AUTOMAÇÃO DE LÓGICA MODAL: UM ESTUDO COMPARATIVO DE ALGUMAS ABORDAGENS

LUIZ FERNANDO PEREIRA DE SOUZA

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Sheila Regina Murgel Veloso

Prof.ª Sheila Regina Murgel Veloso, D.Sc.
(Presidente)

Doris Ferraz de Aragon

Prof.ª Doris Ferraz de Aragon, D.Sc.

Gerson Zaverucha

Prof. Gerson Zaverucha, Ph.D.

Mario R. F. Benevides

Prof. Mario Roberto Folhadela Benevides, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 1992

SOUZA, LUIZ FERNANDO PEREIRA DE

Automação de Lógica Modal: Um Estudo Comparativo de Algumas Abordagens
[Rio de Janeiro], 1992
x, 188 p., 29,7cm. (COPPE/UFRJ, M.Sc.,
Engenharia de Sistemas e Computação, 1992)
Tese - Universidade Federal do Rio de Janeiro
1. Automação de métodos de prova em lógica modal
2. Lógicas não-clássicas 3. Provadores de teoremas
I. COPPE/UFRJ II. Título (série).

"Se você já conseguiu tudo o que planejou, você não planejou o suficiente."
— Martial

Agradecimentos

Agradecemos ao Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, pelos recursos materiais, sem os quais seria impossível para nós realizar este trabalho.

Agradecemos aos membros da banca, que nos honraram com sua participação. E em particular a nossa orientadora, pela dedicação e paciência durante a realização deste trabalho.

Aos colegas Alexandre, Eliana e João pelo apoio mútuo.

Aos colegas Anibal, Claudia e Jonas pelos incentivos.

A Suzan, pela adaptação do latex para o formato tese.

E finalmente, a todos aqueles que, direta ou indiretamente, contribuíram (ou pelo menos não atrapalharam) a realização deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

Automação de Lógica Modal: Um Estudo Comparativo de Algumas Abordagens

Luiz Fernando Pereira de Souza
maio de 1992

Orientadora: Sheila Regina Murgel Veloso
Programa: Engenharia de Sistemas e Computação

Sabemos que a implementação automática de métodos de prova em lógica clássica é um problema bem resolvido, e que há um crescente interesse no uso de lógica modal. Neste contexto, estamos interessados em avaliar métodos de prova automática de lógica modal. Para isto, tomamos quatro diferentes propostas, que são analisadas e comparadas.

Inicialmente definimos uma série de requisitos que um método voltado para implementação automática, idealmente, deveria apresentar. Em seguida, para cada método estudado, é feita uma descrição detalhada de seu funcionamento, e é apresentada uma proposta de implementação. Com isto, conseguimos ter uma boa avaliação dos problemas práticos de realização de cada implementação.

Concluímos o trabalho com uma comparação entre os quatro métodos escolhidos, segundo aqueles requisitos definidos anteriormente.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Modal Logic Automation: A Comparative Study of Some Approaches

Luiz Fernando Pereira de Souza
May, 1992

Thesis Supervisor: Sheila Regina Murgel Veloso
Department : Systems Engineering and Computing

It is known that automatic implementation of classic logic proof methods is a well-solved issue. Nowadays, there is an increasing interest about modal logic applications. In this context, we are interested in the evaluiatiuon of automatic implementations of modal logic proof methods. In order to do this evaluation, four different approaches are analysed and compared.

First, some requirements that an approach intended to be automatically implemented should present are defined. Then, for each approach, its application is presented in detail and it is given a proposal of implementation. In this way, the practical issues related to each implementation are well analysed.

In the last chapter, the four approaches are compared according to the requirements formerly defined.

Índice

Capítulo I

Introdução.

Neste capítulo apresentamos as motivações que levaram ao desenvolvimento deste trabalho e os nossos objetivos. Descrevemos, também, como o trabalho foi desenvolvido e como o texto está organizado.

I.1 Motivações.

Quando desenvolvemos algum trabalho de pesquisa, geralmente, estamos interessados em responder alguma questão, solucionar algum problema, ou estamos simplesmente motivados pela curiosidade de determinar até onde podemos avançar em um determinado campo do conhecimento. No caso deste trabalho, as três alternativas são verdadeiras.

A questão que estamos procurando responder é que, dado que a automação da lógica clássica de primeira ordem é um problema razoavelmente bem resolvido, com soluções que apresentam bom desempenho [Robinson 65] ou oferecem recursos bastante genéricos para modelagem de problemas [Elcook 90,Murray 82], o que podemos esperar em relação à automação de lógica modal de primeira ordem ?

O problema que estamos buscando solucionar é que, dado que lógica modal tem sido cada vez mais usada na modelagem e especificação de sistemas, e dado que é consenso que a mesma linguagem usada durante a especificação de um sistema deve preferivelmente ser usada na validação e se possível na implementação do sistema, o que podemos dispor atualmente a respeito de solução automática de problemas especificados através de lógica modal de primeira ordem ?

Finalmente, estamos particularmente interessados em expandir o conhecimento e tecnologia a respeito de automação de lógica modal de primeira ordem. Tomamos como base os trabalhos realizados até agora que pudemos analisar, determinando as vantagens e qualidades de cada proposta existente e evidenciando seus possíveis

problemas. Estamos procurando novas soluções, mais adequadas às necessidades dos usuários e estamos buscando resolver questões ainda em aberto.

Dentro desta linha de pesquisa, procuramos delinear que tipo de contribuição poderia ser feita no escopo de uma tese de mestrado. Desta forma, definimos que nosso trabalho seria um estudo comparativo de algumas abordagens para automação de lógica modal de primeira ordem. Para cada uma das propostas analisadas, procuramos determinar a classe de problemas que podem ser tratados com o método proposto, suas principais características, seus mais graves problemas e a efetivação e eficiência de sua implementação automática. Para isto, além do estudo e aplicação do método para resolver problemas específicos, também desenvolvemos para algumas das abordagens um protótipo, de forma a avaliar melhor os problemas relacionados a complexidade e praticidade da implementação.

I.2 Objetivos.

O objetivo deste trabalho, então, passa a ser o estudo, a descrição e a comparação de diferentes propostas de métodos de prova automática em lógica modal. Esperamos que estas propostas não sejam apenas uma metodologia para se especificar formalmente e obter soluções para sistemas e problemas de lógica modal, mas que se aproximem o máximo possível de uma linguagem de programação, de forma a unificar dentro de uma mesma teoria lógica a especificação e a solução dos problemas tratados [Quintana 88c].

Podemos subdividir as propriedades que esperamos que estes métodos apresentem em dois tipos: requisitos internos e requisitos externos. No primeiro caso, estaremos preocupados com questões de efetivação e eficiência. No segundo caso, estaremos atentos para os recursos que o método implementado ofereceria ao usuário para a modelagem de seus sistemas e problemas.

I.2.1 Requisitos Internos.

Em uma perspectiva teórica, a completude é uma qualidade essencial. Entretanto, em termos práticos, normalmente, teremos que determinar um meio termo entre completude e eficiência. Ou seja, teremos métodos completos, mas de implementação impossível, e métodos implementáveis, mas com problemas de completude. Note que este problema sempre existirá em qualquer método, visto que a própria lógica modal tem uma complexidade inerente, e qualquer sistema que apresente a completude terá esta mesma complexidade. Ao analisarmos cada proposta é importante determinar

para que classes de problemas a proposta apresenta a completude da solução. É importante observar que, ao longo deste texto, nos métodos baseados em refutação estaremos chamando de completude a completude refutacional. Ou seja, a qualidade do método apresentar a completude, tendo em vista que ele será usado em uma refutação.

Já uma característica que não é essencial em uma abordagem teórica, mas é de vital importância em sistemas automatizados, é a política de busca de soluções. Ou seja, estamos interessamos em como se apresenta o espaço de soluções de cada abordagem, e qual é a forma em que ele é percorrido.

Estaremos chamando de efetivação de uma abordagem à qualidade do método poder ser implementado com os recursos computacionais disponíveis atualmente. Ou seja, mesmo reconhecendo a importância de estudos estritamente teóricos, está fora do escopo deste trabalho estudar propostar que assumem, por exemplo, número infinito de processadores.

Por último, temos que considerar a eficiência dos métodos de prova. Esta qualidade não estará refletindo para nós um índice quantitativo, tal como quantas horas de processamento foram necessárias para fornecer determinada solução. Estaremos apenas interessados em uma avaliação comparativa entre as propostas analisadas, de outra forma, este trabalho se tornaria muito extenso.

I.2.2 Requisitos Externos.

Acreditamos que a qualidade mais importante que uma abordagem oferece ao usuário é a expressividade da linguagem usada para especificação do problema. É justamente na definição da linguagem que serão colocadas as restrições quanto à generalidade da aplicação do método. Ou seja, se determinada condição, digamos presença de um quantificador no escopo de um operador modal, não é aceita pela linguagem, o método logicamente também não aceitará tal condição. Vale lembrar que estaremos considerando que se um método não é completo para determinada condição, então a restrição de não usar esta condição deve estar presente na definição da linguagem de interface com o usuário.

É importante também que o método seja adaptável ao problema sendo tratado. Por exemplo, devem ser providos meios de se selecionar o sistema modal que estamos trabalhando (K, T, K4 etc). Também é interessante que o método seja facilmente modificável, de forma a fazer uso de características específicas de determinados modelos, como por exemplo existência de alguns predicados que possuem sempre a mesma avaliação em todos os mundos.

I.3 Abordagens.

Dentro do cronograma máximo permitido para a execução deste trabalho, pudemos analisar quatro abordagens diferentes. A abordagem 1, proposta por Martín Abadi e Zohar Manna em *"Modal Theorem Proving"* [Abadi 86], é um método bastante genérico para tratamento de lógica modal de primeira ordem, sendo proposto como completo e voltado especificamente para implementação. Como veremos, entretanto, este método apresenta alguns problemas tão grandes quanto as qualidades apregoadas.

A abordagem 2, foi escolhida por ser antiteticamente muito específica quando comparada à primeira abordagem e paradoxalmente proposta pelos mesmos autores, em *"Temporal Logic Programming"* [Abadi 89]. Ela visa a automação de lógica temporal linear e sua implementação é extremamente direta e eficiente.

Para a abordagem 3, escolhemos a proposta feita por Kurt Konilige em *"Resolution and Quantified Epistemic Logics"*, por ser um dos mais importantes trabalhos publicados nesta área nos últimos anos, tendo servido de base para vários outros trabalhos posteriores.

Para última abordagem, escolhemos a proposta feita por Pablo Javier de la Quintana em sua dissertação de doutorado, *"Automated Reasoning for Modal Logics: A Natural Deduction Based Approach"* [Quintana 88a]. Esta abordagem foi escolhida por ser baseada em dedução natural, sendo um método bastante diferente dos anteriormente analisados.

I.4 Organização do Texto.

Este texto está organizado da seguinte forma. No Apêndice A estabelecemos precisamente a notação de lógica clássica que será usada ao longo do trabalho, definindo o significado pretendido de cada símbolo e formalizando os conceitos que serão usados ao longo do texto. No capítulo 2 definimos lógica modal e apresentamos a notação que será usada. Nos capítulos 3, 4, 5 e 6 apresentamos respectivamente as abordagens 1, 2, 3 e 4. No capítulo 7 concluímos nossa dissertação, resumindo as principais conclusões apresentadas no decorrer do texto e comparando as quatro abordagens. Incluímos mais três apêndices, com a listagem dos protótipos das implementações realizadas.

Capítulo II

Lógica Modal.

Este capítulo tem por objetivo apresentar a lógica modal [Hughes 68], estabelecendo a sintaxe que será usada ao longo do trabalho e definindo a semântica associada à cada símbolo. Aconselhamos a leitura do apêndice A, onde é descrita a sintaxe e semântica adotada para lógica clássica de primeria ordem, antes da leitura deste capítulo, de forma a familiarizar o leitor com a notação adotada.

II.1 Operadores Modais.

Atualmente, com o uso de processadores mais potentes, tem sido possível realizar grandes avanços na área de sistemas automáticos com programação em lógica. Isto fez ressurgir recentemente o interesse pelo estudo de lógica modal. Esta lógica é uma extensão da lógica de primeira ordem que é mais adequada para tratar determinadas famílias de problemas, como por exemplo: noções de tempo e noções sobre o conhecimento (epistemologia). Na verdade, até poderíamos expressar a maioria destes problemas com a lógica de primeira ordem, mas seus enunciados seriam por demais sobrecarregados.

Um operador modal pode ser entendido como um conectivo lógico unário que tem subfórmulas como argumentos. Para cada problema específico podemos ter um conjunto de operadores modais, entretanto, por ora, nos preocuparemos apenas com os operadores de necessidade ($\Box$) e de possibilidade ($\Diamond$), que definimos a seguir.

Os operadores de necessidade e possibilidade são definidos a partir da relação de acessibilidade. Esta relação, por sua vez, é definida sobre um conjunto de mundos possíveis. E este conjunto é um componente dos modelos de lógica modal.

II.1.1 Modelos de Lógica Modal.

Intuitivamente podemos pensar em cada mundo possível como uma determinada configuração possível da realidade e no mundo real como a realidade na qual nós nos situamos como observadores.

A cada mundo w_i do modelo teremos associados um domínio D_i, a função de interpretação I_i, que dá significado para cada símbolo não-lógico naquele mundo e uma atribuição V_i para as variáveis livres das fórmulas naquele mundo. Ou seja, dentro de um modelo para lógica modal, um mundo é análogo a uma estrutura para lógica de primeira ordem. Dado um símbolo s, denotaremos por s^i a aplicação da interpretação I_i a este símbolo, ou seja, $s^i = I_i(s)$. Denotaremos por W o conjunto, eventualmente infinito, de mundos do modelo. Na maioria dos modelos, trabalhamos com o mesmo domínio para todos os mundos, pois surgem certas dificuldades semânticas ao se relacionar domínios diferentes para cada mundo, como pode ser visto na seção II.3.1.

A relação de acessibilidade R é uma relação definida em W^2. Se o par $\langle w_i, w_j \rangle$ pertence à relação R, dizemos que o mundo w_i acessa o mundo w_j. Esta relação nos diz, dado um observador em um determinado mundo, sobre quais mundos este observador tem conhecimento. Note que um observador no mundo w_i pode ou não ter conhecimento sobre o mundo w_i.

Um modelo em lógica modal para um conjunto de fórmulas é uma tupla $\langle W, w_0, D, R, V, I \rangle$, onde W é o conjunto de mundos, w_0 é o mundo tomado como mundo real, D é um conjunto de pares $\langle D_i, w_i \rangle$, onde D_i é o domínio do mundo w_i, R é a relação de acessibilidade, V é um conjunto de pares $\langle V_i, w_i \rangle$, onde V_i é a atribuição de valores de D_i às variáveis para o mundo w_i, e I é um conjunto de pares $\langle I_i, w_i \rangle$, onde I_i a função de interpretação para os símbolos não-lógicos no mundo w_i.

Diremos que um modelo de lógica modal satisfaz determinada fórmula (ou conjunto de fórmulas), se a avaliação desta fórmula (destas fórmulas) neste modelo for verdadeira. A notação $M, w \models \alpha$ representará que o modelo M satisfaz a fórmula α no mundo w. Diremos que uma fórmula β é conseqüência lógica de uma fórmula α se para qualquer modelo que tomarmos, em qualquer mundo a fórmula $\alpha \supset \beta$ for avaliada como verdadeira.

II.1.2 Operador Necessidade: $\Box$.

O operador necessidade tem aridade igual a um e sua sintaxe segue a mesma de todos os operadores unários.

$$\Box(p \vee \neg p)$$

Semanticamente a fórmula $\Box(\alpha)$ será verdadeira, em um determinado mundo w_i, se e somente se a subfórmula α for verdadeira em todos os mundos acessados por w_i.

II.1.3 Operador Possibilidade: $\Diamond$.

O operador possibilidade também é unário.

$$\Diamond(p(a) \vee p(b))$$

Semanticamente a fórmula $\Diamond(\alpha)$ será verdadeira, em um determinado mundo w_i, se e somente se a subfórmula α for verdadeira em pelo menos um mundo acessado por w_i.

São válidas as seguintes equivalências:

$$\Box(\alpha) \equiv \neg\Diamond(\neg\alpha)$$

$$\Diamond(\alpha) \equiv \neg\Box(\neg\alpha)$$

Note que em um mundo w_i para o qual não existe mundo acessado, teremos que $\Box(\alpha)$ será verdadeira neste mundo, mas $\Diamond(\alpha)$ será falsa, ambos para qualquer fórmula α.

II.2 Principais Sistemas Modais.

Um sistema da lógica modal é composto por: (1) a definição dos operadores modais e (2) um conjunto de restrições quanto aos modelos que podem ser usados neste sistema. A menos que mencionado o contrário, todos os sistemas que vamos tratar usam apenas os operadores necessidade e possibilidade (definidos acima). As restrições mais usuais se aplicam sobre propriedades da relação de acessibilidade. (Poderíamos definir os sistemas modais de outras formas, como por exemplo através de axiomas.)

Os sistemas mais estudados tem os seguintes nomes: K, T, K4, S4, S5. O sistema S5 exige que a relação de acessibilidade seja exatamente igual a W^2, ou seja que relacione cada mundo com ele mesmo e todos os outros. É importante para nós notar que esta relação é reflexiva, simétrica e transitiva. O S4 exige reflexividade e transitividade da relação de acessibilidade. O K4 exige transitividade, o T exige reflexividade, e o K não impõe nenhuma restrição à relação de acessibilidade.

II.3 Outras Definições.

Terminamos este capítulo com algumas definições que serão usadas em várias abordagens estudadas, por isto achamos melhor apresentar estas definições fora das abordagens, de forma que cada capítulo que trata de uma abordagem específica seja independente dos demais capítulos que versam sobre outras abordagens.

II.3.1 A Fórmula de Barcan.

Chamamos de fórmula de Barcan e sua resersa às seguintes fórmulas:

Fórmula de Barcan: $\forall x \Box \alpha \supset \Box(\forall x \alpha)$

Reversa da Fórmula de Barcan: $\Box(\forall x \alpha) \supset \forall x \Box \alpha$

Onde α é uma subfórmula na qual há uma ocorrência livre da variável x. A importância destas fórmulas se deve a controvérsia surgida da discussão da validade delas [Hughes 68]. Basicamente estas fórmulas tocam no problema de associarmos aos mundos domínios diferentes, a fórmula de Barcan é válida em modelos cujo domínio de cada mundo acessado é um subconjunto do domínio do mundo que o acessa, e a resersa é válida quando o domínio de cada mundo acessado for um superconjunto do domínio do mundo que o acessa.

Por exemplo, tomemos o seguinte modelo $\langle W, w_0, D, R, V, I \rangle$, onde:

$$
\begin{aligned}
&W = \{w_0, w_1\} \\
&D = \{\langle \{a, b\}, w_0 \rangle, \langle \{a, c\}, w_1 \rangle\} \\
&R = \{\langle w_0, w_1 \rangle\} \\
&V = \{\}, \\
&I = \{p^0(a) = p^0(b) = true, \\
&\qquad p^1(a) = true, \\
&\qquad p^1(c) = false\}
\end{aligned}
$$

A fórmula de Barcan no modelo acima apresenta o seguinte problema: ao avaliarmos $\forall x \Box p(x)$ em w_0 um dos valores que pode ser atibuído a x é b. Como $w_0 R w_1$, teremos que avaliar $v(p(x))$ em w_1 com $v_t(x) = b$. Porém como $b \notin D_1$, não podemos ter tal avaliação.

Algums modelos assumem a validade da fórmula de Barcan e sua resersa, para isto, estes modelos impõem domínios idênticos para todos os mundos.

II.3.2 Símbolos Rígidos e Símbolos Flexíveis.

Outro problema relacionado à definições diferentes para mundos distintos se refere aos símbolos não-lógicos, por isto, definimos símbolos rígidos e flexíveis abaixo:

Símbolos Rígidos. São constantes, símbolos predicativos e símbolos funcionais que, em um modelo no qual todos os mundos estão associados ao mesmo domínio, tem a mesma definição ou associação com valores para todos os mundos. Ou seja, a função de avaliacão retorna sempre o mesmo valor para estas constantes, símbolos predicativos ou símbolos funcionais, independente do mundo onde eles estão sendo avaliados.

Símbolos Flexíveis. São constantes, símbolos predicativos e símbolos funcionais para os quais não podemos garantir a mesma definição ou a mesma associação com valores para dois mundos distintos.

Por exemplo, se p é um símbolo predicativo rígido, então as sentenças abaixo são válidas:

$$\forall x_1, \ldots, \forall x_n \Diamond p(x_1, \ldots, x_n) \supset p(x_1, \ldots, x_n)$$

$$\forall x_1, \ldots, \forall x_n p(x_1, \ldots, x_n) \supset \Box p(x_1, \ldots, x_n)$$

Capítulo III

Abordagem 1 – Martín Abadi e Zohar Manna.

A primeira abordagem para automação de prova em lógica modal com quantificadores, que vamos apresentar, foi proposta por Martín Abadi e Zoahr Manna em 1986 [Abadi 86]. Esta abordagem visa a implementação de um sistema completamente automático para prova de fórmulas da lógica modal de primeira ordem; ou seja, fórmulas com operadores modais e variáveis quantificadas. A completude (refutacional) do sistema é afirmada [Abadi 86] e a representação na forma clausal não é requerida.

III.1 Apresentação da Abordagem.

Este sistema se baseia em refutação. Dada uma fórmula α a ser provada, partimos de S_1 igual a $\neg\alpha$, e tentamos criar uma seqüência $S_1, \ldots, S_n$, onde S_n seja igual a $false$. Cada fórmula é gerada a partir de sua predecessora pela aplicação de uma regra. Para cada sistema modal é usado um conjunto diferente de regras. São apresentadas regras para os principais sistemas de lógica modal; entre eles: K, T, K4, S4 e S5.

Os modelos tratados apresentam uma simplificação em relação aos modelos mais gerais da lógica modal: É requerido que os domínios sejam os mesmos para cada um dos mundos. Com isto, teremos a validade da fórmula de Barcan e sua reversa para os modelos aqui tratados, conforme comentado na seção II.3.1. Essa abordagem define modelo como a quintupla $\langle D, W, w_0, R, I\rangle$, onde D já é o domínio comum aos mundos.

III.1.1 Definições e Notações.

A aplicação de cada regra está sujeita a uma série de restrições. A fim de estabelecer estas restrições, definimos abaixo alguns conceitos que serão usados a seguir.

Subfórmula de polaridade positiva. São as subfórmulas de uma fórmula que estão no escopo de um número par de negações explícitas ou implícitas. Dizemos que uma subfórmula está no escopo de uma negação explícita se ela está no escopo de um conectivo de negação. Dizemos que uma subfórmula está no escopo de uma negação implícita se ela está no escopo esquerdo de um conectivo de implicação.

Subfórmula de polaridade negativa. São as subfórmulas de uma fórmula que estão no escopo de um número impar de negações explícitas ou implícitas.

Quantificadores de força universal. São quantificadores universais com polaridade positiva ou quantificadores existenciais com polaridade negativa. Dizemos que um quantificador Q tem polaridade positiva se a subfórmula que ele forma, $(Qx(\alpha))$, tem polaridade positiva. Identicamente, dizemos que um quantificador Q tem polaridade negativa se a subfórmula que ele forma, $(Qx(\alpha))$, tem polaridade negativa. Os quantificadores de força universal serão denotados como $Q^{\forall}$.

Quantificadores de força existencial. São quantificadores existenciais com polaridade positiva ou quantificadores universais com polaridade negativa. Serão denotados como $Q^{\exists}$.

Operadores de força de necessidade. São operadores de necessidade com polaridade positiva ou operadores de possibilidade com polaridade negativa. Dizemos que um operador O tem polaridade positiva se a subfórmula que ele forma, $(O(\alpha))$, tem polaridade positiva. Identicamente, dizemos que um operador O tem polaridade negativa se a subfórmula que ele forma, $(O(\alpha))$, tem polaridade negativa.

Operadores de força de possibilidade. São operadores de possibilidade com polaridade positiva ou operadores de necessidade com polaridade negativa.

Notação. Ao longo deste capítulo serão usadas as seguintes notações: Se $A\langle B\rangle$ é uma fórmula A onde ocorre a subfórmula ou termo B, então $A\langle C\rangle$ é a fórmula A', formada a partir de A, pela substituição de uma ocorrência de B por C. Similarmente, $A[C]$ denota a substituição de todas as ocorrências de B por

C em A. Adicionalmente, se $A\langle B_1, \ldots, B_n\rangle$ é uma fórmula onde ocorrem as subfórmulas B_i, com i variando de 1 até n, então $A\langle C\rangle$ é a substituição de cada B_i por C.

III.1.2 Sintaxe e Aplicação das Regras.

As regras podem ser de dois tipos: regras de simplificação ou regras de dedução. Elas serão diferenciadas pela sintaxe de seus enunciados, através do uso do símbolo $\Longrightarrow$ ou do símbolo $\longrightarrow$, conforme pode ser observado abaixo. O resultado da aplicação de regras de um tipo ou de outro é diferente.

Regras de simplificação.

Forma: $A_1, \ldots, A_n \Longrightarrow B$

Aplicação: Se S_j é uma fórmula que contém uma conjunção de subfórmulas; onde A_i, de i variando de 1 até n, são algumas destas subfórmulas; então podemos gerar a fórmula S_{j+1} removendo todas as subfórmulas A_i e acrescentando à conjunção a subfórmula B.

Exemplo:

$$A, \neg A \Longrightarrow false$$

$$S_j = p \wedge q \wedge \neg p$$
$$S_{j+1} = q \wedge false$$

■

■

O resultado da aplicação de uma regra de simplificação é a substituição das subfórmulas A_i pela subfórmula B.

Regras de dedução.

Forma: $A_1, \ldots, A_n \longrightarrow B$

Aplicação: Se S_j é uma fórmula que contém uma conjunção de subfórmulas; onde A_i, de i variando de 1 até n, são algumas destas subfórmulas; então podemos gerar a fórmula S_{j+1} acrescentando à conjunção a subfórmula B.

Exemplo:

$$\Box A \longrightarrow A$$

$$S_j = p \vee (\neg q \wedge \Box q)$$
$$S_{j+1} = p \vee (\neg q \wedge \Box q \wedge q)$$

■

■

O resultado da aplicação de uma regra de dedução é a inclusão da subfórmula B em uma conjunção.

As regras de simplificação, da forma $A_1, \ldots, A_n \Longrightarrow B$, e as regras de dedução, da forma $A'_1, \ldots, A'_m \longrightarrow B'$, apresentadas no método têm a propriedade de apresentar as fórmulas $(A_1 \wedge \ldots \wedge A_n) \supset B$ e $(A'_1 \wedge \ldots \wedge A'_m) \supset (A'_1 \wedge \ldots \wedge A'_m \wedge B')$, respectivamente válidas, desde que respeitadas as restrições para aplicação das regras. Logo, aplicando as regras de simplificação, estaremos substituindo subfórmulas de uma conjunção por uma conseqüência lógica destas. Desta forma, a sentença S_{j+1}, gerada por uma regra de simplificação ou de dedução, será sempre uma conseqüência lógica de S_j.

Uma vez que poderemos demostrar que $S_j \models S_{j+1}$, para qualquer j durante a aplicação do método, poderemos nos assegurar de sua correção. Pois, se $S_1 = \neg\alpha$, e $S_1 \supset \ldots \supset S_n$, e $S_n = false$, então $\alpha = true$.

Entretanto, vale observar que as fórmulas S_j e S_{j+1} nem sempre serão equivalentes. Isto é, existem regras que darão origem a S_{j+1}, a partir de S_j, tal que existirá uma função de avaliação para a qual a avaliação de S_j será diferente da avaliação de S_{j+1}. Ou melhor, a avaliação de S_j será $false$, enquanto a avaliação de S_{j+1} será $true$. Dizemos neste caso que S_{j+1} é uma conseqüência lógica mais fraca de S_j. O uso destas regras é útil para gerar fórmulas, que embora eventualmente mais fracas, são mais facilmente refutadas.

III.1.3 Restrições.

A seguir listamos a restrição geral a todas as regras. Ao enunciarmos as regra serão mencionadas as restrições específicas a cada uma delas.

Restrição geral. As regras só podem ser aplicadas se as subfórmulas A_i tem polaridade positiva.

Esta restrição é imprescindível para o funcionamento correto da maioria das regras, e para a demonstração da correção feita acima.

III.1.4 Regras Gerais.

A seguir começamos a apresentar as regras. Primeiro listamos as regras independentes do sistema modal e que não são sensíveis à presença de quantificadores.

Regras de simplificação true-false.

$$false \vee A \Longrightarrow A \qquad \text{(R–3.1)}$$
$$false, A \Longrightarrow false \qquad \text{(R–3.2)}$$
$$\Diamond false \Longrightarrow false \qquad \text{(R–3.3)}$$

Estas regras visam a simplificação de subfórmulas contendo os símbolos predicativos *true* ou $false$. Por exemplo, ao provarmos α abaixo, saimos de S_1 e chegamos a S_5, onde usamos a regra R–3.2 para terminar a refutação.

$\alpha = (p \wedge \neg q) \vee \neg p \vee q$

$S_1 = (\neg p \vee q) \wedge p \wedge \neg q$
$\vdots$
$S_5 = (\neg p \vee q) \wedge p \wedge \neg q \wedge q \wedge false$
$S_6 = false$ .R–3.2
■

Regras de negação.

$$\neg\Box A \Longrightarrow \Diamond\neg A \qquad \text{(R–3.4)}$$
$$\neg\Diamond A \Longrightarrow \Box\neg A \qquad \text{(R–3.5)}$$
$$\neg(A \wedge B) \Longrightarrow \neg A \vee \neg B \qquad \text{(R–3.6)}$$
$$\neg(A \vee B) \Longrightarrow \neg A \wedge \neg B \qquad \text{(R–3.7)}$$
$$\neg\neg A \Longrightarrow A \qquad \text{(R–3.8)}$$

Estas regras permitem a manipulação de subfórmulas contendo o operador de negação. No exemplo anterior, poderíamos ter iniciado com $S_1 = \neg\alpha$ diretamente e então aplicar as regras de negação para iniciar a refutação.

$$\alpha = (p \wedge \neg q) \vee \neg p \vee q$$

$$
\begin{aligned}
S_1 &= \neg((p \wedge \neg q) \vee \neg p \vee q) \\
S_2 &= \neg(p \wedge \neg q) \wedge \neg(\neg p \vee q) && .\text{R–3.7} \\
S_3 &= (\neg p \vee \neg\neg q) \wedge \neg(\neg p \vee q) && .\text{R–3.6} \\
S_4 &= (\neg p \vee q) \wedge \neg(\neg p \vee q) && .\text{R–3.8} \\
S_5 &= (\neg p \vee q) \wedge \neg\neg p \wedge \neg q && .\text{R–3.7} \\
S_6 &= (\neg p \vee q) \wedge p \wedge \neg q && .\text{R–3.8} \\
&\vdots \\
&\blacksquare
\end{aligned}
$$

Regra de enfraquecimento.

$$A, B \Longrightarrow A \qquad \text{(R–3.9)}$$

Esta regra normalmente é usada em fórmulas geradas por aplicações de regras de dedução, com ela simplesmente descartamos trechos da fórmula que não nos serão mais úteis.

A aplicação desta regra visa evitar a repetição ao longo da prova de termos de uma conjunção para os quais temos o "sentimento" que não serão mais úteis. Esta regra não é muito importante na implementação automática da abordagem se o uso de memória não for crítico.

Regra de distribuição.

$$A, B_1 \vee \ldots \vee B_n \Longrightarrow (A \wedge B_1) \vee \ldots \vee (A \wedge B_n) \qquad \text{(R–3.10)}$$

Esta regra implementa a propriedade da distribuição do conectivo de disjunção sobre o de conjunção. Abaixo listamos um pequeno exemplo em que esta regra é utilizada.

$$
\begin{aligned}
S_1 &= p \wedge q \wedge (\neg p \vee \neg q) \\
S_2 &= q \wedge ((p \wedge \neg p) \vee (p \wedge \neg q)) && .\text{R–3.10} \\
S_3 &= q \wedge (false \vee (p \wedge \neg q)) && .(*) \\
S_4 &= q \wedge p \wedge \neg q && .\text{R–3.1} \\
S_7 &= false && .(*) \\
&\blacksquare
\end{aligned}
$$

(*): Estes passos necessitam de regras ainda não apresentadas.

■

Regra de corte.

$$\longrightarrow A \vee \neg A \qquad \text{(R–3.11)}$$

Os autores alegam que a existência desta regra é fundamental para a completude da abordagem. Como pode ser observado esta regra permite acrescentar à qualquer conjunção um novo termo formado pela disjunção de uma subfórmula qualquer e sua negação. Como esta subfórmula é qualquer, esta regra é o principal empecilho para a implementação da abordagem. Em um sistema automático a aplicação desta regra depende da aplicação de heurísticas para decidir que subfórmula A deve ser usada. Voltaremos a comentar esta regras na seção III.2.

Regras gerais para operadores modais.

$$\Box A, \Diamond B \longrightarrow \Diamond(A \wedge B) \qquad \text{(R–3.12)}$$
$$\Diamond C \longrightarrow C \qquad \text{(R–3.13) (*)}$$
$$C \longrightarrow \Box C \qquad \text{(R–3.14) (*)}$$

(*) Restrição: C só contém símbolos rígidos.

A primeira regra vale sob qualquer circunstância, enquanto que as outras duas só podem ser aplicadas sobre subfórmulas que não contenham símbolos flexíveis. O significado informal da segunda regra é o seguinte: se uma fórmula tem o mesmo valor em todos os mundos e é verdadeira em pelo menos um mundo, então ela é verdadeira no mundo real. A terceira fórmula tem significado análogo. Vale a pena notar que, embora a aplicação das regras seja unicamente baseada na sintaxe, a verificação das restrições pode ser semântica, como no caso acima. Esta interferência da semântica em procedimentos puramente sintáticos, quando aplicados á lógica clássica de primeira ordem, será discutida na conclusão final.

Um exemplo de uso destas regras:

$$S_1 = \Box p \wedge \Diamond \neg p$$
$$S_2 = \Box p \wedge \Diamond \neg p \wedge \Diamond(p \wedge \neg p) \qquad \text{.R–3.12}$$
$$S_3 = \Box p \wedge \Diamond \neg p \wedge \Diamond(p \wedge \neg p \wedge false) \qquad \text{.(*)}$$
$$S_4 = \Diamond(p \wedge \neg p \wedge false) \qquad \text{.R–3.9}$$
$$S_5 = \Diamond(false) \qquad \text{.R–3.2}$$
$$S_6 = false \qquad \text{.R–3.3}$$

■

(*): Este passo necessita de regra ainda não apresentada.

■

Note que quase todas as regras possuem subfórmulas equivalentes como antecedentes e conseqüentes; por exemplo, a regras $\neg\Box A \Longrightarrow \Diamond\neg A$ tem a equivalência das subfórmulas $\neg\Box A$ e $\Diamond\neg A$ comentadas na seção II.1.3. E as demais regras, como por exemplo a regra de enfraquecimento $A, B \Longrightarrow A$, tem o conseqüente como conseqüência lógica do antecedente; ou seja, $(A \wedge B) \supset A$.

III.1.5 Regra Simplificada de Resolução.

Agora introduzimos a regra simplificada de resolução. Ela também é válida para todos os sistemas modais, mas esta versão é simplificada de forma a não tratar os quantificadores.

Regra simplificada de resolução.

$$A\langle C,\ldots,C\rangle, B\langle C,\ldots,C\rangle \longrightarrow A\langle true\rangle \vee B\langle false\rangle \qquad \text{(R–3.15}^{-}\text{)}$$

Informalmente podemos entender a regra acima da seguinte forma. Temos uma conjunção onde ocorrem como termos as subfórmulas A e B, e estas subfórmulas contêm C como subfórmula. A subfórmula C irá assumir o valor *true* ou o valor $false$; logo, se gerarmos as subfórmulas A' e B' a partir de A e B pela substituição de C por *true* em A e de C por $false$ em B, temos que se $A \wedge B$ for *true* então $A' \vee B'$ também será *true*. Desta maneira estamos acrescentando na conjunção um novo termo que é conseqüência lógica de dois termos antigos. Temos que:

$$(A \wedge B) \supset (A' \vee B') \qquad (1)$$
$$(A \wedge B \wedge (A' \vee B')) \supset (A \wedge B) \qquad (2)$$

Pelo texto acima, mostramos a validade da fórmula (1); e a validade da fórmula (2) pode ser facilmente percebida através da definição dos conectivos de conjunção e implicação. Observando a fórmulas (1) e (2) vemos que a substituição originada por esta regra opera sobre subfórmulas equivalentes.

$$(A \wedge B) \equiv (A \wedge B \wedge (A' \vee B'))$$

Exemplo:

$\alpha = (p \supset q \wedge p) \supset q$

$S_1 = (\neg p \vee q) \wedge p \wedge \neg q$
$S_2 = (\neg p \vee q) \wedge p \wedge \neg q \wedge ((\neg true \vee q) \vee false)$ $\quad .A = \neg p \vee q, B = p, C = p$
$S_3 = (\neg p \vee q) \wedge p \wedge \neg q \wedge q$
$S_4 = (\neg p \vee q) \wedge p \wedge \neg q \wedge q \wedge (\neg true \vee false)$ $\quad .A = \neg q, B = q, C = q$
$S_5 = (\neg p \vee q) \wedge p \wedge \neg q \wedge q \wedge false$
$S_6 = false$

■

Esta regra não pode ser aplicada em ocorrências de C em A ou B que estejam no escopo de algum operador modal, conforme pode ser visto com o auxílio do exemplo abaixo:

$$S_1 = p \wedge \Diamond \neg p$$

A fórmula S_1 é safisteita pelo modelo $M = \langle D, W, w_0, R, I \rangle$, onde:

$$W = \{w_0, w_1\}$$
$$R = \{\langle w_0, w_1 \rangle\}$$
$$I = \{p^0 = true, \; p^1 = false\}$$

Se aplicarmos a regras da resolução simplificada chegamos a uma fórmula inválida.

$S_1 = p \wedge \Diamond \neg p$
$S_2 = .p \wedge \Diamond \neg p \wedge (false \vee \Diamond \neg true)$ $\quad .A = p, B = \Diamond \neg p, C = p$
$S_3 = p \wedge \Diamond \neg p \wedge false$
$S_4 = false$

■

O problema surge pois o termo p em A (p) e em B ($\Diamond \neg p$) está ocorrendo em mundos distintos; logo, pode assumir valores diferentes.

■

III.1.6 Regras Dependentes do Sistema Modal.

As regras dependentes do sistema modal estão relacionadas com a relação de acessibilidade. Isto é, para cada característica que pudermos garantir na relação de

acessibilidade, teremos um conjunto de regras que poderão ser utilizadas, conforme apresentamos abaixo.

Sistema modal	Características da relação de acessibilidade
K	nenhuma especial
T	reflexividade
K4	transitividade
S4	reflexividade e transitividade
S5	reflexividade, simetria e transitividade

Regra válida para relações de acessibilidade reflexivas (eg. T, S4, S5).

$$\Box A \longrightarrow A \qquad \text{(R–3.16)}$$

Regra válida para relações de acessibilidade transitivas (eg. K4, S4, S5).

$$\Box A, \Diamond B \longrightarrow \Diamond(\Box A \wedge B) \qquad \text{(R–3.17)}$$

Regras válidas para relações de acessibilidade simétricas (eg. S5).

$$\Diamond A, \Diamond B \longrightarrow \Diamond(\Diamond A \wedge B) \qquad \text{(R–3.18)}$$
$$A \longrightarrow \Diamond A \qquad \text{(R–3.19)}$$

Podemos notar que as regras estão justamente expressando as características da relação de acessibilidade. Por exemplo, a regra $\Box A \longrightarrow A$ para relações reflexivas. Se a relação é reflexiva então cada mundo acessa a si mesmo. $\Box A$ diz que em todos os mundos acessados A é *true*. Como o mundo real acessa a si mesmo e A é *true* nos mundos acessados, então A é *true* no mundo real.

Um exemplo de uso em S5.

$$\alpha = \neg\Diamond\Diamond p \vee \Box\Diamond p$$

$$S_1 = \neg(\neg\Diamond\Diamond p \vee \Box\Diamond p)$$
$$S_2 = \neg\neg\Diamond\Diamond p \wedge \neg\Box\Diamond p \qquad \text{.R–3.7}$$
$$S_3 = \Diamond\Diamond p \wedge \neg\Box\Diamond p \qquad \text{.R–3.8}$$
$$S_4 = \Diamond\Diamond p \wedge \Diamond\neg\Diamond p \qquad \text{.R–3.4}$$
$$S_5 = \Diamond\Diamond p \wedge \Diamond\Box\neg p \qquad \text{.R–3.5}$$
$$S_6 = \Diamond\Diamond p \wedge \Diamond\Box\neg p \wedge \Diamond(\Diamond\Diamond p \wedge \Box\neg p) \qquad \text{.R–3.18}$$
$$S_7 = \Diamond(\Diamond\Diamond p \wedge \Box\neg p) \qquad \text{.R–3.9}$$

$S_8 = \Diamond(\Diamond\Diamond p \wedge \Box\neg p \wedge \Diamond(\Diamond p \wedge \Box\neg p))$ .R–3.17
$S_9 = \Diamond\Diamond(\Diamond p \wedge \Box\neg p)$ .R–3.9
$S_{10} = \Diamond\Diamond(\Diamond p \wedge \Box\neg p \wedge \Diamond(p \wedge \Box\neg p))$ .R–3.17
$S_{11} = \Diamond\Diamond\Diamond(p \wedge \Box\neg p)$ .R–3.9
$S_{12} = \Diamond\Diamond\Diamond(p \wedge \Box\neg p \wedge \neg p)$ .R–3.16
$S_{13} = \Diamond\Diamond\Diamond(p \wedge \Box\neg p \wedge \neg p \wedge false)$ .R–3.15
$S_{14} = \Diamond\Diamond\Diamond(false)$ .R–3.2
$S_{15} = \Diamond\Diamond(false)$ .R–3.3
$S_{16} = \Diamond(false)$ .R–3.3
$S_{17} = false$ .R–3.3

■

Note bem que a sentença provada acima é válida em S5, mas não é válida em S4, isto se reflete no uso da regra R–3.18, para gerar o termo S_6. Logo, em S4 a fórmula acima não teria sido provada.

■

III.1.7 Regras de Extração de Quantificadores.

Para podermos aplicar a regra completa de resolução, precisamos antes usar as regras de extração de quantificadores.

Regras de extração de quantificadores.

$A\langle Q^{\forall} x B[x]\rangle \Longrightarrow \forall x' A\langle B[x']\rangle$ (R–3.20) (*)
$A\langle Q^{\exists} x B[x]\rangle \Longrightarrow \exists x' A\langle B[x']\rangle$ (R–3.21) (*)

(*) Onde x' é uma variável nova, que não ocorre nem em A nem em B.

Restrição: A regra de extração de quantificadores de força existencial não pode ser aplicada em quantificadores que ocorrem dentro de A no escopo de quantificadores de força universal ou dentro de A no escopo de operadores modais de força de necessidade.

Esta restrição visa não permitir a destruição de dependências entre as variáveis e não sobrepassar o relacionamento entre variáveis e mundos.

Note que as regras acima apenas movimentam os quantificadores. Partindo do princípio que a fórmula a ser provada não continha originalmente quantificadores desnecessários, estes só poderão aparecer ao aplicarmos a regra completa de resolução, que é apresentada abaixo. Logo, é parte integrante do algoritmo de aplicação desta regra a eliminação de quantificadores que tenham se tornado desnecessários.

III.1.8 Regra Completa de Resolução.

Ao aplicarmos a regra completa de resolução estaremos fazendo uso do processo de unificação, estendido de forma a tratar os operadores modais e os quantificadores. Os operadores de necessidade e possibilidade são tratados como conectivos unários comuns, assim como os quantificadores. Entretanto, as variáveis não quantificadas não podem ser unificadas com variáveis quantificadas universalmente.

Regra completa de resolução.

$$Q_1x_1 \ldots Q_hx_hA\langle C_1, \ldots, C_n\rangle, R_1y_1 \ldots R_ky_kB\langle C_{n+1}, \ldots, C_m\rangle \longrightarrow$$
$$S_1z_1 \ldots S_{h+k}z_{h+k}(A\theta\langle true\rangle \vee B\theta\langle false\rangle) \qquad \text{(R–3.15}^+\text{)}$$

Onde θ é um unificador mais geral de $C_1, \ldots, C_m$, e $Q_1, \ldots, Q_h, R_1, \ldots, R_k, S_1, \ldots, S_{h+k}$ são quantificadores que obedecem as seguintes condições:

(i) As variáveis $x_1, \ldots, x_h, y_1, \ldots, y_k$ são todas diferentes.

(ii) A seqüência de quantificadores $S_1z_1 \ldots S_{h+k}z_{h+k}$ é um arranjo das seqüências $Q_1x_1 \ldots Q_hx_h$ e $R_1y_1 \ldots R_ky_k$, na ordem definida abaixo.

(iii) As subfórmulas $C_1\theta, \ldots, C_m\theta$ não estão no escopo de operadores modais ou contém unicamente símbolos rígidos.

(iv) As subfórmulas $C_1\theta, \ldots, C_m\theta$ não ocorrem no escopo de algum quantificador dentro de $A\theta$ ou $B\theta$.

(v) Se $\langle x \Leftarrow t\rangle \in \theta$, então para i, $1 \leq i \leq h+k$, tal que $S_i = \forall$ e $z_i = x$, não pode ocorrer em t uma variável que esteja ligada em $S_{i+1}z_{i+1} \ldots S_{h+k}z_{h+k}$ $(A \vee B)$. Ou seja, t não pode depender de x.

(vi) Se $\langle x \Leftarrow t\rangle \in \theta$ e ocorre um símbolo flexível em t então x não pode ocorrer no escopo de nenhum operador modal em A ou B.

A condição (i) apenas exige a renomeação das variáveis, enquanto a condição (ii) especifica que os quantificadores do novo termo serão formados por um arranjo dos demais quantificadores. A ordenação dentro deste arranjo é estabelecida abaixo. A condição (iii) é específica para lógica modal, e ela justamente evita o uso da regras em situações possivelmente problemáticas entre operadores modais e quantificadores. A condição (iv) visa não deixar que a inclusão de quantificaores como parte de A ou B mascarem o respeito às demais condições. A condição (v) garante que a aplicação do unificador θ não captura variáveis livres. A condição (vi) estende a condição (iii) para os demais termos de A e B que são alterados pela unificação.

Vejamos com mais detelhes os problemas que ocorrem se as condições (iii), (v) e (vi) não forem satisfeitas.

Condição (iii). Suponha, por exemplo, que partirmos da sentença S_1 abaixo:

$$S_1 = \forall x \neg \Box p(x) \wedge \Box p(a)$$

A fórmula S_1 é satisfeita pelo modelo $M = \langle D, W, w_0, R, I \rangle$, onde:

$$\begin{aligned} D &= \{0, 1\} \\ W &= \{w_0, w_1\} \\ R &= \{\langle w_0, w_0 \rangle, \langle w_0, w_1 \rangle, \langle w_1, w_0 \rangle, \langle w_1, w_1 \rangle\} \\ I &= \{a^0 = 0, \\ &\quad a^1 = 1, \\ &\quad p^0(0) = true, \\ &\quad p^1(1) = true, \\ &\quad p^0(1) = p^1(0) = false\} \end{aligned}$$

Logo, a fórmula S_1 não deve ser refutada. Vejamos a aplicação da regra de resolução, sem a verificação da condição (iii):

$$\begin{aligned} S_1 &= \forall x \neg \Box p(x) \wedge \Box p(a) \qquad .\{A = \neg \Box p(x), B = \Box p(a), \\ &\qquad\qquad\qquad\qquad\quad C_a = \Box p(x), C_b = \Box p(a), \\ &\qquad\qquad\qquad\qquad\quad \theta = \{\langle x \Leftarrow a \rangle\}\} \\ S_2 &= \forall x \neg \Box p(x) \wedge \Box p(a) \wedge \\ &\qquad (\neg true \vee false) \\ S_3 &= false \end{aligned}$$

■

Chegamos a um resultado incorreto, pois na aplicação da regra de resolução, o símbolo a, que é flexível, encontra-se no escopo de operadores modais, violando a condição (iii).

■

Condição (v). Tomemos a fórmula S_1 abaixo:

$$\forall x \exists y (\neg p(x) \wedge p(t(y)) \vee p(x) \wedge \neg p(t(y)))$$

Esta fórmula é satisfeita pelo modelo $M = \langle D, W, w_0, R, I \rangle$, onde:

$$\begin{aligned} D &= \{0, 1\} \\ W &= \{w_0\} \\ R &= \{\langle w_0, w_0 \rangle\} \\ I &= \{t^0(0) = 1, \\ &\quad t^0(1) = 0, \\ &\quad p^0(0) = true, \\ &\quad p^0(1) = false\} \end{aligned}$$

Vejamos aplicação da regra de resolução nesta fórmula, sem testar a condição (v):

$S_1 = \forall x \exists y(\neg p(x) \wedge p(t(y)) \vee p(x) \wedge \neg p(t(y)))$

$S_2 = \exists y(\neg p(t(y)) \wedge p(t(y) \wedge$
$(\neg true \vee false)) \vee p(t(y)) \wedge \neg p(t(y)))$ $.\{A = \neg p(x), B = p(t(y)),$
$C_a = p(x), C_b = p(t(y)),$
$\theta = \{\langle x \Leftarrow t(y) \rangle\}\}$

$S_3 = \exists y(p(t(y)) \wedge \neg p(t(y)))$

$S_4 = \exists y(p(t(y)) \wedge \neg p(t(y)) \wedge$
$(\neg true \vee false))$ $.\{A = \neg p(t(y)), B = p(t(y)),$
$C_a = p(t(y)), C_b = p(t(y)),$
$\theta = \{\}\}$

$S_5 = false$

■

De novo, chegamos a um resultado incorreto, pois, a condição (v) não era satisfeita na primeira aplicação da regra de resolução.

Note que esta condição, não está relacionada a operadores modais, ela é genérica para lógica de primeira ordem.

■

Condição (vi). Vamos considerar a seguinte fórmula:

$$\forall x((\neg q(x) \vee \Box p(x) \wedge \Diamond \neg p(a)) \wedge q(a))$$

Esta fórmula é satisfeita pelo seguinte modelo $M = \langle D, W, w_0, R, I \rangle$, onde:

$$
\begin{aligned}
&D = \{0, 1\} \\
&W = \{w_0, w_1\} \\
&R = \{\langle w_0, w_0 \rangle, \langle w_0, w_1 \rangle, \langle w_1, w_1 \rangle\} \\
&I = \{a^0 = 0, a^1 = 1, \\
&\quad p^0(0) = true, p^1(0) = true, \\
&\quad p^0(1) = false, p^1(1) = false, \\
&\quad q^0(0) = true, q^1(0) = false, \\
&\quad q^0(1) = false, q^1(1) = false\}
\end{aligned}
$$

A aplicação da regra de resolução na fórmula acima nos fornece o seguinte resultado, se não nos preocuparmos com a condição (vi):

$S_1 = \forall x((\neg q(x) \vee \Box p(x) \wedge \Diamond \neg p(a)) \wedge q(a))$

$S_2 = (\neg q(a) \vee \Box p(a) \wedge \Diamond \neg p(a)) \wedge q(a) \wedge$
$(\neg true \vee \Box p(a) \wedge \Diamond \neg p(a) \vee false)$.$\{A = \neg q(x) \vee \Box p(x) \wedge \Diamond \neg p(a),$
$B = q(a), C_a = q(x), C_b = q(a),$
$\theta = \{\langle x \Leftarrow a \rangle\}\}$

$S_3 = (\neg q(a) \vee \Box p(a) \wedge \Diamond \neg p(a)) \wedge q(a) \wedge$
$(\Box p(a) \wedge \Diamond \neg p(a))$

$S_4 = \Box p(a) \wedge \Diamond \neg p(a)$ .R–3.9

$S_5 = \Box p(a) \wedge \Diamond \neg p(a) \wedge \Diamond (p(a) \wedge \neg p(a))$ .R–3.12

$S_6 = \Diamond (p(a) \wedge \neg p(a))$ .R–3.9

$S_7 = \Diamond (p(a) \wedge \neg p(a) \wedge (\neg true \vee false))$.$\{A = \neg p(a), B = p(a),$
$C_a = p(a), C_b = p(a),$
$\theta = \{\}\}$

$S_8 = \Diamond (false)$

$S_9 = false$ .R–3.3

■

Chegamos novamente a um resultado incorreto, pois, a condição (vi) não era satisfeita na primeira aplicação da regra de resolução.

■

Após a aplicação da regra completa de resolução os quantificadores que se tornaram redundantes são descartados, como pode ser observado na passagem abaixo,

extraída do exemplo da necessidade da condição (v):

$$S_1 = \forall x \exists y(\neg p(x) \wedge p(t(y)) \vee p(x) \wedge \neg p(t(y)))$$
$$S_2 = \exists y(\neg p(t(y)) \wedge p(t(y) \wedge (\neg true \vee false)) \vee$$
$$p(t(y)) \wedge \neg p(t(y))) \qquad .\{A = \neg p(x), B = p(t(y)),$$
$$C_a = p(x), C_b = p(t(y)),$$
$$\theta = \{\langle x \Leftarrow t(y)\rangle\}\}$$

■

A condição (ii) estabalece que os quantificadores $S_1, \ldots, S_{h+k}$ do termo acrescentado serão um arranjo dos quantificadores dos termos $Q_1x_1 \ldots Q_hx_hA$ e $R_1y_1 \ldots R_ky_kB$. Com o objetivo de diminuir o espaço de busca da ordenação correta destes quantificadores é definida a ordenação parcial ($\prec$) abaixo:

(a) $S_iz_i \prec S_jz_j$ se $\langle z_j \Leftarrow t\langle z_i\rangle\rangle \in \theta$.

(b) $S_iz_i \prec S_jz_j$ se S_iz_i está no escopo de S_jz_j em $Q_1x_1 \ldots Q_hx_h$ ou $R_1y_1 \ldots R_ky_k$.

A ordenação dos quantificadores estará correta se pudermos garantir a validade da sentença abaixo:

$$\forall i \forall j(S_iz_i \prec S_jz_j \supset i < j)$$

III.2 Comentários.

A abordagem é apresentada como um método correto e completo para a implementação automática de prova de fórmulas de vários sistemas da lógica modal. A aplicabilidade da abordagem em um sistema qualquer da lógica modal está atrelada à determinação das regras necessárias para garantir a correção e a completude (refutacional) do método no dado sistema. Na descrição feita acima nos detivemos apenas nas regras necessárias aos sistema K, T, K4, S4 e S5.

A correção do método é facilmente demostrada, pois sendo baseado em refutação, basta garantir que cada regra sempre gere uma fórmula que é conseqüência lógica da antecessora, o que pode ser verificado em todas as regras apresentadas. Ou seja, dada α, partimos de $S_1 = \neg\alpha$, geramos $S_1, \ldots, S_n$, tal que $S_1 \supset \ldots \supset S_n$ e temos que $S_n = false$, logo $S_1 = false$ e $\alpha = true$.

O problema da completude, nesta abordagem é bem mais complexo. Ele se contrapõe à possibilidade de implementação completamente automática por causa da regras do corte ($\longrightarrow A \vee \neg A$). A existência desta regra é essencial para a completude do método [Abadi 86], entretanto, a sua aplicação necessita de um alto grau

de "inteligência" por parte do sistema ou da intervenção do usuário. A descrição original da abordagem sugere que a implementação seja guiada pelo usuário, com a especificação iterativa e interativa da aplicação da regras do corte e da escolha do termo a ser usado nesta regra.

III.3 Implementação.

Com o objetivo de avaliar mais precisamente os problemas e as características da abordagem, foi implementado um protótipo do método. A listagem do programa encontra-se no apêndice B.

Para a implementação foi escolhida a linguagem Prolog, principalmente por quatro motivos: (1) Facilidades oferecidas pela linguagem para implementar o reconhecimento das fórmulas na entrada de dados [Cohen 87] e a formatação delas na saída de dados. (2) Facilidade de manipulação de dados simbólicos, não homogêneos e estruturados, pois as fórmulas são representadas, ora por listas, ora por árvores, e cada elemento destas estruturas pode ter formas diversas (e.g. identificadores, subárvores e quantificadores). (3) Facilidades oferecidas pela linguagem do uso de mecanismos de retrocesso na implementação do algoritmo de busca de soluções. (4) Familiarização do autor deste trabalho com a linguagem.

Como mencionado acima, o próprio mecanismo de retrocesso oferecido pela linguagem será usado para selecionar a regra que deve ser utilizada em determinado momento e para realizar uma eventual volta a um passo anterior da prova, tentando um novo caminho, uma vez que o anteriormente tomado não tenha levado a uma refutação.

Como a implementação não visa ser um sistema profissional, mas apenas um protótipo de testes, algumas simplificações foram adotadas. Por exemplo, o programa espera que o usário não use duas variáveis com o mesmo nome na mesma fórmula. Por exemplo:

$$\forall x p(x) \supset \exists x p(x)$$

A fórmula acima deve ser codificada como:

$$\forall x 1 p(x1) \supset \exists x 2 p(x2)$$

De forma a aumentar a eficiência da implementação e diminuir o espaço de busca, facilitando o acompanhamento da execução, alguns passos que na descrição da abordagem são feitos através do uso de regras foram implemetados de forma procedimental, antes da aplicação das regras propriamente. Desta forma o algoritmo básico pode ser expresso como:

1. Leia uma linha.
2. Converta a linha para uma lista de caracteres.
3. Converta a lista de caracteres em lista de Tokens.
4. Proceda a análise sintática dos Tokens e gere uma estrutura para representar a fórmula lida.
5. Simplifique a fórmula.
6. Extraia os quantificadores possíveis.
7. Elimine os quantificadores desnecessários.
   (a) Se a regra gerada é igual a $false$ pare.
8. Tome uma regra. /* ponto de retrocesso */
9. Aplique a regra.
   (a) Se a regra não pode ser aplicada retorne ao ponto de retrocesso mais recente que ainda possue alternativa.
10. Volte a 6.

Na verdade o ponto de retrocesso indicado no algoritmo acima não é o único presente na busca de soluções, pois dado uma regra ainda temos várias alternativas de como aplicá-la.

Os passos de 1 a 4 fazem a leitura, a análise léxica e sintática da fórmula. Embora estes passos sejam bastante interessantes (como pode ser visto no apêndice) e façam uso das facilidades oferecidas pela linguagem Prolog a sua discussão foge ao escopo deste trabalho.

Os passos 5, 6 e 7, como mencionado antes, implementam alguns procedimentos que poderiam ser feitos por aplicações de regras, mas para aumentar a eficiência, são implementados de forma determinística. O passo 5 aplica simplificações na fórmula, que consiste em percorrer em pós-ordem a árvores que representa a fórmula; ou seja, a partir da raiz, visitar os nós filhos e depois o nó pai. Para cada visita de um nó,

é feito um percurso em profundidade, aplicando as regras abaixo:

$$A \supset B \Longrightarrow \neg A \vee B \qquad \text{(RS–3.1)}$$
$$\neg\neg A \Longrightarrow A \qquad \text{(RS–3.2)}$$
$$\neg(A \vee B) \Longrightarrow \neg A \wedge \neg B \qquad \text{(RS–3.3)}$$
$$\neg(A \wedge B) \Longrightarrow \neg A \vee \neg B \qquad \text{(RS–3.4)}$$
$$A, (B \vee C) \Longrightarrow A \wedge B \vee A \wedge C \qquad \text{(RS–3.5)}$$
$$false, A \Longrightarrow false \qquad \text{(RS–3.6)}$$
$$false \vee A \Longrightarrow A \qquad \text{(RS–3.7)}$$
$$\neg true \Longrightarrow false \qquad \text{(RS–3.8)}$$
$$\neg false \Longrightarrow true \qquad \text{(RS–3.9)}$$
$$true, A \Longrightarrow A \qquad \text{(RS–3.10)}$$
$$true \vee A \Longrightarrow true \qquad \text{(RS–3.11)}$$
$$\Diamond false \Longrightarrow false \qquad \text{(RS–3.12)}$$
$$\Box false \Longrightarrow false \qquad \text{(RS–3.13)}$$
$$\Diamond true \Longrightarrow true \qquad \text{(RS–3.14)}$$
$$\Box true \Longrightarrow true \qquad \text{(RS–3.15)}$$
$$\neg\Box A \Longrightarrow \Diamond\neg A \qquad \text{(RS–3.16)}$$
$$\neg\Diamond A \Longrightarrow \Box\neg A \qquad \text{(RS–3.17)}$$
$$\neg\forall x A \Longrightarrow \exists x \neg A \qquad \text{(RS–3.18)}$$
$$\neg\exists x A \Longrightarrow \forall x \neg A \qquad \text{(RS–3.19)}$$

Desta forma a fórmula resultante se apresentará como uma disjunção de conjunções e as negações só se aplicarão sobre subfórmulas atômicas.

O passo 6 faz a movimentação dos quantificadores do interior para o exterior das fórmulas. Uma vez que o conectivo de negação só se aplicará neste momento à fórmulas atômicas, os únicos problemas da movimentação de quantificadores são a ordem entre eles e a passagem de dentro do escopo para fora do escopo de um operador modal. Dada as regras de extração de quantificadores:

$$A\langle Q^{\forall} x B[x]\rangle \Longrightarrow \forall x' A\langle B[x']\rangle \qquad \text{(R–3.20)}$$
$$A\langle Q^{\exists} x B[x]\rangle \Longrightarrow \exists x' A\langle B[x']\rangle \qquad \text{(R–3.21)}$$

Poderíamos gerar as seguintes regras para movimentar os quantificadores de dentro para fora do escopo de um operador modal:

$$\Diamond\forall x A\langle x\rangle \Longrightarrow \forall x \Diamond A\langle x\rangle \qquad \text{(RS–3.20)}$$
$$\Diamond\exists x A\langle x\rangle \Longrightarrow \exists x \Diamond A\langle x\rangle \qquad \text{(RS–3.21)}$$
$$\Box\forall x A\langle x\rangle \Longrightarrow \forall x \Box A\langle x\rangle \qquad \text{(RS–3.22)}$$
$$\Box\exists x A\langle x\rangle \Longrightarrow \exists x \Box A\langle x\rangle \qquad \text{(RS–3.23)}$$

Entretanto, a regra R–3.21 não pode ser aplicada em quantificadores existênciais que se encontrem no escopo de um operador de necessidade, o que invalida a regra RS–3.23. Vejamos um exemplo do tipo de erro que a aplicação causaria.

Suponha o modelo $M = \langle D, W, w_0, R, I \rangle$ abaixo:

$$\begin{aligned} D &= \{a, b, c\} \\ W &= \{w_0, w_1, w_2\} \\ R &= \{\langle w_0, w_0\rangle, \langle w_0, w_1\rangle, \langle w_0, w_2\rangle, \langle w_1, w_1\rangle, \langle w_2, w_2\rangle\} \\ I &= \{p^0(a) = true, \\ &\quad\ p^1(b) = true, \\ &\quad\ p^2(c) = true, \\ &\quad\ \text{para os demais valores de } x \text{ e } i,\ p^i(x) = false\} \end{aligned}$$

Podemos verificar que a fórmula $\Box(\exists x p(x))$ é verdadeira para todos mundos do modelo, pois, a partir de cada mundo w_i, para todos os mundos w_j acessados por w_i, sempre existe um elemento x do domínio D tal que $p(x) = true$ em w_j.

Entretanto, $\exists x \Box p(x)$ não é verdadeira em w_0, pois, em w_0, não existe um elemento x de D tal que $p(x)$ seja válido em todos os mundos acessados por w_0.

Durante a movimentação de quantificadores, fórmulas do tipo:

$$\forall x p(x) \vee \exists y p(y)$$

poderiam dar origem a qualquer das fórmulas abaixo:

$$\begin{aligned} &(1)\quad \forall x \exists y (p(x) \vee p(y)) \\ &(2)\quad \exists y \forall x (p(x) \vee p(y)) \end{aligned}$$

Entretanto, da forma que o algoritmo foi implementado, sempre daremos preferência para fórmulas do tipo (2), pois a fórmula (1) é sempre conseqüência lógica da fórmula (2), mas o inverso nem sempre é verificado.

O passo 7 elimina quantificadores desnecessários, por exemplo dada a fórmula:

$$\exists y \forall x (p(x) \wedge \neg q(x) \vee false \wedge q(y))$$

aplicamos a simplificação e obtemos:

$$\exists y \forall x (p(x) \wedge \neg q(x))$$

e então descartamos o quantificador $\exists y$ que não é mais necessário:

$$\forall x (p(x) \wedge \neg q(x))$$

O passo 8 toma uma regra e o passo 9 tenta aplicá-la, ambos fazendo uso de mecanismo de retrocesso oferecido pela linguagem Prolog. Vejamos agora aplicação das regras com detalhes.

Planejamos inicialmente implementar o método para o sistema S5, para isto deveriam ser implementadas as regras abaixo, complementando aquelas regras implementadas na simplificação de fórmulas.

$$\Box A, \Diamond B \longrightarrow \Diamond(A \wedge B) \qquad \text{(RG–3.1)}$$
$$\Box A \longrightarrow A \qquad \text{(RG–3.2)}$$
$$\Box A, \Diamond B \longrightarrow \Diamond(\Box A \wedge B) \qquad \text{(RG–3.3)}$$
$$\Diamond A, \Diamond B \longrightarrow \Diamond(\Diamond A \wedge B) \qquad \text{(RG–3.4)}$$
$$A \longrightarrow \Diamond A \qquad \text{(RG–3.5)}$$
$$Q_a A\langle C, \ldots, C\rangle, Q_b B\langle C, \ldots, C\rangle \longrightarrow \qquad \text{(RG–3.6)}$$
$$Q_{ab}(A\langle true\rangle \vee B\langle false\rangle)$$

Uma vez que a busca de soluções em Prolog é feita em profundidade, com retrocesso, mostrou-se ser indispensável a inclusão de teste de ocorrência. Ou seja, ao aplicarmos uma regras, que gera um novo termo a ser incluído em uma conjunção, devemos verificar se este termo já existe na conjunção. Nas regras que geram um termo a partir de outros dois, também devemos verificar se para estes dois termos anteriores não ocorre de um deles ter sido gerado a partir do outro pela aplicação da mesma regra que estamos tentando usar.

Agora, observemos o que ocorre com aplicação de regra (RG–3.5):

$$S_n = p$$
$$S_{n+1} = p \wedge \Diamond p$$
$$S_{n+2} = p \wedge \Diamond p \wedge \Diamond\Diamond p$$
$$\vdots$$

Como pode ser percebido aplicação da regra RG–3.5, embora esteja sempre gerando um terno novo, diferente de todos os demais, fará o sistema entrar em ciclo infinito. A aplicação desta regra requer um bom grau de "inteligência" para decidir se esta aplicação realmente levará a uma decisão sobre a validade ou não da fórmula sendo analisada. O mesmo problema ocorre com a aplicação da regra RG–3.4.

Pela dificuldade apresentada acima, resolvemos limitar a implemetação ao sistema S4. Observemos agora a aplicação das regras RG–3.1, RG–3.2 e RG–3.3.

Dada a fórmula S_n abaixo, podemos aplicar a regra RG–3.1 e obter a fórmula S_{n+1}.

$$S_n = \Box p \wedge \Diamond \neg p$$
$$S_{n+1} = \Box p \wedge \Diamond \neg p \wedge \Diamond(p \wedge \neg p) \qquad \text{.RG–3.1}$$

Por outro lado, partindo da mesma fórmula S_n, podemos aplicar as regras RG–3.3 e RG–3.2 e obter a fórmula S'_{n+2}. Através da regra R–3.9 da descrição da abordagem, podemos demostrar que a regra S_{n+1} obtida acima é uma conseqüência lógica da regra S'_{n+2} obtida abaixo.

$$
\begin{aligned}
S_n &= \Box p \wedge \Diamond \neg p \\
S'_{n+1} &= \Box p \wedge \Diamond \neg p \wedge \Diamond(\Box p \wedge \neg p) && \text{.RG–3.3} \\
S'_{n+2} &= \Box p \wedge \Diamond \neg p \wedge \Diamond(\Box p \wedge \neg p \wedge p) && \text{.RG–3.2} \\
S'_{n+3} &= \Box p \wedge \Diamond \neg p \wedge \Diamond(\neg p \wedge p) && \text{.R–3.9}
\end{aligned}
$$

Logo, se a implementação incluir as regras RG–3.2 e RG–3.3, não será necessária a implementação da regra RG–3.1. A regra RG–3.1 só precisa ser implementada se estivermos trabalhando com um sistema modal que não inclua a regra RG–3.2 ou a regra RG–3.3, como por exemplo os sistemas K, T e K4.

Deste modo, em nosso protótipo, incluimos apenas as regras RG–3.2, RG–3.3 e RG–3.6.

Regra RG–3.2: $\Box A \longrightarrow A$.

1. Obtenha um conjunção da fórmula.
2. Tome um termo desta conjunção da forma $\Box A$.
3. Verifique que o termo A ainda não pertence à conjunção.
4. Inclua o termo A na conjunção.

Os passos 1 e 2 correspondem as pontos de retrocesso. O passo 3 implementa o teste de ocorrência necessário para esta regra. Os passos da implementação desta regra são bastente simples e podem ser acompanhados sem dificultade através do programa listado no apêndice B.

Regra RG–3.3: $\Box A, \Diamond B \longrightarrow \Diamond(\Box A \wedge B)$.

1. Obtenha uma conjunção da fórmula.
2. Tome um termo desta conjunção da forma $\Box A$.
3. Tome um termo desta conjunção da forma $\Diamond B$.
4. Verifique que o termo $\Diamond(\Box A \wedge B)$ não pertence à conjunção, e que o termo B não foi gerado a partir do termo $\Box A$.

5. Inclua o termo $\Diamond(\Box A \wedge B)$ na conjunção.

Neste algoritmo os passos 1, 2 e 3 são pontos de retrocesso. O passo 4 faz o teste de ocorrência para a regra. O passo 5 inclue o novo termo na regra. Na realidade o teste de ocorrência implementado para esta regra não é suficiente para garantir que a prova termine para qualquer fórmula, entretando ele foi suficiente para o nosso propósito. Uma implementação correta deveria, além dos testes realizados, identificar para cada conjunção, quais os termos que serviram como subfórmulas do tipo $\Box A$ e $\Diamond B$ em uma aplicação anterior da regra RG–3.3, de forma a não permitir e reaplicação da regra com os mesmos A e B.

Regra RG–3.6: $Q_a A\langle C,\ldots,C\rangle, Q_b B\langle C,\ldots,C\rangle \longrightarrow Q_{ab}(A\langle true\rangle \vee B\langle false\rangle)$.

1. Obtenha uma conjunção da fórmula.
2. Obtenha um termo $A\langle C_a\rangle$ da conjunção.
3. Obtenha um termo $B\langle C_b\rangle$ da conjunção, distinto de $A\langle C_a\rangle$.
4. Obtenha uma subfórmula C_a de $A\langle C_a\rangle$
5. Obtenha uma subfórmula C_b de $B\langle C_b\rangle$.
6. Seja θ um unificador mais geral de C_a e C_b.
7. Dado $A\theta\langle C,\ldots,C\rangle$ e $B\theta\langle C,\ldots,C\rangle$, gere termo $(A\theta\langle true\rangle \vee B\theta\langle false\rangle)$.
8. Simplifique o termo.
   (a) O resultado da simplificação não pode ser igual a *true* (se for, ocorrerá um retrocesso).
9. Inclua o novo termo na conjunção.
10. Aplique o unificador a toda a fórmula.
11. Simplifique a nova fórmula.

Os passos de 1 a 5 são pontos de retrocesso desta regra. O passo 6 determina um unificador θ para C_a e C_b. O passo 7 cria o novo termo da forma $(A\theta\langle true\rangle \vee B\theta\langle false\rangle)$. O termo é simplificado no passo 8 e é verificado que ele seja diferente de *true*, caso contrário poderemos incluir um termo *true* na conjunção, que em seguida será removido pelo algoritmo de simplificação, causando um ciclo infinito. O passo

9 inclui o novo termo na conjunção. O passo 10 aplica o unificador na fórmula com o novo termo e o passo 11 a simplifica.

Algumas condições necessárias para garantir a perfeita aplicação da regra RG–3.6 não estão sendo testadas. Vejamos quais as modificações necessárias no algoritmo para garantir cada condição.

(i) As variáveis $x_1, \ldots, x_h, y_1, \ldots, x_k$ são todas diferentes.

A condição (i) é garantida pelo usuário, uma vez que o sistema espera que não sejam usadas duas variáveis com o mesmo nome.

(ii) A seqüência de quantificadores $S_1 z_1 \ldots S_{h+k} z_{h+k}$ é um arranjo das seqüências $Q_1 x_1 \ldots Q_h x_h$ e $R_1 y_1 \ldots R_k y_k$.

Uma vez que os quantificadores são movidos, o tanto quanto possível, para o exterior da fórmula, ocorre que os termos de uma conjunção nunca serão subfórmulas do tipo QxA, garantindo esta condição.

(iii) As subfórmulas $C_1\theta, \ldots, C_m\theta$ não estão no escopo de operadores modais ou contém unicamente símbolos rígidos.

Para implementar a verificação desta condição, temos que modificar o predicado que aplica o unificador para em caso de considerar subfórmulas no escopo de algum operador modal, verificar que os símbolos desta subfórmula sejam todos rígidos depois da aplicação do unificador. Vale notar que, a distinção entre símbolos rígidos e flexíveis deve ser feita pelo usuário, incluindo na base em Prolog cláusulas do predicado `rigido/1` que falhem ou sucedam de acordo com o predicado passado como argumento seja flexível ou rídido.

(iv) As subfórmulas $C_1\theta, \ldots, C_m\theta$ não ocorrem no escopo de algum quantificador dentro de $A\theta \vee B\theta$.

Para implementar a verificação desta condição, temos que modificar o predicado que obtém uma subfórmula de um termo para não considerar subfórmulas no escopo de algum quantificador dentro do termo A ou do termo B.

(v) Se $\langle x \Leftarrow t \rangle \in \theta$, então para i, $1 \leq i \leq h + k$, tal que $S_i = \forall$ e $z_i = x$, não pode ocorrer em t uma variável que esteja ligada em $S_{i+1} z_{i+1} \ldots S_{h+k} z_{h+k}$ $(A \vee B)$. Ou seja, t não pode depender de x.

Para implementar a verificação desta condição temos que testar, para cada termo $\langle x \Leftarrow t \rangle$ do unificador, que, se t possui variáveis livres, então estas variáveis não podem depender de x. Para isto, teremos que percorrer a fórmula original, analisando os escopos dos quantificadores.

(vi) Se $\langle x \Leftarrow t \rangle \in \theta$ e ocorre um símbolo flexível em t então x não pode ocorrer no escopo de nenhum operador modal em A ou B.

Igualmente, para implementar esta verificação, temos que testar para cada termo $\langle x \Leftarrow t \rangle$ do unificador, que, se t possui um símbolo flexível, então não pode haver uma ocorrência de x no escopo de algum operador modal em A ou em B.

Como pode ser visto a implementação da verifiação de todas as condições é realizável, trata-se apenas de um grande tratalho de programação.

III.4 Conclusões Parciais.

A principal contribuição desta abordagem é oferecer uma metodologia ao mesmo tempo genérica e simples para prova de fórmulas da lógica modal. Entretanto, devido à generalidade que esta abordagem oferece para o tratamento em lógica modal, ela apresentará dificuldades análogas aos métodos genéricos para tratamento de fórmulas da lógica de primeira ordem. Dentre estas dificuldades, as mais importantes são o teste de ocorrência de termos e a ordem de busca dentro do espaço de soluções.

Embora, originalmente, seja apontada como vantagem do método o tratamento de fórmulas fora da forma cláusal, na realidade é extremamente conveniente que as fórmula a serem tratadas sejam colocadas em uma forma que se aproxime o máximo possível da forma clausal. Isto permite várias simplificações na implementação de cada regra e torna viável a realização do teste de ocorrência. Vale a pena relembrar que é impossível colocar todas as fórmulas exatamente na forma cláusal por causa da restrição que impede que um quantificador de força existencial passe de dentro para fora do escopo de um operador modal de força de necessidade.

O problema da escolha de qual percurso seguir dentro do espaço de soluções para este método é parcialmente decidível [Casanova 87]. Ou seja, se tomarmos o percurso certo, que obrigatoriamente existirá se a fórmula realmente puder ser refutada, sempre encontraremos a resolução da fórmula sendo tratada. Por outro lado, se não tomarmos o percurso correto, não é garantido que a execução termine em tempo finito.

Em nossa implementação utilizamos o próprio mecanismo de execução da linguagem Prolog para fazer a busca no espaço de soluções. Isto implica em percorremos

o espaço com uma busca em profundidade com retrocesso. Entretanto, dado que o problema é parcialmente decidível, a implementação adequada da abordagem deveria percorrer o espaço com uma busca em largura. Logo, seria mais conveniente o uso de uma linguagem procedimental ou de alguma forma de Prolog paralelo para implementar a busca de soluções, tendo em vista a ineficiência de Prolog na implementação de busca em largura.

Por outro lado, a necessidade do teste de ocorrência obriga que uma implementação definitiva faça um controle muito rígido sobre a representação da fórmula. Esta representação teria que conter informações a respeito de toda sua história de modificações da fórmula ao longo da busca. Isto, em conjunto com o problema de busca, desaconselha o uso de linguagens declarativas.

Também devemos manter em mente que a completude do método só é garantida com a implementação da regra do corte, que tem a sua aplicação guiada por intervenção do usuário.

Capítulo IV

Abordagem 2 - Lógica Temporal - Martín Abadi e Zohar Manna.

Esta abordagem visa a automação de lógica temporal, que é uma lógica modal específica para tratar problemas relacionados com o tempo [Lamport 83]. Ela foi proposta por Martín Abadi e Zohar Manna em 1989 [Abadi 89] e nosso interesse nela é a sua comparação com a abordagem 1, proposta pelos mesmos autores e que permite o tratamento de lógicas modais mais gerais. A abordagem corrente consiste basicamente de uma extensão da linguagem Prolog [Clocksin 87] de forma a aceitar um subconjunto da lógica temporal.

IV.1 Apresentação da Abordagem.

A abordagem apresenta um método de prova para lógica temporal que se baseia em uma resolução-LSD (veja seção A.5) estendida para aceitar alguns operadores temporais. As sentenças obrigatoriamente devem estar na forma de cláusulas de Horn e o domínio deve ser o mesmo para todos os mundos.

IV.1.1 Operadores Temporais.

Na lógica temporal de primeira ordem cada mundo está associado ao instante de tempo que ele representa. É assumida que a variação do instante de tempo é discreta, iniciando em zero e sem limite superior. Teremos um único mundo associado ao instante de tempo zero, este mundo é denominado mundo real. Podemos subdividir a lógica temporal em dois sistemas: lógica temporal linear e lógica temporal ramificada.

No sistema linear temos somente um mundo associado à cada instante de tempo,

logo ao ordenarmos os mundos segundo o instante de tempo associado, cada mundo terá apenas um sucessor (usaremos a palavra sucessor com o sentido de sucessor imediato). A relação de acessibilidade é definida associando cada mundo com ele mesmo e com todos os mundos relacionados a instantes posteriores.

No sistema ramificado podemos ter mais de um mundo associado a um mesmo instante de tempo. Logo, cada mundo pode ter mais de um sucessor. Entretanto, os sucessores de um mundo w_t, associado ao instante t, não são necessariamente sucessores de outros mundos associados ao mesmo instante t. A relação de acessibilidade é definida associando cada mundo com ele mesmo e com os mundos sucessores e é fechada segundo a transividade.

Existem vários operadores para lógica temporais, vejamos dois operadores normalmente usados no sistema ramificado:

$\bigcirc^{\exists}A$ – Próximo Existencial: a fórmula à esquerda é verdadeira em um mundo w, se A for verdadeira em pelo menos um mundo que seja sucessor (imediato) de w.

$\bigcirc^{\forall}A$ – Próximo Universal: a fórmula à esquerda é verdadeira em um mundo w, se A for verdadeira em todos os mundos que são sucessores (imediatos) de w.

Ou seja, ao trabalharmos com o sistema ramificado, a definição dos operadores deve levar em consideração e existência de mais de um sucessor. Como a abordagem se restringe ao sistema linear, usaremos apenas o conjunto de operadores cuja definição e o significado pretendido é dado abaixo. Na seção IV.4 consideraremos a aplicabilidade deste método em sistemas ramificados.

$\bigcirc A$ – Próximo: A fórmula à esquerda é verdadeira em um mundo w, se A for verdadeira no mundo sucessor de w, ou seja, se A for verdadeira no próximo instante de tempo.

$\Box A$ – Sempre: A fórmula à esquerda é verdadeira em um mundo w, se A for verdadeira em todos os mundos acessados por w, ou seja, se A for verdadeira no instante associado à w e em todos os instantes posteriores, até o infinito.

$\Diamond A$ – Eventualmente: A fórmula à esquerda será verdadeira em mundo w, se A verdadeira em pelo menos um mundo acessado por w, ou seja, se A for verdadeira em um instante maior ou igual aquele associado a w.

O operador $\Diamond$ pode ser definido em função do operador $\Box$: $\Diamond A = \neg\Box\neg A$, e vice-versa: $\Box A = \neg\Diamond\neg A$.

Como para modelos da lógica temporal, a relação de acessibilidade é definida relacionando cada mundo com ele mesmo e com os mundos associados a instantes futuros, a definição formal dos operadores sempre ($\Box$) e eventualmente ($\Diamond$) na lógica temporal linear é idêntica a definição dos operadores de necessidade ($\Box$) e de possibilidade ($\Diamond$) na lógica modal. Apenas o significado semântico dos símbolos que muda.

Além destes operadores é usada a representação $\bigcirc^k$, com $k \geq 0$, para indicar uma cadeia de k operadores $\bigcirc$.

A abordagem faz uso da proriedade de distribuição do operador $\bigcirc$ no sistema linear, ou seja:

$$\bigcirc(A \wedge B) \equiv \bigcirc A \wedge \bigcirc B$$

Prova: A subfórmula do lado esquerdo da sentença acima, $\bigcirc(A \wedge B)$, é verdadeira em um mundo w se e somente se $(A \wedge B)$ for verdadeira no mundo w_s, que é sucessor de w. A fórmula $(A \wedge B)$ é verdadeira em w_s, se e somente se A for verdadeira em w_s e B também. Dizer que A e B são verdadeiras em w_s é equivalente a dizer que $\bigcirc A$ e $\bigcirc B$ são verdadeiras em w. E $\bigcirc A$ e $\bigcirc B$ são verdadeiras em w se e somente se $(\bigcirc A \wedge \bigcirc B)$ for verdadeira em w. Logo, a validade da fórmula acima é assegurada no sistema linear.

Podemos ainda provar no sistema linear a validade de fórmulas análogas para os demais conectivos lógicos, além do conectivo de conjunção, de forma similar.

O método também assume que as variáveis não são sensíveis aos operadores temporais; ou seja, a sentença abaixo será sempre válida:

$$\forall x \forall y((x = y) \equiv \bigcirc(x = y))$$

IV.1.2 Sintaxe e Semântica das Cláusulas.

As cláusulas aceitas pelo método são de dois tipos: iniciais ou permanentes. As cláusulas iniciais, denotadas pelo símbolo $\longleftarrow$, representam sentenças válidas apenas no mundo real (ou seja, no instante $t = 0$), enquanto que as cláusulas permanentes, denotadas pelo símbolo $\Longleftarrow$, são válidas em todos os mundos.

Cláusulas iniciais.

notação: $B \longleftarrow A_1, \ldots, A_n$

Estas cláusulas representam as sentenças que podem ser colocadas na forma:

$$\forall x_1 \ldots \forall x_k(A_1 \wedge \ldots \wedge A_n \supset B)^*$$

Cláusulas permanentes.

notação: $B \Longleftarrow A_1, \ldots, A_n$

Estas cláusulas representam as sentenças que podem ser colocadas na forma:

$$\forall x_1 \ldots \forall x_k \Box(A_1 \wedge \ldots \wedge A_n \supset B)^*$$

* Onde $A_1, \ldots, A_n$ e B devem ser termos da forma $\bigcirc^k \alpha$, com $k \geq 0$, e α é uma fórmula atômica; ou seja, α não possue conectivos, operadores ou quantificadores. As variáveis x_1 até x_k são aquelas que ocorrem na fórmula. O termo B é dito a cabeça da cláusula, enquanto que a seqüência $A_1, \ldots, A_n$ é denominada de corpo da cláusula.

IV.1.3 Método de Resolução.

O método é apresentado em dois passos. Inicialmente nos preocupamos apenas com cláusulas que seguem a sintaxe acima, ou seja, cláusulas iniciais ou cláusulas permanentes. Nas seções seguintes o método é estendido de forma a também aceitar cláusulas iniciais com o operador $\boxplus$ na cabeça e cláusulas com o operador $\Diamond$ no corpo. Uma vez que o método de resolução-LSD é correto, examinaremos a correção da abordagem pela constatação da correção de cada regra de inferência analisada.

Dada uma consulta, ou cláusula de Horn objetivo, $C_1, \ldots, C_m$ o sistema considerará a cada instante i, a partir de $i = 0$, a consulta $G_i = \bigcirc^i C_1, \ldots, \bigcirc^i C_m$, dando como resposta a instanciação de variáveis θ_i, que é a composição de todas as unificações necessárias para realizar a resolução. O processo de resolução distingue o uso de cláusulas inicias e cláusulas permanentes.

Cláusulas iniciais.

Dado o objetivo G e a cláusula inicial C abaixo, temos como resolvente um novo objetivo G':

$$G = \bigcirc^{i1} A_1, \bigcirc^{i2} A_2, \ldots, \bigcirc^{ik} A_k$$
$$C = \bigcirc^{i1} B \longleftarrow \bigcirc^{j1} B_1, \ldots, \bigcirc^{jn} B_n$$
$$G' = (\bigcirc^{j1} B_1, \ldots, \bigcirc^{jn} B_n, \bigcirc^{i2} A_2, \ldots, \bigcirc^{ik} A_k)\theta$$

Onde θ é um unificador mais geral entre os termos A_1 e B. As variáveis do objetivo e da cláusulas são renomeadas de forma a evitar colisões. Se os termos A_1 e B só possuírem símbolos rígidos, o número de operadores $\bigcirc$ em cada um pode ser diferente.

É facilmente constatada a correção desta regra de resolução, visto que ela apenas estende a inferência normal da resolução-LSD para termos contendo o operador $\bigcirc$.

O fato de podermos unificar dois termos, formados apenas por símbolos rígidos, que apresentam número diferente de operadores $\bigcirc$, pode ser explicado pela definição de símbolo rígido: Se um predicado é composto apenas por símbolos rígidos, então ele terá a mesma avaliação em todos os mundos, ou seja, em todos os instantes de tempo. Logo, a quantidade de operadores $\bigcirc$ neste caso, que determina em que instante o predicado será avaliado, é irrelevante.

Cláusulas permanentes.

No caso de cláusulas permanentes, o número de operadores $\bigcirc$ no termo A_1 deve ser maior ou igual ao número de operadores $\bigcirc$ no termo B. Dado o objetivo G e a cláusula permanente C abaixo, temos como resolvente um novo objetivo G':

$$G = \bigcirc^{i1}A_1, \bigcirc^{i2}A_2, \ldots, \bigcirc^{ik}A_k$$
$$C = \bigcirc^{j}B \Longleftarrow \bigcirc^{j1}B_1, \ldots, \bigcirc^{jn}B_n$$
$$G' = (\bigcirc^{j1+(i1-j)}B_1, \ldots, \bigcirc^{jn+(i1-j)}B_n, \bigcirc^{i2}A_2, \ldots, \bigcirc^{ik}A_k)\theta$$

Onde $i_1 \geq j$ e θ é um unificador mais geral entre A_1 e B.

A partir da definição de cláusula permanente, temos que ela representa uma fórmula da forma $\forall x_1 \ldots \forall x_n \Box(\alpha \supset \bigcirc^j B)$, que expressa que $\forall x_1 \ldots \forall x_n(\alpha \supset \bigcirc^j B)$ é verdadeira para todos os instantes $t \geq 0$. Em particular, se esta última fórmula é verdadeira no instante $t = i_1 - j$, então temos que a fórmula $\forall x_1 \ldots \forall x_n(\bigcirc^{i1-j}\alpha \supset \bigcirc^{i1}B)$ é verdadeira no instante 0. Ou seja, poderiamos ter a cláusula inicial $\bigcirc^{i1}B \longleftarrow \bigcirc^{i1-j}\alpha$. Logo, dado que a resolução de um objetivo com uma fórmula inicial é correta, a resolução com uma cláusula permanente também é, desde que i_1 seja maior ou igual a j.

Note que nesta abordagem, estaremos sempre movendo operadores modais para dentro e para fora do escopo de quantificadores universais sem o menor problema, pois a validade destas operações foi garantida pela exigência do mesmo domínio para todos os mundos, o que faz com que a fórmula de Barcan e sua reversa sejam válidas.

O uso de cláusulas permanentes com símbolos rígidos na cabeça será vista mais adiante, dado que uma resolução deste tipo deve gerar um resolvente com o operador $\Diamond$.

Vejamos um exemplo da aplicação do método: Vamos usar a lógica temporal para simular a seqüência de Fibonacci, associando a cada instante de tempo um termo da seqüência.

Com as três cláusulas abaixo definimos o predicado $fib(x)$, que a cada instante i assume o valor *true* para o argumento x igual ao i-ésimo termo da seqüência de Fibonacci. A primeira cláusula expressa que no instante $t = 0$, o predicado fib é

verdadeiro para o argumento 0. A segunda cláusula expressa que em $t = 1$, $fib(1)$ é verdadeiro. A terceira cláusula expressa que, em um determinado instante $t + 2$, o predicado fib é verdadeiro para o argumento x, dado que fib seja verdadeiro para o argumento y em $t + 1$, e verdadeiro para argumento z em t e ocorre que $x = y + z$.

$1. fib(0) \longleftarrow$
$2. \bigcirc\, fib(1) \longleftarrow$
$3. \bigcirc\, \bigcirc fib(x) \Longleftarrow fib(y), \bigcirc fib(z), x = y + z$

Mostraremos a resolução para quatro consultas G_k. Especificamente, partiremos do objetivo $\bigcirc^k fib(x)$ para k igual a 0, 1, 2 e 3, desta forma obteremos os quatro primeiros elementos da seqüência de Fibonacci.

Para $k = 0$, temos a consulta $G_0 = fib(x)$:

$4. fib(x)$
$5. \square$ $\quad .1, \{\underline{x = 0}\}$
■

Para $k = 1$, temos a consulta $G_1 = \bigcirc fib(x)$:

$4'. \bigcirc\, fib(x)$
$5'. \square$ $\quad .2, \{\underline{x = 1}\}$
■

Para $k = 2$, temos a consulta $G_2 = \bigcirc\, \bigcirc\, fib(x)$:

$4''. \bigcirc\, \bigcirc fib(x)$
$5''. fib(y), \bigcirc fib(z), x = y + z$ $\quad .3, \{y = 0\}$
$6''. \bigcirc\, fib(z), x = 0 + z$ $\quad .1, \{y = 0, z = 1\}$
$7''. x = 0 + 1$ $\quad .2, \{y = 0, z = 1, x = 1\}$
$8''. \square$ $\quad \{y = 0, z = 1, \underline{x = 1}\}$
■

Para $k = 3$, temos a consulta $G_3 = \bigcirc\, \bigcirc\, \bigcirc fib(x)$:

$4'''. \bigcirc\, \bigcirc\, \bigcirc\, fib(x)$
$5'''. \bigcirc\, fib(y), \bigcirc\, \bigcirc\, fib(z), \bigcirc(x = y + z)$ $\quad .3, \{y = 1\}$
$6'''. \bigcirc\, \bigcirc fib(z), \bigcirc(x = 1 + z)$ $\quad .2, \{y = 1, z = x'\}$
$7'''. fib(y'), \bigcirc fib(z'),$ $\quad .3, \{y = 1, z = x', y' = 0\}$
$\quad x' = y' + z', \bigcirc(x = 1 + x')$

$8'''. \bigcirc fib(z'), x' = 0 + z', \bigcirc(x = 1 + x')$	$.1, \{y = 1, z = x', y' = 0, z' = 1\}$
$9'''. x' = 0 + 1, \bigcirc(x = 1 + x')$	$.2, \{y = 1, z = x', y' = 0,$
	$z' = 1, x' = 1\}$
$10'''. \bigcirc (x = 1 + 1)$	$\{y = 1, z = x', y' = 0,$
	$z' = 1, x' = 1, x = 2\}$
$11'''. \square$	$\{y = 1, z = x', y' = 0,$
	$z' = 1, x' = 1, \underline{x = 2}\}$

■

■

IV.1.4 Estendendo o Método.

Até agora foi apresentado a forma mais simples do método, que tem um bom desempenho de execução. Entretanto, esta forma é bastante limitada. São consideradas duas extensões que não causam um impacto muito grande na velocidade de execução. Embora outras extensões pudessem der incorporadas ao método, os autores alegam que isto traria um custo muito muito grande ao processamento.

IV.1.4.1 Operador □ na Cabeça de Cláusulas Iniciais.

Para aceitar cláusulas iniciais com o operador □ na cabeça é feito o desmembramento destas cláusulas em duas. Estas duas novas cláusulas em conjunto implementam a mesma semântica da cláusula original e o modelo de execução permanece inalterado.

O desmembramento é feito da seguinte forma: No lugar da cláusula inicial com o operador □ na cabeça, introduzimos duas cláusulas, uma nova cláusula inicical e uma cláusula permanente. A nova cláusula inicial, terá a mesma cabeça da cláusula original, mas sem o operador □, e terá como corpo um novo predicado rígido r, e este predicado terá como argumentos todas as variáveis que ocorrem na cabeça desta cláusula. A cláusula permanente, terá a cabeça formada pelo predicado r, com os mesmos argumentos, e terá o corpo igual ao da cláusula inicial original. Sinteticamente:

cláusula original: $\square A \longleftarrow \alpha$

cláusulas desmembradas: $\begin{cases} A \Longleftarrow r(x_1, \ldots, x_n) \\ r(x_1, \ldots, x_n) \longleftarrow \alpha \end{cases}$

Onde r é um novo predicado rígido e $x_1, \ldots, x_n$ são todas as variáveis que ocorrem em A.

Intuitivamente podemos compreender e verificar a correção do desmembramento da seguinte forma: A cláusulas inicial do tipo $\Box A \longleftarrow \alpha$ simboliza que se α pode ser deduzido, então A vale para todo o instante $t \geq 0$. A cláusula inicial $r(\ldots) \longleftarrow \alpha$ simboliza que se α pode ser deduzido então $r(\ldots)$ vale no instante $t = 0$, dado que o predicado r é rígido, se $r(\ldots)$ vale no instante $t = 0$ então vale em qualquer instante de tempo $t \geq 0$. A cláusula $A \Longleftarrow r(\ldots)$ simboliza que se $r(\ldots)$ vale no instante $t = i$, então A também vale neste instante. Ou seja, realmente tanto a cláusula original como as duas geradas expressam a mesma propriedade. Os argumentos de r visam preservar o relacionamento entre α e A.

Vejamos um exemplo do uso de cláusulas iniciais com o operador $\Box$ na cabeça: Suponha que estamos modelando uma arquitetura na qual processos especiais chamados drivers são instalados no instante $t = 1$. Estes processos, uma vez iniciados, permanecem no sistema. Note que se determinado processo é instalado no instante $t = 1$, mesmo no instante $t = 0$ estaremos considerando que ele é um driver. Queremos determinar no instante $t = 5$ se determinado processo é um driver. Logo, podemos usar o seguinte programa:

$$\Box driver(x) \longleftarrow \bigcirc instalado(x)$$
$$\bigcirc instalado(impressora) \longleftarrow$$
$$\bigcirc instalado(disco) \longleftarrow$$

$$\bigcirc^5 driver(impressora)?$$

A primeira cláusula expressa que aqueles processos instalados no instante $t = 1$ serão drivers. As cláusulas seguintes dizem que os processos *impressora* e *disco* são instalados no instante $t = 1$. A consulta tenta determinar se o processo *impressora* é um driver no instante $t = 5$. Logo, teremos as seguintes cláusulas

1. $driver(x_1) \Longleftarrow r(x_1)$
2. $r(x_2) \longleftarrow \bigcirc instalado(x_2)$
3. $\bigcirc\ instalado(impressora) \longleftarrow$
4. $\bigcirc\ instalado(disco) \longleftarrow$

A partir da consulta, teremos:

5. $\bigcirc^5\ driver(impressora)$	
6. $\bigcirc^5\ r(impressora)$	$.1, \{x_1 = impressora\}$
7. $\bigcirc\ instalado(impressora)$	$.2^*, \{x_1 = x_2 = impressora\}$
8. $\Box$	$.3, \{x_1 = x_2 = impressora\}$

■

* Note que, na passagem do passo 6 para o passo 7, unificamos $\bigcirc^5 r(impressora)$ diretamemte com a cabeça da cláusula 2 ($r(x_2)$), porque $r(x)$ é por definição rígido.

Logo, passamos a aceitar três tipos de cláusulas:

$$B \longleftarrow A_1, \ldots, A_n$$
$$\Box B \longleftarrow A_1, \ldots, A_n$$
$$B \Longleftarrow A_1, \ldots, A_n$$

IV.1.4.2 Operador $\Diamond$ no Corpo de Cláusulas.

Até agora aceitamos cláusulas dos três tipos listados acima. Entretanto, o corpo de tais cláusulas só pode ser formado por fórmulas atômicas ou fórmulas atômicas precedidas por operadores $\bigcirc$. Consideraremos agora um extensão que permite aceitar a ocorrência do operador $\Diamond$ no corpo da cláusula. Com isto, cada A_i do corpo de uma cláusula poderá ser uma fórmula atômica ou uma fórmula da forma $\Diamond(\alpha)$, precedidas ou não por operadores $\bigcirc$, e α poderá ser da mesma forma que o corpo de uma cláusula. Ou seja, passaremos aceitar cláusulas com corpos dos seguintes tipos:

$$p, q, r$$
$$\bigcirc p, q, \bigcirc \bigcirc r$$
$$p, \Diamond q, \bigcirc r$$
$$\bigcirc p, \bigcirc \Diamond (q, \bigcirc r, \Diamond (s, t)), \bigcirc \bigcirc s$$

Tendo em mente a estrutura padrão da função de acessibilidade em lógica temporal linear, conseguimos provar a validade das sentenças abaixo, que devem ser usadas para simplificar o corpos das cláusulas.

$$S_1 : \Diamond(\Diamond A_1, \ldots, \Diamond A_n) \equiv \Diamond A1, \ldots, \Diamond A_n$$
$$S_2 : \bigcirc \Diamond A \equiv \Diamond \bigcirc A$$
$$S_3 : \bigcirc(A_1, \ldots, A_n) \equiv \bigcirc A_1, \ldots, \bigcirc A_n$$

Desta forma, trabalhamos com cláusulas cujo corpo obedeçam às seguintes regras de formação:

$$CORPO \rightarrow TERMO$$
$$CORPO \rightarrow TERMO, CORPO$$

$$TERMO \rightarrow \Diamond(CORPO)$$
$$TERMO \rightarrow QATÔMICO$$

$$QATÔMICO \rightarrow \bigcirc^k ATÔMICO$$
$$QATÔMICO \rightarrow ATÔMICO$$

Onde *ATÔMICO* são formulas atômicas. Logo, as cláusulas terão corpos com as seguintes formas:

$$p, q, r$$
$$\bigcirc p, q, \bigcirc \bigcirc r$$
$$p, \Diamond q, \bigcirc r$$
$$\bigcirc p, \Diamond(\bigcirc q, \bigcirc \bigcirc r, \Diamond(\bigcirc s, \bigcirc t)), \bigcirc \bigcirc s$$

A primeira modificação no método de resolução é a mudança da ordem de avaliação dos resolventes. No lugar de considerarmos sempre o predicado mais à esquerda, usaremos o predicado mais à esquerda que seja da forma $\bigcirc^k A$, dentro da conjunção mais à esquerda. O exemplo acima seria resolvido na seguinte ordem:

$$\underline{\bigcirc p}, \Diamond(\bigcirc q, \bigcirc \bigcirc r, \Diamond(\bigcirc s, \bigcirc t)), \bigcirc \bigcirc s$$
$$\Diamond(\bigcirc q, \bigcirc \bigcirc r, \Diamond(\bigcirc s, \bigcirc t)), \underline{\bigcirc \bigcirc s}$$
$$\Diamond(\underline{\bigcirc q}, \bigcirc \bigcirc r, \Diamond(\bigcirc s, \bigcirc t))$$
$$\Diamond(\underline{\bigcirc \bigcirc r}, \Diamond(\bigcirc s, \bigcirc t))$$
$$\Diamond(\Diamond(\underline{\bigcirc s}, \bigcirc t))$$
$$\Diamond(\Diamond(\underline{\bigcirc t}))$$
$$\blacksquare$$

Vejamos porque é necessário esta modificação na ordem de avaliação dos objetivos. Suponha que tenhamos as cláusulas 1 e 2 abaixo e a consulta 3:

1. $\bigcirc \bigcirc \bigcirc p \longleftarrow$
2. $\bigcirc \bigcirc q \longleftarrow$
3. $\Diamond(\Diamond q, p)$

A primeira cláusula expressa que p é verdadeiro no instante $t = 3$, enquanto que a segunda cláusula nos diz que q é verdadeiro no instante $t = 2$. A consulta deseja determinar se, em algum instante t maior ou igual a 0, ocorrerá que p seja verdadeiro e que, em algum instante t' maior ou igual a t, q seja verdadeiro. Podemos notar que pelas cláusulas 1 e 2 não podemos concluir que a consulta seja verdadeira.

Observemos a aplicação do método, primeiro, seguindo a ordem estabelecida acima e segundo as regras apresentadas mais tarde nesta seção.

4. $\bigcirc \bigcirc \bigcirc \Diamond q$ .1

A unificação da cláusula inicial 1 com o termo p, dentro do escopo de um operador $\Diamond$ fará com que todos os outros termos neste escopo seja avançados em três instantes de tempo.

5. $\Diamond(\bigcirc \bigcirc \bigcirc q)$ .pela regra S_2 de simplificação.

Deste ponto não conseguimos prosseguir, chegando a conclusão que o objetivo não pode ser provado.

Agora vejamos a aplicação do método, sem seguir a ordem estabelecida.

$$4'.\ \bigcirc^k p, 0 \leq k \leq 2 \qquad .2$$

Uma vez que a unificação do termo q ocorreu no instante de tempo 2, temos que a subfórmula $\Diamond q$ será verdadeira nos instantes t, tal que $0 \leq t \leq 2$. Logo, como $\Diamond q$ e p estão no escopo do mesmo operador $\Diamond$, p fica limitado a ser unificado dentro deste intervalo. Como pode ser observado, ao não seguir a ordem estabelecida, necessitamos incluir informações auxiliares durante a resolução que dificultarão a implementação automática.

■

Vejamos agora as regras de inferência para o método estendido. Consideraremos objetivos das duas formas:

$$(1) \quad \bigcirc^k A, \gamma$$
$$(2) \quad \Diamond(\bigcirc^k A, \alpha), \gamma$$

Já vimos anteriormente a resolução de objetivos da forma (1) com cláusulas iniciais com a cabeça formada por predicados rígidos e flexíveis e cláusulas permanentes com a cabeça formada por predicados flexíveis. Vamos sintetizar os casos já vistos e nos deter nos demais casos:

1. Objetivos da forma: $G_1 = \bigcirc^k A, \gamma$

   (a) Cláusulas iniciais com cabeça com predicado flexível:
   Dada a cláusula: $C = \bigcirc^j B \longleftarrow \beta$
   Se $k = j$, e sendo θ um unificador mais geral entre A e B, como vimos anteriormente, obtemos o seguinte resolvente:

   $$G' = (\beta, \gamma)\theta$$

   (b) Cláusulas iniciais com cabeça com predicado rígido:
   Dada a cláusula: $C = \bigcirc^j B_r \longleftarrow \beta$
   Independente dos valores de k e j, sendo θ um unificador mais geral entre A e B_r, como vimos anteriormente, obtemos o seguinte resolvente:

   $$G' = (\beta, \gamma)\theta$$

(c) Cláusulas permanentes com cabeça com predicado flexível:

Dada a cláusula: $C = \bigcirc^j B \Longleftarrow \beta$

Dado que $k \geq j$, e sendo θ um unificador mais geral entre A e B, como vimos anteriormente, obtemos o seguinte resolvente:

$$G' = (\bigcirc^{k-j}\beta, \gamma)\theta$$

(d) Cláusulas permanentes com cabeça com predicado rígido:

Dada a cláusula: $C = \bigcirc^j B_r \Longleftarrow \beta$

Relembremos como ocorre a resolução de um objetivo com uma cláusula permanente que tenha a cabeça formada por um predicado flexível para depois estabelecer que vantagens podemos obter se a cabeça for fomada por um predicado rígido. No caso de haver símbolos flexíveis, dado que por definição uma cláusula permanente é vedadeira em qualquer instante de tempo, avançamos a cabeça e o corpo da cláusula até o instante que possibilite a unificação da cabeça com o próximo termo do objetivo. No caso do predicado ser rígido, a unificação da cabeça com o termo do objetivo pode se dar em qualquer instante de tempo. Desta forma, a unificação não fixa o instante de tempo em que o corpo da cláusula deve ser resolvido.

Logo, independente dos valores de k e j, e sendo θ um unificador mais geral entre A e B_r, obtemos o seguinte resolvente.

$$G' = (\Diamond\beta, \gamma)\theta$$

Podemos ver que o resolvente acima obtido expressa exatamente o nosso raciocínio anterior: o corpo da cláusula pode ser resolvido em qualquer instante de tempo.

2. Objetivos da forma: $G_2 = \Diamond(\bigcirc^k A, \alpha), \gamma$

(a) Cláusulas iniciais com cabeça com predicado flexível:

Dada a cláusula: $C = \bigcirc^j B \longleftarrow \beta$

Dado que $j \geq k$, para realizar esta resolução avançamos e fixamos o instante de tempo da subfórmula no escopo de operador $\Diamond$. Logo, consideraremos o seguinte objetivo G, de onde obtemos o resolvente G', visto que θ seja um unificador mais geral entre A e B:

$$G = \bigcirc^j A, \bigcirc^{j-k}\alpha, \gamma$$
$$G' = (\beta, \bigcirc^{j-k}\alpha, \gamma)\theta$$

A correção desta resolução pode ser deduzida da própria definição do operador $\Diamond$, visto que fórmula $\Diamond\pi$ é verdadeira se em algum instante futuro π for verdadeira. Logo, se avançamos a subfórmula π de $j-k$ instantes de tempo e chegamos a conclusão que ela é verdadeira naquele instante, então $\Diamond\pi$ é verdadeira. Visto que podemos avançar e fixar o escopo do operador $\Diamond$ no tempo, a resolução recai no caso visto em 1.a.

(b) Cláusulas iniciais com cabeça com predicado rígido:

Dada a cláusula: $C = \bigcirc^j B_r \longleftarrow \beta$

Dado que o termo B_r terá a mesma avaliação em todos os instantes de tempo, podemos fazer a unificação de B_r com A sem fixar o instante de tempo do escopo do operador $\Diamond$. Logo, independente dos valores de k e j obtemos o resolvente G', onde θ é um unificador mais geral entre A e B_r:

$$G' = (\beta, \Diamond\alpha, \gamma)\theta$$

A correção pode ser facilmente estabelecida, baseado nos raciocínios desenvolvidos nos casos anteriores: Visto que B_r, a cabeça da cláusula usada pela resolução é formado por símbolos rígidos, ela pode ser unificada em qualquer instante de tempo. E esta unificação não fixa o tempo dos demais termos no escopo do operador $\Diamond$.

(c) Cláusulas permanentes com cabeça com predicado flexível:

Dada a cláusula: $C = \bigcirc^j B \Longleftarrow \beta$

Neste caso, iremos obter resultados diferentes dependedo dos valores de k e j. Se $k \geq j$, avançamos a cláusula até o instante k, onde obtemos o resolvente G', dado que θ seja um unificador mais geral entre A e B:

$$G' = (\Diamond(\bigcirc^{k-j}\beta, \alpha), \gamma)\theta$$

No caso de $k < j$, consideramos o objetivo G abaixo, obtendo o resolvente G'', onde θ é um unificador mais geral entre A e B. (Podemos incluir o caso $j = k$ tanto no caso acima como no atual, visto que sob esta condição teremos $k - j = k - k = 0$ e as duas resoluções passam a ser idênticas.)

$$G = \Diamond(\bigcirc^j A, \bigcirc^{j-k}\alpha), \gamma$$
$$G'' = (\Diamond(\beta, \bigcirc^{j-k}\alpha), \gamma)\theta$$

A correção deste caso também é facilmente demonstrada: Tanto uma cláusula permanente como o escopo do operador $\Diamond$ podem ser avançados no tempo para possibilitar uma unificação, e nem um dos dois obrigará a

fixação do tempo do outro. A diferenciação que se faz é unicamente para impedir um número negativo de operadores $\bigcirc$.

(d) Cláusulas permanentes com cabeça com predicado rígido:

Dada a cláusula: $C = \bigcirc^j B_r \Longleftarrow \beta$

Este caso é similar ao anterior, a única diferença é que como o predicado rígido pode ser unificado em qualquer instante, não precisamos avançar nem o corpo da cláusula nem o escopo do operador $\Diamond$. Obtemos o resolvente G', onde θ é um unificador mais geral entre A e B_r.

$$G' = (\Diamond\beta, \Diamond\alpha), \gamma)\theta$$

IV.2 Comentários.

Como pode ser percebido as regras de inferência deste método são bastante concisas e de implementação imediata. Mesmo com relação à efeciência ou completude, durante o nosso estudo, implementação e aplicação, dentro da limitação imposta pela forma das cláusulas aceitas, não pudemos detetar nenhuma falha do método.

O problema de uso de lógica temporal ramificada será visto com detalhes na seção IV.4. O único comentário que nos resta a fazer é que em nenhum ponto da exposição da abordagem os autores comparam a definição de lógica temporal usada, e por nós adotada, com modelos de lógica temporal baseados no conceito de intervalos [Halpern 86]. Chega a ser mencionado na apresentação o uso de operadores como "até" ou "enquanto", mas não é feito nenhuma sugestão de como traduzir estes operadores para os suportados pelo método. Deste modo, limitaremos nosso estudo, baseados na suposição que os operadores $\bigcirc$, $\Box$ e $\Diamond$ (lineares ou ramificados) são suficientes para especificar os modelos nos quais estamos interessados.

IV.3 Implementação.

A abordagem foi apresentada dividida em três etapas: (1) algoritmo básico, (2) inclusão de cláusulas iniciais com o operador $\Box$ na cabeça e (3) inclusão de cláusulas com o operador $\Diamond$ no corpo. A implementação foi desenvolvida em duas versões. Na primeira versão as cláusulas na forma proposta são compiladas para Prolog e a resolução fica por conta do próprio mecanismo de execução da linguagem. Esta versão inclui a primeira e segunda etapa do método. Na segunda versão as cláusulas são armazenadas na mesma forma que aparecem na apresentação do método. Elas sofrem apenas as simplificações que foram definidas na seção IV.1.4.2 e são colocadas em

uma forma padronizada. Já o algoritmo de resolução desta versão é implementado através de um programa Prolog. Esta versão inclui as três etapas da abordagem.

IV.3.1 Versão 1.

Nesta versão, que está listada no apêndice C, ocorre a compilação das cláusulas do método para cláusulas Prolog. As cláusulas aceitas seguem a sintaxe definida pelas regras abaixo:

$$CLÁUSULA \rightarrow \texttt{\#}\ OBJETIVO\ \texttt{<--}\ CORPO$$
$$CLÁUSULA \rightarrow OOBJETIVO\ \texttt{<--}\ CORPO$$
$$CLÁUSULA \rightarrow OOBJETIVO\ \texttt{<==}\ CORPO$$

$$CORPO \rightarrow OOBJETIVO\ ,\ CORPO$$
$$CORPO \rightarrow OOBJETIVO$$
$$CORPO \rightarrow$$

$$OOBJETIVO \rightarrow \texttt{o}\ OOBJETIVO$$
$$OOBJETIVO \rightarrow OBJETIVO$$

Onde $OBJETIVO$ é um predicado com seus argumentos, que podem ser variáveis, constantes ou funções, seguindo a sintaxe normal da linguagem Prolog.

As cláusulas iniciais com o operador $\Box$ na cabeça são desmembradas:

cláusula original:

$$\Box p(ARGUMENTOS) \longleftarrow CORPO$$

cláusulas desmembradas:

$$p(ARGUMENTOS) \Longleftarrow zr(n, [ARGUMENTOS])$$
$$zr(n, [ARGUMENTOS]) \longleftarrow CORPO$$

Diferente da proposta original, o desmembramento usa sempre o mesmo predicado, o zr, de aridade 2. O primeiro argumento deste predicado é um inteiro n, que é incrementado para cada cláusulas deste tipo que é compilada, garantindo, desta forma, o correto relacionamento das cláusulas desmembradas e evitando a geração de vários predicados auxiliares. O segundo argumento é uma lista com os argumentos originais da cabeça da cláusula.

Na compilação das cláusulas, tanto na cabeça como no corpo, os operandores $\bigcirc$ que precedem os predicados são contados e é inserido mais um argumento nos predicados, que corresponde ao instante de tempo em que o predicado é avaliado.

Na compilação de cláusulas iniciais ocorrem dois casos: se o predicado da cabeça não é rígido, ele é traduzido como os demais predicados, e o argumento adicionado é um número igual a quantidade de operadores ○ que precediam o predicado; se o predicado da cabeça é rígido, é adicionado uma variável como argumento, desta forma a cabeça da cláusula poderá ser unificada com predicados associados à qualquer instante de tempo. A compilação de predicados do corpo de cláusulas iniciais se procede incluindo como novo argumento um número igual a quantidade de operadores ○ que precediam o predicado.

Exemplos de compilação de cláusuals iniciais:

comandos de entrada:

```
:- entra((o p(X,Y) <-- q(X,Z), o o r(Y,2,Z))).
rigido(pr(Arg_1, Arg_2)) :- true.
:- entra((pr(constante,VARIAVEL) <-- o p(VARIAVEL,21))).
```

cláusulas Prolog geradas:

```
p(A, B, 1) :-
   q(A, C, 0),
   r(B, 2, C, 2).

pr(constante, A, B) :-
   p(A, 21, 1).

rigido(pr(Arg_1, Arg_2)) :-
   true.
```

A cabeça da primeira cláusula que era o `p(X, Y)` foi traduzida para `p(A, B, 1)`, ou seja, os operadores ○ foram contados e eliminados e foi introduzido um novo argumento no predicado para representar esta quantidade. Os nomes das variáveis foram modificados de `X` e `Y` para `A` e `B` porque o interpretador Prolog que usamos não é capaz de dentro de um programa identificar o nome que o usuário deu a uma variável. (Para falar a verdade os demais interpretadores não fazem isto nunca, mas o que usamos consegue preservar o nome das variáveis de cláusulas digitadas ou lidas de um arquivo, como pode ser visto no predicado `rigido/1`.) Os demais objetivos foram traduzidos segundo o mesmo método. A cláusula `rigido/1`, que foi introduzida junto com as cláusulas do método, visa informar ao sistema que o usuário assume que o predicado `pr/2` é um símbolo rígido.

A compilação de cláusulas permanentes é um pouco mais elaborada. O predicado que ocorre na cabeça da cláusula é tratado como descrito acima, sendo `K` o nome

da variável responsável por armazenar o número de operadores ○ que o precediam. Após o reconhecimento da cabeça, é gerada a nova cabeça e a parte inicial do corpo da cláusula Prolog que representará a cláusula sendo compilada. A geração da nova cabeça é feita incluindo uma variável de nome X como argumento da cabeça original. Como primeiro objetivo do corpo da cláusula Prolog, é incluida uma verificação que só sucede se o instante de tempo que será associado com a cabeça da cláusula (variável X) for maior ou igual ao número de operadores ○ que havia na cabeça original (variável K), ou seja: $X \geq K$.

Cada predicado do corpo de uma cláusula permanente dá origem a dois novos objetivos na cláusula Prolog. O primeiro calcula o instante de tempo em que o predicado original deve ser avaliado, somando a diferença $(X - K)$ ao número de operadores ○ que o precediam, e o segundo tenta avaliar o predicado naquele instante de tempo.

Exemplo de compilação de cláusulas permanentes:

comandos entrados:

```
:- entra((o o fib(F) <== fib(A),o fib(B),soma(A,B,F))).
soma(Parc1,Parc2,Total,_) :- Total is Parc1 + Parc2.
```

cláusulas Prolog gerados:

```
fib(A, B) :-
   B >= 2,
   C is 0 +  (B - 2),
   fib(D, C),
   E is 1 +  (B - 2),
   fib(F, E),
   G is 0 +  (B - 2),
   soma(D, F, A, G).

soma(Parc1,Parc2,Total,_) :-
   Total is Parc1 + Parc2.
```

A compilação aqui é um pouco mais complexa, a cabeça da cláusula (`o o fib(F)`) dá origem à cabeça da cláusula Prolog e ao primeiro objetivo do corpo: `fib(A, B) :- B >= 2,...`; onde a variável A representa a variável original F, B receberá o inteiro que representa o instante de tempo atual (variável X no algoritmo acima), que é comparado através do objetivo `B >= 2`, com a quantidades de operadores ○ (variável K do algoritmo acima) para permitir ou não o uso desta cláusula. Além disto,

o tradutor monta a expressão (B - 2), que será usada na compilação dos demais objetivos.

Todos os outros objetivos dão origem a dois predicados. Por exemplo, o objetivo o fib(A) dá origem ao predicados E is 1 + (B - 2) e fib(F, E), onde a variável F representa a variável original A, e a variável E conterá a avaliação do instante de tempo em que o predicado é avaliado, ou seja, o instante em que a cláusula é avaliada (B - 2) somado ao número de operadores $\bigcirc$ que precediam o predicado original, que neste caso vale 1.

Para realizar uma consulta G, o usuário deve fornecer a lista L de variáveis que ele deseja que sejam impressas ao final de cada iteração. Dada a consulta G e a lista L, é gerada a cláusula $(gol(L) \Longleftarrow G)$ e a partir do instante de tempo $i = 0$, é avaliado o objetivo $\bigcirc^i gol(L)$ e impressa a lista de variáveis L.

Exemplo de execução:

```
?- /* carga do compilador */
>
>  [ tempo0 ].

Sim.
```

Neste ponto carregamos o tradutor, e abaixo, damos entrada ao nosso programa em lógica temporal.

```
?- /* serie de fibonacci */
>
>  /* fib(0) <-- */
>  /* O(1) <-- */
>  /* OOfib(X) <== Ofib(Y),fib(Z),X=Y+Z */
>
>  entra((fib(0) <-- true)).

Sim.

?- entra((o fib(1) <-- true)).

Sim.

?- entra((o o fib(X) <== o fib(Y),fib(Z),X is Y + Z)).
```

```
X = _524462
Y = _524474
Z = _524482 ->

Sim.
```

Depois de carregada nossas cláusula, listamos o código gerado, apenas para ilustração.

```
?- listing( fib ).

fib(0, 0) :-
   true(0).
fib(1, 1) :-
   true(0).
fib(A, B) :-
   B >= 2,
   C is 1 +  (B - 2),
   fib(D, C),
   E is 0 +  (B - 2),
   fib(F, E),
   G is 0 +  (B - 2),
   is(A, D + F, G).

Sim.
```

Agora fazemos a consulta, que nos fornecerá os seguintes resultados:

```
?- /* fib(X) ? */
>
> consulta(fib(X),'X' = X).

Avaliando o objetivo para t = 0
confirme: s
variaveis: X = 0

Avaliando o objetivo para t = 1
confirme: s
```

```
variaveis: X = 1

Avaliando o objetivo para t = 2
confirme: s
variaveis: X = 1

Avaliando o objetivo para t = 3
confirme: s
variaveis: X = 2

Avaliando o objetivo para t = 4
confirme: s
variaveis: X = 3

Avaliando o objetivo para t = 5
confirme: n

Não.

?-
```

IV.3.2 Versão 2.

Esta versão, que está listada no apêndice D, inclue a terceira etapa da abordagem, que permite o uso do operador $\Diamond$ no corpo de cláusulas. Foi mostrado na seção IV.1.4.2. que a ordem de resolução dos termos nesta versão não pode ser simplesmente da esquerda para direita. Por causa disto não pudemos fazer, neste caso, a tradução direta das cláusulas do método para cláusulas Prolog. Em nossa implementação, as cláusulas de lógica temporal são armazenadas em uma forma muito próxima àquela proposta pelo método e existe um conjunto de procedimentos em Prolog que simula um motor de inferência que trabalha com estas cláusulas.

As cláusulas aceitas seguem a sintaxe definida abaixo:

$$CLÁUSULA \rightarrow CABEÇA \texttt{<--} CORPO$$
$$CLÁUSULA \rightarrow CABEÇA \texttt{<==} CORPO$$

$$CABEÇA \rightarrow \texttt{\#}\ OBJETIVO$$
$$CABEÇA \rightarrow OOBJETIVO$$

$$OOBJETIVO \rightarrow \texttt{o}\ OOBJETIVO$$
$$OOBJETIVO \rightarrow OBJETIVO$$

$$CORPO \rightarrow CONJUNÇÃO$$
$$CORPO \rightarrow$$

$$CONJUNÇÃO \rightarrow TERMO\ ,\ CONJUNÇÃO$$
$$CONJUNÇÃO \rightarrow TERMO$$

$$TERMO \rightarrow \texttt{o}\ TERMO$$
$$TERMO \rightarrow \texttt{<>}\ TERMO$$
$$TERMO \rightarrow \texttt{(}\ CONJUNÇÃO\ \texttt{)}$$
$$TERMO \rightarrow OBJETIVO$$

Onde *OBJETIVO* é um predicado com seus argumentos, que podem ser variáveis, constantes ou funções, seguindo a sintaxe normal da linguagem Prolog.

As cláusulas com o operador □ na cabeça são desmembradas da mesma forma que a versão 1. As demais cláusulas, inclusive as desmembradas, são armazenadas através do predicado Prolog `clausula/3`, onde o primeiro argumento é a cabeça, o segundo é o átomo "`<--`" ou o átomo "`<==`", fazendo distinção entre cláusulas iniciais e pernanentes, e o terceiro argumento é uma lista com os objetivos do corpo. A cabeça da cláusula é representada como `K.OBJETIVO`, onde `K` é um inteiro que indica a quantidade de operadores ○ que precedem o predicado cabeça e `OBJETIVO` é o próprio predicado que ocorre na cabeça. Na lista que representa o corpo da cláusula cada objetivo que não está no escopo de um operador ◇ é representado da mesma forma que a cabeça. Já os termo da forma $\Diamond\alpha$, são representados com o predicado Prolog `<>` `(L)`, onde `L` é a lista que representa a conjunção α, definida da mesma forma que a lista que representa o corpo de uma cláusula.

Por exemplo a cláusula:

```
p(X) <-- q(A), o <> (r(X),o s(A))
```

seria entendida como

$$p(X) \longleftarrow q(A), \Diamond(\bigcirc r(X), \bigcirc \bigcirc s(A))$$

e armazenada como

```
clausula(0.p(X), <--, [0.q(A), <> ([1.r(X),2.s(A)])]).
```

Logo, como pode ser acompanhado no apêndice, os procedimentos que transformam a cláusula digitada na entrada para a representação interna também aplicam as regras se simplificação (S_1, S_2, S_3) apresentadas na seção IV.1.4.2.

A execução das inferências é controlada por dois predicados principais: `infere/1` e `einfere/1`. Ambos recebem como argumento uma lista de termos, que representa uma cláusula de Horn objetivo. A cada iteração os predicados unificam o próximo termo da lista com uma das cláusulas possíveis e incluem o corpo da cláusula no restante da lista. Deve ser chamada atenção para o fato de que a cada vez que novos elementos são incluídos na lista ela é reordenada, de forma que a ordem em que os termos estão na lista seja a mesma ordem em que eles devem ser avaliados.

A diferença entre os predicados `infere/1` e `einfere/1`, é que `einfere/1` assume que a lista que é recebida como argumento representa uma conjunção que está no escopo de operadores $\Diamond$. Logo, as regras de inferência que cada um executa é diferente. Vejamos um exemplo de cada um:

Exemplo 1.

Considere a cláusula Prolog abaixo.

```
/* 1 */ infere([K.CABECA|CORPO]) :-
/* 2 */   clausula(J.CABECA, <--, NCORPO),
/* 3 */   J = K,
/* 4 */   ordena(NCORPO, CORPO, FCORPO),
/* 5 */   infere(FCORPO).
```

Esta cláusula implementa a regra de inferência 1.a, resumida abaixo:

$$G_1 = \bigcirc^k A, \gamma$$
$$C = \bigcirc^j B \longleftarrow \beta$$
$$k = j$$
$$G' = (\beta, \gamma)\theta$$

A linha 1 da cláusula recebe a lista passada como argumento, que é desmembrada na forma `K.CABECA` e `CORPO`, onde `K` é a quantidade de operadores $\bigcirc$ que precedem

o predicado `CABECA` e `CORPO` é a lista formada pelos demais objetivos. A linha 2 toma uma cláusula inicial, tal que sua cabeça unifique com o predicado `CABECA`, e a linha 3 verifica se o número de operadores $\bigcirc$ coincide em ambos. A linha 4 chama o predicado `ordena/3`, gerando a lista `FCORPO`, que é o resultado da concatenação ordenada do corpo da cláusula inicial tomada com os objetivos restantes. A linha 5 dá prosseguimento à resolução. Deve ser notado que a aplicação do unificador fica embutida no próprio funcionamento da linguagem Prolog.

■

Exemplo 2.

Considere a cláusula Prolog abaixo.

```
/* 1 */ einfere([K.CABECA|CORPO]) :-
/* 2 */   clausula(J.CABECA, <==, NCORPO),
/* 3 */   J < K,
/* 4 */   D is K - J,
/* 5 */   avanca(NCORPO, D, ACORPO),
/* 6 */   ordena(ACORPO, CORPO, FCORPO),
/* 7 */   einfere(FCORPO).
```

Esta cláusula implementa a regra de inferência 2.c, para o caso $j < k$, resumida abaixo:

$$G_2 = \Diamond(\bigcirc^k A, \alpha), \gamma$$
$$C = \bigcirc^j B \Longleftarrow \beta$$
$$j < k$$
$$G' = (\Diamond(\bigcirc^{k-j}\beta, \alpha), \gamma)\theta$$

Da mesma forma que no exemplo anterior a linha 1 recebe a lista passada como argumento e a linha 2 toma uma cláusula. As únicas diferenças são que será considerado que a lista representa uma conjunção que está no escopo do operador $\Diamond$ e que estamos analisando agora o uso de uma cláusula permanente. A linha 3 verifica a condição `J < K` para a aplicação da regra e a linha 4 calcula a diferença `K - J`, que é atribuída à variável `D`. A linha 5 cria o termo $\bigcirc^{k-j}\beta$, ou seja, cada termo da lista `NCORPO` tem sua quantidade de operadores $\bigcirc$ acrescida de `D`. A linha 6 inclue ordenadamente o corpo modificado da cláusula na lista com os demais objetivos. Note que o prosseguimento da inferência se dá com o uso do predicado `einfere/1`, visto que o resolvente é $\Diamond(\bigcirc^{k-j}\beta, \alpha)$, ou seja, $(\bigcirc^{k-j}\beta, \alpha)$ está no escopo de um operador $\Diamond$.

■

A entrada da cláusula objetivo e a apresentação de resultados é feita da mesma forma que a versão 1. Chamamos apenas a atenção para o fato de nosso sistema procurar apenas uma única solução para cada instante de tempo t. Logo, o programa abaixo:

```
entra((o o o o p(4) <-- )).
entra((o o p(2) <-- )).
consulta((<> p(x)), 'X' = X).
```

dará sempre como resultado (no intervalo $0 \leq t \leq 4$) `X = 4`, pois o Prolog sempre toma as cláusulas na ordem que elas se encontram na base.

Por outro lado o programa abaixo:

```
entra((o o o o p(4) <-- )).
entra((o o p(2) <-- )).
entra((o o o q(3) <-- )).
consulta((<> (p(x), (o q(Y)))), ['X' = X, 'Y' = Y]).
```

funcionará corretamente, devolvendo como resultado `X = 2` e `Y = 3` (no intervalo $0 \leq t \leq 1$), pois embora o sistema tome inicialmente a cláusula `o o o o p(4) <--`, como esta não leva a nenhum resultado o retrocesso causa uma re-execução com a cláusula `o o p(2) <--`.

■

IV.4 Paralelo entre a Aplicabilidade no Sistema Linear e no Ramificado.

Na seção IV.1.1 foi dito que a abordagem faz uso da propriedade de distribuição do operador $\bigcirc$ no sistema linear, ou seja:

$$\bigcirc(A \wedge B) \equiv \bigcirc A \wedge \bigcirc B$$

Na verdade, quase todas as regras de inferência do método estão baseadas no fato que podemos incluir um número constante de operadores $\bigcirc$ em cada termo das regras permanentes ou do objetivo sendo resolvido. Isto faz sentido, pois as regras permanentes representam sentenças que podem ser colocadas na forma:

$$\forall x_1 \ldots \forall x_k \Box(A_1 \wedge \ldots \wedge A_n \supset B)$$

E um objetivo representa uma sentença colocada na forma:

$$\exists x_1 \ldots \exists x_k \bigcirc^t (A_1 \wedge \ldots \wedge A_n)$$

Vejamos inicialmente a aplicabilidade do método para os operadores de lógica temporal ramificada $\bigcirc^\exists$, $\Box^\exists$ e $\Diamond^\forall$. Os operadores $\bigcirc^\exists$ e $\bigcirc^\forall$ foram definidos na seção IV.1.1, os demais definimos como:

$$\Box^\exists \alpha \equiv \alpha \wedge \bigcirc^\exists(\alpha \wedge \bigcirc^\exists(\alpha \wedge \bigcirc^\exists(\alpha \wedge \ldots)))$$
$$\Diamond^\forall \alpha \equiv \alpha \vee \bigcirc^\forall(\alpha \vee \bigcirc^\forall(\alpha \vee \bigcirc^\forall(\alpha \vee \ldots)))$$

Conseguimos modelar três contra-exemplos que provam que o método não é capaz de tratar corretamente os três operadores listados acima.

Contra-exemplo 1: $\bigcirc^\exists$.

Seja R a relação de acessibilidade entre os mundos w_0, associado ao instante 0, e w_{1a} e w_{1b}, associados ao instante 1, tal que:

$$\begin{aligned} R = \{&\langle w_0, w_0\rangle, \\ &\langle w_0, w_{1a}\rangle, \\ &\langle w_0, w_{1b}\rangle, \\ &\langle w_{1a}, w_{1a}\rangle, \\ &\langle w_{1b}, w_{1b}\rangle\} \end{aligned}$$

Seja o predicado p verdadeiro somente em w_{1a} e o predicado q verdadeiro somente em w_{1b}. Logo, temos que

$$\bigcirc^\exists p \wedge \bigcirc^\exists q$$

é verdadeiro em w_0, mas

$$\bigcirc^\exists (p \wedge q)$$

é falso.

Concluímos então que o operador $\bigcirc^\exists$ não apresenta a propriedade de distribuição sobre a conjunção, logo ele não pode ser tratado pelo método proposto pela abordagem.

■

Contra-exemplo 2: $\Box^\exists$

Com a mesma relação de acessibilidade definida acima, dado o predicado r, verdadeiro somente nos mundos w_0 e w_{1a}, temos que

$$\Box^\exists r$$

é verdadeiro em w_0. Como o operador $\Box$ é implementado com o auxílio de um predicado rígido, se o usarmos o método, concluiremos que r será verdadeiro em todos os mundos, demonstrando que o método não é capaz de manipular o operador $\Box^{\exists}$ corretamente.

■

Contra-exemplo 3: $\Diamond^{\forall}$

Tomando a relação de acessibilidade do contra-exemplo 1, e os mesmos predicadps p e q, temos que a sentença

$$\Diamond^{\forall}(p \wedge q)$$

será falsa em w_0. Entretanto, a abordagem concluirá que a sentença é verdadeira, uma vez que não existe mecanismo preparado para distinguir os dois mundos associados ao instante 1.

■

Consideremos agora os operadores $\bigcirc^{\forall}$, $\Box^{\forall}$ e $\Diamond^{\exists}$. Sendo os dois últimos definidos como:

$$\Box^{\forall}\alpha \equiv \alpha \wedge \bigcirc^{\forall}(\alpha \wedge \bigcirc^{\forall}(\alpha \wedge \bigcirc^{\forall}(\alpha \wedge \ldots)))$$
$$\Diamond^{\exists}\alpha \equiv \alpha \vee \bigcirc^{\exists}(\alpha \vee \bigcirc^{\exists}(\alpha \vee \bigcirc^{\forall}(\alpha \vee \ldots)))$$

Qualquer aplicação que seja modelada usando unicamente os três operadores de lógica temporal ramificada acima, pode ser modelada simplesmente com lógica temporal linear.

Concluímos então que a abordagem original não pode ser usada para lógica temporal ramificada. Uma modificação que permita isto, em nossa opinião, será na verdade a definição de uma nova abordagem, tendo em vista que as modificações alterarão os conceitos básicos sobre os quais o método está apoiado.

IV.5 Conclusões Parciais.

Como vimos não podemos usar esta abordagem para lógica temporal ramificada. Se isto fosse possível estaríamos muito próximos de usá-la para o sistema modal S4, uma vez que a transitividade e reflexibilidade da relação de acessibilidade estão presentes nos dois sistema.

Visto que só podemos usar esta abordagem em lógica temporal linear a tentativa de aplicar este método em sistemas da lógica modal (K,T etc) só será possível em modelos do S4 que também apresentem a linearidade da relação de acessibilidade e

tenham o mesmo domínio para todos os mundos. O que nos forneceria um método de resolução apenas para sistemas muito restritos.

Se considerarmos apenas o objetivo de estender a linguagem Prolog, de forma a que ela aceite cláusulas de lógica temporal linear, teremos que concluir que o método não só atende perfeitamente aos seus objetivos, como também que isto é feito de forma extremante eficiente, sendo seu tempo de execução muito razoável quando comparado à execução da linguagem Prolog. Ou seja, a inclusão do método não causa uma grande perda de velocidade de execução no Prolog. Também consideramos que a expressividade da linguagem oferecida pelo método ao usuário seja suficiente para a espeficicação dos modelos de interesse (esquecendo, como mencionado anteriormente, os modelos baseados em intervalos de tempo).

Vale ressaltar ainda uma das conclusões do capítulo anterior, que o requisito da abordagem de se expressar as fórmulas na forma clausal facilita em muito a implementação e proporciona um bom desempenho de execução. Veremos mais tarde que esta condição é necessária mas não é suficiente para garantir um bom desempenho.

Capítulo V

Abordagem 3 - Kurt Konolige.

Esta abordagem foi inicialmente definida por Kurt Konolige em 1986 [Konolige 86]. Da forma como o método foi primeiramente apresentado, ele não poderia ser considerado como uma proposta de resolução automática. O próprio autor define o método como um conjunto de resultados interessantes na área de resolução automática de lógica modal. Entretando, Chistopher Geissler e o próprio Konolige apresentam um trabalho posterior [Geissler 86], onde eles determinam os principais problemas na implementação deste método e definem algumas estratégias de solução destes problemas para o sistema K. Embora nem todas as dificuldades de implementação tenham sido resolvidas e as soluções apresentadas sejam bastante ineficientes, este método se destaca por ser completo e ter servido de base para outros trabalhos posteriores [Quintana 88a].

Com base no exposto acima, nossa apresentação desta abordagem será estruturada de forma bastante distinta das demais. Primeiramente apresentaremos o método como definido inicialmente, nos detendo apenas nos aspectos teóricos. Em seguida, analisaremos a proposta de implementação de Geissler. Por último, nos deteremos nos problemas advindos da tentativa de codificação da proposta apresentada.

V.1 Apresentação da Abordagem.

Esta abordagem introduz um mecanismo que permite colocar as senteças da lógica modal em uma forma análoga à forma clausal, tal que os quantificadores possam ser eliminados sem modificar a satisfatibilidade de cada sentença. Por isto, ao apresentarmos o método não nos preocuparemos com o relacionamento entre quantificadores e operadores modais. Este problema será tratado na seção V.1.7, que apresenta o mecanismo mencionado.

A abordagem permite o tratamento de modelos com domínio diferentes em cada mundo possível, desde que a reversa de Barcan seja satisfeita. A aplicação do método se resume em, dado um conjunto de sentenças, determinar se este conjunto é insatisfatível. Logo, o método pode ser usado para provar que uma sentença qualquer é válida ou para provar que uma sentença é conseqüência lógica de um conjunto de premissas.

Vamos mostrar primeiramente o funcionamento geral do método, para depois analisarmos mais detalhadamente alguns passos específicos e apresentarmos o mecanismo de tratamento de quantificadores.

V.1.1 A Linguagem.

Embora o método aceite qualquer sentença da lógica modal de primeira ordem, estas devem ser colocadas em uma forma que é análoga à forma clausal disjuntiva. Passaremos a nos referenciar a esta forma como forma clausal modal.

(definição 5.1) Definimos literal como uma fórmula atômica, ou uma fórmula atômica negada, ou uma fórmula cujo conectivo principal é o operador modal $\Box$ ou a composição de operadores $\neg\Box\neg$ (ou seja, fórmulas do tipo $\Box\alpha$ ou $\neg\Box\neg\alpha$, onde α é uma fórmula qualquer, onde podem ocorrer inclusive quantificadores, operadores modais etc).

(definição 5.2) Uma sentença da lógica modal estará na forma clausal modal se ela se apresentar como uma conjunção de disjunções de literais, e se cada subfórmula no escopo de operador $\Box$ ou na composição de operadores $\neg\Box\neg$ estiver na forma clausal modal.

O primeiro passo para colocar uma sentença na forma clausal é substituir as ocorrências do conectivo de implicação ($\alpha \supset \beta \equiv \neg\alpha \vee \beta$) e internar as ocorrências do conectivo de negação ($\neg\Box\alpha \equiv \Diamond\neg\alpha$, $\neg\Diamond\alpha \equiv \Box\neg\alpha$, etc). Em seguida substituímos a ocorrência dos operadores $\Diamond$, através da equivalência $\Diamond\alpha \equiv \neg\Box\neg\alpha$. Passamos então a considerar a composição de operadores $\neg\Box\neg$ como um único operador, durante este processo. Feito isto, aplicamos a distribuição dos conectivos de disjunção sobre o de conjunção, eliminamos os quantificadores existenciais por skolemização e removemos os quantificadores universais, tomando a precaução de nunca passar um quantificador de dentro para fora ou de fora para dentro do escopo de um operador modal.

Por exemplo, dada a sentença:

$$\forall x(p(x) \supset \Diamond(p(x) \wedge \forall y q(y)))$$

ela será colocada na forma:

$$\neg p(x) \vee \neg\Box\neg(p(x) \wedge q(y))$$

Note que no exemplo acima que com a simples eliminação dos quantificadores perdemos a informação de que a variável y está quantificada dentro do escopo do operador modal. O mecanismo para evitar que isto ocorra será visto na seção V.1.7.

Uma vez na forma clausal modal, as sentenças são reunidas em um conjunto, tal que cada subfórmula da conjunção é incluída como uma fórmula isolada. Ou seja, no conjunto de sentenças só teremos disjunções de literais.

V.1.2 Notações.

Apresentamos aqui algumas notações e definições que serão usadas ao longo deste capítulo.

Dado um conjunto C de literais, consideramos que ele possa ser particionado em três subconjuntos disjuntos, representados por Σ, $\Box\Gamma$ e $\neg\Box\neg\Delta$ ($C = \Sigma \cup \Box\Gamma \cup \neg\Box\neg\Delta$). O conjunto Σ será o subconjunto de C, formado por todos os literais de C que não possuem operadores modais. O conjunto $\Box\Gamma$ será formado por todos os literais de C da forma $\Box\gamma$, onde γ é uma fórmula qualquer. O conjunto $\neg\Box\neg\Delta$ será formado por todos os literais de C da forma $\neg\Box\neg\delta$, onde δ é uma fórmula qualquer.

Dado o conjunto $\Box\Gamma$, representamos por Γ o conjunto formado por fórmulas γ, tal que $\Box\gamma \in \Box\Gamma$. Dado o conjunto $\neg\Box\neg\Delta$, representamos por Δ o conjunto formado por fórmulas δ, tal que $\neg\Box\neg\delta \in \neg\Box\neg\Delta$.

(definição 5.3) Definimos fórmula ou literal básico como aquele em que não ocorrem variáveis.

(definição 5.4) Definimos grau modal de uma fórmula ou um literal como o maior número de aninhamento de operadores modais. O grau modal de um conjunto será o maior grau modal dentre seus elementos.

(definição 5.5) Definimos grau modal de uma variável x como o número de aninhamento de escopos de operadores modais no qual o quantificador Qx ocorre (ou ocorria antes de ser eliminado).

V.1.3 Funcionamento Geral.

O método usa como principal regra de inferência a regra *"total narrow theory resolution"* de Stickel [Stickel 85], estendida para o tratamento dos operadores modais e denominada B-resolução (B de *"Belief"*). Esta regra permite, dado um conjunto de sentenças na forma clausal disjuntiva que atenda a determinada condição, gerar uma conseqüência lógica destas sentenças ou a cláusula vazia, indicando neste último caso que foi determinado que o conjunto é insatisfatível.

Existe uma outra regra que só é utilizada em sistemas da lógica modal que apresentem a propriedade de reflexividade na relação de acessibilidade (T, S4 e S5). Por enquanto nos basta saber que esta regra também gera uma conseqüência lógica a partir de um conjunto formado por apenas uma sentença que atenda determinada condição.

A aplicação do método consiste basicamente em, dado o conjunto inicial de sentenças, a cada passo tomar um subconjunto de sentenças, gerar por uma das regras de inferência uma conseqüência lógica deste subconjunto e incluí-la no conjunto original. A aplicação prossegue até que tomemos um subconjunto que seja insatisfatível, ou seja, um conjunto para o qual a regra de B-resolução gere a cláusula vazia.

A correção do método é garantida pelos três lemas abaixo:

(lema 5.1) Se S é um subconjunto de C, e S é insatisfatível, então o conjunto C também é insatisfatível.

(lema 5.2) Se S é um subconjunto de C, e α é conseqüência lógica de S, então α também é conseqüência lógica de C (monotonicidade).

(lema 5.3) Se α é conseqüência lógica de C, então C é insatisfatível (ou satisfatível) se e somente se $C \cup \{\alpha\}$ for insatisfatível (ou satisfatível respectivamente). Ou seja, a inclusão de conseqüências lógicas em um conjunto não altera a sua satisfatibilidade.

Veremos adiante que a aplicação da regra de B-resolução necessita de dois métodos auxiliares. O primeiro deve determinar se um conjunto de literais é insatisfatível e o segundo deve determinar se um conjunto de sentenças não modais é insatisfatível. Para o primeiro problema a proposta original apresenta um método de dedução. Estaremos assumindo que o segundo problema pode ser resolvido, por exemplo, com o método de resolução convencional [Robinson 65].

Através dos lemas acima temos a correção do método. O autor mostra qual a idéia que foi usada para demostração de sua completude (refutacional): A insatisfatibilidade do conjunto de premissas é reduzida à insatisfatibilidade de algumas sentenças e a insatisfatibilidade deste subconjunto é reduzida à insatisfatibilidade dos termos da disjunção. Com este raciocínio é montado um passo que é usado por indução para provar a completude do método.

V.1.4 Regra de B-Resolução.

Esta regra pode ser entendida como uma extensão da regra de resolução [Robinson 65] para o caso de considerar mais de duas sentenças ao mesmo tempo.

(teorema 5.1)

(caso genérico) Dado um conjunto C, com n sentenças da forma:

$$\begin{aligned} C = \{&A_1 \vee \alpha_1, \\ &A_2 \vee \alpha_2, \\ &\vdots \\ &A_n \vee \alpha_n\} \end{aligned}$$

Onde cada A_i é um literal e cada α_i é uma disjunção de literais. Se o conjunto de literais $\{A_1, A_2, \ldots, A_n\}$ é insatisfatível, então a sentença $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$ é conseqüência lógica do conjunto de sentenças C.

(caso particular) Dado um conjunto de C, com n sentenças da forma:

$$\begin{aligned} C = \{&A_1 \vee \alpha_1, \\ &A_2 \vee \alpha_2, \\ &\vdots \\ &A_{m-1} \vee \alpha_{m-1}, \\ &A_m, \\ &A_{m+1}, \\ &\vdots, \\ &A_n\} \end{aligned}$$

Onde cada A_i é um literal e cada α_i (com i variando de 1 até $m-1$) é uma disjunção de literais. Se o conjunto de literais $\{A_1, A_2, \ldots A_{m-1}, A_m, A_{m+1}, \ldots, A_n\}$ é insatisfatível, então a sentença $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_{m-1}$ é conseqüência lógica do conjunto de sentenças C.

(caso extremo) Dado um conjunto C, com n sentenças da forma:

$$C = \{A_1,$$
$$A_2,$$
$$\vdots,$$
$$A_n\}$$

Onde cada A_i é um literal. Se o conjunto de literais $\{A_1, A_2, \ldots, A_n\}$ é insatisfatível, então C é insatisfatível é a sentença vazia é conseqüência lógica de C.

■

Com base no teorema acima podemos enunciar a regra de B-resolução:

$$\frac{\begin{array}{c} A_1 \vee \alpha_1 \\ A_2 \vee \alpha_2 \\ \vdots \\ A_n \vee \alpha_n \end{array}}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

Onde $\{A_1, A_2, \ldots, A_n\}$ é um conjunto insatisfatível de literais.

O caso de alguns α_i serem nulos não é problemático. No caso de todos os α_i serem nulos a sentença gerada é a cláusula vazia.

Prova do teorema 5.1 (caso genérico): Se o conjunto de literais $\{A_1, A_2, \ldots, A_n\}$ é insatisfatível então para todos os modelos pelo menos um dos termos do conjunto é avaliado como falso. Seja M um modelo qualquer que satisfaça C, e suponha que o literal A_i seja aquele avaliado como falso neste modelo. Se M satisfaz a C então M satisfaz a todas as sentenças de C. Em particular M satisfaz a sentença $A_i \vee \alpha_i$, logo α_i tem que ser avaliado como verdadeiro no modelo M, visto que A_i é falso. Logo, a sentença $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$ será avaliada como verdadeira em M, visto que ela é uma disjunção que inclui um termo verdadeiro. Provamos então que qualquer modelo M, que satisfaça o antecedente da regra, fatisfaz a sentença gerada pela regra de B-resolução, logo esta sentença é uma conseqüência lógica do conjunto de sentenças antecedente da regra.

No caso de todos os α_i serem nulos, o conjunto de literais se confunde com o conjunto de sentenças, logo, se ele for insatisfatível a prova terá terminado.

Como dissemos antes, para a aplicação desta regra é necessário a existência de um método para determinar se um conjunto de literais é insatisfatível. O caso de o conjunto de literais ser insatisfatível pela lógica clássica não será analisado, estaremos assumindo que será usado um método qualquer neste caso, como por exemplo resolução. O caso de o conjunto de literais ser insatisfatível mas o seu conjunto de literais não modais ser satisfatível será analisado na próxima seção.

V.1.5 Redução.

Estamos agora interessados em um método para determinar se um conjunto de literais é insatisfatível para lógica modal. Um método possível, que é sugerido [Geissler 86], pode ser baseado na redução do problema de determinar se um dado conjunto de literais é insatisfatível no problema de determinar se um outro conjunto de literais, obrigatoriamente de grau modal menor que o anterior, é insatisfatível.

Por exemplo, se estamos trabalhando com modelos que satisfazem ao sistema K, e queremos provar a insatisfatibilidade do conjunto de literais $\Box\Gamma \cup \neg\Box\neg\Delta$, basta provar a insatisfatibilidade de um conjunto da forma $\Gamma \cup \{\delta\}$, onde $\delta \in \Delta$ (e Δ não pode ser vazio).

Embora a definição desta redução seja possível no sistema K, ela não funcionará para sistemas mais complexos. Por isto, é necessário a definição de um método um pouco mais complexo de como gerar uma redução para insatisfatibilidade de conjunto de literais nestes sistemas. Para isto, iremos precisar de dois conceitos novos: árvores de correspondências de profundidade n e k-insatisfatibilidade.

(definição 5.6) Dado um modelo de lógica modal $M = \langle W, w_0, D, R, V, I\rangle$ e um inteiro não negativo n, uma árvore de correspondência de M de profundidade n será um modelo $M' = \langle W', w'_0, D', R', V', I'\rangle$, tal que:

1. A relação de acessibilidade R' tem uma estrutura de árvore, ou seja, cada mundo de $W' - \{w'_0\}$ só é acessado por um único outro mundo de W'.

2. Para cada percurso de tamanho j a partir de w_0 no digrafo formado pela relação R, tal que $j \leq n$, existe um percurso correspondente de tamanho j a partir de w'_0 na árvore formada pela relação R'.

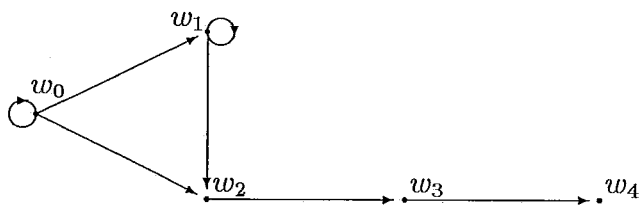
3. Se $w_0, \ldots, w_{j-1}, w_j$ é um percurso no modelo M, tal que $j <= n$, e $w'_0, \ldots$ w'_{j-1}, w'_j é o percurso correspondente no modelo M', então w_j e w'_j tem o mesmo domínio, a mesma atribuição e a mesma interpretação, e $w_0, \ldots, w_{j-1}$ também tem $w'_0, \ldots, w'_{j-1}$ como percurso correspondente.

■

Por exemplo, dado um modelo M, com a relação de acessibilidade R abaixo:

$$\begin{aligned} R = \{&\langle w_0, w_0\rangle, \\ &\langle w_0, w_1\rangle, \\ &\langle w_0, w_2\rangle, \\ &\langle w_1, w_1\rangle, \\ &\langle w_1, w_2\rangle, \\ &\langle w_2, w_3\rangle, \\ &\langle w_3, w_4\rangle\} \end{aligned}$$

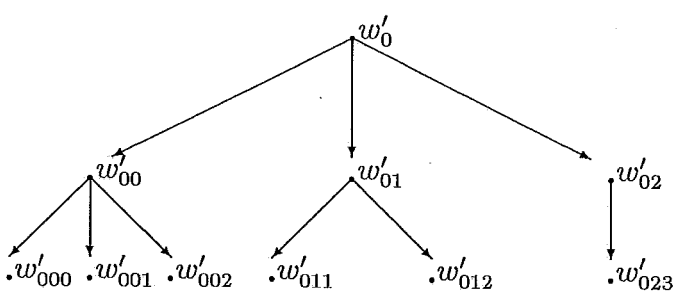
que pode ser representada pelo digrafo:



Uma árvore de correspondência de profundidade 3 de M, seria um modelo M' com a seguinte relação de acessibilidade R':

$$\begin{aligned} R' = \{&\langle w'_0, w'_{00}\rangle, \\ &\langle w'_0, w'_{01}\rangle, \\ &\langle w'_0, w'_{02}\rangle, \\ &\langle w'_{00}, w'_{000}\rangle, \\ &\langle w'_{00}, w'_{001}\rangle, \\ &\langle w'_{00}, w'_{002}\rangle, \\ &\langle w'_{01}, w'_{011}\rangle, \\ &\langle w'_{01}, w'_{012}\rangle, \\ &\langle w'_{02}, w'_{023}\rangle\} \end{aligned}$$

que denota a árvore:



Onde os mundos w'_0, w'_{00} e w'_{000} correspondem ao mundo w_0, os mundos w'_{01}, w'_{001} e w'_{011} correspondem ao mundo w_1, os mundos w'_{02}, w'_{002} e w'_{012} ao mundo w_2, e o mundo w'_{023} corresponde ao mundo w_3.

A árvore de correspondência de profundidade k de um modelo M é na realidade a representação de todos os percursos de tamanho menor ou igual a k que podem ser feitos no digrafo formado pela relação de acessibilidade de M. Logo, qualquer verificação de propriedade de necessite dos percursos de tamanho menor ou igual a k pode ser feita na árvore de correspondência de profundidade k, no lugar de usar o modelo original.

(definição 5.7) Definimos que um conjunto S de sentenças (ou literais) é k-insatisfatível se não existe um modelo M, que possua uma árvore de correspondência M' de profundidade k, tal que o modelo M' satisfaça o conjunto de sentenças S.

Temos que se um conjunto de sentenças S, de grau modal m, é m-insatisfatível, então ele é k-insatisfatível para qualquer $k \geq m$, e em particular ele é insatisfatível. (Observamos que esta propriedade difere da enunciada na apresentação original que é: "Se S é i-insatisfatível então S é j-insatisfatível para $j \geq i$.", visto que esta não é verificada nas fórmulas como $\neg\Box\neg\neg p$, que é 0-insatisfatível mas não é 1-insatisfatível.)

Podemos justificar a propriedade acima de duas formas, ou provando que se um conjunto de grau modal m for $(k+1)$-satisfatível, então ele será k-satisfatível, para $k \geq m$, ou que se um conjunto de grau modal m for satisfatível, então ele será m-satisfatível. Para qualquer uma das formas o raciocínio básico é o mesmo: Dado que o conjunto tem grau modal m, para determinar se ele é satisfeito por um determinado modelo, só precisamos verificar os percursos de tamanho menor ou igual a m no digrafo formado pela relação de acessibilidade. Visto que a árvore de correspondência de profundidade k representa exatamente todos os percursos de tamanho menor ou igual a k, então podemos fazer a verificação usando a árvore no lugar do modelo original.

Se um conjunto de sentenças é insatisfatível pela lógica clássica de primeira ordem, ou seja, se o subconjunto formado pelas sentenças não modais é insatisfatível, então o conjunto é k-insatisfatível para todo $k \geq 0$.

Estamos agora interessados em um método de reduzir o problema de determinar se um conjunto de literais é k-insatisfatível no problema de determinar se outro conjunto de setenças é $(k-1)$-insatisfatível. Com este método, dado um conjunto de literais C_k de grau modal m, reduzimos o problema de determinar se C_k é k-insatisfatível no problema de determinar se um outro conjunto, digamos C_{k-1}, é

$(k-1)$-insatisfatível. Prosseguimos reduzindo para o problema de determinar se C_{k-2} é $(k-2)$-insatisfatível, e assim por diante até encontrarmos um conjunto C_{k-n} que seja insatisfatível pela lógica clássica.

Como C_{k-n} é insatisfatível pela lógica clássica, ele é i-insatisfatível para qualquer i que quizermos. Logo, C_{k-n+1} será $(i+1)$-insatisfatível, e assim por diante até C_k, que será $(i+n)$-insatisfatível. Logo, basta tomar um i tal que $i+n$ seja maior ou igual que m (o grau modal de C_k) e teremos que C_k é insatisfatível. (Note que embora a propriedade de conjuntos k-insatisfatíveis estivesse errada na apresentação original ela não comprometia a correção do método, pois se n for menor que m, podemos tomar i grande o suficiente para garantir a relação $i+n \geq m$.)

(teorema 5.2) Se $C_k = \Sigma \cup \Box\Gamma \cup \neg\Box\neg\Delta$ é um conjunto de literais, satisfatível pela lógica clássica de primeira ordem, onde Σ só contém literais não modais, $\Box\Gamma$ só contém literais da forma $\Box\gamma$ e $\neg\Box\neg\Delta$ só contém literais da forma $\neg\Box\neg\delta$, então C_k é k-insatisfatível se o conjunto de sentenças C_{k-1} definido abaixo for $(k-1)$-insatisfatível. A definição do conjunto C_{k-1} depende do sistema modal, ou seja, das propriedadas apresentadas pela relação de acessibilidade:

- em modelos de K, teremos $C_{k-1} = \Gamma \cup \{\delta\}$
- em modelos de K4, teremos $C_{k-1} = \Gamma \cup \{\delta\} \cup \Box\Gamma$
- em modelos de T, teremos $C_{k-1} = \Gamma \cup \{\delta\}$
  ou $C_{k-1} = \Gamma \cup \Sigma$
- em modelos de S4, teremos $C_{k-1} = \Gamma \cup \{\delta\} \cup \Box\Gamma$
  ou $C_{k-1} = \Gamma \cup \Sigma$
- em modelos de S5, teremos $C_{k-1} = \Gamma \cup \{\delta\} \cup \Box\Gamma \cup \neg\Box\neg\Delta \cup \neg\Box\neg\Sigma$
  ou $C_{k-1} = \Gamma \cup \Sigma$

onde $\delta \in \Delta$.

■

No enunciado o conjunto C_k é considerado satisfatível pela lógica clássica apenas porque se ele já for insatisfatível pela lógica clássica não haveria necessidade de aplicar o método de redução.

Vejamos a idéia por trás deste teorema: Considerando o sistema K, temos que $C_k = \Sigma \cup \Box\Gamma \cup \neg\Box\neg\Delta$ é k-insatisfatível se $C_{k-1} = \Gamma \cup \{\delta\}$ for $(k-1)$-insatisfatível, onde $\delta \in \Delta$. Suponha que exista um modelo M que satisfaça o conjunto de literais C_k em w_0. Dado que o subconjunto $\Box\Gamma$ está contido em C_k e é formado por sentenças

da forma $\Box\gamma$, teremos que o conjunto das sentenças γ, Γ, terá que ser satisfeito pelo modelo M em todos os mundo acessados por w_0. Dado que o subconjunto $\neg\Box\neg\Delta$ está contido em C_k e é formado por sentenças da forma $\neg\Box\neg\delta$, teremos que para cada sentença δ, deve existir pelo menos um mundo acessado por w_0 tal que o conjunto $\Gamma \cup \{\delta\}$ seja satisfeito pelo modelo M.

Agora suponha que exista uma sentença δ, tal que para todo mundo acessado por w_0 o conjunto $\Gamma \cup \{\delta\}$ seja $(k-1)$-insatisfatível ou exista um mundo acessado por w_0 onde o conjunto Γ seja $(k-1)$-insatisfatível. Logo, do ponto de vista do mundo w_0 o conjunto $\Box\Gamma \cup \{\Diamond\delta\}$, ou $\Box\Gamma$ na segunda opção, seria k-insatisfatível, logo o conjunto C_k também seria no mínimo k-insatisfatível.

A redução pode ser justificada nos demais sistemas por raciocínios análogos, só que a complexidade destes acompanha o aumento de complexidade da relação de acessibilidade.

Vejamos um exemplo de aplicação deste teorema, dado o conjunto de literais $C_k = \{\neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$ vamos provar que ele é insatisfatível em modelos que satisfazem ao sistema S5. Aplicando o teorema teremos:

1. O conjunto $C_k = \{\neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$, onde $\Sigma = \{\}$, $\Box\Gamma = \{\}$ e $\neg\Box\neg\Delta = \{\neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$, será k-insatisfatível se o conjunto $C_{k-1} = \{\Box(p \wedge q), \neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$, com $\delta = \Box(p \wedge q)$, for $(k-1)$insatisfatível.

2. O conjunto $C_{k-1} = \{\Box(p \wedge q), \neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$, onde $\Sigma = \{\}$, $\Box\Gamma = \{\Box(p \wedge q)\}$ e $\neg\Box\neg\Delta = \{\neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$, será $(k-1)$-insatisfatível se o conjunto $C_{k-2} = \{p, q, \neg p, \Box(p \wedge q), \neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p, \}$, com $\delta = \neg p$, for $(k-2)$-insatisfatível.

3. O conjunto $C_{k-2} = \{p, q, \neg p, \Box(p \wedge q), \neg\Box\neg\Box(p \wedge q), \neg\Box\neg\neg p\}$ é insatisfatível, pois seu subconjunto $\{p, q, \neg p\}$ o é pelo lógica clássica de primeira ordem. Logo, o conjunto C_k é insatisfatível.

■

Note que se o conjunto de literais incluísse um literal como $\Box(\Box p \vee q)$ o conjunto Γ conteria a fórmula $\Box p \vee q$. Ou seja, o conjunto gerado será, no caso mais genérico, um conjunto de sentenças, e não um conjunto de literais. Isto determinará que teremos que usar aplicações auxiliares do método para podermos validar um passo de inferência na prova principal. Como o método não apresenta uma política de busca de soluções, isto trará graves problemas de implementação.

Vejamos um exemplo do método, como apresentado até agora. Vamos demostrar que a sentença r decorre do conjunto $\{\Box(p \supset q), \Box\neg q, \Box\neg p \supset r)$ no sistema K. Para

isto, demostraremos que o conjunto formado pelas sentenças 1, 2, 3 e 4 abaixo é insatisfatível.

1. $\Box(\neg p \vee q)$
2. $\Box\neg q$
3. $\neg\Box\neg p \vee r$
4. $\neg r$

Tomamos então as sentenças 3 e 4, com o conjunto de literais $\{r, \neg r\}$, insatisfatível pela lógica clássica, e geramos a conseqüência lógica $\neg\Box\neg p$, que é incluída no conjunto como a quinta sentença.

5. $\neg\Box\neg p$

Por último, tomamos as clásusulas 1, 2 e 5, com o conjunto de literais $\{\Box(\neg p \vee q), \Box\neg q, \neg\Box\neg p\}$. Dado que $\Box\Gamma$ é igual a $\{\Box(\neg p \vee q), \Box\neg q\}$ e que $\neg\Box\neg\Delta$ é igual a $\{\neg\Box\neg p\}$, geramos o conjunto $\Gamma \cup \{\delta\}$ como $\{\neg p \vee q, \neg q, p\}$ que é insatisfatível. Logo, geramos a cláusula vazia, terminando a prova.

6. $\Box$

■

V.1.6 Regra para Sistemas com Reflexividade.

Além da regra de B-resolução o método possui uma segunda regra de inferência que pode ser usada em sistemas da lógica modal que apresentem a propriedade de reflexividade da relação de acessibilidade.

(teorema 5.3) Dado um conjunto C, que contenha uma fórmula da forma $\Box\alpha \vee \beta$, onde α é uma subfórmula qualquer e β é uma disjunção de literais (eventualmente nula), então a fórmula $\alpha \vee \beta$ é uma conseqüência lógica do conjunto C.

Com base no teorema acima podemos enunciar a regra para sistemas cuja relação de acessibilidade seja reflexiva.

$$\frac{\Box\alpha \vee \beta}{\alpha \vee \beta}$$

Ou seja, esta regra está generalizando o axioma $\Box\gamma \supset \gamma$ (válido em T, S4 e S5).

V.1.7 O Mecanismo para Quantificadores.

Konolige propôs um mecanismo que permite ocultar os quantificadores, sem prejuízo para a interpretação das sentenças, isto é, mantendo a sua satisfatibilidade. Entretanto, a notação originalmente usada é apropriada apenas para modelos que apresentem a validade da reversa da fórmula de Barcan [Quintana 88a]. Baseado na idéia de Konolige, Quintana redefine o mecanismo com uma nova notação [Quintana 88b], que permite um controle mais refinado do relacionamento entre os domínios.

Para evitar que neste capítulo e no capítulo onde analisamos a abordagem de Quintana apareçam as mesmas idéias com notações diferentes, iremos adotar aqui a notação de Quintana. Entretanto, não eliminaremos a restrição de os modelos terem que satisfazer a reversa de Barcan.

Através deste mecanismo, as variáveis das sentenças serão marcadas, de forma a indicar em que posição (dentro dos escopos dos operadores modais) estavam seus quantificadores, antes de eles serem removidos.

Para a aplicação do mecanismo, as sentenças devem primeiro ser colocadas na forma clausal modal, mas sem eliminar os quantificadores. Em seguida substituímos cada variável x pela representação $**\ldots*x$, onde o número de $*$ é igual ao grau modal da variável. Depois retiramos os quantificadores existenciais através de funções de Skolem. Note que uma variável como $**y$, quantificada existencialmente, dará origem a uma função de Skolem como $**f_s(\ldots)$, ou seja, a skolemização preserva os $*$ da variável. Por último, eliminamos os quantificadores universais.

Por exemplo, dada a sentença:

$$\forall x \Box(p(x) \supset \Diamond(\exists y q(\bar{y}, x)))$$

A sua forma clausal modal será:

$$\forall x \Box(\neg p(x) \vee \neg\Box\neg(\exists y q(y, x)))$$

na qual marcamos as variáveis, obtendo:

$$\forall x \Box(\neg p(x) \vee \neg\Box\neg(\exists y q(**y, x)))$$

para então aplicar a skolemização:

$$\forall x \Box(\neg p(x) \vee \neg\Box\neg q(**f(x), x))$$

e finalmente obtermos a fórmula:

$$\Box(\neg p(x) \vee \neg\Box\neg q(**f(x), x))$$

Com esta marcação temos a possibilidade de reconhecer variáveis, funções e instanciações de variáveis quantificadas em mundos distintos, impedindo assim substituições indesejadas.

V.1.8 Extensão para Primeira Ordem.

As definições das regras de inferência não funcionam corretamente no caso de existirem variáveis e funções marcadas com $*$. Por exemplo, se ocorrer a sentença $\Box\Box p(*x)$ no conjunto C_k, então $\Box\Box p(*x)$ será um dos elementos de $\Box\Gamma$ e $\Box p(*x)$ será um dos elementos de Γ. Logo, o novo conjunto C_{k-1} terá o elemento $\Box p(*x)$, entretanto, como foi retirado o operador modal a variável passou a ficar marcada de forma errada, ou seja, se não retirarmos um $*$ teremos modificado a interpretação da fórmula.

Por outro lado, apenas retirar o $*$ não é suficiente, porque dado o conjunto $C_k = \{\Box\Box p(*x), \Box p(y)\}$, se gerarmos o conjunto $C_{k-1} = \Gamma \cup \Box\Gamma$ removendo o $*$ em Γ teremos $C_{k-1} = \{\Box p(x), p(y), \Box\Box p(*x), \Box p(y)\}$, o que permitirá a unificação de $\Box p(x)$ com $\Box p(y)$ livremente.

De forma a solucionar estes problemas, definimos as três transformações abaixo, que adaptam as tranformações definidas por Konolige para a notação usada por Quintana.

Dado um conjunto C de literais ou sentenças, definimos as transformações C^{*-}, C^{*0} e C^{*+} como:

C^{*-}: Cada termo de C^{*-} é um termo derivado de C, substituindo toda variável ou função da forma $*t$ por t, e as demais variáveis e funções, da forma t, por $n(t)$, onde n é uma função que ainda não ocorria em C. (Note que apenas um $*$ é substituído em cada variável ou função.)

C^{*0}: Cada termo de C^{*0} é um termo derivado de C, substituindo toda variável ou função da forma $*t$ por t, e as demais variáveis e funções, da forma t, por t. (Também apenas um $*$ é eliminado em cada variável ou função.)

C^{*+}: Cada termo de C^{*+} é um termo derivado de C, substituindo toda variável ou função da forma t por $*t$. (Um $*$ é acrescentado em toda variável ou função.)

Com as funções de transformação, redefinimos os conjuntos C_{k-1} da redução como:

- em K, teremos $C_{k-1} = (\Gamma \cup \{\delta\})^{*-}$
- em K4, teremos $C_{k-1} = (\Gamma \cup \{\delta\})^{*-} \cup \Box\Gamma$
- em T, teremos $C_{k-1} = (\Gamma \cup \{\delta\})^{*-}$
  ou $C_{k-1} = (\Gamma \cup \Sigma)^{*0}$

- em S4, teremos $C_{k-1} = (\Gamma \cup \{\delta\})^{*-} \cup \Box\Gamma$
  ou $C_{k-1} = (\Gamma \cup \Sigma)^{*0}$

- em S5, teremos $C_{k-1} = (\Gamma \cup \{\delta\})^{*-} \cup \Box\Gamma \cup \neg\Box\neg\Delta \cup (\neg\Box\Sigma)^{*+}$
  ou $C_{k-1} = (\Gamma \cup \Sigma)^{*0}$

onde $\delta \in \Delta$.

■

E redefinimos a conseqüência lógica gerada pela regra para sistemas com reflexividade como:

$$(\alpha)^{*0} \vee \beta$$

O que estas transformações fazem é simplesmente incrementar ou decrementar o número de $*$ de cada variável das fórmulas, de forma a manter a semântica original de cada fórmula. Por exemplo, dado o conjunto $C = \{\Box\Box p(*x)\}$, a transformação $(\Box C)^{*+} = \{\Box\Box\Box p(**x)\}$, faz com que o escopo de quantificação da variável x continue sendo o primeiro operador modal da fórmula original. Já no caso de termos o conjunto $\Box C = \{\Box\Box p(*x), \Box p(y)\}$, a transformação $(\Box C \cup C)^{*-} = \{\Box\Box p(*x), \Box p(n(y)), \Box p(x), p(n(y))\}$ evidenciará, através da função n, as unificações realizadas entre escopos originalmente diferentes.

Por exemplo, dadas as sentenças 1, 2 e 3 abaixo:

1. $p(a) \vee \Box q(*x)$
2. $\neg q(y)$
3. $\neg p(x)$

Considerando que estamos trabalhando com modelos cujas relações de acessibilidade satisfazem ao sistema S5, podemos usar as sentenças 1 e 2 na regra de B-resolução podendo tomar como C_{k-1} o conjunto de literais $\{q(x), \Box q(*x), \neg\Box\neg q(*y)\}$, ou o conjunto $\{q(x), \neg q(y)\}$. Como o segundo é insatisfatível geramos uma nova sentença:

4. $p(a)$

Neste ponto, reaplicamos a regra de B-resolução com as sentenças 3 e 4, obtendo a sentença vazia.

5. $\Box$

■

V.2 Problemas para Automação.

Em virtude da complexidade envolvida na implementação desta abordagem para sistemas genéricos da lógica modal, são feitas algumas restrições nos modelos aceitos pela proposta de implementação. Desta forma, a partir de agora só estaremos tratando de modelos do sistema K, que apresentem a propriedade do domínio de cada mundo possível ser um subconjunto do domínio de cada mundo acessado (validade da reversa da formulas de Barcan). Embora estas restrições, eventualmente, possam ser contornadas, a proposta de implementação [Geissler 86] afirma que isto seria bastante complexo.

Uma grande dificuldade de implementação é o problema de o método não ter uma política de busca de soluções e necessitar de aplicações recursivas para validar um passo da prova principal. Isto implica em que nem uma busca em largura é suficiente para garantir que a solução seja encontrada. Isto ocorrerá no caso de tomarmos um subconjunto de sentenças tal que um dos conjuntos possíveis de literais não seja insatisfatível.

Outra dificuldade é que as regras apresentadas anteriormente devem ser capazes de manipular sentenças com variáveis livres. Entretanto, embora exista o conceito de unificador mais geral em resolução (com duas sentenças), não é possível a definição de um conceito similar para B-resolução. Por exemplo, dadas as sentenças abaixo:

1. $\Box(p(a) \wedge p(b))$
2. $\neg\Box\neg\neg p(x)$

Dado que $\Box\Gamma$ será $\{\Box(p(a) \wedge p(b))\}$ e que $\neg\Box\neg\Delta$ será $\{\neg\Box\neg\neg p(x)\}$, temos que $C_{k-1} = \Gamma \cup \{\delta\}$ será $\{p(a), p(b), \neg p(x)\}$, que é insatisfatível. Mas existem dois unificadores que poderiam ser utilizados em um prova de insatisfatibilidade deste conjunto: $\{\langle x \leftarrow a\rangle\}$ e $\{\langle x \leftarrow b\rangle\}$. E nem um deles pode ser considerado como "mais geral".

Como será visto mais tarde, a generalização da regra de B-resolução, de forma que ela apresente a completude para modelos de primeira ordem, é um problema bem mais complexo que a determinação de um unificador mais geral.

Outro grave problema de implementação é que a complexidade do método é exponencial, na ordem de $(2^n)^p$, sendo n o número de sentenças no conjunto e p o número de passos necessários para encontrar a solução. Mas mesmo esta avaliação da complexidade é bastante otimista, pois não está levando em consideração que o número de sentenças no conjunto cresce a cada iteração e que cada iteração pode exigir várias provas auxiliares, que por sua vez podem exigir outras provas auxiliares.

Além do problema da complexidade, expresso em função do número de sentenças no banco de dados, a solução para o problema de não existir um quantificador mais geral fará com que a mesma sentença seja usada várias vezes em uma resolução, aumentando ainda mais a largura do espaço de soluções. Por exemplo, dada as sentenças:

$$1.\ q(x) \vee \Box\neg p(*x)$$
$$2.\ \neg\Box\neg(p(*a) \vee p(*b))$$

devemos considerar as instanciações:

$$1_a.\ q(a) \vee \Box\neg p(*a)$$
$$1_b.\ q(b) \vee \Box\neg p(*b)$$
$$2.\ \neg\Box\neg(p(*a) \vee p(*b))$$

de onde obtemos a conseqüência lógica:

$$3.\ q(a) \vee q(b)$$

Ou seja, a sentença $q(x) \vee \Box p(*x)$ é usada duas vezes neste passo, o que faz com que o cálculo da complexidade necessite ser majorado.

V.3 Proposta de Implementação.

Esta proposta visa a implementação do método para o sistema K, de forma a aceitar modelos que apresentem a validade da reversa da fórmula de Barcan. Estaremos supondo a partir de agora que dispomos de recursos computacionais capazes de suportar a execução de um número de processos (programas concorrentes) na ordem de $(2^l)^p$, onde l é o total de literais nas sentenças do conjunto original e p é o número de passos necessários para a prova.

O funcionamento básico do sistema consiste em, 1) dado o conjunto atual de premissas, a cada passo, tomar um subconjunto deste, e a partir deste subconjunto construir um conjunto de literais, tal que tomamos um literal de cada sentença do subconjunto, e 2) provar a insatisfatibilidade do conjunto de literais, gerando uma nova sentença a ser incluida no conjunto de premissas ou gerando a cláusula vazia, terminando a prova[1].

A prova de insatisfatibilidade do conjunto de literais, no caso mais geral, recairá na prova de insatisfatibilidade de outro conjunto de sentenças, logo, usaremos provas

[1]Note que a falta de estratégia leva a uma ineficiência muito grande

auxiliares do próprio método para validar cada passo da prova principal. Como o método não possui nenhuma política de escolha do conjunto de sentenças ou literais que deve ser testado, esta proposta de implementação basicamente define um mecanismo de dedução, onde temos a subdivisão da aplicação da regra de inferência em vários passos. Com isto, devemos tentar todas as alternativas possíveis no espaço de soluções ao mesmo tempo, fazendo o compartilhamento do uso de processador entre os passos dos diversos processos criados para testar cada alternativa.

Note que, uma vez que a prova principal deve criar vários processos para testar cada um dos possíveis subconjuntos de sentenças, cada um destes deve criar vários processos para testar cada possível conjunto de literais, e estes, no caso mais geral, irão aplicar o método recursivamente. O compartilhamento do processador deve ser feito entre os processos dos diversos níveis. Alguma política de prioridade entre os processos dos diversos níveis poderia ser sugerida por um trabalho mais profundo sobre esta abordagem.

V.3.1 Vistas.

De forma a particionar o teste de insatisfatibilidade de cada conjunto de sentenças em passos a proposta sugere um mecanismo chamado vista ("view"). Cabe realçar que este mecanismo provará a insatisfatibilidade para um certo conjunto de literais tomados, logo, devemos ter, para cada conjunto de sentenças, vários destes processos, de forma a garantir a completude do método.

Pelo método de redução apresentado anteriormente, para o sistema K, a determinação da k-insatisfatibilidade de um conjunto de literais $C_k = \Sigma \cup \Box\Gamma \cup \neg\Box\neg\Delta$, é reduzida na determinação da $(k-1)$-insatisfatibilidade do conjunto de sentenças $C_{k-1} = \Gamma \cup \{\delta\}$, onde $\delta \in \Delta$. O mecanismo de vista implementa exatamente a construção e verificação da insatisfatibilidade deste conjunto C_{k-1}. Para isto, serão definidas quatro operações sobre uma vista: abertura, adição de termo, iteração e retorno de uma vista.

A abertura de uma vista corresponderá à inclusão do termo δ no conjunto C_{k-1}, enquanto a adição de termos corresponderá à inclusão dos termos de Γ que forem necessários à prova de insatisfatibilidade. A iteração, que será o processo de inferência dentro de uma vista, será o método convencional de resolução estendido com a possibilidade de abertura recursiva de novas vistas. O fechamento de uma vista ocorre ao conseguirmos provar a insatisfatibilidade do conjunto de literais tomado, e isto se dá com a geração no nível anterior de processamento da fórmulas obtida pela regra de B-resolução com a insatisfatibilidade deste conjunto de literais.

Definimos vista como um conjunto finito de cláusulas anotadas. Onde cada uma destas cláusulas foi gerada a partir de uma cláusula de outro conjunto anterior, e a anotação é um conjunto que para cada cláusula gerada consiste de uma estrutura de dados que contém um trecho da cláusula que lhe deu origem. Ou seja, cada cláusula como $\Box\alpha \vee \beta$ no conjunto anterior poderia dar origem à cláusula $\alpha \vee ans(n, \ldots)$ na vista, com a inclusão do termo $rems(n, \beta)$ na anotação.

O termo *ans* serve para acompanhar os termos incluídos na vista que realmente foram usados para a prova da insatisfatibildade e também marca quais as substituições que as variáveis deste termo sofreram. O termo *rems* serve para registrar a parte restante das fórmulas que deram origem a cada termo. Ou seja, na regra de B-resolução, dado uma sentença da forma $A_i \vee \alpha_i$, construímos um conjunto de literais onde aparece o termo A_i, e uma vez provada a insatisfatibilidade deste conjunto geramos uma fórmula onde ocorre a subfórmula α_i. O termo *rems* justamente registra as subfórmulas α_i. O indexador (n, na descrição acima) entre os termos *ans* e *rems* serve para fazer a correta correlação entre os literais usados e as subfórmulas que farão parte da conseqüência lógica gerada.

Vejamos as operações sobre uma vista:

Abertura de uma Vista: Dado um conjunto com uma cláusula da forma $\neg\Box\neg\alpha \vee \beta$, na forma clausal modal, podemos criar uma vista, incluindo como cláusulas desta vista todas os termos da conjunção gerada ao se colocar a fórmula $((\alpha)^{*-} \vee ans(0, L))$ na forma clausal modal, onde L é o conjunto de variáveis livres da α. Será colocado o termo $rems(0, \beta)$ na anotação.

Adição de um Literal Positivo em uma Vista: Dado um conjunto com uma cláusula da forma $\Box\alpha \vee \beta$, na forma clausal modal, e uma vista aberta anteriormente, podemos adicionar na vista os termos da conjunção gerada ao se colocar a fórmula $((\alpha)^{*-} \vee ans(n, L))$ na forma clausal modal, onde L é o conjunto de variáveis livres de α e n é um novo índice, ainda não usado em uma adição anterior e diferente de 0 (o índice usado na abertura). Será acresentado o termo $rems(n, \beta)$ na anotação.

Iteração em uma Vista: Com as cláusulas de uma vista podemos aplicar a regra convencional de resolução, incluindo a fórmula gerada na vista. Ou então podemos abrir uma nova vista, neste caso as iterações da vista principal passam a ser a aplicações das operações sobre a vista embutida, até que esta seja fechada com o retorno de uma resposta.

Retorno de Resposta de uma Vista: Fechamos uma vista, retornando uma resposta para o processamento anterior quando a última cláusula gerada na vista for composta unicamente por predicados *ans*. Neste momento, incluímos no conjunto anterior, seja ele uma vista anterior ou a prova principal, uma cláusula formada

pela disjunção de termos β'. Sendo que cada um destes termos β' é gerado pela substituição em β das variáveis instanciadas em L por seus valores, onde $rems(n, \beta)$ pertence à anotação e $ans(n, L)$ ocorre na última cláusula gerada na vista atual.

Por exemplo, vejamos como este mecanismo trata aquele nosso caso problemático:

1. $q(x) \vee \Box\neg p(*x)$
2. $\neg\Box\neg(p(*a) \vee p(*b))$

Abrimos então uma vista com o termo $p(a) \vee p(b)$:

1'. $p(a) \vee p(b) \vee ans(0, [])$ anotação:$[rems(0, \Box)]$

Adicionamos agora um novo termo na vista:

2'. $\neg p(x) \vee ans(1, [x])$ anotação:$[rems(0, \Box), rems(1, q(x))]$

Aplicamos a resolução convencional de 1' com 2', obtendo:

3'. $p(b) \vee ans(0, []) \vee ans(1, [a])$ anotação:$[rems(0, \Box), rems(1, q(x))]$

Aplicamos agora a resolução de 3' com 2', obtendo:

4'. $ans(0, []) \vee ans(1, [a]) \vee ans(1, [b])$ anotação:$[rems(0, \Box), rems(1, q(x))]$

Como esta cláusula só contém predicados *ans*, fechamos esta vista, retornando o resultado para o processamento anterior:

3. $q(a) \vee q(b)$

■

Ou seja, o resultado que obtemos com o mecanismo de vista é o mesmo que é obtido ao fazer todas as instanciações necessárias e aplicar a resolução. A vantagem do uso deste mecanismo é que não precisamos "adivinhar" quais as instanciações necessárias e mesmo que incluamos na vista mais sentenças que o necessário, a resposta fornecida só incluirá os termos realmente usados durante o processo de resolução.

Podemos ver que o mecanismo de vista implementa a aplicação da regra de B-resolução em um conjunto de sentenças, para um dado conjunto de literais. Como dissemos antes a implementação precisará criar um processo para cada possível conjunto de sentenças e cada possível conjunto de literais.

V.4 Conclusões Parciais.

O esquema de implementação sugerido requer um processamento paralelo com um número exponencial de processos. Além disto, o modelo de paralelismo requerido não se adequa àqueles tradicionalmente estudados. Isto é, dado um conjunto de sentenças, são criados vários subprocessos para cada possível conjunto de literais. Cada um destes subprocessos poderá recair em três casos distintos: término com prova de insatisfatibilidade, término sem prova de insatisfatibilidade, e não término com os recussos computacionais disponíveis. Os processos que terminaram com a prova da insatisfatibilidade do seu conjunto de literais dão continuidade ao processamento no nível anterior. Ou seja, uma ativação da regra de B-resolução fornecerá vários retornos paralelos, cada um deles sendo processado concomitantemente. Não são mencionados quaisquer mecanismos que realmente evitem a redundância de processamento.

Além da complexidade da implementação, o processo de dedução sugerido a ser implementado dentro de uma vista é o de resolução, limitando não apenas a generalidade do método como também compromentendo a sua completude, tendo em vista de deveremos adotar o uso de cláusulas de Horn em nome da eficiência.

Em suma, mesmo com a proposta de Geissler ainda falta bastante trabalho para transformar as idéias de Konolige em um método de dedução cuja implementação seja eficiente e eficaz. De todas as propostas analisadas esta foi a que mais necessitou, por nossa parte, ser complementada para poder apresentar um aspécto de método voltado à implementação.

Capítulo VI

Abordagem 4 - Pablo de la Quintana.

A abordagem que veremos agora foi proposta por Pablo de la Quintana em 1988 [Quintana 88a,Quintana 88b] e tem como fundamentos alguns trabalhos anteriores [Gabbay 86,Geissler 86,Konolige 86]. Limitaremos a aplicação desta abordagem ao sistema K, tendo em vista que a análise dos outros sistemas seria extremamente longa para o porte deste trabalho. Daremos apenas algumas noções das modificações necessárias para a aplicação do método nos demais sistemas, tomando o sistema K4 como exemplo. Veremos que, mesmo nos limitando ao sistema K, ainda não teremos um método livre de problemas. A pesar dos problemas que apresenta esta abordagem é interessante pela elegância de seu método de dedução.

Este método permitirá o tratamento de modelos com domínios diferentes em cada mundo. É provido um mecanismo para o usuário especificar se existe algum domínio que seja subconjunto de outro.

Neste capítulo, inicialmente apresentaremos a abordagem na forma proposicional e para o sistema K, demonstrando a correção do método e exemplificando seus problemas de completude. Depois, faremos a extenção para lógica de primeira ordem e mostraremos como o método pode ser extendido para outros sistemas modais, tomando o sistema K4 como exemplo. Apresentamos uma proposta de implementação deste método e terminamos o capítulo com algumas conclusões.

VI.1 Apresentação da Abordagem.

A aplicação do método proposto é análogo a uma prova por dedução natural [Manna 74]. As fórmulas tomadas como premissas são expressas em uma linguagem própria, que entre outras modificações, elimina os quantificadores. Estas sentenças, agru-

padas em um conjunto, formam um banco de dados. A fórmula a ser provada, ou gol, também é expressa na mesma linguagem. Uma restrição da abordagem é que, embora possamos incluir no banco de dados qualquer fórmula da lógica modal, o gol só poderá conter variáveis quantificadas existencialmente. Isto não chega a ser uma limitação ou comprometer a aplicação do método, pois as variáveis quantificadas universalmente podem ser simuladas com o uso de regras de generalização.

Assim como as regras da dedução natural se aplicam sobre elementos na forma de um par $(\Gamma \rightarrow \alpha)$, formado por um conjunto de premissas (Γ) e por uma fórmula (α), as regras do método se aplicam sobre triplas da forma $(BD, G_0) \vdash G$, formadas por um conjunto de premissas denominado "banco de dados" (BD), uma fórmula denominada "gol atual" (G) e uma outra fórmula denominada "gol inicial" (G_0). (*Note que a denominação usada aqui de "gol inicial" é apenas para manter a correspondência com a apresentação original da abordagem, a fórmula que representa o gol inicial não corresponde obrigatoriamente à fórmula que desejamos provar.)* Denominamos esta tripla de estado e seu significado pretendido será equivalente ao par de dedução natural $BD \cup \{\neg G_0\} \rightarrow G$.

Como cada regra do método possui uma seqüência de regras de dedução natural equivalente, listaremos na próxima seção as regras de dedução natural, de forma a podermos fazer a equivalência entre um método e o outro. Embora esta equivalência já tenha sido feita no trabalho de Quintana [Quintana 88a], veremos isto aqui de uma forma mais detalhada. Na seção seguinte definiremos a linguagem utilizada para expressar as fórmulas para então apresentar o método.

VI.1.1 Regras da Dedução Natural.

Em uma prova por dedução natural, partimos das premissas, guiados pela forma da fórmula que queremos provar, e a cada aplicação de uma regra geramos um par premissas-fórmula $(\Gamma \rightarrow \alpha)$ que é deduzido a partir dos pares anteriores ou do próprio conjunto de premissas. Em uma dedução pelo método proposto, partimos da tripla que contém as premissas iniciais, e duas instâncias da fórmula que queremos provar $((BD, G_0) \vdash G_0)$, e a cada aplicação de uma regra geramos um conjunto de triplas. Cada tripla deste conjunto sofrerá a aplicação de nova regra até que todas as triplas geradas sejam da forma $(BD_n, G_m) \vdash true$.

Concluímos então que, a aplicação do método e a dedução natural, usarão regras equivalentes, mas aplicadas na ordem inversa. Em vista disto, em vez de usar a

notação usual para dedução natural, da forma:

$$\frac{\langle \text{antecedentes} \rangle}{\langle \text{conseqüentes} \rangle}$$

expressaremos as regras da dedução natural da forma:

$$\langle \text{conseqüentes} \rangle \text{ SE } \langle \text{antecedentes} \rangle$$

evitando assim a confusão que a notação usual de dedução natural causaria com a apresentração desta abordagem.

Vejamos então os axiomas e as regras de dedução natural [Manna 74], Γ será um conjunto de fórmulas e A e B serão duas fórmulas quaisquer:

$\Gamma \cup \{A\} \rightarrow A$	axioma da premissa
$\Gamma \rightarrow true$	axioma do $true$
$\Gamma \rightarrow \neg false$	axioma do $false$
$\Gamma \cup \{A\} \rightarrow B$ SE $\Gamma \rightarrow B$	introdução da premissa
$\Gamma \rightarrow B$ SE $\Gamma \cup \{A\} \rightarrow B$ E $\Gamma \cup \{\neg A\} \rightarrow B$	eliminação da premissa
$\Gamma \rightarrow A \wedge B$ SE $\Gamma \rightarrow A$ E $\Gamma \rightarrow B$	introdução do $\wedge$
$\Gamma \rightarrow A$ SE $\Gamma \rightarrow A \wedge B$	eliminação do $\wedge$
$\Gamma \rightarrow B$ SE $\Gamma \rightarrow A \wedge B$	eliminação do $\wedge$
$\Gamma \rightarrow A \supset B$ SE $\Gamma \cup \{A\} \rightarrow B$	introdução do $\supset$
$\Gamma \rightarrow B$ SE $\Gamma \rightarrow A$ E $\Gamma \rightarrow A \supset B$	eliminação do $\supset$
$\Gamma \rightarrow \neg A$ SE $\Gamma \cup \{A\} \rightarrow B$ E $\Gamma \cup \{A\} \rightarrow \neg B$	introdução do $\neg$
$\Gamma \rightarrow B$ SE $\Gamma \rightarrow A$ E $\Gamma \rightarrow \neg A$	eliminação do $\neg$
$\Gamma \rightarrow A$ SE $\Gamma \rightarrow \neg\neg A$	eliminação do $\neg\neg$

As demais regras de dedução natural não são necessárias para nosso objetivo.

VI.1.2 A Linguagem.

São aceitas como premissas quaisquer sentenças da lógica modal de primeira ordem, entretanto, estas sentenças devem ser reescritas na linguagem que descrevemos aqui. Os estados da forma $(BD_i, G_j) \vdash G_k$ são formados por um conjunto de sentenças (BD_i) e duas sentenças (G_j e G_k) expressas nesta linguagem.

Formalmente as sentenças bem formadas (sbf) da linguagem são definidas da seguinte forma:

1. Toda fórmula atômica (predicado) é uma sbf.

2. O predicado $false$, é uma sbf, mas não é considerado uma fórmula atômica

3. Se α é uma conjunção de sbfs, então $\Box\alpha$ e $\Diamond\alpha$ são sbfs, ditas sbfs modais.

4. Se α é uma conjunção de sbfs e β é uma sbf modal ou uma fórmula atômica, então $\alpha \supset \beta$ é uma sbf.

5. Se α é uma conjunção de sbfs, mas não é uma sbf modal, então $\alpha \supset false$ é uma sbf.

Ou seja, a linguagem só possui os conectivos de implicação e conjunção, sendo que o conectivo de conjunção nunca pode ser o conectivo principal de uma sentença. Além disto, dada uma sentença da forma $\alpha \supset \beta$, se β for o predicado $false$, α não pode ser uma sentença modal, ou seja, não pode ser da forma $\Box\gamma$ ou $\Diamond\gamma$.

Sugerimos aqui uma série de passos que podem ser aplicados para transformar as sentenças da lógica modal de primeira ordem em sentenças da linguagem:

- Passo 1: Inicialmente os quantificadores existenciais das fórmulas a serem colocadas no banco de dado devem ser eliminados através de funções de Skolem. Os quantificadores universais do gol são eliminados através da substituição das suas variáveis por constantes novas (que ainda não ocorrem no banco de dados ou no gol). Assumiremos que os demais quantificadores são tratados como será exposto na seção VI.1.8, o que permitirá que eles sejam eliminados sem perda de significado, isto é, mantendo a satisfatibilidade das sentenças.

- Passo 2: A linguagem não permite fórmulas do tipo $\neg\Box\alpha$ ou $\neg\Diamond\alpha$, logo estas devem ser reescritas através das equivalências:

$$\neg\Box\alpha \equiv \Diamond\neg\alpha$$
$$\neg\Diamond\alpha \equiv \Box\neg\alpha$$

- Passo 3: A linguagem não possue conectivo de negação, logo, fórmulas do tipo $\neg\alpha$ devem ser espressas com o auxílio do predicado $false$.

$$\neg\alpha \equiv \alpha \supset false$$

- Passo 4: Também não é admitido o uso do conectivo de disjunção, que deve ser eliminado pela equivalência:

$$\alpha \vee \beta \equiv (\alpha \supset false) \supset \beta$$

- Passo 5: À direita de um conectivo de implicação só se admite fórmulas atômicas, o predicado $false$, e fórmulas cujo conectivo principal seja modal. Logo, usamos as duas equivalências abaixo para eliminar os demais casos:

$$\alpha \supset (\beta \wedge \gamma) \equiv \alpha \supset \beta \wedge \alpha \supset \gamma$$
$$\alpha \supset (\beta \supset \gamma) \equiv (\alpha \wedge \beta) \supset \gamma$$

- Passo 6: Em fórmulas que tem como conectivo principal o de conjunção, o método trata cada termo da conjunção como uma fórmula isolada. Ou seja, se desejamos incluir a fórmula $\alpha \wedge \beta$ no banco de dados, incluímos a fórmula α e a fórmula β. Quando queremos provar a sentença $\alpha \wedge \beta$ a partir de um banco de dados BD, procuramos provar α e β a partir de BD com seqüências de passos isoladas (na extensão para lógica de primeira ordem esta característica será redefinida).

■

VI.1.3 O Método de Dedução.

Para podermos definir mais claramente o que será considerado como uma prova, precisamos antes definir alguns conceitos auxiliares e apresentar maiores detalhes de outros conceitos já definidos. Um estado será uma tripla, representada por $(BD, G_0) \vdash G$, onde BD é o conjunto de premissas atuais denominado banco de dados, G_0 é uma fórmula denominada de gol inicial e G é uma fórmula denominada o gol atual. Fazendo analogia com dedução natural assumiremos que o significado pretendido do estado $(BD, G_0) \vdash G$ é que podemos deduzir G do conjunto de premissas $BD \cup \{\neg G_0\}$, ou seja, com a notação de dedução natural temos que:

$$(BD, G_0) \vdash G \equiv BD \cup \{\neg G_0\} \rightarrow G$$

A presença do termo $\neg G_0$ ficará claro na apresentação da regra R–6.5. Provamos ao final desta seção que a interpretação dada por nós a um estado é precisamente igual ao significado pretendido pelo autor da abordagem, ou seja, que G é conseqüência lógica de BD, se iniciamos nossa prova com o gol inicial G_0.

Definimos estado inicial como o estado $(BD, G_0) \vdash G_0$, onde BD é o conjunto de premissas inicial e o gol inicial e o gol atual são iguais a fórmula que queremos provar. (Note que o gol inicial só é obrigatoriamente igual a fórmula a ser provada no estado inicial.) Definimos estado terminal como um estado na forma $(BD_i, G_j) \vdash true$.

Definimos estados sucessores de um estado E como aqueles gerados pela aplicação de uma regra qualquer. Note que um estado sucessor poderá ter o banco de dados, o gol inicial ou o gol atual diferentes dos do estado E.

Uma prova será uma árvore (finita) que tem como raiz o estado inicial, as folhas são estados terminais, e todo nó pai tem exatamente como nós filhos, os estados sucessores gerados pela aplicação de uma regra qualquer. Iremos observar que a aplicação de determinadas regras exige o uso do próprio método para realizar provas

auxiliares, logo, poderemos ter árvores de provas auxiliares associadas a alguns nós pais.

Consideraremos o estado $E = (BD_i, G_j) \vdash G_k$ como verdadeiro, ou seja, dada uma regra da forma "Se E faça A", consideraremos o seu antecedente satisfeito, se existe uma prova para o estado E.

Uma regra da forma:

Se o estado atual é $(BD_i, G_i) \vdash G_j$ então gere os estados sucessores
$(DB_{i_1}, G_{i_1}) \vdash G_{j_1} \ldots (BD_{i_n}, G_{i_n}) \vdash G_{j_n}$

será traduzida para a regra de dedução natural:

$$BD_i \cup \{\neg G_i\} \rightarrow G_j \text{ SE } DB_{i_1} \cup \{\neg G_{i_1}\} \rightarrow G_{j_1} \text{ E } \ldots \text{ E } BD_{i_n} \cup \{\neg G_{i_n}\} \rightarrow G_{j_n}$$

Note que o estado inicial $((BD, G_0) \vdash G_0)$ em notação de dedução natural representa $BD \cup \{\neg G_0\} \rightarrow G_0$ (ou melhor $BD \rightarrow G_0$) e as folhas da ávore de prova $((BD_i, G_j) \vdash true)$ representam $BD_i \cup \{\neg G_j\} \rightarrow true$ que é uma tautologia. Assim se percorrermos a ávore gerada pela aplicação do método das folhas para a raiz, teremos uma prova em dedução natural para $BD \rightarrow G_0$.

A escolha da regra a ser aplicada é feita pelo conectivo principal do gol atual. Para fórmulas atômicas temos várias regras, conforme veremos adiante.

- Regra para conjunção (R–6.$\wedge$): Dado que o estado atual é da forma $(BD, G_0) \vdash \alpha \wedge \beta$, gere os seguintes estados sucessores: $(BD, G_0) \vdash \alpha$ e $(BD, G_0) \vdash \beta$.

Com a notação de dedução natural esta regra expressa:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \wedge \beta \text{ SE } BD \cup \{\neg G_0\} \rightarrow \alpha \text{ E } BD \cup \{\neg G_0\} \rightarrow \beta$$

Ou seja, a regra é exatamente a regra de introdução do $\wedge$ em dedução natural. Dado que o método de dedução natural é correto concluímos que esta regra é correta.

- Regra para implicação (R–6.$\supset$): Dado que o estado atual é da forma $(BD, G_0) \vdash \alpha \supset \beta$, gere o seguinte estado sucessor: $(BD \cup \{\alpha\}, G_0) \vdash \beta$. (Se α é da forma $\gamma_1 \wedge \gamma_2 \wedge \ldots \wedge \gamma_n$ então o novo banco de dados será $BD \cup \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$.

Reescrevendo a regra com a notação de dedução natural, temos

$$BD \cup \{\neg G_0\} \rightarrow \alpha \supset \beta \text{ SE } BD \cup \{\neg G_0\} \cup \{\alpha\} \rightarrow \beta$$

que é exatamente a regra de introdução do $\supset$. As regras de dedução natural de eliminação do $\wedge$ e de introdução do $\wedge$ nos garantem a correção de introduzirmos os termos de uma conjunção isoladamente no banco de dados.

- Regras para fórmulas atômicas e o predicado $false$: Dado que o estado atual é da forma $(BD, G_0) \vdash \alpha$, onde α é uma fórmula atômica ou $false$, gere o estado sucessor de acordo com um dos seguintes casos:

  R–6.1. Se $\alpha \in BD$, então gere um estado sucessor terminal: $(BD, G_0) \vdash true$.

  R–6.2. Se $false \in BD$, então gere um estado sucessor terminal: $(BD, G_0) \vdash true$.

  R–6.3. Se $(\beta \supset \alpha) \in BD$, então gere o estado $(BD, G_0) \vdash \beta$.

  R–6.4. Se $(\beta \supset false) \in BD$, então gere o estado $(BD, G_0) \vdash \beta$.

  R–6.5. Gere o estado $(BD, G_0) \vdash G_0$.

Estas regras não são determinísticas, podemos ter várias alternativas de estados sucessores, para um dado estado atual. Façamos analogia de cada caso com dedução natural para verificar a correção desta regra.

A regra R–6.1 expressa:

$$(BD, G_0) \vdash \alpha \quad \text{SE } \alpha \in BD$$

O fato de α ser atômico não afetará nossa análise em nenhum dos casos.

Podemos reescrever a regra acima como:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \quad \text{SE } \alpha \in BD$$

que é equivalente ao axioma da premissa.

A regra R–6.2 expressa:

$$(BD, G_0) \vdash \alpha \quad \text{SE } false \in BD$$

Podemos reescrever a regra acima como:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \quad \text{SE } false \in BD$$

A regra acima não faz parte das regras usuais de dedução natural, mas pode facilmente ser deduzida delas:

1. $\Gamma \cup \{false\} \rightarrow false$ .axioma da premissa
2. $\Gamma \cup \{false\} \rightarrow \neg false$ .axioma do $false$
3. $\Gamma \cup \{false\} \rightarrow \alpha$ .eliminação do $\neg$ com 1 e 2

■

Como não assumimos nenhuma característica especial para Γ ou α, podemos escrever $\Gamma \cup \{false\} \rightarrow \alpha$ como axioma, ou:

$$\Gamma \rightarrow \alpha \text{ SE } false \in \Gamma \qquad \text{(regra derivada 1)}$$

como uma regra derivada de dedução natural, que é exatamente igual a regra R–6.2 reescrita.

A regra R–6.3 expressa:

$$(BD, G_0) \vdash \alpha \text{ SE } (\beta \supset \alpha) \in BD \text{ E } (BD, G_0) \vdash \beta$$

Podemos reescrever esta regra como:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \text{ SE } (\beta \supset \alpha) \in BD \text{ E } BD \cup \{\neg G_0\} \rightarrow \beta$$

Como pelo axioma da premissa temos que:

$$\Gamma \cup \{\beta \supset \alpha\} \rightarrow \beta \supset \alpha$$

então a regra em questão pode ser reduzida à regra de eliminação do $\supset$.

A regra R–6.4 expressa:

$$(BD, G_0) \vdash \alpha \text{ SE } (\beta \supset false) \in BD \text{ E } (BD, G_0) \vdash \beta.$$

Podemos reescrever a regra como:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \text{ SE } (\beta \supset false) \in BD \text{ E } BD \cup \{\neg G_0\} \rightarrow \beta$$

Vejamos se podemos derivar a regra acima das regras de dedução natural. Assumindo como antecedente $\Gamma \cup \{\beta \supset false\} \rightarrow \beta$, temos:

1. $\Gamma \cup \{\beta \supset false\} \rightarrow \beta$	.antecedente
2. $\Gamma \cup \{\beta \supset false\} \rightarrow \beta \supset false$	.axioma da premissa
3. $\Gamma \cup \{\beta \supset false\} \rightarrow false$	.eliminação do $\supset$ com 1 e 2
4. $\Gamma \cup \{\beta \supset false\} \rightarrow \neg false$	.axioma do $false$
5. $\Gamma \cup \{\beta \supset false\} \rightarrow \alpha$	.eliminação do $\neg$ com 3 e 4

■

Evidenciando o antecedente temos:

$$\Gamma \cup \{\beta \supset false\} \rightarrow \alpha \text{ SE } \Gamma \cup \{\beta \supset false\} \rightarrow \beta \qquad \text{(regra derivada 2)}$$

Logo, a regra R–6.4 reescrita é equivalente a regra derivada 2 de dedução natural.

A regra R–6.5 expressa:

$$(BD, G_0) \vdash \alpha \text{ SE } (BD, G_0) \vdash G_0,$$

que pode ser reescrita como:

$$BD \cup \{\neg G_0\} \rightarrow \alpha \text{ SE } BD \cup \{\neg G_0\} \rightarrow G_0$$

Usando o axioma da premissa como

$$DB \cup \{\neg G_0\} \rightarrow \neg G_0$$

podemos reduzir a regra acima à regra da eliminação do $\neg$ da dedução natural:

1. $\Gamma \cup \{\neg G_0\} \rightarrow G_0$	.antecedente
2. $\Gamma \cup \{\neg G_0\} \rightarrow \neg G_0$	.axioma da premissa
3. $\Gamma \cup \{\neg G_0\} \rightarrow \alpha$	.eliminação de $\neg$ com 1 e 2

■

Evidenciando o antecedente:

$$\Gamma \cup \{\neg G_0\} \rightarrow \alpha \text{ SE } \Gamma \cup \{\neg G_0\} \rightarrow G_0$$

■

Com isto demonstramos que as regras do método para lógica proposicional são corretas. Com a inclusão de regras de instanciação temos um método correto para lógica de primeira ordem. Gabbay demonstrou que as regras apresentadas também são completas em relação à lógica de primeira ordem [Gabbay 84,Gabbay 85].

Ficou faltando provar que a interpretação usada por nós é correta e completa, ou seja, se demonstrar $BD \cup \{\neg G_0\} \rightarrow G_0$ significa demonstrar que G_0 é conseqüência lógica de BD. Isto é, temos que provar a correção das regras:

$$\Gamma \rightarrow G_0 \text{ SE } \Gamma \cup \{\neg G_0\} \rightarrow G_0$$
$$\Gamma \cup \{\neg G_0\} \rightarrow G_0 \text{ SE } \Gamma \rightarrow G_0$$

A segunda fórmula é exatamente a regra de introdução da premissa de dedução natural. Vamos então usar dedução natural para demonstrar a primeira fórmula. Assumindo $\Gamma \cup \{\neg G_0\} \rightarrow G_0$ como antecedente:

1. $\Gamma \cup \{\neg G_0\} \rightarrow G_0$	.antecedente
2. $\Gamma \cup \{\neg G_0\} \rightarrow \neg G_0$	.axioma da premissa
3. $\Gamma \rightarrow \neg\neg G_0$	.introdução do $\neg$ com 1 e 2
4. $\Gamma \rightarrow G_0$	.eliminação do $\neg\neg$ com 3

■

Evidenciando o antecedente:

$$\Gamma \rightarrow G_0 \quad \text{SE } \Gamma \cup \{\neg G_0\} \rightarrow G_0$$

■

Vejamos um exemplo de aplicação do método, vamos provar $(\neg\neg p \supset p)$ a partir de um banco de dados vazio. Convertendo a sentença para a linguagem temos $((p \supset false) \supset false) \supset p$.

Partindo do estado inicial abaixo, teremos:

1. $(\{\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash ((p \supset false) \supset false) \supset p$

2. $(\{(p \supset false) \supset false\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash p$ R–6.⊃

3. $(\{(p \supset false) \supset false\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash p \supset false$ R–6.4

4. $(\{(p \supset false) \supset false, p\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash false$ R–6.⊃

5. $(\{(p \supset false) \supset false, p\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash ((p \supset false) \supset false) \supset p$ R–6.5

6. $(\{(p \supset false) \supset false, p\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash p$ R–6.⊃

7. $(\{(p \supset false) \supset false, p\},$
   $((p \supset false) \supset false) \supset p$
   $) \vdash true$ R–6.1

■

Em uma prova deste método, partimos da sentença que queremos demonstrar até chegar a uma tautologia ($\Gamma \rightarrow true$). O fato de chegarmos a uma tautologia

realmente garante a correção da prova, visto que existe uma prova de dedução natural que gera todas as fórmulas acorrendo nesta prova da última para a primeira, nesta ordem.

Vamos usar este mesmo exemplo para mostrar como cada passo pode ser substituido por uma seqüência de passos de dedução natual. Ou seja, para cada linha n da prova acima, que dá origem a uma linha $n+1$, iremos apresentar uma derivação por dedução natural que gera a linha n, a partir da linha $n+1$. Cada linha da prova acima será traduzida para sua correspondente em notação de dedução natural. Primeiramente vamos apresentar a dedução correspondente a prova completa, para depois colocar as duas deduções com seus passos correspondentes.

Na prova intercalada, tomaremos $G_0 = ((p \supset false) \supset false) \supset p$:

$0'$.	(linha 7):	$\neg G_0, p \rightarrow true$	
$1'$.	(linha 6):	$\neg G_0, p \rightarrow p$	.axioma da prem.
$2'$.		$\neg G_0, p, (p \supset false) \supset false \rightarrow p$	.Int. prem. 1
$3'$.	(linha 5):	$\neg G_0, p \rightarrow G_0$	.Int $\supset$ 2
$4'$.		$\neg G_0, p \rightarrow \neg G_0$	.axioma da prem.
$5'$.	(linha 4):	$\neg G_0, p \rightarrow false$	.Elim. $\neg$ 3 e 4
$6'$.	(linha 3):	$\neg G_0 \rightarrow p \supset false$	.Int. $\supset$ 5
$7'$.		$\neg G_0, (p \supset false) \supset false \rightarrow p \supset false$	.Int. prem. 5
$8'$.		$\neg G_0, (p \supset false) \supset false \rightarrow (p \supset false) \supset false$	.axioma da prem.
$9'$.		$\neg G_0, (p \supset false) \supset false \rightarrow false$	.Elim. $\supset$ 7 e 8
$10'$.		$\neg G_0, (p \supset false) \supset false \rightarrow \neg false$	.axioma do $false$
$11'$.	(linha 2):	$\neg G_0, (p \supset false) \supset false \rightarrow p$	.Elim. $\neg$ 9 e 10
$12'$.	(linha 1):	$\neg G_0 \rightarrow G_0$	.Int. $\supset$ 11
$13'$.		$G_0 \rightarrow G_0$	.axioma da prem.
$14'$.		$\rightarrow G_0$	.Elim. prem. 12 13

■

VI.1.4 Extensão para Operadores Modais.

Chamamos atenção para o fato de que as regras abaixo são para o sistema K, onde não podemos assumir nenhuma propriedade especial da relação de acessibilidade. Nas extensões para outros sistemas são justamente as regras abaixo que são modificadas ou estendidas de forma a refletir as propriedades específicas da relação de acessibilidade.

- Regra para o operador necessidade (R–6.$\Box$): Dado que o estado atual é da forma $(BD, G_0) \vdash \Box\alpha$, então gere o estado sucessor $(BD', \alpha) \vdash \alpha$ onde BD' é

o banco de dados genérico de BD, construído pelo algoritmo abaixo.

Dado um banco de dados BD, o banco de dados genérico BD' é construído a partir de BD obedecendo às seguintes regras:

1. Se existe uma premissa da forma $\Box\beta$ em BD então inclua a premissa β em BD'.
2. Se existe uma premissa da forma $(\gamma \supset \Box\beta)$ em BD e podemos demonstrar $(BD, G_0) \vdash \gamma$, então inclua a premissa β em BD'.[1]

- Regra para o operador possibilidade (R–6.$\Diamond$): Dado que o estado atual é da forma $(BD, G_0) \vdash \Diamond\alpha$, então gere o estado o sucessor $(BD^*, \alpha) \vdash \alpha$, onde BD^* é um banco de dados particular de BD, construído pelo algoritmo abaixo.

  Dado um banco de dados BD, um banco de dados particular BD^* (se existir) é construído a partir de BD obedecendo às seguintes regras:

  1. Todo banco de dados particular de BD será um superconjunto do banco de dados genérico de BD ($BD' \subset BD^*$).
  2. Todo banco de bados particular de BD deve incluir uma e exatamente uma sentença δ que satisfaça uma das seguintes condições:
     (a) Se existir uma premissa da forma $\Diamond\beta$ em BD então δ pode ser igual a β.
     (b) Se existir uma premissa da forma $(\gamma \supset \Diamond\beta)$ em BD e pudermos demonstrar $(BD, G_0) \vdash \gamma$, então δ pode ser igual a β.[1]

Iremos apresentar mais adiante exemplos que demonstram a incompletude destas regras (mesmo com as modificações incluídas). Em vista disto, nos preocuparemos apenas na demonstração de suas correções, faremos isto através da semântica, enquanto Quintana fez por correspondência à dedução natural. Para isto, iremos usar as seguintes notações: $M, w \models \alpha$, indicará que a fórmula α é satisfeita pelo modelo M no mundo w. $M, w \models BD$, indicará que todas as fórmulas do conjunto BD são satisfeitas pelo modelo M no mundo w.

As provas de correção das regras R–6.$\Box$ e R–6.$\Diamond$ serão baseadas nas definições de banco de dados genérico e banco de dados particular, e constatando que se o antecedente da regra é válido o conseqüente também será válido. Entretanto, a construção dos banco de dados genérico e particular usam provas auxiliares do próprio método

[1]Incluímos aqui uma modificação de Zaverucha [Zaverucha 89], pois pela regra original teríamos que demonstrar $(BD, \gamma,) \vdash \gamma$, o que traz problemas de completude.

para determinar a inclusão ou não de determinadas sentenças no banco. E estas provas eventualmente usarão as regras R–6.$\Box$ e R–6.$\Diamond$.

Ou seja, a correção das regras R–6.$\Box$ e R–6.$\Diamond$ estará baseada na coreção da construção dos bancos de dados genérico e particular respectivamente, mas estas últimas correções estarão baseadas na coreção de todo o método, inclusive das regras R–6.$\Box$ e R–6.$\Diamond$. Vamos então resolver este aparente erro antes de apresentar a correção das regras.

Primeiramente, vamos supor que na construção de algum banco de dados genérico ou particular nos utilizamos das regras R–6.$\Box$ ou R–6.$\Diamond$, o que implica em constuir outro banco de dados genérico ou particular. Prosseguindo com este raciocínio só temos duas opções: ou em algum momento a construção de um banco de dados genérico ou particular não necessitará da aplicação da regra R–6.$\Box$ ou R–6.$\Diamond$, ou sempre iremos precisar da regra R–6.$\Box$ ou da regra R–6.$\Diamond$ na construção de um banco de dados genérico ou particular.

No primeiro caso, resolvemos o nosso problema, pois a construção do último banco de dados estará baseada somente nas outras regras do método, que já provamos serem corretas. Uma vez que a construção do último banco de dados for correta, demonstramos a correção da última aplicação da regra R–6.$\Box$ ou da regra R–6.$\Diamond$, para então obter a correção do penúltimo banco de dados, e assim por diante até a primeira aplicação da regra R–6.$\Box$ ou da regra R–6.$\Diamond$.

O segundo caso não pode ser algoritmicamente resolvido, logo todo problema que recair neste caso entrará fatalmente em ciclo infinito. Ou seja, não precisamos nos preocupar com a correção destes casos.

Por último, também podemos fazer analogia à dedução natural para justificar a nossa prova. Por exemplo, a regra de introdução de $\wedge$ diz que se $\Gamma \rightarrow A$ pode ser deduzido e que se $\Gamma \rightarrow B$ pode ser deduzido, então podemos deduzir $\Gamma \rightarrow A \wedge B$, mas não é feita nenhuma restrição no uso da própria regra de introdução do $\wedge$ na dedução dos antecedentes $\Gamma \rightarrow A$ ou $\Gamma \rightarrow B$.

Vejamos então a correção da regra R–6.$\Box$, traduzindo-a para a notação de dedução natural, teremos:

$$BD \cup \{\neg G_0\} \rightarrow \Box\alpha \text{ SE } BD' \cup \{\neg\alpha\} \rightarrow \alpha$$

onde $BD' = \{\beta / \Box\beta \in BD$ ou $(\gamma \supset \Box\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma\}$

Vamos mostrar que se o antecedente da regra for válido então o conseqüente terá que ser válido também. Supomos então que o antecedente é válido:

$$\models BD' \cup \{\neg\alpha\} \rightarrow \alpha$$

Temos agora que provar que o conseqüente também será válido, ou seja, para qualquer modelo M, a sua avaliação em um mundo qualquer w_q será verdadeira:

$$M, w_q \models BD \cup \{\neg G_0\} \rightarrow \Box\alpha$$

Para os modelos que não satisfazem $BD \cup \{\neg G_0\}$, a fórmula $BD \cup \{\neg G_0\} \rightarrow \Box\alpha$ será trivialmente verdadeira. Vamos então nos preocupar com os modelos nos quais $BD \cup \{\neg G_0\}$ é satisfeito. Logo, assumindo:

$$M, w_q \models BD \cup \{\neg G_0\}$$

temos que demonstrar:

$$M, w_q \models \Box\alpha$$

Segundo a definição do operador $\Box$, para demonstrar a fórmula acima temos que demonstrar α para todo o mundo w acessado por w_q:

$$M, w \models \alpha$$

Antes, mostraremos que $M, w \models BD'$ para todo o mundo w acessado por w_q.

Para provar que o modelo M satisfaz o conjunto BD' em todos os mundos acessados por w_q, temos que demonstrar que todos as premissas de BD' são satisfeitas por M nestes mundos. Como as premissas de BD' obedecem a duas regras de formação, vamos então provar que dada uma premissa qualquer de BD', incluída pela regra 1 ou pela regra 2, M satisfaz esta premissa em todos os mundos acessados por w_q.

regra 1: $\Box\beta \in BD$ então $\beta \in BD'$

Ou seja, temos que provar que: Se $\Box\beta \in BD$ então $M, w \models \beta$ para todo o mudo w acessado por w_q.

Sabemos:

$$M, w_q \models BD \cup \{\neg G_0\}$$

Logo, se:

$$\Box\beta \in BD$$

temos:

$$M, w_q \models \Box\beta$$

isto é $M, w \models \beta$ para todo mundo w acessado por w_q.

regra 2: $(\gamma \supset \Box\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma$ então $\beta \in BD'$

Ou seja, temos que provar que: Se $(\gamma \supset \Box\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma$ então $M, w \models \beta$ para todo o mudo w acessado por w_q.

Sabemos:

$$M, w_q \models BD \cup \{\neg G_0\}$$

Logo, se:

$(\gamma \supset \Box\beta) \in BD$ e
$BD \cup \{\neg G_0\} \rightarrow \gamma$
(e as demais regras do método são corretas)

temos:

$M, w_q \models \gamma \supset \Box\beta$
$M, w_q \models \gamma$ (porque o método é correto)
$M, w_q \models \Box\beta$

Logo $M, w \models \beta$ para todo mundo w acessado por w_q.

Provamos então que, se $M, w_q \models BD \cup \{\neg G_0\}$ então vale:

$$M, w \models BD' \text{ para todo mundo } w \text{ acessado por } w_q.$$

Como a fórmula $BD' \cup \{\neg\alpha\} \rightarrow \alpha$ é válida por suposição, temos:

$$M, w \models \alpha \text{ para todo mundo } w \text{ acessado por } w_q$$

pois, se $M, w \models \neg\alpha$, teríamos $M, w \models BD' \cup \{\neg\alpha\}$ e então $M, w \models \alpha$, que é impossível. Temos então:

$$M, w_q \models \Box\alpha$$

Ou seja, se assumirmos o antecedente da regra como válido, para todo o modelo M que tomarmos, em um mundo qualquer w_q, se

$$M, w_q \models BD \cup \{\neg G_0\}$$

então

$$M, w_q \models \Box\alpha$$

Logo, temos que a fórmula

$$BD \cup \{\neg G_0\} \rightarrow \Box\alpha$$

será válida, em vista do método de dedução natural ser completo. O que termina nossa prova.

Vejamos então a correção da regra R–6.$\Diamond$:

$$BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha \text{ SE } BD^* \cup \{\neg\alpha\} \rightarrow \alpha$$

onde $BD^* = BD' \cup \{\delta\}$, dado que δ é um elemento do conjunto abaixo:

$$\{\beta / \Diamond\beta \in BD \text{ ou } (\gamma \supset \Diamond\beta) \in BD \text{ e } BD \cup \{\neg G_0\} \rightarrow \gamma\}$$

Vamos mostrar que se o antecedente da regra for válido então o conseqüente terá que ser válido também. Supomos então que o antecedente é válido, para algum banco de dados particular:

$$\models BD^* \cup \{\neg\alpha\} \rightarrow \alpha$$

Temos agora que provar que o conseqüente também será válido, ou seja, para qualquer modelo M, a sua avaliação em um mundo qualquer w_q será verdadeira:

$$M, w_q \models BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha$$

Assumindo:

$$M, w_q \models BD \cup \{\neg G_0\}$$

temos que demonstrar:

$$M, w_q \models \Diamond\alpha$$

Segundo a definição do operador $\Diamond$, para demonstrar o fórmula acima temos que demonstrar α para algum mundo acessado w por w_q:

$$M, w \models \alpha$$

Para isto, precisamos antes provar que: $M, w \models BD^*$ para algum w.

Como já provamos que, uma vez assumido $M, w_q \models BD \cup \{\neg G_0\}$, então vale $M, w \models BD'$ para todos os mundos w acessados por w_q, falta apenas provar que a sentença da forma δ é satisfeita em pelo menos um mundo acessado por w_q.

regra para a sentença δ: Se $\Diamond\beta \in BD$ ou $(\gamma \supset \Diamond\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma$ então δ pode ser igual a β

Ou seja, temos que provar: Se $\Diamond\beta \in BD$ ou $(\gamma \supset \Diamond\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma$ então $M, w \models \beta$ para algum mundo w acessado por w_q.

Logo, se: $\Diamond\beta \in BD$ ou $(\gamma \supset \Diamond\beta) \in BD$ e $BD \cup \{\neg G_0\} \rightarrow \gamma$ temos:

$$M, w_q \models \Diamond\beta$$

ou

$$M, w_q \models \gamma \supset \Diamond\beta,$$
$$M, w_q \models \gamma \text{ e}$$
$$M, w_q \models \Diamond\beta$$

Em ambos os casos $M, w \models \beta$, para algum mundo w acessado por w_q.

Provamos então que, se $M, w_q \models BD \cup \{G_0\}$ então vale:

$$M, w \models BD^* \text{ para algum } w \text{ acessado por } w_q.$$

Voltando à regra que queremos provar, assumimos:

$$\models BD^* \cup \{\neg\alpha\} \rightarrow \alpha$$

e

$$M, w_q \models BD \cup \{\neg G_0\}$$

e mostramos que:

$$M, w \models BD^* \text{ para algum mundo } w \text{ acessado por } w_q$$

Como a fórmula $BD^* \cup \{\neg\alpha\} \rightarrow \alpha$ é válida, temos:

$$M, w \models \alpha \text{ para algum mundo } w \text{ acessado por } w_q$$

temos então:

$$M, w_q \models \Diamond\alpha$$

Ou seja, se assumirmos o antecedente da regra como válido, para todo o modelo M que tomarmos, em um mundo qualquer w_q, teremos:

$$M, w_q \models BD \cup \{\neg G_0\} \supset \Diamond\alpha$$

Logo, temos que a fórmula

$$BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha$$

será válida, em vista do método de dedução natural ser completo. O que termina nossa prova.

Chamamos atenção para o fato que a definição de banco de dados particular exige a introdução de exatamente uma sentença δ no banco de dados genérico. Caso

contrário teríamos a validade das duas sentenças abaixo. Sendo que a primeira (1) não é válida no sistema K e a segunda (2) não é válida nem no sistema S5.

$$\Box\alpha \supset \Diamond\alpha \qquad (1)$$
$$\Diamond\alpha \wedge \Diamond\beta \supset \Diamond(\alpha \wedge \beta) \qquad (2)$$

Vejamos um exemplo simples da aplicação destas regras: Considere um banco de dados inicial formados pelas seguintes premissas:

$$BD = \{\Box q, p, p \supset \Diamond(q \supset \Box p)\}$$

A partir deste banco de dados queremos provar a sentença $G_0 = \Diamond\Box p$.

Para este banco de dados teremos o seguinte banco de dados particular:

$$BD^* = \{q, q \supset \Box p\}$$

conforme veremos a seguir. E este banco de dados particular terá o seguinte banco de dados genérico:

$$(BD^*)' = \{p\}$$

Então vejamos:

1. $(BD, \Diamond\Box p) \vdash \Diamond\Box p$
2. $(BD^*, \Box p) \vdash \Box p$ .R–6.$\Diamond$
3. $((BD^*)', p) \vdash p$ .R–6.$\Box$
4. $((BD^*)', p) \vdash true$ .R–6.1

■

Para aplicar a regra R–6.$\Diamond$ na linha 2, precisamos construir o banco de dados particular de $BD = \{\Box q, p, p \supset \Diamond(q \supset \Box p)\}$. O banco de dados genérico será $\{q\}$, pois a sentença $\Box q$ pertence ao banco de dados atual. A sentença δ só pode ser igual à $q \supset \Box p$, pois temos a premissa $p \supset \Diamond(\delta)$ no banco de dados atual e podemos provar p a partir deste. Logo, o banco de dados particular será $\{q, q \supset \Box p\}$.

Para aplicar a regra R–6.$\Box$ na linha 3, precisamos calcular o banco de dados genérico do banco de dados atual $BD^* = \{q, q \supset \Box p\}$, que será $\{p\}$, pois $q \supset \Box p$, pertence ao banco de dados e podemos provar q a partir deste.

VI.1.5 Melhorando a Completude.

Como dissemos anteriormente, embora a proposta seja extremamente elegante, ela peca por apresentar problemas relacionados à completude. Vejamos um exemplo que não pode ser resolvido.

Seja o banco de dados $\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\}$, vamos tentar provar $\Box q$ a partir destas premissas. Antes de aplicar o método, vamos analisar se o gol é conseqüência lógica do nosso conjunto de premissas, temos:

$$BD = \{\neg p, \neg(\Box q \wedge r) \supset p\}$$
$$G_0 = \Box q$$

Trocando a última premissa por uma fórmula equivalente ($\alpha \supset \beta \equiv \neg\beta \supset \neg\alpha$) teremos:

$$BD = \{\neg p, \neg p \supset (\Box q \wedge r)\}$$
$$G_0 = \Box q$$

Ou seja, $\Box q$ realmente é uma conseqüência lógica do banco de dados. Vamos verificar então se o método consegue obter uma prova para o nosso exemplo:

1. $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\},$
   $\Box q$
   $) \vdash \Box q$

Como no banco de dados não há nenhuma sentença da forma $\Box\alpha$ ou $\beta \supset \Box\alpha$, o banco de dados genérico será vazio.

2. $(\{\},$
   $q,$
   $) \vdash q$ .R–6.□

Neste ponto, não podemos aplicar nenhuma outra regra. Logo, o método não foi capaz de realizar a prova.

Existem estudos que propõem modificações do método para resolver os problemas de incompletude que foram anteriormente identificados [Zaverucha 91]. Vamos apresentar uma modificação original, que resolve alguns destes problemas[2]. A modificação se baseia no fato de que podemos provar qualquer sentença a partir de um conjunto de premissas inconsistente (regra derivada 1). Acrescentamos, então, a seguinte regra de formação do banco de dados genérico.

3. Se podemos provar $(BD, G_0) \vdash false$, então inclua a premissa $false$ em BD'.

Vejamos agora se conseguimos obter a prova acima.

1. $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\},$
   $\Box q$
   $) \vdash \Box q$

[2]Outras modificações, mais completas que esta, foram propostas por Zaverucha [Zaverucha 89].

Não há nenhuma sentença da forma $\Box\alpha$ ou $\beta \supset \Box\alpha$ no banco de dados. Aplicando a terceira regra de formação vamos usar uma prova auxiliar para tentar demonstrar $false$ a partir do banco de dados atual.

$1'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\},$
$\Box q$
$) \vdash false$

$2'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\},$
$\Box q$
$) \vdash p$ .R–6.3

$3'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p\},$
$\Box q$
$) \vdash (\Box q \wedge r) \supset false$ .R–6.3

$4'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p, \Box q, r\},$
$\Box q$
$) \vdash false$ .R–6.$\supset$

$5'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p, \Box q, r\},$
$\Box q$
$) \vdash p$ .R–6.3

$6'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p, \Box q, r\},$
$\Box q$
$) \vdash (\Box q \wedge r) \supset false$ .R–6.3

$7'.$ $(\{p \supset false, ((\Box q \wedge r) \supset false) \supset p, \Box q, r\},$
$\Box q,$
$) \vdash \Box q$ .R–6.5

O Banco de dados genérico do banco de dados atual será $\{q\}$, tendo em vista que a premissa $\Box q$ pertence ao banco de dados atual. A tentativa de provar $false$ a partir do banco de dados atual levará a um estado anterior $(4')$, logo a regra de

formação 3 do banco de dados genérico, neste caso, acabará não sendo usada.

$8'.\ (\{q\},$
$\quad q,$
$\quad) \vdash q$ .R–6.□

$9'.\ (\{q\},$
$\quad q,$
$\quad) \vdash true$ .R–6.1

Assim obtemos pela nova regra de formação do banco de dados particular o banco $BD' = \{false\}$, que é usado no prosseguimento da prova original.

$2.\ (\{false\},$
$\quad q$
$\quad) \vdash q$ .R–6.□

$3.\ (\{false\},$
$\quad q$
$\quad) \vdash true$ .R–6.2

■

Ou seja, com a modificação introduzida passamos gerar uma prova correta para o problema considerado.

VI.1.6 Extensão para Lógica de Primeira Ordem.

Ao iniciarmos uma prova por este método, incluímos no banco de dados do estado inicial as fórmulas que representam as premissas. Para isto, removemos completamente os quantificadores destas fórmulas. O primeiro passo deste processo é a eliminação das variáveis quantificadas existencialmente através de funções de Skolem. Em seguida, como todas as variáveis restantes são quantificadas universalmente, seus quantificadores são omitidos. Na seção VI.1.8 veremos um mecanismo que permite distinguir se uma fórmula no banco de dados como $\Box p(x)$ esta representando a premissa $\forall x \Box p(x)$ ou a premissa $\Box(\forall x p(x))$.

Já uma fórmula no gol inicial, ou no gol atual, representa que estamos interessados na sua prova para uma instância qualquer de suas variáveis. Logo, todas as suas variáveis são consideradas como quantificadas existencialmente. Ou seja, o gol $\Box p(x)$ estará representando que estamos interessados em provar ou a fórmula

$\exists x \Box p(x)$ ou $\Box(\exists x p(x))$ (a seleção entre a fórmula que o gol representa também será feita pelo mesmo mecanismo que veremos na seção VI.1.8).

Existe uma terceira classe de variáveis. São as variáveis de fórmulas que foram incluidas no banco de dados pela regra R-6.$\supset$. Estas variáveis, embora estejam em fórmulas do banco de dados, são quantificadas existencialmente. Desta forma, necessitamos de uma notação que permita distinguir se uma variável de uma fórmula no banco de dados está quantificada universalmente ou existencialmente. Para isto, passaremos a representar as variáveis quantificadas existencialmente como letras sublinhadas, como por exemplo:

1. $(\{\}, (p(\underline{x}, \underline{y}) \wedge q(\underline{y})) \supset p(a, \underline{x})) \vdash (p(\underline{x}, \underline{y}) \wedge q(\underline{y})) \supset p(a, \underline{x})$
2. $(\{p(\underline{x}, \underline{y}), q(\underline{y})\}, (p(\underline{x}, \underline{y}) \wedge q(\underline{y})) \supset p(a, \underline{x})) \vdash p(a, \underline{x})$ .R-6.$\supset$
3. $(\{p(a, a), q(a)\}, (p(a, a) \wedge q(a)) \supset p(a, a)) \vdash true$ .R-6.1

■

Note que as variáveis quantificadas existencialmente permanecem relacionadas entre si, isto é, a instanciação de uma ocorrência da variável $\underline{x}$ afetará todas as ocorrências desta variável no banco de dados, no gol inicial e gol atual. Este procedimento não será aplicado às variáveis quantificadas universalmente, vejamos com cuidado esta diferenciação. (A notação usada por nós aqui difere da original, veremos porque na seção VI.1.7.)

Ao fazermos a extensão para a lógica de primeira ordem, estamos exigindo que as regras apresentadas anteriormente sejam modificadas de forma que as condições como $\alpha \in BD$ sejam entendidas como: Dada a formula α e o banco de dados BD, verifique que existe uma fórmula β em BD, tal que uma instância de α seja igual a uma instância de β, ou seja se α e β unificam.

Existem três tipos de termos que podem ser unificados entre si: as variáveis quantificadas universalmente, as variáveis quantificadas existencialmente, estejam elas no banco de dados, no gol inicial ou no gol atual, e os demais termos como constantes ou funções. Só consideraremos como válidas as unificações nas quais as substituições de variáveis são feitas entre elementos do mesmo tipo, ou substituindo um elemento de um tipo por um outro elemento de tipo mais restritivo, onde consideramos as variáveis quantificadas existencialmente como mais restritivas que as quantificadas universalmente, e os termos como constantes e funções os mais restritivos entre os três tipos.

Outra modificação do funcionamento é que, como uma fórmula no banco de dados original só possue variáveis quantificadas universalmente, podemos gerar instâncias destas fórmulas sem modificar a fórmula no banco de dados ou outras fórmulas unificadas com ela anteriormente. Já no caso de uma fórmula incluída da banco de

dados pela regra R–6.$\supset$ ou de uma fórmula no gol atual, a geração de uma instância dela deve afetar todas as ocorrências das variáveis quantificadas existencialmente que foram modificadas, sejam estas ocorrências no banco de dados, no gol inicial ou no gol atual.

Vejamos um exemplo com o tipo de problema que o mecanismo apresentado evita: Suponha que não usemos este mecanismo na dedução abaixo, obteríamos:

1. $(\{\}, (p(x) \wedge (p(y) \supset q(y))) \supset q(a)) \vdash (p(x) \wedge (p(y) \supset q(y))) \supset q(a)$
2. $(\{p(x), p(y) \supset q(y)\}, (p(x) \wedge (p(y) \supset q(y))) \supset q(a)) \vdash q(a)$ .R–6.$\supset$
3. $(\{p(x), p(y) \supset q(y)\}, (p(x) \wedge (p(y) \supset q(y))) \supset q(a)) \vdash p(y)$ .R–6.3
4. $(\{p(x), p(y) \supset q(y)\}, (p(x) \wedge (p(y) \supset q(y))) \supset q(a)) \vdash true$ .R–6.1

■

Repare que na derivação acima, como as fórmulas incluídas no banco de dados não tiveram suas variáveis quantificadas existencialmente marcadas, perdemos a informação que as variáveis x e y do gol precisam ser unificadas com a constante a para garantir a validade do gol.

Uma outra modificação que deve ser feita no método é que as provas paralelas criadas pela regra R–6.$\wedge$, ou indiretamente pela regra R–6.$\supset$ quando o lado esquerdo do conectivo $\supset$ for uma conjunção, deverão, cada uma, considerar as substituições nas variáveis quantificadas existencialmente das outras provas. Por exemplo, dado o estado $(BD, G_0) \vdash p(x) \wedge q(x, z)$, geraremos as provas paralelas $(BD, G_0) \vdash p(x)$ e $(BD, G_0) \vdash q(x, z)$, caso a dedução da primeria unifique a variável x com o termo t, a segunda prova passará a ser $(BD, G_0) \vdash q(t, z)$. Vejamos um exemplo, onde este funcionamento é importante, seja o banco de dados abaixo:

$$BD = \{p(a), q(b), q(x) \supset r(x), (r(y) \wedge p(y)) \supset s(y)\}$$

Dado o gol inicial $G_0 = s(w)$, temos:

1. $(BD, G_0) \vdash s(\underline{w})$
2. $(BD, G_0) \vdash r(\underline{w}) \wedge p(\underline{w})$ .R–6.3

3a. $(BD, G_0) \vdash r(\underline{w})$ .R–6.$\wedge$
4a. $(BD, G_0) \vdash q(\underline{w})$ .R–6.3
5a. $(BD, G_0) \vdash true$ .R–6.1, $\{\langle w \leftarrow b \rangle\}$

3b. $(BD, G_0) \vdash p(\underline{w})$ .R–6.$\wedge$
4b. $(BD, G_0) \vdash p(b)$ .aplicando $\{\langle w \leftarrow b \rangle\}$

de onde não podemos prosseguir, logo, a prova não pode ser terminada.

O funcionamento do mecanismo apresentado nesta seção é coerente com a regra de dedução natural de introdução do $\exists$. Por exemplo, a seqüência:

$$\Gamma, p(a) \rightarrow q(a)$$
$$\Gamma \rightarrow p(a) \supset q(a)$$
$$\Gamma \rightarrow \exists x(p(x) \supset q(x))$$

no método poderia ser representada como:

$$(BD, G_0) \vdash p(\underline{x}) \supset q(\underline{x})$$
$$(BD \cup \{p(\underline{x})\}, G_0) \vdash q(\underline{x})$$
$$(BD \cup \{p(a)\}, G_0) \vdash \alpha$$

onde existe, por exemplo, uma sentença da forma $\alpha \supset q(a)$ em BD. Ou seja, enquanto na dedução natural a instanciação ou generalização é feita "a priori", no método, as instanciações só serão feitas após as fórmulas no escopo dos quantificadores existenciais serem desmembradas, isto implica na necessidade de se manter os termos da fórmula original relacionados, de forma que a instanciação de um deles afete os demais.

VI.1.7 Diferença de Notações.

A notação usada para permitir a extensão do método para lógica de primeira ordem difere daquela adotada na apresentação original da abordagem. Esta diferenciação se dá em dois pontos: na distinção entre variáveis quantificadas universalmente e existencialmante e na representação de ramos paralelos dentro de uma prova.

A abordagem original separa em dois banco de dados diferentes os dois tipos de fórmulas. No primeiro são colocadas as fórmulas que só possuem variáveis quantificadas universalmente e no segundo são colocadas as fórmulas que só possuem variáveis quantificadas existencialmente. Observemos, entretanto o exemplo abaixo:

1. $(\{p(x) \supset \Box q(x), ...\}, p(\underline{y}) \supset \Box r(\underline{y})) \vdash p(\underline{y}) \supset \Box r(\underline{y})$
2. $(\{p(x) \supset \Box q(x), ..., p(\underline{y})\}, p(\underline{y}) \supset r(\underline{y})) \vdash \Box r(\underline{y})$ .R–6.$\supset$
3. $(\{q(x), ...\}, r(\underline{y})) \vdash r(\underline{y})$ .R–6.$\Box$

$\vdots$

O termo p(y) incluído no banco de dados no passo 2, tem a variável y quantificada como existencialmente e ela está associada à variável y do gol. No passo 3, ao gerarmos o banco de dados genérico o termo $q(x)$ foi incluído porque havia a premissa $p(x) \supset \Box q(x)$ no banco de dados e $p(x)$ pode ser provado a partir daquele banco de dados. Observe que durante a prova de $p(x)$, a variável x foi unificada com a variável

$\underline{y}$ de $p(\underline{y})$. Logo, a premissa $q(x)$ no banco de dados genérico deve ter sua variável x quantificada existencialmente, e esta variável deve estar associada à variável $\underline{y}$ do gol $r(\underline{y})$.

Já se no lugar da subfórmula $q(x)$ houvesse uma subfórmula como $q(x,z)$, a variável x desta fórmula seria quantificada existencialmente e a variável z seria quantificada universalmente. Deste modo é possível a existência de sentenças tanto com variáveis quantificadas universalmente como existencialmente, logo, a separação das fórmulas na abordagem original não consegue manipular este tipo de sentença. Por causa disto, o banco de dados da proposta original também poderá conter variáveis unificadas eixtencialmente. Logo, preferímos manter um só banco de dados, com os dois tipos de variáveis.

Por exemplo, dada a derivação abaixo, temos:

1. $(\{p(x) \supset \Box q(x,b)\}, p(\underline{y}) \supset \Box q(a,\underline{y})) \vdash p(\underline{y}) \supset \Box q(a,\underline{y})$
2. $(\{p(x) \supset \Box q(x,b), p(\underline{y})\}, p(\underline{y}) \supset \Box q(a,\underline{y})) \vdash \Box q(a,\underline{y})$ .R–6.$\supset$
3. $(\{q(x,b)\}, q(a,\underline{y})) \vdash q(a,\underline{y})$ .R–6.$\Box$
4. $(\{q(x,b)\}, q(a,b)) \vdash true$ .R–6.1 $\{\langle y \leftarrow b\rangle\}$

■

que é uma derivação incorreta. Aplicando o mecanismo definido acima, teríamos:

1. $(\{p(x) \supset \Box q(x,b)\}, p(\underline{y}) \supset \Box q(a,\underline{y})) \vdash p(\underline{y}) \supset \Box q(a,\underline{y})$
2. $(\{p(x) \supset \Box q(x,b), p(\underline{y})\}, p(\underline{y}) \supset \Box q(a,\underline{y})) \vdash \Box q(a,\underline{y})$ .R–6.$\supset$
3. $(\{q(\underline{y},b)\}, q(a,\underline{y})) \vdash q(a,\underline{y})$ .R–6.$\Box$

que não pode ser concluída. Ou seja, funciona corretamente.

A outra diferenciação é que na abordagem original, quando é feita a extensão para lógica de primeira ordem o gol inicial é substituido por uma lista de gols. Ou seja, em nossa apresentação, quando temos que resolver uma conjunção de termos, resolvemos cada termo em ramos de provas separados, propagando as instanciações das anteriores para a próxima prova. Na abordagem original, Quintana matém todos os termos das conjunções em uma lista única, desta forma ele consegue expressar, na apresentação teórica, que instanciações de variáveis quantificadas existencialmente em um ramo da prova afetam todos os ramos. Preferimos não usar esta notação e deixar o efeito de instanciações de cada ramo em outro de forma implícita para não termos que redefinir todas as regras apresentadas anteriormente com a nova notação.

Em nossa proposta de implementação, como poderá ser visto mais tarde, adotamos a representação dos gols a serem resolvidos também em forma de lista, baseamos entretanto esta idéia no esquema de implementação de resolução apresentado por

Cohen [Cohen 85], no qual os gols a serem resolvidos são colocados em uma lista e a inferência considera sempre o primeiro elemento da lista para aplicar uma regra de resolução.

É importante ressaltar que, estas diferenças de notação proporcionaram uma apresentação mais simples no nosso caso, e não comprometem em nada nenhuma das caracteristicas do método para os casos considerados, ou seja, o sistema K e K4. Para os demais sistemas não acreditamos que esta mudança de notação cause algum problema.

VI.1.8 O Relacionamento entre Operadores e Quantificadores.

No inicio deste capítulo dissemos que os quantificadores das fórmulas incluídas no banco de dados inicial são eliminados, e isto é feito de tal forma a mantermos a satisfatibilidade das sentenças inalteradas. Vejamos como isto é feito.

- Passo1: Os quantificadores devem ser movidos para a posição mais externa possível de uma fórmula. Eles não devem passar de dentro para fora do escopo de um operador modal. A movimentação pode ser feita pela função de transição t abaixo. Esta função deve ser aplicada de forma a sempre usarmos as regras abaixo na ordem em que elas são enunciadas. Ou seja, dada uma fórmula na qual podemos aplicar mais de uma das regras abaixo, devemos sempre aplicar aquela que foi enunciada primeiro.

$$
\begin{aligned}
&t(\forall x A(x) \supset B) \rightarrow \exists x t(A(x) \supset B)\\
&t(\exists x A(x) \supset B) \rightarrow \forall x t(A(x) \supset B)\\
&t(A \supset \forall x B(x)) \rightarrow \forall x t(A \supset B(x))\\
&t(A \supset \exists x B(x)) \rightarrow \exists x t(A \supset B(x))\\
&t(A \supset B) \rightarrow t(A) \supset t(B)\\
&t(\forall x A(x) \wedge B) \rightarrow \forall x t(A(x) \wedge B)\\
&t(\exists x A(x) \wedge B) \rightarrow \exists x t(A(x) \wedge B)\\
&t(A \wedge B) \rightarrow t(A) \wedge t(B)\\
&t(\forall x A(x)) \rightarrow \forall x t(A(x))\\
&t(\exists x A(x)) \rightarrow \exists x t(A(x))\\
&t(\Box A) \rightarrow \Box t(A)\\
&t(\Diamond A) \rightarrow \Diamond t(A)\\
&t(A) \rightarrow A
\end{aligned}
$$

- Passo 2: Definimos grau modal de uma variável x como o número de escopos de operadores modais no qual o quantificador Qx se encontra. Devemos substituir cada variável x pela representação $**...*x$, onde o número de $*$ é igual ao grau modal da variável.

  Por exemplo, dada a sentença:

  $$\forall x \Box(p(x) \supset \Diamond(\exists y q(y,x)))$$

  geramos a sentença:

  $$\forall x \Box(p(x) \supset \Diamond(\exists y q(**y,x))$$

- Passo 3: Mova os quantificadores para a esquerda da fórmula, mantendo a ordenação entre eles.

- Passo4: Elimine os quantificadores existenciais através de funções de Skolem. Estas funções continuarão marcadas com os $*$.

  No exemplo acima teríamos:

  $$\forall x \Box(p(x) \supset \Diamond(q(** f_s(x),x)$$

- Passo 5: Elimine os quantificadores universais.

■

Seguindo os passos acima conseguimos eliminar os quantificadores de forma que a representação resultante pode ser usada para se reconstituir univocamente a fórmula original. Vejamos agora como o algoritmo de unificação do método funciona de forma a levar em consideração o escopo das variáveis e funções marcadas.

Suponha que nosso banco de dados contém as seguintes sentenças:

$$\forall x \Box p(x)$$
$$\Box(\forall x p(x)) \supset q$$

Convertendo as sentenças teremos:

$$\Box p(x)$$
$$\Box p(*x) \supset q$$

Nosso interesse é determinar se podemos neste caso unificar as variáveis x e $*x$. O método permite que esta seleção seja feita pelo usuário. Para esta especificação é provida uma notação baseada no conceito de subtipo. Ou seja, as variáveis x e $*x$

tem seus valores tomados em mundos distintos, digamos w_i e w_j, cujos domínios são D_i e D_j respectivamente. O usuário poderá especificar se $D_i \subset D_j$, $D_j \subset D_i$, ambos ou nenhum deles, permitindo ou não a unificação de variáveis nestes domínios. Uma vez unificada a variável passará a pertencer ao domínio mais restritivo. Por exemplo, se x varia no domínio D_a, $***x$ no domínio D_b, $D_c \subset D_a$ e $D_c \subset D_b$, então x pode ser unificada com $***x$ e a variável resultante estará no domínio D_c.

Para especificar o relacionamente desejado entre os domínios o método oferece as seguintes notações:

$$a << **a \quad (1)$$
$$*b << b \quad (2)$$

A notação estabelece que expressões usando a letra a determinam características para determinados domínios, ou seja, para variáveis de determinado grau modal. Já expressões usando a letra b determinam características para variáveis de grau modal relativo, ou seja, entre variáveis de grau modal i e grau modal $i+2$ por exemplo. A expressão (1) acima denota que o domínio das variáveis de grau 1 (por exemplo x) é um subconjunto do domínio de variáveis de grau 2 (por exemplo $**x$). A expressão (2) denota que uma variável de grau $i+1$ pode ser unificada com uma variável de grau i, sendo $i+1$ o domínio mais restritivo.

Por exemplo, se temos a sentença $p(x) \supset q(x)$ no banco de dados e o gol $q(***f(y))$, e foram definidas as seguintes relações de subtipo:

$$**a << a \quad (1)$$
$$*b << **b \quad (2)$$

Podemos gerar a instância $p(f(y)) \supset q(f(y))$ daquela premissa do banco de dados. Ao unificarmos $q(***f(y))$ com $q(f(y))$, temos pela relação 1 acima que o domínio de $**f(y)$ é um subconjunto do domínio de $f(y)$, e pela regra 2 temos que os domínios de $*f(y)$ e $**f(y)$ são subconjuntos do domínio de $***f(y)$. Logo, $f(y)$ pode ser unificado com $***f(y)$, e o termo resultante estará no domínio de $**f(y)$. Pela regra R–6.$\supset$ prosseguimos com o gol atual igual a $p(**f(y))$.

VI.1.9 Extensão para Outros Sistemas Modais.

Como dissemos anteriormente, nesta seção iremos exemplificar como podemos estender o método para outros sistemas modais. Para tal, tomaremos a extensão para o sistema K4, que é incompleta também.

O sistema K4 difere do sistema K somente por a sua relação de acessibilidade ser obrigatoriamente transitiva. Logo, as definições de banco de dados genérico, banco

de dados particular, e as regras para os operadores $\Box$ e $\Diamond$ devem ser revistas, de forma a fazer uso desta informação adicional.

Banco de Dados Genérico.

No sistema K4 teremos a validade do axioma abaixo, que representa a transitividade da relação de acessibilidade.

$$\Box\alpha \supset \Box\Box\alpha$$

Logo, podemos definir o banco de dados genérico em K4 (BD'^4) a partir do banco de dados genérico em K (BD').

Para toda sentença α, tal que $\alpha \in BD'$, temos que $\alpha \in BD'^4$
Para toda sentença α, tal que $\alpha \in BD'$, temos que $\Box\alpha \in BD'^4$

Ou seja, α estará no banco de dados genérico se $\Box\alpha$ é deduzido a partir das premissas. Pelo axioma, se $\Box\alpha$ é conseqüência lógica então $\Box\Box\alpha$ também é. Tomando $\Box\beta$ da regra de formação do banco de dados genérico igual a $\Box\Box\alpha$, teremos que foi provado que β $(\Box\alpha)$ é conseqüência lógica, logo $\Box\beta$ também é, e β $(\Box\alpha)$ também deverá fazer parte do banco de dados genérico.

Banco de Dados Particular.

A única modificação a ser feita no banco de dados particular é que ele passe a considerar o banco de dados genérico de K4.

$$BD^{*4} = BD'^4 \cup \{\delta\}$$

Onde a definição de δ é a mesma que em K.

Regra para $\Box$:

Permanece inalterada, apenas passa a considerar o banco de dados genérico para K4.

Se o estado atual é $(BD, G_0) \vdash \Box\alpha$ então gere $(BD'^4, \alpha) \vdash \alpha$

Regra para $\Diamond$:

A regra para o operador $\Diamond$ passa a ser:

Se o estado atual é $(BD, G_0) \vdash \Diamond\alpha$ então gere $(BD^{*4}, \alpha) \vdash \alpha$ ou $(BD^{*4}, \alpha) \vdash \Diamond\alpha$

Observe que a inclusão do termo antecedente $(BD^{*4}, \alpha) \vdash \Diamond\alpha$ ocasionará uma reaplicação recursiva desta regra, o que é coerente com a propriedade de transitividade.

Teríamos aqui que repetir toda a demonstração realizada para as regras R–6.$\Box$ e R–6.$\Diamond$ para demonstrar a correção das modificações realizadas. Como as demonstrações das regras modificadas são praticamente iguais às das regras originais vamos nos limitar a apresentar aqui as diferenças entre elas. O leitor interessado poderá simplesmente reler as demonstrações das regras originais, substituindo os trechos apresentados aqui.

A primeira modificação necessária é a demonstração que, dado um modelo M, tal que em um mundo w_q qualquer, se $M, w_q \models BD \cup \{\neg G_0\}$ teremos $M, w \models BD'^4$ para todo mundo w acessado por w_q. Ou seja, temos que provar que as sentenças incluídas no banco de dados genérico pelas novas regras atendem a propriedade estabelecida acima.

regra 1: $\Box\beta \in BD$ então $\beta \in BD'$ e $\beta \in BD'^4$

(vista anteriormente)

regra 2: $(\gamma \supset \Box\beta) \in DB$ e $(BD, G_0) \vdash \gamma$ então $\beta \in BD'$ e $\beta \in BD'^4$

(vista anteriormente)

regra 3: $\Box\beta \in BD$ então $\beta \in BD'$ e $\Box\beta \in BD'^4$

Sabemos:

$$M, w_q \models BD \cup \{\neg G_0\}$$

e

$$\Box\beta \supset \Box\Box\beta \qquad \text{(axioma de K4)}$$

Logo, se

$$\Box\beta \in BD$$

temos

$$M, w_q \models \Box\beta \text{ e } M, w_q \models \Box\Box\beta$$

portanto

$$M, w \models \Box\beta \text{ para qualquer } w \text{ acessado por } w_q.$$

regra 4: $(\gamma \supset \Box\beta) \in DB$ e $(BD, G_0) \vdash \gamma$ então $\beta \in BD'$ e $\Box\beta \in BD'^4$

Sabemos

$$M, w_q \models BD \cup \{\neg G_0\}$$

e

$$\Box\beta \supset \Box\Box\beta \qquad \text{(axioma de K4)}$$

Logo, se

$$\gamma \supset \Box\beta \in BD \text{ e } (BD, G_0) \vdash \gamma$$

temos que

$$M, w_q \models \gamma \supset \Box\beta \text{ e } M, w_q \models \gamma$$

logo

$$M, w_q \models \Box\beta \text{ e } M, w_q \models \Box\Box\beta$$

portanto

$$M, w \models \Box\beta \text{ para qualquer } w \text{ acessado por } w_q$$

A segunda, e última, modificação necessária é a demonstração da correção da nova regra para o operador $\Diamond$:

Se o estado atual é $(BD, G_0) \vdash \Diamond\alpha$ então gere $(BD^{*4}, \alpha) \vdash \alpha$ ou $(BD^{*4}, \alpha) \vdash \Diamond\alpha$

Esta regra pode ser desmembrada em duas:

Se o estado atual é $(BD, G_0) \vdash \Diamond\alpha$ então gere $(BD^{*4}, \alpha) \vdash \alpha$
Se o estado atual é $(BD, G_0) \vdash \Diamond\alpha$ então gere $(BD^{*4}, \alpha) \vdash \Diamond\alpha$

Como a primeira regra é idêntica à regra para o sistema K, que já foi provada ser correta, vamos nos deter na correção da segunda regra. Reescrevendo a regra com a notação de dedução natural, temos:

$$BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha \;\; \text{Se} \;\; BD^{*4} \cup \{\neg\alpha\} \rightarrow \Diamond\alpha$$

Supomos então que o antecedente é válido, para algum banco de dados particular:

$$\models BD^{*4} \cup \{\neg\alpha\} \rightarrow \Diamond\alpha$$

Temos agora que provar que o conseqüente também será válido, ou seja, para qualquer modelo M, a sua avaliação em um mundo qualquer w_q será verdadeira, isto é:

$$M, w_q \models BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha$$

Para os modelos que não satisfazem $BD \cup \{\neg G_0\}$, a fórmula $BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha$ será trivialmente verdadeira. Vamos então nos preocupar com os modelos nos quais $BD \cup \{\neg G_0\}$ é satisfeito. Logo, assumindo:

$$M, w_q \models BD \cup \{\neg G_0\}$$

temos que demonstrar:

$$M, w_q \models \Diamond\alpha$$

Segundo a definição do operador $\Diamond$, para demonstrar a fórmula acima temos que demonstrar α para algum mundo w acessado por w_q, isto é:

$$M, w \models \alpha$$

Provamos anteriormente para a regra R–6.$\Diamond$, que dado um banco de dados BD^*, se $M, w_q \models BD \cup \{\neg G_0\}$, para algum mundo w acessado por w_q, então

$$M, w \models BD^*$$

Como as regras de construção de BD^{*4} a partir de BD'^4 são as mesmas da construção de BD^* a partir de BD', e já foi provado que BD'^4 é satisfeito em todos mundos acessados de w_q, se $M, w_q \models BD \cup \{\neg G_0\}$, temos que:

Se $M, w_q \models BD \cup \{G_0\}$ então $M, w \models BD^{*4}$ para algum mundo w acessado por w_q.

Voltando à regra que queremos provar, assumimos:

$$\models BD^{*4} \cup \{\neg\alpha\} \rightarrow \Diamond\alpha \text{ e } M, w_q \models BD \cup \{\neg G_0\}$$

Como a fórmula $BD^* \cup \{\neg\alpha\} \rightarrow \Diamond\alpha$ é válida, temos que se $M, w \models \neg\alpha$ teremos $M, w \models \Diamond\alpha$, logo temos:

$$M, w \models \Diamond\alpha \vee \alpha$$

então:

$$1)\ M, w \models \alpha$$

ou

$$2)\ M, w \models \Diamond\alpha$$

Em ambos os casos tem-se $M, w_q \models \Diamond\alpha$, já que a relação de acessibilidade é transitiva. Ou seja:

$$\text{Se } M, w_q \models BD \cup \{\neg G_0\} \text{ então } M, w_q \models \Diamond\alpha, \text{ para qualquer } w_q$$

Logo, temos que a fórmula

$$BD \cup \{\neg G_0\} \rightarrow \Diamond\alpha$$

será válida. O que termina nossa prova.

Particularmente preferiríamos que o novo antecedente fosse $(BD^{*4}, \Diamond\alpha) \vdash \Diamond\alpha$, o que seria mais coerente, entretanto isto é irrelevante neste caso, pois dado que o gol atual do novo antecedente será $\Diamond\alpha$, seremos obrigados a utilizar a regra para o operador $\Diamond$, que gera como estado sucessor uma tripla onde o seu gol inicial não tem nenhuma relação com o gol inicial do estado atual.

VI.2 Comentários.

O que esta abordagem propõe na verdade é a extensão para lógica modal de um método de dedução automática correto e completo para lógica de primeira ordem. O método é construído de maneira que as sentenças sejam expressas em uma forma que permite determinar precisamente qual a próxima regra a ser aplicada, baseado apenas no conectivo principal da sentença. A única excessão são as fórmulas atômicas, que possuem um conjunto de regras que deve ser tentado na ordem estabelecida.

Ao efetuar a extensão para lógica modal o autor usa um mecanismo para tratar o relacionamento entre operadores, quantificadores e variáveis similar ao que havia sido proposto por Konolige [Konolige 86].

Entretanto, a proposta original não é completa. Como dissemos no início do capítulo existem estudos que se propõem a corrigir todos os problemas de completude que já foram detectados [Zaverucha 91]. Como estas correções não foram publicadas, e na falta de uma prova formal de que elas realmente formam um sistema completo, preferimos em nossa apresentação apresentar apenas uma modificação original.

VI.3 Implementação.

Apresentamos agora uma proposta de implementação da abordagem voltada para a linguagem Prolog, onde discutimos os problemas práticos encontrados. O algoritmo 1 abaixo resume o funcionamento central do método:

1. Para cada premissa:
   (a) Traduza a premissa para a linguagem aceita.
   (b) Inclua a premissa traduzida no banco de dados BD.
2. Leia as relações de subtipo.
3. Leia a sentença a ser provada.
4. Faça G_0 ser o gol traduzido para a linguagem aceita.
5. Faça E_0 ser o estado inicial $(BD, G_0) \vdash G_0$.
6. Seja A uma árvore inicialmente vazia.
7. Faça E_0 ser a raiz da árvore A.
8. Seja L uma lista inicialmente vazia.
9. Inclua E_0 na lista L.
10. Se o retrocesso voltar a este ponto:
    (a) Pare e diga que não foi encontrada solução.
11. Se a lista L está vazia:
    (a) Pare e imprima a solução.
12. Aplique as regras de inferência (algoritmo 2 apresentado a seguir).
13. Elimine os estados terminais da lista L.
14. Volte ao passo 11.

Consideramos que os passos necessários para colocar uma sentença qualquer da lógica modal na linguagem aceita estão suficientemente detalhados na apresentação da abordagem, e que a sua implementação não oferece maior dificuldade.

A árvore criada no passo 6, será usada para impedir que em algum ponto do processamento seja gerado um estado que tenha ele mesmo como antecedente. Se isto ocorresse o sistema entraria em ciclo infinito. Consideraremos daqui por diante que o próprio procedimento de incluir um novo estado na árvore fará o percurso desde o nó pai até a raiz, para realizar este teste de ocorrência. Se o teste acusar a existência de um ciclo, é provocado um retrocesso que tentará soluções alternativas

ou fará com que a prova termine com insucesso (passo 10). De forma a possibilitar a implementação do percursos descrito acima, a árvore será montada tendo os nós filhos referenciando os nós pais.

A lista criada no passo 8, será uma estrutura de dados que sempre conterá nós que são folhas da árvore A. Note que no passo 13 os estados terminais são removidos da lista, de forma que ela só contenha os estados que realmente ainda devem ser resolvidos para chegarmos a dedução da fórmula sendo provada.

O passo 10 cria um ponto de retrocesso, que só será alcançado quando todos os percursos no espaço de soluções tiverem sido tentados. E, neste caso, o sistema para informando que não foi encontrada nenhuma solução. O percurso pelo espaço de soluções é feito pelo passo 12.

O passo 12 percorre a lista L, aplicando para cada estado presente na lista, uma regra de dedução. Os estados da lista são substituidos pelos estados sucessores, os quais passam a referenciar o seu estado antecessor. Isto fará a montagem da árvore. Neste exato ponto deve ser realizado o teste de ocorrência. Vejamos então o algoritmo 2 que implementa o passo 12:

- Para cada estado E na lista L:

    1. Se o gol atual de E não é uma fórmula atômica:
        (a) Tome a regra apropriada para o conectivo principal do gol atual.
    2. Se o gol atual de E é uma fórmula atômica:
        (a) Tome uma regra para fórmula atômica (que devem ser tentadas na mesma ordem em que foram apresentadas).
    3. Aplique a regra:
        (a) Se a aplicação não for possível ocorrerá um retrocesso.
    4. Faça os nós sucessores gerados referenciarem o nó E. (Testando a existência de ciclo.)
    5. Substitua o nó E na lista por seus sucessores.

Como dissemos anteriormente, a escolha de qual regra aplicar só não é deterministica no caso de fórmulas atômicas. Neste caso as regras deve ser tentadas, via retrocesso, na mesma ordem em que elas foram apresentadas. Note que, além da aplicação de uma regra para fórmulas atômicas poder causar retrocesso, a aplicação da regra R–6.$\Diamond$ também pode falhar, isto ocorre no caso de não existir um banco de dados particular.

Analisemos agora a aplicação das regras. Tomemos por exemplo a regra R–6.$\supset$.

R–6.$\supset$: Se o estado atual é $(BD, G_0) \vdash \alpha \supset \beta$ então gere $(BD \cup \{\alpha\}, G_0) \vdash \beta$

Ela poderia ser implementada pelo seguinte algoritmo:

1. Seja E o estado atual da forma $(BD, G_0) \vdash G$.
2. Seja G da forma $\alpha \supset \beta$.
3. Seja $BD_1 = BD \cup \{\alpha\}$.
4. Seja $E_1 = (BD_1, G_0) \vdash \beta$.
5. E_1 será o estado sucessor.

Como pode ser visto a implementação das regras é bastante imediata. O único ponto que não está bem claro no algoritmo apresentado acima é a necessidade de se separar as fórmulas que representam os seus fechos universais daquelas que representam os seus fechos existenciais no banco de dados. Ou seja, estamos considerando que a construção $BD \cup \{\alpha\}$ irá marcar a sentença α como existencialmente quantificada. Vejamos a importância desta diferenciação através da análise de implementação da regra R–6.1.

R–6.1: Se o estado atual é $(BD, G_0) \vdash \alpha$ e $\alpha \in BD$ então gere $(BD, G_0) \vdash \mathit{true}$

1. Seja E o estado atual da forma $(BD, G_0) \vdash \alpha$, onde α é uma fórmula atômica.
2. Verifique se existe uma sentença α' em BD, tal que α unifique com α'.
   (a) Se não existir ocorrerá um retrocesso.
3. O estado sucessor será um estado terminal.

O ponto que merece nossa atenção é a unificação dos termos α e α'. Esta unificação não pode ser feita diretamente pelo Prolog pois a unificação das variáveis precisa obedecer o algoritmo de subtipo que veremos a seguir. No momento vamos nos preocurar com a diferenciação das variáveis universais e existenciais. Se α' é uma sentença que representa o seu fecho existencial, ou seja ela foi incluída no banco de dados pela regra R–6.$\supset$, então a unificação das variáveis da sentença α' com as variáveis da sentença α (depois de verificado a existência de um subtipo em comum) pode ser feita diretamente pelo Prolog. Isto ocasionará que as variáveis das demais sentenças existenciais no banco de dados e os gols sofram as alterações necessárias.

Já no caso de uma sentença do banco de dados que representa o seu fecho universal, a unificação deve se processar de forma a que as variáveis da própria sentença

no banco de dados, e as variáveis de outras sentenças unificadas anteriormente com ela, não sejam alteradas. Uma forma simples de implementar este funcionamento, é gerar uma cópia da sentença α', e depois realizar a unificação de α com esta cópia.

Durante a unificação de variáveis estaremos testando a existência de um subtipo em comum e associando este subtipo à variável após a unificação. A abordagem sugere um predicado Prolog que determina o subtipo em comum de duas variáveis. Dado dois tipos `A` e `B`, o predicado `comp_type(A,B,C)` devolve em `C` o subtipo em comum, se existir. Os subtipos são representados como `sub_type(D,E)`, onde `D` é subtipo de `E`.

```
comp_type(A,A,A) :- !.
comp_type(A,B,A) :- sub_type(A,B), !.
comp_type(A,B,B) :- sub_type(B,A), !.
comp_type(A,B,C) :- sub_type(A,D),
                    comp_type(B,D,C).
```

Até este ponto a implementação da abordagem é direta e eficiente e quase determinística. O único ponto de retrocesso até o momento é a decisão de qual regra aplicar se o gol atual for uma fórmula atômica. Vejamos finalmente a implementação do cálculo dos banco de dados genérico e particular, que nos fará mudar radicalmente de opinião sobre o método proposto.

O problema da implementação do banco da dados genérico é que para cada termo da forma $\alpha \supset \Box\beta$ ou $\alpha \supset \Diamond\beta$ no banco de dados atual, temos que fazer a dedução de uma prova auxiliar, toda vez que aplicamos a regra R–6.$\Box$ ou a regra R–6.$\Diamond$.

De forma a minimizar o número de provas auxiliares propomos o seguinte mecanismo de implementação, baseado em técnicas de avaliação relaxada ("lazy evaluation" [Barendregt 84]). Primeiro, faremos com que os banco de dados entre cada estado e seus sucessores sejam referências para a mesma estrutura de dados em comum. O banco de dados criado pela regra R–6.$\supset$, será uma exceção, ou seja, ao incluirmos uma sentença em um banco de dados, estaremos criando uma nova instância do banco de dados atual, apenas as variáveis quantificadas existencialmente permanecem associadas entre o banco de dados anterior e o criado pela regra R–6.$\supset$. Na aplicação das regras R–6.$\Box$ e R–6.$\Diamond$, como os banco de dados gerados são totalmente diferentes do banco de dados anterior, eles também não serão representados pela mesma estrutura de dados.

Além do mecanismo descrito acima, teremos uma variável associada a cada premissa da forma $\alpha \supset \beta$ em um banco da dados. Esta variável inicialmente é deixada livre (sem valor associado). Ela pode passar a valer "V", quando o gol α já houver

sido provado a partir do banco de dados atual, ou "N", quando uma tentativa de provar α a partir do banco de dados atual houver falhado. Ao gerarmos o novo banco de dados pela regra R–6.$\supset$, esta variável é copiada, entretanto se ela vale "N" ou se ela está livre, a cópia passará a ser uma nova variável livre.

Com o mecanismo acima evitamos que uma mesma avaliação seja refeita várias vezes. Além disto, ao gerarmos um banco de dados genérico não avaliaremos o objetivo α imediatamente. Esta avaliação será postergada até que o termo β seja necessário na aplicação do método ("lazy evaluation"). Ou seja, junto à cada sentença β no banco de dados genérico BD', que foi incluida por uma premissa da forma $\alpha \supset \Box\beta$, haverá uma referência para um estado $E = (BD, G_0) \vdash \alpha$, que deve ser resolvido se desejamos utilizar a premissa β do banco de dados genérico BD'. O mesmo funcionamento é utilizado para as sentenças do banco de dados particular que foram tomadas do banco de dados genérico.

Esta implementação não só permite alguma eficiência ao método como também resolve alguns problemas de completude. Vejamos o cálculo do banco de dados genérico abaixo:

$$BD = \{\Box p \supset \Box q, \Box p\}$$

Seguindo a descrição teórica, como existe uma sentença $\Box p \supset \Box q$, no banco de dados BD, devemos incluir q no banco de dados genérico se pudermos provar $(BD, G_0) \vdash \Box p$. Entretanto, para realizar esta prova auxiliar precisaremos calcular outro banco de dados genérico. Ou seja, entramos em ciclo.

Já pela nossa implementação, dado BD acima, geramos imediatamento o banco BD' abaixo, com a presença condicional de premissas:

$$BD' = \{q \text{ (se } (BD, G_0) \vdash \Box p), p\}$$

Para resolver a prova auxiliar acima, caso seja necessário usar a premissa q do banco de dados genérico, aplicamos os seguintes passos:

1. $(\{\Box p \supset \Box q, \Box p\}, G_0)) \vdash \Box p$
2. $(\{q \text{ (se } (BD, G_0) \vdash \Box p), p\}, p) \vdash p$ .R–6.$\Box$
3. $(\{q \text{ (se } (BD, G_0) \vdash \Box p) p\}, p) \vdash true$ .R–6.1

■

Embora ainda possamos construir um banco de dados, tal que o algoritmo de construçao do banco de dados genérico entre em ciclo, diminuimos, com a nossa implementação, o número destes casos. Logo, aumentamos o poder do método.

Para o cálculo do banco de dados particular, a abordagem sugere uma técnica que será apresentada aqui modificada de forma a usar avaliação relaxada também. O

problema a ser solucionado reside no fato que o cálculo do banco de dados particular pode ser feito muitos passos antes da possível falha ocasionada pelo escolha da premissa δ errada. Uma forma de evitar que a execução tenha que retroceder até o cálculo do banco de dados particular é incluir neste banco, durante a sua construção, uma estrutura de dados que contenha todas as possíveis premissas do tipo δ. Uma vez tomada uma premissa desta estrutura todas as outras são invalidadas até que um possível retrocesso faça com que seja tentada outra premissa. Ou seja, no lugar de retrocedermos até o cálculo do banco de dadas particular, iremos retroceder apenas até o uso de uma sentença do tipo δ.

As sentenças incluídas na estrutura mencionada acima também usarão a avaliação relaxada. Ou seja, para as premissas da forma $\alpha \supset \Diamond\beta$ no banco de dados, incluiremos a premissa β na estrutura condicionada a dedução do estado $(BD, G_0) \vdash \alpha$.

A abordagem original apresenta uma proposta de modificação que usa um mecanismo parecido com este e que produz os mesmos resultados, mas é extremamente menos eficiente. Pela proposta de Quintana, durante a construção de um banco de dados genérico ou de um banco de dados particular, ao encontrarmos termos da forma $\alpha \supset \Diamond\beta$ ou $\alpha \supset \Box\beta$ no banco de dados atual, incluímos o termo β no novo banco de dados e prosseguimos com a prova. Ao terminamos este ramo da prova verificamos para cada fórmula β do novo banco de dados que foi utilizada durante a prova se a sua subfórmula α correspondente realmente pode ser deduzida do banco de dados atual. Como pode ser percebido, com este mecanismo podemos prosseguir por um ramo da prova, que pode ser extremamente grande, e chegar ao final a conclusão que este ramo não deveria ter sido tentado.

VI.4 Conclusões Parciais.

Os fundamentos teóricos nos quais este método está baseado não são de forma nenhuma de fácil compreensão em um primeiro contato. Pelo contrário, em uma primeira visão do método é comum ter-se a impressão que o método provaria não só as fórmulas válidas, mas também todas as outras. Isto ocorre porque o método ora trabalha dedutivamente, ora trabalha com redução ao absurdo, mas esta diferenciação não é notada durante a aplicação do método. Entretanto, mostramos cuidadosamente que todas as regras são corretas.

Embora a compreensão da base teórica por trás do funcionamento geral não seja trivial, a modificação do método para se adequar aos vários sistemas da lógica modal não oferece maiores problemas, em relação as demais abordagens. A principal

modificação é feita nas definições de banco de dados genérico e particular, que devem refletir características da relação de acessibilidade. Esta modificação nos parece ser relativamente fácil para cada sistema, embora, assim como nas outras abordagens, caberá ao modificador a validação das modificações realizadas.

A outra modificação necessária é a revisão das regras R–6.□ e R–6.◇ para cada sistema. Esta revisão já exige um maior cuidado. Quintana apresenta as modificações para os principais sistemas em sua dissertação de doutorado [Quintana 88a], que também são incompletas.

Já do ponto de vista de implementação as sugestões apresentadas na apresentação original da abordagem não são suficientes para garantir a efetividade e alguma eficiência na implementação do sistema. A nossa modificação, usando técnicas de avaliação relaxada, proporcionaram ao método uma eficiência razoável. Uma avaliação do método original por uma pessoa que não conheça avaliação relaxada iria classificar a abordagem como, no mínimo, pouco prática.

Mesmo em uma visão mais prática, consideramos que os problemas de incompletude realmente devem ser analisados com base em outras modificações propostas ao método [Zaverucha 91] de forma a proporcionar ao sistema uma maior aplicabilidade.

Por último, temos duas grandes contribuições a destacar nesta abordagem. A primeira é o uso de uma linguagem para expressar as sentenças que permite a imediata identificação de que regra de inferência deve ser usada. A segunda é a extensão feita ao método de tratamento de variáveis de Konolige, permitindo a maior flexibilidade no tratamento de domínios diferentes entre as propostas analisadas por nós.

Capítulo VII

Conclusões.

O objetivo básico deste trabalho é a comparação das abordagens apresentadas. Esta comparação é feita segundo o ponto de vista de alguns requisitos. Antes de apresentar a comparação propriamente vamos resumir o que cada requisito estará representando neste trabalho.

VII.1 Resumo dos Requisitos.

Os requisitos e suas definições são os seguintes:

Completude: Consideramos este requisito como a mais importante característica de comparação entre as abordagens. Ele terá para nós dois significados, sutilmente diferentes. Para as abordagens que funcionam dedutivamente, consideramos que a abordagem é completa se ela for capaz de gerar uma prova para todas as sentenças válidas. Já para as abordagens que trabalham com redução ao absurdo, usamos o conceito de completude refutacional, consideramos que uma abordagem é completa se ela for capaz de gerar uma refutação para todas as sentenças insatisfatíveis. No caso de abordagens que não são completas, estaremos considerando que uma abordagem é "mais completa" que outra se a primeira for capaz de gerar deduções para um conjunto de problemas que inclue o conjunto de problemas para o qual a segunda abordagem é capaz de gerar deduções. Note que poderemos ter duas avaliações de completude para o mesmo método, a primeira considerará o método como apresentado em seus fundamentos teóricos. Por outo lado, visto que a maioria dos métodos não pode ser perfeitamente implementado na forma que eles são propostos, teremos uma segunda avaliação da completude em cada método, que considerará as restrições que foram feitas ao método de forma a permitir sua implementação.

Política de Busca de Soluções: Neste requisito, estamos interessados em como se apresenta o espaço de soluções de cada método e como o método percorre

este espaço.

Efetivação: Esta característica refletirá o quanto cada proposta realmente se aproxima de um método que pode ser satisfatoriamente implementado com os recursos computacionais disponíveis atualmente.

Eficiência: Observe que neste ponto não estaremos interessados em uma análise detelhada de desempenho em parâmetros como tempo de execução, uso de memória etc, estaremos apenas interessados em uma comparação qualitativa com relação a quantidade de passos necessários para se completar uma prova, levando em consideração todos os possíveis retrocessos e necessidade de processamento em paralelo. Mesmo esta comparação também será bastante intuitiva, pois como os métodos são muitos diferentes, o custo de um passo em cada um deles varia muito.

Expressividade: Neste requisito estaremos analisando que restrições cada método impõe na utilização de toda a expressividade permitida pela lógica modal. Deve ser chamada a atenção para dois tipos de restrição diferentes: uma que exige a reescrita das sentenças, e outra que proibe o uso de determinadas construções (o uso de quantificadores no escopo de operadores modais, por exemplo). O primeiro tipo não chega a ser uma restrição, pois qualquer tipo de reescrita pode ser implementado automaticamente, inclusive sem o conhecimento do usuário. O segundo tipo de restrição é o que realmente estará no nosso interesse.

Adaptabilidade: Aqui estaremos interessados em analisar o quão fácil é modificar cada método, de forma que o usuário possa tirar o melhor proveito do método para aplicações específicas.

VII.2 Resumo das Abordagens.

Agora resumiremos cada abordagem estudada, evidenciando seu funcionamento, suas vantagens e restrições e estabelecendo como cada uma se apresenta segundo os requisitos estabelecidos.

Com o intuito de poupar espaço e facilitar a leitura passaremos a referenciar cada abordagem pelo número de ordem de sua apresentação, ou seja: abordagem 1: *"Modal Theorem Proving"* de Martín Abadi e Zohar Manna, abordagem 2: *"Temporal Logic Programming"* de Martín Abadi e Zohar Manna, abordagem 3: *"Resolution and Quantified Epistemic Logic"* de Kurt Konolige, e abordagem 4: *"Automatic Reasoning for Modal Logics: A Natural Deduction Based Approach"* de Pablo de la Quintana.

VII.2.1 Abordagem 1.

Esta abordagem se destina a lógica modal em geral e sua única restrições é que todos os mundos obrigatoriamente possuem o mesmo domínio. Sua aplicação é baseada em refutação, e consiste basicamente no uso de regras que reescrevem a negação da fórmula sendo provada até obter a fórmula *false*. Esta regras representam axiomas, equivalências e formas de se obter conseqüências lógicas.

Uma vantagem do método é permitir que a fórmula sendo provada, ou refutada, possa ser expressa no início da aplicação sem nehuma modificação. Entretanto, esta abordagem só é eficientemente implementada se o sistema reescrever internamente as fórmulas em uma forma que se aproxima da forma clausal.

Como desvantagem, o principal problema que encontramos neste método é a aplicação da regra do corte, que permite reescrever uma fórmula, acrescentando em uma de suas conjunções um termo formado pela disjunção de uma subfórmula qualquer e sua negação. Esta regra apresenta dois problemas: quando aplicá-la e, ao aplicá-la qual subfórmula acrescentar. Consideramos que estas questões não podem ser respondidas atualmente de forma satisfatória por um sistema completamente automático. Logo, nesta abordagem, temos a contraposição entre efetivação da implementação e completude do método.

Completude: Na apresentação teórica do método ele é completo (refutacionalmente), entretanto, se retirarmos a regra do corte, para permitir uma implementação completamente automática, a completude não pode ser verificada.

Política de Busca de Soluções: Para o sistema S4, ou outros sistemas mais simples, o método pode ser satisfatoriamente implementado com busca em profundidade com retrocesso, embora necessite ser incorporado ao método mecanismos de teste de recorrência não previstos na apresentação original. As regras específicas para o sistema S5 só podem ser implementadas com busca em largura, pois é muito difícil decidir se a aplicação destas regras realmente contribuirão para a solução do problema, caso contrário elas fatalmente farão a implementação entrar em ciclo infinito.

Efetivação: Sem a regra do corte, e com os testes de recorrência a proposta é perfeitamente implementável.

Eficiência: Do ponto de vista de eficiência a abordagem possibilita uma implementação bastante razoável quando comparada aos outros métodos. Em nossa implementação não codificamos todos os testes necessários à aplicação da regra de resolução. Estes testes encontram-se detalhadamente descritos no final da seção III.3. O impacto a ser causado pela implementação destes testes não devem comprometer

de forma significante a eficiência do método.

Expressividade: Do ponto de vista de expressividade o único problema que o método apresenta é a restrição de apresentar o mesmo domínio em todos os mundo. Fora isto, toda a expressividade da lógica modal é aceita por este método.

Adaptabilidade: Quanto à adaptabilidade, para modificar o método para um sistema específico precisam ser fornecidas as regras de reescrita específicas para este sistema. Dado um conjunto de regras para um sistema específico, caberá ao modificador do sistema garantir que estas regras são corretas e garantem a completude esperada.

VII.2.2 Abordagem 2.

Esta abordagem se destina a lógica temporal linear, ela consiste em uma extensão da linguagem Prolog, restringindo assim a sua aplicação ao tipo de problemas que podem ser tratados por esta linguagem (ou seja, uso de cláusulas de Horn). Além disto, só são admitidos os operadores temporais lineares $\bigcirc$, $\Diamond$ e $\Box$, sendo que o $\Diamond$ só pode ser usado no corpo das cláusulas e o $\Box$ só pode ser usado na cabeça. O fato de os domínio terem que ser os mesmos em todos os mundos é bastante razoável em lógica temporal.

O processo de dedução é baseado em uma extensão do mecanismo tradicional de resolução para trabalhar com os operadores temporais lineares. Embora isto seja por uma lado restrito, este funcionamento garante as principais vantagens do método: sua eficiência e sua simplicidade. Como desvantagem temos a obrigatoriedade de expressar nossas fórmulas em uma forma similar à cláusulas de Horn.

Completude e Política de Busca de Soluções: A completude e a política de busca de soluções neste método são análogas às apresentadas pela linguagem Prolog, cujos problemas são bem conhecidos.

Efetivação e Eficiência: Como uma extensão ao método de resolução usado na implementação do Prolog, a abordagem também apresenta a efetivação e eficiência comparáveis às daquela linguagem.

Expressividade: Quanto a expressividade, o sistema está limitado pela linguagem aceita, ou seja, cláusulas de Horn com os operadores $\bigcirc$, $\Diamond$ e $\Box$.

Adaptabilidade: Pelo que foi discutido nas seções IV.4 e IV.5 é praticamente impossível realizar qualquer extensão significativa neste método, sem necessitar modificar o método completamente. Queremos dizer com isto que o funcionamento do metodo está apoiado em características específicas da lógica temporal linear, qualquer tentativa de estendê-lo para outras lógicas que não apresentem estas ca-

racterísticas invalidariam o funcionamento básico do próprio método.

VII.2.3 Abordagem 3.

Esta abordagem se destina a lógica modal, e sua apresentação original só apresenta o mecanismo de dedução para o sistema modal K. São aceitos domínios diferentes para os mundos, desde que seja preservada a validade da reversa da fórmulas de Barcan (ou seja, o domínio de cada mundo acessado é um subconjunto do mundo atual). Embora a apresentação afirme que estas restrições possam se eliminadas consideramos que isto envolverá um trabalho extremamente grande. Outra restrição é a necessidade de se expressar as fórmulas em uma forma análoga a forma clausal.

A aplicação do método consiste em, dado um conjunto de premissas, gerar conseqüências lógicas destas premissas, que são incluídas no conjunto original, até que seja possível gerar a cláusula vazia como conseqüência lógica, indicando a insatisfatibilidade do conjunto original.

A principal vantagem deste método é sua completude aliada a expressividade das fórmulas aceitas. A principal desvantagem, entretanto, é que o método não é efetivamente implementável. Ou melhor, a sua implementação exige recursos computacionais muitíssimo grandes, devido à necessidade da busca de soluções se processar em largura e massivamente paralela. (Neste caso não existe uma contraposição evidente entre completude e efetividade, mas podemos assumir que as características que levam este método ser completo também o tornam não implementável com os nossos recursos disponíveis.)

Completude: Logo, do ponto de vista do requisito de completude temos duas avaliações, em uma visão teórica deste método temos que ele é completo. Entretanto, do ponto de vista de implementação, não podemos considerar esta abordagem como um método voltado para implementação. A complexidade de realizar esta implementação é comparável a complexidade de se implementar automaticamente uma dedução natural.

Política de Busca de Soluções: Para este método a única avaliação que podemos fazer neste requisito é que não é apresentada uma política eficaz de busca de soluções, o que faz com que todas as possíveis alternativas tenham que ser percorridas paralelamente.

Efetivação: Logo, do ponto de vista de implementação, não podemos considerar esta abordagem como um método voltado para implementação, dado os recusos que podemos dispor atualmente.

Eficiência: Neste contexto, consideramos este metodo como extremamente ine-

ficiente.

Expressividade: Quanto à expressividade o principal problema é a necessidade dos domínios dos mundos precisarem garantir a validade da reversa da fórmula de Barcan.

Adaptabilidade: Quanto à adaptabilidade, redefinir o mecanismo de vistas para cada sistema modal não é uma tarefa nem um pouco trivial. Além do que o mecanismo resultante para sistemas complexos, como o S5, poderá ser tão ou mais complexo que a implementação direta do método teórico.

VII.2.4 Abordagem 4.

Esta abordagem se destina a lógica modal em geral e sua restrição é que tanto as premissas como a fórmula sendo provada precisam ser reescritas em uma linguagem própria.

A aplicação do método é de certa forma similar a uma resolução Prolog, a principal diferença é que no lugar de existir apenas uma regra de inferência existem várias. A linguagem em que as fórmulas são expressas permite a identificação imediata de que regra deve ser aplicada a cada instante.

A principal vantagem do método é permitir o tratamento de modelos com domínios diferentes em cada mundo. Outro ponto notável nesta abordagem é sua elegância e simplicidade. Como principal desvantagem, além das restrições já mecionadas, temos a pouca eficiência de sua implementação, principalmente quando comparada por exemplo ao Prolog, pois algumas das regras de inferência desta abordagem são não determinísticas[1].

Completude: Como dissemos na apresentação desta abordagem ela apresenta sérios problemas de completude, estudos realizados por outros autores [Zaverucha 91] demostraram que estes problemas poderiam ser resolvidos, entretanto, como estas correções não foram publicadas, preferimos nos deter neste trabalho na apresentação original desta abordagem.

Política de Busca de Soluções: Quanto ao problema de busca de soluções esta abordagem pode em algumas aplicações gerar um espaço de busca muito largo, entretanto, na maioria dos casos conseguimos varrer estes espaços com uma busca em profundidade com retrocesso. Os demais casos ocasionam ciclo infinito independente da largura do espaço de soluções.

Efetivação e Eficiência: Já o requisito de efetivação da implementação, co-

[1]Dizemos que uma regra é não determinística quando a sua aplicação cria um ponto de retrocesso.

mo dissemos na apresentação da abordagem, temos duas avaliações. Pela proposta original, que não define precisamente nenhum mecanismo de implementação, a abordagem seria de implementação bastante inificiente. Com a nossa proposta de usar técnicas de avaliação relaxada na implementação conseguimos obter uma sistema razoavelmente eficiente.

Expressividade: Quanto a expressividade da linguagem aceita o único problema, que já foi comentado, é a proibição de quantificadores universais na fórmula sendo provada, mas que pode ser contornado.

Adaptabilidade: Do ponto de vista de adaptabilidade temos duas considerações a fazer: (1) Se a adaptação a ser feita for apenas relativa a relação de acessibilidade, então ela consiste em redefinir a construção dos bancos de dados genérico e particular e a aplicação das regras para operadores modais, por outro lado, se for necessário outro tipo de alteração, as modificações a serem feitas não são claras. (2) Mesmo considerando apenas modificações que alterem somente a relação de acessibilidade, o sistema obtido não apresentará as mesmas características de eficiência e será bem mais complexo que a implementação apresentada para o sistema K.

VII.3 Comparação Segundo os Requisitos.

Iremos comparar agora cada uma das abordagens apresentadas segundo os requisitos estabelecidos:

Completude: Temos duas propostas facilmente classificáveis: a abordagem 2 não apresenta maiores problemas e sua completude é igual à apresentada pela linguagem Prolog, a abordagem 4 apresenta problemas de completude facilmente identificáveis, assim como pudemos ilustrar durante a apresentação deste método. Já para as abordagens 1 e 3, embora estes métodos apresentem a completude em sua apresentação teórica, os sistemas implementados a partir deles não são completos. Fica aqui então respondida a principal questão tratada neste trabalho: ainda não dispomos (entre as propostas analisadas) de um método de dedução para lógica modal, voltado para implementação, que apresente a completude.

Política de busca de Soluções: Neste caso também temos duas abordagens facilmente classificáveis: a abordagem 2 se comporta exatamente como a linguagem Prolog, que tem um modelo de busca de soluções bem estudado, e a abordagem 3, não só apresenta um espaço muito largo, como cada um de seus ramos e subramos devem ser percorridos em paralelo. A abordagem 1 pode ser resolvida com percurso em profundidade com retrocesso para os sistemas mais simples, entretanto, para o sistema S5, a única implementação possível é a busca em largura. Já a abordagem

4, embora possa gerar um espaço razoavelmente largo e possa provocar muitos retrocessos, pode ser implementada satisfatoriamente com uma busca em profundidade com retrocesso.

Efetivação: Enquanto a abordagem 2 é perfeitamente implementável, as demais só são implementadas ao custo de algumas retrições: na abordagem 1 abandonamos a regra do corte e na abordagem 3 e 4 restringimos o método à aplicação em modelos do sistema K.

Eficiência: A abordagem 2 apresenta um eficiência comparável à linguagem Prolog. A abordagem 4, com a nossa proposta de implementação usando técnicas de avaliação relaxada, apresenta uma eficiência razoável, assim como a abordagem 1, quando comparadas às demais. Quanto à abordagem 3, ela é a mais ineficiente.

Expressividade: Consideramos aqui que o fato de precisarmos reescrever as sentenças da lógica modal em algumas abordagens não é uma desvantagem, pois isto sempre pode se feito automaticamente. Neste contexto temos duas considerações a fazer como principais diferenças de expressividade entre as abordagens: a liberdade do uso de quantificadores nas sentenças e a possibilidade de definição de diferentes domínios para cada mundo. No caso da abordagem 1 ela peca no segundo ponto. A abordagem 4 peca no primeiro, enquanto a abordagem 2 em ambas, embora, como dissemos antes, o segundo ponto não é importante em se tratando de lógica temporal. Já a abordagem 3 é a que permite a melhor expressividade.

Adaptabilidade: Neste ponto todas as abordagems vistas apresentam deficiências. A abordagem 2 não parece poder ser satisfatoriamente modificada. Para as demais abordagens, embora seja fácil visualizar as alterações a serem feitas, é bastante difícil provar que tais alterações não comprometem os demais requisitos, como completude, efetivação, e eficiência.

Como podemos ver algumas abordagens se destacam em alguns requisitos mas são deficientes em outros. Para uma avaliação mais precisa de qual abordagem deve ser usada em cada caso precisamos antes conhecer detalhadamente os tipos de problemas que estamos querendo modelar. Por exemplo, se queremos uma modelagem em S4, com domínios iguais, podemos usar perfeitamente a abordagem 1. Por outro lado, se queremos tratar modelos em K, com domínios diferentes, podemos usar a abordagem 4. É importante notar que nenhuma delas atende perfeitamente ao objetivo de oferecer um método completamente automático para dedução de lógica modal de primeira ordem. Acreditamos ter levantado, ao longo deste trabalho as principais vantagens e deficiências de cada abordagem.

VII.4 Observações Adicionais Sobre as Abordagens de Abadi e Manna.

As duas abordagens de Abadi e Manna, isto é, as abordagens 1 e 2, possuem alguns pontos em comuns: ambas usam setas simples e duplas ($\longrightarrow$ ou $\longleftarrow$ e $\Longrightarrow$ ou $\Longleftarrow$) para denotar regras ou sentenças de tipos diferentes, em ambas existe o conceito de símbolos rígidos e em ambas os domínios de todos os mundos são necessariamente os mesmos. No mais elas são muito diferentes.

A abordagem 1 se destina à lógica modal em geral, enquanto a 2 se destina a lógica temporal. A primeira é baseada em refutação e suas regras geram uma conseqüência lógica da fórmula atual, a segunda é uma extensão do método de resolução. A abordagem 1 contém algumas regras que tiram proveito do fato de existirem predicados rígidos, a abordagem 2 implementa algumas de suas regras de inferência usando as propriedades de predicados rígidos.

Embora cada uma das duas propostas sejam baseadas em dois tipos diferentes de regras, podemos observar que a diferenciação de um tipo para o outro é completamente distinta nas duas abordagem. Na abordagem 1 um tipo de regras substitui termos em uma conjunção, enquanto o outro tipo apenas acrescenta termos à uma conjunção. Na abordagem 2 temos um tipo de regra que só se aplica ao instante inicial, enquando o outro pode ser usado em qualquer instante de tempo.

Logo, o que precisa ficar bem claro é que a segunda abordagem não se trata se uma adaptação ou extensão da primeira, qualquer que seja o ponto de vista, obrigatoriamente temos que considerá-las como duas propostas totalmente diferentes.

Deste forma, acreditamos que estas abordagens possam ser classificadas da seguintes forma. A abordagem 1, para lógica modal, é um método genérico, de implementação pouco eficiente e que só apresenta a completude se a aplicação for guiada pelo usuários ou através de heurísticas. Sendo que a determinação destas heurísticas um problema completamente em aberto. A abordagem 2, para lógica temporal linear, é um método extremamente específico, que estende a linguagem Prolog e tem sua eficiência de execução similar a esta linguagem.

Logo, concluímos que o objetivo de apresentar um método satisfatoriamente implementável não foi conseguido pela abordagem 1, o que deve ter levado os autores passar a considerar problemas mais restritos, como a lógica temporal linear, e neste novo contexto realmente conseguiram um ótimo sistema.

VII.5 Conclusões Gerais.

Como pudemos notar ao longo deste trabalho, os métodos para automação de lógica modal de primeira ordem não são sistemas completamente fechados, eles se apresentam mais como arcabouços, esquemas ou protótipos, que o usuário deve moldar, de forma a tirar o melhor proveito do sistema para os seus problemas específicos.

Além disto, podemos notar que, ao contrário de sistemas para lógica clássica de primeira ordem, os métodos para lógica modal não são puramente sintáticos. Ou seja, enquanto a definição da aplicação de um método para lógica clássica pode ser feita exclusivamente baseada na sintaxe usada para modelar o problema, em lógica modal, temos várias informações semânticas, como existência de símbolos rígidos ou características da relação de acessibilidade, que são essenciais durante a aplicação dos métodos.

Uma forte contraposição também identificada nos métodos analisados foi a entre a completude e efetivação das abordagens. Acreditamos que deva existir uma restrição (assim como cláusulas de Horn são para lógica clássica) que possibilite a definição de sistemas completamente automáticos. Entretanto, a definição de tal subconjunto da lógica modal é um problema em aberto.

Em suma, não acreditamos que o ideal possa ser alcançado a curto prazo, mas identificamos um grande volume de esforço neste sentido atualmente. E podemos dizer que, se nehuma destas abordagem é realmente a implementação ideal, elas têm a qualidade de contribuir com características que nos conduzem na procura de uma proposta com uma melhor relação entre completude e efetivação.

VII.6 Futuro.

Como este trabalho tinha como objetivo comparar quatro abordagens, que julgamos serem representativas do estágio atual de pesquisa na área de automação de lógica modal, não pudemos nos deter com maior profundidade em nenhuma delas. Desta forma, deixamos aqui como sugestão de trabalhos futuros, a continuidade do estudo de cada abordagem, analisando profundamente os problemas encontrados em cada uma, compilando as sugestões apresentadas neste trabalho e analisando os pontos que não pudemos considerar.

Por exemplo, a abordagem 3 foi classificada por nós como não implementável com os recursos computacionais que dispomos, principalmente por falta de sugestões de estratégias. Entretanto, poderia ser feito um trabalho de complementação das idéias de Konolige, de forma a fechar um sistema, e analisar como este sistema po-

deria ser implementado nas novas arquiteturas paralelas que estão sendo estudadas atualmente.

Mesmo nas demais abordagens, muito trabalho ainda ficou por ser feito, principalmente na adaptação destas abordagens para sistema modais mais complexos, ou para problemas específicos. Uma pesquisa interessante seria verificar se as modificações da abordagem 4, para sistemas mais complexos, também se beneficiariam das técnicas de avaliação relaxada, ou seria necessário mecanismos mais poderosos para estas extensões.

Outros estudos que poderiam ser feitos são na área de análise de complexidade, determinando a complexidade de cada abordagem e apontando a mais eficiente. Também poderiam ser feitas tentativas de ser definir métodos adequados para lógica temporal não linear e lógica temporal de intervalos. Outro campo de pesquisa não explorado foram as adaptações dos vários métodos para arquiteturas paralelas.

Em suma, acreditamos ter dado alguns passos, ensaiados outros, considerado vários caminhos, mais ainda há muito o que percorrer nesta linha de pesquisa

Bibliografia

[Abadi 86] M. Abadi e Z. Manna, *"Modal Theorem Proving"*, Proceedings of the 8^{th} International Conference on Automated Dedution, Oxford, 1986, pp. 172-189.

[Abadi 89] M. Abadi e Z. Manna, *"Temporal Logic Programming"*, Journal of Symbolic Computation, volume 8, número 3, 1989, pp. 277-285.

[Barendregt 84] H. P. Barendregt, *"Studies in Logic and the Foundations of Mathematics - volume 103: The Lambda calculus - Its Syntax and Semantic"*, North-Holland, 1984, 621 p.

[Casanova 87] M. A. Casanova, F. A. C. Giorno e A. L. Furtado, *"Programação em Lógica e a Linguagem Prolog"*, Edgard Blücher, São Paulo, 1987, 461 p.

[Clocksin 87] W. F. Clocksin e C. S. Mellish, *"Programming in Prolog"*, Springer-Verlag, Berlim, 1987, 281 p.

[Cohen 87] J. Cohen e T. J. Hickey, *"Parsing and Compiling Using Prolog"*, ACM Transactions on Programming Languages and Systems, volume 9, número 2, New York, 1987, pp. 125-163.

[delCerro 86] L. F. del Cerro, *"MOLOG: A System that extends PROLOG with Modal Logic"*, New Generation Computing, número 4, 1986, pp. 35-50.

[Elcook 90] E. W. Elcook, *"ABSYS: The First Logic Programming Language - A Restropective and a Commentary"*, The Journal of Logic Programming, volume 9, número 1, North-Holland, 1990, pp. 1-18.

[Enjalbert 89] P. Enjalbert e L. F. del Cerro, *"Modal Resolution in Clausal Form"*, Theoretical Computer Science, número 65, North-Holland, 1989, pp. 1-33.

[Gabbay 84] D. M. Gabbay e U. Reyle, *"N-Prolog: An Extension of Prolog with Hypothetical Implication"*, Journal of Logic Programming, volume 4, 1984, pp. 319-355.

[Gabbay 85] D. M. Gabbay, *"N-Prolog: An Extension of Prolog with Hypothetical Implication 2 - Logical Foundations and Negation as Failure"*, Journal of Logic Programming, volume 4, 1985, pp. 251-283.

[Gabbay 86] D. M. Gabbay e F. Kriwaczek, *"A Goal Directed Theorem Prover for Intuitionistic and Classic Logic Based on Conjunction and Implication"*, Technical Report, Imperial College, 1986.

[Geissler 86] C. Geissler e K. Konolige, *"A Resolution Method for Quantified Modal Logic of Knowledge and Belief"*, Proceedings of Theoretical Aspect of Reasoning about Knowledge (J. Halpern ed.), Morgan Kaufmann Publishers, Palo Alto, 1986, pp. 309-324.

[Halpern 86] H. Y. Halpern e Y. Shoham, *"A Propositional Modal Logic of Time Interval"*, Proceedings of the Symposium on Logic in Computer Science, Cambridge, 1986, pp. 279-292.

[Hughes 68] G. E. Hughes e M. J. Cresswell, *"An Introduction to Modal Logic"*, Methuen and Co Ltd, London, 1968, 388 p.

[Konolige 86] K. Konolige, *"Resolution and Quantified Epistemic Logics"*, Proceedings of the 8^{th} International Conference on Automated Deduction, Oxford, 1986, pp. 199-208.

[Kowalski 71] R. A. Kowalski e D. Kuehner, *"Linear Resolution with Selection Function"*, Journal of Artificial Intelligence, volume 2, 1971, pp. 227-260.

[Kowalski 74] R. A. Kowalski, *"Predicate Logic as a Programming Language"*, Proceedings of IFIP 74, North-Holland, 1977, pp. 569-574.

[Lamport 83] L. Lamport, *"Specifying Concurrent Program Modules"*, ACM Transaction on Programming Languages and Systems, volume 5, 1983, pp. 190-222.

[Manna 74] Z. Manna, *"Mathemathical Theory of Computation"*, McGraw-Hill Book Conmpany, New York, 1974, 448 p.

[McDermott 82] D. McDermott, *"Nonmonotonic Logic II: Nonmonotonic Modal Theories"*, Journal of the ACM, volume 29, número 1, 1982, pp. 33-57.

[Murray 82] N. V. Murray, *"Completely Non-Clausal Theorem Proving"*, Artificial Intelligence, volume 18, número 1, North-Holland, 1982, pp. 67-85.

[Quintana 88a] P. J. de la Quintana Brüggemann, *"Automated Reasoning for Modal Logics: A Natural Deduction Based Approach"*, Ph.D. thesis, University of London, 1988, 172 p.

[Quintana 88b] P. J. de la Quintana, *"Computing Quantifiers in Predicate Modal Logics"*, versão preliminar de artigo aceito no ECAI 88, 1988, 13 p.

[Quintana 88c] P. J. de la Quintana, *"Bridging the Gap Between Modal Logic and Logic Programming"*, versão preliminar, 1988, 7p.

[Robinson 65] J. A. Robinson, *"A Machine Oriented Logic Based on the Resolution Principle"*, Journal of ACM, volume 12, número 1, 1965, pp. 23-41.

[Stickel 85] M. E. Stickel, *"Automated Deduction by Theory Resolution"*, Proceedings of the 9^{th} International Joint Conference on Artificial Intelligence, Pittsburg, 1985.

[Zaverucha 89] G. Zaverucha, *"Konolige's Deduction Model of Belief (Viewed Through on Extension of Quintana's Theorem Prover)"*, lecture notes, Imperial College Seminar, 1989.

[Zaverucha 91] G. Zaverucha, comunicação pessoal.

Anexo A

Lógica Clássica de Primeira Ordem.

Incluímos neste apêndice a definição da sintaxe que é usada ao longo do texto para lógica clássica de primeira ordem. Também é apresentada a semâtica associada a cada símbolo. Além disto, apresentamos alguns conceitos relativos à lógica clássica que usamos ao longo do texto. Aconselhamos que este apêndice seja lido antes do capítulo II, de forma a tornar o leitor familiarizado com as definições e conceitos aqui apresentados.

A.1 Lógica Clássica de Primeira Ordem.

Lógica de primeira ordem é uma linguagem formal [Manna 74] cujos componentes e regras de formação definimos a seguir. Usaremos a denominação fórmula para designar as cadeias da linguagem da lógica de primeira ordem, e usaremos a denominação subfórmula para designar uma fórmula que faz parte, ou seja, é uma subcadeia, de uma fórmula maior.

A.1.1 Sintaxe.

O alfabeto da lógica de primeira ordem é composto por símbolos de pontuação, símbolos lógicos, que são: conectivos lógicos, quantificadores e variáveis; e por símbolos não-lógicos, que são: constantes, símbolos funcionais e símbolos predicativos. Cada um destes subconjuntos é definido abaixo:

símbolos de pontuação:

- parênteses direito e esquerdo: (e)
- vírgula: ,

símbolos lógicos:

- conectivos lógicos:
  - conectivo de negação: $\neg$
  - conectivo de conjunção: $\vee$
  - conectivo de disjunção: $\wedge$
  - conectivo de implicação: $\supset$
- quantificadores:
  - quantificador universal: $\forall$
  - quantificador existencial: $\exists$
- variáveis: Um conjunto infinito de símbolos distintos dos demais. Para nós os elementos deste conjunto poderão ser as letras x, y e z, estas letras subscritas (e.g. x_0, x_1, x_2), e estas letras seguintas de apóstrofos (e.g. x, x', x'').

símbolos não-lógicos:

- constantes: Um conjunto de símbolos distintos dos demais. Para nós os elementos deste conjunto poderão ser as letras iniciais do alfabeto (e.g. a,b,c,d,e), números (e.g. 0,1,2,3,4) ou palavras (e.g. riesling, cabernet, merlot).
- símbolos funcionais: Um conjunto de símbolos distintos dos demais. Para nós os elementos deste conjunto poderão ser as letras f, g ou h. A cada símbolo funcional estará associado um número natural maior que zero que será denominado de aridade do símbolo funcional.
- símbolos predicativos: Um conjunto de símbolos distintos dos demais. Para nós os elementos deste conjunto poderão ser as letras p, q, r, s ou palavras, eventualmente unidas por hífens (e.g. seco, meio-doce, doce, tinto, branco). A cada símbolo predicativo estará associado um número natural que será denominado de aridade do símbolo predicativo.

Denominaremos de termo os seguintes elementos:

- Toda variável é um termo (e.g. x_1).
- Toda constante é um termo (e.g. a, riesling).
- Se f é um símbolo funcional, onde n é a aridade de f, e se $t_1, \ldots, t_n$ é uma seqüência de n termos, então $f(t_1, \ldots, t_n)$ também é um termo (e.g. $f(a, h(b))$).

Denominaremos de fórmula os seguintes elementos:

- Se p é um símbolo predicativo, onde n é a aridade de p, e se $t_1, \ldots, t_n$ é uma seqüência de n termos, então $p(t_1, \ldots, t_n)$ é uma fórmula (e.g. $p(a, f(b))$, seco(cabernet)). Cada t_i será chamado de um argumento do símbolo predicativo. (Para símbolos predicativos de aridade 0 os parenteses que englobam os termos são suprimidos, e.g. p.) Este tipo de fórmula será chamado de fórmula atômica ou predicado.
- Se α e β são fórmulas, então $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$ e $(\alpha \supset \beta)$ também são fórmulas. (Note que ao longo deste texto usaremos letra gregas (e.g. α, β, γ) ou as letras maiúsculas A, B ou C para representar fórmulas e subfórmulas.)
- Se α é uma fórmula e x uma variável, então $\forall x(\alpha)$ e $\exists x(\alpha)$ também são fórmulas.

Dada a fórmula $(\neg\alpha)$ dizemos que a subfórmula α está no escopo de um conectivo de negação. Igualmente para as fórmulas $(\alpha \wedge \beta)$ ou $(\alpha \vee \beta)$ dizemos que α e β estão no escopo de um conectivo de conjunção ou de disjunção. Já para a fórmula $(\alpha \supset \beta)$, dizemos que α está no escopo esquerdo de um conectivo de implicação, enquanto β está no escopo direito do mesmo conectivo.

Se uma fórmula α está no escopo de determinado conectivo, dizemos que todas as possíveis subfórmulas formadoras de α também estão no escopo do mesmo conectivo. Por exemplo, na fórmula $((p \wedge q) \supset (p \vee q))$, a primeira ocorrência da subfórmula p está no escopo dos conectivos de conjunção e de implicação, neste último caso, no escopo esquerdo.

Com relação à formula $Qx(\alpha)$, onde Q é o quantificador $\forall$ ou o quantificador $\exists$, dizemos que α está no escopo do quantificador Q e que x é a variável quantificada por ele. Por abuso de linguagem, também denominaremos a seqüência Qx de quantificador, de forma a podermos especificar exatamente um quantificador dentro de uma fórmula.

A ocorrência de uma variável x é dita ligada se esta ocorrência está no escopo de um quantificador Q que quantifica a variável x. A ocorrência de uma variável x fora do escopo de algum quantificador que a quantifique é dita uma ocorrência livre. A palavra sentença será usada para designar fórmulas da lógica que só contenham ocorrência de variáveis ligadas.

A fim de não sobrecarregar graficamente as fórmulas, definimos precedências entre os conectivos lógicos, quantificadores e operadores modais (que serão definidos na seção II.1), eliminando assim os parênteses desnecessários. A ordem de precedência será a seguinte: operadores unários, conectivo de implicação, conectivo de conjunção e conectivo de disjunção. Os operadores unários são o conectivo de negação, os quantificadores (junto com a variável sendo quantificada, ou seja, a seqüência Qx) e os operadores modais.

Abaixo ilustramos o uso da eliminação de parenteses.

a fórmula:	pode ser expressa como:
$\forall x(\exists y((p(x) \wedge p(y))))$	$\forall x \exists y(p(x) \wedge p(y))$
$((p(a) \wedge p(b)) \vee p(c))$	$p(a) \wedge p(b) \vee p(c)$
$((p \wedge \exists y(q(y))) \supset \exists(z(p \vee q(z))))$	$p \wedge \exists y q(y) \supset \exists z(p \vee q(z))$

A.1.2 Semântica.

Cada símbolo lógico possui um significado intrínseco à própria lógica de primeira ordem. O mesmo não ocorre com os símbolos não-lógicos. O significado destes últimos depende da estrutura de primeira ordem que associamos à nossa linguagem. Uma estrutura de primeira ordem é o par $\langle D, I \rangle$, onde D é um domínio e I é a função de interpretação, que dá um significado para cada símbolo não-lógico. Ou seja, dado um alfabeto, para o conjunto de fórmulas que podemos gerar estará associado uma estrutura de primeira ordem $\langle D, I \rangle$, onde I é uma função que associa a cada constante, a cada símbolo funcional de aridade n_f e a cada símbolo predicativo de aridade n_p do alfabeto, respectivamente, um elemento de D, uma função de D^{n_f} em D e uma relação em D^{n_p}.

Para uma fórmula da lógica de primeira ordem, dada uma estrutura de primeira ordem $\langle D, I \rangle$ e uma atribuição V, que associa a cada variável de um conjunto L um elemento de D e dado que todas as variáveis livres da fórmula pertencem ao conjunto L, podemos avaliar esta fórmula como verdadeira ou falsa. Definiremos a função de avaliação v sobre $\langle D, I \rangle$ e V da seguinte forma:

- avaliação de termos

$v_t(x) = V(x)$, para as variáveis

$v_t(c) = I(c)$, para as constantes

$v_t(f(t_1, \ldots, t_n)) = I(f)(v(t_1), \ldots, v(t_n))$, para os símbolos funcionais

- avaliação de fórmulas

$v(p(t_1, \ldots, t_n)) =$ verdadeira, se $(v_t(t_1), \ldots, v_t(t_n)) \in I(p)$

$v(p(t_1, \ldots, t_n)) =$ falsa, caso contrário

$v(\neg\alpha) =$ verdadeira, se $v(\alpha) =$ falsa

$v(\neg\alpha) =$ falsa, se $v(\alpha) =$ verdadeira

$v(\alpha \wedge \beta) =$ verdadeira, se $v(\alpha) = v(\beta) =$ verdadeira

$v(\alpha \wedge \beta) =$ falsa, caso contrário

$v(\alpha \vee \beta) =$ falsa, se $v(\alpha) = v(\beta) =$ falsa

$v(\alpha \vee \beta) =$ verdadeira, caso contrário

$v(\alpha \supset \beta) =$ falsa, se $v(\alpha) =$ verdadeira e $v(\beta) =$ falsa

$v(\alpha \supset \beta) =$ verdadeira, caso contrário

$v(\forall x(\alpha)) =$ verdadeira, se para qualquer função de avaliação v' que definirmos, tal que v' difira de v no máximo no valor que é atribuído à variável x, tivermos que $v'(\alpha) =$ verdadeira.

$v(\forall x(\alpha)) =$ falsa, se podemos definir uma função de avaliação v', tal que v' difira de v no máximo no valor que é atribuído à variável x, e $v'(\alpha) =$ falsa.

$v(\exists x(\alpha)) =$ verdadeira, se existir pelo menos uma função de avaliação v', tal que v' difira de v no máximo no valor que é atribuído à variável x, e tivermos que $v'(\alpha) =$ verdadeira.

$v(\exists x(\alpha)) =$ falsa, se não existir uma função de avaliação v', tal que v' difira de v no máximo no valor que é atribuído à variável x, e $v'(\alpha) =$ verdadeira.

Definimos dois símbolos predicativos constantes de aridade zero: $true$ e $false$. Eles possuem a mesma avaliação para qualquer estrutura de primeira ordem que associamos à nossa linguagem. Temos que: $v(true)$ = verdadeira e $v(false)$ = falsa para todas as estruturas de primeira ordem. Usaremos a notação $\alpha = true$ para as fórmulas α, tais que $v(\alpha)$ = verdadeira para a estrutura de primeira ordem que estivermos usando. Da mesma forma, $\alpha = false$ denotará que a fórmula α é tal que $v(\alpha) = false$ para a estrutura de primeira ordem em uso.

Desta forma, podemos representar sinteticamente as definições para conectivos lógicos descritas acima através das seguintes tabelas:

α	$(\neg\alpha)$
true	false
false	true

α	β	$(\alpha \wedge \beta)$
true	true	true
true	false	false
false	true	false
false	false	false

α	β	$(\alpha \vee \beta)$
true	true	true
true	false	true
false	true	true
false	false	false

α	β	$(\alpha \supset \beta)$
true	true	true
true	false	true
false	true	false
false	false	true

Diremos que uma fórmula é válida quando a sua avaliação for verdadeira para qualquer função de avaliação, domínio e interpretação. Da mesma forma diremos que uma fórmula é inválida se existe um domínio, uma interpretação e uma função de avaliação, os quais a avaliação da fórmula é falsa.

Diremos que uma fórmula é satisfatível se existe um domínio, uma interpretação e uma função de avaliação para a qual a avaliação da fórmula é verdadeira. Da mesma forma diremos que uma fórmula é insatisfatível se não existe um domínio, uma interpretação e uma função de avaliação para os quais a avaliação da fórmula seja verdadeira.

Podemos notar que se α é uma fórmula válida, então $\neg(\alpha)$ é uma fórmula insatisfatível. E que se α é uma fórmula satisfatível, então $\neg(\alpha)$ é uma fórmula inválida.

Usaremos o símbolo $\models$ para indicar a validade de fórmulas. A notação $\models \alpha$ indicará que a fórmula α é válida. A notação $\alpha \models \beta$ indicará que a fórmula $(\alpha \supset \beta)$ é válida. E a notação $\alpha_1, \ldots, \alpha_n \models \beta$ será equivalente à $(\alpha_1 \wedge \ldots \wedge \alpha_n) \models \beta$.

A.2 Forma Clausal.

Algumas abordagens de automação de lógica exigem que as fórmulas sejam colocadas na forma clausal, outras não exigem. Um fórmula está na forma clausal se ela é uma sentença que só possui quantificadores universais, todos agrupados à esquerda da fórmula, e se o escopo dos quantificadores (i.e. a parte da fórmula que permanece se nós eliminarmos os quantificadores) se apresenta como uma conjunção de disjunções com as negações se aplicando apenas a fórmulas atômicas. Existem algoritmos [Casanova 87] que geram uma sentança S na fórmula clausal a partir de qualquer fórmula F, tal que não existe uma função de avaliação tal que que $v(S) \neq v(F)$.

Para nós é importante saber que as variáveis quantificadas existencialmente são eliminadas através da utilização de funções de Skolem. De forma resumida, substituiremos cada variável x quantificada existencialmente por uma nova função, que recebe como argumentos todas as variáveis quantificadas universalmente tais que x encontra-se no escopo dos quantificadores daquelas variáveis.

A.3 Cláusulas de Horn.

Uma cláusula de Horn é uma representação especial que podemos usar para determinadas sentenças. Se uma sentença está na forma clausal e ela se apresenta como uma disjunção de subfórmulas onde no máximo uma destas subfórmulas é uma fórmula atômica e todas as demais subfórmulas são negações de fórmulas atômicas, então esta sentença pode ser colocada na forma de cláusula de Horn.

Dada uma sentença do tipo

$$\forall x_1, \ldots, \forall x_n (A \vee \neg B_1 \vee \ldots \vee \neg B_m)$$

onde A e B_1 até B_m são fórmulas atômicas e x_1 até x_n são todas as variáveis que ocorrem em $(A \vee \neg B_1 \vee \ldots \vee \neg B_m)$, então podemos colocá-la na forma de cláusula de Horn, que será:

$$A \leftarrow B_1, \ldots, B_m$$

A subfórmula A é dita a cabeça da cláusula, enquanto a seqüência $B_1, \ldots, B_m$ é dita corpo da cláusula. Se m for igual a 0, a fórma da cláusula de Horn será:

$$A \leftarrow$$

Estes tipos de cláusulas de Horn são denominados de cláusulas (de Horn) definidas.

Já, uma vez fornecida uma sentença da forma

$$\forall x_1, \ldots, \forall x_n (\neg B_1 \vee \ldots \vee \neg B_m)$$

onde de B_1 até B_m são fórmulas atômicas e x_1 até x_n são todas as variáveis que ocorrem em $(\neg B_1 \vee \ldots \vee \neg B_m)$, então podemos colocá-la na fórma de cláusula de Horn, que será:

$$B_1, \ldots, B_m$$

Este tipo de cláusula de Horn é denominado de cláusula (de Horn) objetivo.

Também definimos a cláusula vazia, que, para qualquer função de avaliação, é falsa, e é representada por: $\square$.

A.4 Unificação.

Uma substituição θ é um conjunto de elementos da forma $\langle$variável $\Leftarrow$ termo$\rangle$, tal que: (1) não ocorrem dois elementos com a mesma variável à esquerda do símbolo $\Leftarrow$ e (2) dada uma fórmula α, podemos gerar a fórmula $\alpha\theta$ substituindo em α, simultaneamente, para cada termo $\langle x_i \Leftarrow t_i \rangle$ de θ, todas as ocorrências de x_i por t_i.

Por exemplo:

$$\theta = \{\langle x \Leftarrow y \rangle, \langle y \Leftarrow f(z) \rangle\}$$
$$\alpha = p(x) \supset q(y)$$
$$\alpha\theta = p(y) \supset q(f(z))$$

A unificação é a aplicação de uma substituição em um conjunto de fórmulas, de forma que as fórmulas resultantes sejam todas iguais. Ou seja, dado o conjunto de fórmulas $S_1, \ldots, S_n$, θ será um unificador destas fórmulas se e somente se as fórmulas $S_1\theta, \ldots, S_n\theta$ forem todas iguais.

Normalmente a unificação é considerada como um processo envolvendo fórmulas atômicas, entretanto, para nossos propósitos, precisamos considerar a unificação como uma operação mais geral, capaz de tratar qualquer tipo de fórmula.

Por exemplo:

$$S_1 = p(x_1) \supset q(g(y_1))$$
$$S_2 = p(f(x_2)) \supset q(x_2)$$
$$\theta = \{\langle x_1 \Leftarrow f(g(y_1)) \rangle, \langle x_2 \Leftarrow g(y_1) \rangle\}$$
$$S_1\theta = p(f(g(y_1))) \supset q(g(y_1))$$
$$S_2\theta = p(f(g(y_1))) \supset q(g(y_1))$$

Note que deverá sempre ser feito o uso de subscrição ou outra forma de renomeação das variáveis, de forma a não nos preocuparmos com problemas causados pela ocorrência de variáveis com o mesmo nome em fórmulas diferentes.

Chamamos θ de unificador mais geral de um dado conjunto de fórmulas, se e somente se θ for um dos mais simples unificadores possíveis para estas fórmulas.

Ou seja, θ será um unificador mais geral para as fórmulas $S_1, \ldots, S_n$ se e somente se para todo unificador θ_1 de $S_1, \ldots, S_n$, sempre existir um unificador θ_2 tal que θ_1 possa ser expresso por uma composição de θ e θ_2, isto é: $(S_1\theta)\theta_2 = S_1\theta_1 = \ldots = (S_n\theta)\theta_2 = S_n\theta_1$.

A.5 Resolução-LSD.

A resolução-LSD [Kowalski 71] é um método para avaliar se uma fórmula é verdadeira, baseada na hipótese de um conjunto de fórmulas (tomadas como premissas), ser formada por fórmulas verdadeiras. A fórmula a ser provada deve estar na forma de cláusula de Horn objetivo e as fórmulas tomadas como premissas devem estar na forma de cláusulas de Horn definidas.

Dada uma cláusula definida C_j da forma $B_0 \leftarrow B_1, \ldots, B_n$ e uma cláusula objetivo O da forma $A_1, A_2, \ldots, A_m$. Se θ é um unificador mais geral entre B_0 e A_1 então a extensão de O por C_j é a cláusula objetivo O' da forma $(B_1, \ldots, B_n, A_2, \ldots, A_m)\theta$, que também chamamos de resolvente.

Uma prova pelo método resolução-LSD é a seqüência $D_1, \ldots, D_n$, tal que D_1 seja a cláusula objetivo a ser provada, e para cada i, $1 < i \leq n$, D_i é uma extensão de D_{i-1} por qualquer uma das das cláusulas definidas tomadas como premissa, e D_n é a cláusula vazia.

Anexo B

Implementação da Abordagem 1.

Listamos aqui os arquivos relativos à implementação da abordagem 1 de Martín Abadi e Zohar Manna.

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Programa Principal                                      *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/abadi.pro        Data de Criacao : 04.04.91  *
 *      Versao : 1.00                        Ultima Alteracao: 15.04.91  *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      Carga doa outros modulos                                         *
 *************************************************************************/

:- [ -lex, -aux, -reg1, -reg2, -reg3 ].


/*************************************************************************
 *      Predicado Principal                                              *
 *************************************************************************/

roda :-
        write( 'Entre com a formula: ' ),
        read_string( 100, STR ),
        list_text( LISTA, STR ),
        lex_converte( LISTA, LFORM ),
        ana_formula( LFORM, FORMULA ),
        prova( FORMULA, 0 ).


/*************************************************************************
 *      Aplica o Metodo                                                  *
 *************************************************************************/

prova( false, N ) :-
        !,
        write( 'FIM. false: sentenca provada.' ),
        nl.
```

```
prova( FORM, 30 ) :-
        !,
        write( 'Chega ...' ),
        nl.

prova( FORMULA, 0 ) :-
        !,
        write( '0. ' ),
        imp_formula( FORMULA ),
        nl,
        simplifica( FORMULA, F1 ),
        extrai( F1, F2 ),
        elimina( F2, F3 ),
        prova( F3, 1 ).

prova( FORMULA, N ) :-
        write( N ),
        write( '. ' ),
        imp_formula( FORMULA ),
        nl,
        N1 is N + 1,
        regra( FORMULA, F1 ),
        extrai( F1, F2 ),
        elimina( F2, F3 ),
        prova( F3, N1 ).
```

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Predicados Auxiliares                                   *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/aux.pro          Data de Criacao : 01.04.91  *
 *      Versao : 1.00                        Ultima Alteracao: 09.09.91  *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      Obtem uma conjuncao de uma sentenca                              *
 *************************************************************************/

obtem_conj( t( v, A, B ), CONJ, t( v, NA, B ), X ) :-
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( v, A, B ), CONJ, t( v, A, NB ), X ) :-
        !,
        obtem_conj( B, CONJ, NB, X ).

obtem_conj( t( =>, A, B ), CONJ, t( =>, NA, B ), X ) :-
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( =>, A, B ), CONJ, t( =>, A, NB ), X ) :-
        !,
        obtem_conj( B, CONJ, NB, X ).

obtem_conj( t( 'EE', VAR, A ), CONJ, t( 'EE', VAR, NA ), X ) :-
        !,
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( 'VV', VAR, A ), CONJ, t( 'VV', VAR, NA ), X ) :-
        !,
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( #, A, [] ), CONJ, t( #, NA, [] ), X ) :-
        !,
        obtem_conj( A, CONJ, NA, X ).
```

```
obtem_conj( t( <>, A, [] ), CONJ, t( <>, NA, [] ), X ) :-
        !,
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( ~, A, [] ), CONJ, t( ~, NA, [] ), X ) :-
        !,
        obtem_conj( A, CONJ, NA, X ).

obtem_conj( t( ^, A, B ), CONJ, t( ^, NA, B ), X ) :-
        sub_conj( A , CONJ, NA, X ).

obtem_conj( t( ^, A, B ), CONJ, t( ^, A, NB ), X ) :-
        sub_conj( B , CONJ, NB, X ).

obtem_conj( t( ^, A, B ), CONJ, X, X ) :-
        list_conj( B, [], AUX ),
        list_conj( A, AUX, CONJ ).


/*************************************************************************
 *      Procura Conjuncoes dentro de Conjuncoes                          *
 *************************************************************************/

sub_conj( t( ^, A, B ), CONJ, t( ^, NA, B ), X ) :-
        sub_conj( A, CONJ, NA, X ).

sub_conj( t( ^, A, B ), CONJ, t( ^, A, NB ), X ) :-
        !,
        sub_conj( B, CONJ, NB, X ).

sub_conj( TERMO, CONJ, NTERMO, X ) :-
        obtem_conj( TERMO, CONJ, NTERMO, X ).


/*************************************************************************
 *      Coloca os Termos da Conjuncao em uma Lista                       *
 *************************************************************************/

list_conj( t( ^, A, B ), INI, FIM ) :-
        !,
        list_conj( B, INI, AUX ),
        list_conj( A, AUX, FIM ).

list_conj( TERMO, INI, [ TERMO | INI ] ).
```

```
/*************************************************************************
 *      Concatenacao de lista                                            *
 *************************************************************************/

junta( [], L, L ) :-
        !.

junta( [ C | R1 ], L, [ C | R2 ] ) :-
        junta( R1, L, R2 ).


/*************************************************************************
 *      Retira um Elemento Existente de uma Lista                        *
 *************************************************************************/

retira( X, [ X | R ], R ).

retira( X, [ C | R ], [ C | NR ] ) :-
        retira( X, R, NR ).


/*************************************************************************
 *      Une duas Listas sem Repeticao                                    *
 *************************************************************************/

uniao( [], L, L ) :-
        !.

uniao( [ C | R ] , L, [ C | NR ] ) :-
        retira( C, L, NL ),
        !,
        uniao( R, NL, NR ).

uniao( [ C | R ], L, [ C | NR ] ) :-
        uniao( R, L, NR ).


/*************************************************************************
 *      Pertinencia a uma Lista                                          *
 *************************************************************************/

pertence( X, [ X | _ ] ).
```

```
pertence( X, [ _ | R ] ) :-
        pertence( X, R ).


/*************************************************************************
 *      Nao Pertinencia a uma Lista                                      *
 *************************************************************************/

nao_pertence( X, L ) :-
        pertence( X, L ),
        !,
        fail.

nao_pertence( _, _ ).


/*************************************************************************
 *      Obtem um Termo Qualquer                                          *
 *************************************************************************/

obtem_termo( FORM, FORM, X, X ).

obtem_termo( t( OP_UNA, FORM, [] ), TERMO, t( OP_UNA, NFORM, [] ), X ) :-
        !,
        obtem_termo( FORM, TERMO, NFORM, X ).

obtem_termo( t( OP_BIN, A, B ), TERMO, t( OP_BIN, NA, B ), X ) :-
        obtem_termo( A, TERMO, NA, X ).

obtem_termo( t( OP_BIN, A, B ), TERMO, t( OP_BIN, A, NB ), X ) :-
        obtem_termo( B, TERMO, NB, X ).


/*************************************************************************
 *      Separa os Quantificadores de uma Formula                         *
 *************************************************************************/

separa_quant( t( 'VV', X, FORM ), [ X | RABO ], V_ELO, ELIST, E_ELO ) :-
        !,
        separa_quant( FORM, RABO, V_ELO, ELIST, E_ELO ).

separa_quant( t( 'EE', X, FORM ), VLIST, V_ELO, [ X | RABO ], E_ELO ) :-
        !,
        separa_quant( FORM, VLIST, V_ELO, RABO, E_ELO ).
```

```
separa_quant( t( OP_UNA, FORM, [] ), VLIST, V_ELO, ELIST, E_ELO ) :-
        !,
        separa_quant( FORM, VLIST, V_ELO, ELIST, E_ELO ).

separa_quant( t( OP_BIN, AFORM, BFORM ), VLIST, V_ELO, ELIST, E_ELO ) :-
        !,
        separa_quant( AFORM, VLIST, VAUX, ELIST, EAUX ),
        separa_quant( BFORM, VAUX, V_ELO, EAUX, E_ELO ).

separa_quant( TERMO, VLIST, VLIST, ELIST, ELIST ).


/*************************************************************************
 *      Monta uma Conjuncao                                              *
 *************************************************************************/

monta_conj( [ A, B ], t( ^, A, B ) ) :-
        !.

monta_conj( [ A | R ], t( ^, A, CONJ ) ) :-
        monta_conj( R, CONJ ).
```

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Analisador Lexico e Sintatico                           *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/lex.pro          Data de Criacao : 30.01.91  *
 *      Versao : 1.00                        Ultima Alteracao: 10.04.91  *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      Analise de formulas                                              *
 *************************************************************************/

ana_formula( L, FORM ) :-
        ana_sentenca( L, [], FORM ),
        !.
ana_formula( _, _ ) :-
        write( 'Erro na analise da formula' ),
        nl,
        fail.


/*************************************************************************
 *      SENTENCA -> CONJUNCAO                                            *
 *                | CONJUNCAO v SENTENCA                                 *
 *************************************************************************/

ana_sentenca( L , L3, SENT ) :-
        ana_conjuncao( L, L2, CONJ ),
        ana_sentenca2( L2, L3, CONJ, SENT ).
ana_sentenca2( [ v | L ], L2, CONJ, t( v, CONJ, SENT ) ) :-
        !,
        ana_sentenca( L, L2, SENT ).
ana_sentenca2( L, L, CONJ, CONJ ).
```

```
/*************************************************************************
 *      CONJUNCAO -> IMPLICACAO                                          *
 *                 | IMPLICACAO ^ CONJUNCAO                             *
 *************************************************************************/

ana_conjuncao( L, L3, CONJ ) :-
        ana_implicacao( L, L2, IMP ),
        ana_conjuncao2( L2, L3, IMP, CONJ ).
ana_conjuncao2( [ ^ | L ], L2 , IMP, t( ^, IMP, CONJ ) ) :-
        !,
        ana_conjuncao( L, L2, CONJ ).
ana_conjuncao2( L, L, IMP, IMP ).


/*************************************************************************
 *      IMPLICACAO -> TERMO                                              *
 *                  | TERMO => IMPLICACAO                                *
 *************************************************************************/

ana_implicacao( L, L3, IMP ) :-
        ana_termo( L, L2, TERMO ),
        ana_implicacao2( L2, L3, TERMO, IMP ).
ana_implicacao2( [ => | L ], L2, TERMO, t( =>, TERMO, IMP ) ) :-
        !,
        ana_implicacao( L, L2, IMP ).
ana_implicacao2( L, L, TERMO, TERMO ).


/*************************************************************************
 *      TERMO -> ( SENT )                                                *
 *             | EE x TERMO                                              *
 *             | VV x TERMO                                              *
 *             | # TERMO                                                 *
 *             | <> TERMO                                                *
 *             | ~ TERMO                                                 *
 *             | PRED( ARG, ..., ARG)                                    *
 *************************************************************************/

ana_termo( [ '(' | L ], L2, SENT ) :-
        !,
        ana_sentenca( L, [ ')' | L2 ], SENT ).
ana_termo( [ 'EE', VAR | L ], L2, t( 'EE', VAR, TERMO ) ) :-
        !,
        ana_var( VAR ),
        ana_termo( L, L2, TERMO ).
```

```
ana_termo( [ 'VV', VAR | L ], L2, t('VV', VAR, TERMO ) ) :-
        !,
        ana_var( VAR ),
        ana_termo( L, L2, TERMO ).
ana_termo( [ # | L ], L2, t( #, TERMO, [] ) ) :-
        !,
        ana_termo( L, L2, TERMO ).
ana_termo( [ <> | L ], L2, t( <>, TERMO, [] ) ) :-
        !,
        ana_termo( L, L2, TERMO ).
ana_termo( [ ~ | L ], L2, t( ~, TERMO, [] ) ) :-
        !,
        ana_termo( L, L2, TERMO ).
ana_termo( [ PRED | L ], L2, ESTR ) :-
        ana_pred( PRED ),
        ana_pred2( L, L2, PRED, ESTR ).

ana_var( VAR ) :-
        list_text( L, VAR ),
        L = [ 'x | _ ].

ana_pred( PRED ) :-
        atom( PRED ).

ana_pred2( [ '(' , ARG | L ], L2, PRED, ESTR ) :-
        !,
        ana_pred3( L, L2, PREDS ),
        ESTR =.. [ PRED, ARG | PREDS ].
ana_pred2( L, L, PRED, PRED ).

ana_pred3( [ ',', ARG | L ], L2, [ ARG | ARGS ] ) :-
        !,
        ana_pred3( L, L2, ARGS ).
ana_pred3( [ ')' | L ], L, [] ).


/*************************************************************************
 *      Conversao lexica de caracteres para simbolos                     *
 *************************************************************************/

lex_converte( CHAR, SIMB ) :-
        lex_percorre( CHAR , SIMB),
        !.
```

```
lex_converte( _, _ ) :-
        write( 'Erro lexico na formula.' ),
        nl,
        fail.

lex_percorre( [], [] ) :-
        !.
lex_percorre( [ ‘  | L ], SIMB ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘v | L ], [ v | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘^ | L ], [ ^ | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘( | L ], [ '(' | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘) | L ], [ ')' | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘# | L ], [ # | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘~ | L ], [ ~ | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘, | L ], [ ',' | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘=, ‘> | L ], [ => | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘E, ‘E | L ], [ 'EE' | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘V, ‘V | L ], [ 'VV' | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
lex_percorre( [ ‘<, ‘> | L ], [ <> | SIMB ] ) :-
        !,
        lex_percorre( L, SIMB ).
```

```
lex_percorre( [ C | L ], [ IDENT | SIMB ] ) :-
        C >= 'a,
        C =< 'z,
        lex_ident( L, L2, RESTO ),
        list_text( [ C | RESTO ], STR ),
        atom_string( IDENT, STR ),
        lex_percorre( L2, SIMB ).


lex_ident( [ C | L ], L2, [ C | RESTO ] ) :-
        C >= 'a,
        C =< 'z,
        !,
        lex_ident( L, L2, RESTO ).
lex_ident( [ C | L ], L2, [ C | RESTO ] ) :-
        C >= '0,
        C =< '9,
        !,
        lex_ident( L, L2, RESTO ).
lex_ident( L, L, [] ).


/*************************************************************************
 *      Impressao de formulas                                            *
 *************************************************************************/

imp_formula( t( v, CONJ, SENT ) ):-
        !,
        put( '( ),
        imp_formula( CONJ ),
        write( ' v ' ),
        imp_formula( SENT ),
        put( ') ).

imp_formula( t( ^, IMP, CONJ ) ):-
        !,
        put( '( ),
        imp_formula( IMP ),
        write( ' ^ ' ),
        imp_formula( CONJ ),
        put( ') ).
```

```
imp_formula( t( =>, TERMO, IMP ) ):-
        !,
        put( '( ),
        imp_formula( TERMO ),
        write( ' => ' ),
        imp_formula( IMP ),
        put( ') ).

imp_formula( t( 'EE', VAR, TERMO ) ):-
        !,
        write( '(EE ' ),
        write( VAR ),
        put( ' ' ),
        imp_formula( TERMO ),
        put( ') ).

imp_formula( t('VV', VAR, TERMO ) ):-
        !,
        write( '(VV ' ),
        write( VAR ),
        put( ' ' ),
        imp_formula( TERMO ),
        put( ') ).

imp_formula( t( #, TERMO, [] ) ):-
        !,
        write( '#' ),
        imp_formula( TERMO ).

imp_formula( t( <>, TERMO, [] ) ):-
        !,
        write( '<>' ),
        imp_formula( TERMO ).

imp_formula( t( ~, TERMO, [] ) ):-
        !,
        put( '~ ),
        imp_formula( TERMO ).

imp_formula( TERMO ) :-
        write( TERMO ).
```

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Regras Simples                                          *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/reg1.pro         Data de Criacao : 01.04.91 *
 *      Versao : 1.00                        Ultima Alteracao: 12.04.91 *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      Simplificacoes                                                   *
 *************************************************************************/


/*************************************************************************
 *      A => B  : ~A v B                                                 *
 *************************************************************************/

simp( t( =>, A, B ), NAVB ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( v, NA, B ), NAVB ).


/*************************************************************************
 *      ~~A : A                                                          *
 *************************************************************************/

simp( t( ~, t( ~, A, [] ), [] ), A ) :-
        !.


/*************************************************************************
 *      ~( A v B ) : ~A ^ ~B                                             *
 *************************************************************************/

simp( t( ~, t( v, A, B ), [] ), NAENB ) :-
        !,
        simp( t( ~, A, [] ), NA ),
```

```
        simp( t( ~, B, [] ), NB ),
        simp( t( ^, NA, NB ), NAENB ).


/*************************************************************************
 *      ~( A ^ B ) : ~A v ~B                                             *
 *************************************************************************/

simp( t( ~, t( ^, A, B ), [] ), NAVNB ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( ~, B, [] ), NB ),
        simp( t( v, NA , NB ), NAVNB ).


/*************************************************************************
 *      A ^ ( B v C ) : A ^ B v A ^ C                                    *
 *************************************************************************/

simp( t( ^, A, t( v, B, C ) ), AEBVAEC ) :-
        !,
        simp( t( ^, A, B ), AEB ),
        simp( t( ^, A, C ), AEC ),
        simp( t( v, AEB, AEC ), AEBVAEC ).

simp( t( ^, t( v, B, C ), A ), BEAVCEA ) :-
        !,
        simp( t( ^, B, A ), BEA ),
        simp( t( ^, C, A ), CEA ),
        simp( t( v, BEA, CEA ), BEAVCEA ).


/*************************************************************************
 *      false ^ A : false                                                *
 *************************************************************************/

simp( t( ^, false, A ), false ) :-
        !.

simp( t( ^, A, false), false ) :-
        !.
```

```
/*************************************************************************
 *      false v A : A                                                    *
 *************************************************************************/

simp( t( v, false, A ), A ) :-
        !.

simp( t( v, A, false ), A ) :-
        !.


/*************************************************************************
 *      ~ true : false                                                   *
 *************************************************************************/

simp( t( ~, true, [] ), false ) :-
        !.


/*************************************************************************
 *      ~ false : true                                                   *
 *************************************************************************/

simp( t( ~, false, [] ), true ) :-
        !.


/*************************************************************************
 *      true ^ A : A                                                     *
 *************************************************************************/

simp( t( ^, true, A ), A ) :-
        !.

simp( t( ^, A, true ), A ) :-
        !.


/*************************************************************************
 *      true v A : true                                                  *
 *************************************************************************/

simp( t( v, true, A ), true ) :-
        !.
```

```
simp( t( v, A, true ), true ) :-
        !.


/*************************************************************************
 *      <>false : false                                                  *
 *************************************************************************/

simp( t( <>, false, [] ), false ):-
        !.


/*************************************************************************
 *      [] false : false                                                 *
 *************************************************************************/

simp( t( #, false, [] ), false ) :-
        !.


/*************************************************************************
 *      <>true : true                                                    *
 *************************************************************************/

simp( t( <>, true, [] ), true ):-
        !.


/*************************************************************************
 *      [] true : true                                                   *
 *************************************************************************/

simp( t( #, true, [] ), true ) :-
        !.


/*************************************************************************
 *      ~[]A : <>~A                                                      *
 *************************************************************************/

simp( t( ~, t( #, A, [] ), [] ), PNA ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( <>, NA , [] ), PNA ).
```

```
/*************************************************************************
 *      ~<>A : []~A                                                      *
 *************************************************************************/

simp( t( ~, t( <>, A, [] ), [] ), NNA ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( #, NA , [] ), NNA ).


/*************************************************************************
 *      ~VVx A : EEx ~A                                                  *
 *************************************************************************/

simp( t( ~, t( 'VV', X, A ), [] ), EXNA ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( 'EE', X , NA ), EXNA ).


/*************************************************************************
 *      ~EEx A : VVx ~A                                                  *
 *************************************************************************/

simp( t( ~, t( 'EE', X, A ), [] ), VXNA ) :-
        !,
        simp( t( ~, A, [] ), NA ),
        simp( t( 'VV', X , NA ), VXNA ).


/*************************************************************************
 *      O resto                                                          *
 *************************************************************************/

simp( X, X ).


/*************************************************************************
 *      Simplificacao                                                    *
 *************************************************************************/

simplifica( t( O, A, B ), ST ) :-
        !,
        simplifica( A, SA ),
```

```
        simplifica( B, SB ),
        simp( t( O, SA, SB ), ST ).

simplifica( TERMO, TERMO ).


/*************************************************************************
 *      Extracao de Quantificadores                                      *
 *************************************************************************/

extrai( t( 'VV', X, FORM ), t( 'VV', X, NFORM ) ) :-
        !,
        extrai( FORM, NFORM ).

extrai( t( 'EE', X, FORM ), t( 'EE', X, NFORM ) ) :-
        !,
        extrai( FORM, NFORM ).

extrai( FORM, NFORM ) :-
        normaliza( FORM, NFORM, _, SUBFORM, _, SUBFORM, normal ).


/*************************************************************************
 *      Separa os Quantificadores                                        *
 *************************************************************************/

normaliza( t( 'VV', X, F ), t( 'VV', X, Q ), E, TERMO, ET, NFORM, MODO ) :-
        !,
        normaliza( F, Q, E, TERMO, ET, NFORM, MODO ).

normaliza( t( 'EE', X, F ), t( 'EE', X, Q ), _, TERMO, _, NFORM, normal ) :-
        !,
        normaliza( F, Q, _, TERMO, _, NFORM, normal ).

normaliza( t( 'EE', X, F ), Q, t( 'EE', X, E ), QT, ET, NF, restrito ) :-
        !,
        normaliza( F, Q, E, QT, ET, NF, restrito ).

normaliza( t( ~, F, [] ), QT, ET, QT, ET, t( ~, F, [] ), MODO ) :-
        !.

normaliza( t( <>, F, [] ), Q, E, QTERMO, ETERMO, t( <>, NFORM, [] ), MODO ) :-
        !,
        normaliza( F, Q, E, QTERMO, ETERMO, NFORM, MODO ).
```

```
normaliza( t( #, F, [] ), Q, ET, QT, ET, t( #, E, [] ), MODO ) :-
        !,
        normaliza( F, Q, E, QT, NFORM, NFORM, restrito ).

normaliza( t( OPER, A, B ), Q, EE, QT, ET, t( OPER, NA, NB ), MODO ) :-
        !,
        normaliza( A, AQ, EE, [], XE, NA, MODO ),
        normaliza( B, BQ, XE, [], ET, NB, MODO ),
        junta_quant( AQ, BQ, Q, QT ).

normaliza( FORM , QTERMO, ETERMO, QTERMO, ETERMO, FORM, MODO ).


/*************************************************************************
 *      Junta duas Series de Quantificadores Independentes               *
 *************************************************************************/


junta_quant( [], [], TERMO, TERMO ) :-
        !.

junta_quant( [], t( QQ, X, RESTO ), t( QQ, X, NRESTO ), TERMO ) :-
        !,
        junta_quant( [], RESTO, NRESTO, TERMO ).

junta_quant( t( QQ, X, RESTO ), [], t( QQ, X, NRESTO ), TERMO ) :-
        !,
        junta_quant( [], RESTO, NRESTO, TERMO ).

junta_quant( t( 'EE', X, RESTO ), QUANT, t( 'EE', X, NRESTO ), TERMO ) :-
        !,
        junta_quant( RESTO, QUANT, NRESTO, TERMO ).

junta_quant( QUANT, t( 'EE', X, RESTO ), t( 'EE', X, NRESTO ), TERMO ) :-
        !,
        junta_quant( RESTO, QUANT, NRESTO, TERMO ).

junta_quant( t( 'VV', X, RESTO ), QUANT, t( 'VV', X, NRESTO ), TERMO ) :-
        junta_quant( RESTO, QUANT, NRESTO, TERMO ).
```

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Regras nao tao Simples                                  *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/reg2.pro        Data de Criacao : 11.04.91 *
 *      Versao : 1.00                       Ultima Alteracao: 24.04.91 *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      Eliminacao de quantificadores Desnecessarios                     *
 *************************************************************************/

elimina( FORMULA, NFORMULA ) :-
        elim( FORMULA, NFORMULA, LISTA ).


/*************************************************************************
 *      Joga Fora os Quantificadores Desnecessarios                      *
 *************************************************************************/

elim( t( 'VV', X, FORM ), FORM_FINAL, LISTA_FINAL ) :-
        !,
        elim( FORM, NFORM, LISTA ),
        jogafora( 'VV', X, LISTA, NFORM, FORM_FINAL, LISTA_FINAL ).

elim( t( 'EE', X, FORM ), FORM_FINAL, LISTA_FINAL ) :-
        !,
        elim( FORM, NFORM, LISTA ),
        jogafora( 'EE', X, LISTA, NFORM, FORM_FINAL, LISTA_FINAL ).

elim( t( OP_UNA, FORM, [] ), t( OP_UNA, NFORM, [] ), LISTA ) :-
        !,
        elim( FORM, NFORM, LISTA ).

elim( t( OP_BIN, A, B ), t( OP_BIN, NA, NB ), LISTA ) :-
        !,
        elim( A, NA, LA ),
        elim( B, NB, LB ),
```

```
        uniao( LA, LB, LISTA ).

elim( TERMO, TERMO, LISTA ) :-
        TERMO =.. [ PRED | ARGS ],
        separa_vars( ARGS, LISTA ).


/*************************************************************************
 *      Separa as Variaveis dentre a Lista de Argumentos                 *
 *************************************************************************/

separa_vars( [], [] ) :-
        !.

separa_vars( [ C | R ], [ C | L ] ) :-
        ana_var( C ),
        !,
        separa_vars( R, L ).

separa_vars( [ C | R ], L ) :-
        separa_vars( R, L ).

/*************************************************************************
 *      Gera a Nova Formula com ou sem o Quantificador                   *
 *************************************************************************/

jogafora( QUANT, X, LISTA, FORM, t( QUANT, X, FORM ), NLISTA ) :-
        retira( X, LISTA, NLISTA ),
        !.

jogafora( QUANT, X, LISTA, FORM, FORM, LISTA ).


/*************************************************************************
 *      Unificacao de duas Formulas                                      *
 *************************************************************************/

unifica( FORM, A, B, UNIFICADOR, NOVOUNIFICADOR ) :-
        separa_quant( FORM, VLIST, [], ELIST, [] ),
        unif( A, B, VLIST, ELIST, UNIFICADOR, NOVOUNIFICADOR ).
```

```
/*************************************************************************
 *      Unificacao de duas Formulas (continuacao)                        *
 *************************************************************************/

unif( t( 'EE', X, A ), B, VLIST, ELIST, UNIF, NUNIF ) :-
        !,
        unif( A, B, VLIST, ELIST, UNIF, NUNIF ).

unif( A, t( 'EE', X, B ), VLIST, ELIST, UNIF, NUNIF ) :-
        !,
        unif( A, B, VLIST, ELIST, UNIF, NUNIF ).

unif( t( OP_UNA, A, [] ), PAR2, VLIST, ELIST, UNIF, NUNIF ) :-
        !,
        PAR2 = t( OP_UNA, B, [] ),
        unif( A, B, VLIST, ELIST, UNIF, NUNIF ).

unif( t( OP_BIN, A1, A2 ), PAR2, VLIST, ELIST, UNIF, NUNIF ) :-
        !,
        PAR2 = t( OP_BIN, B1, B2 ),
        unif( A1, B1, VLIST, ELIST, UNIF, AUX ),
        unif( A2, B2, VLIST, ELIST, AUX, NUNIF ).

unif( TERMOA, TERMOB, VLIST, ELIST, UNIF, NUNIF ) :-
        TERMOA =.. [ PRED | ARGA ],
        TERMOB =.. [ PRED | ARGB ],
        unif_args( ARGA, ARGB, VLIST, ELIST, UNIF, NUNIF ).


/*************************************************************************
 *      Unifica Argumentos                                               *
 *************************************************************************/

unif_args( [], [], VLIST, ELIST, UNIF, UNIF ) :-
        !.

unif_args( [ X | RA ], [ X | RB ], VLIST, ELIST, UNIF, NUNIF ) :-
        !,
        unif_args( RA, RB, VLIST, ELIST, UNIF, NUNIF ).

unif_args( [ XA | RA ], [ XB | RB ], VL, EL, UNIF, [ a( XA, XB ) | NUNIF ] ) :-
        pertence( XA, VL ),
        !,
        unif_args( RA, RB, VL, EL, UNIF, NUNIF ).
```

```
unif_args( [ XA | RA ], [ XB | RB ], VL, EL, UNIF, [ a( XB, XA ) | NUNIF ] ) :-
        pertence( XB, VL ),
        !,
        unif_args( RA, RB, VL, EL, UNIF, NUNIF ).

unif_args( [ XA | RA ], [ XB | RB ], VL, EL, UNIF, [ a( XA, XB ) | NUNIF ] ) :-
        pertence( XA, EL ),
        !,
        unif_args( RA, RB, VL, EL, UNIF, NUNIF ).

unif_args( [ XA | RA ], [ XB | RB ], VL, EL, UNIF, [ a( XB, XA ) | NUNIF ] ) :-
        pertence( XB, EL ),
        unif_args( RA, RB, VL, EL, UNIF, NUNIF ).
```

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna                                          *
 *                                                                       *
 *      Modulo : Regras nao Deterministicas                              *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/abadi/reg3.pro         Data de Criacao : 11.04.91  *
 *      Versao : 1.00                        Ultima Alteracao: 09.09.91  *
 *                                                                       *
 *************************************************************************/


/*************************************************************************
 *      [] u, <>v --> <>(u ^ v)                                          *
 *************************************************************************/

/* Redundante em S4 */

/*************************************************************************
 *      []u --> u                                                        *
 *************************************************************************/

regra( FORM, NFORM ) :-
        obtem_conj( FORM, CONJ, NFORM, ELO ),
        pertence( t( #, U, [] ), CONJ ),
        nao_pertence( U, CONJ ),
        monta_conj( [ U | CONJ ], ELO ).


/*************************************************************************
 *      Qa A<u,...,u>,Qb B<u,...,u> --> Qab(A<true>,B<false>)            *
 *************************************************************************/

regra( FORM, SFORM ) :-
        obtem_conj( FORM, CONJ, NFORM, ELO ),
        retira( A, CONJ, CONJ2 ),
        pertence( B, CONJ2 ),
        obtem_termo( A, UA, _, _ ),
        obtem_termo( B, UB, _, _ ),
        unifica( FORM, UA, UB, [], UNIFICADOR ),
        aplica_unif( UNIFICADOR, UA, U ),
        aplica_unif( UNIFICADOR, A, UNI_A ),
```

```
        aplica_unif( UNIFICADOR, B, UNI_B ),
        troca_todos( U, UNI_A, true, NA),
        troca_todos( U, UNI_B, false, NB ),
        simplifica( t( v, NA, NB ), TERMO ),
        TERMO \= true,
        monta_conj( [ TERMO | CONJ ], ELO ),
        aplica_unif( UNIFICADOR, NFORM, UNI_FORM ),
        simplifica( UNI_FORM, SFORM ).


aplica_unif( UNIF, [ C | R ], [ NC, NR ] ) :-
        !,
        aplica_unif( UNIF, C, NC ),
        aplica_unif( UNIF, R, NR ).

aplica_unif( UNIF, t( OP_UNA, FORM, [] ), t( OP_UNA, NFORM, [] ) ) :-
        !,
        aplica_unif( UNIF, FORM, NFORM ).

aplica_unif( UNIF, t( OP_BIN, A, B ), t( OP_BIN, NA, NB ) ) :-
        !,
        aplica_unif( UNIF, A, NA ),
        aplica_unif( UNIF, B, NB ).

aplica_unif( UNIF, TERMO, NTERMO ) :-
        TERMO =.. [ PRED | ARGS ],
        aplica_unif_args( UNIF, ARGS, NARGS ),
        NTERMO =.. [ PRED | NARGS ].

aplica_unif_args( UNIF, [], [] ) :-
        !.

aplica_unif_args( UNIF, [ X | R ], [ NX | NR ] ) :-
        pertence( a( X, NX ), UNIF ),
        !,
        aplica_unif_args( UNIF, R, NR ).

aplica_unif_args( UNIF, [ X | R ], [ X | NR ] ) :-
        aplica_unif_args( UNIF, R, NR ).


troca_todos( U, U, NU, NU ) :-
        !.
```

```
troca_todos( U, t( OP_UNA, A, [] ), NU, t( OP_UNA, NA, [] ) ) :-
        !,
        troca_todos( U, A, NU, NA ).

troca_todos( U, t( OP_BIN, A, B ), NU, t( OP_BIN, NA, NB ) ) :-
        !,
        troca_todos( U, A, NU, NA ),
        troca_todos( U, B, NU, NB).

troca_todos( _, FORM, _, FORM ).


/*************************************************************************
 *      []u,<>v --> <>([]u ^ v)                                          *
 *************************************************************************/

regra( FORM, NFORM ) :-
        obtem_conj( FORM, CONJ, NFORM, ELO ),
        pertence( t( #, U, [] ), CONJ ),
        pertence( t( <>, V, [] ), CONJ ),
        nao_repete( U, V, CONJ ),
        monta_conj( [ t( <>, t( ^, t( #, U, [] ), V ), [] ) | CONJ ], ELO ).


nao_repete( U, V, CONJ ) :-
        pertence( t( <>, t( ^, t( #, U, [] ), V ), [] ), CONJ ),
        !,
        fail.

nao_repete( U, V, CONJ ) :-
        e_pertence( t( #, U, [] ), V ),
        !,
        fail.

nao_repete( _, _, _ ).


e_pertence( T, T ) :-
        !.

e_pertence( T, t( ^, T, _ ) ) :-
        !.

e_pertence( T, t( ^, _, X ) ) :-
        e_pertence( T, X ).
```

```
/*************************************************************************
 *      <>u,<>v --> <>(<>u ^ v)                                          *
 *************************************************************************/

/* S5 */

/*************************************************************************
 *      u --> <>u                                                        *
 *************************************************************************/

/* S5 */
```

Anexo C

Implementação da Abordagem 2 - Versão 1.

Listamos aqui o arquivo relativo à primeira versão da implementação da abordagem 2 de Martín Abadi e Zohar Manna para lógica temporal.

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna para Logica Temporal                     *
 *                                                                       *
 *      Modulo : Programa Principal                                      *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/tempo/tempo0.pro       Data de Criacao : 16.09.91  *
 *      Versao : 1.00                        Ultima Alteracao: 30.09.91  *
 *                                                                       *
 *************************************************************************/

/*************************************************************************
 *      Define os operadores                                             *
 *************************************************************************/

        op( 1100, xfx, <-- ),
        op( 1100, xfx, <== ),
        op(  500,  fy, o   ),
        op(  500,  fx, #   ).

/*************************************************************************
 *      Define predefinidos para todos os momentos                       *
 *************************************************************************/

true( _ ).
znl( _ ) :- nl.
zwrite( A, _ ) :- write( A ).
is( A, B, _ ) :- is( A, B ).

/*************************************************************************
 *      Entrada de clausulas                                             *
 *************************************************************************/

entra(( # CABECA <-- CORPO )) :-
        !,
        CABECA =.. [ PREDICADO | ARGUMENTOS ],
        ctr_inc( n, N ),
        entra(( CABECA <== zr( N, ARGUMENTOS ))),
        entra(( zr( N, ARGUMENTOS ) <-- CORPO )).

rigido( rz( _, _ ) ).
```

```
entra(( OCABECA <-- CORPO )) :-
        zsepara( OCABECA, _, CABECA ),
        rigido( CABECA ),
        !,
        CABECA =.. LISTA,
        zappenda( _, LISTA, NLISTA ),
        NCABECA =.. NLISTA,
        zinicial( CORPO, NCORPO ),
        assertz(( NCABECA :- NCORPO )).

entra(( OCABECA <-- CORPO )) :-
        zsepara( OCABECA, K, CABECA ),
        !,
        CABECA =.. LISTA,
        zappenda( K, LISTA, NLISTA ),
        NCABECA =.. NLISTA,
        zinicial( CORPO, NCORPO ),
        assertz(( NCABECA :- NCORPO )).

entra(( OCABECA <== CORPO )) :-
        zsepara( OCABECA, K, CABECA ),
        !,
        CABECA =.. LISTA,
        zappenda( X, LISTA, NLISTA ),
        NCABECA =.. NLISTA,
        zpermanente( CORPO, NCORPO, X - K ),
        assertz(( NCABECA :- X >= K, NCORPO )).

/*************************************************************************
 *      Separa operadores o da formula atomica                           *
 *************************************************************************/

zsepara( o OCABECA, K, CABECA ) :-
        !,
        zsepara( OCABECA, KO, CABECA ),
        inc( KO, K ).

zsepara( CABECA, 0, CABECA ).

/*************************************************************************
 *      Inclui um elemento no fim de uma lista                           *
 *************************************************************************/

zappenda( R, [], [ R ] ) :-
        !.
```

```
zappenda( NR, [ C | R ], [ C | L ] ) :-
        zappenda( NR, R, L ).

/*************************************************************************
 *      Converte corpo de clausulas iniciais                             *
 *************************************************************************/

zinicial(( OOBJETIVO, CORPO ), ( NOOBJETIVO, NCORPO )) :-
        !,
        zobjini( OOBJETIVO, NOOBJETIVO ),
        zinicial( CORPO, NCORPO ).

zinicial( OOBJETIVO, NOOBJETIVO ) :-
        zobjini( OOBJETIVO, NOOBJETIVO ).

/*************************************************************************
 *      Converte objetivo de clausula inicial                            *
 *************************************************************************/

zobjini( OOBJETIVO, NOBJETIVO ) :-
        zsepara( OOBJETIVO, K, OBJETIVO ),
        OBJETIVO =.. LISTA,
        zappenda( K, LISTA, NLISTA ),
        NOBJETIVO =.. NLISTA.

/*************************************************************************
 *      Converte corpo de clausulas permanente                           *
 *************************************************************************/

zpermanente(( OOBJETIVO, CORPO ), ( NOOBJETIVO, NCORPO ), DELTA ) :-
        !,
        zobjperm( OOBJETIVO, NOOBJETIVO, DELTA ),
        zpermanente( CORPO, NCORPO, DELTA ).

zpermanente( OOBJETIVO, NOOBJETIVO, DELTA ) :-
        zobjperm( OOBJETIVO, NOOBJETIVO, DELTA ).

/*************************************************************************
 *      Converte objetivo de clausula permanente                         *
 *************************************************************************/

zobjperm( OOBJETIVO, (X is K + DELTA, NOBJETIVO ), DELTA ) :-
        zsepara( OOBJETIVO, K, OBJETIVO ),
        OBJETIVO =.. LISTA,
```

```
        zappenda( X, LISTA, NLISTA ),
        NOBJETIVO =.. NLISTA.

/*************************************************************************
 *      Realiza uma consulta                                             *
 *************************************************************************/

consulta( GOL, VARIAVEIS ) :-
        abolish( zgol/2 ),
        entra(( zgol( VARIAVEIS ) <== GOL )),
        zroda( 0 ).

/*************************************************************************
 *      Avalia o objetivo no instante t = T                              *
 *************************************************************************/

zroda( T ) :-
        write( 'Avaliando o objetivo para t = ' ),
        write( T ),
        nl,
        write( 'confirme: ' ),
        read_string( 10, $s$ ),
        zgol( VARIAVEIS, T ),
        !,
        write( 'variaveis: ' ),
        write( VARIAVEIS ),
        nl,
        nl,
        inc( T, T1 ),
        zroda( T1 ).
```

Anexo D

Implementação da Abordagem 2 - Versão 2.

Listamos aqui o arquivo relativo à segunda versão da implementação da abordagem 2 de Martín Abadi e Zohar Manna para lógica temporal.

```
/*************************************************************************
 *                                                                       *
 *      Abordagem Abadi & Manna para Logica Temporal (versao completa)   *
 *                                                                       *
 *      Modulo : Programa Principal                                      *
 *                                                                       *
 *      Autor  : Luiz Fernando Pereira de Souza                          *
 *                                                                       *
 *      Arquivo: tese/tempo/tempo2.pro       Data de Criacao : 23.09.91  *
 *      Versao : 1.00                        Ultima Alteracao: 30.09.91  *
 *                                                                       *
 *************************************************************************/

/*************************************************************************
 *      Define os operadores                                             *
 *************************************************************************/

:-      op( 1100, xfx, <-- ),
        op( 1100, xf , <-- ),
        op( 1100, xfx, <== ),
        op( 1100, xf , <== ),
        op(  500,  fy, o   ),
        op(  500,  fy, <>  ),
        op(  500,  fx, #   ),
        op(  500, xfy, '.' ).

/*************************************************************************
 *      Inferecia de objetivos com tempo determinado                     *
 *************************************************************************/

infere( [ <> ECORPO | CORPO ] ) :-
        !,
        einfere( ECORPO ),
        infere( CORPO ).

infere( [ K.CABECA | CORPO ] ) :-
        clausula( J.CABECA, <--, NCORPO ),
        (
                rigido( CABECA )
        ;
                J = K
        ),
        ordena( NCORPO, CORPO, FCORPO ),
        infere( FCORPO ).
```

```
infere( [ _.CABECA | CORPO ] ) :-
        clausula( _.CABECA, <==, NCORPO ),
        rigido( CABECA ),
        einfere( NCORPO ),
        infere( CORPO ).

infere( [ K.CABECA | CORPO ] ) :-
        !,
        clausula( J.CABECA, <==, NCORPO ),
        J =< K,
        D is K - J,
        avanca( NCORPO, D, ACORPO ),
        ordena( ACORPO, CORPO, FCORPO ),
        infere( FCORPO ).

infere( [] ).

/*************************************************************************
 *      Ordena objetivos restantes, os com tempo determinado primeiro    *
 *************************************************************************/

ordena( [ K1.CAB1 | CRP1 ], LISTA2, [ K1.CAB1 | FCRP ] ) :-
        !,
        ordena( CRP1, LISTA2, FCRP ).

ordena( [ <> ECP | CRP1 ], [ K2.CAB2 | CRP2 ], [ K2.CAB2 | FCRP ] ) :-
        !,
        ordena( [ <> ECP | CRP1 ], CRP2, FCRP ).

ordena( [ <> EC1 | CRP1 ], LISTA2, [ <> EC1 | LISTA ] ) :-
        !,
        junta( CRP1, LISTA2, LISTA ).

ordena( [], LISTA2, LISTA2 ).

/*************************************************************************
 *      Junta duas listas                                                *
 *************************************************************************/

junta( [], L, L ) :-
        !.

junta( [ C | R ], L, [ C | NL ] ) :-
        junta( R, L, NL ).
```

```
/*************************************************************************
 *      Avanca um objetivo no tempo, (incluindo mais operadores 'O's)    *
 *************************************************************************/

avanca( [ <> ECORPO | CORPO ], D, [ <> NECORPO | NCORPO ] ) :-
        !,
        avanca( ECORPO, D, NECORPO ),
        avanca( CORPO, D, NCORPO ).

avanca( [ K.CABECA | CORPO ], D, [ NK.CABECA | NCORPO ] ) :-
        !,
        NK is K + D,
        avanca( CORPO, D, NCORPO ).

avanca( [], _, [] ).

/*************************************************************************
 *      Inferecia de objetivos com tempo indeterminado                   *
 *************************************************************************/

einfere( [ <> CABECA | CORPO ] ) :-
        !,
        einfere( CABECA ),
        einfere( CORPO  ).

einfere( [ _.CABECA | CORPO ] ) :-
        clausula( _.CABECA, <--, NCORPO ),
        rigido( CABECA ),
        infere( NCORPO ),
        einfere( CORPO ).

einfere( [ K.CABECA | CORPO ] ) :-
        clausula( J.CABECA, <--, NCORPO ),
        J >= K,
        D is J - K,
        infere( NCORPO ),
        avanca( CORPO, D, ACORPO ),
        infere( ACORPO ).

einfere( [ _.CABECA | CORPO ] ) :-
        rigido( CABECA ),
        clausula( _.CABECA, <==, NCORPO ),
        einfere( NCORPO ),
        einfere( CORPO ).
```

```
einfere( [ K.CABECA | CORPO ] ) :-
        clausula( J.CABECA, <==, NCORPO ),
        J >= K,
        D is J - K,
        avanca( CORPO, D, ACORPO ),
        ordena( NCORPO, ACORPO, FCORPO ),
        einfere( FCORPO ).

einfere( [ K.CABECA | CORPO ] ) :-
        clausula( J.CABECA, <==, NCORPO ),
        J < K,
        D is K - J,
        avanca( NCORPO, D, ACORPO ),
        ordena( ACORPO, CORPO, FCORPO ),
        einfere( FCORPO ).

einfere( [] ).

/*************************************************************************
 *      Entrada de clausulas                                             *
 *************************************************************************/

entra(( # CABECA <-- )) :-
        !,
        CABECA =.. [ PREDICADO | ARGUMENTOS ],
        ctr_inc( n, N ),
        entra(( CABECA <== rigido_especial( N, ARGUMENTOS ))),
        entra(( rigido_especial( N, ARGUMENTOS ) <-- )).

entra(( # CABECA <-- CORPO )) :-
        !,
        CABECA =.. [ PREDICADO | ARGUMENTOS ],
        ctr_inc( n, N ),
        entra(( CABECA <== rigido_especial( N, ARGUMENTOS ))),
        entra(( rigido_especial( N, ARGUMENTOS ) <-- CORPO )).

rigido( rigido_especial( _, _ ) ).

entra(( OCABECA <-- )) :-
        !,
        separa( OCABECA, K, CABECA ),
        assertz( clausula( K.CABECA, <--, [] ) ).
```

```
entra(( OCABECA <-- CORPO )) :-
        !,
        separa( OCABECA, K, CABECA ),
        corpo( CORPO, LISTA ),
        assertz( clausula( K.CABECA, <--, LISTA ) ).

entra(( OCABECA <== )) :-
        !,
        separa( OCABECA, K, CABECA ),
        assertz( clausula( K.CABECA, <==, [] ) ).

entra(( OCABECA <== CORPO )) :-
        separa( OCABECA, K, CABECA ),
        corpo( CORPO, LISTA ),
        assertz( clausula( K.CABECA, <==, LISTA ) ).

/*************************************************************************
 *      Separa operadores o da formula atomica                           *
 *************************************************************************/

separa( o OCABECA, K, CABECA ) :-
        !,
        separa( OCABECA, K0, CABECA ),
        inc( K0, K ).

separa( CABECA, 0, CABECA ).

/*************************************************************************
 *      Converte o corpo de uma clausula                                 *
 *************************************************************************/

corpo( CORPO, LISTA ) :-
        corpox( CORPO, LISTAX ),
        ordena( LISTAX, LISTA ).

/*************************************************************************
 *      Convercao incial do corpo da clausula sem a ordenacao            *
 *************************************************************************/

corpox(( OBJETIVO, CORPO ), [ NOBJETIVO | NCORPO ] ) :-
        !,
        objetivo( OBJETIVO, NOBJETIVO ),
        corpox( CORPO, NCORPO ).
```

```
corpox( OBJETIVO, [ NOBJETIVO ] ) :-
        objetivo( OBJETIVO, NOBJETIVO ).

/*************************************************************************
 *      Converte um objetivo do corpo de uma clausula                    *
 *************************************************************************/

objetivo( OOBJETIVO, <> LISTA ):-
        separa( OOBJETIVO, K, <> CORPO ),
        !,
        corpo( CORPO, CLISTA ),
        avanca( CLISTA, K, LISTA ).

objetivo( OOBJETIVO, K.OBJETIVO ):-
        separa( OOBJETIVO, K, OBJETIVO ).

/*************************************************************************
 *      Ordenacao da lista gerada para representar o corpo da clausula   *
 *************************************************************************/

ordena( LISTA, ORDENADA ) :-
        divide( LISTA, LISTA_OBJ, LISTA_EVE ),
        junta( LISTA_OBJ, LISTA_EVE, ORDENADA ).

/*************************************************************************
 *      Separacao da lista em termos com e sem o operador <>             *
 *************************************************************************/

divide( [ <> LISTA  | RESTO ], LISTA_OBJ, [ <> LISTA | LISTA_EVE ] ) :-
        !,
        divide( RESTO, LISTA_OBJ, LISTA_EVE ).

divide( [], [], [] ) :-
        !.

divide( [ OOBJETIVO | RESTO ], [ OOBJETIVO | LISTA_OBJ ], LISTA_EVE ) :-
        divide( RESTO, LISTA_OBJ, LISTA_EVE ).

/*************************************************************************
 *      Realiza uma consulta                                             *
 *************************************************************************/

consulta( GOL, VARIAVEIS ) :-
        corpo( GOL, LGOL ),
        ctr_set( t, -1 ),
```

```
        testa( T ),
                avanca( LGOL, T, TGOL ),
                resolva( TGOL ),
                write( 'variaveis: ' ),
                write( VARIAVEIS ),
                nl,
                nl,
        fail.

/*************************************************************************
 *      Verifica se deve inferir as consulta para o instante T           *
 *************************************************************************/

testa( _ ) :-
        ctr_is( t, T0 ),
        inc( T0, T ),
        ctr_set( t, T ),
        write( 'Avaliando o objetivo para t = ' ),
        write( T ),
        nl,
        write( 'confirme: ' ),
        read_string( 10, S ),
        S \= $s$,
        !,
        fail.

testa( T ) :-
        ctr_is( t, T ).

testa( T ) :-
        testa( T ).

/*************************************************************************
 *      Garante que cada instante T so seja avaliado uma unica vez       *
 *************************************************************************/

resolva( GOL ) :-
        infere( GOL ),
        !.

resolva( _ ) :-
        write( 'solucao nao encontrada' ),
        nl.
```