@inteligência-artificial

Mestrado em Informática - UFPR

Diogo Cezar Teixera Batista
Marcos Castilho

Curitiba

12 de maio de 2010

Sumário

1 Informações da Disciplina

- Nome: *Inteligência Artificial - opt. CI 311A*;
- Código: *CI 727*;
- Professor: *Marcos Castilho*;
- Horários: *2ª das 13:30 as 15:10 / 4ª das 13:30 as 15:10*;
- Método de Avaliação:
  - Seminário;
  - Entrega de Relatório Técnico;
  - Implementação;
  - Prova.
- Página da disciplina: *www.inf.ufpr.br/marcos/ci311*

2 Lógica

A lógica é uma ciência de índole matemática e fortemente ligada à Filosofia. Já que o pensamento é a manifestação do conhecimento, e que o conhecimento busca a verdade, é preciso estabelecer algumas regras para que essa meta possa ser atingida. Assim, a lógica é o ramo da filosofia que cuida das regras do bem pensar, ou do pensar correto, sendo, portanto, um instrumento do pensar.

Um sistema lógico é um conjunto de *axiomas*[1] e regras de *inferência*[2] que visam representar formalmente o raciocínio válido. Diferentes sistemas de lógica formal foram construídos ao longo do tempo quer no âmbito escrito da Lógica Teórica, quer em aplicações práticas na computação e em Inteligência artificial.

Tradicionalmente, lógica é também a designação para o estudo de sistemas prescritivos de raciocínio, ou seja, sistemas que definem como se "deveria" realmente pensar para não errar, usando a razão, dedutivamente e indutivamente. A forma como as pessoas realmente raciocinam é estudado nas outras áreas, como na psicologia cognitiva.

Como ciência, a lógica define a estrutura de declaração e argumento para elaborar fórmulas através das quais estes podem ser codificados. Implícita no estudo da lógica está a compreensão do que gera um bom argumento e de quais argumentos são falaciosos.

A lógica filosófica lida com descrições formais da linguagem natural. A maior parte dos filósofos assumem que a maior parte do raciocínio "normal"pode ser capturada pela lógica, desde que se seja capaz de encontrar o método certo para traduzir a linguagem corrente para essa lógica.

2.1 Lógica Proposicional

Em lógica e matemática, uma lógica proposicional é um sistema formal no qual as fórmulas representam *proposições*[3] que podem ser formadas pela combinação de proposições atômicas usando conectivos lógicos e um sistema de regras de derivação, que permite que certas fórmulas sejam estabelecidas como "teoremas" do sistema formal.

Em termos gerais, um cálculo é frequentemente apresentado como um sistema formal que

[1] Um axioma é uma sentença ou proposição que não é provada ou demonstrada e é considerada como óbvia ou como um consenso inicial necessário para a construção ou aceitação de uma teoria. Por essa razão, é aceito como verdade e serve como ponto inicial para dedução e inferências de outras verdades (dependentes de teoria).

[2] Em lógica, inferência é a passagem, através de regras válidas, do antecedente ao conseqüente de um argumento.

[3] Proposição é um termo usado em lógica para descrever o conteúdo de asserções. Uma asserção é um conteúdo que pode ser tomado como verdadeiro ou falso.

consiste em um conjunto de expressões sintáticas (fórmulas bem formadas, ou fbfs), um subconjunto distinto dessas expressões, e um conjunto de regras formais que define uma relação binária específica, que se pretende interpretar como a noção de equivalência lógica, no espaço das expressões.

Quando o sistema formal tem o propósito de ser um sistema lógico, as expressões devem ser interpretadas como asserções matemáticas, e as regras, conhecidas como regras de inferência, normalmente são preservadoras da verdade. Nessa configuração, as regras (que podem incluir axiomas) podem então ser usadas para derivar "inferir" fórmulas representando asserções verdadeiras.

O conjunto de axiomas pode ser vazio, um conjunto finito não vazio, um conjunto finito enumerável, ou pode ser dado por axiomas esquemáticos. Uma gramática formal define recursivamente as expressões e fórmulas bem formadas (fbfs) da linguagem. Além disso, pode se apresentar uma semântica para definir verdade e valorações (ou interpretações).

A linguagem de um cálculo proposicional consiste em:

1. um conjunto de símbolos primitivos, definidos como fórmulas atômicas, proposições atômicas, ou variáveis, e
2. um conjunto de operadores, interpretados como operadores lógicos ou conectivos lógicos.

Uma fórmula bem formada (fbf) é qualquer fórmula atômica ou qualquer fórmula que pode ser construída a partir de fórmulas atômicas, usando conectivos de acordo com as regras da gramática.

EXEMPLO 1: Para exemplificar as vertentes descritas da lógica proposicional, imaginemos o seguinte exemplo:

Um homem parado em frente a um muro de concreto.

Imaginemos as seguintes afirmações:

i Ele avançou um pass;

ii Ele avançou um lago;

iii Ele avançou um passo;

iv Ele recuou um passo.

A afirmação *i* apresenta um erro de linguagem pois a palavra "pass" não existe na lingua portuguesa.

A afirmação *ii* está escrita de forma correta, tanto do ponto de vista da linguagem quando da semântica, entretanto ao se inserir a afirmação no contexto, ela perde seu sentido, pois de acordo com uma interpretação racional, na situação, não faria sentido um homem avançar um lago, o que torna a afirmação falsa.

A afirmação *iii* assim como a afirmação *ii* está escrita de forma correta, entretanto uma interpretação no contexto apresentado também retornaria uma afirmação falsa, pois um muro impede o homem de dar um passo a frente.

A afirmação *iv* está escrita também de forma correta, e uma interpretação para ela no contexto apresentado retornaria uma afirmação verdadeira.

EXEMPLO 2: Imaginando as seguintes sentenças:

p Marcos é professor;

q Marcos não sabe dar aulas;

r O Brasil foi campeão em 2006;

t Maria é professora.

Independente da interpretação que cada uma das sentenças represente, não podemos dizer que a sentença p e q ou p e t (apesar de apresentarem elementos iguais (Marcos, professor(a))), tem uma relação do ponto de vista da lógica proposicional'.

O que segue apresenta uma definição uma lógica proposicional:

- de sua linguagem, que é a coleção particular de símbolos primitivos e operadores,
- do conjunto de axiomas, ou fórmulas distinguidas, e
- do conjunto de regras de inferência (Semântica).

2.1.1 Linguagem

A linguagem se refere ao modo como se escreve uma sentença, em linguagem de programação por exemplo essa se refere as regras de escritas estabelecidas pelas linguagens de programação.

Por exemplo:

- Escrever o comando `els` é um erro de linguagem, pois o correto seria escrever `else`;
- Escrever `printf('olá mundo")` também é um erro de linguagem, pois é definido o padrão de se utilizar abertura e fechamento de aspas simples ou duplas.

2.1.2 Semântica

A semântica refere-se ao estudo do significado, em todos os sentidos do termo. A semântica opõe-se com frequência à sintaxe, caso em que a primeira se ocupa do que algo significa, enquanto a segunda se debruça sobre as estruturas ou padrões formais do modo como esse algo é expresso (por exemplo, escritos ou falados).

2.1.3 Axiomática

O termo axiomática designa o conjunto dos pressupostos iniciais e das proposições que deles derivam.

Os axiomas foram transformados numa espécie de leis aceites sem discussão, ignorando a necessidade de evidência como critério de seleção. A partir dos axiomas é possível, através de exercícios de dedução, obter todos os teoremas de uma determinada ciência, ou seja, a axiomática tem por objetivo através da manipulação de seus axiomas, provar que determinado teorema.

3 Linguagem

A linguagem especifica quais serão os elementos trabalhos para representar as sentenças afim gerar um resultado.

3.1 Proposição

Uma proposição é uma declaração afirmativa à qual se pode associar um valor verdadeiro ou falso, mas não ambos.

Por exemplo, "*O Brasil fica na América*" é uma proposição verdadeira, enquanto "*A lua é de queijo*" é uma proposição falsa.

A proposição é o elemento básico a partir do qual os argumentos são construídos, sendo também o principal objeto de estudo na lógica proposicional.

3.2 Símbolos proposicionais

São os símbolos que irão representar nossas sentenças, são representados por letras minúsculas do alfabeto, possivelmente indexadas e normalmente atribuídos as letras finais do alfabeto.

Exemplo: $p, q, r, p_1, q_1, r_1 \ldots$

3.3 Conectivos

São os elementos que executam as operações com os símbolos proposicionais.

- $\wedge$ - corresponde ao conectivo E (conjunção);
- $\vee$ - corresponde ao conectivo OU (disjunção);
- $\neg$ - corresponde ao conectivo NOT (negação);
- $\rightarrow$ - corresponde ao conectivo de atribuição ou IMPLICA EM (implicação);
- $\leftrightarrow$ - corresponde ao conectivo SE E SOMENTE SE ou BI-IMPLICAÇÃO (dupla implicação);

3.4 Constantes

Existem também a representação para as constantes *verdadeiro* ou *true* ($\top$) e *falso* ou *false* ($\bot$).

3.5 Separadores

Afim de agrupar a precedência as sentenças denotaremos o uso de parênteses $(\ldots)$.

3.6 Fórmulas

Símbolos proposicionais são *fórmulas atômicas*[4]. Essas fórmulas são normalmente representadas pelas letras iniciais do alfabeto e em letras maiúsculas.

Exemplo: $A, B \ldots$

O conjunto de símbolos proposicionais dão origem a uma fórmula ou FOR ou fórmula bem formada(fbf).

- $\top$ e $\bot$ estão em uma fórmula;

[4]Fórmula atômica ou átomo, na Lógica matemática, é uma fórmula que não pode ser dividida em subfórmulas, ou seja, uma fórmula sem conectivos que representa uma Proposição.

- Fórmulas atômicas estão em uma fórmula;
- Se A ou B estão em uma fórmula então também estão:
  - $(A \wedge B)$;
  - $(A \vee B)$;
  - $(A \rightarrow B)$;
  - $(A \leftrightarrow B)$;
  - $(\neg A)$.

3.7 Subfórmula

É possível identificar subfórmulas dentro de uma fórmula descrita, seja $(A \wedge B)$ ou $(A \vee B)$ ou $(A \rightarrow B)$ ou $(A \leftrightarrow B)$ ou $(\neg A)$, então A e B são subfórmulas.

Por exemplo, consideremos a fórmula $((p \wedge q) \rightarrow r)$:

p, q, r, $(p \wedge q)$ e $((p \wedge q) \rightarrow r)$ são subfórmulas.

3.8 Substituição Uniforme

A substituição uniforme é a troca de determinado trecho de código por uma fórmula ou seu significado, afim de simplificar ou demonstrar a representação de uma fórmula geral.

Seja A uma fórmula contendo uma subfórmula B podemos fazer a substituição de B por outra subfórmula C: $A[B/C]$.

Exemplo: $A : ((p \rightarrow q) \wedge (r \vee s))$.

Se definirmos r como $t_1 \rightarrow t_2$, então:

$A[r/t_1 \rightarrow t_2] : ((p \rightarrow q) \wedge ((t_1 \rightarrow t_2) \vee s))$.

4 Teoria dos Modelos (Semântica)

Na Teoria dos Modelos (Semântica) podemos representar os conectivos apresentados anteriormente pelas respectivas tabelas verdade:

Conectivo E pode ser representando pela Tabela 1.

O conectivo OU pode ser representando pela Tabela 2

O conectivo NOT pode ser representando pela Tabela 3.

O conectivo IMPLICA EM pode ser representando pela Tabela 4.

O conectivo de BI-IMPLICAÇÃO pode ser representando pela Tabela 5.

Tabela 1: Tabela verdade do conectivo E

Entrada1	Entrada2	Saída
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

Tabela 2: Tabela verdade do conectivo OU

Entrada1	Entrada2	Saída
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

Tabela 3: Tabela verdade do conectivo NO

Entrada1	Entrada2	Saída
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	VERDADEIRO

Tabela 4: Tabela verdade do conectivo de IMPLICAÇÃO

Entrada1	Entrada2	Saída
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	VERDADEIRO

Com a teoria dos modelos ou semântica, atribuímos interpretações para as fórmulas representadas.

A semântica é o estudo da representação de conceitos matemáticos. É assumido que existam fórmulas pre-existente, e investiga-se o que pode ser concluído de tal coleção de fórmula, algumas operações e/ou relações entre estes objetos, e alguns axiomas.

Tabela 5: Tabela verdade do conectivo de BI-IMPLICAÇÃO

Entrada1	Entrada2	Saída
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	VERDADEIRO

4.1 Interpretação

É uma função (I) do conjunto de símbolos proposicionais em $\{0, 1\}$.

Então I tem o conjunto de átomos (ATM): $I : ATM \rightarrow \{0, 1\}$.

Exemplo: adotemos o exemplo de $ATM = \{p, q, r\}$,

então,

as soluções possíveis para esse grupo de símbolos proposicionais é:

$$I(p) = 0, I(q) = 0, I(r) = 0;$$
$$I(p) = 0, I(q) = 0, I(r) = 1;$$
$$I(p) = 0, I(q) = 1, I(r) = 1;$$
$$I(p) = 0, I(q) = 1, I(r) = 0;$$
$$I(p) = 1, I(q) = 1, I(r) = 1;$$
$$I(p) = 1, I(q) = 1, I(r) = 0;$$
$$I(p) = 1, I(q) = 0, I(r) = 0;$$
$$I(p) = 1, I(q) = 0, I(r) = 1;$$

O número de resultado possíveis para preencher toda a tabela verdade é de ordem 2^{β}, onde:

- 2 é o número de resultados possíveis em $ATM : \{0, 1\}$.
- β é o número de símbolos proposicionais na fórmula testada.

4.2 Interpretação das Fórmulas

Podemos representar um padrão matemático (Interpretação $I(\ldots)$) para cada um de nossos conectivos partindo do seguinte princípio:

Se $I(p \wedge q) = 1$ SE e SOMENTE SE $I(p) = 1$ e $I(q) = 1$. Caso contrário, $I(p \wedge q) = 0$.

assim, $I(p \wedge q) = MIN\{I(p), I(q)\}$.

Se p E q são $\top$ então pode-se assumir que pegando o mínimo valor em todas as opções de resposta obtêm-se a resposta para a expressão definida:

$$min\{0,0\} = 0$$
$$min\{0,1\} = 0$$
$$min\{1,0\} = 0$$
$$min\{1,1\} = 1$$

outras definições para os demais conectores:

- $I(p \vee q) = MAX\{I(p), I(q)\}$;
- $I(\neg p) = 1 - I(p)$;
- $I(p \rightarrow q) = MAX\{I(\neg p), I(q)\}$;
- $I(p \leftrightarrow q) = \begin{cases} 1 & \text{se} \quad I(p) = I(q) \\ 0 & \text{c.c.;} \end{cases}$
- $I(\top) = 1$;
- $I(\bot) = 0$;

4.3 Modelo

Uma interpretação ($I(\ldots)$) é um modelo para uma fórmula A quando $I(A) = 1$.

Exemplo: $((p \vee q) \rightarrow r)$;

Se as entradas I_1 para p, q e r forem respectivamente: 1,0,1 então I_1 é modelo.

Mas se as entradas I_2 para p, q e r forem respectivamente: 1,1,0 então I_2 não é modelo.

4.4 Validade

Uma fórmula A é valida se para *toda* interpretação I, $I(A) = 1$.

Exemplo: A é $p \rightarrow (q \rightarrow p)$.

Sua tabela verdade é dada por:

$p = 0$ e $q = 1$ então $I(A)$ é: 1
$p = 0$ e $q = 0$ então $I(A)$ é: 1
$p = 1$ e $q = 1$ então $I(A)$ é: 1
$p = 1$ e $q = 0$ então $I(A)$ é: 1

4.5 Satisfabilidade

O problema de satisfabilidade é um problema do tipo NP-COMPLETO, e tem sido objeto de estudos de muitos algoritmos afim de achar ao menos uma solução satisfasível em uma gama de possibilidades.

Uma fórmula A é satisfasível (SAT) se existe *pelo menos* uma interpretação I para a qual $I(A) = 1$.

Tendo, $((p \vee q) \rightarrow p)$ como uma fórmula SAT mas não válida;

Tendo, $p \rightarrow (q \rightarrow p)$ como uma fórmula SAT e válida.

4.6 Consequência Lógica

Uma fórmula é consequência lógica de um conjunto de fórmulas $A_1, A_2, A_3, \ldots, A_n$ (denotado $((A_1, A_2, A_3, \ldots, A_n) \models A)$) quando A for válida sempre que $A_1, A_2, A_3, \ldots, A_n$ também forem.

Por exemplo, diz-se que "Caco é verde" é uma consequência lógica de "todos os sapos são verdes" e "Caco é um sapo", porque seria "auto-contraditório" afirmar estas últimas sentenças e negar a primeira.

5 Teoria da Prova (Axiomática)

É um conjunto de fórmulas derivável de outro conjunto de fórmulas.

A semântica e a axiomática formam um conjunto distinto de tentativas de provar se uma determinada fórmula é ou não é valida.

A semântica utiliza um conjunto de preposições (combinações de valores verdadeiros ou falsos) para testar a fórmula em questão, enquanto que a axiomática utiliza regras pré-deduzidas para provar se uma fórmula é válida sem a necessidade de procurar por todas as suas ocorrências no conjunto de preposições.

Dentre as técnicas utilizadas para se obter uma prova axiomática estão:

- Sistema dedutivo de Hilbert;
- Resolução;
- Tableau;
- Sequentes.

As teorias de prova podem ser constituídas de regras axiomáticas ou de inferência, sendo possível ter 0 ou n regras axiomáticas e pelo menos uma regra de inferência.

5.1 Sistema Dedutivo de Hilbert

Na axiomática de Hilbert são definidas 11 axiomáticas e 1 regra de inferência conhecida por *Modus Ponens*.

DEFINIÇÂO: São os seguintes esquemas de axiomas da lógica proposicional:

A01 $A \rightarrow (B \rightarrow A)$;

A02 $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$;

A03 $A \rightarrow (B \rightarrow A \wedge B)$;

A04 $(A \wedge B) \rightarrow A$;

A05 $(A \vee B) \rightarrow B$;

A06 $A \rightarrow A\neg B$;

A07 $B \rightarrow B\neg A$;

A08 $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A\neg B \rightarrow C))$;

A09 $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$;

A10 $\neg\neg A \rightarrow A$;

A11 $\neg\bot$.

5.1.1 Modus Ponens

DEFINIÇÂO: É a regra de inferência adotada para essa lógica proposicional.

Pode ser definida literalmente por:

Se P, então Q.

P.

Portanto Q.

ou, em notação lógica:

$$p \rightarrow q$$

$$p \therefore q$$

Onde o sinal $\therefore$ representa o ato de concluir q a partir de $p \rightarrow q$ e p. Este sinal pode ser lido como "portanto".

O argumento tem duas premissas. A primeira premissa é a condição "se - então", nomeadamente que p implica q. A segunda premissa é que p é verdadeiro. Destas duas premissas pode ser logicamente concluído que q tem de ser também verdadeiro.

Um exemplo literal:

Se chover, então fico em casa.

Choveu.

Então fico em casa.

5.2 Exemplo utilizando axiomática de Hilbert e Modus Ponens

Para exemplificar como se prova um teorema pelas axiomáticas de Hilbert em conjunto com a regra *Modus Ponens* vamo utilizar a fórmula $p \rightarrow p$.

Vamos utilizar primeiramente as axiomáticas A_{01} e A_{02}:

A01 $A \rightarrow (B \rightarrow A)$;

A02 $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$;

Agora devemos criar instâncias desses axiomas substituindo de forma empírica as preposições A, B e C por p, $p \rightarrow p$ e p respectivamente para a IA_{02} e A e B por p e p para a instância IA_{01}.

Então temos,

IA01 $p \rightarrow (p \rightarrow p)$;

IA02 $(p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow ((p \rightarrow p) \rightarrow p) \rightarrow (p \rightarrow p))$;

Aplicamos *Modus Ponens* nessas instâncias e obtemos:

MP $p \rightarrow ((p \rightarrow p) \rightarrow p) \rightarrow (p \rightarrow p)$;

Instanciando novamente A_{01} temos:

IA01 $p \rightarrow (p \rightarrow p) \rightarrow p$;

E finalmente aplicando o *Modus Ponens* nas duas últimas fórmulas temos $p \rightarrow p$.

5.3 Instanciando um Axioma

Um esquema de axioma instânciado é chamado de *instância de axioma*.

Por exemplo se utilizássemos o axioma A06 de Hilbert, poderíamos representar:

$p \rightarrow p \neg q$;

5.4 Prova ($\vdash$)

DEFINIÇÃO: Seja A uma fórmula; uma prova de A é uma lista finita de fórmulas: $(A_1, A_2, \ldots, A_n)$, tais que:

- $An = A$;
- A_i é:
  - Uma instância de axioma;
  - Foi obtido pela aplicação da regra *Modus Ponens*.
- A é provável $\vdash A$ se existe uma prova para A;
- A é consistente se $\neg A$ não é provável;
- Se não é inconsistente.

Isso demonstra que para que uma fórmula A seja provada é necessário que o contrário dessa fórmula seja negado.

Para se provar uma fórmula de forma axiomática, é necessário fazer uma derivação de um conjunto de axiomas (no caso os axiomas de Hilbert) e pelo menos uma regra de inferência (no caso Modus Ponens).

As derivações devem ocorrer de forma a casar as duas vertentes e formar assim uma prova para determinada fórmula.

5.4.1 Propriedades

As propriedades são teoremas definidos para definir um conjunto de fórmulas derivadas válidas e/ou prováveis.

5.4.2 Adequação

LEMA: A regra *Modus Ponens* preserva a validade.

- $\models A$
- $\models A \rightarrow B$ então,
- $\models B$

Pois se todo o conjunto de A é válido, então qualquer valor que é implicado ($\rightarrow$) de um valor $\top$ (mesmo que $\bot$) também é $\top$. Veja Tabela 4

5.4.3 Substituição

LEMA: A substituição uniforme preserva a validade:

Se A e B são fórmulas e A sua proposição se $\models A$ então $\models A[P/B]$.

Isso significa que garantindo que a fórmula A seja válida, então não importa a substituição de P por B baseando-se no lema da *adequação*.

5.5 Teoremas

Esses teoremas demonstram a conexão das duas vertentes (semântica e axiomática) pois a semântica é capaz de provar através de um caso completo de testes se uma determinada fórmula é *válida*, enquanto que a axiomática consegue através de deduções de fórmulas chegar a uma *prova* consistente.

A questão principal é que uma fórmula valida é provável, e uma fórmula provável é valida, para os problemas de lógica proposicional.

5.5.1 Teorema da Adequação

Se A é provável ($\vdash A$) então A é válido ($\models A$).

5.5.2 Teorema da Completude

Se A é válido ($\models A$) então A é provável ($\vdash A$).

5.5.3 Teorema da Dedução

$A \vdash B$ se e somente se $\vdash A \rightarrow B$

B é *provável* aceitando-se A como premissa se e somente se A for válido implicando em B.

5.5.4 Teorema da Consequência Lógica

$A \models B$ se e somente se $\models A \rightarrow B$

B é *válido* aceitando-se A como premissa se e somente se A for válido implicando em B.

5.5.5 Decibilidade

É a existência de um procedimento de decisão.

A lógica proposicional é decidível, isso é, existe um procedimento efetivo que para qualquer fórmula A dada como entrada. Para A sempre haverá uma resposta. $\top$ se A é válido e $\bot$ em caso contrário.

Não importa quanto tempo essa resposta demore, a lógica proposicional garante que haverá uma resposta.

5.6 Equivalências

As equivalências servem para dedução ou em caráter mais específico redução, de uma fórmula maior em uma fórmula menor.

- $\vdash A \rightarrow B \leftrightarrow \neg A \wedge B$;
- $\vdash (A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$;
- $\vdash \bot \leftrightarrow P \wedge \neg P$;
- $\vdash \neg A \leftrightarrow A \rightarrow \bot$;
- $\vdash A \vee B \leftrightarrow \neg(\neg A \wedge \neg B)$;
- $\vdash A \vee B \leftrightarrow (A \rightarrow \bot) \rightarrow B$;
- $\vdash A \wedge B \leftrightarrow \neg(\neg A \vee \neg B)$;
- $\vdash A \wedge B \leftrightarrow (A \rightarrow (B \rightarrow \bot)) \rightarrow \bot$;
- $\vdash \neg\neg A \leftrightarrow A$;
- $\vdash A \wedge A \leftrightarrow A$;
- $\vdash A \vee A \leftrightarrow A$;
- $\vdash A \wedge (B \wedge C) \leftrightarrow (A \wedge B) \wedge C$;

- $\vdash A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C$;
- $\vdash A \wedge B \leftrightarrow B \wedge A$;
- $\vdash A \vee B \leftrightarrow B \vee A$;

5.7 Teorema de Morgan

O Teorema de Morgan define regras usadas para converter operações lógicas $\vee$ em $\wedge$ e vice versa.

- $\vdash (A \vee B) \wedge C \leftrightarrow (A \wedge C) \vee (B \wedge C)$;
- $\wedge(A \wedge B) \vee C \leftrightarrow (A \vee C) \wedge (B \vee C)$;

5.8 Fórmula Normal Conjuntiva

Essa é uma forma de escrita que traduz as fórmulas numa linguagem mais agradável para seu processamento.

DEFINIÇÃO: Um literal é uma proposição de sua negação (fórmula atômica). ($[q, \neg q]$).

Uma cláusula é uma disjunção de literais.

Exemplo:

$$p \wedge \neg r \wedge s \wedge \neg t.$$
$$x \rightarrow (p \vee q). \text{ estaria errado.}$$

Uma fórmula está em sua forma normal conjuntiva (FNC) quando for uma conjunção de cláusulas.

Usamos:

- $L_1, L_2, \ldots$ para literais;
- $C_1, C_2, \ldots$ para cláusulas;

Exemplo:

$p \wedge (\neg p \vee q) \wedge (r \vee \neg s \vee \neg r) \wedge \top$

Podemos classificar em:

- C_1 - p;

- C_2 - $(\neg p \vee q)$;
- C_3 - $(r \vee \neg s \vee \neg r)$;
- C_4 - $\top$;

As cláusulas devem ser compostas por elementos atômicos;

As cláusulas devem ser ligadas pelo conectivo $\vee$;

5.9 Transformação em FNC

Deve-se aplicar equivalências a fórmula original para obter uma fórmula FNC.

Exemplos:

C1 $\neg(p \wedge q) \therefore \neg p \vee \neg q$;

C2 $\neg(p \vee q) \therefore \neg p \wedge \neg q$;

C3 $p \rightarrow (q \wedge r) \therefore \neg p \vee (q \wedge r) \therefore (\neg q \vee q) \wedge (\neg p \vee p)$;

5.10 Princípio da Resolução

O princípio da resolução é uma regra de inferência que dá origem a uma técnica de demonstração por *refutação* para sentenças e inferências da lógica proposicional e da lógica de primeira ordem.

Foi escrita em 1965 por J. Alan Robinson.

O princípio foi baseado do literal puro do algorítmo de Davis-Putnam.

Se duas cláusulas:

C1 $A \vee B$;

C2 $\neg B \vee C$.

são verdadeiras, então:

1. ou A é $\top$;
2. ou C é $\top$;
3. ou então ambos são $\top$.

DEFINIÇÃO: Um literal pode ser definido como um símbolo ou sua negação.

DEFINIÇÃO: Uma cláusula é uma fórmula do tipo $L_1 \vee L_2 \vee L_3 \vee \ldots \vee L_n$ com $(n > 0)$ onde cada L_i é uma literal com $(1 \leqslant i \leqslant n)$

DEFINIÇÃO: Uma fórmula está em FNC se é da forma $C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_m$ com $(m > 0)$ onde cada C_i com $(1 \leqslant i \leqslant m)$ é uma cláusula.

5.10.1 Regra da Resolução

Sejam as fórmulas:

F1 $p \vee \alpha$;

F2 $\neg p \vee \beta$.

verdadeiras, então: $\alpha \vee \beta$

α e β podem ser sub-fórmulas que podem ser por exemplo:

- $p \vee q \vee \neg r$, tendo como $\alpha \equiv q \vee \neg r$;
- $\neg p \vee s \vee t \vee w$ tendo como $\beta \equiv s \vee t \vee w$.

α e β são cláusulas e p é um símbolo proposicional.

Para provar α iniciamos com $\neg\alpha$, que deve ser transformando primeiramente para FNC.

5.10.2 Exemplo de Prova por Resolução

Vamos provar de forma refutacional o seguinte temorema: $\neg(p \vee q) \rightarrow (\neg p \wedge \neg q)$.

Primeiro passo: Negar a fórmula, então:

$\neg(\neg(p \vee q) \rightarrow (\neg p \wedge \neg q))$

Segundo passo: Deduzir a fórmula de modo a chegar a sua forma normal:

C1 $\neg(\neg(p \vee q) \rightarrow (\neg p \wedge \neg q)) \therefore$

C2 $\neg(\neg\neg(p \vee q) \vee (\neg p \wedge \neg q)) \therefore$

C3 $\neg((p \vee q) \vee (\neg p \wedge \neg q)) \therefore$

C4 $\neg(p \vee q) \wedge \neg(\neg p \wedge \neg q) \therefore$

C5 $\neg p \wedge \neg q \wedge (\neg\neg p \vee \neg\neg q) \therefore$

C6 $\neg p \wedge \neg q \wedge p \vee q$

Para chegar a cláusula $C2$ utilizamos a regra de equivalência $A \rightarrow B \equiv \neg A \vee B$ na cláusula $C1$ assumindo que $\neg(p \vee q)$ é A e $\neg p \wedge \neg q$ é B.

Para chegar a cláusula $C4$ utilizamos a regra de equivalência $A \vee B \equiv \neg(\neg A \wedge \neg B)$ que pode ser escrita como $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ na cláusula $C3$ assumindo que p é A e q é B.

A cláusula $C6$ está em sua forma normal, e também pode ser representada pela notação: $< \neg p, \neg q, p \vee q >$

Na sua forma normal podemos desmembrar o teorema em 3 cláusulas:

C1 $\neg p$;

C2 $\neg q$;

C3 $p \vee q$.

O próximo passo é escolher uma proposição n de forma empírica e procurar dentre as demais cláusulas se existe alguma cláusula correspondente a $\neg n$.

Escolhendo a cláusula $C1$ $\neg p$ achamos a sua cláusula correspondente $C3$ $p \vee q$, tendo então $\neg p \wedge p \vee q$ que resulta em q; assim definimos uma nova cláusula $C4$ q.

Agora escolhemos uma outra cláusula no caso $C2$ $\neg q$ e sua cláusula correspondente $C4$ q que gera uma contradição e consequentemente um resultado vazio representado por $\square$.

Uma outra fórmula de provar o resultado vazio para a negação desse teorema seria alternar as escolhas, começando a escolha da cláusula $C2$ $\neg q$ e a cláusula $C3$ $p \vee q$ resultando em $C4$ p que gera uma contradição entre $C1$ p e $C4$ p resultando em $\square$.

5.10.3 Estratégia Linear das Cláusulas Ordenadas (OL - Resolução)

DEFINIÇÃO: Uma dedução linear de uma cláusula C_n de um conjunto Σ de entrada com cláusula topo (ou base) C_1 em Σ é uma sequência de cláusulas $< C_1, C_2, C_3, \ldots, C_n >$ onde C_{i+1} com $(1 \leqslant i \leqslant n)$ é uma resoluente de C_i (cláusula centro) e β (cláusula literal) onde β e Σ ou $\beta = C_j$ com $(1 \leqslant j \leqslant i)$

Essa estratégia é completa.

Cláusula Ordenada: Uma cláusula é considerada ordenada como uma sequência de literais mais um mecanismo que determina a condição sobre a qual uma cláusula deve ser usada como cláusula do centro.

Basta armazenar o literal usado na dedução junto com o resoluente, isso reduz o espaço de busca e evita possíveis buscas redundantes e até mesmo loopings infinitos.

Seja, $C_1 \vee L_1$ e $C_2 \vee \neg L_1$ podemos resolver L_1 da primeira cláusula com $\neg L_1$ da segunda. Teoricamente poderíamos simplismente apagar L_1 da listas de preposições, entretanto ao invés disso apenas marcamos ela como *framed*.

A resolução das duas primeiras cláusulas ficam então: $C_1 \vee [L_1] \vee C_2$.

Um literal *framed* é um literal que já foi resolvido, ele é mantido na lista para evitar redundâncias e *loopings* infinitos.

Com o conceito de *framed*, podemos utilizar duas regras para eliminação de preposições *framed* e *não framed* que são:

RESOLUÇÃO: Se o último literal da cláusula centro for complementar de um literal p, ou seja $(p, [\neg p])$ *framed*, então pode-se remover esse literal da lista.

ELIMINAÇÃO: Quando se obtém uma cláusula com literal *framed* não seguido de nenhum literal "*não framed*" esse também pode ser removido da lista de literais.

EXEMPLO 1: Vamos assumir um conjunto $\Sigma = \{q \vee p, \neg p \vee r, \neg r \vee \neg p, \neg q\}$.

Em um primeiro momento elegemos uma das cláusulas para começar a resolução, computacionalmente fica mais fácil escolher a primeira ou a última, vamos escolher então $q \vee p$.

1 $q \vee p$

Depois disso devemos pegar a última preposição da cláusula, no caso p, e varrer as demais cláusulas buscando uma preposição negada, ou seja $\neg p$. A segunda cláusula da lista corresponde a busca $\neg p \vee r$, então resolvemos: $q \vee p \neg p \vee r$ sem a regra da preposição *framed* obteríamos $q \vee r$ entretanto como queremos manter a preposição *framed* anotamos $q \vee [p] \vee r$.

2 $q \vee [p] \vee r$

Devemos agora então procurar pelo último elemento negado da nova cláusula r, ele se encontra na terceira cláusula $\neg r \vee \neg p$, então resolvemos $q \vee [p] \vee r \neg r \vee \neg p$, obtendo:

3 $q \vee [p] \vee [r] \vee \neg p$

Nesse momento podemos aplicar a regra de redução obtendo:

4 $q \vee [p] \vee [r]$

Agora pode-se aplicar a regra de eliminação obtendo:

5 q

Ainda resta uma preposição, então devemos procurar novamente sua negação na lista de cláusulas. A última cláusula da lista $\neg q$ contempla nossa busca e é elegida.

6 $\neg q$

Nesse momento entramos em contradição, pois $q \neg q$. Chegamos ao fim da resolução com vazio $\square$.

7 $\square$

EXEMPLO 2: Um outro exemplo é o conjunto $\Sigma = \{p \vee q, \neg p, \neg q \vee r, \neg r \vee s, \neg r \vee \neg s\}$.

Seguindo os mesmos conceitos do exemplo anterior, temos:

1 $p \vee q$ - escolhemos $\neg q \vee r$

2 $p \vee [q] \vee r$ - escolhemos $\neg r \vee s$

3 $p \vee [q] \vee [r] \vee s$ - escolhemos $\neg r \vee \neg s$

4 $p \vee [q] \vee [r] \vee [s] \vee \neg r$ - aplicamos redução e eliminação

5 p

6 $p \neg p$

7 $\square$

É importante esclarecer que a Estratégia Linear das Cláusulas Ordenadas só funciona com o recurso de *backtrack*, ou seja, caso em algum momento não se consiga resolver as cláusulas, é necessário voltar ao nível de resolução anterior, enquanto não se acha outra saída. Caso volte todos os níveis com as diferentes combinações testadas e não se consiga resolver, pode-se afirmar que o teorema não tem resolução.

5.10.4 Árvore de Prova

Para implementar uma árvore de prova precisamos de alguns recursos extras, para otimizar a busca pelas cláusulas e um possível *backtrack*.

Para efetuar uma busca eficiente pelas cláusulas sugere-se a criação de duas matrizes que irão armazenar a ocorrência das preposições $P[n]$ e das preposições negadas $N[m]$. As linhas dessas matrizes são indexadas pelas preposições do conjunto a ser calculado e as colunas pelas cláusulas. Assim é possível mapear as cláusulas e marcar nas matrizes as ocorrências de preposição.

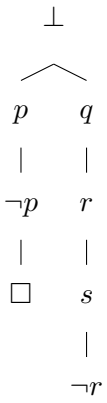
Essa técnica facilita na hora de buscar pelo inverso de uma preposição específica.

Ainda é necessário armazenar uma outra matriz com as preposições completas, para buscas e cálculos.

Outra matriz ainda é necessária para armazenar quais foram as preposições buscadas, caso haja a necessidade de fazer um *backtrack*.

A estrutura de um nodo da árvore deve ter ponteiros diretos para seu pai e seus irmãos. Isso é necessário pela busca escolhida (em largura ou profundidade) na árvore e também para eventuais chamadas de *backtrack*.

Se aplicássemos a estrutura de uma árvore no conjunto $\Sigma = \{p \vee q, \neg p, \neg q \vee r, \neg r \vee s, \neg r \vee \neg s\}$.



5.11 Sistema de Tableau

O Sistema de Tableau é um sistema refutacional, proposto inicialmente por Jaak Hintikka em 1955, com contribuições de Beth em 1959 e Smullyan em 1968.

Esse sistema usa conjunto de fórmulas no lugar de conjunção de fórmulas.

É uma árvore construída a partir de regras.

DEFINIÇÃO: São definição de um tableau:

1. Um ramo de um tableau é tido ser fechado caso ele contiver duas fórmulas A e $\neg A$, para quaisquer fórmulas A;

2. Um tableau é fechado quando todos os seus ramos são fechados, caso contrário o tableau é aberto;

3. Um tableau fechado prova que a fórmula em questão é um teorema.

Uma prova de A no sistema de tableau se inicia com a fórmula $\neg A$ na raiz, e segue aplicando as regras até obter um tableau fechado.

5.11.1 Regras

As regras no sistema tableau são classificadas em regras do tipo A e regras do tipo B.

TIPO A: Se houver um padrão de preposições do tipo A_n então deve-se criar dois nodos na mesma ramificação de forma subsequente $(N_1, [N_2])$.

A_1 $A \wedge B$ então $N_1 : A$ e $N_2 : B$

A_2 $\neg(A \vee B)$ então $N_1 : \neg A$ e $N_2 : \neg B$

A_3 $\neg(A \rightarrow B)$ então $N_1 : A$ e $N_2 : \neg B$

A_4 $\neg\neg A$ então $N_1 : A$

TIPO B: Se houver um padrão de preposições do tipo B_n então deve-se criar dois nodos ramificados (N_1, N_2).

B_1 $A \vee B$ então $N_1 : A$ e $N_2 : B$

B_2 $A \rightarrow B$ então $N_1 : \neg A$ e $N_2 : B$

B_3 $\neg(A \wedge B)$ então $N_1 : \neg A$ e $N_2 : \neg B$

5.11.2 Teoremas

O sistema tableaux garante os teoremas da adequação e completude:

adequação O sistema de tableau para a lógica proposicional é adequado pois garante que se o tableau fechar a fórmula em questão é um teorema;

completude O sistema tableau para a lógica proposicional é completo pois garante que se a fórmula em questão é um teorema então irá resultar em um tableau fechado.

5.11.3 Estratégia

De modo a evitar muitas ramificações na árvore que será escrita pelo sistema de tableau, é recomendável procurar aplicar as regras do tipo A antes das regras do tipo B

5.11.4 Busca pelo Resultado

Em aplicações práticas do sistema de tableaux, é recomendável que se eleja como raiz o resultado buscado, com objetivo de eliminar a resolução de preposições irrelevantes para o resultado buscado.

Por exemplo:

Imagine o conjunto $\Sigma = \{p \vee q, \neg p, \neg q \vee r, \neg r \vee s, \neg r \vee \neg s\}$;

Se busca por $r \vee s$.

Nesse caso devemos eleger $r \vee s$ como raiz do sistema tableaux, e só depois esgotar suas ramificações. Isso garante que preposições irrelevantes como $\{p, q\}$ sejam calculadas sem necessidade.

5.11.5 Exemplos

EXEMPLO 1 Provar que $p \rightarrow (q \rightarrow p)$ é um teorema.

O primeiro passo é negar a fórmula e colocar esse nodo em sua raiz, então a raiz é dada por: $\neg(p \rightarrow (q \rightarrow p))$

$$\begin{array}{c} \neg(p \rightarrow (q \rightarrow p)) \\ | \\ p \\ | \\ \neg(q \rightarrow p) \\ | \\ q \\ | \\ \neg p \end{array}$$

Os nodos p e $\neg(q \rightarrow p)$ foram obtidos através da regra A_3 das regras do tipo A que afirma que $\neg(A \rightarrow B)$ então $N_1 : A$ e $N_2 : \neg B$. No caso p representou A e $(q \rightarrow p)$ representou B.

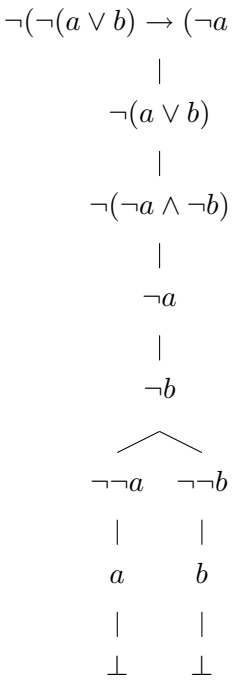
Com somente 1 preposição não se consegue aplicar nenhuma das regras, então o alvo passa a ser $\neg(q \rightarrow p)$.

$\neg(q \rightarrow p)$ também se encaixa na regra do tipo A_3 das regras do tipo A. No caso A foi representado por q e B por q.

Obtemos então $\neg p$ que entra em contradição com a preposição p no único ramo do tableau, então podemos considerar o tableau fechado, assim $p \rightarrow (q \rightarrow p)$ é um teorema.

EXEMPLO 2 Provar que $\neg(a \vee b) \rightarrow (\neg a \wedge \neg b)$ é um teorema.

O primeiro passo é negar a fórmula e colocar esse nodo em sua raiz, então a raiz é dada por: $\neg(\neg(a \vee b) \rightarrow (\neg a \wedge \neg b))$



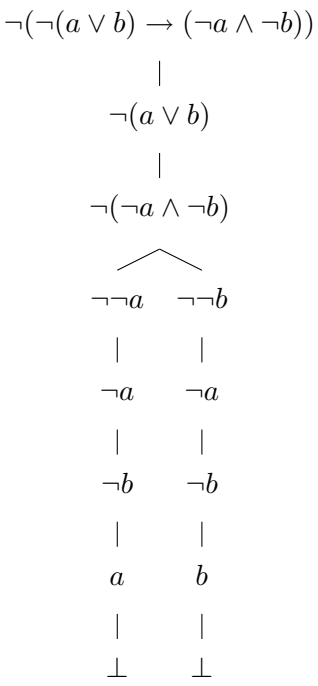
Para obter $\neg(a \vee b)$ e $\neg(\neg a \wedge \neg b)$ utilizou-se a regra A_3 das regras do tipo A que afirma que $\neg(A \rightarrow B)$ então $N_1 : A$ e $N_2 : \neg B$. No caso $\neg(a \vee b)$ representou A e $(\neg a \wedge \neg b)$ representou B.

Para obter $\neg a$ e $\neg b$ utilizou-se a regra A_3 em $\neg(a \vee b)$.

Nesse ponto nada mais pode ser calculado com $\neg a$ ou $\neg b$, então aplica-se uma regra do tipo B_3 em $\neg(\neg a \wedge \neg b)$. Assim se obtém $\neg\neg a$ e $\neg\neg b$ que entram em contradição com as preposições anteriores a e b fechando assim cada uma um ramo da árvore e também o tableaux, então podemos considerar o tableau fechado. $\neg(a \vee b) \rightarrow (\neg a \wedge \neg b)$ é um teorema.

Ainda para esse mesmo tableaux, poderíamos ter uma derivação dada pela ordem de escolha das preposições a serem substituídas.

Se ao invés de escolher $\neg(a \vee b)$ tivéssemos escolhido $\neg(\neg a \wedge \neg b)$, então a construção da árvore seria dada de maneira diferente.



Para se obter $\neg\neg a$ e $\neg\neg b$ aplicou-se a regra B_3 em $\neg(\neg a \wedge \neg b)$.

A partir desse momento escolhe-se uma das ramificações até fechar esse ramo.

Quando se obteve $\bot$ no ramo $\neg\neg a$, deve-se voltar até a fórmula $\neg(\neg a \wedge \neg b)$ e aplicar novamente a regra B_3 até fechar essa outra ramificação.

6 Lógica de Predicados de 1ª Ordem

Assim como a lógica proposicional, a lógica de predicados de primeira ordem também define regras para sua linguagem, semântica e axiomática.

6.1 Linguagem

ALFABETO:

- Pode ser definido como um conjunto enumerável de símbolos predicativos de $0, 1$ ou mais argumentos,

  Por exemplo: $(p, q, p_1, q_1, pai, mae, \ldots)$;

- Um conjunto enumerável de variáveis,

  Por exemplo: $(x, y, x_1, y_1, \ldots)$;

- Um conjunto enumerável de funções de $0, 1$ ou mais argumentos,

  Por exemplo: $(f, g, f_1, g_1, pai\_de, \ldots)$;

  Funções de zero argumentos são consideradas constantes,

  Por exemplo: $(A, B, C, \ldots)$;

- Quantificadores: $\forall, \exists$,

  Por exemplo: q pode ser $\forall$ ou $\exists$;

- Conectivos já definidos na lógica proposicional: $\bot, \neg, \vee, \wedge, \rightarrow$ e parênteses (e).

TERMOS:

- Toda variável é um termo;

- $F(t_1, t_2, \ldots, t_n)$ é um termo se F é uma função de n argumentos e $t_1, t_2, \ldots, t_n$ são termos.

FÓRMULA:

- Se p é um símbolo predicativo de n argumentos e se $t_1, t_2, \ldots, t_n$ são termos então $P(t_1, t_2, \ldots, t_n)$ é uma fórmula atômica.

- O conjunto FOR de fórmulas (FBF's) é definido da mesma maneira que na lógica proposicional, mais o seguinte:

  - QxA é ma fórmula;
  - Se Q é uma quantificação;
  - x é uma variável e A é uma fórmula.

- Exemplo 1: $P(x, y) \rightarrow Q(a)$.
- Exemplo 2: $\forall x(P(x) \rightarrow \neg\neg P(x))$.

6.2 Escopo de um Quantificador

Nas fórmulas:

- $\forall xA$
- $\exists xA$

A é dito estar no escopo do quantificador.

6.3 Variável Livre/Ligada

Uma ocorrência de uma variável é livre se ela não está no escopo de um quantificador, senão é ligada.

EXEMPLO 1: $\forall x((P(x) \vee \exists xP(x)) \wedge Q(y))$.

No exemplo, $Q(y)$ está ligado a $\forall x$; $P(x)$ está ligado a $\exists x$ e $P(x)$ está livre.

6.4 Fórmula Fechada/Aberta

Uma fórmula é aberta se contém alguma variável livre, se não é fechada.

O exemplo 1 pode ser considerado fechado.

EXEMPLO 2: $\forall x\forall y((P(x) \vee \exists xP(x)) \wedge Q(y))$.

6.5 Substituição de Variáveis

Uma substituição de variáveis é uma função do conjunto de variáveis no conjunto de termos.

A aplicação de uma substituição em uma expressão E é o resultado da troca simultânea de trocas de ocorrências de variáveis livres em L por seu termo associado.

Se E é uma expressão, então $(E)'s$ é uma instância de E.

NOTAÇÃO: Seja $\{X_1, X_2, \ldots, X_n\}$ um conjunto de variáveis tais que $S(x)$ é diferente de x. A substituição S então é denotada:

$$S = \{X_1 \setminus T_1, \ldots, X_n \setminus T_n\}$$

EX_1 $(P(x) \vee Q(x,y))\{x \setminus z\}$ substituindo temos: $(P(z) \vee Q(z,y))$;

EX_2 $(P(x) \vee Q(x,y))\{x \setminus y\}$ substituindo temos: $(P(y) \vee Q(y,y))$;

EX_3 $\forall x Q(x,y)\{x \setminus z\}$ substituindo temos: $\forall x Q(x,y)$;

EX_4 $(P(x) \vee Q(x,y))\{x \setminus F(x)\}$ substituindo tempos: $(P(F(x)) \vee Q(F(x),y))$;

EX_5 $(P(x) \vee Q(x,y))\{x \setminus y, y \setminus a\}$ substituindo temos: $(P(y) \vee Q(y,a))$;

EX_6 $(P(x) \vee Q(x,y))\{x \setminus y, y \setminus x\}$ substituindo temos: $(P(y) \vee Q(y,x))$;

No EX_3 não se pode substituir z por x pois a variável está ligada.

No EX_4 não se aplica recursividade em x quando já se substituiu por $F(x)$.

No EX_5 não se aplica a substituição de y nas variáveis y já substituídas.

No EX_6 também não se aplica recursividade nas variáveis a serem substituídas.

6.6 Teoria dos Modelos (Semântica)

6.6.1 Interpretação

- Um conjunto não vazio D chamado domínio;
- Uma função IV do conjunto de variáveis em D.
- Uma função IF que associa a cada função de n argumentos uma aplicação de D^n em D;
- Uma função IP que associa a cada predicado de n argumentos um subconjunto de D^n.

EXEMPLOS: Considere os exemplos:

Conjunto de times de futebol. $\sum = \{coxa, atletico, palmeiras, inter, \ldots\}$.

Conjunto de números inteiros $\sum = \{1, 2, 3, 4, 5, 6, 7, \ldots\}$.

Dados os conjuntos é possível associar uma variável a um elemento dos conjuntos, por exemplo é possível associar x ao time $inter$, y ao time $palmeiras$ e z ao time $coxa$.

Ou ainda x ao número 6 e y ao número 7.

Dessa forma poderíamos criar uma função $soma(x, y) \rightarrow z$ ou $par(x) \rightarrow \{2, 4, 6\}$.

É importante lembrar que o nome da função não condiz necessariamente com a escolha dos elementos do conjunto.

6.6.2 Interpretação de Termos e Fórmulas

Uma interpretação I pode ser extendida aos termos da seguinte forma:

$$I(F(T_1, T_2, \ldots, T_n) = (I(F), I(T_1), I(T_2), \ldots, I(T_n))$$

VARIANTE: Uma interpretação I' é uma variante em x de I.

PARA FÓRMULAS:

- $I(P(T_1, T_2, \ldots, T_n)) = \top$ se e somente se $I(P)$ contém $I(T_1), I(T_2), \ldots, I(T_n)$;
- $I(\forall xA) = \top$ se e somente se $I'(A) = \top$ para a variante I' de I em x;
- $I(\exists xA) = \top$ se e somente $I'(A) = \top$ para alguma variante I' de I em x;
- Para os conectivos é similar a lógica proposicional: $I(A \wedge B) = \top \therefore I(A) = \top e I(B) = \top$.

EXEMPLO 1: $D = \{alice, beto, carlos\}$

- $I(x) = alice$
- $I(y) = beto$
- $I(z) = carlos$
- $I(gosta) = \{(alice, carlos), (beto, alice), (alice, carlos)\}$

Sejam as interpretações:

- $gosta(z, x) = \bot$ - pois carlos não aparece na interpretação de $gosta$;
- $\exists x gosta(x, z) = \top$ - pois $(alice, carlos)$ está no conjunto de interpretação de $gosta$;
- $\exists x gosta(z, x) = \bot$ - pois não existe nenhum elemento que satisfaça a interpretação;
- $\forall x \neg gosta(x, x) = \top$ - pois a negação de uma mentira (no caso ninguém gosta de si mesmo) é uma verdade.

EXEMPLO 2: A dificuldade está na tentativa de se achar um modelo para as fórmulas escritas em lógica de primeira ordem, considere a seguinte fórmula:

$$(\forall x \forall y R(x,y)) \wedge (\forall x \neg R(x,x)) \wedge (\forall x \forall y \forall z((R(x,y) \wedge R(y,z)) \rightarrow R(x,z)))$$

Se imaginássemos o conjunto dos números reais $D = \mathbb{Z}$ com a inferência de que $R(x,y)$ possui $x < y$.

Teríamos para essa fórmula um resultado satisfatível, ou seja um modelo.

Entretanto a variação infinita de combinações para o conjunto D e as regras de inferência R podem derivar outras soluções.

6.6.3 Satisfabilidade

Uma fórmula é satisfatível se existe pelo menos uma interpretação válida para ela (modelo).

6.6.4 Validade

Uma fórmula é válida se qualquer interpretação é um modelo.

6.6.5 Consequência Lógica

Consequência lógica é definida também como na lógica proposicional:

$$A_1, A_2, A_3, \ldots, A_n \models A$$

6.7 Teoria da Prova (Axiomática)

A lógica de primeira ordem herda os axiomas da lógica proposicional adaptados.

- $\forall x A \rightarrow ((A)\{x \setminus t\}$;
- $A\{x \setminus t\} \rightarrow \exists x A$.

6.7.1 Regras de Inferência

Modus Ponens É dado por:

$$A \rightarrow B$$

$$B \therefore A$$

Regra para quantificadores:

$$\frac{A \rightarrow B}{A \rightarrow (\forall x B)}$$

$$\frac{A \rightarrow B}{(\exists x A) \rightarrow B}$$

NOTA: Noções de dedução, prova e teorema são como na lógica proposicional.

A lógica assim definida,

- é adequada;
- é completa.

$$\vdash A \text{ se e somente se } \models A$$

A lógica de primeira ordem não é decidível: $\nexists$ um procedimento de decisão que termina com um "não" se a fórmula não for válida.

Mas é semidecidível: se for válida, $\exists$ o procedimento de decisão.

$$\Gamma \vdash A \vdash \Gamma \rightarrow A$$

$$\Gamma \models A \models \Gamma \rightarrow A$$

6.8 Formas Normais

Em Lógica de primeira ordem necessitamos de outras formas normais para obter a FNC (Forma Normal Conjuntiva)

- Forma Normal Prenex;
- Forma Normal de Skolen;
- Forma Normal Conjuntiva;

Antes de ver as formas normais, precisamos definir algumas propriedades:

6.8.1 Equivalências

As equivalências da lógica de primeira ordem são herdadas das equivalências da lógica proposicional, mas seus quantificadores são tratados de forma diferente.

Seja A sem ocorrência livre de x, então:

$\vdash A$ se e somente se $\vdash \forall x A$.

A é satisfativel se e somente se $\exists x A$ é satisfatível;

$$\vdash Q_1 x Q_2 x A \leftrightarrow Q_2 x A$$

Seja A sem ocorrência livre de x, então:

$$\vdash Q x A \leftrightarrow A$$
$$\vdash (A)\{x \setminus t\} \leftrightarrow A$$

6.8.2 Troca de Nomes

É prova que pode-se efetuar troca de nomes de variáveis, mantendo equivalências

$$\vdash Q x A \leftrightarrow Q y (A)\{x \setminus y\}$$

6.8.3 Distribuição

Na distribuição, pode-se isolar o quantificador $\forall$ caso os elementos estejam ligados por um $\wedge$, e com o quantificador $\exists$ caso os elementos estejam ligados por um $\vee$

$$\vdash (\forall x A) \wedge (\forall x B) \leftrightarrow \forall x (A \wedge B)$$
$$\vdash (\exists x A) \vee (\forall x B) \leftrightarrow \exists x (A \vee B)$$

6.8.4 Alargamento do Escopo dos Quantificadores

Seja B sem ocorrência livre de x, então

$$\vdash (Q x A) \wedge B \leftrightarrow Q x (A \wedge B)$$
$$\vdash (Q x A) \vee B \leftrightarrow Q x (A \vee B)$$

EXEMPLO: $(\exists x P(x)) \vee \neg P(x) \nleftrightarrow \exists x(P(x) \vee P(x))$

6.9 Troca de Quantificadores

Podemos efetuar a troca de quantificadores quando:

$$\neg \forall x P(x) \leftrightarrow \exists x \rightarrow P(x)$$
$$\neg \exists x P(c) \leftrightarrow \forall x \neg P(x)$$

6.10 Forma Normal Prenex

DEFINIÇÃO: Uma fórmula A está em forma normal prenex (FNP) se ela é da forma:

$$Qx_1 Qx_2 \ldots Qx_n B$$

Onde B é uma fórmula sem quantificadore. B é chamado de matriz e A sequência de Qx_1, Qx_2 Qx_n é o prefixo.

EXEMPLO 1:

$$\exists x(P(x) \rightarrow \forall x P(x))$$
$$\exists x(\neg P(x) \vee \forall x P(x))$$
$$\exists x(\neg P(x) \vee \forall y P(y))$$
$$\exists x \forall y(\neg P(x) \vee P(y))$$

A troca da implicação por $\vee$ foi herdada das propriedades da lógica proposicional, onde $(\vdash A \rightarrow B \leftrightarrow \neg A \wedge B)$.

Na segunda linha aplicou-se a substituição $\{x \setminus y\}$ para o quantificador universal $\forall$

Por fim, aplicou-se a propriedade de distribuição.

Pode-se dizer que $\exists x \forall y(\neg P(x) \vee P(y))$ está em FNP.

EXEMPLO 2:

$$\neg(P(x) \rightarrow ((\exists y Q(x,y) \wedge \exists y R(y)))$$

$$\neg(\neg P(x) \vee ((\exists y Q(x,y) \wedge \exists y R(y))))$$

$$(\neg\neg P(x) \wedge \neg((\exists y Q(x,y) \wedge \exists y R(y))))$$

$$P(x) \wedge (\neg(\exists y Q(x,y) \wedge \exists y R(y))$$

$$P(x) \wedge (\neg\exists y Q(x,y) \wedge \exists y R(y))$$

$$P(x) \wedge (\forall y \neg Q(x,y) \vee \forall y R(y))$$

$$P(x) \wedge (\forall y \neg Q(x,y) \vee \forall z R(z))$$

$$\forall y \forall z (P(x) \wedge \neg Q(x,y) \vee R(y))$$

A troca da implicação por $\vee$ foi herdada das propriedades da lógica proposicional, onde $(\vdash A \rightarrow B \leftrightarrow \neg A \wedge B)$.

Ao se negar $(\neg\neg P(x)$ o conectivo $\vee$ é trocado para $\wedge$.

Ao se obter $\neg\exists y$ aplica-se a propriedade de substituição dos quantificadores.

Substitui-se $\{y \setminus z\}$.

Por fim, aplicou-se a propriedade de distribuição.

Pode-se dizer que $\forall y \forall z (P(x) \wedge \neg Q(x,y) \vee R(y))$ está em FNP.

TEOREMA: Para toda a entrada A o algorítmo de transformação em FNP termina e retorna uma fórmula em FNP equivalente à fórmula normal de entrada.

6.11 Forma Normal de Skolen

Uma fórmula está em FNS se ela está em FNP e não possui quantificadores existências.

EXEMPLO 1

$$\exists x \forall y (\neg P(x) \vee P(y))$$

EXEMPLO 2

$$\forall y \forall z (P(x) \wedge (\neg Q(x,y) \vee \neg R(z))))$$

O Exemplo 1 não está em FNS. O Exemplo 2 está em FNS.

6.11.1 Algoritmo

ENTRADA: Uma fórmula A;

SAÍDA: Uma fórmula em FNS;

INÍCIO

COLOCAR A em FNP;

PARA TODO QUANTIFICADOR RXISTENCIAL QUE APAREÇA EM A FAÇA

substituir por uma nova função (f) de ariedade = número de quantificadores universais que antecedem $\exists x$ na fórmula em FNP, no prefixo e aplicar a substituição $\{x \setminus f(x_1, x_2, \ldots, x_n)\}$

APAGAR $\exists x$ DA FÓRMULA

FIM PARA

FIM

Ariedade é a quantidade de elementos em uma função, por exemplo $F(a, b, c)$ possui ariedade igual a 3.

EXEMPLO 1:

$$\forall x \exists y P(x, y) \therefore \forall x P(x, F(x))$$

$$\exists y \forall x P(x, y) \therefore \forall x P(x, A)$$

Note que no primeiro caso a ariedade da primeira fórmula é igual a 1, pois existe apenas um quantificador universal antes do quantificador existencial. Substitui-se por uma fórmula do tipo $F(x)$ com apenas 1 variável.

No segundo caso, como não existe nenhum quantificador universal antes de um quantificador existencial, substitui-se por uma constante no caso A.

EXEMPLO 2:

$$\exists u \forall x \exists y \forall t \exists t (P(x) \wedge Q(y) \wedge P(x, z, t) \wedge S(y) \wedge K(v)) \therefore$$

$$\forall x \forall z (P(x) \wedge Q(f(x)) \wedge P(x, z, G(x, z)) \wedge S(F(x)) \wedge K(A)))$$

As seguintes substituições foram efetuadas:

- Ariedade 0 : $\{u \setminus A\}$;

- Ariedade 1 : $\{y \setminus F(x)\}$;
- Ariedade 2 : $\{t \setminus G(x, z)\}$;

TEOREMA: Para toda entrada A, o algorítmo acima termina e retorna uma fórmula A' em FNS tal que A é satisfatível se e somente se A' é satisfatível.

6.12 Forma Normal Conjuntiva

Uma fórmula está em Forma Normal Conjuntiva se está em Forma Normal Skolen e a matriz está em FNC, no sentido proposicional.

NOTAÇÃO: Como só sobram quantificadores universais, pode-se considerar a fórmula sem eles e assumir que toda variável está quantificada universalmente, ficando assim, só a matriz.

6.13 Resolução

6.13.1 Unificador de uma Substituição s

Unifica dois termos se ela os torna iguais. s unifica T e T' se $(T)s = (T')s$.

EXEMPLO: Para a equação : $\{x = F(y), y = z\}$, pode-se aplicar as seguintes substituições

$$s = \{x \setminus F(a), y \setminus a, z \setminus a\}$$
$$s' = \{x \setminus F(y), z \setminus y\}$$

No exemplo a última equação $s' = \{x \setminus F(y), z \setminus y\}$ é a mais geral, ou seja substitui as cláusulas com o menor esforço.

Se aplicássemos a primeira substituição nas cláusulas, teríamos: $\{F(a) = F(a), a = a\}$

Se aplicássemos a segunda substituição nas cláusulas, teríamos: $\{F(y) = F(y), y = y\}$

DEFINIÇÃO: Unificador mais geral Sejam T e T' dois termos, S um unificador de T e T'. S é chamado de UMG se para todo unificador S' de T e T' existe uma substituição S' tal que $S' = S'' \circ S$.

$\circ$ representa composto.

6.13.2 Algorítmo

ESCOLHER UMA EQUAÇÃO DE E (CONJUNTO DE EQUAÇÕES)

APLICAR:

SE FOR $T = T$, APAGAR;

SE FOR $F(T_1, \ldots, T_n) = G(T'_1, \ldots, T'_n)$, FALHA;

SE FOR $F(T_1, \ldots, T_n) = F(T_1, \ldots, T_n)$ SUBSTITUIIR POR N EQUAÇÕES $T_1 = T'_1, \ldots, T_n = T'_n$

SE FOR $X = T$ E X APARECE EM T, FALHA;

SE FOR $X = T$ E X NÃO APARECE EM T ENTÃO SUBSTITUIR X POR T EM TODAS AS FÓRMULAS DE E;

ATÉ QUE NÃO HAJA MAIS NADA PARA FAZER.

EXEMPLO 1:

$$\{x = y', F(y) = u\}$$
$$UMG = \{x \setminus y', u \setminus F(y)\}$$

EXEMPLO 2:

$$\{G(y) = G(z), x = F(y)\}$$
$$\{y \setminus z, x \setminus F(y)\}$$
$$\{y \setminus z, x \setminus F(z)\}$$
$$UMG = \{y \setminus z, x \setminus F(z)\}$$

EXEMPLO 3:

$$\{x = y, x = F(x)\}$$

Ao se substituir $x \setminus y$ acontece uma falha de ocorrência. Então não é possível achar um unificador.

EXEMPLO 4:

$$\{x = G(y), F(x) = z, y = a\}$$
$$\{x \setminus G(a), F(y) \setminus z, y \setminus a\}$$
$$\{x \setminus G(a), F(G(a)) \setminus z, y \setminus a\}$$
$$UMG = \{x \setminus G(a), z \setminus F(G(a)), y \setminus a\}$$

6.14 Método da Resolução para Lógica de Primeira Ordem

6.14.1 Resolvente

Sejam C e C' dias cláusulas tais que:

$$C = \{p(T_1, T_2, \ldots, T_n)\} \cup D$$
$$C' = \{\neg p(T'_1, T'_2, \ldots, T'_n)\} \cup D'$$

D e D' corresponde ao resto das cláusulas.

Seja S um Unificador Mais Geral (UMG) de $\{T_1 = T'_1, T_2 = T'_2, \ldots, T_n = T'_n\}$, então: $D \cup D'$ é resolvente de C e C'.

EXEMPLO 1 Sejam as cláusulas:

$$C = \{P(x, F(y)), Q(x, y, z)\}$$
$$C' = \{\neg P(y, u), \neg P_1(y), P_2(H(u, v))\}$$

Podemos notar um padrão possível de se aplicar a regra do resolvente em C por $P(x, F(y))$ e em C' por $\neg P(y, u)$.

Para que isso seja possível aplica o resolvente, devemos achar um Unificador Mais Geral que consiga igualas as fórmulas.

Antes de achar o UMG é necessário aplicar a propriedade de troca de nomes, trocando o nome das variáveis iguais nas cláusulas, aplicamos então a substituição $E = \{y = y'\}$ na cláusula C', ficando:

$$C' = \{\neg P(y', u), \neg P_1(y'), P_2(H(u, v))\}$$

Agora aplicamos a UMG $\{x \setminus y', u \setminus F(y)\}$ obtendo:

$$C = \{P(y', F(y)), Q(x, y, z)\}$$
$$C' = \{\neg P(y', F(y)), \neg P_1(y'), P_2(H(F(y), v))\}$$

Aplicando a regra resolvente, ficamos com:

$$\{Q(x, y, z), \neg P_1(y'), P_2(H(F(y), v))\}$$

EXEMPLO 2

$$C = \{P(x), P(x_1)\}$$
$$C' = \{\neg P(y), \neg P(y_1)\}$$

Não é necessário a troca de nomes pois não há variáveis com nomes iguais nas duas cláusulas.

A UMG é definida por $\{x \setminus y\}$.

Então a resolvente é obtida:

$$\{P(x_1), \neg P(y_1)\}$$

6.14.2 Fator

Seja C uma cláusula:

$$C = \{p(T_1, T_2, \ldots, T_n)\}, p(T'_1, T'_2, \ldots, T'_n) \cup D$$
$$C = \{\neg p(T_1, T_2, \ldots, T_n)\}, \neg p(T'_1, T'_2, \ldots, T'_n) \cup D$$

E S um UMG para $\{T_1 = T'_1, T_2 = T'_2, \ldots, T_n = T'_n\}$, então um fator de C é:

$$\{p(T_1, T_2, \ldots, T_n) \cup D\}S$$

OU

$$\{\neg p(T_1, T_2, \ldots, T_n) \cup D\}$$

EXEMPLO 1 Seja a cláusula:

$$C = \{\neg P(x, p), \neg P(F(y), y), Q(y, z)\}$$

Aplica-se a UMG: $\{x \setminus F(a), y \setminus a\}$

O fator é dado por: $\{\neg P(F(a), a), Q(a, z)\}$

EXEMPLO 2

$$C = \{\neg P(y), \neg P(F(y))\}$$

Ao tentar se fazer a substituição $\{y = F(y)\}$, a tentativa falha, então $\nexists$ fator da cláusula C.

NOTAÇÃO Pode-se escrever as fórmulas de duas formas, separadas por cláusulas ou unidas por conectivos.

Seja, $P(x, y) \wedge Q(a) \wedge (\neg P(b, z) \vee S(w, v)$ pode-se representar as conjunções separadas por linhas e as dijunções separadas por vírgula.

A mesma fórmula pode ser representada por:

$$\{P(x, y)\}$$
$$\{Q(a)\}$$
$$\{(\neg P(b, z), S(w, v)\}$$

O sistema formal de resolução é um sistema refutacional que tem zero axiomas e duas regras de inferência: regra do resolvente e regra do fator.

É possível provar que o sistema formal da resolução é correto e completo mas semidecidível. Isso significa que não há garantia que um sistema computacional vai um dia achar um resultado SAT para uma fórmula dada.

Então o método da resolução é um sistema formal que tem um conjunto de zero axiomas e duas regras de inferência, a regra resolvente e a regra do fator.

Uma refutação de um conjunto de cláusulas $\mathbb{C}$ é uma sequência $C_1, C_2, \ldots, C_n$, tal que:

- C_n é a cláusula vazia. Representada em conjuntos por $\{\}$ e em fórmua por $\square$.
- Cada C_i:
  - Ou pertence a C;
  - Ou é um resolvente de duas cláusulas C e C' no conjunto $C_1, C_2, \ldots, C_n$;
  - Ou então é um fator de algum $C_j \leqslant j < i$.

Para demonstrar que o sistema refutacional acha a satisfabilidade, é possível demonstrar que $\vdash \Gamma \wedge \neg F$ é deduzido de $\Gamma \vdash F$.

$$\Gamma \vdash F$$
$$\vdash \Gamma \rightarrow F$$
$$\vdash \neg(\Gamma \rightarrow F)$$
$$\vdash \neg(\neg\Gamma \vee F)$$
$$\vdash \Gamma \wedge \neg F$$

Basta obter a FNC de $\Gamma \wedge \neg F$ e pelo SFR tentar obter $\square$. Se conseguir então Γ é SAT.

EXEMPLO 1 Sejam as cláusulas:

$$C = \{P(x), P(x')\}$$
$$C' = \{\neg P(y), \neg P(y')\}$$

É possível obter pela notação conjuntiva:

(1)	$\{P(x), P(x')\}$	
(2)	$\{\neg P(y), \neg P(y')\}$	
(3)	$\{P(x'), \neg P(y')\}$	$(1,2)\{x \setminus y\}$
(4)	$\{P(x)\}$	$(1)\{x \setminus y'\}$
(5)	$\{\neg P(y')\}$	$(2)\{y \setminus y'\}$
(6)	$\{\}$	$(4,5)\{x \setminus y'\}$

Ou então pela notação de fórmulas:

(1)	$\{P(x) \vee P(x')\}$	
(2)	$\{\neg P(y) \vee \neg P(y')\}$	
(3)	$\{P(x') \vee \neg P(y')\}$	$(1,2)\{x \setminus y\}$
(4)	$\{P(x)\}$	$(1)\{x \setminus y'\}$
(5)	$\{\neg P(y')\}$	$(2)\{y \setminus y'\}$
(6)	$\square$	$(4,5)\{x \setminus y'\}$

Para obter a equação 3 aplicou-se a regra resolvente em $(1, 2)$, substituindo $\{x \setminus y\}$.

Para obter a equação 4 aplicou-se a regra do fator em 1, substituindo $\{x \setminus y'\}$.

Para obter a equação 5 aplicou-se a regra do fator em 2, substituindo $\{y \setminus y'\}$.

E finalmente, para se obter a equação 6 se aplicou a regra resolvente em $(4, 5)$.

É importante salientar que a ordem, com que as equações são geradas, influencia na tentativa de se fechar a prova.

Por exemplo se em um primeiro momento aplicássemos a regra resolvente entre $(2, 3)$, substituindo $\{y \setminus x'\}$ obteríamos de cara a cláusula vazia.

EXEMPLO 2 Seja uma cláusula: $\{P(x)\}, \{\neg P(y)\}, \{P(F(y))\}, \{\neg P(F(F(z)))\}$

(1)	$\{P(x)\}$	
(2)	$\{\neg P(y), P(F(y))\}$	
(3)	$\{\neg P(F(F(z)))), P(F(y))\}$	
(4)	$\{P(F(y'))\}$	$(1, 2)\{y \setminus y', x \setminus y\}$
(5)	$\{P(F(F(y'))\}$	$(3, 2)$
(6)	$\{\}$	

EXEMPLO 3 Nesse terceiro exemplo, vamos utilizar os conceitos descritos para exemplificar uma situação palpável, que exemplifica a utilização da lógica proposicional no mundo real.

Como o exemplo será constituído do início, todas as etapas serão descritas até a dedução que prova a o teorema a ser testado.

Vamos utilizar primeiramente a definição das sentenças em lingua portuguesa, e depois mapeá-las para a linguagem de primeira ordem.

Imagine as seguintes situações:

1. Alguns pacientes gostam de todos os médicos;
2. Nenhum paciente gosta de enfermeiro que aplica injeção;
3. Nenhum medico é enfermeiro que aplica injeção.

Poderíamos mapear essas sentenças da seguinte forma:

1. $\exists x(paciente(x) \wedge \forall y(medico(y) \rightarrow gosta(x, y)))$;

2. $\forall x(paciente(x) \rightarrow \forall y(injecao(y) \rightarrow \neg gosta(x,y)))$;

3. $\forall x(medico(x) \rightarrow \neg injecao(x))$.

Supondo $\Gamma = \{1,2\}$ e $\alpha = \{3\}$, provar que $\Gamma \vdash \alpha$.

Ou seja provar que as sentenças 1 e 2 provam a sentença 3.

Para isso, basta encontrar $\square$ pelo sistema formal da refutação a partir de $\Gamma \wedge \neg\alpha$.

Para isso devemos fazer uma série de transformações nas sentenças até chegar ao conjunto de cláusulas em FNC.

Abreviações Vamos assumir as seguintes abreviações:

- p - paciente;
- m - médico;
- g - gosta;
- i - injeção;

Aplicando FNC na Sentença 1

$$
\begin{array}{lll}
(1) & \exists x(P(x) \wedge \forall y(M(y) \rightarrow G(x,y))) & (elimina \rightarrow) \\
(2) & \exists x(P(x) \wedge \forall y(\neg M(y) \vee G(x,y))) & (FNP) \\
(3) & \exists x \forall y(P(x) \wedge (\neg M(y) \vee G(x,y))) & (FNS) \\
(4) & \forall y(P(A) \wedge (\neg M(y) \vee G(A,y))) & \\
(5) & P(A) \wedge (\neg M(y) \vee G(A,y)) & \\
(6) & \{P(A)\}, \{\neg M(y), G(A,y)\} &
\end{array}
$$

Aplicando FNC na Sentença 2

$$
\begin{array}{ll}
(1) & \forall x(P(x) \rightarrow \forall y(I(y) \rightarrow \neg G(x,y))) \\
(2) & \forall x(\neg P(x) \vee (\forall y(I(y) \rightarrow \neg G(x,y)))) \\
(3) & \forall x \forall y(\neg P(x) \vee \neg I(y) \vee \neg G(x,y))) \\
(4) & \neg P(x) \vee \neg I(y) \vee \neg G(x,y) \\
(5) & \{\neg P(x), \neg I(y), \neg G(x,y)\}
\end{array}
$$

Aplicando FNC na Sentença 3 Essa é a sentença que deve ser provada, por isso aplicamos as transformações em sua forma negada.

$$
\begin{array}{ll}
(1) & \neg(\forall x(\neg M(x) \vee \neg I(x))) \\
(2) & \exists x(\neg(\neg M(x) \vee \neg I(x)))) \\
(3) & \exists x(M(x) \vee I(x)) \\
(4) & (M(B) \vee I(B)) \\
(5) & \{M(B)\}, \{I(B)\}
\end{array}
$$

Gerando a Prova Com as cláusulas geradas, basta unir as cláusulas e procurar por $\square$.

$$
\begin{array}{ll}
(1) & \{P(A)\} \\
(2) & \{\neg M(y), G(A, y)\} \\
(3) & \{\neg P(x), \neg I(y), \neg G(x, y)\} \\
(4) & \{M(B)\} \\
(5) & \{I(B)\}
\end{array}
$$

Essas foram as cláusulas já encontradas, agora basta aplicar as regras de inferência de resolução e fatoração.

Podemos obter então:

$$
\begin{array}{lll}
(6) & \{\neg I(y'), \neg G(a, y)\} & (1, 3)\{x \setminus a\} \\
(7) & \{\neg I(y'), \neg M(y')\} & (2, 6)\{y \setminus y'\} \\
(8) & \{\neg M(B)\} & (5, 7)\{y' \setminus b\} \\
(9) & \{\} & (4, 8)
\end{array}
$$

6.15 Automatização do Processo

Uma das técnicas propostas para a automatização do processo é a resolução por saturação.

6.15.1 Resolução por Saturação

A resolução por saturação segue os seguintes passos:

- Seja um conjunto de cláusulas $\mathbb{C}$.
- Gerar a sequência $C_0, C_1, C_2, \ldots, C_n$ (cada C_i é uma cláusula);
- Até encontrar $\square$.

Assim, $\mathbb{C}_0$ é C, e C_n é resolvente de C_1 e C_2 — $C_1 \subset U_{c=0}^{n-1}\ C_i, C_2 e C_n - 1$.

EXEMPLO: Sejam as cláusulas $\mathbb{C}_0$:

$$
\begin{array}{ll}
(1) & p \vee q \\
(2) & p \vee \neg q \\
(3) & \neg p \vee q \\
(4) & \neg p \vee \neg q
\end{array}
$$

Obtém-se um conjunto $\mathbb{C}_1$ pelas combinações de cada uma das preposições com uma correspondente sua negada no conjunto inicial $\mathbb{C}_0$:

$$
\begin{array}{lll}
(5) & q \vee q & (1,3)p \\
(6) & q \vee \neg q & (1,4)p \\
(7) & p \vee q & (1,2)q \\
(8) & p \vee \neg p & (1,4)q \\
(9) & \neg q \vee q & (2,3)p \\
(10) & \neg q \vee \neg q & (2,4)p \\
(11) & p \vee \neg p & (2,3)p \\
(12) & \neg p \vee \neg q & (3,4)p
\end{array}
$$

Para gerar as equações $5, 6$ assumiu-se a variável atual p e buscou-se pelo espaço de cláusulas $\mathbb{C}_0$ suas negações que foram encontradas em $3, 4$.

Para gerar as equações $7, 8$ assumiu-se a variável atual q e buscou-se pelo espaço de cláusulas $\mathbb{C}_0$ suas negações que foram encontradas em $2, 4$.

Ou seja, para todas as preposições não negadas, varre-se toda a base de cláusulas procurando sua negação e gerando uma nova cláusula.

No conjunto $\mathbb{C}_1$ não foi possível se obter $\square$, ele está no conjunto $\mathbb{C}_2$ que possui 27 cláusulas.

Logo, esse algoritmo puro é ineficiente, pois:

- Apresenta redundância, pois a mesma cláusula é obtida de várias maneiras;
- Não tem tautologia;

6.16 Aumentando a Eficiência

Para aumentar a eficiência nos algoritmos de resolução podemos aplicar 3 técnicas, são elas:

1. Eliminar cláusulas puras:
   - São cláusulas que contém apenas um literal, e em todo o conjunto de cláusulas não aparece a sua forma negada;
   - Exemplo: $\{p, z \vee r, \neg z \neg r\}$; nesse a cláusula p é uma cláusula pura.
2. Eliminar tautologias:
   - Cláusulas que contenham um literal p e seu complementar $\neg p$;
   - Exemplo: $p \vee \neg p$;
3. Eliminar cláusulas subjugadas:
   - C_1 subjuga C_2 se e somente se existe uma substituição S tal que $S(C_1) \equiv C_2$ com C_2 como a cláusula subjugada;
   - Na presença de C_1, C_2 é inútil;
   - Exemplo 1: $\{p, p \vee q\}$; como já temos a cláusula p, $p \vee q$ se torna inútil;
   - Exemplo 2: $\{P(x), P(A) \vee R(y)\}$ como já temos a cláusula $P(x)$, $P(A) \vee R(y)$ se torna inútil;

7 Espaço de Pesquisa

É o conjunto de todos os resolventes que podem ser gerados a partir das cláusulas de entrada.

O procedimento de prova "navega" por esse espaço em busca de $\square$.

8 Conjunto Suporte

Seja $\mathbb{C}$ o conjunto de cláusulas de entrada.

Um conjunto $B \leqslant \mathbb{C}$ é denominado conjunto suporte de $\mathbb{C}$, se e somente se, $\mathbb{C}-B$ é satisfatível.

Uma dedução por conjunto suporte (*set of suport*) é uma dedução na qual para toda a aplicação da regra de resolução, as premissas não estão, ambas em $\mathbb{C} - B$.

Prova-se que um conjunto é satisfatível se e somente se existe uma dedução por Conjunto Suporte.

Como escolher o Conjunto Suporte B ?

Sugestões:

- Ou todas as cláusulas de $\mathbb{C}$ que tem literais com ligação;
- Ou todas as cláusulas de $\mathbb{C}$ que não tem literais com ligação;
- Para determinar $\Gamma \models \alpha$:
  - Seja $\mathbb{C}(\Gamma)$ o conjunto de cláusulas correspondentes a Γ e,
  - Seja $\mathbb{C}(\alpha)$ o conjunto de cláusulas correspondentes a α;

  Caso $\mathbb{C}(\Gamma)$ seja consistente, escolher $\mathbb{C}(\neg\alpha)$ como conjunto suporte.

8.1 Vantagem

Todo resolvente tem a ver com a consulta (sentença a se provada).

EXEMPLO: Imaginando um modelo matemático populado por pontos:

- Se y é acessível a partir de x e y é adjacente a z então z é acessível a partir de x;
- Todo ponto é acessível a partir dele mesmo;
- a é adjacente a b;
- b é adjacente a e;
- c é adjacente a d;
- a é adjacente a c;

Provar que d é adjacente a partir de a.

CONVERTENDO PARA CLÁUSULAS: Assumindo:

- ad como adjacente;
- ac como acessível;

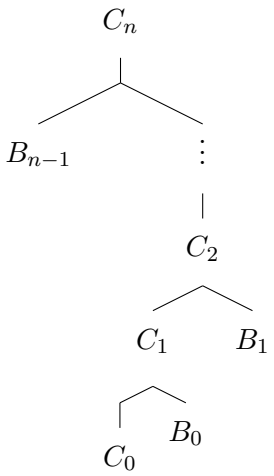
(1)	$\neg ac(x,y) \vee \neg ad(y,z) \vee ac)x,z)$	
(2)	$ac(x,x)$	
(3)	$ad(a,b)$	
(4)	$ad(b,c)$	
(5)	$ad(c,d)$	
(6)	$ad(a,c)$	
(7)	$\neg ac(a,d)$	$consulta(suporte)$
(8)	$\neg ac(a,y) \vee ad(y,b)$	$(7,8)$
(9)	$\neg ac(a,c)$	$(8,5)$
(10)	$\neg ac(a,c) \vee \neg ad(y,d)$	$(9,1)$
(11)	$\neg ac(a,d)$	$(10,6)$
(12)	$\square$	$(11,2)$

Nesse exemplo, resolver de outra maneira, que não com conjunto suporte corre o risco de entrar em recursividade, pois $\neg ac(x,y)$ está na mesma cláusula que $ac(x,z)$.

Note que a partir da geração da cláusula suporte, sempre deve-se utilizar uma cláusula que esteja no espaço do conjunto suporte, para procurar efetivamente $\square$.

9 Resolução Linear

Seja um conjunto de cláusulas $\mathbb{C}$, uma dedução linear de uma cláusula $\mathbb{C}_n$ a partir de C, tendo a cláusula C_0 como cláusula de partida, é uma sequência $C_1, C_2, \ldots, C_n$ tal que $C_\alpha \in \mathbb{C}$ para todo $1 \leqslant i \leqslant n$, C_i for obtido, resolvendo-se $C_i = 1$ com alguma cláusula C_i, com $1 \leqslant j \leqslant i-1$.



Uma refutação linear é mda dedução linear de $\square$.

EXEMPLO 1:

(1)	$p \vee q$	
(2)	$\neg p \vee q$	
(3)	$p \vee \neg q$	
(4)	$\neg p \vee \neq q$	
(5)	q	$(1, 2)$
(9)	p	$(5, 3)$
(10)	$\neg q$	$(6, 4)$
(12)	$\square$	$(7, 5)$

- Dedução linear preserva a completude;
- É compatível com conjunto suporte;
- Uso de subjulgação é importante, mas é caro.

EXEMPLO 2:

(1)	$\neg irmao(x, y) \vee irmao(y, x)$
(2)	$irmao(caim, abel)$

Provar:

(3)	$irmao(juca, caim)$

(4) $\neg irmao(caim, juca)$

(5) $\neg irmao(juca, caim)$

A cláusula 3 subjulga a cláusula 5, então não existe dedução.

9.1 Dedução Linea de Entrada

É uma dedução linear na qual só pode resolver C_i como cláusula de $\mathbb{C}$.

9.2 Resolução de Entrada

É a obtenção de $\square$ por dedução de entrada;

- perde-se a completude;
- implementação eficiente;
- PROLOG:
  - Não faz checagem de ocorrência na unificação (em muitos casos);
  - Só aceita cláusula de Horn.

9.3 Cláusula de Horn

É uma cláusula na qual somente um literal tem o símbolo da negação:

$$\neg p \rightarrow q \wedge r \wedge s$$

9.4 Resolução Unitária

Uma das premissas deve ser uma cláusula unitária, foi provado que é equivalente à resolução de entrada.

9.5 Maquina de Inferência x Árvore de Prova

Uma máquina de inferência deduz fórmulas para sistemas baseados em conhecimento, enquanto uma árvore de prova utiliza provador de teorema.

9.6 Dicas de Implementação para Árvore de Prova em Lógica de Predicados

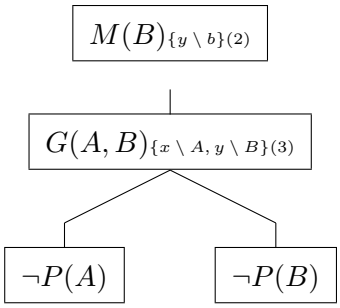
Deve-se criar uma estrutura para cada uma das cláusulas a serem resolvidas, onde a cláusula vinda da refutação é o nodo raiz. Esse nodo dará origem a seu nodo filho proveniente do algoritmo de unificação, ou das técnicas de fatoração e refutação.

É importante armazenar para cada estrutura qual foi o conjunto de substituição UMG utilizado, bem como os ponteiros para seu pai, e seu irmão da direita, caso haja.

Sendo,

$$
\begin{array}{ll}
(1) & \{P(A)\} \\
(2) & \{\neg M(y), G(A,y)\} \\
(3) & \{\neg P(x), \neg I(y), \neg G(x,y)\} \\
(4) & \{M(B)\} \\
(5) & \{I(B)\}
\end{array}
$$

Uma representação de sua implementação em árvore seria dada por:



- Lista por onde o algoritmo caminhou;
- Busca em Profundidade ou em Largura:
  - Em busca em Profundidade pode-se demorar muito para obter um resultado, que uma busca em largura se obteria mais facilmente;
- Teste de profundidade máxima, se passar de n assume-se que não se obteve um resultado;
- Tabela de símbolos proposicionais, devem armazenar todos os símbolos e onde há sua ocorrência positiva e negativa;
- Algoritmo de Unificação;

- Lista de Folhas;
- Regras de Fatoração e Redução;
- Algoritmo de Resolução;
- Backtrack Inteligente;
- Conversão para FNC.

10 Raciocínio com Ações

A lógica clássica convencional ainda não consegue um mecanismo que possa mapear de forma lógica um cenário, um acontecimento e as respectivas mudanças que esse acontecimento implica nesse cenário.

Teoricamente ao se simular uma situação corriqueira como acender uma luz em uma sala, em uma representação lógica, perderia-se toda a representação do restante que não a luz e o interruptor, impossibilitando a geração de seu modelo computacional.

10.1 Cálculo de Situações

Esse cálculo represente a descrição de um cenário ou mundo.

Adota-se uma notação parecida com a lógica proposicional, entretanto atribui-se a captura de ações.

Adota-se também pre-condições e pós-condições para essas ações.

EXEMPLO

$$(\neg dead(x, s_0) \wedge loaded(s) \rightarrow (dead(x, do(shoot, s_0))))$$

Nesse problema temos uma pré-condição: $\neg dead(x, s_0) \wedge loaded(s)$, um efeito *dead* e uma ação *shoot*.

Esse tipo de representação gera 3 principais problemas:

1. Problema da Persistência (*frame problem*) - Esse problema é acarretado por não poder deduzir qual o estado das coisas depois que o tiro foi disparado;
    - Utiliza o princípio da lei da inércia, para descrever que o que já está parado tende a ficar parado;

- Esse problema pode ser melhorado com um algoritmo de minimização do estado das coisas;
- Cai em lógica de segunda ordem em uma propriedade chamada circuscrição.

2. Problema da Ramificação - Define quais ações são consequência das que já passaram, por exemplo é possível afirmar que: $walking(c) \rightarrow \neg dead(s)$;
3. Problema da Qualificação - São as pré-condições para que uma ação seja efetuada, o problema é que são infinitas pré-condições.

São as divisões:

- Planejamento: Assume que se conheçe o cenário antigo e o novo, mas não se conhece os passos que dão origem ao novo cenário;
- Prediction: Conheçe-se o cenário antido e os passos que darão origem, mas não se tem conhecimento de todos os elementos do novo cenário;
- Post Diction: Conhece-se os passos que dão origem e o novo cenário, mas não se tem idéia do cenário que deu origem.

10.2 Strips

STRIPS (Stanford Research Institute Problem Solver)

É um planejador automático desenvolvido por Richard Fikes e Nils Nilsson in 1971.

Essa linguagem é a base para muitas linguagens que tratam o problema de planejamento automatizado.

É uma linguagem de descrição de domínios com notação baseada na lógica.

É uma maneira de se transformar estados em outros (cenários) baseando-se em:

- Uma busca em um espaço de estados;
- Regras de transformação.

ESTADO: É um conjunto de símbolos (que tem como idéia expressar preposições).

É comum se usar notação de lógica de primeira ordem para expressar esses estados.

UM PROBLEMA DE PLANEJAMENTO: É uma quadra $PP < S, G, A, C >$, onde:

S É a situação atual;

G É o objetivo a ser alcançado;

A É o conjunto de descrições das ações;

C É o conjunto de condições.

O elemento A é uma tripla $A < PRE, ID, EFF >$, onde:

PRE É um conjunto de símbolos (preposições) que determinam quando a ação é aplicável;

ID É o nome da ação;

EFF É uma dupla $EFF < DEL, ADD >$ onde:

- Cada DEL e ADD é um conjunto de símbolos (preposições);
- Quando uma ação é aplicada, em uma entrada S o efeito é um estado S no qual se acrescentam as preposições de ADD e se removem as preposições de DEL.

10.3 Um exemplo de problema STRIPS

Uma macaco em um laboratório. Esse macaco quer somente bananas. No laboratório existem três locais: A, B e C.

O macaco está inicialmente na localização A.

Existe uma caixa na localização C.

Existem algumas bananas na localização B, mas elas estão penduradas no teto.

O macaco precisa subir na baixa para pegar as bananas.

ESTADO INICIAL:

- *At(A)*;
- *Level(low)*;
- *BoxAt(C)*;
- *BananasAt(B)*.

ESTADO FINAL (OBJETIVO):

- *Have(Bananas).*

AÇÕES: Assumi-se *pre* como pré-condições e eff como efeitos, tendo *add* como a lista de preposições que será adicionadas e *del* como a lista de preposições que será removida.

- Mover de x para y (*Move(x, y)*):

  pre At(X), Level(low);

  eff *add At(y);*

  del At(x).

- Subir na caixa (*ClimbUp(x)*):

  pre At(x), BoxAt(x), Level(low);

  eff *add Level(high);*

  del Level(low).

- Mover a caixa de x para y, e o macaco se move também (*MoveBox(x, y)*):

  pre At(X), BoxAt(X), Level(low);

  eff *add BoxAt(Y), At(Y);*

  del BoxAt(X), At(X).

- Pegar as Bananas (*TakeBananas(x)*):

  pre At(x), BananasAt(x), Level(high);

  eff *add Have(bananas);*

Com o problema descrito é possível nele uma navegação no formato de árvore assumindo o estado inicial como raiz e o objetivo como nodo folha que queremos alcançar.

Muitas ramificação são criadas para representar todas as possibilidades, o melhor caso para esse problema seria:

$$Move(a, c) \rightarrow MoveBox(c, b) \rightarrow ClimbUp(b) \rightarrow TakeBananas(b)$$

10.4 Busca Heurítica

A busca apresentada na sessão anterior, obviamente gera uma explosão populacional impossibilitando uma busca viável.

A primeira técnica adotada para tentar melhorar a busca no espaço gerado foi a de se eleger o nodo a ser explodido por sua semelhança com o objetivo, para isso, deve-se refinar a definição do objetivo, mapeando várias ocorrências que implicam na ação-resultado.

No exemplo dos macacos o objetivo poderia ser mapeado para:

- *have(bananas);*
- *bananasAt(b);*
- *at(b);*
- *boxAt(b);*

A princípio somente *have(bananas)* é tido como objetivo, entretanto os outros ítens também devem ser verdadeiros para que o primeiro seja.

A partir do espaço de busca refinado, procura-se pelo caminho que apresenta os resultados que mais coincidem com o resultado.

Entretanto, essa heurística foi rapidamente abandonada quando se notou o problema de *máximo local*, esse problema encontra na função de respostas um resultado que a princípio é o maior possível,

mas não é o resultado esperado, nesse ponto o algoritmo dica travado, pois, ao avançar ou retroceder, sempre encontra um resultado pior que o já encontrado, mas o resultado-objetivo está uma posição diferente no gráfico polinomial.

10.5 Grafo de Planos

Em 1995, um algoritmo foi gerado, de forma a encontrar resultados satisfatórios. O algoritmo de Grafo de Planos, visa mapear a busca heurística de forma mais esperta, dividindo um espaço de buscas

por camadas separadas de *Lpar* para preposições e *Limpar* para efeitos. A idéia do algoritmo é mapear o que já foi mostrado anteriormente de forma a não se separar por conjuntos de estados, mas sim,

identificar quais foram as preposições e efeitos que deram origem a ele.

A cada nível n avançado todas as preposições do nível n_{i-1} são copiadas.

As novas preposições são agregadas ao nível n a partir dos efeitos que podem ser aplicados aos nível n_{i-1}.

Os níveis são gerados até que se chegue a preposições objetivo.

Depois disso, deve-se retroceder no grafo, buscando de forma recursiva e apoiada pela tecnologia de *backtrack* afim de se se estabelecer uma conexão entre o nodo objetivo e o nodo inicial, traçando assim o plano de ações.

10.5.1 Relação de Exclusão Mútua (Mutex)

Para melhorar a busca, em uma situação que se chega a uma em *backtrack* define-se algumas situações de Mutex.

MUTEX: Duas preposições ou ações são definidas como Mutex, quando não podem acontecer ao mesmo tempo. Isso as torna inconsistentes.

Em um backtrack quando se identifica uma cláusula ou efeito do tipo Mutex, assume-se que aquele caminho não irá trazer um bom resultado e efetua-se a operação de *backtrack*.

Uma operação pode ser definida como Mutex quando:

NEGAÇÃO: Existe uma preposição p e sua negação $\neg p$ em um mesmo nível que dão origem a efeitos α e β.

EFEITOS INCONSISTENTES: O efeito de uma ação é a negação de outro efeito da ação.

INTERFERÊNCIA: Uma ação deleta a pre-condição de outra ação.

COMPETIÇÃO POR NECESSIDADE: As ações têm pre-condições que são mutuamente exclusivas no nível n_{i-1}.

10.6 Heurística Aliada com Grafo de Planos

O ato de se chegar até o nível n que acha um conjunto de preposições-objetivo, não é tão caro quanto se retroceder no espaço de busca, afim de achar o plano de ações.

Pensando nisso, dois alemães propuseram a utilização da busca heurística alterando o método de qual seria o nodo a ser explodido na busca, para cada nodo de estado gerado, faz-se uma busca pelo grafo de planos daquele estado e verifica-se

quantos níveis são necessários para atingir o objetivo, dessa forma o nodo que obtiver o caminho mais curto é elegido para ser explodido.

11 SatPlan

Proposto por Kautz & Selmann em 1992

Planejamento como dedução no cálculo de situações.

EXEMPLO: Em um mundo de blocos, se tem n blocos dispostos empilhados sobre uma mesa. Em dada situação com apenas 2 blocos, o bloco A está sobre o bloco B após a ação $move(A, B)$ ter sido executado a ação S_0:

$$on(A, B, result(move(a, b), s_0))$$

Esse tipo de situação gera alguns problemas já conhecidos, e necessita de um sistema proposicional finito. Entretanto pode ser mapeado para lógica proposicional, simplificando as operações lógicas.

11.1 Linguagem

- livre de funções e quantificadores;
- tipada com igualdade.

Uma notação que não se utiliza em lógica e que foi proposta no SatPlan foi a utilização de igualdade, com isso é possível definir os critérios de algumas preposições, como: $x \neq x'$.

Cada um dos conjuntos finitos tipados contém um conjunto finito de indivíduos cujos nomes são únicos termos constantes.

EXEMPLO: Dois tipos (para o mundo de blocos)

- blocos $(A, B, C, \ldots)$;
- tempo faixa de tempo $\mathbb{N}$.

Um conjunto finito de fórmulas é abreviado por um esquema (que pode conter representações da lógica de primeira ordem $\forall$ e $\exists$

- a partir de um esquema, basta iterar aos tipos e gerar as fórmulas;

- expressões aritméticas do tipo $i + 1$ são interpretadas como instâncias do tempo;
- existe um número que é a maior constante de tempo;
- $on(x, y, i)$ representa que x está sobre y em um tempo i;
- $clear(x, i)$ bloco x está livre
- $move(x, y, z, i)$ x foi movido de y para z no tempo i;

Nessa representação, uma ação é uma preposição.

EXEMPLO:

$$\forall x, y, z, ion(x, y, i) \wedge clear(x, i) \wedge clear(z, i) \wedge move(x, y, z, i) \rightarrow on(x, z, i + 1) \wedge clear(y, i + 1)$$

11.2 Problemas

Entretanto essa representação gera alguns problemas:

11.2.1 Frame Axioms

É um problema que ocorre pela consequência de ações, ou seja, se perde o que acontece com os outros blocos no momento que se move um dos blocos no cenário.

Deve-se então representar todas as possibilidades nos tempos i:

$$on(A, B, 1) \wedge on(B, table, 1) \wedge clear(A, 1) \wedge$$
$$on(B, A, 2) \wedge on(B, A, 3) \wedge clear(table, 1) \wedge$$
$$clear(table, 2) \wedge clear(table, 3)$$

11.2.2 Modelos Anômanos

Na lógica proposicional $\perp$ sempre implica verdadeiro, então procura-se eliminar elementos $\perp$ antes de uma implicação.

12 Planejamento como Satisfabilidade

Não é um teorema a ser provado.

É a busca por qualquer modelo que corresponda a um plano válido.

Deve-se utilizar preposições para criar:

- descrição da situação inicial e final;
- descrição das ações;
- descrição dos "frame axioms":

$$on(C, D, 1) \wedge move(A, B, 1) \rightarrow on(C, D, 1)$$

Nota-se que mesmo sabendo que a situação de C e D são independentes de A e B é necessário deixar explícito através de uma preposição como a do exemplo anterior.

- eliminação dos modelos anômalos (cada problema possui uma solução diferente):
  1. uma ação implica suas pré-condições e efeitos (eliminar pre-condição falsa):

$$\forall x, y, z, imove(x, y, z) \rightarrow (clear(x, i) \wedge clear(z, i) \wedge on(x, y, i))$$

  2. apenas uma ação ocorre em um dado tempo:

$$\forall x, x', y, y', z, z', i(x \neq x' \vee y \neq y' \vee z \neq z') \rightarrow \neg move(x, y, z, y) \vee \neg move(x', y', z', i)$$

  3. alguma ação ocorre em um dado tempo:

$$\forall i < N \exists x, y, zmove(x, y, z, i)$$

Se o estado iniciar for completamente especificado, estes axiomas garantem que qualquer modelo corresponde a um prazo válido.

No problema exemplo, o único modelo contém $move(a, b, mesa, 1)$ e $move(b, mesa, a, 2)$.

12.1 Vantagens (Algumas)

1. pode-se criar restrições para o planejamento:

$$\neg clear(A, 5) \vee \neg clear(B, 6)$$

Apenas inserindo essas inferências no banco de inferências, garantimos que no passo 5 A será um bloco sem ninguém sobre ele e no passo 6 o bloco B também.

2. fatos dependentes do domínio: Exemplo: mover e voltar os blocos

$$\neg(move(x, y, table, 0) \wedge move(x, table, y, 1))$$

O passo de mover para a mesa e voltar poderia se inserido no banco de inferências e não seria reproduzido, evitando passos desnecessários.

12.2 Análise

c é o número de elementos (constantes) do maior tipo. $MAX(n, i)$, onde n é o número de constantes e i é o tempo;

d profundidade máxima dos quantificadores dos esquemas:

$$\forall x, x', y, y', z, z', i(x \neq x' \vee y \neq y' \vee z \neq z') \rightarrow \neg move(x, y, z, y) \vee \neg move(x', y', z', i)$$

Nesse caso o quantificador $\forall$ tem uma profundidade igual a 7.

k número de literais do maior esquema;

Então o tamanho total da teoria é limitado por $\Theta(k * c^d)$.

Para diminuir Θ devemos diminuir d.

Para isso podemos substituir uma fórmula com muitos quantificadores, seguindo o seguinte esquema:

$$\forall i, x_1, x_2, x_1 \neq x_2 \rightarrow \neg object(x_1, i) \vee \neg object(x_2, i)$$
$$\forall i, y_1, y_2, y_1 \neq i_2 \rightarrow \neg source(y_1, i) \vee \neg source(y_2, i)$$
$$\forall i, z_1, z_2, z_1 \neq z_2 \rightarrow \neg dest(x_1, i) \vee \neg dest(x_2, i)$$

12.3 Resultados obtidos pelo SatPlan

	move			object, source, dest		
Problema	Prop.	Cláusulas	Tam.	Prop.	Cláusulas	Tam.
Reversal	429	22.418	51.753	215	993	2.533
Hanoi	1005	63.049	137.106	488	1.554	3.798
Huge	>7000	>3.500.000	>8.000.000	996	5.945	15.51

Problema	Vars	Size	SSAT	DPLL
RandomC	500	6.450	1.6h	-
ColoningB	2.250	18.0576	6h	-
RandomA	100	1.290	6s	28m
RandomB	140	1.806	14s	4.7h
Hanoi	288	3.798	-	13h

Problema	Original		Expandido	
-	Size	Time	Size	Time
Anomalia	933	26s	1.325	1.9s
Reversal	2.533	-	3.889	1.2m