionistischen Aussagenkalkuls - FRANCE, oct de 1998. In french.

[6] Marcos A. Castilho, Luis Farinas del Cerro, Olivier Gasquet, e Andreas Herzig. Modal tableaux with propagation rules and structural rules. *Fundamenta Informaticae*, 32(3-4):281–297, 1997.

[7] Marcos A. Castilho, Olivier Gasquet, e Andreas Herzig. Formalizing action and change in modal logic I: the frame problem. *Journal of Logic and Computation*, volume 9(5):701–735, 1999.

[8] Marcos A. Castilho, Andreas Herzig, e Camilla Schwind. Raisonnement sur les actions: les approches basÃ©es sur la causalité et la dépendance. Robert Jeansoulin, editor, *Nouveaux défis en Sciences de l'Information: Documents et Evolution*. Cepadues, Toulouse, sep de 2000.

[9] Brian F. Chellas. *Modal Logic: an introduction*. Cambridge University Press, 1980.

[10] Haskell B. Curry. The elimination theorem when modality is present. *The Journal of Symbolic Logic*, 17(4):249–265, 1952.

[11] George E. Hughes e Max J. Cresswell. *An introduction to modal logic*. Methuen and Co. Ltd., London, 1968.

[12] Stefan Edelkamp e JH̃offmann. Relatório técnico.

[13] Erwin Engeler. *Algorithmic Properties of Structures: Selected Papers of E. Engeler*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1993.

[14] Richard Fikes e Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *IJCAI*, páginas 608–620, 1971.

[15] J. J. Finger. *Exploiting constraints in design synthesis*. Tese de Doutorado, Stanford, CA, USA, 1987.

[16] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel, Dordrecht, 1983.

[17] Maria Fox e Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[18] Michael Gelfond e Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17(2, 3 e 4):301–321, 1993.

[19] Matthew L. Ginsberg e David E. Smith. Reasoning about action II: The qualification problem. Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, páginas 259–287, Los Altos, California, 1987. Morgan Kaufmann.

[20] Kurt Godel. Zum intuitionistischen aussagenkalkul. K. Berka e L. Kreiser, editors, *Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik (vierte Auflage)*, páginas 199–200. Akademie-Verlag, Berlin, 1986.

[21] Robert Goldblatt. Mathematical modal logic: a view of its evolution, 2000.

[22] Rajeev Gore. Tableau methods for modal and temporal logics. Relatório Técnico TR-ARP-15-95, Australian National University, nov de 1995.

[23] Cordell Green. An application of theorem proving to problem solving. *IJCAI-69—Proceedings of the 1st International Joint Conference on Artificial Intelligence*, páginas 219–239. Washington, DC, May de 1969.

[24] S. Hanks e D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. páginas 390–395, 1987.

[25] K. J. J. Hintikka. Form and content in qualification theory. *Acta Philosophica Fennica*, 8:3–55, 1955.

[26] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 26(1):53–56, 1983.

[27] M. Kalsbeek. Gentzen systems for logic programming styles, 1994.

[28] S. Kanger. *Provability in Logic*. Stockholm Studies in Philosophy, University of Stockholm, Almqvist and Wiksell, Sweden, 1957.

[29] Henry Kautz e Bart Selman. Planning as satisfiability. *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, páginas 359–363, New York, NY, USA, 1992. John Wiley & Sons, Inc.

[30] Henry A. Kautz e Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. *AAAI/IAAI, Vol. 2*, páginas 1194–1201, 1996.

[31] Saul A. Kripke. A completeness theorem in modal logic. *jsl*, 24:1–15, 1959.

[32] E. J. Lemmon. New foundations for lewis modal systems. *J. Symb. Log.*, 22(2):176–186, 1957.

[33] Clarence Lewis. *A Survey of Symbolic Logic*. University of California Press, 1918. Republished by Dover, 1960.

[34] Clarence Lewis. Strict implication, an emendation. *Journal of Philosophy, 17*, 1920.

[35] Vladimir Lifschitz. Frames in the space of situations (research note). *Artif. Intell.*, 46(3):365–376, 1990.

[36] Fabio Massacci. Strongly analytic tableaux for normal modal logics. Alan Bundy, editor, *Proceedings of the Twelfth International Conference on Automated Deduction (CADE'94)*, volume 814, páginas 723–737, Berlin, 1994. Springer-Verlag.

[37] J. McCarthy. Epistemological problems of artificial intelligence. páginas 46–52, 1987.

[38] John McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986. Reprinted in.

[39] John McCarthy e Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. B. Meltzer e D. Michie, editors, *Machine Intelligence 4*, páginas 463–502. Edinburgh University Press, 1969. Reprinted in.

[40] D. McDermott. Pddl — the planning domain definition language, 1998.

[41] D. McDermott. Planning competition results, 1998.

[42] M. Ohnishi e K. Matsumoto. Gentzen method in modal calculi i. *Osaka Mathematical Journal*, 9:113–130, 1957.

[43] Sally PopKorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[44] W. Rautenberg. *Klassische und Nichtklassische Aussagenlogik*. Vieweg, Wiesbaden, 1979.

[45] George. A. Robinson. The present state of mechanical theorem proving. *a paper presented at the Fourth Systems Symposium*, páginas 19–20, November de 1994.

[46] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1).

[47] Roberta Vanessa Rojo. Uma implementação genérica para métodos de tableau modais com uma aplicação específica. Dissertação de Mestrado, Depto. Informática — Universidade Federal do Paraná, Curitiba, 2003.

[48] Erik Sandewall. *Features and fluents (vol. 1): the representation of knowledge about dynamical systems*. Oxford University Press, Inc., New York, NY, USA, 1994.

[49] Murray Shanahan. Solving the frame problem. a mathematical investigation of the common sense law of inertia. páginas 1186–1188, 1998.

[50] Fabiano Silva. *Redes de Planos: Uma proposta para a solução de problemas de planejamento em inteligência artificial usando Redes de Petri*. Tese de Doutorado, CEFET, Fevereito de 2005.

[51] Fabiano Silva, Marcos A. Castilho, e Luis Allan Kunzle. Petriplan: A new algorithm for plan generation (preliminary report). *IBERAMIA-SBIA '00: Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI*, páginas 86–95, London, UK, 2000. Springer-Verlag.

[52] R. M. Smullyan. *First-order logic*. Springer Verlag, Berlim, 1968.

[53] Michael Thielscher. Computing ramifications by postprocessing. Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, páginas 1994–2000, San Francisco, 1995. Morgan Kaufmann.

[54] Lawrence Wos, George A. Robinson, e Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, 1965.

[55] J. Yanov. On equivalence of operator schemes. 1:1–100, 1993.

[56] J. J. Zeman. *Modal Logic: The Lewis-Modal Systems*. Oxford University Press, 1973.