ROBERTA VANESSA ROJO

UMA IMPLEMENTAÇÃO GENÉRICA PARA MÉTODOS DE TABLEAUX MODAIS COM UMA APLICAÇÃO ESPECÍFICA EM RACIOCÍNIO SOBRE AÇÕES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática pelo Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Marcos Alexandre Castilho

CURITIBA

2003

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna *Roberta Vanessa Rojo*, avaliamos o trabalho intitulado. *"Uma Implementação Genérica Para Métodos de Tableaux Modais Com Uma Aplicação Específica em Raciocínio Sobre Ações"*, cuja defesa foi realizada no dia 27 de fevereiro de 2003, às quatorze horas, no Auditório da Informática da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 27 de fevereiro de 2003.

Prof. Dr. Marcos Alexandre Castilho
DINF/UFPR (Orientador)

Profª. Dra. Renata Wassermann
IME/USP

Prof. Dr. José Carlos Cifuentes
DMAT/ET/UFPR

Prof. Dr. Jair Donadelli Jr.
DINF/UFPR

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

TERMO DE APROVAÇÃO

ROBERTA VANESSA ROJO

UMA IMPLEMENTAÇÃO GENÉRICA PARA MÉTODOS DE TABLEAUX MODAIS COM UMA APLICAÇÃO ESPECÍFICA EM RACIOCÍNIO SOBRE AÇÕES

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná pela seguinte banca examinadora:

Orientador: Marcos Alexandre Castilho
Departamento de Informática, UFPR

Profa. Dra. Renata Wasserman
Instituto de Matemática e Estatística, USP

Prof. Dr. José Carlos Cifuentes
Departamento de Matemática, UFPR

Prof. Dr. Jair Donadelli Jr.
Departamento de Informática, UFPR

"Dedico este trabalho à minha mãe Ivone por ser meu referencial como Profissional na Educação. Ao meu pai Beto que, ao me alfabetizar, deu o pontapé inicial para eu chegar até aqui. E ao meu noivo Walter que, com sua fé em minha capacidade, amor e paciência, foi, como sempre, meu grande companheiro."

Agradecimentos

Aos meus pais, à Bá, ao Caco e à Fran, pelo amor e apoio que recebo sempre; à Bianca e ao Claudinho por iluminarem meus dias e ao meu noivo Walter, pelo carinho e compreensão, principalmente nos momentos difíceis.

Ao Marcos, que além de me orientar brilhantemente, tornou-se um grande amigo pelos ensinamentos, pela atenção e pela compreensão nos momentos de dificuldade no desenvolvimento deste trabalho.

Ao professor Décio Krause, por me apresentar as muitas possibilidades no estudo das lógicas não clássicas e por despertar em mim a vontade de pesquisar e procurar cada vez mais aprimorar meu conhecimento.

À tia Elza, pela acolhida e ao Cássio, por estar comigo nesta caminhada.

Aos amigos conquistados no mestrado: Clodis, Eduardo e Glória pela presença e carinho.

Aos meus veteranos Aldri, Fabiano, Ivan, Paulo e Tiago pelo companheirismo, dicas e socorros.

Aos professores Alexandre e Aurora, pelas palavras de estímulo nos momentos de desânimo.

Aos colegas da UNIVEL, principalmente ao grande amigo Aníbal e ao Heriberto pelo incentivo e por entender as minhas ausências; também por permitir meu afastamento, ainda que por pouco tempo, para me dedicar integralmente à pesquisa.

Ao Programa de Fomento à Pós-Graduação (PROF) e ao Programa de Pós-Graduação em Informática da UFPR, pelo apoio financeiro e oportunidade a mim oferecidos.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho ...

MUITO OBRIGADA !

Sumário

Lista de Figuras

Resumo

Neste trabalho apresentamos uma implementação de um provador de teoremas baseado em tableau semânticos para diversas lógicas modais. A implementação segue a idéia de se dividir as regras de tableau em regras estruturais e de propagação, o que permite que o provador possa ser configurado sob demanda para provar fórmulas em lógicas modais diversas. Cabe ao usuário informar quais são os axiomas que caracterizam a lógica modal desejada e o provador escolhe as regras de tableau correspondentes. A implementação também permite ao usuário determinar a estratégia de aplicação de regras a ser utilizada em cada prova. O trabalho mostra uma aplicação deste provador no contexto de raciocínio sobre ações.

Abstract

This research presented the implementation of a theorem prover based in a semantic tableau for several modal logics. The implementation follows the idea of dividing the tableau rules in propagation rules and structural rules, that enables the prover to be configured on demand to prove formulae in several modal logics. The user has to inform which are the axioms that caracterize the modal logic wanted in case, and the prover chooses the correspondent tableau rules. The implementation also allows the user to determine the rules application strategy to be used in each proof. The research presented an application of this prover in the reasoning context about actions.

Capítulo 1

Introdução

A Inteligência Artificial (IA) surgiu do desejo do homem de criar mecanismos para realizar tarefas que, apesar de todo desenvolvimento tecnológico, ainda são melhor realizadas por seres humanos, aquelas que exigem comportamento "inteligente".

Um dos principais objetivos da IA é desenvolver métodos para realizar tais tarefas através da representação computacional do comportamento humano.

McCarthy afirma que um programa de computador capaz de agir inteligentemente no mundo deve possuir uma representação geral desse mundo. Mais ainda, é preciso uma representação para a maneira de se manipular as informações do mundo. Para tanto, apresenta uma linguagem formal que permite tais representações, o cálculo de situações [35].

Esse trabalho deu início às bases da linha de pesquisa conhecida atualmente por *IA Simbólica* (IAS) sobretudo no que se refere à *representação do conhecimento.*

Existem diversas teorias para formalizar o conhecimento, como por exemplo sistemas a base de regras de produção, as redes semânticas, quadros (*frames*) e, de particular interesse neste texto, a lógica formal.

A lógica é o estudo mais antigo sobre a natureza do raciocínio e do próprio conhecimento, e continua sendo uma teoria muito utilizada em tais representações pois permite inferências sobre os mesmos.

A teoria original – a lógica clássica – foi criada para expressar o raciocínio de princípios

matemáticos onde ambiguidade e imprecisão não são aceitas. Por essa razão, as representações são feitas baseadas em mundos perfeitos onde não há tons de cinza, meias verdades ou conceitos vagos.

Apesar de ser a mais conhecida (ainda se confunde a palavra "lógica" com a própria lógica clássica), a lógica clássica não é suficiente para representar de um modo mais completo o conhecimento humano.

Com o objetivo de minimizar esse problema existem outras lógicas, conhecidas como *lógicas não-clássicas*, que modificam de alguma forma a lógica clássica, seja estendendo seu domínio ou derrogando um ou mais dos seus princípios.

A lógica multivalorada [39], por exemplo, modifica a lógica clássica derrubando o princípio do terceiro excluído, isto é, existem outros valores além de verdadeiro ou falso. Outro exemplo interessante é a lógica paraconsistente [8] que restringe o princípio da não-contradição sem tornar o sistema trivial. Isto é são aceitas inconsistências mas evita-se que a partir de duas premissas contraditórias se possa deduzir uma fórmula qualquer.

Neste trabalho estaremos interessados nas lógicas denominadas "modais". Nestas, a lógica clássica é estendida através de dois novos operadores, chamados de modalidades.

Nosso interesse se justifica pois as lógicas modais fornecem ferramentas poderosas para formalizar uma grande variedade de discursos e raciocínio humanos.

Na lógica modal, uma proposição é *necessária* se é verdadeira em toda situação possível e *possível* se é verdadeira em pelo menos uma situação. A partir desses modos – possibilidade e necessidade – obtém-se quatro noções fundamentais: necessidade, impossibilidade, contingência e possibilidade.

Estas noções são representadas por dois operadores modais básicos: *é possível* e *é necessário*, além de sua combinação com a negação. Essas noções definem as *modalidades* básicas onde uma proposição pode ser considerada verdadeira ou falsa.

Na verdade, os operadores modais podem representar não apenas noções de necessidade e possibilidade, mas também uma gama bastante variada de outras noções não menos importantes para representação do conhecimento humano, tais como: obrigação, crença,

saber, percepção, tempo, ações, dentre outras.

No entanto, do ponto de vista da IAS, representar conhecimento em lógica é apenas parte do problema. A outra parte, não menos importante, é a realização de inferências a partir deste conhecimento.

Em lógica, isto se produz pela verificação de validade das fórmulas. Do ponto de vista da IA, estamos no contexto de prova automática de teoremas.

A busca por um procedimento automático de prova de teoremas é antigo. Sabe-se que Leibniz (1646-1716) já demonstrava interesse no assunto, interesse que foi revivido no início do século XX por Peano e Hilbert. Em 1930, Herbrand [23] deu um passo importante quando propôs um método mecânico para prova de teoremas em lógica de primeira ordem. Apesar de mecânico, o método era muito difícil de aplicar porque era feito "a mão" pois, vale lembrar, na época ainda não havia computadores.

Nos anos 60, Gilmore [20] implementou, com sucesso, o método de Herbrand para um computador digital e foi seguido por Davis e Putnam [10] que propuseram um procedimento um pouco mais eficiente. O próximo passo, e mais importante, foi dado por J. A. Robinson em 1965 [40] ao desenvolver uma única regra de inferência que era uma simplificação muito mais eficiente do método de Davis e Putnam. Tal método foi chamado de "Princípio da Resolução" que, além de eficiente, era simples de implementar.

Durante anos o princípio da resolução foi o método de prova automática mais usado para prova de teoremas em lógica de primeira ordem. É um método fácil de aplicar pois possui uma única regra de inferência. Essa técnica é base do funcionamento da linguagem PROLOG (que nasceu de um provador de teoremas baseado em resolução) e de um modo geral, pode-se dizer que também é base da programação lógica.

Paralelamente aos trabalhos de Herbrand, ainda nos anos 30, Gentzen [43] propôs uma teoria de dedução baseada em regras de manuseio de suposições ao invés da usual formulação axiomática. Essa teoria ficou conhecida como *método da dedução natural* cujo propósito era construir um sistema formal que se aproximasse ao máximo do raciocínio humano.

Gentzen ainda desenvolveu o cálculo de seqüentes [19] que pode ser considerado um método intermediário entre o sistema de dedução natural e os tableau semânticos.

O cálculo de seqüentes executa a prova em uma árvore cujos nós são gerados através de aplicações de regras. O nó raiz contém a negação da fórmula que se deseja provar e os nós seguintes contém listas de fórmulas chamadas "seqüentes" que são estruturas muito rígidas, cuja ordem das fórmulas é importante e a quantidade de vezes que uma fórmula aparece também.

Os métodos de tableau surgiram em 1955 com Hintikka [27] seguido por outros. Mais tarde, Fitting [16] e Goré [21] desenvolveram adaptações dos tableau semânticos para lógicas modais.

Assim como o cálculo de seqüentes, os tableaux são métodos de prova por refutação que executam a prova em uma árvore gerada através da aplicação de regras. A principal diferença é que os tableaux são semânticos e os nós da árvore contém conjuntos de fórmulas, uma estrutura bem mais flexível que seqüentes.

Nos dias de hoje, encontramos muitos trabalhos de implementação de provadores automáticos de teoremas para lógicas modais baseados em tableaux semânticos.

No entanto, a maioria das implementações desses provadores está restrita a realizar provas de teoremas em um único sistema modal através de uma estratégia, ou a ordem de aplicação das regras de tableau, pré-definida. Mas como existem muitas lógicas modais diferentes, o usuário pode querer verificar se uma fórmula é um teorema nesse ou naquele sistema modal, e nesse caso, precisará executar provas em programas diferentes.

Além de querer provar fórmulas em vários sistemas modais diferentes, o usuário também pode achar interessante utilizar diferentes estratégias de busca ou ainda, criar uma nova lógica baseada em outros sistemas modais.

Nesses casos, quando o usuário precisa usar uma lógica ou estratégias de busca um pouco diferentes da lógica e estratégia implementadas no seu sistema, ele deve fazer alterações no código ou até implementar seu próprio provador. Testes para verificar o que pode acontecer se mudar a ordem na aplicação das regras ou outras tentativas acabam sendo impossíveis se o usuário não tiver em mãos o código fonte do provador para fazer

as alterações necessárias [12].

Em situações como essas, é interessante o uso de um provador genérico que permita ao usuário informar a lógica onde ele deseja provar uma fórmula e também a estratégia (a ordem de aplicação das regras e o tipo de busca).

Neste trabalho apresentamos a implementação de um provador genérico de teoremas para lógicas modais baseado em tableau. O usuário informa a fórmula, o sistema modal (ou multimodal) e a estratégia de aplicação de regras a ser utilizada durante a prova e o programa verifica, de forma automática, se tal fórmula é válida ou não no sistema em questão.

O texto está estruturado da seguinta maneira: no capítulo 2, fazemos uma breve introdução às lógicas modais onde a ênfase maior é dada à semântica, apresentando intuitivamente o conceito de mundos possíveis de Kripke através da apresentação do sistema $S5$.

No capítulo 3, as lógicas modais são abordadas de modo mais formal, o conceito de relação de acessibilidade é introduzido e a relação entre a semântica e a axiomática das lógicas modais fica evidente.

A seguir, no capítulo 4, apresentamos o método de tableau com regras de propagação e regras estruturais. Esse método é modular, o que nos permitiu a implementação genérica do provador, cujos detalhes de implementação são apresentados no capítulo 5.

No capítulo 6 apresentamos uma aplicação com o intuito de demonstrar como é possível adaptar o provador para outras lógicas, através da implementação de outras regras de tableau.

Por fim, no capítulo 7 traçamos nossos comentários conclusivos acerca do trabalho e apresentamos algumas sugestões de trabalhos futuros.

Capítulo 2

Uma lógica diferente

Neste capítulo, e no seguinte, apresentamos a lógica modal como uma alternativa ao uso da lógica clássica na representação do conhecimento. A estrutura do texto está baseada em Herzig [24] e recomendamos como leitura complementar os textos de Hughes & Cresswell [28], Chellas [7] e Popkorn [37].

O desenvolvimento da lógica modal tem uma história interessante que Blackburn [2] sugere dividir em 3 períodos: a era sintática, a era clássica e a era moderna.

Desde os tempos de Aristóteles que os lógicos comentam sobre o estudo de modalidades [3]. Mas somente em meados do século XX, com os trabalhos de Lewis [32, 33], é que foram apresentadas as formalizações matemáticas para a lógica modal.

Foi o início da era sintática, que foi marcada pelo intenso estudo na sintaxe e também pela falta de uma interpretação formal adequada. Lewis não foi o primeiro a considerar o raciocínio modal, mas foi o primeiro a propor um sistema simbólico mais claro, estendendo a linguagem do cálculo proposicional com a inclusão de uma modalidade unária que ele denominou I significando "é impossível que". O sistema modal de Lewis é muito diferente dos estudados hoje porque ao invés de estender a axiomática do cálculo proposicional apenas com a inclusão de axiomas exclusivamente modais, Lewis definiu sua própria axiomática. Anos depois, Gödel [18] desenvolveu um sistema modal mais parecido com o que vemos nos dias de hoje.

Outros autores como Feys, Carnap e Lemmon, também desenvolveram estudos sobre o tema. Naquela época, essa incipiente lógica modal ainda era muito criticada quanto a sua aplicabilidade porque não havia uma semântica bem definida que pudesse representar formalmente todas as possibilidades relevantes.

Uma semântica adequada só surgiu em 1959 com os trabalhos de Kripke [30] e esse ano marca o início de uma nova fase para a lógica modal, a era clássica.

A nova semântica de Kripke revolucionou o estudo da lógica modal, pois problemas antes muito difíceis e complexos encontraram solução simples e direta com o uso dessa caracterização semântica, que será melhor descrita ao longo deste capítulo.

O uso da lógica modal na ciência da computação marca o início da era moderna. Isto se deu a partir dos anos 70 com trabalhos de Pratt em PDL [38], quando a falta de aplicabilidade da lógica modal deixou de ser um problema. Surgiram então muitos trabalhos no sentido de como aplicar a lógica modal em ciência da computação. Essa era foi marcada pela intensa produção científica e o estabelecimento da lógica modal como uma ferramenta muito útil para a ciência da computação.

Intuitivamente, a lógica modal é uma extensão sintática da lógica clássica que introduz novos conectivos unários $\Box_i$ e $\Diamond_i$ com i pertencente ao conjunto I de índices. Tais conectivos são mais comumente conhecidos como *operadores modais* e servem, respectivamente, para representar as noções de necessidade e possibilidade.

Nesse sentido, se temos uma expressão da forma $\Box_i A$, onde A é uma fórmula qualquer, então temos que A é forçada a ser verdadeira em toda situação possível e sua condição de verdade é *necessariamente verdadeira* ou *necessária.*

Já em uma expressão representada pela fórmula $\Box_i \neg A$, A é falsa em qualquer situação possível e dizemos que A é *necessariamente falsa* ou *impossível.* Da mesma forma, a expressão $\neg\Box_i A$ diz que A não é necessária, logo é uma *contingência.* E ainda, a expressão $\neg\Box_i\neg A$ afirma que não é necessário que A seja falsa, ou que A não é *impossível* e dizemos que A é *possível.*

O que torna este estudo tão interessante é que os operadores modais não precisam ser

interpretados apenas em termos de necessidade e possibilidade. Dependendo do contexto em que são empregados, eles permitem formalizações de outros conceitos que, quando empregados em uma proposição, não possuem um valor lógico preciso. Entre tais conceitos, ou modalidades, podemos citar: a *crença*, o *conhecimento*, a *obrigação*, a *permissão*, a *incerteza*, a *execução de um programa* ou de uma *ação*, etc.

Originalmente havia apenas um conectivo $\Box$[1], mas há ocasiões em que é interessante adicionar vários (possivelmente infinitos) conectivos $\Box$ com significados diferentes, representados por $\Box_i$, um para cada i pertencente ao conjunto I.

Os operadores modais podem ser indexados conforme a modalidade a ser formalizada. Assim, a fórmula $\Box_i A$ pode representar expressões diferentes, dependendo do conceito que estamos interessados em formalizar:

- "O agente i acredita que A" é uma interpretação doxástica;
- "O agente i sabe A" é uma interpretação epistêmica;
- "O agente i é obrigado a A" para uma interpretação deôntica;
- "A é verdade no instante i" para uma interpretação temporal;
- "A é verdade com grau i" em se tratando de interpretação em termos de incerteza;
- "A é verdade depois de executado o programa i" interpretando em termos de programas;
- "A é verdade depois de executada a ação i" para uma interpretação em termos de ações;
- "A é provável em um sistema formal i" em algum sistema formal i;

Na literatura, podemos encontrar várias famílias de lógicas modais. As interpretações particulares dos operadores modais têm motivado aplicações dessas lógicas, em particular na inteligência artificial, sobretudo na representação do conhecimento, raciocínio sobre ações e na verificação de programas.

[1] $\Box$ pode ser representado assim sem o índice quando conjunto I for unário.

No restante deste capítulo, introduzimos as lógicas modais através da explanação relativamente informal de algumas das suas principais características. Deixamos para o próximo capítulo uma análise mais aprofundada sobre estas lógicas.

2.1 Uma linguagem mais geral

Com o uso dos operadores modais, temos uma linguagem mais geral do que o cálculo proposicional que, como já citamos, permite formalizar expressões que representam conceitos mais vagos, cujo valor de verdade não é expresso apenas em termos de verdadeiro ou falso. A seguir, apresentamos, uma introdução aos principais conceitos sintáticos usados em lógica modal.

As sentenças não-atômicas da linguagem são formadas a partir dos conjuntos de fórmulas atômicas (que são as sentenças mais simples da lógica modal) e de conectivos já conhecidos da lógica clássica proposicional com a adição do conjunto dos operadores modais.

O conjunto formado pelas fórmulas atômicas ou símbolos proposicionais é o conjunto:

$$FLM_o = \{p, q, r, p_1, p_2, ...\}$$

- O conjunto dos conectivos clássicos é o conjunto $\{\wedge, \neg\}$.
- O conjunto dos operadores modais $\{\Box_i : i \in Index\}$ onde $\Box_i$ pode ser representado por $[i]$ ou simplesmente $\Box$ no caso de $Index$ ser unitário.
- A união do conjunto dos conectivos clássicos com o conjunto dos operadores modais mais o $\top$ (*verum*) forma o conjunto de operadores da lógica modal.

O operador $\top$ é constante; $\neg$ e $\Box_i$ são operadores unários; e $\wedge$ é binário.

Definição 2.1.1 (Abreviaturas) O conectivo clássico $\vee$ é definido como abreviatura de $\neg(\neg A \wedge \neg B)$. $A \rightarrow B$ é definido como abreviatura de $\neg(A \wedge \neg B)$. Do mesmo modo,

o conectivo clássico $\leftrightarrow$ abrevia $\neg(A \wedge \neg B) \wedge \neg(B \wedge \neg A)$, o operador modal $\Diamond_i A$ abrevia $\neg\Box_i\neg A$ e o operador $\bot$ (*falso*) abrevia $\neg\top$.

A ordem de precedência dos operadores é a mesma da lógica clássica e os operadores modais e a negação têm maior prioridade.

O conjunto das sentenças é o conjunto FLM tal que:

Definição 2.1.2 (Fórmulas da Lógica Modal)

- $\top \in FLM$;
- Seja FLM_o, o conjunto das sentenças atômicas, então $FLM_o \subseteq FLM$;
- Se $A \in FLM$ então $\neg A \in FLM$;
- Se $A \in FLM$ e $B \in FLM$, então $(A \wedge B) \in FLM$;
- Se $A \in FLM$ e $\Box_i$ é um operador modal, então $(\Box_i A) \in FLM$.

Chamamos *subsentença* de uma sentença A qualquer sentença que é parte de A, incluindo a própria sentença A. A definição formal do conjunto $Sub(A)$, formado pelas subsentenças de A, é mostrada a seguir:

Definição 2.1.3 (Subsentença)

1. $Sub(p) = \{p\}$ para todo $p \in FLM_0$.
2. $Sub(\top) = \{\top\}$.
3. $Sub(\neg A) = \{\neg A\} \cup Sub(A)$.
4. $Sub(A \wedge B) = \{A \wedge B\} \cup Sub(A) \cup Sub(B)$.
5. $Sub(\Box_i A) = \{\Box_i A\} \cup Sub(A)$.

Exemplo 2.1.1 (Conjunto Sub) Seja a sentença $\Box_i(A \rightarrow \neg\Diamond_i(B \wedge \neg A))$, o conjunto formado por suas subsentenças é: $Sub(\Box_i(A \rightarrow \neg\Diamond_i(B \wedge \neg A))) = \{\Box_i(A \rightarrow \neg\Diamond_i(B \wedge \neg A)), (A \rightarrow \neg\Diamond_i(B \wedge \neg A)), A, \neg\Diamond_i(B \wedge \neg A), \Diamond_i(B \wedge \neg A), B \wedge \neg A, B, \neg A\}$.

Outra maneira de ver a estrutura de uma sentença é por meio de uma *árvore de construção* (Figura2.1). As subsentenças são escritas nos nodos da árvore e os ramos indicam a ordem de aplicação das operações sintáticas. Uma árvore de construção não mostra apenas as subsentenças de uma sentença mas também indica suas ocorrências. Na figura 2.1, a subsentença A tem duas ocorrências na sentença $\Box_i(A \rightarrow \neg\Diamond_i(B \wedge \neg A))$ e por isso aparece em dois nodos da árvore.

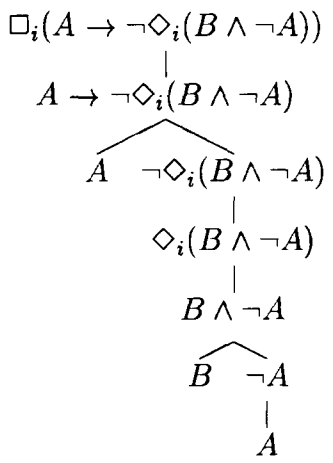


Figura 2.1: Ilustração da árvore de construção para $\Box_i(A \rightarrow \neg\Diamond_i(B \wedge \neg A))$

Definição 2.1.4 (Modalidade) Chamamos de *modalidade* qualquer seqüência finita, possivelmente vazia, dos operadores $\neg, \Box_i, \Diamond_i$. Por exemplo, $\Box_i$, $\neg\neg$, $\Box_i\Diamond_i$, $\Diamond_i\Box_i\Box_i$, $\Box_i\neg\Diamond_i$, $\Box_j\Box_i$. A modalidade vazia ou nula é representada por ϕ, por exemplo ϕA que é mesmo que A. Uma modalidade pode ser *afirmativa* ou *negativa* e classificamos uma modalidade como afirmativa, no caso de $\neg$ ocorrer um número par de vezes (incluindo zero) e negativa se o $\neg$ ocorrer um número ímpar de vezes.

Definição 2.1.5 (Substituição) Sejam A, B e B' sentenças da lógica modal, chamamos de substituição $A[B/B']$, a sentença resultante da troca de uma ou mais ocorrências de B por B' em A.

Exemplo 2.1.2 (Substituição) Seja $A = \Box_i(p \rightarrow \neg\Diamond_i(q \wedge \neg p))$, $B = p$ e $B' = r \vee \Box_i\neg q$, $A[B/B']$ pode ser uma das sentenças abaixo:

$$\Box_i((r \vee \Box_i\neg q) \rightarrow \neg\Diamond_i(q \wedge \neg p))$$

$$\Box_i(p \rightarrow \neg\Diamond_i(q \wedge \neg(r \vee \Box_i\neg q)))$$

$$\Box_i((r \vee \Box_i\neg q) \rightarrow \neg\Diamond_i(q \wedge \neg(r \vee \Box_i\neg q)))$$

Definição 2.1.6 (Esquemas) Chamamos de esquemas um conjunto de sentenças que são usualmente da mesma forma.

Exemplo 2.1.3 (Esquemas) O esquema $\Diamond_i A \rightarrow \Box_i\Diamond_i A$ se refere a todas as sentenças nesta forma: uma condicional com uma possibilidade como antecedente e a necessidade de uma possibilidade como consequente.

Definição 2.1.7 (Esquemas de axiomas) Chamamos de esquemas de axiomas o conjunto de axiomas que são usualmente da mesma forma.

Definição 2.1.8 (Esquema de regra de inferência) Um esquema de *regra de inferência* é uma tupla $\langle A, A_1, A_2, ..., A_n\rangle$, para $n \geq 1$, escrita $\frac{A_1, A_2, ..., A_n}{A}$, onde $A_1, A_2, ..., A_n$ são esquemas.

Vimos a lógica modal tratada sob o ponto de vista da linguagem, fórmulas e regras para formar tais fórmulas. Na próxima seção, estaremos interessados na interpretação dessas fórmulas.

2.2 Uma semântica geral

O conceito de mundos possíveis nos quais sentenças da linguagem são verdadeiras ou falsas será melhor explicado nesta seção.

Ao trabalhar com as noções de necessidade e possibilidade, fazemos afirmações sobre vários mundos diferentes (ou situações possíveis). Uma sentença na forma $\Box_i A$ é verdadeira se e somente se A é verdadeira em todo mundo possível. Já uma sentença na forma $\Diamond_i A$ será verdadeira só no caso de A ser verdadeira em pelo menos um mundo possível. Ou seja, já não temos um único valor verdade como na lógica clássica, estamos agora diante de vários, onde cada valor assumido é em relação a um mundo possível. Logo, o valor verdade de A pode ser diferente em mundos diferentes.

Nos vemos, então, diante de vários mundos possíveis incluindo nosso próprio mundo real, onde as sentenças de uma linguagem são verdadeiras ou falsas. Desse modo, precisamos delinear essa noção de mundos, o que é feito através de um modelo.

Definição 2.2.1 (Modelo) Um modelo é um par $\langle W, V \rangle$ onde W é um conjunto não vazio de mundos possíveis e V é uma função de valoração tal que $V : FLM \rightarrow \mathcal{P}(w)$, onde se $A \in FLM$, $V(A) = \{w \in W | A \text{ é verdadeira em } w\}$.

Ao observarmos a figura 2.2, vemos um exemplo de modelo onde o conjunto de mundos possíveis é formado pelos mundos w_1, w_2 e w_3. $\Box_i A$ é verdadeira em w_1, pois A é verdadeira em todos os mundos do modelo (w_1, w_2 e w_3). $\Diamond_i B$ é verdadeira em w_1, pois há pelo menos um mundo no modelo onde B é verdadeira (w_3).

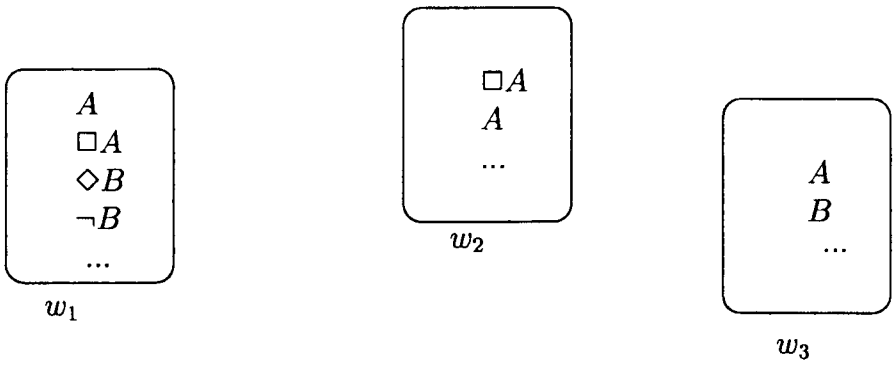


Figura 2.2: Ilustração de um fragmento da semântica de mundos possíveis.

Uma representação formal é feita aplicando-se a definição 2.2.1 onde $W = \{w_1, w_2, w_3\}$ e $V(A) = \{w_1, w_2, w_3\}$, $V(\Box_i A) = \{w_1, w_2\}$, $V(\Diamond_i B) = \{w_1\}$, $V(B) = \{w_3\}$ e $V(\neg B) = \{w_1\}$.

Em termos de mundos possíveis em um modelo, o valor de verdade para as sentenças é dado de acordo com sua forma. Para uma sentença A e um mundo possível w em um modelo $M = \langle W, V \rangle$, representamos $\models_w^M A$ para expressar que A é verdadeira no mundo w pertencente ao modelo M.

Neste caso, as condições de verdade são as seguintes:

Definição 2.2.2

1. $\vDash_w^M A$ sse $w \in V(A)$;

2. $\vDash_w^M \top$;

3. $\vDash_w^M \neg A$ sse não $\vDash_w^M A$;

4. $\vDash_w^M A \wedge B$ sse ambos $\vDash_w^M A$ e $\vDash_w^M B$;

5. $\vDash_w^M \Box_i A$ sse para todo w' em M, $\vDash_{w'}^M A$;

É interessante apresentar a condição de verdade para as abreviaturas:

6. $\neg \vDash_w^M \bot$;

7. $\vDash_w^M A \vee B$ sse ou $\vDash_w^M A$ ou $\vDash_w^M B$ ou ambos;

8. $\vDash_w^M A \rightarrow B$ sse se $\vDash_w^M A$ então $\vDash_w^M B$;

9. $\vDash_w^M A \leftrightarrow B$ sse $\vDash_w^M A$ se e somente se $\vDash_w^M B$;

10. $\vDash_w^M \Diamond_i A$ sse para algum w' em M $\vDash_{w'}^M A$.

Se uma sentença A for verdadeira em todo mundo possível de todos os modelos, então A é dita *válida*, e representamos por $\vDash A$. Formalmente, temos:

Definição 2.2.3 (Validade) $\vDash A$ sse para todo modelo M e todo mundo possível w em M, $\vDash_w^M A$.

Em se tratando da lógica da necessidade e da possibilidade, nós estamos interessados em conhecer quais sentenças são válidas e quais não são. Por exemplo, todas as sentenças da forma $\Box_i A \rightarrow A$ e todas as sentenças da forma $\Diamond_i A \rightarrow \Box_i \Diamond_i A$ são válidas, enquanto nem toda sentença da forma $A \rightarrow \Box_i A$ é válida.

Exemplo 2.2.1 ($\vDash \Box_i A \rightarrow A$) De acordo com tal fórmula, o que é necessário é verdade e é lido da seguinte maneira: se necessário A então A. Para verificar que esse esquema é válido, basta provar que se w é um mundo possível em qualquer modelo M, então $\vDash_w^M \Box_i A \rightarrow A$ se verifica. E, para isso, é suficiente mostrar que se $\vDash_w^M \Box_i A$ então $\vDash_w^M A$

(pela cláusula 8 da definição de verdade). Então suponha que que $\vDash_w^M \Box_i A$. Pela cláusula 5 da definição de verdade, isto significa dizer que $\vDash_{w'}^M A$ para todo mundo w' no modelo M (inclusive para w). Logo $\vDash_w^M A$.

Exemplo 2.2.2 ($\vDash \Diamond_i A \rightarrow \Box_i \Diamond_i A$) O esquema diz que se possível A, então é necessariamente possível A ou o que é possível é necessariamente possível. Para verificar esta validade, suponha que $\vDash_w^M \Diamond_i A$, para um mundo possível w pertencente ao modelo M. Isto é o mesmo que dizer que o modelo M tem um mundo possível w' tal que $\vDash_{w'}^M A$ (conforme cláusula 10 da definição de verdade). Então, segue (novamente pela clausula 10) que não importa qual mundo possível no modelo escolhamos, $\Diamond_i A$ se verifica, isto é, $\vDash_{w'}^M \Diamond_i A$ para todo mundo possível w' pertencente ao modelo. Mas, por 5, isto quer dizer que $\vDash_w^M \Box_i \Diamond_i A$ que é exatamente o que queremos mostrar.

Para provar a não validade de um esquema basta mostrar um modelo onde haja um mundo no qual o esquema não seja válido. Tal modelo é chamado *contra-modelo* e pode ser representador por $\nvDash$.

Definição 2.2.4 (Contra-modelo) Se uma sentença A for falsa em um mundo w pertencente a um modelo M, dizemos que A é falsa no modelo. E quando tal sentença é falsa em M, dizemos que tal modelo é um contra-modelo para A.

Exemplo 2.2.3 ($\nvDash A \rightarrow \Box_i A$) Seja w_1 e w_2 dois mundos distintos, seja $W = \{w_1, w_2\}$ e seja $V(p_n) = \{w_1\}$ onde p_n é uma fórmula qualquer pertencente ao conjunto de fórmulas da lógica modal e n pertence ao conjunto dos números naturais. Então $M = \langle W, V \rangle$ é um modelo no qual $\vDash_{w_1}^M p_n$, pois $w_1 \in V(p_n)$ mas $\nvDash \Box_i p_n$ (pois o mundo $w_2 \notin V(p_n)$). Assim, temos um modelo onde $\nvDash_{w_1}^M p_n \rightarrow \Box_i p_n$ o que prova que $\nvDash_{w_1}^M A \rightarrow \Box_i A$.

Comparando a lógica modal com a lógica clássica proposicional, podemos dizer que a lógica modal é um super-conjunto da lógica clássica porque a primeira inclui a segunda. O que vale dizer, em parte, que toda sentença proposicionalmente válida é também uma sentença válida na lógica modal. Isto é se A é uma *tautologia*, então $\vDash A$ - com que estamos afirmando que toda tautologia da lógica clássica proposicional é uma sentença válida na lógica modal [7, p.6].

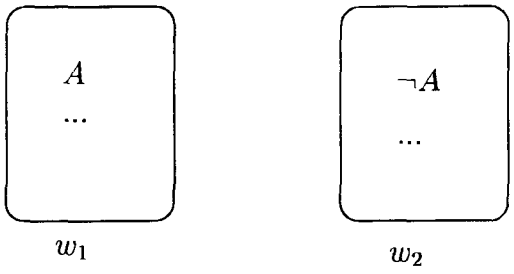


Figura 2.3: Um contra-modelo para $A \rightarrow \Box_i A$.

Definição 2.2.5 (Consequência lógica) B é consequencia lógica de A se, e somente se, todo modelo de A é um modelo de B e escrevemos $A \models B$.

Neste capítulo, apresentamos uma lógica diferente por ser menos restrita que a lógica clássica na representação do conhecimento sobre o mundo real, a lógica modal. Com o objetivo de fazer uma introdução mais intuitiva da nova lógica aqui apresentada, nossa discussão estava mais preocupada em apresentar os conceitos semânticos.

A lógica modal é, na realidade, uma família de lógicas modais. Apesar disso, neste capítulo, nossa explicação se ateve na semântica de uma lógica modal mais genérica, cujos modelos apresentam mundos que se relacionam entre si sem restrição. Mas quando a relação entre os mundos do modelo não é assim tão generalizada, precisamos nos preocupar com a maneira com que um mundo se relaciona com outro, ou como um mundo pode ser acessado a partir de outro e essa *relação de acessibilidade* é um dos fatores que torna a lógica modal interessante.

No próximo capítulo, apresentamos uma semântica capaz de representar as diversas relações entre os mundos pertencentes a um modelo, bem como a axiomatização de lógicas modais e a estreita relação existente entre a semântica e a axiomática dessas lógicas.

Capítulo 3

Lógica modal

No capítulo anterior, apresentamos informalmente a lógica modal. Nesta apresentação, a partir de um mundo qualquer, pode-se acessar todos os outros mundos do modelo (inclusive o próprio). Neste capítulo esta visão é generalizada.

A motivação é que há situações em que é interessante acessar apenas "pedaços" do modelo e não todos os mundos. Por exemplo, em casos de mundos que possuem algo em comum, ou que estão de alguma forma relacionados entre si. Neste caso, precisamos de alguma maneira representar esses mundos e o relacionamento entre eles.

Em 1959, Kripke [30] aprimorou a idéia inicial de Leibniz[1] propondo que o conceito de "possivelmente verdade" fosse modificado para "verdade em algum mundo possível", ou "verdade em algum mundo acessível", ou "em alguma situação possível", etc. A partir dessa idéia, Kripke propôs uma generalização de sua semântica que representa também essa "relação" entre os mundos possíveis através do que ele chamou de *relação de acessibilidade* definindo "mundos possíveis" como sendo todos os mundos "acessíveis" a partir do mundo (ou estado) atual.

A relação de acessibilidade proposta por Kripke fornece uma semântica mais forte para a lógica modal e permite representar tais "pedaços" de mundos possíveis visualizando apenas os mundos que estão relacionados entre si.

[1]Para Leibniz [41] existem conceitos de verdade. O que é verdade em algum mundo possível é "possivelmente verdade" e o que é verdade em todos os mundos possíveis pois é eterno é "necessariamente verdade".

Neste capítulo, apresentaremos a nova semântica que representa as relações entre os mundos do modelo, assim como a axiomatização de algumas lógicas modais. No final da leitura deste capítulo deverá ficar claro ao leitor a relação entre os axiomas das lógicas modais e a semântica dos mundos possíveis de Kripke.

3.1 Modelos, verdade e validade

A semântica de Kripke apresentada no capítulo anterior nos permite visualizar os mundos definidos na lógica modal como estados representados por conjunto de fórmulas mas não consegue representar a maneira como esses mundos se relacionam entre si. Por isso, Kripke sugeriu uma generalização da noção de modelo com o intuito de evidenciar o relacionamento entre os mundos.

Nessa generalização, define-se uma *relação de acessibilidade* entre os mundos de um modelo. Tal relação deve mostrar que a partir de um determinadado mundo pode-se acessar ou "visualizar" outros mundos do modelo, eventualmente todos. É importante salientar que as noções de necessidade e possibilidade são redefinidas em função da relação de acessibilidade. Uma fórmula é necessária em um mundo se for verdadeira em todos os mundos *acessíveis* a partir do mundo atual e não mais em *todos* os mundos existentes. Do mesmo modo, uma fórmula é possível em um mundo se for verdadeira em *pelo menos um* mundo acessível a partir do atual.

Essa generalização dos modelos de Kripke é baseada no conceito de "estrutura" (*frame*), que são estruturas relacionais simples com uma relação binária como definida a seguir.

Definição 3.1.1 (Estrutura) Uma estrutura é um par $\langle W, R_i \rangle$ tal que:

1. W é um conjunto de mundos possíveis;

2. R_i é uma relação binária sobre W (i.e. $R_i \subseteq W \times W$).

Podemos agora redefinir modelos, desta vez baseados em estruturas:

Definição 3.1.2 (Modelo padrão da lógica modal ou modelo de Kripke) um modelo padrão para a lógica modal é um par $\langle \mathcal{F}, V \rangle$ onde:

1. $\mathcal{F}$ é uma estrutura;

2. V é uma função de valoração que mapeia fórmulas em subconjuntos de W.

Um modelo padrão também pode ser representado como a tripla $\langle W, R_i, V \rangle$, onde W é o conjunto dos mundos possíveis, R_i relação binária sobre W e V a função de valoração.

Dados dois mundos w_1 e w_2 pertencentes ao conjunto de mundos possíveis W, R_i representa uma relação binária entre os mundos e escrevemos $w_1 R_i w_2$ para representar que o mundo w_2 é acessível a partir de w_1, ou que w_2 é o sucessor de w_1, ou ainda que w_1 "enxerga" w_2. Escrevemos $w_1 \not R_i w_2$ para dizer que w_1 não está relacionado com w_2.

Em função desta nova maneira de encarar os modelos, não é difícil perceber que as características da lógica em questão dependem das propriedades da relação de acessibilidade. Algumas propriedades interessantes que podemos destacar são: a reflexividade, a serialidade, transitividade, euclidianidade e simetria, para citar apenas as mais comuns. Apenas para efeito de clareza no texto, relembramos aqui as principais definições:

- Seja uma estrutura $\langle W, R_i \rangle$ qualquer. Dizemos que a relação de acessibilidade é reflexiva se para todo mundo w pertencente a W, wR_iw (figura 3.1).

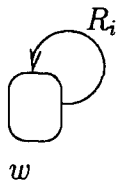


Figura 3.1: Estrutura com relação de acessibilidade reflexiva.

- A relação de acessibilidade é dita serial, quando para todo mundo w_1 pertencente a W existe um mundo w_2, tal que $w_1 R_i w_2$ (figura 3.2).

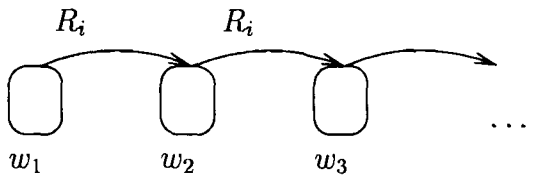


Figura 3.2: Estrutura com relação de acessibilidade serial.

- A relação de acessibilidade é dita transitiva quando para todo w_1, w_2, w_3, se $w_1 R_i w_2$ e $w_2 R_i w_3$ então $w_1 R_i w_3$ (figura 3.3).

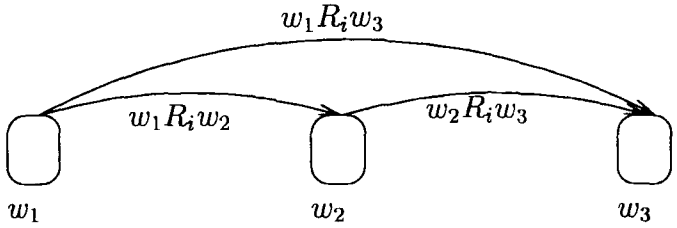


Figura 3.3: Estrutura com relação de acessibilidade transitiva.

- Uma relação de acessibilidade é dita euclidiana se, para todo w_1, w_2, w_3, se $w_1 R_i w_2$ e $w_1 R_i w_3$ então $w_2 R_i w_3$ (figura 3.4).

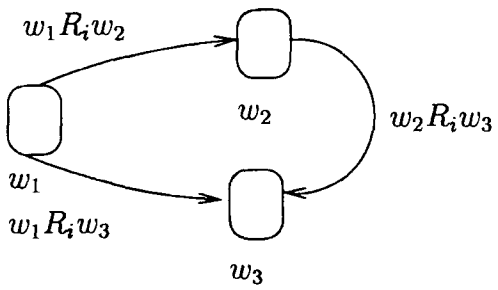


Figura 3.4: Estrutura com relação de acessibilidade euclidiana.

- A relação de acessibilidade é dita simétrica, se para quaisquer dois mundos w_1 e w_2, se $w_1 R_i w_2$ então $w_2 R_i w_1$ (figura 3.5).

As condições de verdade para as sentenças não-modais são as mesmas citadas no

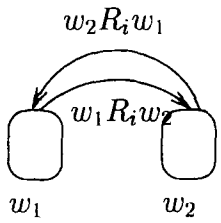


Figura 3.5: Estrutura com relação de acessibilidade simétrica.

capítulo anterior mas, como agora as relações de acessibilidade estão sendo consideradas, as condições de verdade para as sentenças modais devem ser redefinidas. Assim, $\Box_i A$ é verdadeiro em um mundo w_1 se e somente se, A for verdadeiro em todo mundo w_2 acessível a partir de w_1, ou seja, em todo mundo w_2 de modo que $w_1 R_i w_2$. $\Diamond_i A$ é verdadeiro em w_1 se e somente se, A é verdadeira em algum mundo possível w_2 acessível a partir de w_1, ou seja em algum mundo w_2 tal que $w_1 R_i w_2$. Formalmente temos[2]:

Definição 3.1.3 (Verdade) Seja w_1 um mundo em um modelo padrão $M = \langle W, R_i, V \rangle$:

1. $\models^M_{w_1} \Box_i A$ sse $\models^M_{w_2} A$, para todo w_2 em W tal que $w_1 R_i w_2$;

2. $\models^M_{w_1} \Diamond_i A$ sse $\models^M_{w_2} A$ para algum w_2 em W tal que $w_1 R_i w_2$.

De acordo com $\models^M_{w_1} \Box_i A$, a fórmula $\Box_i A$ é verdade em um mundo w_1 quando A for verdade em *todos* os mundos possíveis w_2 que estejam relacionados com w_1 segundo a relação R_i. $\models^M_{w_1} \Diamond_i A$ afirma que $\Diamond_i A$ é verdade em um mundo w_1 somente se A for verdade em *pelo menos um* mundo possível w_2 relacionado com w_1.

É importante lembrar que a definição de modelo apresentada no capítulo anterior está incluída nesta, apenas observando-se que a relação de acessibilidade é universal e todos os mundos pertencentes a esse modelo, estão relacionados entre si.

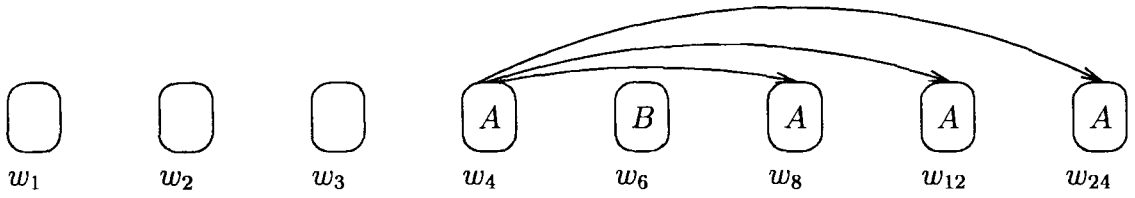
Outra situação importante é quando um mundo w não se relaciona com nenhum outro mundo do modelo, nesse caso conforme a definição de verdade 3.1.3 $\models^M_w \Box_i A$ porque A é verdade em todos os mundos que se relacionam com w (neste caso, nenhum) satisfazendo a condição de verdade para $\Box_i$ e $\not\models^M_w \Diamond_i A$ pois, pela definição de verdade 3.1.3, A deveria

[2]*Assim como no capítulo anterior, definimos verdade para o* $\Diamond_i$*, apesar de* $\Diamond_i$ *ser definido em função de* $\neg\Box_i\neg$

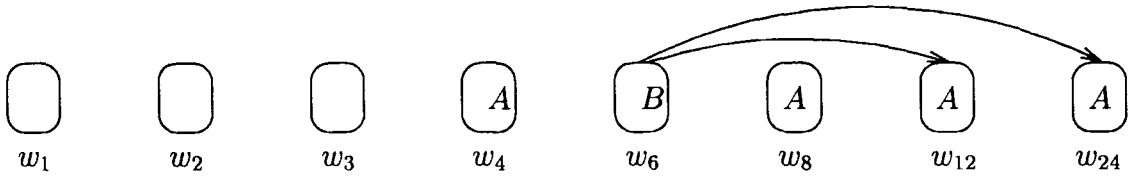
ser verdade em pelo menos um mundo relacionado a w, e como não há nenhum, a condição de verdade não se satisfaz.

Exemplo 3.1.1 Seja M o modelo onde $W = \{w_1, w_2, w_3, w_4, w_6, w_8, w_{12}, w_{24}\}$ e $w_x R_i w_y$ se e somente se, $x \neq y$ and y é divisível por x. Seja a função de valoração V onde $V(A) = \{w_4, w_8, w_{12}, w_{24}\}$ e $V(B) = \{w_6\}$. Temos:

- $\models^M_{w_4} \Box_i A$: afirma sobre a validade da sentença $\Box_i A$ e se explica pelo fato do mundo w_4 estar relacionado apenas com os mundos w_8, w_{12} e w_{24} que são mundos onde A é verdade.

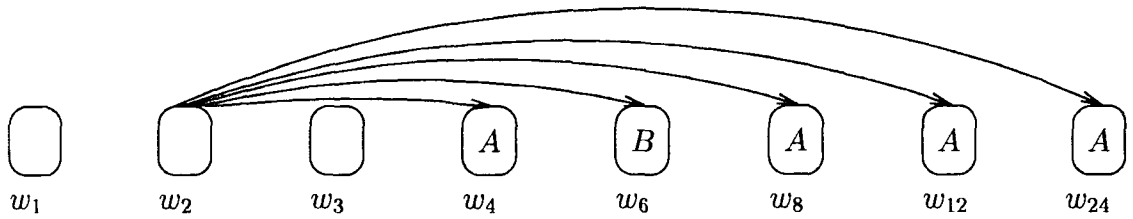


- $\models^M_{w_6} \Box_i A$: esta afirmação se verifica pois o mundo w_6 esta relacionado apenas com os mundos w_{12} e w_{24} nos quais A é verdade conforme a figura a seguir.

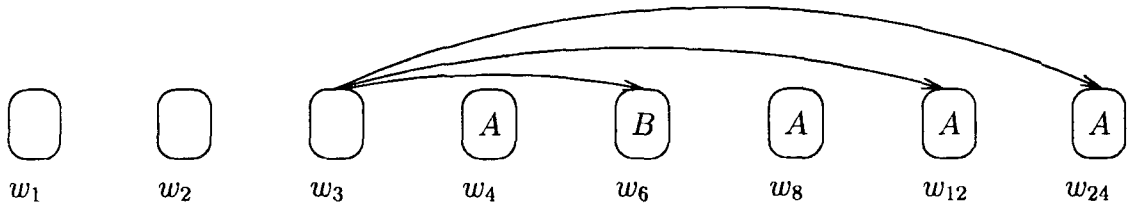


- $\not\models^M_{w_2} \Box_i A$: como podemos observar na próxima figura, esta afirmação se confirma pois apesar de w_2 estar relacionado com todos os mundos pertencentes a $V(A)$, ele também está relacionado com o mundo w_6 que não pertence a $V(A)$.

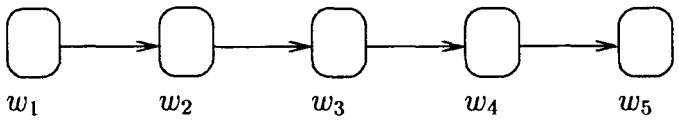
- $\models^M_{w_3} \Diamond_i A$: esta afirmação está representada na figura abaixo onde observamos sua validade, pois w_3 está relacionado w_6,w_{12} e w_{24}, então é correto afirmar que em pelo



menos um mundo ao qual w_3 está relacionado A é verdade.



Exemplo 3.1.2 Seja $\mathcal{F} = (\{w_1, w_2, w_3, w_4, w_5\}, R_i)$, onde $w_x R_i w_y$ sse $y = x + 1$:



Se escolhermos uma valoração V em $\mathcal{F}$ de modo que $V(A) = \{w_2, w_3\}$, $V(B) = \{w_1, w_2, w_3, w_4, w_5\}$ e $V(C) = \emptyset$, temos um modelo $M = (\mathcal{F}, V)$ onde:

- $\models^M_{w_1} \Diamond_i \Box_i A$
- $\not\models^M_{w_1} \Diamond_i \Box_i A \rightarrow A$
- $\models^M_{w_2} \Diamond_i (A \wedge \neg C)$
- $\models^M \Box_i B$

É evidente que a sentença $\Box_i B$ é válida nos mundos w_1, w_2, w_3 e w_4, mas podemos afirmar que é válida no mundo w_5? Se notarmos que w_5 não tem sucessor (mundo acessível a partir dele), então podemos afirmar que B é verdade em *todos* os sucessores de w_5, logo $\Box_i B$ é válida em w_5.

3.2 Axiomatização de sistemas modais

A axiomática dos sistemas de lógica modal pode ser definida adicionando-se àquela da lógica clássica novos axiomas relativos aos operadores modais.

Vamos considerar somente as lógicas modais normais (que serão definidas em 3.2.1) pois essas são mais bem comportadas e tem alguns resultados gerais mais interessantes para este nosso trabalho. O estudo das lógicas modais normais é mais importante quando estamos particularmente interessados em lógicas epistêmicas (ligadas ao saber) ou deônticas (ligadas ao dever ou às normas). Um estudo detalhado sobre lógicas modais não-normais foi apresentado por Gasquet em [17].

O sistema modal mais conhecido é o sistema K (chamado assim em homenagem a Saul Kripke). Os axiomas de K são todas as instâncias de tautologias do cálculo proposicional mais o axioma K como esquemas de axiomas e tomando *modus ponens* (MP), a *regra da necessitação* (R_iN) e a *regra da substituição uniforme* como regras de inferência.

Sua importância se deve ao fato de o axioma K permitir transformar $\Box_i(A \rightarrow B)$ (uma fórmula com $\Box_i$) em $\Box_i A \rightarrow \Box_i B$ (uma implicação)[2].

Mas, em muitas situações o sistema K ainda é muito fraco e não consegue representar a necessidade da maneira desejável, por isso existem outros sistemas modais normais baseados em K.

Definição 3.2.1 (Lógica modal normal) Uma lógica modal normal é um conjunto de fórmulas que contém todas as tautologias e K como axiomas e *modus ponens*, *regra da necessitação* e a *regra da substituição uniforme*[3] como regras de inferência.

O axioma K e as regras de inferência MP, R_iN e SU são esquemas da forma:

[3]leva esse nome porque é a substituição, como definida em 2.1.5, em todas as ocorrências.

K. $\quad \Box_i(A \to B) \to (\Box_i A \to \Box_i B)$

MP. $\quad \frac{A \to B, A}{B}$

RN. $\quad \frac{A}{\Box_i A}$, A tautologia

SU. $\quad \frac{A}{B}$

Em SU, B é obtido pela troca de todos os símbolos pertencentes a FLM_0 em A por fórmulas arbitrárias.

Assim, a regra da substituição uniforme SU é construída de tal forma que qualquer instância de substituição de um esquema de axioma ou teorema é também um teorema [21].

A lógica K é a lógica modal normal minimal porque sua axiomatização é feita utilizando além das tautologias apenas o axioma K. A adição de fórmulas apropriadas da figura 3.6 à axiomática de K resulta em uma nova lógica modal normal.

Quando uma lógica modal é axiomatizada através da adição dos axiomas $A_1, A_2, ..., A_n$ a K, então seu nome é escrito como $KA_1A_2...A_n$. Algumas lógicas modais são mais conhecidas na literatura por outro nome, por exemplo, a lógica modal $KT5$, definida pelos axiomas K, T e 5, é mais tradicionalmente conhecida por $S5$ e a lógica modal $KT4$, definida pelos axiomas K, T e 4 é mais conhecida como $S4$. Aqui, preferencialmente, adotaremos o nome tradicional de cada lógica modal.

Um outro axioma interessante é o axioma $T : \Box_i A \to A$ que não é válido em K mas é claramente desejável, pois T afirma que se algo é necessário, estamos falando de algo que deve ser aceito em todas as situações que possamos imaginar (ou tivermos acesso), especialmente a situação atual. Esse axioma exprime exatamente a noção de reflexividade, se $\Box_i A$ em um mundo qualquer é interessante que A seja válido neste mesmo mundo. A lógica modal KT resulta da adição do axioma T a lógica modal K. Por outro lado, numa interpretação em termos de crença, $\Box_i A$ indica que o agente i acredita em A mas isto não

Axioma	*Fórmula Correspondente*
K	$\Box_i(A \to B) \to (\Box_i A \to \Box_i B)$
T	$\Box_i A \to A$
D	$\Box_i A \to \Diamond_i A$
4	$\Box_i A \to \Box_i \Box_i A$
5	$\Diamond_i A \to \Box_i \Diamond_i A$
B	$A \to \Box_i \Diamond_i A$
C	$\Diamond_i \Box_i A \to \Box_i \Diamond_i A$
De	$\Diamond_i A \to \Diamond_i \Diamond_i A$
M	$\Box_i \Diamond_i A \to \Diamond_i \Box_i A$
L	$\Box_i((A \wedge \Box_i A) \to B) \vee \Box_i((B \wedge \Box_i B) \to A)$
3	$\Box_i(\Box_i A \to B) \vee \Box_i(\Box_i B \to A)$
X	$\Box_i \Box_i A \to \Box_i A$
F	$\Box_i(\Box_i A \to B) \wedge (\Diamond_i \Box_i B \to A)$
R	$\Diamond_i \Box_i A \to (A \to \Box_i A)$
G	$\Box_i(\Box_i A \to A) \to \Box_i A$
Grz	$\Box_i(\Box_i(A \to \Box_i A) \to A) \to A$
Go	$\Box_i(\Box_i(A \to \Box_i A) \to A) \to \Box_i A$
Z	$\Box_i(\Box_i A \to A) \to (\Diamond_i \Box_i A \to \Box_i A$
Zbr	$\Box_i(\Box_i A \to A) \to (\Box_i \Diamond_i \Box_i A \to \Box_i A$
Zem	$\Box_i \Diamond_i \Box_i A \to (A \to \Box_i)$
Dum	$\Box_i(\Box_i(A \to \Box_i A) \to A) \to (\Diamond \Box_i A \to \Box_i A)$
Dbr	$\Box_i(\Box_i(A \to \Box_i A) \to A) \to (\Box_i \Diamond_i \Box_i A \to \Box_i A)$

Figura 3.6: Axiomas e fórmulas correspondentes.

implica que A seja verdade. Se $\Box_i$ trata de "saber" então A é verdade.

Exemplo 3.2.1 Seja o modelo $M = \langle W, R_i, V \rangle$ onde $W = \{w_1, w_2, ..., w_n\}$ e $w_x R_i w_y$ se e somente se, $x = y$, ou seja, M é um modelo onde as relações de acessibilidade são apenas reflexivas.

Reescrevendo $\Box_i A \to A$ conforme 2.1.1 temos $\neg(\Box_i A \wedge \neg A)$. Então, conforme 2.2.2, temos:

$\not\models_w^M \neg(\Box_i A \wedge \neg A)$

não $\models_w^M \neg(\Box_i A \wedge \neg A)$ por 3

não não$\models_w^M (\Box_i A \wedge \neg A)$

$\models_w^M (\Box_i A \wedge \neg A)$ por 4

$\models_w^M \Box_i A \wedge \models_w^M \neg A$ por 3

$\models_w^M \Box_i A \wedge$ não $\models_w^M A$

para todo $w' \in W$ tal que wR_iw', $\models_{w'}^M A$ e não $\models_w^M A$ (*)

Como a relação de acessibilidade é reflexiva, (*) é uma contradição. Então concluímos que *não existe uma estrutura* que seja reflexiva onde a fórmula $\Box_i A \rightarrow A$ não seja válida, logo é válida em todas as estruturas onde a relação de acessibilidade seja reflexiva, o que não é o caso da lógica K onde a propriedade reflexiva não se verifica.

Muitos lógicos acreditam que KT ainda é muito fraca para formalizar a lógica da necessidade e da possibilidade. Eles recomendam outros axiomas para tratar da iteração, ou repetição de operadores modais. São os chamados *axiomas de iteração* e os axiomas 4 e 5 (veja figura 3.6) são exemplos [21]. O axioma 4 expressa a noção de transitividade e o axioma 5 a relação euclidiana.

Exemplo 3.2.2 Seja o modelo $M = \langle W, R_i, V \rangle$ onde $W = \{w_1, w_2, ..., w_n\}$ e onde a relação de acessibilidade é reflexiva e transitiva. Portanto, pela transitividade, se $w_x R_i w_y$ e $w_y R_i w_z$ então $w_x R_i w_z$ e pela reflexividade, é aceitável $x = y = z$. Reescrevendo $\Box_i A \rightarrow \Box_i \Box_i A$ conforme 2.1.1 temos $\neg(\Box_i A \wedge \neg\Box_i \Box_i A)$

Suponha que $\models_{w_1}^M \Box_i A$. Então, por 5 da definição 2.2.2, $\models_w^M A$ para qualquer $w \in M$, inclusive o próprio w_1 (pela reflexividade). E, se A é válida em **todos** os mundos do modelo, temos $\models_{w_2}^M \Box_i A$, $\models_{w_3}^M \Box_i A$, etc. Ou seja, $\models_w^M \Box_i A$ para todos os mundos w. E se $\Box_i A$ é válida é todos os mundos do modelo, concluímos que o mesmo vale para $\models_w^M \Box_i \Box_i A$. Seguindo o mesmo raciocínio, provamos $\models_w^M \Box_i \Box_i \Box_i A$ e $\models_w^M \Box_i \Box_i \Box_i \Box_i A$ e $\cdots$.

$S4$ é um sistema que resulta da adição do axioma 4 a lógica modal KT (poderia ser chamada de $KT4$). É possível provar que o sistema $S4$ possui relação de acessibilidade

reflexiva e transitiva. De modo similar $S5$ é formado pela adição do axioma 5 a KT. Pode-se provar que o sistema $S5$ possui relação de acessibilidade reflexiva e euclidiana (que é equivalente à ser reflexiva, simétrica e transitiva ou seja, uma relação de equivalência).

Em $K4$, a sentença $\Box_i\Box_i A$ é equivalente a sentença $\Box_i A$. Como resultado, qualquer seqüência de $\Box_i$ pode ser substituída por um único $\Box_i$, e o mesmo acontece com uma seqüência de $\Diamond_i$. Isto pode dar a impressão de que axiomas de iteração são supérfluos. Dizer que A é necessariamente necessário é considerado um modo prolixo de dizer que A é necessário. Em $S4$, uma seqüência de operadores do mesmo tipo pode ser substituída pelo uso do tal operador uma única vez.

O sistema $S5$ tem princípios mais fortes para simplificar seqüências de operadores modais. Em tal sistema, seqüências contendo os operadores $\Box_i$ e $\Diamond_i$ ao mesmo tempo são equivalentes ao uso do último operador em seqüência. Então, por exemplo, dizer: *é possível que A seja necessário* é o mesmo que dizer que A é necessário.

Dessa forma, para $S4$, $\Box_i\Box_i\cdots = \Box_i$ e $\Diamond_i\Diamond_i\Diamond_i\cdots = \Diamond_i$ e para $S5$, $MMMMM\cdots\Box_i = \Box_i$ e $MMMM\cdots\Diamond_i = \Diamond_i$ para cada M sendo um operador modal $\Box_i$ ou $\Diamond_i$.

O sistema B (em homenagem ao lógico Brouwer) é formado pela adição do axioma B (veja figura 3.6) ao sistema KT. O axioma B deixa clara a relação de simetria e por isso o sistema B possui as relações de acessibilidade reflexiva (devido a T) e simétrica (devido a B).

Um sistema que possui apenas a relação de acessibilidade simétrica é chamado KB e sua axiomatização é feita acrescentando-se o axioma B ao sistema K. O sistema de lógica modal onde modelos possuem relação de acessibilidade simétrica (devido a B) e transitiva (devido a 4) é o sistema $KB4$, mais conhecido como $B4$. O sistema que possui as relações simétricas, transitivas e reflexivas (devido a T) é o $KT4B$ que é uma outra axiomatização para o sistema $S5$. Isto ocorre porque é possível provar que uma relação reflexiva e euclidiana é de equivalência.

O axioma B surge como um ponto importante sobre a interpretação de fórmulas modais. B diz que "se A, então A é necessariamente possível". Pode-se argumentar que B deve sempre ser adotado em qualquer lógica modal, claro que "se A, então é necessário

que A seja possível".

Entretanto, há um problema com esta afirmação que pode ser exposto ao notar que a fórmula $\Diamond_i \Box_i A \rightarrow A$ é provável no sistema KB. Logo, $\Diamond_i \Box_i A \rightarrow A$ deve ser aceitável se o axioma B o é. Mas, $\Diamond_i \Box_i A \rightarrow A$ diz que se A é possivelmente necessário, então A é verdade, e tal afirmação não pode ser considerada verdadeira em todos os sistemas modais.

Interessante notar que o axioma B parece óbvio e uma fórmula provada no sistema KB nem sempre parece *totalmente aceitável*. Isso se deve ao fato de haver uma ambiguidade importante na interpretação natural para a fórmula $A \rightarrow \Box_i \Diamond_i A$.

Nós precisamos deixar evidente a forte ligação entre a axiomática e a semântica das lógicas modais. A tabela da figura 3.7 sumariza a relação existente entre alguns axiomas e a respectiva propriedade na relação de acessibilidade.

Axioma	*Relação de acessibilidade*
T	Reflexiva
D	Serial
4	Transitiva
5	Euclidiana
B	Simétrica
De	Densidade
C	Confluência

Figura 3.7: Axiomas e condições da relação de acessibilidade correspondentes.

Na figura 3.8 encontramos um resumo da axiomatização de alguns importantes sistemas modais.

Sistema Modal	*Axiomas*	*Estruturas onde é válida*
K	K	Todas as estruturas
$K4$	$K+4$	Nas estruturas transitivas
T	$K+T$	Nas estruturas reflexivas
B	$K+B$	Nas estruturas simétricas
KD	$K+D$	Nas estruturas onde vale a serialidade
$S4$	$K+T+4$	Nas estruturas reflexivas e transitivas
$S5$	$K+T+4+B$	Nas estruturas onde haja uma relação de equivalência

Figura 3.8: Relação entre semântica e axiomática.

3.3 Lógicas multimodais

Durante os capítulos 2 e 3, nós apresentamos os operadores modais indexados por um índice i ($\Box_i$) e deixamos vago sua importância. Nesta seção, mostraremos sua função na caracterização de lógicas multimodais e na representação do conhecimento.

As lógicas multimodais são particularmente adequadas para representação de combinação de noções de conhecimento, crença, tempo e ação.

Podemos expressar, por exemplo, que *um agente i sabe do conhecimento do agente j* por meio da fórmula:

$$\Box_i \Box_j A$$

onde A representa o conhecimento do agente j e $\Box_j A$ o conhecimento do agente i.

Podemos ainda expressar que depois de executada a *ação i*, A é verdade, e que depois de executada a ação j seguida da i, A é verdade:

$$\Box_j \Box_i A$$

Um sistema é chamado de sistema multimodal homogêneo se para todos os operadores indexados, a axiomatização do sistema correspondente é a mesma. Caso contrário, o sistema é dito multimodal heterogêneo e então, nesses casos, a cada operador modal corresponde uma axiomatização.

Há momentos em que pode ser necessária a inclusão de *axiomas de interação*, como por exemplo, quando queremos expressar que um agente sabe sobre o conhecimento de outro agente:

Um axioma de interação típico - conhecido como axioma da inclusão - é $I_{i,j} : \Box_i A \rightarrow \Box_j A$ que significa, em uma interpretação doxástica, que tudo o que é crido pelo agente i também é pelo agente j. Ou em uma interpretação em termos de conhecimento, que tudo que o agente i conhece, o agente j também conhece.

Um outro axioma de interação é o da transitividade mútua: $4M_{i,j} : \Box_i A \rightarrow \Box_j \Box_i A$ que expressa que o agente j está ciente do conhecimento de i.

Em termos de lógica temporal, pode ser necessário o uso do axioma de persistência sugerindo que $\Box_i A$ é "i sabe A" e $\Box_t A$ é "A sempre é verdade" $P_{i,j} : \Box_i \Box_t A \rightarrow \Box_t \Box_i A$ que pode expressar que se o agente i sabe que A sempre acontece então sempre i saberá que A acontece.

Assim como nas lógicas monomodais (com um operador modal apenas), onde a definição do sistema é apresentado em função da definição de seu conjunto de axiomas, os sistemas multimodais também o são. Por exemplo os dois sistemas a seguir: $K_i + T_i + 5_i$ e $K_i + T_i + 4_i + K_j + D_j + I_{i,j}$ são chamados de $KT5_i$ e $KT4_i + KD_j + I_{i,j}$.

A principal característica de sistemas multimodais está na facilidade de expressar modalidades complexas. Isto se faz pela composição de diferentes tipos de operadores modais, permitindo projetar situações onde agentes podem ter diferentes tipos de raciocínio e diferentes modos de interação entre tais raciocínios e, também, estudar simultaneamente várias modalidades (conhecimento e tempo ou conhecimento e crença).

Vamos considerar o exemplo a seguir, que mostra um sistema multimodal com modalidades representando ações e crenças de agentes. Tal exemplo, que ilustra uma aplicação de uma lógica multimodal na representação do conhecimento, foi apresentado em [13] e se inspira na fábula "a raposa e o corvo", na qual a raposa quer comer o queijo que está no bico do corvo. Para facilitar a leitura $\Box_{raposa}$ será escrito $[raposa]$.

Exemplo 3.3.1 Seja $[raposa]$ um operador modal axiomatizado somente pelo axioma K representando a crença de uma raposa. Seja $[elogia]$ e $[canta]$ dois operadores do tipo K representando respectivamente as ações onde a raposa elogia o corvo e aquela onde o corvo canta. Consideremos também, o operador $[sempre]$ do tipo $KT4$:

(A_1) $T[sempre] : [sempre]A \rightarrow A$

(A_2) $4[sempre] : [sempre]A \rightarrow [sempre][sempre]A$

Para melhor caracterizarmos este exemplo, vamos assumir os eguintes axiomas de transitividade mútua para expressar o fato que se A é sempre verdade, então continua sendo verdade depois de executadas as ações elogiar e cantar.

(A_3) $4M[sempre][elogia] : [sempre]A \rightarrow [elogia][sempre]A$

(A_4) $4M[sempre][canta] : [sempre]A \rightarrow [canta][sempre]A$

A representação das idéias de que: (1) a raposa acredita que se elogiar o corvo, então ele ficará encantado; e (2) significa que a raposa acredita que no momento em que o corvo estiver encantado será possível que ele cante e então deixe o queijo cair:

(1) $[raposa][elogia]encantado\_corvo$

(2) $[raposa][sempre](encantado\_corvo \rightarrow \langle canta\rangle caido\_queijo)$

Desta teoria podemos provar que "a raposa acredita que elogiar o corvo pode fazê-lo cantar e então soltar o queijo", ou seja:

$[raposa][elogia]\langle canta\rangle caido\_queijo$

Nestes capítulos vimos as lógicas modais normais, aquelas cujo sistema minimal é o sistema K. Vimos ainda a ligação entre os axiomas das lógicas modais e as propriedades das relações de acessibilidade e que para obter outros sistemas modais a partir de K, basta variar as propriedades da relação de acessibilidade variando os axiomas. O mesmo se aplica às lógicas multimodais que são generalizações das lógicas monomodais.

No exemplo 3.3.1 mostramos como usar lógicas multimodais na representação do conhecimento que é a solução de parte do problema do uso de IAS na representação do conhecimento.

Como dissemos na introdução, a outra parte do problema é fazer inferências a partir do conhecimento representado e isso será apresentado no próximo capítulo.

Capítulo 4

Tableaux modais com regras de propagação e regras estruturais

Neste capítulo apresentaremos sistemas de tableau semânticos para as lógicas modais. Recomendamos [16] e [21] para uma leitura mais aprofundada sobre o assunto.

Nos últimos 20 anos surgiram trabalhos no sentido de apresentar métodos efetivamente automáticos de prova para a lógica modal. Sistemas de resolução para várias lógicas modais foram apresentados a partir do trabalho original de Fariñas del Cerro [14].

Em uma outra linha, vários trabalhos foram apresentados no sentido de traduzir as lógicas modais para a lógica clássica, com o objetivo de permitir a aplicação dos sistemas de prova automática já consolidados, notadamente resolução, então sua resposta era traduzida de volta para a lógica modal [13].

Contudo, tais metodologias mostraram-se limitadas e difíceis de se generalizar. A família de métodos que melhor se integraram às famílias de lógicas modais foram os métodos de tableau e de seqüentes. Ambos são métodos de prova por refutação: a partir de uma fórmula A e de um conjunto de regras bem definidas, obtém a forma disjuntiva de subsentenças de $\neg A$ em uma estrutura de árvore de modo que cada disjunto ocupe um ramo da árvore. A seguir procura-se contradições nas subsentenças em todos os ramos da árvore de prova construída [21].

A história da construção de tais árvores pode ser traçada através de duas direções:

uma semântica (tableaux) e outra sintática (seqüentes).

A diferença básica entre os tableau e os seqüentes é que o primeiro é semântico e a preocupação principal está no resultado da prova, a ordem em que as regras são aplicadas (estratégia) pode ser alterada. Já os seqüentes são mais rigorosos, mais formais havendo sempre uma ordem na aplicação das regras que deve ser obedecida.

A abordagem sintática começou com o trabalho de Gerhard Gentzen [19] e com os trabalhos de Curry [9], Ohnishi e Matsumoto [36]. Os autores ainda não tinham intuição semântica da lógica modal para guiá-los e por isso a abordagem foi essencialmente sintática, o que dificultou o trabalho.

O método de tableau semântico para a lógica proposicional surguiu em meados dos anos 50 com os trabalhos de Hintikka [27] e Beth [1]. Mais tarde, Smullyan [42] apresentou uma notação uniforme para os trabalhos de Beth e Hintikka. Não houve tentativa de adaptar tableau para a lógica modal enquanto não surgia uma semântica adequada para ela.

Os trabalhos de Fitting [16] e Goré [21] consolidaram os métodos de tableau para as lógicas modais e hoje são os mais utilizados para essas lógicas. Fitting adaptou os trabalhos de Smullyan para a lógica modal enquanto, Goré segue a linha dos trabalhos de Hintikka.

Os dois métodos de prova usam regras que propagam fórmulas em uma árvore de prova a fim de que tal árvore simule as propriedades de um modelo de Kripke.

O problema desses métodos é que suas árvores de prova são muito restritivas pois nelas a propagação das fórmulas é somente top-down, de modo que eles falham quando se tenta projetar, por exemplo, regras de tableau para uma lógica modal cujas propriedades da relação de acessibilidade estão baseadas em axiomas de *densidade* ($\Diamond_i p \rightarrow \Diamond_i \Diamond_i p$) ou de *confluência* ($\Diamond_i \Box_i p \rightarrow \Box_i \Diamond_i p$).

Outro problema é que tanto Goré quanto Fitting, em nome da funcionalidade, criaram regras de tableau para cada sistema modal. Por exemplo, para o sistema K existe uma regra chamada K que representa o axioma K e para o sistema $S4$ há uma regra chamada

$S4$ que agrupa os axiomas K e 4, o que torna difícil um provador genérico.

A solução que foi apresentada por Castilho *et al*[4] generalizou a estrutura adjacente ao tableau para um grafo e criou regras para representar as propriedades das relações de acessibilidade.

Essa nova maneira de representar tableau em um grafo acíclico dirigido com um nó raiz (RDAG) onde a propagação das fórmulas não precisa mais ser exclusivamente *top-down*, permitiu projetar regras para relações de acessibilidade com propriedades mais complexas, como por exemplo, aquelas que envolvem axiomas de interações entre densidade ou confluência.

Esta nova filosofia deixou o método mais modular o que permite a implementação de um provador genérico. Para garantir a modularidade, tais regras foram divididas em dois tipos:

- Regras de propagação – Se em um nó há um determinado padrão de fórmula então, conforme a regra para este padrão, a fórmula é propagada para um outro nó relacionado a este (a mesma ou outra fórmula gerada pela regra). Tais regras incluem, por exemplo, os axiomas T, 4, B e 5 (propriedades da reflexividade, transitividade, simetria e euclidianidade, respectivamente) e formam o grupo 1;

- Regras estruturais – Essas regras só podem ser aplicadas quando existe um determinado padrão de relação entre nós. Adiciona-se um(s) novo(s) nó(s) e aresta(s) conforme a regra para tal padrão. Exemplos de tais regras são as relativas aos axiomas D, De e C (respectivamente as propriedades da serialidade, densidade e confluência) e formam o grupo 2.

As regras para os conectivos clássicos e para $\Diamond_i$ são comuns a todos os sistemas modais.

Desta maneira, o método é bem mais genérico, porque as regras foram criadas de modo a simular as propriedades da relação de acessibilidade das lógicas modais. Assim, para provar se uma fórmula é verdade em uma lógica modal, basta tomar as regras que representam os conectivos clássicos e o $\Diamond_i$ e acrescentar as regras que representam as propriedades da relação de acessibilidade da lógica modal em questão.

Assim, é possível fazer uma correspondência biunívoca entre o nome que caracteriza a lógica modal, o axioma correspondente e a regra de tableau que deve ser incluída no provador.

4.1 Notação e terminologia

Nesta seção, apresentamos as convenções e notações para posterior apresentação do método:

Definição 4.1.1 (convenções notacionais)

1. $\perp$ denota a constante proposicional *falso* e $\emptyset$ denota o conjunto vazio;
2. p, q denotam proposições atômicas;
3. $A, B, C, ...$ 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;

Definição 4.1.2 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-satisfatível se, e somente se, é satisfatível em um ρ-modelo. Tal fórmula é ρ-válida, se e somente se, é válida na classe de todos os ρ-modelos e será representada por $\models_\rho A$. Assim, A é um teorema de um sistema representado por um conjunto de propriedades se e somente se é ρ-válida.

O método de tableau que será em seguida apresentado é baseado em grafo acíclico orientado e com um nó raíz (RDAG) com algumas propriedades adicionais; seja ρ o conjunto dessas propriedades adicionais, define-se:

Definição 4.1.3 Um ρ-RDAG é uma tripla ($\mathcal{N}$, Σ , FOR) onde:

- $(\mathcal{N}, \Sigma)$ é um grafo acíclico orientado (DAG), i.e. um grafo orientado que não contém ciclos, com um nó distinto chamado raiz que pode acessar todos os nós em um feixe transitivo de Σ;
- $(\mathcal{N}, \Sigma)$ 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.

Para simplificar a notação, não é feita distinção entre os nós e seus conjuntos de fórmulas associados, de modo que se escreve $A \in x$ ao invés de $A \in \text{FOR}(x)$. Pelo mesmo motivo, não é feita distinção entre um ρ-RDAG$(\mathcal{N}, \Sigma)$ e a relação binária Σ, o que significa que também não é feita distinção entre estruturas rotuladas e estruturas.

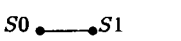
Tal noção se estende para grafos:

Definição 4.1.4 Um RGRAPH é um grafo que tem uma raiz, e um ρ-RGRAPH é um RGRAPH que satisfaz todas as propriedades de ρ.

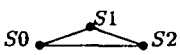
Como usual, $\Sigma(x)$ representa o conjunto dos nós acessíveis a partir de x por Σ : $\Sigma(x) = \{y \in \mathcal{N} : (x,y) \in \Sigma\}$. Σ^n representa os pares (x, y) de forma que há um caminho de comprimento n entre x e y. A relação diagonal $\{(x, x) : x \in \mathcal{N}\}$ pode ser representada por I ou por Σ^0.

Com o objetivo de facilitar a visualização, existe uma representação gráfica para RDAG onde as arestas são implicitamente orientadas da esquerda para a direita.

denota um nó S

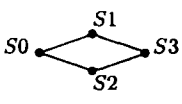


denota $(S0, S1) \in \Sigma$



denota $(S0, S1), (S0, S2), (S1, S2) \in \Sigma$

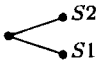
denota $(S0, S1), (S0, S2) \in \Sigma$



denota $(S0, S1), (S0, S2), (S1, S3), (S2, S3) \in \Sigma$

Os dois últimos diagramas não envolvem ordem entre $S1$ e $S2$, por exemplo,

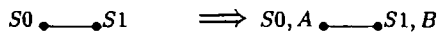
pode tranquilamente ser representada como



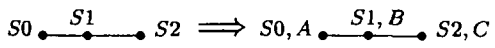
Algumas vezes também, uma parte do grafo pode ser reescrita na aplicação de uma regra e nesses casos a representação dever ser como a seguir. O grafo antes da aplicação da regra está do lado esquerdo da flexa dupla e é pré-condição para a aplicação da mesma. O grafo resultante da aplicação da regra está do lado direito da flexa dupla.

reescreve o nó S como $S \cup \{A\}$, i.e. adiciona a fórmula A ao nó S,

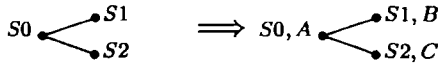
adiciona o novo nó $S1$ aos sucessores do nó S,



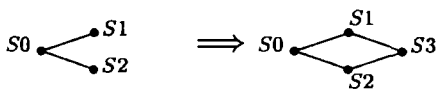
adiciona a fórmula A no nó $S0$ e a fórmula B em $S1$,



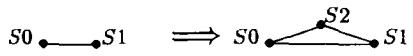
adiciona a fórmula A a $S0$, B a $S1$ e C a $S2$,



adiciona a fórmula A ao nó $S0$, B em $S1$ e C em $S2$,

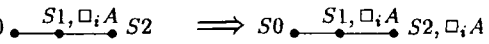


cria um novo nó $S3$ e o define como um mesmo sucessor de $S1$ e $S2$,



adiciona o novo nó $S2$ entre $S0$ e $S1$.

Esta representação permite levar em conta restrições de aplicabilidade das regras, por exemplo, uma regra da forma:



é lida "se $S1$ é um nodo relacionado com um nodo antecessor e um sucessor e, ainda, contém uma fórmula $\Box_i A$ no conjunto definido pelo nó $S2$ então, propaga a fórmula $\Box_i A$ para o nó sucessor".

4.2 Regras de propagação e regras estruturais

Nesta seção apresentamos as regras de tableau que aqui aparecem divididas em 3 grupos. O grupo das regras clássicas mais a regra $\Diamond_i$ é sempre utilizado. Os grupos das regras de propagação e estruturais, rotulados como "Grupo 1" e "Grupo 2" respectivamente, fornecem regras que simulam as propriedades das relações de acessibilidade.

Apesar do conectivo $\vee$ ser definido a partir de $\wedge$ e $\neg$, sua regra é mostrada abaixo.

As regras de tableau são:

- Regras clássicas e $\Diamond_i$:

  - Regra $\bot$: $A, \neg A, S \Longrightarrow A, \neg A, \bot, S$

  - Regra $\neg$: $\neg\neg A, S \Longrightarrow \neg\neg A, A, S$

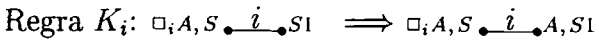
  - Regra $\wedge$: $A \wedge B, S \Longrightarrow A \wedge B, A, B, S$

  - Regra $\vee$: $\neg(A \wedge B), S \Longrightarrow \neg(A \wedge B), C, S$

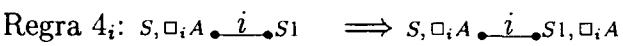
    onde C é $\neg A$ ou $\neg B$

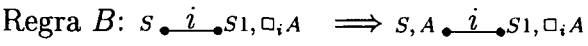
- Regra $\Diamond_i$: 

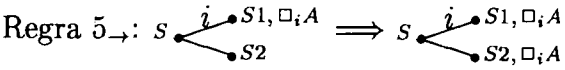
- Regras de propagação (Grupo 1):

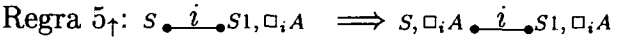
  - Regra K_i: 

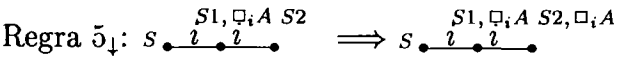
  - Regra T_i: 

  - Regra 4_i: 

  - Regra B: 

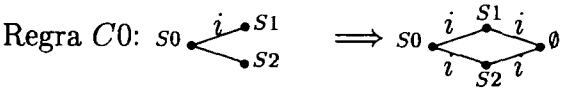
  - Regra $5_{\rightarrow}$: 

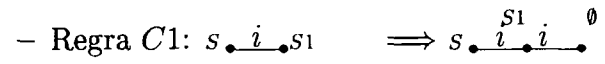
  - Regra $5_{\uparrow}$: 

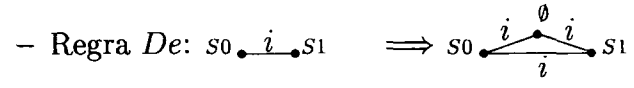
  - Regra $5_{\downarrow}$: 

- Regras estruturais (Grupo 2):

  - Regra D: 

  - Regra $C0$: 

– Regra $C1$: 

– Regra De: 

Ao definir um cálculo de tableau para um sistema denotado por $\rho_1 \cup \rho_2$, devemos associar um conjuntos de regras. Todos os cálculos de tableau que podem ser definidos a partir dessas regras devem conter as regras clássicas, $\Diamond_i$ e K e uma ou mais regras estruturais e/ou de propagação que são escolhidas conforme as propriedades da relação de acessibilidade da lógica modal em questão.

A figura abaixo ilustra um comparativo entre as propriedades relacionais e as regras de tableau:

	Propriedades	*Regras*	
Grupo 1	Ref	T	Regras
	Sym	B	de
	Tr	4	Propagação
	Eucl	$5_\uparrow 5_\downarrow 5_\rightarrow$	
Grupo 2	Ser	D	Regras
	Dens	De	Estruturais
	Conf	$C0C1$	

Figura 4.1: Comparativo entre propriedades relacionais e regras de tableau

Note que o axioma 5 é representado pelas regras $5_\uparrow$, $5_\downarrow$ e $5_\rightarrow$ e o axioma C pelas regras $C0$ e $C1$ o que significa que quando a lógica em questão possui a propriedade euclidiana, todas as regras 5 devem ser aplicadas e o mesmo ocorre com a confluência e as regras $C0$ e $C1$.

A semântica das lógicas modais e a axiomática estão embutidas nas regras de tableau. Observe a regra K_i: se existe um mundo S relacionado com um mundo S_1 e se em S a fórmula $\Box_i A$ é válida, deve-se propagar A para S_1. O que simula o axioma K.

Tomando a regra T_i: se em um mundo S a fórmula $\Box_i A$ é válida, propague a fórmula A no próprio mundo S. O que simula propriedade da reflexividade que diz que se o mundo atual é acessível a partir de si mesmo, então para cada fórmula $\Box_i A$ deve haver

uma fórmula A correspondente no mesmo mundo.

A regra B_i é aplicável se há um mundo S e um mundo $S1$ acessível a partir de S e se em $S1$ a fórmula $\Box_i A$ é válida, então propague a fórmula A para o mundo S. O que simula a propriedade simétrica: sejam dois mundos possíveis S e $S1$ e se SR_iS1, então $S1R_iS$ e logo para cada fórmula $\Box_i A$ válida em $S1$ deve haver uma fórmula A correspondente em S.

A condição para aplicar a regra 4 é a existência de dois mundos S e $S1$ onde $S1$ seja acessível a partir de S e ainda, se em S a fórmula $\Box_i A$ se verifica, propague a fórmula $\Box_i A$. Essa regra simula o axioma 4 e, conseqüentemente, a propriedade da transitividade.

Da mesma maneira as outras regras simulam outras propriedades da relação de acessibilidade.

Definição 4.2.1 Um tableau é dito "fechado" se pelo menos um nó contém $\perp$; do contrário é dito "aberto". Uma fórmula é fechada para um tableau se e somente se todos os seus tableaux estão fechados.

4.3 Exemplos de provas

Com o objetivo de ilustrar a aplicação das regras de tableau aos conjuntos de fórmulas, nesta seção apresentamos alguns exemplos de prova.

Ao iniciar uma prova, devemos escolher a fórmula a ser provada e o sistema modal no qual se deseja verificar a validade de tal fórmula.

Depois, tomamos as regras clássicas, $\Diamond_i$ e K_i (por serem comuns a todos os tableaux) e escolhemos as regras que serão aplicáveis à fórmula conforme as propriedades da relação de acessibilidade inerentes ao sistema escolhido, ou os axiomas que formam a axiomatização deste sistema.

Por fim, escolhemos a estratégia, ou seja, a ordem de aplicação das regras.

Exemplo 4.3.1 Para verificar a validade da fórmula $\Box_i A \rightarrow A$ no sistema modal K_i, sabemos que precisamos das regras:$\perp$, $\neg$, $\wedge$, $\vee$, $\Diamond_i$ e K_i. O próximo passo é escolher a estratégia que, no caso pode ser a mesma ordem. A seguir, reescrevemos a fórmula

conforme a definição 2.1.1: $\neg(\Box_i A \wedge \neg A)$. Como o método é por refutação, reescrevemos a fórmula negada: $\Box_i A \wedge \neg A$.

$$\Box_i A \wedge \neg A\ (\wedge)$$
$$|$$
$$\Box_i A; \neg A$$

Notamos que depois de aplicada a regra $\wedge$ não há a possibilidade de aplicar nenhuma outra regra. Assim, tendo esgotado todas as possibilidades de aplicação de regra e não encontrando inconsistência, o tableau não é fechado, logo a fórmula $\Box_i A \rightarrow A$ não é um teorema de K_i.

Exemplo 4.3.2 Agora vamos verificar a validade da mesma fórmula $\Box_i A \rightarrow A$ no sistema modal $K_i T_i 4_i$ (popularmente conhecido como $S4_i$), sabemos que precisamos das regras:$\perp$, $\neg$, $\wedge$, $\vee$, $\Diamond_i$, K_i, T_i e 4_i. O próximo passo é escolher a estratégia que, no caso pode ser a mesma ordem. A seguir, encontramos a forma normal conjuntiva negada da fórmula : $\Box_i A \wedge \neg A$.

$$\Box_i A \wedge \neg A\ (\wedge)$$
$$|$$
$$\Box_i A; \neg A\ (T_i)$$
$$|$$
$$\Box_i A; \neg A; A\ (\perp)$$
$$|$$
$$\perp$$

Como, encontramos inconsistência, o tableau é fechado pra $S4_i$, logo a fórmula $\Box_i A \rightarrow A$ é um teorema de $S4_i$.

O método de tableau apresentado neste capítulo é modular pois as regras foram criadas para *simular* os axiomas ou as relações de acessibilidade das lógicas modais. O que deixou o método bastante genérico, permitindo assim, a implementação genérica de um provador de teoremas para lógicas modais.

Implementando as regras clássicas, as regras de propagação, as regras estruturais e um algoritmo que verifique a possibilidade de aplicação destas regras e verifique se o tableau

está ou não fechado, temos a implementação de um provador genérico onde o usuário fica livre para fazer vários testes com uma mesma sentença. A adequação e a completude deste método foram demonstradas em [4].

No próximo capítulo apresentamos a implementação de um provador genérico de teoremas baseado no método de tableau aqui apresentado.

Capítulo 5

GTTP: um provador modal genérico

Neste capítulo apresentaremos a implementação do GTTP[1], um provador genérico de teoremas para lógicas modais baseado em métodos de tableau [4].

A apresentação está dividida em seções nas quais explicaremos a arquitetura, a modelagem e a implementação do GTTP como um provador que permite provas em lógicas modais e multimodais a partir de um conjunto de regras de estratégias, conforme definido pelo usuário.

O GTTP implementa as idéias desenvolvidas em Castilho *et al* [4], segundo as quais é possível classificar as regras em dois tipos: as estruturais e as de propagação. Conforme dissemos no capítulo 4, isto permite que o usuário possa definir a lógica informando os axiomas que a definem e o provador usa as regras correspondentes.

5.1 Arquitetura do GTTP

A arquitetura geral do GTTP foi concebida em 3 módulos básicos: um *analisador léxico e sintático*, um *módulo controlador* e um *módulo de ferramentas de tableau*. Estes são módulos distintos mas que se relacionam durante as execuções do provador.

O módulo *analisador léxico e sintático* é responsável por verificar se o usuário digitou uma fórmula bem formada e seus procedimentos são invocados pelo módulo controlador de provas.

[1] *Generic Tableaux Theorem Prover*

A gramática aceita neste módulo foi implementada em YACC usando como reconhecedor léxico o LEX [31]. A seguir, a representação da gramática na forma BNF[2]:

Sentença → Sentença_Atomica | Sentença_Complexa

Sentença_Atomica → Constante | Variavel_Proposicional

Sentença_Complexa →(Sentença)

| Sentença conectivo Sentença

| ¬ Sentença

| [indice]Sentença

| ⟨indice⟩Sentença

Conectivo → ∧ | ∨ |→|↔

Constante → Verdadeiro | Falso

Variavel_Proposicional → $A \mid B \mid C \mid \cdots$

indice → $a \mid b \mid c \mid \cdots$

O módulo *controlador de provas* consiste do programa principal, o responsável pela entrada de dados, por invocar o analisador léxico e sintático, fazer as inicializações, criar a raiz do tableau, disparar a execução da prova e retornar o resultado da prova. Basicamente, este módulo atua como interface entre o usuário, o analisador léxico e sintático e os procedimentos de prova.

O módulo de ferramentas de tableau é justamente o que implementa os objetos das entidades envolvidas no processo de prova através do método de tableau, tal como apresentamos no capítulo 4. Este módulo é formado pelas classes: *Formula*, *Nodo* e *Tableau*.

Na figura 5.1 verificamos o esquema da arquitetura do provador através da ilustração do relacionamento entre os módulos o que compõem.

Nas seções 5.2, 5.3 e 5.4 veremos as classes que modelam uma estrutura de tableau e os procedimentos que executam a prova.

[2]Backus-Naur Form

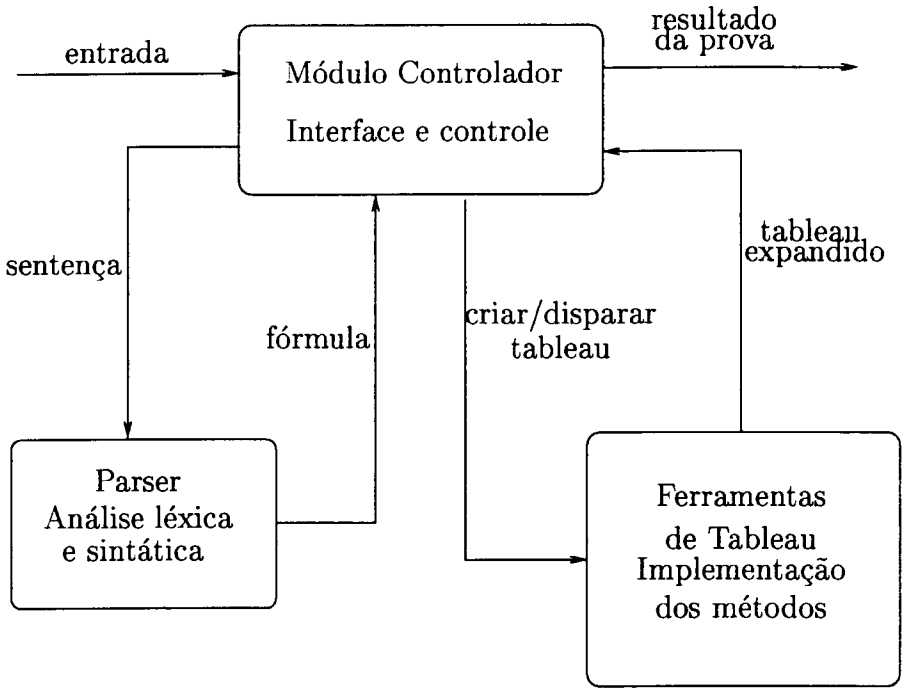


Figura 5.1: Arquitetura geral do GTTP

5.2 Classe Fórmula

A classe *Fórmula* efetua a modelagem de uma fórmula válida da lógica modal. Um objeto desta classe apresenta como atributos:

- *sentença* - um atributo do tipo *string* que constitui a palavra que representa a fórmula;
- *tipo* - identificador de tipo principal da fórmula (constante, conjunção, disjunção, negação, necessidade, ...) para fazer o casamento de padrões com as regras de tableau;
- *regra* - contém *true* se a regra de tableau correspondente ao tipo da fórmula foi aplicada a esta fórmula;
- *índice* - um atributo do tipo *string* para armazenar o índice do operador modal. Caso a fórmula seja de um tipo não-modal ou modal, mas não-indexada (caso monomodal), esse atributo continua vazio (pois assim é por *default*). Caso seja uma fórmula modal indexada, esse atributo recebe o conteúdo do índice do operador

modal da fórmula;

- *esquerda* - um ponteiro para a subfórmula à esquerda;
- *direita* - um ponteiro para a subfórmula à direita.

Estes dois últimos atributos servem para gerar a árvore de construção da fórmula. Tal árvore é gerada no início da execução da prova para que, a partir de uma fórmula, se tenha acesso a todas as suas subfórmulas. No caso de uma constante, que não possui subfórmulas, esses ponteiros apontam para *null*.

Para facilitar o entendimento, vamos observar a árvore de construção da fórmula $\langle a \rangle(\Box A \wedge [b]B \wedge \Diamond C)$ do exemplo 5.2.1 representada na figura 5.2.

Exemplo 5.2.1 A fórmula em questão é do tipo *possibilidade*, não possui fórmula à esquerda mas possui a subfórmula $\Box A \wedge [b]B \wedge \Diamond C$ à direita. Como o operador modal $\langle a \rangle$ é indexado, o índice da fórmula é a.
Analisando a subfórmula à direita: $\Box A \wedge [b]B \wedge \Diamond C$, verificamos que também é uma fórmula do tipo *conjunção*. Como conjunção não é um operador modal, o atributo índice é vazio. A subfórmula da esquerda é $\Box A$ e a da direita é $[b]B \wedge \Diamond C$.
As subfórmulas $\Box A$ e $[b]B \wedge \Diamond C$ também são fórmulas. A fórmula $\Box A$ é do tipo *necessidade* e não é indexada, sem fórmula à esquerda e com a fórmula A à direita. A fórmula A é uma *constante* e, portanto sem subfórmulas e sem índice.
A fórmula $[b]B \wedge \Diamond C$ é uma conjunção (e por isso com índice vazio) e com as subfórmulas $[b]B$ e $\Diamond C$.
$[b]B$ é uma fórmula do tipo *necessidade* com o índice b, sem subfórmula à esquerda e com a subfórmula B à direita. Como B é uma fórmula do tipo *constante* não possui subfórmulas nem índice.
$\Diamond C$ é uma fórmula do tipo *possibilidade*, sem índice, sem subfórmula à esquerda e com a subfórmula C à direita que, como toda constante, não possui índice nem subfórmulas.

Os métodos desta classe consistem de funções que retornam as subfórmulas e a negação da fórmula em questão. São eles:

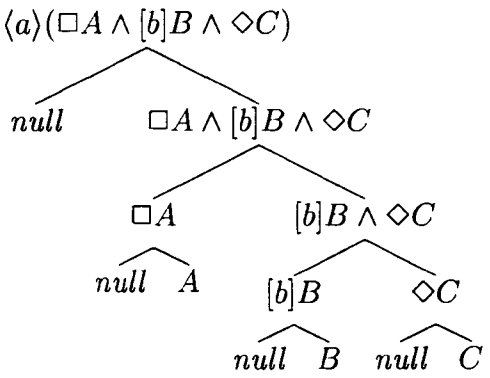


Figura 5.2: Árvore de construção da fórmula $\langle a\rangle(\Box A \wedge [b]B \wedge \Diamond C)$.

- *pegar_tipo()*, *pegar_regra()*, *pegar_índice()*, *pegar_sentença()*: retornam, respectivamente, o conteúdo dos atributos *tipo, regra, índice* e *sentença*;

- *setar_tipo()*, *setar_regra()*, *setar_índice()*, *setar_índice(índice)* e *setar_sentença(palavra)*: alteram o conteúdo dos atributos *tipo, regra, índice*(para *null* ou *índice*) e *sentença*, respectivamente;

- *pegar_formula_esquerda()*, *pegar_formula_direita()*: retornam os endereços das subfórmulas à esquerda e à direita, respectivamente;

- *neg()*, *ajuste()*: em conjunto, fazem os ajustes necessários à fórmula informada pelo usuário. Esses ajustes são feitos para que haja o casamento entre o tipo da fórmula e as regras de tableau e são por exemplo, a substituição de uma fórmula na forma $\neg\neg A$ por A, de $A \rightarrow B$ por $\neg(A \wedge \neg B)$ de modo que a fórmula de entrada tenha, no máximo, os conectivos $\wedge, \neg, \Box_i e \Diamond_i$ e, por fim, a negação da fórmula;

- *esta_em(formulas)*: é um método auxiliar para verificar se uma determinada fórmula está em um conjunto de fórmulas.

5.3 Classe Nodo

A classe *Nodo* contém as definições de um objeto do tipo *nodo*, que representa um nó da árvore de prova. Um *nodo* apresenta como atributos:

- *conjunto_de_formulas*: uma lista de fórmulas que são as sentenças resultantes da aplicação das regras de tableau;
- *nodo_esquerda*: um ponteiro para o nodo filho à esquerda;
- *nodo_direita*: um ponteiro para o nodo filho à direita;
- *historico4*, *historicoT*, *historicoD*: as duas primeiras são listas de fórmulas criadas para evitar ciclos nas aplicações das regras T e 4 e a última para evitar que nas aplicações da regra $\Diamond$ sejam gerados mundos iguais.

Os métodos implementados por um objeto da classe *nodo* consistem de procedimentos que executam operações em conjuntos de fórmulas:

- *pegar_esquerda()*, *pegar_direita()*, *pegar_conjunto_formula()*, *pegar_historico()*, *pegar_historicoD()*, *pegar_historicoT()*: retornam, o conteúdo dos atributos *nodo_esquerda*, *nodo_direita*, *conjunto_de_formulas*, *historico*, *historicoD* e *historicoT*, respectivamente;
- *setar_esquerda(nodo_esquerda)*, *setar_direita(nodo_direita)*, *setar_conjunto_formula()*, *setar_historico()*, *setar_historicoD()*, *setar_historicoT()*: atualiza conteúdo dos atributos *nodo_esquerda*, *nodo_direita*, *conjunto_de_formulas* e históricos, respectivamente;
- *retirar(formula)*: retira *formula* do conjunto de fórmulas;
- *tirabox_X()*, *tiradiamond_X()*: retornam a fórmula sob o escopo dos operadore modais $\Box_i$ e $\Diamond_i$, respectivamente;
- *inconsistente()* : retorna *true* se encontrou a inconsistência no conjunto de fórmulas do nó;
- *adiciona_formula(formula)*: adiciona uma fórmula no conjunto de fórmulas do nó;
- *adiciona_historico4(formula)*, *adiciona_historicoD(formula)*, *adiciona_historicoT(formula)*: adicionam uma fórmula no conjunto dos históricos;

5.4 Classe Tableau

Na *classe tableau* está definida a estrutura apresentada por cada objeto *tableau* criado durante o procedimento de prova. Um tableau possui como atributos:

- *relação*: um vetor de *palavras* que armazena na primeira posição, o índice do operador que gerou o *tableau* atual e nas demais posições, os índices que estão relacionados com o primeiro através de axiomas de interação;
- *mundos_acessíveis*: um conjunto de mundos acessíveis, que são usados para simular as mudanças de mundo empreendidas durante a prova, na realidade é uma lista de apontadores para objetos do tipo *tableau*;
- *raiz*: um objeto do tipo *nodo*, que constitui a raiz árvore de prova dentro do tableau em questão;
- *estratégia* e *aplica_regra*: o primeiro é um vetor de ponteiros para os métodos que verificam a aplicabilidade de uma regra, o segundo é um vetor de ponteiros para os métodos que aplicam as regras. Esses vetores são inicializados conforme a estratégia escolhida pelo usuário;
- *fechado*: contém *true* se o tableau em questão está fechado, isto é, quando encontramos $\perp$.

Os métodos definidos para todos os objetos do tipo *tableau* são, na realidade, as implementações das regras de tableau conforme definidas em [4], além de métodos para verificação da possibilidade de se aplicar determinada regra. Sendo assim, tem-se os métodos seguintes :

- *pegar_raiz()*: retorna o nó raiz do tableau atual;
- *setar_relação(índice)*: método que insere uma relação (índice) no vetor de relações de acessibilidade do tableau, esse método é utilizado como auxiliar na aplicação da *regra_i()*, referente aos axiomas de interação;

- *pegar_relação()*, *está_na_relacao(índice)*: retorna o conteúdo do vetor de relações de acessibilidade do tableau e verifica se a ação está no vetor de relações, respectivamente;

- *inicialização(EntraEstratégia)*: inicializa os atributos *estratégia* e *aplica_regra* com os endereços dos métodos que implementam as regras informadas na estratégia, que é definida pelo usuário em tempo de execução;

- *pegar_mundos_acessíveis()*: retorna o conteúdo do atributo *mundos_acessíveis*;

- *insere_mundo()*: insere um novo objeto tableau no conjunto *mundos_acessíveis*;

- *inconsistência* : este método varre o conjunto de fórmulas do nodo atual a procura de uma fórmula e sua negação e se encontrar, substitui as duas fórmulas pela constante *falso* ($\perp$). Ele está implementado para testar fórmulas complexas e não apenas fórmulas atômicas[3];

- *procura_e()*, *procura_ou()*, *procura_neg()*, *procura_pos()*, *procura_T()*, *procura_K()*, *procura_4()*, *procura_i()*: métodos que implementam funções booleanas de varredura na lista de fórmulas do nó atual à procura de uma fórmula onde possa ser aplicada uma determinada regra;

- *regra_e()*, *regra_ou()*, *regra_neg()*, *regra_diamond()*, *regra_K()*, *regra_T()*, *regra_4()*, *regra_i()*: métodos que implementam os algoritmos de aplicação das regras *e*, *ou*, *negação*, *possibilidade*, K, T, 4, e regra i, respectivamente, na fórmula pertencente ao conjunto de fórmulas do nó atual, a fórmula onde vai ser aplicada a regra será informada pelos métodos anteriores; Estes métodos foram implementados seguindo fielmente as idéias de Castilho *et al*[4];

- *expande()* e *expande(nodo, regra)*: o primeiro método inicializa um novo objeto tableau e o segundo é o método responsável pela execução da prova e será melhor descrito na seção 5.5.

[3] A maior parte das implementações executa este teste apenas em fórmulas atômicas.

A maneira como a classe *tableau* foi projetada garantiu a implementação genérica do GTTP, pois nessa classe foram criados os métodos que implementam as regras de tableau, os vetores de ponteiros para estes métodos e o método que controla a execução da prova conforme a lógica e estratégia informadas pelo usuário.

GTTP é genérico no tocante à lógica modal e à estratégia de ordem de aplicação das regras mas a busca na árvore é feita em profundidade.

Na próxima seção apresentamos a maneira como as classes descritas nesta seção foram implementadas.

5.5 Implementação

Nesta seção veremos como está implementado o GTTP, segundo as especificações explicadas na seção anterior. Basicamente, mostramos como resolver os principais problemas que surgem da especificação.

O GTTP foi implementado a partir de dois provadores específicos para as lógicas modais K e $S4$ [44]. Desses provadores, o GTTP manteve a mesma arquitetura e modelagem, aproveitou alguns métodos que foram apenas adaptados e outros totalmente remodelados como, por exemplo, os que controlam a entrada de dados e os métodos responsáveis pela execução da prova.

Ao começar a execução do provador o usuário deve informar a fórmula a ser provada, os axiomas do sistema modal e a estratégia de aplicação das regras. No exemplo 5.5.1 mostramos a entradas de dados para executar duas provas no GTTP.

A primeira linha contém a entrada para verificar se a fórmula $\Box A \rightarrow B \vee C$ é um teorema da lógica modal K segundo a estratégia de aplicação de regras $\&|\neg < k$ (aplica exaustivamente a regra *e*, em seguida exaustivamente a regra *ou*, em seguida exaustivamente a regra *negação*, em seguida exaustivamente a regra *diamond* e, por último, exaustivamente a regra K.

A segunda linha é a entrada para verificar se a fórmula $\Box A \rightarrow A$ é um teorema em $KT4$, seguindo a estratégia de aplicação exaustiva das regras *e*, *ou*, *negação*, *diamond*, *k*

e 4, nesta ordem.

Exemplo 5.5.1 (Entrada de dados no GTTP)

```
[]A->B|C   ; k    ; &|~<k    ;
[]A->A     ; kt4  ; &|~t<k4  ;
```

A função *main()* ativa a execução do *analisador léxico e sintático* através da função *yyparse()*[4] . Além da análise léxica e sintática da fórmula é feita a inicialização dos atributos dos objetos *fórmula* necessários para a representação da fórmula em questão.

Após a verificação da fórmula, o processo de prova se inicia pela criação de um objeto *tableau*. Isso é feito na função *main()* pela chamada do método *inicializaçao()*.

O método *inicialização()*, da classe tableau, recebe a estratégia armazenada na variável *EntraEstrategia*[5] e inicializa os vetores *estratégia* e *aplica_regra* que, conforme definidos na seção anterior, são, respectivamente, um vetor de ponteiros para os métodos que verificam a possibilidade de aplicação de uma regra e um vetor de ponteiros para os métodos que de fato aplicam as regras.

Essas estruturas foram escolhidas porque permitem a implementação genérica como nos propomos, uma vez que, são inicializadas em tempo de execução conforme a estratégia escolhida pelo usuário.

Os métodos para verificação e aplicação de uma mesma regra estão na mesma posição nos dois vetores. Por exemplo, se o usuário deseja que seja aplicada a regra $\wedge$ primeiro, a primeira posição do vetor *estratégia* contém o endereço do método *procura_e()* e a primeira posição do vetor *aplica_regra()* contém o endereço do método *regra_e()*.

Optamos por criar dois vetores similares porque durante a execução da prova, o método *expand(nodo, regras)* contém um laço que varre o vetor *estratégia* ativando a execução dos métodos apontados por este, incrementando a variável *regras*.

Esta variável representa o índice dos vetores, até que um método apontado pelo vetor retorne *true* para a aplicação da regra. Então, o *expand* chama a execução do método

[4]Função gerada pelo YACC a partir implementação da gramática com o reconhecedor léxico LEX.
[5]Essa variável contém a estratégia informada pelo usuário.

apontado pelo vetor *aplica_regra* na posição apontada pela variável *regra*.

O exemplo 5.5.2 demonstra o processo de inicialização desses vetores:

Exemplo 5.5.2 Se a estratégia desejada pelo usuário for aplicar todas as regras &, depois todas as regras | e, por último aplicar todas as regras K, ele deverá digitar $\&|k$. A ordem de teste de aplicação das regras será: a regra *e* (representada por &), a seguir regra *ou* (representada por |) e por fim a regra K. Então, *EntraEstrategia*=" $\&|k$" e o vetor *estrategia* é inicializado com os endereços dos métodos *procura_e()*, *procura_ou()* e *procura_k()*. Analogamente para o vetor *aplica_regra* com relação aos métodos *regra_e()*, *regra_ou()* e *regra_k()*, nesta ordem. Assim, *Estratégia*=[*&Procura_e*,*&Procura_ou*, *&Procura_k*] e *AplicaRegra* =[*&Regra_e*,*&Regra_ou*,*&Regra_k*].

Depois de realizadas as inicializações, é chamada a execução do método *expand()* que chama o método *toma_raiz()*. Este retorna o conteúdo do nó raiz do objeto tableau, inicializa a variável *regra*, que é o contador que controla qual regra está-se tentando aplicar, e evoca a execução do método *expand(nodo,regras)*. O método *expand()* é chamado no início da execução da prova em um novo mundo (objeto *tableau*), quando nenhuma regra foi aplicada ainda e o conjunto de fórmulas está todo no nó raiz.

Como os dois métodos estão tratando da expansão da árvore de prova, foi empregado o recurso de "sobrecarga de função" suportado pelo paradigma de programação orientada a objetos, que permite dar nomes iguais a funções com parâmetros diferentes.

O método *expand(nodo, regras)* recebe como parâmetros um ponteiro para o nó atual e um ponteiro para a variável *regras* que contém a informação sobre qual regra está sendo aplicada. Esse é o método que controla a execução da prova e, por isso, é melhor descrito na próxima seção.

Método expand(nodo, regras)

O método *expand(nodo,regras)* primeiro verifica se há inconsistência no nó atual chamando a execução do método *inconsistencia()* e, em caso afirmativo retorna *fechado = 1* e termina a prova. Caso contrário, aplica as regras de tableau correspondentes à lógica em questão

conforme estratégia descrita no vetor *estratégia*. Isto é melhor descrito a seguir para cada regra.

As regras clássicas, quando aplicadas, geram um novo nó do tipo *nodo* no tableau atual (a regra "ou" gera dois nós). Os atributos do novo nó são inicializados. O *conjunto de fórmulas* do novo nó recebe o conteúdo do nó atual menos a fórmula onde está sendo aplicada a regra, isso é feito com o método *retirar(formula)* da classe *nodo*.

A seguir, conforme a regra que está sendo aplicada, inserindo-se a nova fórmula no final do conjunto de fórmulas do novo nó. Então é chamada uma nova instância do método *expand(nodo, regras)* através de recursão, passando o novo nó e o valor "zero" como parâmetros a fim de se tentar aplicar regras no novo nó, desde a primeira escolhida pelo usuário.

Na figura 5.3 encontramos um trecho do código do método *expand()*.

Por exemplo, se o nó atual contém a seguinte lista de fórmulas: $A;B;\Box A;C \wedge D$; e o provador está aplicando a regra e sobre $C \wedge D$, então gera-se um novo nó, filho do nó atual; o atributo *nodo_esquerda* do nó atual recebe o endereço do novo nó; o atributo *conjunto_de_fórmulas* do novo nó recebe as seguintes fórmulas: $A;B;\Box A$; ao final do *conjunto_de_fórmulas* do novo nó são incluídas as fórmulas C (fórmula à esquerda de $C \wedge D$) e D (fórmula à direita de $C \wedge D$).

Especificamente na aplicação da regra *ou*, é criado um nó à direita e um nó à esquerda do nó atual. A prova continua sendo executada criando nós à esquerda até encontrar um nó inconsistente, aí todas as instâncias de *expand()* são desempilhadas até o nó onde foi aplicada a regra *ou* e a execução continua à direita.

Se for aplicada a *regra_*$\Diamond$ é criado um novo objeto tableau que representa um novo mundo relacionado ao tableau (mundo) atual. Esse relacionamento é feito através do atributo *mundos_acessíveis* que é uma lista dos objetos tableau relacionados com o objeto tableau atual.

Depois de aplicar a regra_$\Diamond$, optamos por continuar tentando aplicar regras no nodo atual e, consequentemente, no tableau atual. Só há avanço para o próximo mundo quando não há mais regras a ser aplicadas no tableau atual, ou seja, quando o vetor *estratégia* foi

```
//varre o vetor estrategia tentando encontar um método que
//retorne true
ENQUANTO (((this->*estrategia[regras])!=NULL) && (fechado != 0)){
  SE (((this->*estrategia[regras])(node, *formula,  r, &mundo))){
     PodeAplicar=1;
     break;
  }
   ++regras;
}
//se encontrou uma regra que pode ser aplicada
SE PodeAplicar{
//se a regra aplicada alterou um mundo acessível e encontrou
//inconsistência vai para o novo mundo
SE ((this->*AplicaRegra[Rules])(node, *formula, r, &mundo)){
   expand((*mundo).pega_raiz(), regras);
 }
SENÃO
//se existe nó à esquerda
  SE (&(node.pega_esquerda())!=NULL){
      regras=0;
      //expansão à esquerda a partir da primeira regra
      expand(node.pega_esquerda(), regras);
      //se encontrou inconsistencia a esquerda e tem nó a direita
      //caminhar à direita
      SE ((fechado)&& (&(node.pega_direita())!=NULL)){
        exclui os mundos possíveis gerados qdo da expansão à esquerda
        regras=0;
        expand(node.pega_direita(), regras);
      }
     }
     SENÃO
     {//se não existe nó à esquerda, continua tentando aplicar as regras neste nó
        regras=0;
        expand(node,regras);
      }
}
SENÃO//se não encontrou nenhuma regra a ser aplicada{
 //varrer mundos acessíveis
 SE ( &(pega_mundos_acessíveis()!=vazio){
    este_mundo;//var auxiliar
    PARA (este_mundo = mundos_acessíveis.início;
          este_mundo = mundos_acessíveis.fim; ++este_mundo)
    {
     (*este_mundo).inicializacao(&EntraEstrategia);
      //encontrando um mundo possível fechado, sai da recursão
      SE (fechado) RETORNA;
    }
     RETORNA;
 }
}
...
```

Figura 5.3: Trecho do código do método expand(nodo,regras).

varrido até o fim e nenhum método retornou *true*.

Essa maneira de caminhar nos mundos é uma restrição na estratégia, que não pode ser alterada pelo usuário. Sabemos que a melhor opção é mesmo só mudar de mundo depois de esgotada todas as aplicações de regra no mundo atual porque, caso contrário, corre-se o risco de chegar em um mundo com poucas fórmulas, em que não há regras a ser aplicadas e ter que voltar ao mundo anterior.

Ao aplicar cada regra clássica, o atributo lógico *regra* da fórmula em que foi aplicada a regra é atualizado para que não se tente aplicar a mesma regra na mesma fórmula. Quando todas as fórmulas da lista de fórmulas do nó atual estiverem com o atributo *regra*= "1", não há fórmula a ser aplicada naquele nó.

No caso da regra_$\Diamond$, o atributo *históricoD* é atualizado para que o método *expand()* não aplique a regra em fórmulas iguais, o que geraria mundos possíveis iguais comprometendo a eficiência do provador.

As regras T e 4 têm o problema de ocorrência de *looping*. Quando aplicada a regra 4 em fórmulas do tipo $\Box\Diamond A$, por exemplo, pode-se gerar uma ocorrência infinita da mesma fórmula através dos ramos. A solução adotada é a sugerida em [26], ou seja criamos os atributos *históricoT* e *historico4* que guardam as fórmulas em que já foram aplicadas as regras T e 4, respectivamente.

Se houver um outro nó no mesmo mundo, avança-se para o novo nó. Caso não haja nó à esquerda nem à direita, vai se efetuar a busca nos mundos relacionados ao mundo atual. Isto é feito através de um laço que varre a lista de mundos acessíveis do começo ao fim.

Cada regra de propagação e estrutural possui sua condição de aplicação. Por exemplo, para aplicar a regra K ou a regra 4, além de existir uma fórmula na forma $\Box A$ no mundo atual, é necessarío que exista um outro mundo possível relacionado a este. E essa verificação é feita checando se há elementos no atributo *mundos possíveis* do objeto tableau atual.

Para aplicar a regra T, basta que haja uma fórmula na forma $\Box A$. Para aplicar a regra B, é necessário que haja um mundo possível relacionado ao mundo atual onde haja

uma fórmula do tipo $\Box A$ e assim para as outras regras, conforme [4].

Foi implementada ainda a regra *i* que verifica se há axiomas de interação informados pelo usuário. Se o usuário informar o axioma de interação $[a]A \rightarrow [b]A$, significa que um tableau que possui a relação b no atributo relação, passa a ter também a relação a. Isto é feito varrendo-se os atributos *relação* de todos os objetos *tableau* da lista de mundos acessíveis a procura da relação b e ao encontrar, insere b no final do vetor *relação*.

Também foram implementadas as regras 5, C e De correspondentes ao axioma 5, ao axioma de confluência e ao axioma de densidade, respectivamente.

5.6 Testes

Com o intuito de comprovar a confiabilidade do GTTP, nesta seção apresentamos uma bateria de testes realizada para as lógicas monomodais K, KT, S4, S5 e K45 e para algumas lógicas multimodais.

Utilizamos diversas fórmulas diferentes e tomamos o cuidado de testar fórmulas que utilizassem diferentes regras e diferentes estratégia e aplicação das mesmas.

As fórmulas modais e multimodais avaliadas foram coletadas da literatura (notadamente dos exercícios propostos em [2],[37],[7],[25]) pois isto garante que o resultado correto seja conhecido.

Colocamos 90 fórmulas para serem provadas em lote. Tais fórmulas foram assim divididas:

- 30 fórmulas provadas em K;
- 21 em KT;
- 19 em $S4$;
- 10 em $S5$;
- 10 para algumas lógicas multimodais.

Cada linha do arquivo de entrada (mostrado no apêndice B) contém: a fórmula, os axiomas modais da lógica para cada modalidade que houver na lógica e a estratégia de

aplicação das regras, além do valor esperado sobre a validade da fórmula na lógica em questão.

O GTTP encontrou corretamente o resultado esperado em todas as fórmulas desta bateria de testes e o resultado gerou um arquivo de saída que é mostrado no apêndice C.

Na próxima seção traçamos um comparativo entre o GTTP e dois provadores conhecidos para lógicas modais.

5.7 Comparação com outros provadores

Nesta seção brevemente comparamos o GTTP com dois conhecidos provadores para lógicas modais: o LWB[25] e LOTREC[12].

The Logics Workbench

Desenvolvido na Universidade de Bern (Suíça), o *Logics Workbench - LWB*, é um provador baseado no cálculo de seqüentes que permite provas em diversas lógicas mas, ao contrário do GTTP, ele não é genérico. Além do mais, as provas são construídas automaticamente o que não permite ao usuário alterar estratégia de aplicação de regras ou de busca.

LWB suporta provas em diversas lógicas tais como: lógica clássica, intuicionista, lógica temporal, *tense logic* (pltl e tk), lógica proposicional linear (ll), algumas lógicas não-monotônicas (lógica autoepistêmica, circunscrição, hipótese de mundo fechado, lógicas *default*) e, daí nosso interesse, algumas lógicas modais. São elas K, KT, S4 e S5 e multi-modais kn, ktn e s4n.

O LWB é, na verdade, um pacote que reúne diversos provadores de teoremas distintos em um mesmo ambiente, cada provador é ativado pelo usuário carregando-se o módulo correspondente. Como cada módulo foi desenvolvido separadamente, o *LWB* não pode ser considerado um provador genérico.

Por outro lado, o LWB possui interface gráfica amigável e versões para diversas plataformas. Há uma versão *on-line* onde o usuário pode executar uma seção via web [29]. Também é possível obter o pacote por *ftp*, permitindo a instalação e uso local.

Como falamos no começo desta seção, o LWB não é um provador genérico como o GTTP e possui estratégia fixa, tirando a liberdade que é dada ao usuário do GTTP.

The Logical Tableau Research and Engineering Companion

O LOTREC - *Logical Tableau Research and Engineering Companion*, assim como o GTTP, está baseado no conceito de provador genérico para lógicas modais e está baseado na mesma filosofia de [4].

A principal vantagem do LOTREC sobre o GTTP está no fato do primeiro ter sido projetado para ser tão genérico a ponto de permitir ao usuário inserir novas regras de tableau em tempo de execução através de uma linguagem própria.

Na figura 5.4 observamos um exemplo de definição da linguagem:

```
//LINGUAGEM
connector falsum 0  true  "FALSUM"        4
connector and    2  true  "_ & _"         3
connector not    1  true  "~_"            5
connector nec    1  true  "[]_"           4
```

Figura 5.4: Definição da linguagem no LOTREC.

Observe por exemplo, o segundo conectivo: *and* é seu nome, 2 é o número de argumentos, as outras duas colunas servem para representação gráfica e 3 é a prioridade do conectivo em relação aos outros.

Na figura 5.5 podemos observar a definição das regras de tableau:

Para o LOTREC, a regra de tableau consiste de duas partes: *descritor* e *ação*. A primeira contém a aplicabilidade da regra e a segunda contém as operações no tableau.

O LOTREC ainda está em implementação mas há uma versão beta para *download* disponível em [15] e exemplos para executar provas em K e $K4$.

Ao contrário, o GTTP já está concluído e consegue realizar provas em qualquer lógica modal definida a partir de qualquer combinação do axioma K com os axiomas T, 4, 5, B, C (confluência) e D (densidade).

```
//REGRAS DE TABLEAU
// classical:
rule "and"
  descriptor hasElement node0   (and (variable A) (variable B))
  action add node0              (variable A)
  action add node0              (variable B)
end
rule "not not"
  descriptor hasElement node0   (not (not (variable A)))
  action add node0              (variable A)
end
rule "not and"
  descriptor hasElement node0   (not and (variable A) (variable B))
  action duplicate node0 begin node0 node1 end
  action add node0              (not variable A)
  action add node1              (not variable B)
end
// modal:
rule "diamond"
  descriptor hasElement node0   not nec (variable A)))
  descriptor isNotMarked node0  CONTAINED
  action newNode node0  node1
  action link node0     node1   (R)
  action add            node1   not (variable A)
end
rule "K"
  descriptor links node0 node1  (R)
  descriptor hasElement node0   (nec (variable A))
  action     add        node1   (variable A)
end
```

Figura 5.5: Definição das regras de tableau no LOTREC.

É interessante notar que o GTTP e o LOTREC são complementares, isto é, LOTREC possui a interface desenvolvida e o GTTP, o provador. Provador esse que é extensível para outras lógicas modais cujas regras de tableau ainda não estejam implementadas.

No GTTP, para obter-se tal adaptação basta inserir no código os métodos que implementem a verificação de aplicabilidade da nova regra e outro que efetue a aplicação dessa regra, o resto fica por conta do usuário que deve inserir as novas regras na hora de executar uma prova para essa nova lógica.

Essa facilidade de adaptação do GTTP para outras lógicas é abordada no capítulo 6 onde mostramos uma aplicação do GTTP no raciocínio sobre ações.

Capítulo 6

Uma aplicação do GTTP no raciocínio sobre ações

Neste capítulo nós mostraremos o potencial do provador genérico GTTP definido no capítulo anterior. Faremos isto integrando o poder representacional das lógicas modais com a flexibilidade permitida pelo GTTP e veremos como tratar cenários envolvendo raciocínio sobre ações. Nesta seção usaremos $[\alpha]A$ para representar $\Box_\alpha A$ apenas para facilitar a leitura, uma vez que usaremos nomes longos para as ações.

6.1 Uma lógica modal para raciocinar sobre ações

Os problemas da área de raciocínio sobre ações são geralmente modelados no conhecido formalismo do cálculo de situações [35]. Este é um cálculo de primeira ordem para o qual as inferências são feitas usando-se métodos circunscritivos, o que significa que estão no indecidível cálculo de segunda ordem [34].

Castilho *et al* [6] apresentaram uma alternativa para esta teoria baseada numa lógica multimodal chamada $\mathcal{LAP}$ (Lógica de Ações e Planos), com a principal vantagem de esta possuir um cálculo decidível baseado em tableau, daí nosso interesse.

Basicamente, $\mathcal{LAP}$ é uma lógica com um conjunto de operadores modais representando ações. Seja ACT um conjunto de ações atômicas (como *andar* ou *abrir*). Então, para $\alpha \in ACT$, é usada a lógica K_α para se descrever o comportamento desta ação α. Assim,

temos uma família de lógicas modais K_α, onde $\alpha \in ACT$.

Com algumas convenções, esta lógica é ideal para raciocinar sobre ações. Por exemplo, a fórmula: $Porta\_Aberta \rightarrow [fechar\_porta]Porta\_Fechada$ representa o fato de que se a porta estiver aberta, executar a ação de fechar a porta nos leva a um estado onde a porta está fechada.

Em termos gerais $L \rightarrow [\alpha]A$ é lido: "Se L é verdade, então após α temos A".

Para representar leis que são válidas sempre, foi definido um operador $\Box$ do tipo $S4$. Assim, a fórmula: $\Box(L \rightarrow [\alpha]A)$, significa que "sempre que L for verdade, A é verdade após α". O operador $\Box$ é relacionado com cada $\alpha \in ACT$ por um axioma de interação: $\Box A \rightarrow [\alpha]A$. Sendo do tipo $S4$, garante-se que as fórmulas no escopo de $\Box$ serão propagadas em razão da iteração inerente ao axioma 4.

$\mathcal{LAP}$ é a lógica multimodal $\Box_{KT4} + K_{\alpha,\alpha \in ACT} + I_{\Box,\alpha}, \alpha \in ACT$.

Definição 6.1.1 ($\mathcal{LAP}$-modelos) Também chamados de modelos para $\mathcal{LAP}$, são quádruplas na forma $M = (W, (R_\alpha)_{\alpha \in ACT}, R_\Box, \tau)$ onde W é um conjunto de mundos, $R_\Box$ e cada R_α são relações binárias em W e τ é uma interpretação dos átomos.

As condições de verdade são as usuais, em particular:

- $\models^M_w [\alpha]A$ se para todo $w' \in W$: $wR_\alpha w'$ implica $\models^M_{w'} A$;
- $\models^M_w \Box A$ se para todo $w' \in W$: $wR_\Box w'$ implica $\models^M_{w'} A$.

Representar cenários clássicos que envolvem ações em $\mathcal{LAP}$ é trivial. Considere por exemplo, o cenário do tiro em Yale [22], onde alguém tem uma arma descarregada, tem disponível as ações para *carregar* a arma e para *atirar* com ela. O objetivo do cenário é provar que após as ações *carregar* a arma, e depois de um certo tempo *atirar* terem sido executadas, haverá a morte de uma pessoa.

Considerando que a pessoa esteja inicialmente viva, em $\mathcal{LAP}$ escrevemos assim, para representar a situação inicial e o comportamento das ações:

Exemplo 6.1.1 [cenário do tiro em Yale]

$$LEIS = \left\{ \begin{array}{l} \Box\langle esperar\rangle\top, \Box\langle carregar\rangle\top, \Box\langle atirar\rangle\top, \\ \Box[atirar]\neg Carregado, \\ \Box(Carregado \rightarrow [atirar]\neg Vivo), \\ \Box(\neg Vivo \rightarrow [atirar]\neg Vivo), \\ \Box((\neg Carregado \wedge Vivo) \rightarrow [atirar]Vivo) \\ \Box(Carregado \rightarrow [esperar]Carregado), \\ \Box(\neg Carregado \rightarrow [esperar]\neg Carregado), \\ \Box(Vivo \rightarrow [esperar]Vivo), \\ \Box(\neg Vivo \rightarrow [esperar]\neg Vivo), \\ \Box[carregar]Carregado \\ \Box(\neg Vivo \rightarrow [carregar]\neg Vivo), \\ \Box(Vivo \rightarrow [carregar]Vivo), \end{array} \right\};$$

$$BC = \{\neg Carregado, Vivo\}.$$

É possível em $\mathcal{LAP}$ provar:

$$\models_{\mathcal{LAP}} (BC \wedge LEIS) \rightarrow [carregar][esperar][atirar](\neg Carregado \wedge \neg Vivo).$$

Isto é, a pessoa estará morta depois que as ações carregar a arma, esperar e atirar forem executadas.

Axiomática e tableau para $\mathcal{LAP}$

A axiomática de $\mathcal{LAP}$ é formada pelos seguintes esquemas de axiomas e regras de inferências:

PL	todas as tautologias da lógica clássica proposicional,
$K([\alpha])$	$[\alpha](A \to B) \to ([\alpha]A \to [\alpha]B)$,
$K(\Box)$	$\Box(A \to B) \to (\Box A \to \Box B)$
$T(\Box)$	$\Box A \to A$
$4(\Box)$	$\Box A \to \Box\Box A$
$I(\Box, [\alpha])$	$\Box A \to [\alpha]A$
MP	de A e $A \to B$ infere B
$RN(\Box)$	de A infere $\Box A$.
$RN([\alpha])$	de A infere $[\alpha]A$

Um método de tableau para $\mathcal{LAP}$ foi definido em [6] e pode ser colocado nos moldes do que foi apresentado no capítulo 4.

- Regra $\bot$: $A, \neg A, S \Longrightarrow A, \neg A, \bot, S$
- Regra $\neg$: $\neg\neg A, S \Longrightarrow \neg\neg A, A, S$
- Regra $\wedge$: $A \wedge B, S \Longrightarrow A \wedge B, A, B, S$
- Regra $\vee$: $\neg(A \wedge B), S \Longrightarrow \neg(A \wedge B), C, S$ onde C é $\neg A$ ou $\neg B$
- Regra $\Diamond$: $\Diamond A, S \Longrightarrow \Diamond A, S \bullet\!\!-\!\!\bullet A$
- Regra $\langle\alpha\rangle$: $\langle\alpha\rangle A, S \Longrightarrow \langle\alpha\rangle A, S \bullet\!\!\underset{\alpha}{-}\!\!\bullet A$
- Regra $K_\Box$: $\Box A, S \bullet\!\!-\!\!\bullet S1 \Longrightarrow \Box A, S \bullet\!\!-\!\!\bullet A, S1$
- Regra $K_{[\alpha]}$: $[\alpha]A, S \bullet\!\!\underset{\alpha}{-}\!\!\bullet S1 \Longrightarrow [\alpha]A, S \bullet\!\!\underset{\alpha}{-}\!\!\bullet A, S1$
- Regra $T_{[\alpha]}$: $[\alpha]A, S \Longrightarrow [\alpha]A, A, S$
- Regra $4_{[\alpha]}$: $S, [\alpha]A \bullet\!\!\underset{\alpha}{-}\!\!\bullet S1 \Longrightarrow S, [\alpha]A \bullet\!\!\underset{\alpha}{-}\!\!\bullet S1, [\alpha]A$
- Regra $I_{(\alpha,\beta)}$: $S \bullet\!\!\underset{\alpha}{-}\!\!\bullet S1 \Longrightarrow S \bullet\!\!\underset{\alpha,\beta}{-}\!\!\bullet S1$

Infelizmente em $\mathcal{LAP}$ o problema do quadro [35] não está resolvido: há a necessidade de um número muito grande de axiomas de quadro nas representações. Na seção seguinte, tratamos da solução proposta em [6].

6.1.1 $\mathcal{LAP} \rightsquigarrow$

Castilho e seus colegas [6] resolveram o problema do quadro em $\mathcal{LAP}$ introduzindo uma relação de dependência entre ações e literais.

Basicamente, substitui-se os axiomas de quadro $\Box(L \rightarrow [\alpha]L)$ por uma informação de independência entre a ação α e o literal L: $\alpha \not\rightsquigarrow \neg L$. Representando-se a independência através do complementar do conjunto, ou seja, a dependência, chega-se a solução. Para maiores detalhes ver [5].

Axiomática e tableau para $\mathcal{LAP} \rightsquigarrow$

Para adaptar $\mathcal{LAP}$ para comportar esta solução, Castilho e seus colegas construíram a base de informações sobre dependência como uma informação extra lógica e adicionaram à semântica de $\mathcal{LAP}$ as seguintes restrições nos $\mathcal{LAP}$-modelos:

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

Exemplo 6.1.2 (O cenário de Yale representado em $\mathcal{LAP} \rightsquigarrow$)

$$\rightsquigarrow = \left\{ \begin{array}{l} carregar \rightsquigarrow Carregado, \\ atirar \rightsquigarrow \neg Carregado, \\ atirar \rightsquigarrow \neg Vivo \end{array} \right\};$$

$$LEIS = \left\{ \begin{array}{l} \Box\langle esperar\rangle\top, \\ \Box\langle carregar\rangle\top, \\ \Box\langle atirar\rangle\top, \\ \Box[carregar]Carregado, \\ \Box[atirar]\neg Carregado, \\ \Box(Carregado \rightarrow [atirar]\neg Vivo), \\ \Box((\neg Carregado \wedge Vivo) \rightarrow [atirar]Vivo), \end{array} \right\};$$

$$BC = \{\neg Carregado, Vivo\}.$$

É possível provar em $\mathcal{LAP} \rightsquigarrow$

$\models_{\mathcal{LAP}\rightsquigarrow} (BC \wedge LEIS) \rightarrow [carregar][esperar][atirar](\neg Carregado \wedge \neg Vivo)$

A relação de independência $esperar \not\rightarrow \neg Carregado$ garante que a arma continua carregada depois de esperar, o que é um problema nos métodos clássicos.

Para adaptar o método de tableau foi necessário escrever duas regras que captassem a semântica da relação de dependência:

- Regra RP: $L, S \underset{\bullet}{\alpha}_{\bullet} S1 \overset{\alpha \not\rightarrow \neg L}{\Longrightarrow} L, S \underset{\bullet}{\alpha}_{\bullet} L, S1$
- Regra RR: $S \underset{\bullet}{\alpha}_{\bullet} L, S1 \overset{\alpha \not\rightarrow L}{\Longrightarrow} L, S \underset{\bullet}{\alpha}_{\bullet} L, S1$

São duas regras de propagação, levam em conta a semântica da relação de dependência quando um novo nó é gerado. Na primeira, sempre que L está em algum nó do tableau, se $\alpha \not\rightarrow \neg L$ então o literal L é copiado para todos os mundos acessíveis a partir daquele nó através de α. Essa regra é chamada de RP (Regra de Persistência).

Agora , suponha por exemplo que $\alpha \not\rightarrow P$ e $\beta \not\rightarrow \neg P$ por um átomo P e ações α, β. Então, a fórmula $\langle\alpha\rangle P \wedge \langle\beta\rangle \neg P$ é insatisfatível em $\mathcal{LAP} \rightsquigarrow$, mas a regra RP não é suficiente para a prova. É preciso resolver isto usando a regra RR que propaga literais do novo mundo para o mundo que o criou.

Na próxima seção mostraremos como adaptar o GTTP para suportar esta teoria.

6.2 Adaptando GTTP para raciocinar sobre ações

Nesta seção mostramos como estender o GTTP para provar fórmulas em $\mathcal{LAP}$ e $\mathcal{LAP} \rightsquigarrow$. Isto é feito implementando-se métodos que executem as regras de tableau da lógica em questão.

6.2.1 Adaptando GTTP para $\mathcal{LAP}$

O GTTP suporta provas em lógicas multimodais com axiomas de interação. Logo, é capaz de provar fórmulas em $\mathcal{LAP}$. Assim, não há adaptações a fazer.

Basta que o usuário entre com as informações corretas. Depois de informar a fórmula e a estratégia, deve informar os axiomas de interação entre $\Box$ e $[\alpha]$, um para cada operador $[\alpha]$ que aparecer na fórmula. O usuário ainda deve especificar os axiomas de cada operador modal. Não esquecendo que para $\Box$ os axiomas são T, K e 4 e para os operadores $[\alpha]$, apenas K.

Por exemplo, no caso do cenário do tiro, os operadores modais utilizados são $\Box$, $[carregar]$, $[esperar]$ e $[atirar]$, Assim, o usuário deve informar que há interação entre $\Box$ e $[carregar]$, entre $\Box$ e $[esperar]$ e ainda entre $\Box$ e $[atirar]$.

As regras de tableau para $\mathcal{LAP}$ surgem através da combinação das regras de tableau para K_α e das regras para $S4$. Assim temos, em comum às duas lógicas, as regras clássicas e o axioma de iteração. De K_α as regras $\langle\alpha\rangle$, $K[\alpha]$ e de $S4$ a regra $\Diamond$, a regra K, T e 4.

Uma vantagem é que as regras K, $\Diamond$, T e 4 foram implementadas de modo genérico de forma que um mesmo método serve para provas com operadores indexados ou não.

A seguir, um exemplo de entrada de dados no GTTP para provas em $\mathcal{LAP}$ onde o usuário deve informar a fórmula a ser provada, os axiomas para cada operador e a estratégia de aplicação das regras[1]:

```
(~LOADED&[][load]L&[](LOADED->[wait]LOADED)&
[](~LOADED->[wait]~LOADED))->[load]LOADED;
. tk4 load k wait k  . load . wait ; &|~tik4;
```

6.2.2 Adaptando GTTP para $\mathcal{LAP}\rightsquigarrow$

A adaptação do GTTP para $\mathcal{LAP}\rightsquigarrow$ se deu apenas pela implementação das regras RP e RR.

Além da inclusão dos métodos que implementam as regras em si, foi preciso incluir mais uma entrada informada pelo usuário. Agora, além de informar a formula, a estratégia, as regras de tableau para cada operador e as interações, o usuário também deve informar o conjunto que contém as relações de dependência.

[1]No exemplo, os axiomas para os operadores modais são representados informando o índice do operador seguido dos axiomas. Dessa forma está informado que $\Box$ (representado por ".") possui os axiomas $T+K+4$ e os operadors $[wait]$ e $[load]$ o axioma K e os axiomas de interação $I(\Box,load)$ e $I(\Box,wait)$.

O princípio da aplicação das regras RP e RR é o mesmo das demais regras. Primeiro foi implementado um método que verifica se a regra é aplicável naquele momento e depois um método de aplicação da regra. Os novos métodos fazem parte da classe *Tableau*. Outra adaptação necessária foi no método *inicialização(EntraEstrategia)* para inserir os endereços dos novos métodos nos vetores *estrategia* e *aplica_regra* caso sejam solicitadas pelo usuário.

No caso da regra RR, o método que verifica se a regra pode ser aplicada é o método *busca_RR(nodo, formula, mundo)*. Esse método procura um literal no nó atual, varre a lista de mundos acessíveis até encontrar um onde a regra possa ser aplicada. Essa verificação é feita pesquisando se no atributo *relações* e no conjunto das relações de dependência (representado em uma matriz que é uma variável auxiliar) contém um índice que cause a negação do literal encontrado, em caso afirmativo a regra não pode ser aplicada, não encontrando índice que cause a negação do literal, a regra pode ser aplicada, os parâmetros *mundo* e *formula* são atualizados com o endereço do objeto *tableau* onde a regra poderá ser aplicada e com o literal a ser propagado.

A aplicação da regra é feita pelo método *regra_RR(nodo, formula, mundo)* que copia o literal *formula* para o mundo *mundo*.

Para a regra RP, o método que verifica se a regra pode ser aplicada é o método *busca_RP(nodo, formula, mundo)*. Esse método varre a lista de mundos acessíveis até encontrar um onde a regra possa ser aplicada. Essa verificação é feita procurando um literal no conjunto de fórmulas de cada mundo acessível. Encontrando um literal, o método verifica no atributo *relações* e no conjunto das relações de dependência desse mundo se existe um índice que cause o literal encontrado, em caso afirmativo a regra não pode ser aplicada, não encontrando índice que cause o literal, a regra pode ser aplicada, os parâmetros *mundo* e *formula* são atualizados com o endereço do objeto *tableau* onde a regra poderá ser aplicada e com o literal a ser propagado.

A aplicação da regra é feita pelo método *regra_RP(nodo, formula, mundo)* que copia o literal encontrado em um mundo acessível para o nodo atual.

Para executar provas em $\mathcal{LAP} \rightsquigarrow$ o usuário deve informar a fórmula, os axiomas

para cada operador modal, a estratégia e, por último a relação de dependência. No exemplo abaixo vemos como fica a entrada de dados para executar a prova da fórmula $\neg(\langle a\rangle L \wedge \langle b\rangle \neg L)$ na lógica axiomatizada por $T + K + 4$ para $\Box$ e K para $[a]$ e $[b]$ mais os axiomas de interação $I(\Box,[a])$ e $I(\Box,[b])$ com as seguintes relações de dependência: $l \rightsquigarrow L$, $s \rightsquigarrow L$ e $s \rightsquigarrow A$. Ou seja, $\mathcal{LAP}\rightsquigarrow$[2] :

```
~(<a>L&<b>~L) ;  . tk4 a k b k ; . a . b ; &|~t<ik4fs  ; l L s ~L s ~A ;
```

E assim mostramos como o GTTP além de realizar provas em diversas lógicas modais, é facilmente expansível para outras lógicas. Para novas lógicas que sejam formadas pelos axiomas modais cujas regras de tableau correspondentes já estão implementadas, basta passar as informações corretas para o GTTP.

Relembrando, para lógicas que utilizem regras diferentes, basta criar um método que procure a aplicabilidade da regra e outro método que aplique a regra e atualizar o método *inicialização(EntraEstrategia)* para que insira nos vetores *estrategia* e *aplica_regra* os endereços para os novos métodos quando o usuário quiser utilizá-los em uma prova.

[2]Se olharmos só axiomatização a lógica definida no exemplo é $\mathcal{LAP}$, sabemos que a prova será executada em $\mathcal{LAP}\rightsquigarrow$ devido à informação das relações de dependência: $l \rightsquigarrow L$, $s \rightsquigarrow L$ e $s \rightsquigarrow A$ representadas por "l L", "s L" e "s A", respectivamente.

Capítulo 7

Conclusão

Neste trabalho apresentamos a especificação e a implementação de um provador de teoremas para lógicas modais normais baseado em métodos de tableau, que contrariamente aos provadores comumente encontrados, pode ser considerado um provador genérico.

Denominado GTTP, ele foi projetado para receber como dados de entrada a especificação de uma lógica e a partir disto se autoconfigurar para poder provar fórmulas nesta lógica. Esta especificação é feita a partir dos nomes dos axiomas que definem a lógica modal em questão, como é usual em lógicas modais.

Acreditamos que este provador vem preencher uma lacuna no contexto de representação do conhecimento, uma vez que as lógicas modais já eram consideradas um formalismo capaz de representar noções importantes do raciocínio humano, tais como obrigação, crença, saber, percepção, tempo, ações e programas dentre outras.

A implementação de tal provador somente foi possível graças a teoria de que pode-se associar as propriedades semânticas da relação de acessibilidade que define uma lógica modal com uma ou mais regras de tableau. Isto foi bem capturado a partir da teoria de que as regras de tableau podem ser classificadas em dois tipos, as regras de propagação e as regras estruturais.

Infelizmente, isso não ocorre com os tableau tradicionais para as lógicas modais. Geralmente são criadas regras de tableau para lógicas específicas, muitas vezes agrupando em uma única regra a representação de dois ou mais axiomas. São provadores específicos para

lógicas específicas, ou, quando muito, um conjunto de provadores independentes reunidos dentro de uma mesma interface.

Também tomamos o cuidado de permitir ao usuário do nosso sistema a liberdade de definir parte da estratégia de busca. Esta liberdade é restrita à indicação da ordem em que as regras serão aplicadas. O objetivo é permitir experiências de verificação de completude das estratégias. Por exemplo, no sistema de tableau para K, qualquer estratégia que trate a regra de possibilidade antes de aplicar todas as regras de disjunção leva o provador a encontrar uma prova incorreta da fórmula.

Até onde pudemos averiguar, GTTP é o único provador modal implementado que consegue realizar provas em qualquer lógica multimodal (em particular monomodal) que seja definida como qualquer combinação do axioma K (pois estamos no quadro das lógicas normais) com os axiomas T, 4, 5, B, C (confluência) e D (densidade).

GTTP foi possível de ser desenvolvido graças a algumas características interessantes das linguagens orientadas por objetos, notadamente a sobrecarga de funções e a possibilidade de se criar vetores de ponteiros para métodos. A programação orientada por objetos também facilitou o desenvolvimento dos módulos e classes e tornou o código suficientemente modular permitindo a evolução do código para poder tratar também de outras lógicas modais ainda não suportadas.

Como exemplo deste último aspecto, nós mostramos como adaptar o GTTP para provas na Lógica de Ações e Planos utilizada para representar cenários envolvendo raciocínio sobre ações. Assim, o GTTP é hoje o primeiro provador em funcionamento que é capaz de corretamente provar cenários com situações reconhecidas como problemáticas na literatura por envolverem persistência e ramificações.

Uma outra contribuição do nosso trabalho foi a apresentação introdutória simplificada das lógicas modais, com a preocupação didática de facilitar a introdução de não conhecedores de lógicas modais neste interessante assunto.

Como trabalhos futuros acreditamos que muito pode ser feito no que se refere à interface do GTTP. Acreditamos que uma integração da interface proposta pela equipe que

desenvolveu o LOTREC pode casar perfeitamente com este trabalho.

Como o objetivo era desenvolver um provador de teoremas, não nos preocupamos com a explicação de respostas obtidas. Estas podem ser importantes coadjuvantes no ensino de lógica e de métodos de tableau, pois podem servir de base para programação de uma interface gráfica que destaque o conceito de "mundos" existentes nas lógicas modais.

A implementação do GTTP também pode ser integrada ao LabPAT, o projeto desenvolvido pela equipe do Laboratório de Inteligência Computacional que pretende desenvolver um laboratório para provadores automáticos de teoremas. O GTTP pode constituir o módulo de "lógicas modais" ao lado do módulo "lógica clássica de primeira ordem" já disponível para sistemas baseados em resolução [11].

Neste sentido, o GTTP poderia ainda ser programado de maneira a torná-lo mais genérico, pois atualmente ele realiza provas apenas em profundidade na árvore do tableau, e é limitado por obrigar as provas a aplicar todas as regras possíveis em um mesmo mundo antes de passar para outros.

Acreditamos que com estas alterações o GTTP não será apenas um provador de teoremas mas parte de um sistema maior que visa melhorar o conhecimento e o aprendizado sobre lógicas modais aplicadas à representação do conhecimento.

Referências Bibliográficas

[1] E. W. Beth. The foundations of mathematics. 1959.

[2] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic.* Cambridge University Press, Cambridge, UK, 2001.

[3] R. Blanché. *História da Lógica de Aristóteles a Bertrand Russel.* Edições 70, Lisboa, PT, 1985.

[4] M. Castilho, L. Fariñas del Cerro, O. Gasquet, and A. Herzig. Modal tableaux with propagation rules and structural rules. *Fundamenta Informaticae*, 32(3/4), 1997.

[5] M.A. Castilho. *Modèles logiques pour le raisonnement sur les actions.* PhD thesis, Université Paul Sabatier, Toulouse, France, October 1998. In french.

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

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

[8] Newton Afonso Carneiro Costa. *Sistemas Formais Inconsistentes.* Editora da UFPR, Curitiba, 1963.

[9] H.B. Curry. The elimination theorem when modality is present. *Journal of Symbolic Logic*, 17:249–265, 1952.

[10] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of ACM*, 3:201–215, 1960.

[11] L Dinnouti, R Montano, F Silva, and M Castilho. Labpat: um laboratório de provadores automáticos de teoremas. In *In CTIC: Concurso de Trabalhos de Iniciaçãoo Científica*. Sociedade Brasileira de Computação, 1995.

[12] Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, and Fabio Massacci. Lotrec: the generic tableau prover for modal and description logics. In *International Joint Conference on Automated Reasoning*, LNCS, page 6. Springer Verlag, 18-23 juin 2001.

[13] Luis Fariñas del Cerro and Andreas Herzig. Modal deduction with application in epistemic and temporal logics. In Dov M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume Volume 4 - Epistemic and Temporal Reasoning, pages 499–594, Oxford, 1995. Clarendon Press.

[14] Luis Fariñas del Cerro and M. Pentonnen. On the complexity of the satisfiability of modal horn clauses. *Journal of Logic Programming*, 4:72–99, 1987.

[15] L. Fariñas del Cerro, O. Gasquet, A. Herzig, D. Fauthoux, O. Gasquet, D. Longin, T. Polacsek, and S Suwanmanee. Lotrec - a general tableaux theorem prover in java. http://www.irit.fr/ACTIVITES/LILaC/Lotrec.

[16] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel, Dordrecht, 1983.

[17] O. Gasquet. *Déduction en logique multimodale par traduction*. PhD thesis, Université Paul Sabatier, Toulouse, France, 1994. In french.

[18] K. Gödel. Eine interpretation des intuitionistischen aussagenkalkülus. In *Ergebnisse eines mathematischen Kolloquiums 4*, pages 34–40, 1933.

[19] Gerhard Gentzen. Untersuchungen über das Logische Schliessen. *Math. Zeitschrift*, 39:176,–210,405–431, 1935. English translation [19].

[20] Gilmore. A proof method for quantification theory:its justification and realization. *IBM J. Res. Develop*, pages 28–35, 1960.

[21] Rajeev Goré. Tableau methods for modal and temporal logics. Technical report, Automated Reasoning Project, Australian National University, http://arp.anu.edu.au/ rpg/techreports.html, 1997.

[22] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics and the frame problem. *Proc. Nat. (US) Conf. on Artificial Intelligence (AAAI'86)*, pages 328–333, 1986.

[23] J. Herbrand. Recherches sur la théorie de la demonstation. *Travaux de la société des sciences et de lettres de Varsovie*, Classe III Sci:33, 1930.

[24] Andreas Herzig. Reasoning about actions - notas de aulas. j, 2000.

[25] Alain Heuerding. *Sequent Calculi for Proof Search in Some Modal Logics - Theory, Implementation, Applications.* PhD thesis, Universität Bern, 1996.

[26] Alain Heuerding, Michael Seyfried, and Heinrich Zimmermann. Efficient loop-check for backward proof search in some non-classical propositional logics. *Tableaux 96*, 1(1071):210–225, 1996.

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

[28] G. E. Hughes and M. J. Cresswell. *An introduction to modal logic.* Methuen and Co. Ltd., London, 1968.

[29] Gerhard Jaeger. The logics workbench. http://www.lwb.unibe.ch.

[30] Saul Kripke. A completeness theorem on modal logics. *Journal of Symbolic Logic*, 24(1), March 1959.

[31] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc.* O'Reilly & Associates, 1995.

[32] C.I. Lewis. A survey of symbolic logic. *j*, 1918.

[33] C.I. Lewis. Strict implication, an emendation. *Journal of Philosophy*, 17, 1920.

[34] John McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[35] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edimburgh University Press, 1969.

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

[37] Sally Popkorn. *First Steps in Modal Logic*. Cambridge University Press, Cambridge, Inglaterra, 1994.

[38] V. R. Pratt. Considerations on floyd-hoare logic. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1976.

[39] N Rescher. *Many-valued logic*. McGraw-Hill, 1969.

[40] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12:23–41, 1965.

[41] Theobaldo Miranda Santos Santos. *Manual de Filosofia*. Companhia Editora Nacional, 6 edition, 1954.

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

[43] M. E. Szabo, editor. *The collected papers of Gehard Gentzen*. Noth-Holland Pub. Co, Amsterdam, 1969.

[44] Ivan Varzinczak. Provadores automáticos de teoremas para as lógicas monomodais proposicionais k e s4. Technical report, Universidade Federal do Paraná, 2000.

APÊNDICES

```
/* tableau.c
 * Implementação dos métodos definidos tem tableau.h
 * ultimas modificações: 10/02/2003 */

#include "tableau.h"
// Classe Formula
Formula::Formula()
{ }
Formula::~Formula()
{ }
//nega uma fórmula
Formula& Formula::negation()
{
 Formula *neg=new Formula;
 switch(this->get_type())
 {
  case NEGATION:
   neg->set_type(DOUBLE_NEGATION);
   break;
  case NECESSITY:
   neg->set_type(POSSIBILITY);
   break;
  case POSSIBILITY:
   neg->set_type(DOUBLE_NEGATION);
   break;
  case CONJUNCTION:
   neg->set_type(DISJUNCTION);
   break;
  case DISJUNCTION:
   neg->set_type(DOUBLE_NEGATION);
   break;
  case DOUBLE_NEGATION:
   neg->set_type(DOUBLE_NEGATION);
   break;
  case PROP_VARIABLE:
   neg->set_type(NEGATION);
   break;
 }
 neg->set_sentence("~"+this->get_sentence());
 neg->set_left(*this);
 neg->set_right();
 return *neg;
}

//transforma fórmula substituindo
//abreviaturas e nega
Formula& Formula::neg()
{
 Formula *neg=new Formula;
 switch(this->get_type())
 {
  case PROP_VARIABLE:
   neg->set_type(NEGATION);
   neg->set_sentence("~"+this->get_sentence());
   neg->set_left(*this);
   neg->set_right();
   neg->set_action(this->get_action());
   neg->set_rule(0);
   break;
  case PROP_CONSTANT:
   neg->set_type(PROP_CONSTANT);
   if(this->get_sentence()=="true")
  neg->set_sentence("false");
   else
  neg->set_sentence("true");
   neg->set_left();
   neg->set_right();
   neg->set_action(this->get_action());
   neg->set_rule(0);
   break;
  case NEGATION:
   *neg=this->get_left();
   break;
  case CONJUNCTION:
   neg->set_type(NEG_CONJUNCTION);
   if(this->get_sentence().substr(0,1)=="(")
  neg->set_sentence("~"+this->get_sentence());
   else{
  neg->set_sentence('('+this->get_sentence()+')');
  neg->set_sentence("~"+neg->get_sentence());
    }
   neg->set_left(*this);
   neg->set_right();
   neg->set_action(this->get_action());
   neg->set_rule(0);
   break;
  case NECESSITY:
   neg->set_type(NEG_NECESSITY);
   neg->set_sentence("~"+this->get_sentence());
   neg->set_left(*this);
   neg->set_right();
   neg->set_action(this->get_action());
   neg->set_rule(0);
   break;
  case NEG_CONJUNCTION:
   *neg=this->get_left();
   break;
  case NEG_NECESSITY:
   *neg=this->get_left();
   break;
  case DOUBLE_NEGATION:
   *neg=this->get_left();
   break;
 }
 return *neg;
}

Formula& Formula::adjust()
{
 Formula* adj=new Formula;
 Formula* tmp=new Formula;
//auxiliares
 Formula* tmp1=new Formula;
 Formula* tmp2=new Formula;
 Formula* tmp3=new Formula;
 Formula* tmp4=new Formula;

 switch(this->get_type())
 {
  case PROP_VARIABLE:
```

```
  adj=this;
  break;
case PROP_CONSTANT:
  adj=this;
  break;
case NEGATION:
  adj=this;
  break;
case CONJUNCTION:
  adj->set_type(CONJUNCTION);
  adj->set_left(get_left().adjust());
  adj->set_right(get_right().adjust());
  adj->set_action(this->get_action());
  if(this->get_sentence().substr(0,1)=="(")
  adj->set_sentence("("+adj->get_left().get_sentence()+
"&"+adj->get_right().get_sentence()+")");
else
  adj->set_sentence(adj->get_left().get_sentence()+
    "&"+adj->get_right().get_sentence());
  break;
case DISJUNCTION:
  adj->set_type(NEG_CONJUNCTION);
  tmp->set_left(get_left().adjust().neg());
  tmp->set_right(get_right().adjust().neg());
  tmp->set_sentence(tmp->get_left().get_sentence()+
    '&'+tmp->get_right().get_sentence());
  tmp->set_type(CONJUNCTION);
  adj->set_left(*tmp);
  adj->set_right();
  adj->set_sentence('('+adj->get_left().get_sentence()+')');
  adj->set_sentence("~"+adj->get_sentence());
  adj->set_action(this->get_action());
  break;
case IMPLICATION:
  adj->set_type(NEG_CONJUNCTION);
  tmp->set_left(get_left().adjust());
  tmp->set_right(get_right().adjust().neg());
  tmp->set_sentence(tmp->get_left().get_sentence()+
    '&'+tmp->get_right().get_sentence());
  tmp->set_type(CONJUNCTION);
  adj->set_left(*tmp);
  adj->set_right();
  adj->set_sentence('('+adj->get_left().get_sentence()+')');
  adj->set_sentence("~"+adj->get_sentence());
  adj->set_action(this->get_action());
  break;
case BICONDITIONAL:
  adj->set_type(CONJUNCTION);
  // ramo a esquerda
  tmp2->set_type(CONJUNCTION);
  tmp2->set_left(get_left().adjust());
  tmp2->set_right(get_right().adjust().neg());
  tmp2->set_sentence('('+tmp2->get_left().get_sentence()+
    '&'+tmp2->get_right().get_sentence()+')');
  tmp1->set_type(NEG_CONJUNCTION);
  tmp1->set_left(*tmp2);
  tmp1->set_right();
  tmp1->set_sentence("~"+tmp2->get_sentence());
  // ramo a direita
  tmp4->set_type(CONJUNCTION);
  tmp4->set_left(get_right().adjust());
  tmp4->set_right(get_left().adjust().neg());
  tmp4->set_sentence('('+tmp4->get_left().get_sentence()+
    '&'+tmp4->get_right().get_sentence()+')');
  tmp3->set_type(NEG_CONJUNCTION);
  tmp3->set_left(*tmp4);
  tmp3->set_right();
  tmp3->set_sentence("~"+tmp4->get_sentence());
  // raiz
  adj->set_left(*tmp1);
  adj->set_right(*tmp3);
  adj->set_sentence(tmp1->get_sentence()+
    '&'+tmp3->get_sentence());
  adj->set_action(this->get_action());
  break;
case NECESSITY:
  adj->set_type(NECESSITY);
  adj->set_left(get_left().adjust());
  adj->set_sentence("["+this->get_action()+"]"+
    adj->get_left().get_sentence());
  adj->set_right();
  adj->set_action(this->get_action());
  adj->set_ok_if(false);
  break;
case POSSIBILITY:
  adj->set_type(NEG_NECESSITY);
  tmp->set_type(NECESSITY);
  tmp->set_left(get_left().adjust().neg());
  tmp->set_right();
  tmp->set_sentence("["+this->get_action()+"]"+
    "("+tmp->get_left().get_sentence()+")");
  adj->set_left(*tmp);
  adj->set_sentence('~'+adj->get_left().get_sentence());
  adj->set_right();
  adj->set_action(this->get_action());
  adj->set_ok_if(false);
  break;
case DOUBLE_NEGATION:
  *adj=(this->get_left().get_left()).adjust();
  break;
case NEG_CONJUNCTION:
  adj=this;
  break;
case NEG_DISJUNCTION:
  *adj=get_left().adjust().neg();
  break;
case NEG_IMPLICATION:
  *adj=get_left().adjust().neg();
  break;
case NEG_BICONDITIONAL:
  *adj=get_left().adjust().neg();
  break;
case NEG_NECESSITY:
  adj->set_type(NEG_NECESSITY);
  adj->set_left(get_left().adjust());
  adj->set_sentence("~"+adj->get_left().get_sentence());
  adj->set_action(this->get_action());
  break;
case NEG_POSSIBILITY:
  *adj=get_left().adjust().neg();
```

```
 }
 return *adj;
}
//varre o vetor multimodal para ver se a regra pode ser aplicada a tal indice
bool Formula::is_ok(char R)
{
 string acao = action;
 bool achou = false;
 int i = 0;
 while(Multimodal[i][0]!="tchau" && Multimodal[i][0] != acao)
  {if ((acao == "")&& (Multimodal[i][0]==".")) break;
  i++;
  }
 if ((Multimodal[i][0] == acao) || (Multimodal[i][0]=="." && acao==""))
  {
   int pos = 0;
   string regras = Multimodal[i][ i ];
   regras = regras + '\0';
   while(regras[pos]!='\0'){
 if (regras[pos]== R){
  achou = true;
  break;
 }
 pos++;
   }
  }
 else achou = false;
 if (achou) return true;
 else  return false;
}
//verifica se uma fórmula está dentro de uma lista de fórmulas
bool Formula::is_in(list<Formula>& set)
{
 list<Formula>::iterator it_set=set.begin();
 while(it_set!=set.end())
  {
   if(get_sentence()==(*it_set).get_sentence())
 return true;
   ++it_set;
  }
 return false;
}

bool Formula::is_in_action(list<Formula>& set)
{
 list<Formula>::iterator it_set=set.begin();
 while(it_set!=set.end())
  {
   if(get_action()==(*it_set).get_action())
 return true;
   ++it_set;
  }
 return false;
}
//classe Node
Node::Node()
{
 left = NULL;
 right = NULL;
}
```

```
Node::~Node()
{ }

list<Formula>& Node::complementary(Formula formula)
{
 list<Formula> *complement=new list<Formula>();
 list<Formula>::iterator it_set=(this->get_formula_set()).begin();
 list<Formula>::iterator it_comp=(*complement).begin();

 while(it_set!=(this->get_formula_set()).end())
  {
   if((*it_set).get_sentence()!=formula.get_sentence())
 (*complement).insert((*complement).end(), *it_set);
   ++it_set;
   ++it_comp;
  }
 return *complement;
}
//retira os operadores [ ] e <> de X
list<Formula>& Node::unbox_X()
{
 list<Formula> *unbox_set=new list<Formula>();
 list<Formula>::iterator it_set=(this->get_formula_set()).begin();
 list<Formula>::iterator it_box=(*unbox_set).begin();
 while(it_set!=(this->get_formula_set()).end())
 {
  if(*it_set).get_type()==NECESSITY)
   (*unbox_set).insert((*unbox_set).end(), (*it_set).get_left());
  ++it_set;
  ++it_box;
 }
 return *unbox_set;
}

list<Formula>& Node::undiamond_X()
{
 list<Formula> *undiamond_set=new list<Formula>();
 list<Formula>::iterator it_set=(this->get_formula_set()).begin();
 list<Formula>::iterator it_diamond=(*undiamond_set).begin();
 while(it_set!=(this->get_formula_set()).end())
  {
   if((*it_set).get_type()==POSSIBILITY)
 (*undiamond_set).insert((*undiamond_set).end(),(*it_set).get_left());
   ++it_set;
   ++it_diamond;
  }
 return *undiamond_set;
}
//Verifica se o nó tem inconsistencia
bool Node::inconsistent()
{
 Formula* form=new Formula;
 form->set_sentence("false");
 if(form->is_in(get_formula_set()))
  return true;
 return false;
}
//impressao na tela
void Node::print()
```

```
{
 list<Formula>::iterator it=(this->get_formula_set()).begin();
 while(it!=(this->get_formula_set()).end())
   {
    cout << (*it).get_sentence() << "  ;  " ;
    ++it;
   }
 cout<<"\n";
}
void Node::print_history()
{
 list<Formula>::iterator it=(this->get_history()).begin();
 cout << "history: \n";
 while(it!=(this->get_history()).end())
   {
    cout << (*it).get_sentence() << "  ;  ";
    ++it;
   }
}
void Node::print_historyD()
{
 list<Formula>::iterator it=(this->get_historyD()).begin();
 cout << "history Diamond: \n";
 while(it!=(this->get_historyD()).end())
   {
    cout << (*it).get_sentence() << "  ;  ";
    ++it;
   }
}
void Node::print_historyT()
{
 list<Formula>::iterator it=(this->get_historyT()).begin();
 cout << "history Regra T: \n";
 while(it!=(this->get_historyT()).end())
   {
    cout << (*it).get_sentence() << "  ;  ";
    ++it;
   }
}
//classe tableau
Tableau::Tableau()
{
 root=new Node();
 accessible_worlds.clear();
}

Tableau::Tableau(Formula& formula):root()
{
 get_root().add_formula(formula);
 accessible_worlds.clear();
}

Tableau::~Tableau()
{ }

bool Tableau::is_ok_it(Formula& formula)
{
 //verifica se há axioma de iteracao entre o indice da formula e o vetor de relacoes do mundo
 bool achou = false;
 vector<string>::iterator it_action = this->relation.begin();
 int i = 0;
 while ((it_action)!=this->relation.end()){
  while(interaction[i][0]!="endi")
    {
 if (interaction[i][0]==formula.get_action()  &&
       interaction[i][1] == *it_action )
  return true;
 if (formula.get_action()=="" &&
       interaction[i][0]=="." && interaction[i][1] == *it_action)
  return true;
 i++;
    }
  ++it_action;
 }
 return false;
}

bool Tableau::is_in_relation(Formula& formula)
{
 /*verifica se a acao da formula está no vetor de relacao do mundo*/
 vector<string>::iterator it_action= this->relation.begin();
 while(it_action!=this->relation.end())
 {
  if(formula.get_action()==(*it_action))
    return true;
  if (formula.get_action()=="" && (*it_action)==".")
    return true;
  ++it_action;
 }
 return false;
}
//impressao na tela
void Tableau::print_relation()
{
 vector<string>::iterator it_action= this->relation.begin();
 while(it_action!=this->relation.end())
   {
    cout << *it_action << "  " ;
    ++it_action;
   }
}
//varre o vetor multimodal para ver se a regra pode ser aplicada a tal indice
bool Tableau::search_dep(string literal, string acao)
{
 int i, j=0;
 for (i=0;i<10; i++)
  if (dep[i][j]=="endd")
    break;
  else
    if ((dep[i][j]==acao) && (dep[i][j+1]==literal))
 return true;
 return false;
}

void Tableau::initialization(string* EntraEstrategia)
{
 int posicao = 0;
 fechado = false;
 for (posicao=0;posicao<15;posicao++)
   {
```

```
  switch((*EntraEstrategia)[posicao])
{
case '&':
 (this->Estrategia[posicao]) = &Tableau::Search_and;
 (this->AplicaRegra[posicao]) = &Tableau::and_rule;
 break;

case '|':
 (this->Estrategia[posicao]) = &Tableau::Search_or;
 (this->AplicaRegra[posicao]) = &Tableau::or_rule;
 break;

case '~':
 (this->Estrategia[posicao]) = &Tableau::Search_neg;
 (this->AplicaRegra[posicao]) = &Tableau::neg_rule;
 break;

case 't':
 (this->Estrategia[posicao]) = &Tableau::Search_necT;
 (this->AplicaRegra[posicao]) = &Tableau::t_rule;
 break;

case 'k':
 (this->Estrategia[posicao]) = &Tableau::Search_k;
 (this->AplicaRegra[posicao]) = &Tableau::k_rule;
 break;

case '4':
 (this->Estrategia[posicao]) = &Tableau::Search_4;
 (this->AplicaRegra[posicao]) = &Tableau::R4_rule;
 break;

case '<':
 (this->Estrategia[posicao]) = &Tableau::Search_pos;
 (this->AplicaRegra[posicao]) = &Tableau::diamond_rule;
 break;
case 'b':
 (this->Estrategia[posicao]) = &Tableau::Search_b;
 (this->AplicaRegra[posicao]) = &Tableau::b_rule;
 break;
case '5':
 (this->Estrategia[posicao]) = &Tableau::Search_5;
 (this->AplicaRegra[posicao]) = &Tableau::5_rule;
 break;
case 'D':
 (this->Estrategia[posicao]) = &Tableau::Search_D;
 (this->AplicaRegra[posicao]) = &Tableau::D_rule;
 break;
case 'e':
 (this->Estrategia[posicao]) = &Tableau::Search_e;
 (this->AplicaRegra[posicao]) = &Tableau::e_rule;
 break;

case 'i':
 (this->Estrategia[posicao]) = &Tableau::Search_i;
 (this->AplicaRegra[posicao]) = &Tableau::i_rule;
 ok_i2 = true;
 break;

case 'f':
 (this->Estrategia[posicao]) = &Tableau::Search_sf;
 (this->AplicaRegra[posicao]) = &Tableau::sf_rule;
 break;

case 's':
 (this->Estrategia[posicao]) = &Tableau::Search_sb;
 (this->AplicaRegra[posicao]) = &Tableau::sb_rule;
 break;

default:
 (this->Estrategia[posicao]) = NULL;
 break;
}
 }
this->expand();
}

void Tableau::expand()
{
 int Rules =0;
 expand(this->get_root(), Rules);
}
//esse método que controla a prova
void Tableau::expand(Node& node, int& Rules)
{
 Tableau* world = NULL;
 bool PodeAplicar=0;
 bool temRegra=1;
 //se o nó contem inconsistencia "false" sai
 if (node.inconsistent()){
  fechado = true;
  closed=1;
  //remove todos os elementos da lista ligada de formulas do nodo
  node.get_formula_set().clear();
  //retorna da recursão
  return;
 }
 //senão verifica se tem uma fórmula que é a negação da outra
 else
  {
   if (inconsistency(node))
 expand(node, Rules);
   //senão tenta aplicar regras cfe estratégia do usuário
   else
 {
  Formula* formula = new Formula;
  Formula* form1 = new Formula;
  Formula* form2 = new Formula;
  int auxRules = 0;
  char r;

  //varre o vetor estratégia tentando aplicar regras
  while( ((this->*Estrategia[Rules])!=NULL) && (closed != 0))
   {
    if (( (this->*Estrategia[Rules])(node, *formula,  r, &world)))
        {
         PodeAplicar=1;
         break;
        }
    ++Rules;
```

```
 }
//se encontrou uma regra que pode ser aplicada
if(PodeAplicar)
 {
  if ((this->*AplicaRegra[Rules])(node, *formula, r, &world))
       {
        expand((*world).get_root(), Rules);
       }
  else
       //se existe nó à esquerda
       if (&(node.get_left())!=NULL)
        {
         Rules =0;
         int Tam =0;
         if (&(this->get_access_worlds()) != NULL )
          Tam =  this->get_access_worlds().size();
         //expansão à esquerda
         expand(node.get_left(), Rules);
         //se encontrou inconsistencia a esquerda e tem nó a direita
         //caminhar à direita
         if ((closed==1)&& (&(node.get_right())!=NULL))
          {
               int Tam_esquerda = this->get_access_worlds().size();
               //exclui os mundos possiveis gerados qdo da expansão à esquerda
               if (&(this->get_access_worlds()) != NULL ){
                if(Tam==0)
                  this->get_access_worlds().clear();
                else{
                  for (int i=0;i<Tam_esquerda - Tam;i++)
                    this->accessible_worlds.pop_back();
                 }
               }
               Rules=0;
               //expansão à direita
               expand(node.get_right(), Rules);
          }
        }
       else
        {//se não existe nó à esquerda, continua tentando aplicar as regras neste nó
         Rules=0;
         expand(node,Rules);
        }
}
else//se não encontrou nenhuma regra a ser aplicada
 {
  closed = -2;
  //varrer mundos acessiveis
  if ( &(get_access_worlds().back())!=NULL )
       {
        list<Tableau>::iterator it_world;
        for (it_world = this->accessible_worlds.begin();
           it_world!= this->accessible_worlds.end(); ++it_world)
         {
          (*it_world).initialization(&EntraEstrategia);
          //encontrando um mundo possivel fechado, sai da recursão
          if (closed == 1) return;
         }//end for
        return;
       }//end if tem outros mundos
  }//end else não pode aplicar regras
  }
   //else if procura inconsistencia
  }
}//fim do expand

void Tableau::insert_world(Tableau& new_world)
{
 accessible_worlds.insert(accessible_worlds.end(), new_world);
}
//procura se pode aplicar regra E
bool Tableau::Search_and(Node& node,Formula& formula,char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 //enquanto nao fim da lista de fórmulas
 while ( it_formula != (node.get_formula_set()).end() )
  {
   if (((*it_formula).get_type() == CONJUNCTION) &&
      ((*it_formula).get_rule()!=1))
  {
   formula = *it_formula;
   world = NULL;
   return true;
  }
    ++it_formula;
  }// end of while
 return false;
}
//aplica regra e
bool Tableau::and_rule(Node& node,Formula& formula,char& R,Tableau** world)
{
 formula.set_rule(1);
 Node *left=new Node();
 //aqui o node passa a apontar para o no left
 node.set_left(*left);
 node.set_right();

 //INICIALIZA O NOVO NÓ
 left->set_formula_set(node.complementary(formula));

 //se tenho A&B, meu novo nó tem A, B
 left->add_formula(formula.get_left());
 left->add_formula(formula.get_right());
 left->set_history(node.get_history());
 left->set_left();
 left->set_right();
 world = NULL;
 R = '&';
 return false;
}
//procura uma fórmula e sua negação e troca pro false
bool Tableau::inconsistency(Node& node)
{
 int found = 0;
 list<Formula>::iterator it_formula1=(node.get_formula_set()).begin();
 list<Formula>::iterator it_formula2=(node.get_formula_set()).begin();
 while( it_formula1 != (node.get_formula_set()).end() )
  {
   it_formula2 = it_formula1;
   ++it_formula2;
   if( (*it_formula1).get_sentence().substr(0,1)=="~")
```

```
{
 string subformula1=(*it_formula1).get_sentence().
        substr(1,(*it_formula1).get_sentence().size());
 while(it_formula2!=(node.get_formula_set()).end())
  {
   if((*it_formula2).get_sentence()==subformula1)
        {
         found=1;
         break;
        }
   ++it_formula2;
  }
 if(found==1)
  break;
 ++it_formula1;
}
  else
{
 string formula1=(*it_formula1).get_sentence();
 while(it_formula2!=(node.get_formula_set()).end())
  {
   if((*it_formula2).get_sentence()=="~"+formula1)
        {
         found=1;
         break;
        }
   ++it_formula2;
  }// end of while
 if(found==1)
  break;
 ++it_formula1;
}
 }//do primeiro while
if(found==1)
 {
  node.set_formula_set(node.complementary(*it_formula1));
  node.set_formula_set(node.complementary(*it_formula2));
  Formula *falsity=new Formula;
  falsity->set_sentence("false");
  falsity->set_type(PROP_CONSTANT);
  falsity->set_left();
  falsity->set_right();
  node.add_formula(*falsity);
  closed=1;
  fechado = true;
  closed=1;
  return true;
 }
else
  return false;
}
//procura se pode aplicar regra ou
bool Tableau::Search_or(Node& node,Formula& formula,char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 while ( it_formula != (node.get_formula_set()).end() )
  {
   if((*it_formula).get_type()==NEG_CONJUNCTION)&&
     ((*it_formula).get_rule()!=1))
      {
       formula = *it_formula;
       (*it_formula).set_rule(1);
       return true;
      }
    ++it_formula;
   }
 return false;
}
//aplica regra ou
bool Tableau::or_rule(Node& node,Formula& formula,char& R,Tableau** world)
{
 formula.set_rule(1);
 Node *left=new Node();
 Node *right=new Node();
 node.set_left(*left);
 node.set_right(*right);
 //inicializa nó a esquerda
 left->set_formula_set(node.complementary(formula));
 left->add_formula( ( (formula.get_left()).get_left()).neg() );
 left->set_history(node.get_history());
 //inicializa nó a direita
 right->set_formula_set(node.complementary(formula));
 right->add_formula((formula.get_left().get_right()).neg());
 right->set_history(node.get_history());
 R = '|';
 return false;
}
//procura se pode aplicar regra negação
bool Tableau::Search_neg( Node& node,Formula& formula,char& R ,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 while ( it_formula != (node.get_formula_set()).end() )
  {
   if(( (*it_formula).get_type()==DOUBLE_NEGATION )&&
     ((*it_formula).get_rule()!=1))
  {
   formula = *it_formula;
   (*it_formula).set_rule(1);
   return true;
  }
    ++it_formula;
  }
 return false;
}
//aplica regra negação
bool Tableau::neg_rule(Node& node,Formula& formula,char& R,Tableau** world)
{
 formula.set_rule(1);
 Node *left=new Node();
 node.set_left(*left);
 node.set_right();
 left->set_formula_set(node.complementary(formula));
 left->add_formula((formula.get_left()).get_left());
 left->set_history(node.get_history());
 return false;
}
//procura se pode aplicar regra K
bool Tableau::Search_k(Node& node,Formula& formula,char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
```

```
list<Tableau>::iterator it_world=get_access_worlds().begin();
R='k';
//Se há outros mundos varrer a lista de mundos possiveis
while(it_world != (get_access_worlds()).end())
{
 while(it_formula!=(node.get_formula_set()).end())
   {
    if (((*it_formula).is_ok(R)))
 if ( ((*it_formula).get_type()==NECESSITY) &&
     (!(*it_formula).get_left().is_in( (*it_world).
          get_root().get_formula_set())) &&
     ((*it_world).is_in_relation(*it_formula)))
   {
    formula = *it_formula;
    *world = &(*it_world);
    R= 'k';
    return true;
   }
    ++it_formula;
   }
 it_formula = node.get_formula_set().begin();
 ++it_world;
}
 return false;
}
//aplica regra K
bool Tableau::k_rule(Node& node,Formula& formula,char& R,Tableau** world)
{
 formula.get_left().set_rule(false);
 (**world).get_root().add_formula(formula.get_left());
 (**world).get_root().set_history(node.get_history());
 R= 'k';
 if (inconsistency((**world).get_root())){
  return true;
 }
 else
  return false;
}
//procura se pode aplicar regra <>
bool Tableau::Search_pos(Node& node,Formula& formula,char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 R = '<';
 while(it_formula!=(node.get_formula_set()).end())
   {
    if( (((*it_formula).get_type() == POSSIBILITY)||
    ((*it_formula).get_type() == NEG_NECESSITY) ) &&
    (((*it_formula).get_rule() != 1)&&
    (!(*it_formula).is_in(node.get_historyD()))) &&
    ((*it_formula).is_ok(R)))
   {
    formula = *it_formula;
    (*it_formula).set_rule(1);
    return true;
   }
     ++it_formula;
   }
 return false;
}
//aplica regra <>
```

```
bool Tableau::diamond_rule(Node& node,Formula& formula,char& R,Tableau** world)
{
 Tableau *new_world=new Tableau;
 new_world->set_relation(formula.get_action());
 if ( (formula).get_type()== NEG_NECESSITY)
  new_world->get_root().add_formula((formula.get_left()).get_left().neg());
 else
  if ((formula).get_type()== POSSIBILITY)
   new_world->get_root().add_formula(formula.get_left());
 node.add_historyD(formula);
 new_world->get_root().set_history(node.get_history());
 this->insert_world(*new_world);
 formula.set_rule(1);
 return false;
}
//procura se pode aplicar regra T
bool Tableau::Search_necT(Node& node, Formula& formula, char& R, Tableau** world)
{
 R='t';
 if (&(node.get_left()) == NULL)
  {
   list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
   if (&(node.get_left()) == NULL)
  {
   while(it_formula!=(node.get_formula_set()).end())
    {
     if (((*it_formula).get_type()==NECESSITY) &&
          ((*it_formula).get_rule()!=1) &&
          ((*it_formula).is_ok(R)) &&
          (!((*it_formula).get_left().is_in(node.get_formula_set())))
           )
          {
           formula = *it_formula;
           (*it_formula).set_rule(1);
           return true;
          }
     ++it_formula;
    }
  }
  }
 return false;
}
//aplica a regra T
bool Tableau::t_rule(Node& node,Formula& modality,char& R,Tableau** world)
{
 Node *left=new Node();
 node.set_left(*left);
 node.set_right();
 left->set_formula_set(node.get_formula_set());
 left->add_formula(modality.get_left());
 left->set_historyT(node.get_historyT());
 left->add_historyT(modality);
 R = 't';
 return false;
}
//procura se pode aplicar regra 4
bool Tableau::Search_4(Node& node,Formula& formula,char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 list<Tableau>::iterator it_world=get_access_worlds().begin();
```

50 Fórmula: ~(<>~A&~(~[]A&B))
Operador: [ ] Axiomas: tk
Estratégia: &|~t<k
gttp: Não esperado: Não

51 Fórmula: ([](A&B&C)&[](B&C))->([]B&[]([]C&A))
Operador: [ ] Axiomas: tk
Estratégia: &|~t<k
gttp: Não esperado: Não

52 Fórmula: A->B|A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

53 Fórmula: <>(A<->A)
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

54 Fórmula: []A->A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

55 Fórmula: ([]A&~<>B)->A&~B
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

56 Fórmula: [](([]A|<>(~A|B))
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

57 Fórmula: [](A&B)<->([]A&[]B)
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

58 Fórmula: <>A|<>B-><>(A|B)
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

59 Fórmula: ~<>(<>A|B)->[][]~A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

60 Fórmula: <>[]<>[]A-><>[]A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

61 Fórmula: <>[]A-><>[]<>[]A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Sim esperado: Sim

62 Fórmula: A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

63 Fórmula: <>(A&B)|<><>~A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

64 Fórmula: A&[]A|[][]~A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

65 Fórmula: ~[]A->[]~[]A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

66 Fórmula: <>A&<>B-><>(A&B)
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

67 Fórmula: []([]A|[]B)->[]A|[]B
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

68 Fórmula: <>A&[]B-><>(A&C)|[]A
Operador: [ ] Axiomas: tk4
Estratégia: &|~t<k4
gttp: Não esperado: Não

69 Fórmula: <>[]~A
Operador: [ ] Axiomas: tk4
Estratégia: &|~<kt4
gttp: Não esperado: Não

70 Fórmula: <>[]A|<>[]~A
Operador: [ ] Axiomas: tk4
Estratégia: &|~<kt4
gttp: Não esperado: Não

71 Fórmula: []A->A
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Sim esperado: Sim

72 Fórmula: (<>A->[]<>A)
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Sim esperado: Sim

73 Fórmula: []A-><>A
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Sim esperado: Sim

74 Fórmula: []true

Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Sim esperado: Sim

---

75 Fórmula: <>true
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Sim esperado: Sim

---

76 Fórmula: <>A->A
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Não esperado: Não

---

77 Fórmula: <>A->[]A
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Não esperado: Não

---

78 Fórmula: []<>A->A
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Não esperado: Não

---

79 Fórmula: (<>A&<>B)-><>(A&B)
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Não esperado: Não

---

80 Fórmula: ([]A->[]B)->[](A->B)
Operador: [ ] Axiomas: tk5
Estratégia: &|~t<k5
gttp: Não esperado: Não

---

81 Fórmula: [a]A&B->B&[a]A&[a]A
Operador: [a] Axiomas: k
Estratégia: &|~<k
gttp: Sim esperado: Sim

---

82 Fórmula: [b]A&[b](A->B)->[b]B
Operador: [b] Axiomas: k
Estratégia: &|~<k
gttp: Sim esperado: Sim

---

83 Fórmula: (~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->[b]A
Operador: [ ] Axiomas: tk4
Operador: [a] Axiomas: k
Operador: [b] Axiomas: k
Axiomas de interação: [ ]->[a]
[ ]->[b]
Estratégia: &|~t<ik4
gttp: Sim esperado: Sim

---

84 Fórmula: (~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->[a]~A
Operador: [ ] Axiomas: tk4
Operador: [a] Axiomas: k
Operador: [b] Axiomas: k
Axiomas de interação: [ ]->[a]
[ ]->[b]
Estratégia: &|~t<ik4
gttp: Sim esperado: Sim

---

85 Fórmula:
([sempre][raposa][elogia]ENCANTADO&[raposa][sempre](ENCANTADO-
><canta>CAIDO))->[raposa][elogia]<canta>CAIDO
Operador: [sempre] Axiomas: tk4
Operador: [elogia] Axiomas: k
Operador: [canta] Axiomas: k
Operador: [raposa] Axiomas: k
Axiomas de interação: [sempre]->[elogia]
[sempre]->[canta]
[sempre]->[raposa]
Estratégia: &|~t<ik4
gttp: Sim esperado: Sim

---

86 Fórmula: [a]A&B->~(B&[a]A&[a]A)
Operador: [a] Axiomas: k
Estratégia: &|~<k
gttp: Não esperado: Não

---

87 Fórmula: [b]A&[b](A->B)->~[b]B
Operador: [b] Axiomas: k
Estratégia: &|~<k
gttp: Não esperado: Não

---

88 Fórmula: (~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->~[b]A
Operador: [ ] Axiomas: tk4
Operador: [a] Axiomas: k
Operador: [b] Axiomas: k
Axiomas de interação: [ ]->[a]
[ ]->[b]
Estratégia: &|~t<ik4
gttp: Não esperado: Não

---

89 Fórmula: (~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->~[a]~A
Operador: [ ] Axiomas: tk4
Operador: [a] Axiomas: k
Operador: [b] Axiomas: k
Axiomas de interação: [ ]->[a]
[ ]->[b]
Estratégia: &|~t<ik4
gttp: Não esperado: Não

---

90 Fórmula:
([sempre][raposa][elogia]ENCANTADO&[raposa][sempre](ENCANTADO-
><canta>CAIDO))->~[raposa][elogia]<canta>CAIDO
Operador: [sempre] Axiomas: tk4
Operador: [elogia] Axiomas: k
Operador: [canta] Axiomas: k
Operador: [raposa] Axiomas: k
Axiomas de interação: [sempre]->[elogia]
[sempre]->[canta]
[sempre]->[raposa]
Estratégia: &|~t<ik4
gttp: Não esperado: Não

```
R= '4';
while (it_world != get_access_worlds().end())
 {
  it_formula = (node.get_formula_set()).begin();
  while(it_formula!=(node.get_formula_set()).end())
 {
  if( ((*it_formula).get_type()==NECESSITY) &&
  ( !((*it_formula).is_in(((*it_world).get_root()).get_formula_set() )))
          &&(!(*it_formula).is_in(node.get_history())))
    if (((*it_formula).is_ok(R)) )
     if ((*it_world).is_in_relation(*it_formula))
           {
            formula = *it_formula;
            *world = &(*it_world);
            return true;
           }
  ++it_formula;
 }
   ++it_world;
  }
 return false;
}
//aplica regra 4
bool Tableau::R4_rule(Node& node,Formula& modality,char& R,Tableau** world)
{
 (**world).get_root().add_formula(modality);
 node.add_history(modality);
 (**world).get_root().set_history(node.get_history());
 (modality).set_rule(1);
 if (inconsistency((**world).get_root()))
   return true;
 return false;
}
//verifica se aplica regra 1(a,b)
bool Tableau::Search_i(Node& node, Formula& formula, char& R,Tableau** world)
{
 list<Formula>::iterator it_formula=(node.get_formula_set()).begin();
 list<Tableau>::iterator it_world=get_access_worlds().begin();
 if (!(get_access_worlds().empty()))
  {
   while (it_world != (get_access_worlds()).end())
 {
  it_formula = (node.get_formula_set()).begin();
  while(it_formula!=(node.get_formula_set()).end())
   {
    if (((*it_formula).get_type()==NECESSITY) ||
        ((*it_formula).get_type()==NEG_POSSIBILITY))
         if    ((*it_world).is_ok_i(*it_formula))
           if (!((*it_world).is_in_relation(*it_formula))){
             formula = *it_formula;
             *world = &(*it_world);
             return true;
            }
     ++it_formula;
    }
  ++it_world;
 }
  }
 else return false;
}
```

```
//aplica regra 1(a,b)
bool Tableau::i_rule(Node& node, Formula& formula, char& R, Tableau** world)
{
 (**world).set_relation(formula.get_action());
 return false;
}
//procura se pode aplicar SF
bool Tableau::Search_sf(Node& node, Formula& formula, char& R, Tableau** world)
{
 list<Formula>::iterator it_formula1=(node.get_formula_set()).begin();
 list<Tableau>::iterator it_world= get_access_worlds().begin();
 string literal, not_literal, buff1, buff2;
 bool achei = false;
 while(it_formula1!=(node.get_formula_set()).end())
  {
   if (   ((*it_formula1).get_type()==PROP_VARIABLE)||
     ( ((*it_formula1).get_type()== NEGATION) &&
        ((*it_formula1).get_left().get_type()==PROP_VARIABLE))
 {
  literal = (*it_formula1).get_sentence();
  not_literal = ((*it_formula1).neg()).get_sentence();
  it_world = get_access_worlds().begin();
  while(it_world != (get_access_worlds()).end())
   {
    achei = false;
    vector<string>::iterator aux=((*it_world).relation.begin());
    while (aux != (*it_world).relation.end())
          {
           if (search_dep(not_literal, *aux)){
            achei = true;
            break;
           }
           aux++;
          }
    if ((!achei))
          if ( ( !((*it_formula1).is_in((*it_world).
            get_root().get_formula_set()))))
           {
            formula = (*it_formula1);
            *world = &(*it_world);
            return true;
           }
     ++it_world;
    }
  }
   ++it_formula1;
  }
 return false;
}
//aplica SF
bool Tableau::sf_rule(Node& node, Formula& formula, char& R, Tableau** world)
{
 (**world).get_root().add_formula(formula);
 if (inconsistency((**world).get_root()))
  return true;
 else return false;
}
//procura se pode aplicar SB
bool Tableau::Search_sb(Node& node, Formula& formula, char& R, Tableau** world)
{
```

```
if (!(get_access_worlds().empty()) )
 {
  list<Tableau>::iterator it_world= get_access_worlds().begin();
  list<Formula>::iterator it_formula1=
       (*it_world).get_root().get_formula_set().begin();
  string literal;
  bool achei;
  while(it_world != (get_access_worlds()).end())
{
 it_formula1=(*it_world).get_root().get_formula_set().begin();
 while(it_formula1!= (*it_world).get_root().get_formula_set().end())
  {
   if ( ((*it_formula1).get_type()==PROP_VARIABLE)||
          (((*it_formula1).get_type()== NEGATION) &&
         (*it_formula1).get_left().get_type()==PROP_VARIABLE))
         {
          literal = (*it_formula1).get_sentence();
          vector<string>::iterator aux = (*it_world).relation.begin();
          while(aux != (*it_world).relation.end())
           {
            achei = false;
            if (search_dep(literal, *aux)){
                 achei = true;
                 break;
            }
            aux++;
           }
          if (!achei)
           if ( !(*it_formula1).is_in( node.get_formula_set()))
            {
                 formula = (*it_formula1);
                 *world = &(*it_world);
                 return true;
            }
         }//fim se
    ++it_formula1;
   }//end while formula
  ++it_world;
 }//end while world
  }
 return false;
}
//aplica SB
bool Tableau::sb_rule(Node& node, Formula& formula, char& R, Tableau** world)
{
 node.add_formula(formula);
 return false;
}
//procura se pode aplicar regra B ou 5up
bool Tableau::Search_b(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world=get_access_worlds().begin();
 list<Formula>::iterator it_formula;
 while(  next_world != get_access_worlds().end())
  {
   it_formula=(((*next_world).get_root()).get_formula_set()).begin();
   while( it_formula != ( ((*next_world).get_root()).get_formula_set()).end())
 {
  if ( ((*it_formula).get_type()==NECESSITY) &&
      ((*it_formula).get_rule()!=1) )
      {
       formula = *it_formula;
       (*it_formula).set_rule(1);
       R='b';
       return true;
      }
   ++it_formula;
 }//end while
   ++next_world;
  }
 return false;
}
//aplica B
bool Tableau::b_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 node.add_formula(formula.get_left());
 node.print();
}
//aplica regra 5up
bool Tableau::u5_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 node.add_formula(formula);
 node.print();
}
//procura se pode aplicar 5->
bool Tableau::Search_5r(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world1=get_access_worlds().begin();
 list<Formula>::iterator it_formula;
 while(  next_world1 != get_access_worlds().end())
  {
   it_formula=(((*next_world1).get_root()).get_formula_set()).begin();
   while( it_formula != ( ((*next_world1).get_root()).get_formula_set()).end())
 {
  if ( ((*it_formula).get_type()==NECESSITY) &&
      ((*it_formula).get_rule()!=1) )
   {
    formula = *it_formula;
    (*it_formula).set_rule(1);
    R='5';
    return true;
   }
  ++it_formula;
 }
   ++next_world1;
  }
 return false;
}
//aplica regra 5->
bool Tableau::r5_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world1=get_access_worlds().begin();
 list<Formula>::iterator it_formula;
 bool achei = false;
 while(  next_world1 != get_access_worlds().end())
  {
   it_formula=(((*next_world1).get_root()).get_formula_set()).begin();
   while( it_formula != ( ((*next_world1).get_root()).get_formula_set()).end())
 {
  if ( ((*it_formula).get_type()==NECESSITY) &&
```

```
      ((*it_formula).get_rule()!=1) )
    {
     formula = *it_formula;
     (*it_formula).set_rule(1);
     achei = true;
     break;
    }
   ++it_formula;
  }
    if (achei)
  break;
    ++next_world1;
   }
  list<Tableau>::iterator next_world2=get_access_worlds().begin();
  while(next_world2 != get_access_worlds().end())
   {
    ((*next_world2).get_root()).add_formula(*it_formula);
    ++next_world2;
   }
  return false;
}
//procura se pode aplicar D
bool Tableau::Search_D(Node& node, Formula& formula, char& R,  Tableau** world)
{
 return true;
}
//aplica D
bool Tableau::D_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 Tableau *new_world=new Tableau;
 insert_world(*new_world);
 return false;
}
//procura se pode aplicar C0
bool Tableau::Search_C0(Node& node, Formula& formula, char& R,  Tableau** world)
{
 if (this->get_access_worlds().size()>=2)
   return true;
 else return false;
}
//aplica C0
bool Tableau::C0_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world1 = get_access_worlds().begin();
 list<Tableau>::iterator next_world2 = get_access_worlds().begin();
 next_world2++;
 Tableau *new_world=new Tableau;
 (*next_world1).insert_world(*new_world);
 (*next_world2).insert_world(*new_world);
}
//procura C1
bool Tableau::Search_C1(Node& node, Formula& formula, char& R,  Tableau** world)
{
 if (this->get_access_worlds().size()==1)
   return true;
 else return false;
}
//aplica C1
bool Tableau::C1_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world = get_access_worlds().begin();
 Tableau *new_world=new Tableau;
 (*new_world).insert_world(*new_world);
}
//procura De
bool Tableau::Search_e(Node& node, Formula& formula, char& R,  Tableau** world)
{
 if ((this->get_access_worlds().size())==1 )
   return true;
 else return false;
}
//aplica De
bool Tableau::e_rule(Node& node, Formula& formula, char& R,  Tableau** world)
{
 list<Tableau>::iterator next_world = get_access_worlds().begin();
 Tableau *new_world=new Tableau;
 insert_world(*new_world);
 (*next_world).insert_world(*new_world);
}
```

```
/* tableau.h
 * Definições das classes
 * Últimas modificações: 10/02/2003
 */

#define PROP_VARIABLE 0
#define PROP_CONSTANT 1
#define NEGATION 2
#define CONJUNCTION 3
#define DISJUNCTION 4
#define IMPLICATION 5
#define BICONDITIONAL 6
#define NECESSITY 7
#define POSSIBILITY 8
#define DOUBLE_NEGATION 9
#define NEG_CONJUNCTION 10
#define NEG_DISJUNCTION 11
#define NEG_IMPLICATION 12
#define NEG_BICONDITIONAL 13
#define NEG_NECESSITY 14
#define NEG_POSSIBILITY 15

#include<fstream>
#include<sstream>
#include<list>
#include<vector>
#include<string>

extern int closed;
extern string sistema;
extern string EntraEstrategia;
extern string Multimodal[10][10];
extern bool ok_i2;
extern string interaction[10][2];
extern string dep[10][2];

//classe definida para modelar uma fórmula
class Formula{
 private:
// especifica o tipo principal de uma fórmula (conj, disj, etc.)
  int type;
//informa se a regra foi aplicada ou não
  bool rule;
  bool ok_i;
//especifica a ação do operador modal
  string action;
//subfórmulas à esquerda e à direita
  Formula *left;
  Formula *right;

 public:

  string sentence;

  Formula();
  ~Formula();

  int get_type(){ return type; }
  void set_type(int type){ this->type=type; }
  void set_rule(bool rule){this->rule=rule;}
  bool get_rule(){return rule;}
  void set_ok_i(bool ok_i){this->ok_i=ok_i;}

  //se houver, recebe a acao
  void set_action(string action){this->action=action;}
  void set_action(){this->action="";}
  void print_action(){cout << this->action ;}
  string get_action(){return action; }

  bool is_ok(char R);
  string get_sentence(){ return sentence; }
  void set_sentence(string sentence){ this->sentence=sentence; }

  Formula& get_left(){ return *left; }
  Formula& get_right(){ return *right; }

  void set_left(Formula& left){ this->left=&left; }
  void set_left(){ this->left=NULL; }
  void set_right(Formula& right){ this->right=&right; }
  void set_right(){ this->right=NULL; }

  Formula& negation();
  Formula& neg();
  Formula& adjust();
  bool is_in(list<Formula>& set);

  bool is_in_action(list<Formula>& set);
};
// classe que define um nó de um tableau
class Node{
 private:
  list<Formula> formula_set;
  list<Formula> history;
  list<Formula> historyD;
  list<Formula> historyT;

  Node *left;
  Node *right;
 public:
  Node();
  ~Node();

  Node& get_left(){ return *left; }
  Node& get_right(){ return *right; }

//seta os ponteiros para o nó a esquerda e a direita ou para nulo qdo nao tem
  void set_left(Node& left){ this->left=&left; }
  void set_left(){ this->left=NULL; }
  void set_right(Node& right){ this->right=&right; }
  void set_right(){ this->right=NULL; }

  list<Formula>& get_formula_set(){ return formula_set; }
  list<Formula>& get_history(){ return history; }
  list<Formula>& get_historyD(){return historyD;}
  list<Formula>& get_historyT(){return historyT;}

  void set_formula_set(list<Formula>& formula_set)
      { this->formula_set=formula_set; }
  void set_history(list<Formula>& history){ this->history=history; }
  void set_history(){ this->history.empty(); }
```

```
  void set_historyD(list<Formula>& historyD){this->historyD=historyD;}
  void set_historyT(list<Formula>& historyT){this->historyT=historyT;}

// Retira a fórmula da lista de fórmulas
  list<Formula>& complementary(Formula formula);
// Retira [] de []X
  list<Formula>& unbox_X();
  list<Formula>& undiamond_X();

  bool inconsistent();

  void add_formula(Formula& formula){ formula_set.insert(formula_set.end(), formula); }
  void add_history(Formula& formula){ history.insert(history.end(), formula); }
  void add_historyD(Formula& formula){historyD.insert(historyD.end(), formula);}
  void add_historyT(Formula& formula){historyT.insert(historyT.end(), formula);}

  void print();
  void print_history();
  void print_historyD();
  void print_historyT();
};
//Classe que define um tableau - um mundo possível de Kripke
class Tableau{
 private:
  Node *root;
//o conjunto de mundos acessíveis a partir deste
  list<Tableau> accessible_worlds;
  vector<string> relation;
  bool fechado;

  typedef bool (Tableau::*ApontaRegra)(Node& node, Formula& formula, char& R, Tableau** world);
 public:

//Estrategia é um vetor de ponteiros para funções membro da classe Tableau
  ApontaRegra Estrategia[15], AplicaRegra[15];

  int tamanho;
  Tableau* ptrRules;
  Tableau();
  Tableau(Formula& formula);
  ~Tableau();

  void set_relation(string relation){this->relation.push_back(relation);}

  vector<string> get_relation(){return relation;}

  bool is_ok_i(Formula& formula);
  bool is_in_relation(Formula& formula);

  Node& get_father(){return *father;}
  void set_father(Node& father){this->father=&father;}
  void set_father(){this->father=NULL;}
  Node& get_root(){ return *root; }
  void set_root(Node& root){ this->root=&root; }

  Node& get_end_right(){return (*root).get_right();}
  Node& get_end_left(){return (*root).get_left();}


  void print_relation();
```

```
  void initialization(string *EntraEstrategia);
  bool caminha_arvore(Node& node, Formula& formula);
  bool caminha_arvore_sb(Node& node, Formula& formula);
  bool search_dep(string literal, string acao);

  void expand();
  void expand(Node& node, int& Rules);

  list<Tableau>& get_access_worlds(){ return accessible_worlds; }
  void insert_world(Tableau& new_world);
//métodos que verificam a possibilidade de aplicação de uma regra
  bool Search_and(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_or(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_neg(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_pos(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_necT(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_k(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_4(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_b(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_5r(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_D(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_C0(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_C1(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_e(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_i(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_sf(Node& node, Formula& formula, char& R, Tableau** world);
  bool Search_sb(Node& node, Formula& formula, char& R, Tableau** world);

//métodos que aplicam as regras
  bool and_rule(Node& node, Formula& conjunction, char& R, Tableau** world);
  bool inconsistency(Node& node);
  //REGRAS CLASSICAS E <>
  bool or_rule(Node& node, Formula& disjunction, char& R, Tableau** world);
  bool neg_rule(Node& node, Formula& double_negation, char& R, Tableau** world);
  bool diamond_rule(Node& node, Formula& diamond, char& R, Tableau** world);
//REGRAS DE PROPAGAÇÃO
  bool k_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool t_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool b_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool u5_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool r5_rule(Node& node, Formula& modality, char& R, Tableau** world);
//REGRAS ESTRUTURAIS
  bool D_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool C0_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool C1_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool e_rule(Node& node, Formula& modality, char& R, Tableau** world);
 //outras regras
 //axioma de iteração
  bool i_rule(Node& node, Formula& modality, char& R, Tableau** world);
 //regra SF e SB
  bool sf_rule(Node& node, Formula& modality, char& R, Tableau** world);
  bool sb_rule(Node& node, Formula& modality, char& R, Tableau** world);

};
```

```
{]A&B->B&{]A&{]A                    ; . k ; ; &|~<k ; Sim
[](([]A|<>(~A|B))                   ; . k ; ; &|~<k ; Sim
[]A&[]((A->B)->[]B                  ; . k ; ; &|~<k ; Sim
[]A|[]B->[](A|B)                    ; . k ; ; &|~<k ; Sim
[](A&B)<->([]A&[]B)                 ; . k ; ; &|~<k ; Sim
[](A&B)<->([]A&[]B)                 ; . k ; ; &|~<k ; Sim
[]~A&~<>B|<>A|<>B                   ; . k ; ; &|~<k ; Sim
<>A|<>B-><>(A|B)                    ; . k ; ; &|~<k ; Sim
[]~A|<>((([]B-><>A)->[]B)->[]B)     ; . k ; ; &|~<k ; Sim
~[](<>A|B)-><>[]~A                  ; . k ; ; &|~<k ; Sim
~<>~A->[]A                          ; . k ; ; &|~<k ; Sim
<>A->~[]~A                          ; . k ; ; &|~<k ; Sim
~[]~A-><>A                          ; . k ; ; &|~<k ; Sim
[](A&B)->([]A&[]B)                  ; . k ; ; &|~<k ; Sim
[]A&[]B->[](A&B)                    ; . k ; ; &|~<k ; Sim
(<>A|<>B)-><>(A|B)                  ; . k ; ; &|~<k ; Sim
<>(A|B)->(<>A|<>B)                  ; . k ; ; &|~<k ; Sim

[]A                          ; . k ; ; &|~<k ; Não
<>true                       ; . k ; ; &|~<k ; Não
<>[]A|<>[]~A                 ; . k ; ; &|~<k ; Não
[]A->A                       ; . k ; ; &|~<k ; Não
~[]A->[]~[]A                 ; . k ; ; &|~<k ; Não
[]~A->[]~<>A                 ; . k ; ; &|~<k ; Não
<>A&<>B->(A&B)               ; . k ; ; &|~<k ; Não
(<>A&<>B)-><>(A&B)           ; . k ; ; &|~<k ; Não
<>[]<>[]A->[]A               ; . k ; ; &|~<k ; Não
<>[]A-><>[]<>A               ; . k ; ; &|~<k ; Não
A&[]A|[]]~A                  ; . k ; ; &|~<k ; Não
<>~A->[]A                    ; . k ; ; &|~<k ; Não
[](<>A|B)-><>[]~A            ; . k ; ; &|~<k ; Não

A->B|A                       ; . tk ; ; &|~t<k ; Sim
<>(A<->A)                    ; . tk ; ; &|~t<k ; Sim
[]A->A                       ; . tk ; ; &|~t<k ; Sim
([]A&~<>B)->A&~B             ; . tk ; ; &|~t<k ; Sim
[]A&B->B&[]A&[]A             ; . tk ; ; &|~t<k ; Sim
[](([]A|<>(~A|B))            ; . tk ; ; &|~t<k ; Sim
{](A&B)<->([]A&[]B)          ; . tk ; ; &|~t<k ; Sim
<>A|<>B-><>(A|B)             ; . tk ; ; &|~t<k ; Sim
<>((([]A->~B)->[]A)->[]A)    ; . tk ; ; &|~t<k ; Sim
~[](<>A|B)->~<>~<>~A         ; . tk ; ; &|~t<k ; Sim

A                            ; . tk ; ; &|~t<k ; Não
<>[]~A                       ; . tk ; ; &|~t<k ; Não
A&[]A|[]]~A                  ; . tk ; ; &|~t<k ; Não
<>[]A|<>[]~A                 ; . tk ; ; &|~t<k ; Não
~[]A->[]~[]A                 ; . tk ; ; &|~t<k ; Não
[]~A->[]~<>A                 ; . tk ; ; &|~t<k ; Não
<>A&<>B->~(A&B)              ; . tk ; ; &|~t<k ; Não
<>[]<>[]A->~[]A              ; . tk ; ; &|~t<k ; Não
<>[]A-><>[]<>[]A             ; . tk ; ; &|~t<k ; Não
~(<>~A&~(~[]A&B))            ; . tk ; ; &|~t<k ; Não
([](A&B&C)&[](B&C))->([]B&[](]C&A)) ; . tk ; ; &|~t<k ; Não

A->B|A                       ; . tk4 ; ; &|~t<k4 ; Sim
<>(A<->A)                    ; . tk4 ; ; &|~t<k4 ; Sim
[]A->A                       ; . tk4 ; ; &|~t<k4 ; Sim
([]A&~<>B)->A&~B             ; . tk4 ; ; &|~t<k4 ; Sim
[](([]A|<>(~A|B))            ; . tk4 ; ; &|~t<k4 ; Sim
[](A&B)<->([]A&[]B)          ; . tk4 ; ; &|~t<k4 ; Sim
<>A|<>B-><>(A|B)             ; . tk4 ; ; &|~t<k4 ; Sim
~<>(<>A|B)->[]([]~A          ; . tk4 ; ; &|~t<k4 ; Sim
<>[]<>[]A->~<>[]A            ; . tk4 ; ; &|~t<k4 ; Sim
<>[]A->~<>[]~[]A             ; . tk4 ; ; &|~t<k4 ; Sim

A                            ; . tk4 ; ; &|~t<k4 ; Não
<>(A&B)|<><>~A               ; . tk4 ; ; &|~t<k4 ; Não
A&[]A|[]]~A                  ; . tk4 ; ; &|~t<k4 ; Não
~[]A->[]~[]A                 ; . tk4 ; ; &|~t<k4 ; Não
<>A&<>B->~(A&B)              ; . tk4 ; ; &|~t<k4 ; Não
[](A|B)->[]A|[]B             ; . tk4 ; ; &|~t<k4 ; Não
<>A&[]B->~(A&C)|[]A          ; . tk4 ; ; &|~t<k4 ; Não
<>[]~A                       ; . tk4 ; ; &|~t<k4 ; Não
<>[]A|<>[]~A                 ; . tk4 ; ; &|~t<k4 ; Não


[]A->A                       ; . tk5 ; ; &|~t<k5 ; Sim
(<>A->[]<>A)                 ; . tk5 ; ; &|~t<k5 ; Sim
[]A->~<>A                    ; . tk5 ; ; &|~t<k5 ; Sim
[]true                       ; . tk5 ; ; &|~t<k5 ; Sim
<>true                       ; . tk5 ; ; &|~t<k5 ; Sim

<>A->A                       ; . tk5 ; ; &|~t<k5 ; Não
<>A->[]A                     ; . tk5 ; ; &|~t<k5 ; Não
[]<>A->A                     ; . tk5 ; ; &|~t<k5 ; Não
(<>A&<>B)->~<>(A&B)          ; . tk5 ; ; &|~t<k5 ; Não
([]A->[]B)->[](A->B)         ; . tk5 ; ; &|~t<k5 ; Não


[a]A&B->B&[a]A&[a]A ;
a k ; ; &|~<k ; Sim

[b]A&[b](A->B)->[b]B ;
b k ; ; &|~<k ; Sim

(~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->[b]A ;
. a . b ; . tk4 a k b k ; &|~t<ik4 ; Sim

(~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->[a]~A;
. a . b ; . tk4 a k b k ; &|~t<ik4 ; Sim

([sempre][raposa][elogia]ENCANTADO&
[raposa][sempre](ENCANTADO-><canta>CAIDO))->
[raposa][elogia]<canta>CAIDO ;
sempre elogia sempre canta sempre raposa ;
sempre tk4 elogia k canta k raposa k ; &|~t<ik4 ; Sim


[a]A&B->~(B&[a]A&[a]A) ;
a k ; ; &|~<k ; Não

[b]A&[b](A->B)->~[b]B ;
b k ; ; &|~<k ; Não

(~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->~[b]A ;
. a . b ; . tk4 a k b k ; &|~t<ik4 ; Não

(~A&[][b]A&[](A->[a]A)&[](~A->[a]~A))->~[a]~A ;
. a . b ; . tk4 a k b k ; &|~t<ik4 ; Não

([sempre][raposa][elogia]ENCANTADO&
[raposa][sempre](ENCANTADO-><canta>CAIDO))->
~[raposa][elogia]<canta>CAIDO ;
sempre elogia sempre canta sempre raposa ;
sempre tk4 elogia k canta k raposa k ; &|~t<ik4 ; Não

quit
```

```
1   Fórmula: []A&B->B&[]A&[]A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

2   Fórmula: [](([]A|<>(~A|B))
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

3   Fórmula: []A&[](A->B)->[]B
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

4   Fórmula: []A|[]B->[](A|B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

5   Fórmula: [](A&B)<->([]A&[]B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

6   Fórmula: [](A&B)<->([]A&[]B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

7   Fórmula: []~A&~<>B|<>A|<>B
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

8   Fórmula: <>A|<>B-><>(A|B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

9   Fórmula: []~A|<>((([]B-><>A)->[]B)->[]B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

10   Fórmula: ~[](<>A|B)-><>[]~A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

11   Fórmula: ~<>~A->[]A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

12   Fórmula: <>A->~[]~A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

13   Fórmula: ~[]~A-><>A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

14   Fórmula: [](A&B)->([]A&[]B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

15   Fórmula: []A&[]B->[](A&B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

16   Fórmula: (<>A|<>B)-><>(A|B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

17   Fórmula: <>(A|B)->(<>A|<>B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Sim  esperado: Sim

18   Fórmula: []A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

19   Fórmula: <>true
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

20   Fórmula: <>[]A|<>[]~A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

21   Fórmula: []A->A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

22   Fórmula: ~[]A->[]~[]A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

23   Fórmula: []~A->[]~<>A
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

24   Fórmula: <>A&<>B-><>(A&B)
Operador: [ ]  Axiomas:  k
Estratégia: &|~<k
gttp:  Não  esperado: Não

25   Fórmula: (<>A&<>B)-><>(A&B)
Operador: [ ]  Axiomas:  k
```

Estratégia: &|-<k
gttp: Não esperado: Não

---

26 Fórmula: <>[]<>[]A->[]<>[]A
Operador: [ ] Axiomas: k
Estratégia: &|-<k
gttp: Não esperado: Não

---

27 Fórmula: <>[]A-><>[]<>A
Operador: [ ] Axiomas: k
Estratégia: &|-<k
gttp: Não esperado: Não

---

28 Fórmula: A&[]A|[][]~A
Operador: [ ] Axiomas: k
Estratégia: &|-<k
gttp: Não esperado: Não

---

29 Fórmula: <>~A->[]A
Operador: [ ] Axiomas: k
Estratégia: &|-<k
gttp: Não esperado: Não

---

30 Fórmula: [](<>A|B)-><>[]~A
Operador: [ ] Axiomas: k
Estratégia: &|-<k
gttp: Não esperado: Não

---

31 Fórmula: A->B|A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

32 Fórmula: <>(A<->A)
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

33 Fórmula: []A->A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

34 Fórmula: ([]A&~<>B)->A&~B
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

35 Fórmula: []A&B->B&[]A&[]A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

36 Fórmula: [](([]A|<>(~A|B))
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

37 Fórmula: [](A&B)<->([]A&[]B)
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

38 Fórmula: <>A|<>B-><>(A|B)
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

39 Fórmula: <>((([]A-><>B)->[]A)->[]A)
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

40 Fórmula: ~[](<>A|B)-><><><>~A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Sim esperado: Sim

---

41 Fórmula: A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

42 Fórmula: <>[]~A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

43 Fórmula: A&[]A|[][]~A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

44 Fórmula: <>[]A|<>[]~A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

45 Fórmula: ~[]A->[]~[]A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

46 Fórmula: []~A->[]~<>A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

47 Fórmula: <>A&<>B-><>(A&B)
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

48 Fórmula: <>[]<>[]A-><>[]A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não

---

49 Fórmula: <>[]A-><>[]<>[]A
Operador: [ ] Axiomas: tk
Estratégia: &|-t<k
gttp: Não esperado: Não