INE5430
Inteligência Artificial

Tópico:

- Mecanismos de Raciocínio em Lógica, Prova Automática de Teoremas e PROLOG

Raciocínio Inferencial

- As principais características do motor de inferência disponível em shells para sistemas especialistas dizem respeito às seguintes funcionalidades:
  - Método de Raciocínio,
  - Estratégia de Busca,
  - Resolução de Conflito e
  - Representação de Incerteza e Imprecisão.
- Estas características compõem o Mecanismo de Raciocínio do Sistema Especialista.

Raciocínio Inferencial

- Definição
  - É aquele que se baseia em regras válidas de inferência. Pode ser aplicado em sistemas que adotam a representação do conhecimento sob a forma de regras de produção (os sistemas de produção) ou sob a forma de lógica.
- Modo de Raciocínio
  - Existem basicamente dois modos de raciocínio:
    - Raciocínio para a Frente ou Forward Chaining, e
    - Raciocínio para Trás ou Backward Chaining.

Raciocínio Inferencial

- Raciocínio para a Frente
  - Consiste em começar com fatos encontrados em uma base de conhecimentos e manipulá-los com as regras (de inferência) tentando chegar a uma conclusão.
  - É também chamado de raciocínio dirigido por dados ("data driven").
  - A parte esquerda da regra (os antecedentes ou estado) é comparada com a descrição da situação atual contida na memória de trabalho. As regras que satisfazem a esta descrição tem a sua parte direita (ação ou novo estado) executada, o que, em geral, significa a introdução d novos fatos na memória de trabalho.

Raciocínio Inferencial

- Raciocínio para a Frente
  - Exemplo:

REGRAS

- 1. $A \Rightarrow C$
- 2. $B \Rightarrow D$
- 3. $C \wedge D \Rightarrow E$

MEMÓRIA DE TRABALHO

- A e B

- C por 1
- D por 2
- E por 3

Raciocínio Inferencial

- Raciocínio para Trás
  - Começa usando a conclusão e tenta provar se são verdadeiras ou falsas as premissas.
  - É também chamado de raciocínio dirigido por objetivos ("*goal driven*").
  - O comportamento do sistema é controlado por uma lista de objetivos. Um objetivo por ser satisfeito diretamente por um elemento da memória de trabalho, ou podem existir regras que permitam inferir algum dos objetivos correntes, isto é, que contenham uma descrição deste objetivo em suas partes direitas.
  - As regras que satisfazem esta condição têm as instâncias correspondentes às suas partes esquerdas adicionadas à lista de objetivos correntes.

Raciocínio Inferencial

- Raciocínio para Trás
  - Caso uma destas regras tenha todas as suas condições satisfeitas diretamente pela memória de trabalho, o objetivo em sua parte direita é também adicionado à memória de trabalho.
  - Um objetivo que não possa ser satisfeito diretamente pela memória de trabalho, nem inferido através de uma regra, é abandonado.
  - Quando o objetivo inicial é satisfeito, ou não há mais objetivos, o processamento termina.

Raciocínio Inferencial

- Raciocínio para Trás
  - O encadeamento para trás destaca-se em problemas nos quais há um grande número de conclusões que podem ser atingidas, mas o número de meios pelos quais elas podem ser alcançadas não é grande (um sistema de regras de alto grau de fan out), e em problemas nos quais não se pode reunir um número aceitável de fatos antes de iniciar-se a busca por respostas.
  - O encadeamento para trás também é mais intuitivo para o desenvolvedor, pois é fundamentada na recursão, um meio elegante e racional de programação, para onde a própria Programação em Lógica se direcionou.

Raciocínio Inferencial

- Raciocínio para Trás
  - Exemplo:

REGRAS

- 1. A $\Rightarrow$ C
- 2. B $\Rightarrow$ D
- 3. C ^ D $\Rightarrow$ E

MEMÓRIA DE TRABALHO

- A e B

LISTA DE OBJETIVOS

- E

- C e D por 3
- C por 1 e A na M.T.
- D por 2 e B na M.T.

Raciocínio Inferencial

- O tipo de encadeamento normalmente é definido de acordo com o tipo de problema a ser resolvido.
- Problemas de planejamento, projeto e classificação tipicamente utilizam encadeamento para a frente,
- Problemas de diagnóstico, onde existem apenas apenas algumas conclusões possíveis mas um grande número de estados iniciais, utilizam encadeamento para trás.

Raciocínio em Sistemas Lógicos
Introdução

- Importância como teoria matemática.
- Adequada como método de representação de conhecimento.
- É O SISTEMA FORMAL MAIS SIMPLES DE QUE APRESENTA UMA TEORIA SEMÂNTICA INTERESSANTE DO PONTO DE VISTA DA REPRESENTAÇÃO DO CONHECIMENTO.
- Ainda hoje grande parte da pesquisa em IA está ligada direta ou indiretamente à Lógica.

Introdução

- De maneira geral um sistema lógico consiste em um conjunto de fórmulas e um conjunto de regras de inferência.
- As fórmulas são sentenças pertencentes a uma linguagem formal cuja sintaxe é dada.
- Uma regra de inferência é uma regra sintática que quando aplicada repetidamente a uma ou mais fórmulas verdadeiras gera apenas novas fórmulas verdadeiras.
- A seqüência de fórmulas geradas através da aplicação de regras de inferência sobre um conjunto de inicial de fórmulas é chamada de prova.
- A parte de lógica que estuda as provas é chamada teoria de provas.

Raciocínio em Sistemas Lógicos
Introdução

- Gödel e Herbrand na década de 30 mostraram que que qualquer fórmula verdadeira pode ser provada.
- Church e Turing em 1936 mostraram que não existe um método geral capaz de decidir, em um número finito de passos, se uma fórmula é verdadeira.
- Um dos primeiros objetivos da IA foi a Prova Automática de Teoremas, a partir da segunda metade da década de 60, sendo que a partir daí a lógica passou a ser estudada com método computacional para a solução de problemas.
- O método explora o fato de expressões lógicas poderem ser colocadas em formas canônicas (apenas com operadores "e", "ou" e "não"). O resultado permite a manipulação computacional bastante eficiente.

Raciocínio em Sistemas Lógicos
Introdução

- Teoria da Resolução de Robinson - 1965. Transforma a expressão a ser provada para a forma normal conjuntiva ou forma clausal. Existe uma regra de inferência única, chamada regra da resolução.Utiliza um algoritmo de casamento de padrões chamado algoritmo de unificação.
- Base para a Linguagem Prolog.

Lógica Proposicional

- Semântica do Cálculo Proposicional
  - Uma fbf pode ter uma interpretação a qual define a semântica da linguagem. Uma interpretação pode ser considerada como um mapeamento do conjunto das fbfs para um conjunto de valores de verdade {V, F} ou {Verdadeiro, Falso}.
  - Símbolos Proposicionais
    - podem ter qualquer significado
  - Símbolos constantes
    - V - verdadeiro: como o mundo é
    - F - falso: como o mundo não é
  - Sentenças complexas: o significado é derivado das partes
  - Conectivos: podem ser pensados como funções nas quais entram dois valores verdade e sai um

Lógica Proposicional

- Semântica do Cálculo Proposicional
  - TABELA VERDADE

P	Q	$\neg$ P	P $\wedge$ Q	P V Q	P $\rightarrow$ Q	P $\leftrightarrow$ Q
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	V	F	V	V	F
F	F	V	F	F	V	V

Lógica Proposicional

- Regras de Inferência

Regra	Tautologia	Nome
$\begin{array}{l} p \\ \hline \therefore p \vee q \end{array}$	$p \rightarrow (p \vee q)$	Adição
$\begin{array}{l} p \wedge q \\ \hline \therefore p \end{array}$	$(p \wedge q) \rightarrow p$	Simplifi cação
$\begin{array}{l} p \\ q \\ \hline \therefore p \wedge q \end{array}$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunção
$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$	$[p \wedge (p \rightarrow q)] \rightarrow q$	Modus Ponens
$\begin{array}{l} \neg q \\ p \rightarrow q \\ \hline \therefore \neg p \end{array}$	$[\neg q \wedge (p \rightarrow q)] \rightarrow \neg p$	Modus Tollens
$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$	$[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$	Silogismo hipotético
$\begin{array}{l} p \vee q \\ \neg p \\ \hline \therefore q \end{array}$	$[(p \vee q) \wedge \neg p] \rightarrow q$	Silogismo disjuntivo
$\begin{array}{l} p \vee q \\ \neg p \vee r \\ \hline \therefore q \vee r \end{array}$	$[(p \vee q) \wedge (\neg p \vee r)] \rightarrow (q \vee r)$	Resolução

Lógica Proposicional

- Regras de Inferência – Exemplo
  - Mostre que as hipóteses "Não está fazendo sol esta tarde e está mais frio do que ontem", "Nós iremos nadar somente se fizer sol", "Se nós não formos nadar, então nós vamos velejar", e "Se nós formos velejar, então estaremos em casa no final da tarde." levam à conclusão: "Nós estaremos em casa no final da tarde."
- Sejam as proposições:
  - p: "Está fazendo sol esta tarde."
  - q: "Está mais frio do que ontem."
  - r: "Nós iremos nadar."
  - s: "Nós iremos velejar."
  - t: "Estaremos em casa no final da tarde."

Lógica Proposicional

- Regras de Inferência – Exemplo
  - Então as hipóteses são: ¬p ^ q, r → p, :r → s, e s → t.
  - E a conclusão é simplesmente: t.

Passo	Justificativa
1. $\neg p \wedge q$	Hipótese
2. $\neg p$	1, Simplifi cação
3. $r \rightarrow p$	Hipótese
4. $\neg r$	2, 3, Modus Tollens
5. $\neg r \rightarrow s$	Hipótese
6. s	4, 5, Modus Ponens
7. $s \rightarrow t$	Hipótese
8. t	6, 7, Modus Ponens

Lógica de Primeira Ordem

- Introdução
  - A Lógica das Proposições tem um poder de representação limitado.
  - Na Lógica Proposicional se utiliza apenas sentenças completas, isto é, as proposições para representar o conhecimento sobre o Mundo.
  - A Lógica de Primeira Ordem ou Lógica dos Predicados, ou Cálculo dos Predicados, é uma extensão da Lógica das Proposições em que se consideram variáveis e quantificadores sobre as variáveis.
  - A Lógica dos Predicados se preocupa em introduzir noções lógicas para expressar qualquer conjunto de fatos através de Classes de Atributos e de Quantificadores.

Lógica de Primeira Ordem

- Introdução
  - Lógica de Primeira Ordem considera o mundo com:
    - Objetos (casas, cores, etc.)
    - Relações (maior que, dentro, tem cor, etc.)
    - Propriedades (vermelho, redondo, etc.)
    - Funções (pai de, melhor amigo, etc.)

Lógica de Primeira Ordem

- Quantificadores:
  - São operadores lógicos que em vez de indicarem relações entre sentenças, expressam relações entre conjuntos designados pelas classes de atributos lógicos.
  - Quantificador Universal ($\forall$):
    - Este tipo de quantificador é formado pelas expressões "todo" e "nenhum".
  - Quantificador Existencial ($\exists$):
    - Este tipo de quantificador é formado pelas expressões "existe um", "existe algum", "pelo menos um" ou "para algum".

Lógica de Primeira Ordem

- Quantificadores:
  - Exemplos:
    - Todo homem é mortal, ou seja, qualquer que seja x (do Universo), se x é Homem, então x é Mortal.
      - $\forall x\ (H(x) \rightarrow M(x))$.
    - Nenhum homem é vegetal, ou sejam qualquer que seja x, se x é Homem, em x NÃO É Vegetal.
      - $\forall x\ (H(x) \rightarrow \sim V(x))$.
    - Pelo menos um homem é inteligente, ou seja, existe pelo menos um x em que x seja Homem e x seja Inteligente.
      - $\exists x(H(x)\ \hat{}\ I(x))$

Lógica de Primeira Ordem

- Variáveis:
  - Designam objetos "desconhecidos" do Universo. "Alguém". São normalmente representados por letras minúsculas de "u" a "z".
- Letras Nominais:
  - Designam objetos "conhecidos" do Universo. "João", "Pedro", etc. São normalmente representados por letras minúsculas de "a" a "t".
- Predicados:
  - Descrevem alguma coisa ou característica de um ou mais objetos. São normalmente denotados por letras maiúsculas.
  - João ama Maria: $A(a,b)$
  - João ama alguém: $\exists x A(a,x)$
  - João ama todo mundo: $\forall x\ A(a,x)$

Lógica de Primeira Ordem

- Regras de Inferência para o Cálculo de Predicados
  - Todas as regras definidas no Cálculo Proposicional continuam válidas no Cálculo de Predicados, apenas referenciando-as para os quantificadores.
    - Ex.: $\sim F(a) \vee \exists x F(x), \exists x F(x) \rightarrow P$]- $F(a) \rightarrow P$.

Prova:

1.	$\sim F(a) \vee \exists x F(x)$	Premissa
2.	$\exists x F(x) \rightarrow P$	Premissa
3.	$F(a)$	Hipótese
4.	$\sim\sim F(a)$	3 DN
5.	$\exists x F(x)$	1,3 SD
6.	P	2,5 MP
7.	$F(a) \rightarrow P$	3,6 PC

Lógica de Primeira Ordem

- Regras de Inferência para o Cálculo de Predicados
  - Intercâmbio de Quantificadores

1. $\sim(\forall x \sim F(x)) \quad = \quad \exists x F(x)$
2. $\sim(\forall x\ F(x)) \quad = \quad \exists x \sim F(x)$
3. $\forall x \sim F(x) \quad = \quad \sim(\exists x F(x))$
4. $\forall x\ F(x) \quad = \quad \sim(\exists x \sim F(x))$

$\forall x \neg GostarPagar(x,Impostos) \equiv \neg \exists x\ GostarPagar(x,Impostos)$

- Como ∀ é na verdade uma conjunção sobre o universo de objetos e o ∃ é uma disjunção, não é surpreendente que eles obedeçam as Lei de De Morgan.

Lógica de Primeira Ordem

- Igualdade ou Identidade
  - É um símbolo que se adiciona ao Cálculo de Predicados com o propósito de expressar o fato de dois termos se referirem ao mesmo objeto, ou seja, "é idêntico a" ou "é a mesma coisa que".

  Exemplos:

  - O Pai de João é Henrique.
    Pai_de(João)= Henrique

  Pai de João e Henrique se referem ao mesmo objeto.

  - O Pai de João é também Avô de Pedro.
    Pai_de(João) = Avô_de(Pedro)

Lógica de Primeira Ordem

- Regras de Inferência para o Cálculo de Predicados

Regra de Inferência	Nome	Nota
$\dfrac{\forall x P(x)}{\therefore P(c)}$	*Instanciação* Universal	c específi co
$\dfrac{P(c) \text{ para um } c \text{ arbitrário}}{\therefore \forall x P(x)}$	Generalização Universal	c arbitrário
$\dfrac{\exists x P(x)}{\therefore P(c) \text{ para algum elemento } c}$	*Instanciação* Existencial	c específi co (mas não conhecido)
$\dfrac{P(c) \text{ para algum elemento } c}{\therefore \exists x P(x)}$	Generalização Existencial	c específi co e conhecido

Lógica de Primeira Ordem

- Exemplo: West é criminoso?
  1. $\forall x,y,z$ Americano(x)^Arma(y)^Nação(z)^Hostil(z)^ Vender(x,y,z) $\rightarrow$ Criminoso(x)
  2. $\exists x$ Possui(Iraque,x) ^ Míssil(x)
  3. $\forall x$ Possui(Iraque,x) ^ Míssil(x) $\rightarrow$ Vender(West,Iraque,x)
  4. $\forall x$ Míssil(x) $\rightarrow$ Arma(x)
  5. $\forall x$ Inimigo(América,x) $\rightarrow$ Hostil(x)
  6. Americano(West)
  7. Nação(Iraque)
  8. Inimigo(Iraque,América)
  9. Nação(América)

Lógica de Primeira Ordem

- Prova

10. Possui(Iraque,M1) ^ Míssil(M1) - 2 EE
11. Possui(Iraque,M1) - 10 ^E
12. Míssil(M1) - 10 ^E
13. Míssil(M1) →Arma(M1) - 4 EU
14. Arma(M1) - 12,13 MP
15. Possui(Iraque,M1) ^ Míssil(M1) →Vender(West,Iraque,M1)-3EU
16. Vender(West,Iraque,M1) - 11,12 MP
17. Americano(West)^Arma(M1)^Nação(Iraque)^Hostil(Iraque)^ Vender(West,Iraque,M1) →Criminoso(West) – 1 EU
18. Inimigo(América,Iraque) →Hostil(Iraque) - 5 EU
19. Hostil(Iraque) - 8,18 MP
20. Americano(West)^Arma(M1)^Nação(Iraque)^Hostil(Iraque)^ Vender(West,Iraque,M1) - 6,14,7,19,16 ^I
21. Criminoso(West) - 20,17 MP

Lógica de Primeira Ordem

- Se formularmos o processo de achar uma prova como um processo de busca, então esta prova é a solução de um problema de busca:
  - Estado Inicial = Base de Conhecimento (sentenças 1 – 9)
  - Operadores = regras de inferência aplicáveis
  - Estado Final = Base de Conhecimento contendo a sentença Criminoso(West)
- Isto é muito difícil pois a solução está na profundidade 12 e o fator de ramificação é bastante grande. Porque?

Árvore de Refutação

- São uma outra maneira de garantir a decidibilidade da Lógica Proposicional.
- REGRAS PARA ÁRVORE DE REFUTAÇÃO

1. Inicia-se colocando-se as PREMISSAS e a NEGAÇÃO DA CONCLUSÃO.

   (A idéia é encontrar contradições de modo a poder concluir a validade da conclusão.)

2. Aplica-se repetidamente uma das regras a seguir:

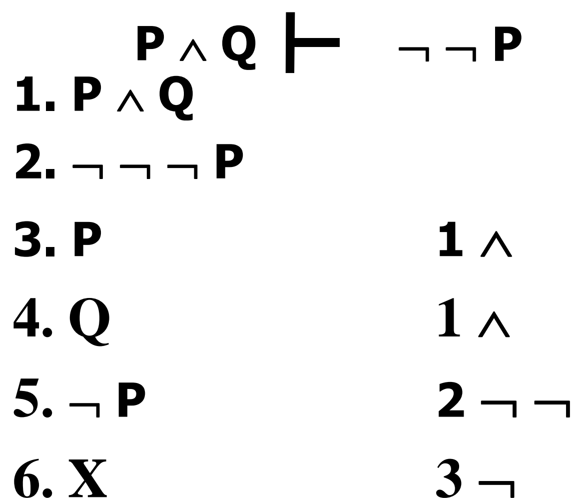
   2.1. Negação ($\neg$): Se um ramo aberto contém uma fórmula e sua negação, coloca-se um "X" no final do ramo, de modo a representar um ramo fechado.

   (um ramo termina se ele se fecha ou se as fórmulas que ele contém são apenas fórmulas-atômicas ou suas negações, tal que mais nehuma regra se aplica às suas fórmulas. Desta forma tem-se um ramo fechado, que é indicado por um X, enquanto o ramo aberto não é representado por um X.)

Árvore de Refutação

2.2. Negação Negada (¬ ¬): Se um ramo aberto contém uma fórmula não ticada da forma ¬ ¬ Ø, tica-se ¬ ¬ Ø e escreve-se Ø no final de cada ramo aberto que contém ¬ ¬ Ø ticada.

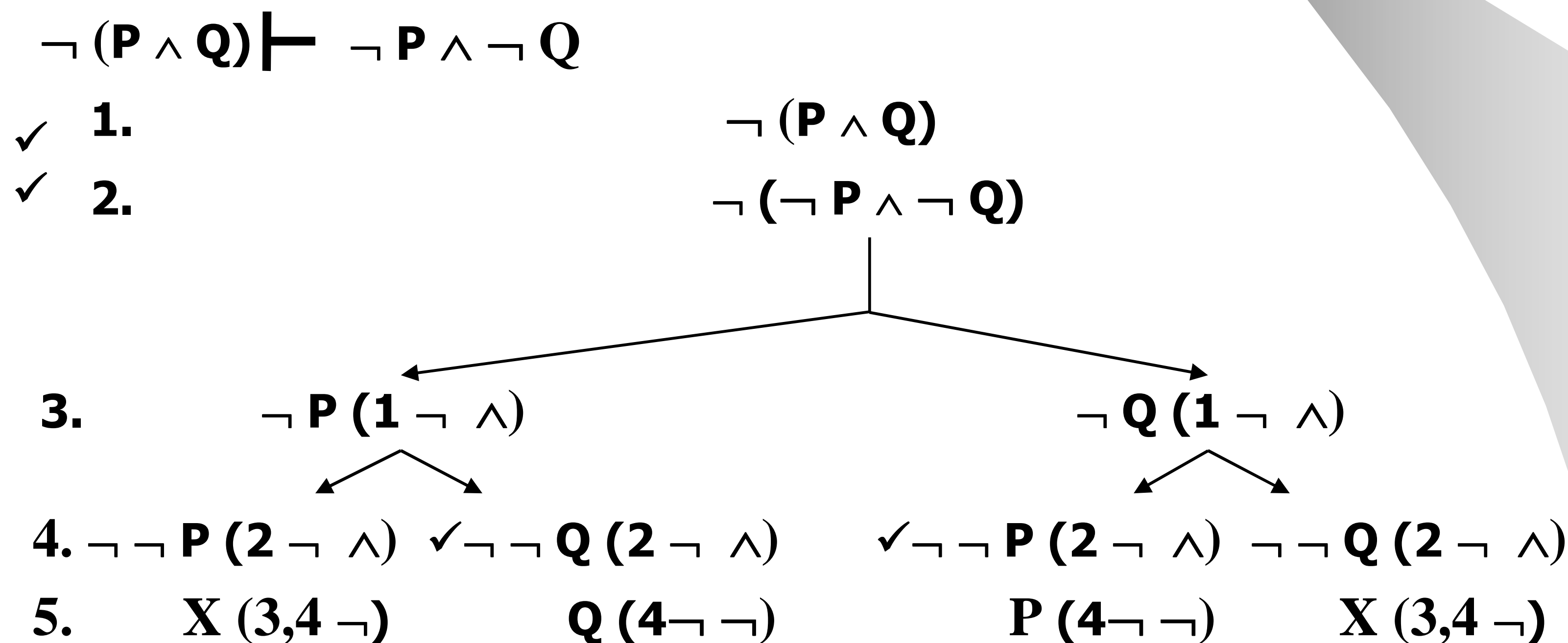
2.3. Conjunção (∧): Se um ramo aberto contém uma fórmula não ticada da forma Ø ∧ ß, tica-se, Ø∧ß e escreve-se Ø e ß no final de cada ramo aberto que contém Ø ∧ ß ticada.



A árvore de refutação está COMPLETA, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou CONTRADIÇÕES, e portanto, a FORMA É VÁLIDA.

Árvore de Refutação

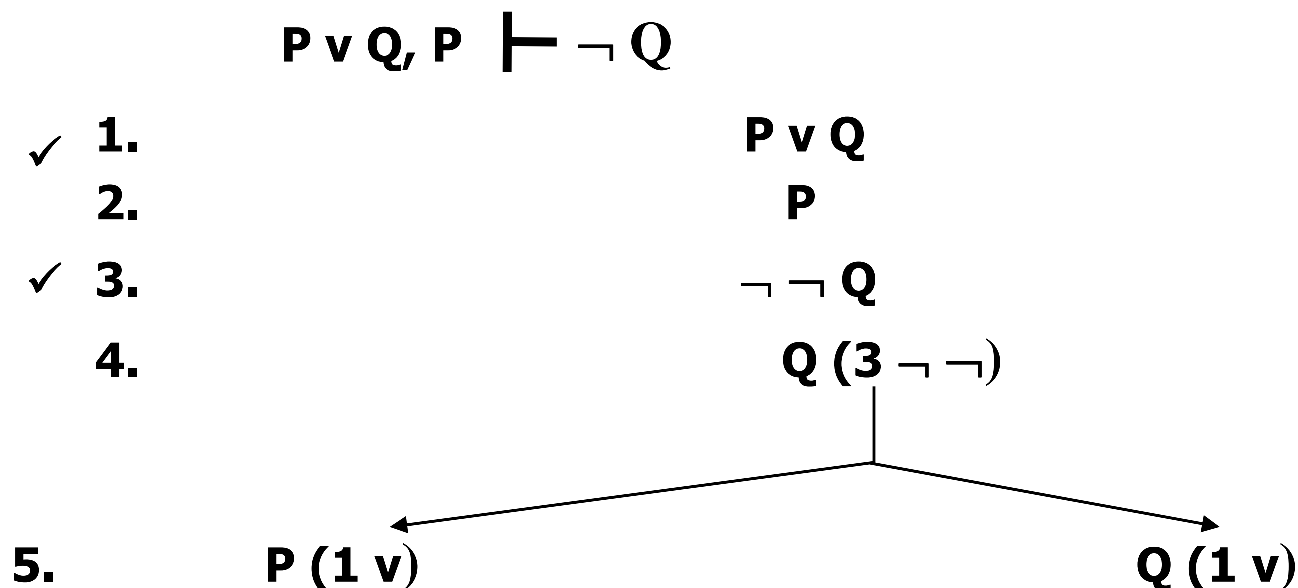
2.4. Conjunção Negada (¬ ∧): Se um ramo aberto contém uma fórmula não ticada da forma ¬ (Ø∧ß), tica-se, ¬ (Ø∧ß) e BIFURCA-SE o final de cada ramo aberto que contém ¬ (Ø ∧ ß) ticada, no final do primeiro ramo se esreve ¬ Ø e no final do segundo ramo se escreve ¬ ß.



O exemplo acima nos mostra que há dois ramos abertos, conseqüentemente a fórmula é inválida, o que significa que estes ramos são contra-exemplos.

Árvore de Refutação

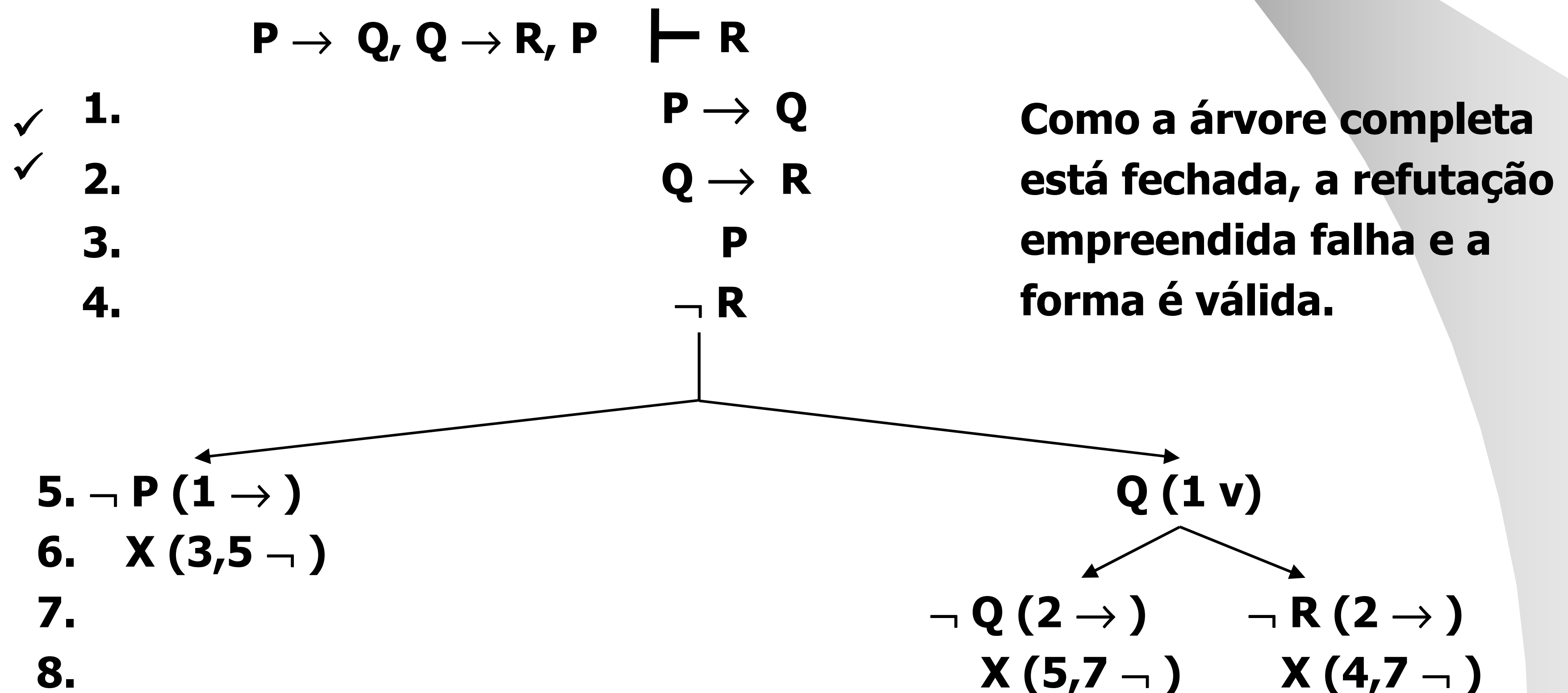
2.5. Disjunção (v): Se um ramo aberto contém uma fórmula não ticada da forma Øvß, tica-se, Øvß e BIFURCA-SE o final de cada ramo aberto que contém Ø v ß ticada, no final do primeiro ramo se esreve Ø e no final do segundo ramo se escreve ß.



O exemplo acima nos mostra que há dois ramos abertos, conseqüentemente a fórmula é inválida, o que significa que estes ramos são contra-exemplos.

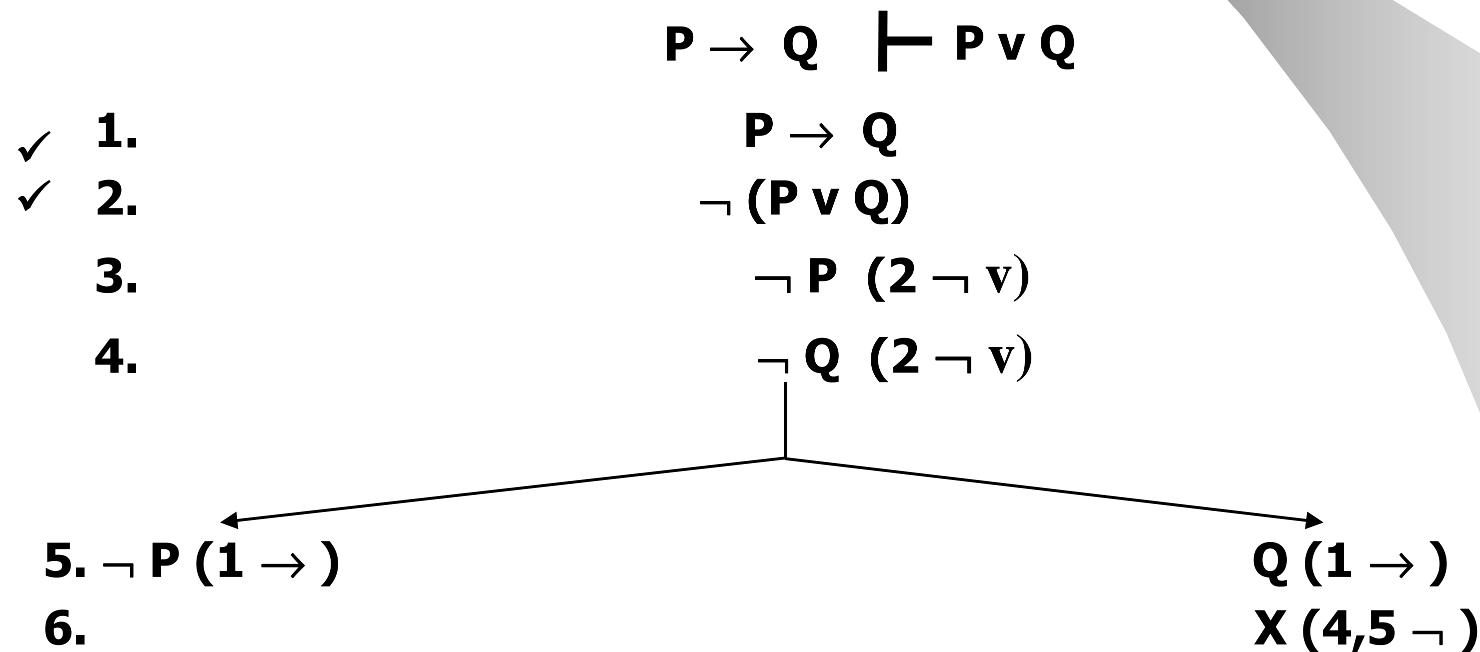
Árvore de Refutação

2.6. Condicional (→): Se um ramo aberto contém uma fórmula não ticada da forma Ø → ß, tica-se, Ø → ß e BIFURCA-SE o final de cada ramo aberto que contém Ø→ ß ticada, no final do primeiro ramo se esreve ¬ Ø e no final do segundo ramo se escreve ß.



Árvore de Refutação

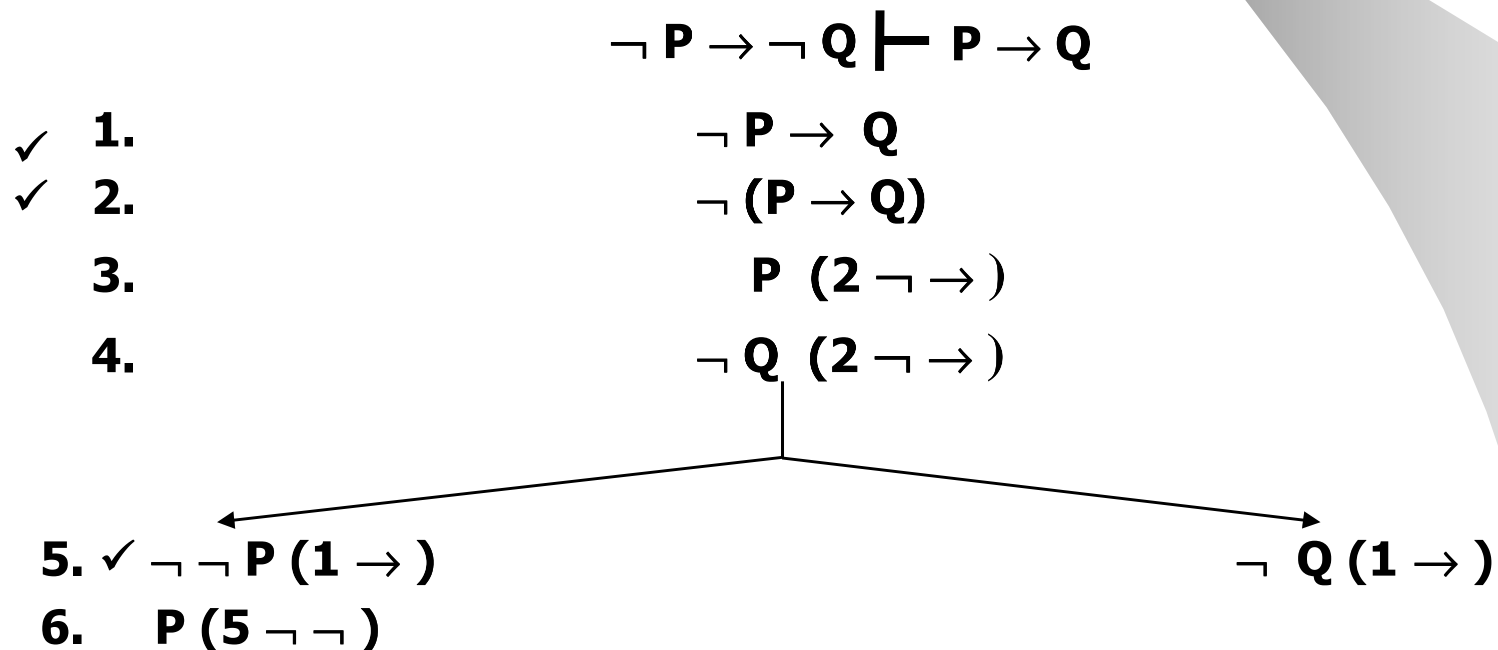
2.7. Disjunção Negada (¬ v): Se um ramo aberto contém uma fórmula não ticada da forma ¬ (Øvß), tica-se, ¬ (Øvß) e ESCREVE-SE ¬ Ø e ¬ ß no final de cada ramo aberto que contém ¬ (Øvß) ticada.



O ramo aberto indica que a forma é inválida

Árvore de Refutação

2.8. Condicional Negado (¬ →): Se um ramo aberto contém uma fórmula não ticada da forma ¬(Ø→ß), tica-se, ¬ (Ø → ß) e ESCREVE-SE Ø e ¬ ß no final de cada ramo aberto que contém ¬ (Ø→ß) ticada.



Árvore de Refutação

2.9. Bicondicional (↔): Se um ramo aberto contém uma fórmula não ticada da forma Ø ↔ ß, tica-se, Ø ↔ ß e BIFURCA-SE o final de cada ramo aberto que contém Ø ↔ ß ticada, no final do primeiro ramo se esreve Ø e ß e no final do segundo ramo se escreve ¬ Ø e ¬ ß.

2.10. Bicondicional Negado (¬ ↔): Se um ramo aberto contém uma fórmula não ticada da forma ¬ (Ø ↔ ß), tica-se, ¬ (Ø ↔ ß) e BIFURCA-SE o final de cada ramo aberto que contém ¬ (Ø ↔ ß) ticada, no final do primeiro ramo se esreve Ø e ¬ß e no final do segundo ramo se escreve ¬ Ø e ß.

P ↔ Q, ¬ P ⊢ ¬ Q

Árvore de Refutação Generalizada

- São uma generalização da técnica utilizada na Lógica Proposicional.
- A técnica de árvore de refutação generalizada incorpora as regras da lógica proposicional e acrescenta 6 novas regras para inferir em sentenças que contém quantificadores e o predicado de identidade.
- Algumas árvores do cálculo dos predicados empregam somente as regras do cálculo proposicional.
- NO CÁLCULO DE PREDICADOS, AS ÁRVORES DE REFUTAÇÃO NÃO APRESENTAM UMA LISTA COMPLETA DE CONTRA-EXEMPLOS, MAS SIM, UM "MODELO DE UNIVERSO" QUE CONTÉM EXATAMENTE OS OBJETOS MENCIONADOS PELO NOME NO RAMO.

Lógica de Primeira Ordem

- Árvores de Refutação

$\forall x\, P(x) \rightarrow \forall x\, G(x), \neg \forall x\, G(x) \vdash \neg \forall x\, P(x)$

✓ 1. $\forall x\, P(x) \rightarrow \forall x\, G(x)$

2. $\neg \forall x\, G(x)$

3. $\forall x\, P(x)$

4. $\neg \forall x\, P(x)$ 1→ | $\forall x\, G(x)$ 1→

5. X 3,4 ¬ | X 2,4 ¬

A árvore de refutação está COMPLETA, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou CONTRADIÇÕES, e portanto, a FORMA É VÁLIDA.

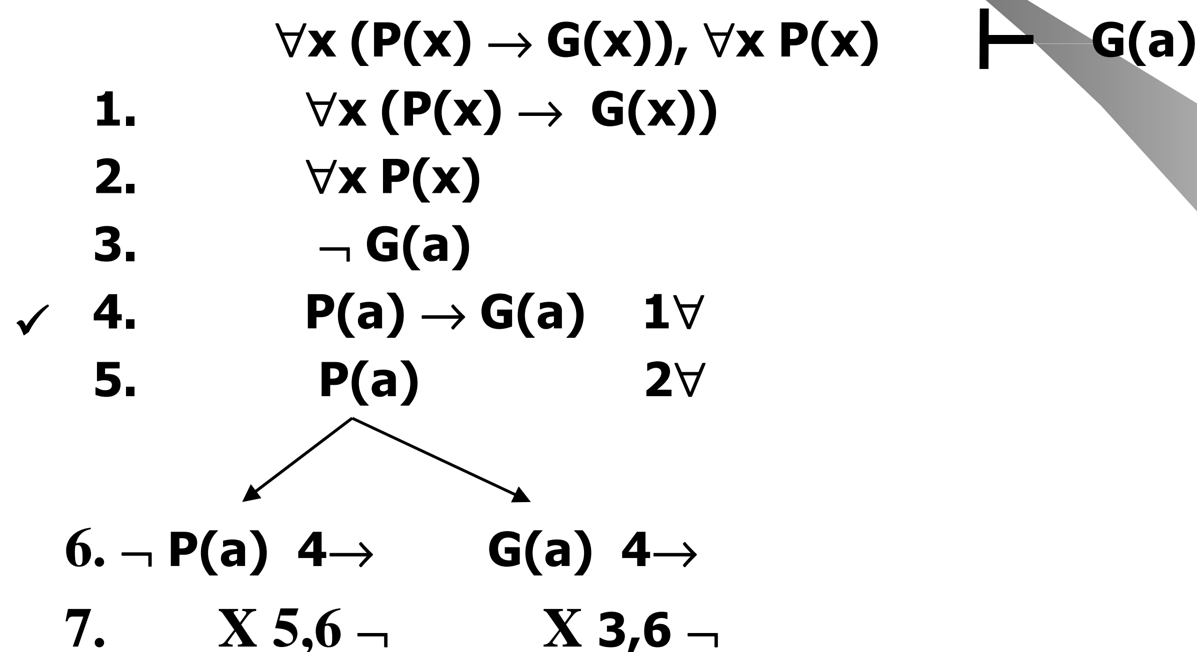
Lógica de Primeira Ordem

- Árvores de Refutação
  - 1. Quantificação Universal (∀):
    - Se uma fórmula bem formada do tipo ∀ ß Ø aparece num ramo aberto e se α é uma constante (ou letra nominal) que ocorre numa fbf naquele ramo, então ESCREVE-SE $Ø^{\alpha}$ / ß (o resultado de se substituir todas as ocorrências ß em Ø por α) no final do ramo.
    - Se nehuma fbf contendo uma letra nominal aparece no ramo, então escolhemos uma letra nominal α e ESCREVE-SE $Ø^{\alpha}$ / ß no final do ramo.
    - Em cada caso, NÃO TICAMOS ∀ ß Ø.

Lógica de Primeira Ordem

- Árvores de Refutação

- 1. Quantificação Universal (∀):

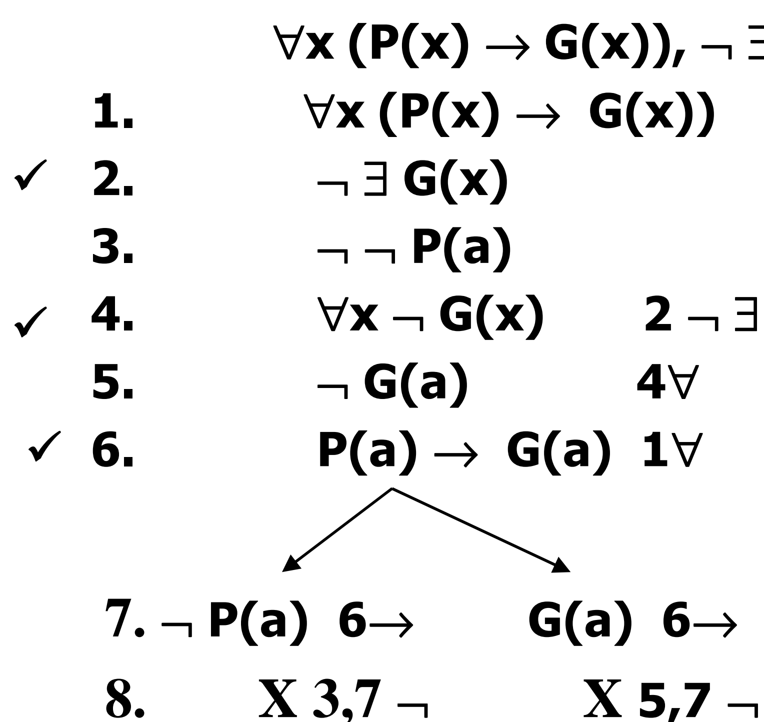


A árvore de refutação está COMPLETA, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou CONTRADIÇÕES, e portanto, a FORMA É VÁLIDA.

Lógica de Primeira Ordem

- Árvores de Refutação

- 2. Quantificação Existencial Negada (¬ ∃):
  - Se uma fórmula bem formada não ticada da forma ¬∃ßØ aparece num ramo aberto, tica-se a fórmula e ESCREVE-SE ∀ß¬ Ø no final de cada ramo aberto que contém a fbf ticada.



A árvore de refutação está COMPLETA, isto é, com todos os ramos fechados, logo, a busca de uma refutação para o argumento de negar a conclusão falhou, pois só encontrou CONTRADIÇÕES, e portanto, a FORMA É VÁLIDA.

Lógica de Primeira Ordem - Árvores de Refutação

- 3. Quantificação Universal Negada ($\neg \forall$):
  - Se uma fórmula bem formada não ticada da forma $\neg\forall\, \beta\emptyset$ aparece num ramo aberto, tica-se a fórmula e ESCREVE-SE $\exists\beta\neg\,\emptyset$ no final de cada ramo aberto que contém a fbf ticada.

$\exists x\, (\forall y\, P(x,y)) \vdash \forall x\, (\exists y\, P(y,x))$

✓	**1.**	$\exists x\, (\forall y\, P(x,y))$	
✓	**2.**	$\neg\forall x\, (\exists y\, P(y,x))$	
	3.	$\forall y\, P(a,y)$	**1** $\exists$
✓	**4.**	$\exists x\, (\neg \exists y\, P(y,x))$	**2** $\neg\forall$
✓	**5.**	$\neg \exists y\, P(y,b)$	**4** $\exists$
	6.	$\forall y\, \neg\, P(y,b)$	**5** $\neg\, \exists$
	7.	$\neg\, P(a,b)$	**6** $\forall$
	8.	$P(a,b)$	**3** $\forall$
	9.	**X**	**7,8** $\neg$

A fórmula testada é válida

Lógica de Primeira Ordem

- Árvores de Refutação

- 4. Quantificação Existencial (∃):
  - Se uma fórmula bem formada não ticada da forma ∃ßØ aparece num ramo aberto, tica-se a fórmula e escolhe-se uma letra nominal α QUE NÃO APARECEU NAQUELE RAMO e ESCREVE-SE $Ø^{\alpha}$ / ß (o resultado de se substituir todas as ocorrências ß em Ø por α) no final do ramo.

∃x P(x) ├ ∀x P(x)

✓	**1.**	**∃x P(x)**	
✓	**2.**	**¬∀x P(x)**	
	3.	**P(a)**	**1 ∃**
✓	**4.**	**∃x ¬ P(x)**	**2 ¬∀**
	5.	**¬ P(b)**	**4 ∃**

A fórmula testada é INVÁLIDA POR HAVER RAMOS ABERTOS (linha 5)

Lógica de Primeira Ordem

- Árvores de Refutação

– 5. Identidade (=):

- Se uma fórmula do tipo $\alpha = \beta$ aparece num ramo aberto e se uma outra fbf Ø contendo α ou β aparece não ticada naquele ramo, então escrevemos no final do ramo qualquer fbf que não esteja no ramo, que é o resultado de se substituir uma ou mais ocorrências de qualquer uma dessas letras nominais pela outra em Ø.
- Não se tica $\alpha = \beta$ nem Ø.

a = b ⊢ P(a,b) → P(b,a)

	1.	**a = b**	
	2.	**¬ (P(a,b) → P(b,a))**	
✓	**3.**	**¬ (P(a,a) → P(a,a))**	**1,2 =**
	4.	**P(a,a)**	**3 ¬ →**
	5.	**¬ P(a,a)**	**3 ¬ →**
	6.	**X**	**4,5 ¬**

A fórmula testada é válida

Lógica de Primeira Ordem

- Árvores de Refutação

- Identidade Negada (¬=):
  - Fechamos qualquer ramo aberto no qual uma fbf do tipo ¬ (α = α) ocorra.

a = b ⊢ b = a

	1.	**a = b**	
	2.	**¬ (b = a)**	
✓	**3.**	**¬ (a = a)**	**1,2 =**
	4.	**X**	**3 ¬ =**

A fórmula testada é válida

Prova Automática de Teoremas

- A capacidade de se demonstrar teoremas é uma das partes integrantes da inteligência humana.
- Este tipo de prova foi pesquisada e desenvolvida a partir da segunda metade dos anos 60.
- A partir da introdução, por Robinson e Smullyan, em 1960,de procedimentos eficientes para demonstração automática de teoremas por computador, a lógica passou a ser estudada também como método computacional para a solução de problemas.
- Uma das áreas que mais faz uso desta técnica é a dos Sistemas Especialistas (SEs).
- O objetivo principal da Prova Automática de Teoremas é provar que uma fórmula (teorema) é conseqüência lógica de outras fórmulas.

Prova Automática de Teoremas

- Os métodos adotados normalmente não utilizam a prova direta (através de regras de inferência), mas sim a PROVA POR REFUTAÇÃO (prova indireta), demonstrando que a negação da fórmula leva a inconsistências.
- SE A NEGAÇÃO DE UM TEOREMA É FALSA, ENTÃO ELE SERÁ VERDADEIRO.
- Os procedimentos de prova exploram o fato de expressões lógicas (fórmulas) poderem ser colocados em formas canônicas, isto é, apenas com os operadores "e", "ou" e "não".
- O método da prova por refutação aplicado à lógica de primeira ordem é muito conveniente e com seu emprego não haverá perda de generalidade, porém, exige-se que as fórmulas estejam na forma de cláusulas.

Prova Automática de Teoremas

- A TEORIA DA RESOLUÇÃO, proposta por Robinson em 1965 a partir dos trabalhos de Herbrand, Davis e Putnam, parte da transformação da fórmula a ser provada para a forma canônica conhecida como forma clausal.
- O método é baseado em uma regra de inferência única, chamada REGRA DA RESOLUÇÃO, e utiliza intensivamente um algoritmo de casamento de casamento de padrões chamado ALGORITMO DE UNIFICAÇÃO.
- O fato de ser possível associar uma semântica operacional a um procedimento de prova automática de teoremas permitiu a definição de uma linguagem de programação baseada em lógica, a linguagem PROLOG.
- Ainda hoje a área de prova automática de teoremas permanece bastante ativa, sendo objeto de diversas conferências internacionais.

Prova Automática de Teoremas

Algumas Definições

- PROVA: É a demonstração de que um teorema (ou fórmula) é verdadeiro.
- FORMA NORMAL CONJUNTIVA: É quando uma fórmula F for composta de uma conjunção de outras fórmulas (F1 ^ F2 ^ ... ^ Fn).
- FORMA NORMAL DISJUNTIVA: É quando uma fórmula F for composta de uma disjunção de outras fórmulas (F1 v F2 v ... v Fn).
- FORMA NORMAL PRENEX: É quando numa fórmula F, na lógica de primeira ordem, todos os quantificadores existentes prefixam a fórmula, isto é, se e somente se estiver na forma $Q_1 x_1 \ldots Q_n x_n (M)$.

Onde: $Q_i x_i = \forall x_i$ ou $\exists x_i$, e

(M) = uma fórmula que não contenha quantificadores.

Prova Automática de Teoremas

- **Procedimento para Obtenção da Forma Normal Prenex**

1. Eliminar os conectivos lógicos $\rightarrow$ e $\leftrightarrow$ usando as seguintes leis:
   - $F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$
   - $(F \rightarrow G) = \neg F \vee G$
2. Repetir o uso das seguintes leis:
   - $\neg \neg F = F$
   - $\neg (F \vee G) = \neg F \wedge \neg G$
   - $\neg (F \wedge G) = \neg F \vee \neg G$
   - $\neg (\forall x F(x)) = \exists x(\neg F(x))$
   - $\neg (\exists x F(x)) = \forall x(\neg F(x)$

   Estas leis são utilizadas para trazer os sinais de negação para antes dos átomos.
3. Padronizar as variáveis, se necessário, de modo que cada quantificador possua sua própria variável.

Prova Automática de Teoremas

- **Procedimento para Obtenção da Forma Normal Prenex**

4. Usar as leis abaixo de forma a mover os quantificadores para a esquerda da fórmula para obter a Forma Normal PRENEX.
   - $Qx\ F(x) \vee G = Qx\ (F(x) \vee G)$
   - $Qx\ F(x) \wedge G = Qx\ (F(x) \wedge G)$
   - $\forall x\ F(x) \wedge \forall x\ G(x) = \forall x\ (F(x) \wedge G(x))$
   - $\exists x\ F(x) \vee \exists x\ G(x) = \exists x\ (F(x) \vee G(x))$
   - $Q_1 x\ F(x) \vee Q_2 x\ G(x) = Q_1 x\ Q_2 z (F(x) \vee G(z))$
   - $Q_3 x\ F(x) \wedge Q_4 x\ G(x) = Q_3 x\ Q_4 z (F(x) \wedge G(z))$

EXEMPLO 1

$\forall x\ P(x) \rightarrow \exists x\ Q(x)$

- $\forall x\ P(x) \rightarrow \exists x\ Q(x) = \neg \forall x\ P(x) \vee \exists x\ Q(x)$
- $\exists x\ (\neg P(x)) \vee \exists x\ Q(x)$
- $\exists x\ (\neg P(x) \vee Q(x))$

Prova Automática de Teoremas

- **Procedimento para Obtenção da Forma Normal Prenex**

EXEMPLO 2

$\forall x\ \forall y\ ((\exists\ z\ (P(x,z)\ \hat{}\ P(y,z)) \rightarrow \exists\ u\ Q(x,y,u)) =$

- $\forall x\ \forall y\ (\neg\ (\exists\ z\ (P(x,z)\ \hat{}\ P(y,z)))\ v\ \exists\ u\ Q(x,y,u)) =$
- $\forall x\ \forall y\ (\forall z\ (\neg\ P(x,z)\ v\ \neg\ P(y,z)))\ v\ \exists\ u\ Q(x,y,u)) =$
- $\forall x\ \forall y\ \forall z \exists\ u\ (\neg\ P(x,z)\ v\ \neg\ P(y,z)\ v\ Q(x,y,u))$

Prova Automática de Teoremas

- **Eliminação dos quantificadores existenciais (Skolemização ou Funções de Skolem)**
  - Quando uma fórmula está na forma normal Prenex, pode-se eliminar os quantificadores existenciais por uma função, se as variáveis estiverem no escopo do quantificador universal; caso estejam fora, substitui-se por uma constante.
  - As constantes e funções usadas para substituir as variáveis existenciais são chamadas constante e funções de Skolem
  - Ex.: $\forall x\ \exists\ y\ P(x,y)$ Skolemizando: $\forall x\ P(x,f(x))$
    - onde f(x) tem por único propósito garantir que existe algum valor (y) que depende de x pois está dentro do seu escopo. No entanto, se o quantificador existencial não residir no escopo do quantificador universal, como em $\exists\ y\ \forall x\ P(x,y)$, a variável quantificada existencialmente será substituída por uma constante $\forall x\ P(x,a)$ que assegure sua existência, assim como sua independência de qualquer outra variável.

Prova Automática de Teoremas

- **Procedimento para Obtenção da Forma Clausal**
  - Cláusula é uma disjunção de literais

1. Passar para a forma normal PRENEX.
2. Skolemizar as variáveis quantificadas existencialmente.
3. Abandona-se os quantificadores pré-fixados.

EXEMPLO

$\forall x\ \forall y\ ((\exists\ z\ (P(x,z)\ \wedge\ P(y,z)) \rightarrow \exists\ u\ Q(x,y,u)) =$

- $\forall x\ \forall y\ (\neg\ (\exists\ z\ (P(x,z)\ \wedge\ P(y,z)))\ v\ \exists\ u\ Q(x,y,u)) =$
- $\forall x\ \forall y\ (\forall z\ (\neg\ P(x,z)\ v\ \neg\ P(y,z)))\ v\ \exists\ u\ Q(x,y,u)) =$
- $\forall x\ \forall y\ \forall z \exists\ u\ (\neg\ P(x,z)\ v\ \neg\ P(y,z)\ v\ Q(x,y,u))$
- $\forall x\ \forall y\ \forall z\ (\neg\ P(x,z)\ v\ \neg\ P(y,z)\ v\ Q(x,y,f(x,y,z)))$
- $\neg\ P(x,z)\ v\ \neg\ P(y,z)\ v\ Q(x,y,f(x,y,z))$

que é perfeitamente equivalente à fórmula original.

Prova Automática de Teoremas

- **Resolução: Um Procedimento Completo de Inferência**
  - Seria útil, do ponto de vista computacional, que tivéssemos um procedimento de prova que realizasse, em uma única operação, a variedade de processos envolvidos no raciocínio, com declarações da lógica dos predicados.
  - Este procedimento é a RESOLUÇÃO, que ganha sua eficiência por operar em declarações que foram convertidas à forma clausal, como mostrado anteriormente.
  - A Resolução produz provas por REFUTAÇÃO, ou seja, para provar uma declaração (mostrar que ela é válida), a resolução tenta demonstrar que a negação da declaração produz uma contradição com as declarações conhecidas (não é possível de ser satisfeita).

Prova Automática de Teoremas

- **Resolução: Um Procedimento Completo de Inferência**

A BASE DA RESOLUÇÃO

- É um processo interativo onde, em cada passo, duas cláusulas, denominadas cláusulas paternas, são comparadas (resolvidas), resultando em uma nova cláusula, dela inferida.
- A nova cláusula representa maneiras em que as duas cláusulas paternas interagem entre si.

Prova Automática de Teoremas

- **Resolução: Um Procedimento Completo de Inferência**

A BASE DA RESOLUÇÃO

Exemplo:

- Inverno v Verão
- ¬ Inverno v Frio

As duas cláusulas deverão ser verdadeiras (embora pareçam independentes, são realmente conjuntas).

- Agora, observamos que apenas um entre Inverno e ¬Inverno será verdadeiro, em qualquer ponto. Se Inverno for verdadeiro, então Frio também deverá ser, para garantir a verdade da segunda cláusula. Se ¬Inverno for verdadeiro, então também Verão deverá ser, para garantir a verdade da primeira cláusula.

Prova Automática de Teoremas

- **Resolução: Um Procedimento Completo de Inferência**
  - Assim, dessas duas cláusulas, podemos deduzir que
    - Verão v Frio
    - Esta é a dedução feita pelo procedimento de resolução.
  - A resolução opera tirando suas cláusulas que contenham cada uma, o mesmo literal, neste exemplo Inverno.
  - O literal deverá ocorrer na forma positiva numa cláusula e na forma negativa na outra.
  - O resolvente é obtido combinando-se todos os literais das duas cláusulas paternas, exceto aqueles que se cancelam.
  - Se a cláusula produzida for vazia, então foi encontrada uma CONTRADIÇÃO, o que valida a fórmula.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL
  - Na Lógica Proposicional, o procedimento para produzir uma prova pela resolução da proposição S, com relação a um conjunto de axiomas F, é o seguinte:

1. Converter todas as proposições de F em cláusulas.
2. Negar S e converter o resultado em cláusulas. Acrescente-as ao conjunto de cláusulas obtidas no passo 1.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL

  3. Repetir até que seja encontrada uma contradição ou não se possa fazer progresso:

     3.1. Escolher duas cláusulas, que serão chamadas cláusulas pais.

     3.2. Resolva-as. A cláusula resultante, denominada *resolvente*, será a disjunção de todos os literais de ambas as cláusulas pais, com a seguinte exceção:

     Se houver qualquer par de literais L e $\neg$ L, tal que uma das cláusulas pais contenha L e a outra $\neg$ L, então elimine tanto L como $\neg$ L do resolvente.

     3.3. Se o resolvente for uma cláusula vazia, terá sido encontrada uma contradição. Se não for, acrescente-o ao conjunto de cláusulas disponíveis para o procedimento.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL
  EXEMPLO: P, (P ^ Q) → R, S v T → Q , T|— R
  - Primeiro convertemos os axiomas em cláusulas.
    1. P
    2. ¬ P v ¬ Q v R
    3. ¬ S v Q
    4. ¬ T v Q
    5. T
    6. ¬ R
  - Começamos então a escolher a par de cláusulas para resolver. Embora qualquer par de cláusulas possa ser resolvido, apenas aqueles pares que contenham literais complementares produzirão um resolvente com possibilidade de produzir uma cláusula vazia.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA PROPOSICIONAL

  EXEMPLO: $P, (P \wedge Q) \rightarrow R, S \vee T \rightarrow Q, T \vdash R$

  - Começamos por resolver com a cláusula $\neg R$, pois ela é uma das cláusulas que deverão estar envolvidas na contradição que estamos tentando encontrar.

1. P
2. $\neg P \vee \neg Q \vee R$
3. $\neg S \vee Q$
4. $\neg T \vee Q$
5. T
6. $\neg R$

---

7. $\neg P \vee \neg Q$ (2 e 6)
8. $\neg Q$ (1 e 7)
9. $\neg T$ (4 e 8)
10. VAZIA (5 e 9)

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DOS PREDICADOS
  - Na Lógica Proposicional é fácil determinar que dois literais não possam ser verdadeiros ao mesmo tempo. (Simplesmente procure L e ¬ L)
  - Na Lógica dos Predicados este processo de casamento ("matching") é mais complicado.Por exemplo Homem(Henry) e ¬ Homem(Henry) é uma contradição, enquanto que Homem(Henry) e ¬Homem(Spot) não o é.
  - Assim, para determinar contradições, precisamos de um procedimento de matching que compare dois literais e descubra se existe um conjunto de substituições que os torne idênticos.
  - O ALGORITMO DE UNIFICAÇÃO é um procedimento recursivo direto que faz exatamente isto.

Prova Automática de Teoremas

- O ALGORITMO DE UNIFICAÇÃO
  - Para apresentar a unificação, consideramos as fórmulas como lista em que o primeiro elemento é o nome do predicado e os elementos restantes são os argumentos.
    - TentarAssassinar (Marco Cesar)
    - TentarAssassinar (Marco (Soberanode (Roma)))
  - Para tentar unificar dois literais, primeiro conferimos se seus primeiros elementos são iguais. Caso contrário não há meio de serem unificados, independentemente de seus argumentos.
  - Se o primeiro casar, podemos continuar com o segundo e assim por diante.
  - Constantes, funções e predicados diferentes não podem casar, os idênticos podem. Uma variável pode casar com outra variável, ou com qualquer constante, função ou expressão de predicados.

Prova Automática de Teoremas

- O ALGORITMO DE UNIFICAÇÃO

UNIFICA (L1, L2)

1. Se L1 ou L2 for um átomo, então faça o seguinte:
    1.1. Se L1 e L2 forem idênticos, retornar NIL
    1.2. Caso contrário, se L1 for uma variável, faça
        1.2.1. Se L1 ocorrer em L2, retornar F;
        1.2.2. Caso contrário, retornar (L2/L1)
    1.3. De outro modo, se L2 for uma variável, faça
        1.3.1. Se L2 ocorrer em L1, retornar F;
        1.2.2. Caso contrário, retornar (L1/L2)
    1.4. Caso contrário, retornar F.
2. Se comprimento(L1) não for igual a comprimento(L2) retornar F.
3. Designar a SUBST o valor NIL. (ao final do procedimento, SUBST conterá todas as substituições utilizadas para unificar L1 e L2).

Prova Automática de Teoremas

- O ALGORITMO DE UNIFICAÇÃO

UNIFICA (L1, L2)

4. Para i=1 até o número de elementos de L1, faça:
   - 4.1. Chame UNIFICA com o i-ésimo elemento de L1 e o i-ésimo elemento de L2, colocando o resultado em S.
   - 4.2. Se S = F, retornar F.
   - 4.3. Se S não for igual a NIL, faça:
     - 4.3.1. Aplicar S tanto ao final de L1 como de L2.
     - 4.3.2. SUBST := APPEND(S,SUBST)
     - 4.3.3. Retornar SUBST

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - Duas fórmulas-atômicas são contraditórias se uma delas puder ser unificada com o não da outra. Assim, por exemplo, Homem(x) e ¬ Homem(Spot) podem ser unificados.
  - Isto corresponde à intuição que diz que não pode ser verdadeiro para todos os x, que Homem(x) se houver conhecimento de haver algum x, digamos Spot, para o qual Homem(x) é falso.
  - Na lógica de predicados utilizaremos o algoritmo de unificação para localizar pares de fórmulas-atômicas que se cancelem.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  1. Converter todas as declarações de F em cláusulas.
  2. Negar S e converter o resultado em cláusulas. Acrescentá-las ao conjunto de cláusulas obtidas em 1.
  3. Repetir até que uma contradição seja encontrada, e nenhum progresso possa ser feito, ou até que se tenha gasto um quantidade pré-determinada de esforço:
     3.1. Escolher duas cláusulas e chamá-las de cláusulas pais.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS

  3.2. Resolvê-las. O resolvente será a disjunção de todos os literais de ambas as cláusulas pais com as substituições apropriadas realizadas, ressalvando-se o seguinte:

  3.2.1. Se houver um par de literais T1 e ¬ T2 tal que uma das cláusulas pais contenha T1 e a outra contenha T2, e ainda se T1 e T2 forem unificáveis, então nem T1 nem T2 devem aparecer no resolvente.

  3.2.2. Chamaremos T1 e T2 literais complementares. Utilize a substituição produzida pela unificação para criar o resolvente.

  3.3. Se o resolvente for uma cláusula vazia, então foi encontrada uma contradição. Se não for, acrescente-o ao conjunto de cláusulas disponíveis para o procedimento.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - Se a escolha de cláusulas a resolver em cada passo for feita de maneira sistemática, o procedimento de resolução encontrará uma contradição, se ela existir.
  - Isto contudo, poderá levar muito tempo.
  - Existem estratégias opcionais para acelerar o processo.
    - Resolver apenas pares de cláusulas que contenham literais complementares, pois somente essas resoluções produzem cláusulas novas mais difíceis de satisfazer que seus pais.
    - Eliminar cláusulas do tipo tautologias e cláusulas que estejam incluídas em outras cláusulas (P v Q é incluída por P).
    - Sempre que possível, resolver com uma das cláusulas que estamos tentando refutar ou com uma cláusula gerada por uma resolução com tal cláusula.
    - Sempre que possível, dar preferência a cláusulas com um único literal.

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - EXEMPLO:
  1. Marco era um homem.
  2. Marco era um pompeiano.
  3. Todos os pompeianos eram romanos.
  4. César era um soberano.
  5. Todos os romanos ou eram leiais a César ou o odiavam.
  6. Cada um de nós é leal a alguém.
  7. As pessoas tentam assassinar soberanos a quem não sejam leais.
  8. Marco tentou assassinar César.
  - Logo, Marco odiava César?

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - EXEMPLO:

1. Homem(Marco)
2. Pompeiano(Marco)
3. $\forall x$ Pompeiano(x) $\rightarrow$ Romano(x)
4. Soberano(Cesar)
5. $\forall x$ Romano(x) $\rightarrow$ (LealA(x,Cesar) v Odiar(x,Cesar))
6. $\forall x \exists y$ LealA(x,y)
7. $\forall x$ $\forall$ y (Homem(x) ^ Soberano(y) ^ TentarAssassinar(x,y) $\rightarrow$ ~LealA(x,y))
8. TentarAssassinar(Marco,Cesar)

- Logo, Odiar(Marco, Cesar)

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - EXEMPLO:
    - Primeiro convertemos os axiomas em cláusulas.

1. Homem(Marco)
2. Pompeiano(Marco)
3. ¬ Pompeiano(x1) v Romano(x1)
4. Soberano(Cesar)
5. ¬ Romano(x2) v LealA(x2,Cesar) v Odiar(x2,Cesar))
6. LealA(x3,f(x3))
7. ¬ Homem(x4) v ¬Soberano(y1) v ¬TentarAssassinar(x4,y1) v ¬LealA(x4,y1)
8. TentarAssassinar(Marco,Cesar)
9. ¬ Odiar(Marco, Cesar)

- Começamos então a escolher o par de cláusulas para resolver

Prova Automática de Teoremas

- RESOLUÇÃO NA LÓGICA DE PREDICADOS
  - EXEMPLO:

10. ¬ Romano(Marco) v LealA(Marco,Cesar) (SUBST(Marco,x2) em 5 e 9)
11. ¬ Pompeiano(Marco) v LealA(Marco,Cesar) (SUBST(Marco,x1 em 3 e 10)
12. LealA(Marco,Cesar)   (2 e 11)
13. ¬ Homem(Marco) v ¬ Soberano(Cesar) v ¬TentarAssassinar(Marco,Cesar)

    (SUBST(Marco,x4) e SUBST(Cesar,y1) em 7 e 12)
14. ¬ Soberano(Cesar) v ¬TentarAssassinar(Marco,Cesar) (1 e 13)
15. ¬TentarAssassinar(Marco,Cesar) (4 e 14)
16. VAZIA   (8 e 15)

PROLOG

- Introdução e Histórico
  - PROgramming in LOGic é fruto de pesquisas na área de Prova Automática de Teoremas.
  - Foi criada por Robert Kowalski (na parte teórica), Maarten van Emden (na demostração experimental) e Alain Colmerauer (na implementação) por volta de 1970 na Universidade de Marselha, França.
  - O primeiro compilador eficiente foi desenvolvido na Universidade de Edimburgo, Escócia.
  - A linguagem PROLOG também é um provador automático de teoremas, onde a estratégia de cláusulas adotada é a "Selective Linear Resolution for Definite Clauses".
  - É uma linguagem declarativa, onde se diz "o que fazer" para atingir um objetivo, o que leva a um nível mais elevado de abstração na solução dos problemas.

PROLOG

- Introdução e Histórico
  - Segundo Bratko, "pensar a respeito do problema e aprender a programar em PROLOG constitui-se em um desafio intelectual excitante".
  - Cada linha de PROLOG corresponde a uma afirmação.
  - A variável compreendida na afirmação deve ser entendida como UNIVERSALMENTE quantificada. Assim, a declaração pai_de(X,Y) corresponde a $\forall X \forall Y$ pai_de(X,Y).
  - PROLOG só admite em suas declarações CLÁUSULAS DE HORN.
  - CLÁUSULAS DE HORN só admitem um literal positivo.
  - Lembre-se que $A \rightarrow B$ pode ser escrita sob a forma $\neg A \vee B$.

PROLOG

- Cláusulas de Horn
  - Em lógica, uma cláusula de Horn é uma cláusula (disjunção de literais) com no máximo um literal positivo.
  - Uma cláusula de Horn com exatamente um literal positivo é dita uma cláusula definida (ou regra);
  - Uma cláusula de Horn sem literais positivos é às vezes dita cláusulaobjetivo (ou fato), especialmente no contexto da programação lógica.
  - Uma fórmula de Horn é uma fórmula na forma normal conjuntiva cujas cláusulas são todas de Horn; em outras palavras, é uma conjunção de cláusulas de Horn.
  - As cláusulas de Horn têm um papel essencial na programação lógica e são importantes na lógica construtiva.

PROLOG

- Cláusulas de Horn
  - A seguinte fórmula é um exemplo de cláusula de Horn (definida):

$$\neg p \vee \neg q \vee \cdots \vee \neg t \vee u$$

  - Em Prolog isto se escreve como:

    u :- p, q, ..., t

  - Usando a lógica clássica proposicional, tal fórmula pode ser reescrita ainda, de forma equivalente, da seguinte forma:

$$(p \wedge q \wedge \cdots \wedge t) \rightarrow u$$

PROLOG

- Cláusulas de Horn
  - A relevância das cláusulas de Horn para demonstrações de teoremas através do princípio da resolução reside no fato de que a resolução de duas cláusulas de Horn é uma cláusula de Horn.
  - Além disso, a resolução de uma cláusula objetivo e uma cláusula definida dá origem a uma nova cláusula objetivo, e resoluções deste gênero dão base à programação lógica e linguagem de programação Prolog.
  - No contexto da demonstração automática de teoremas, resoluções envolvendo cláusulas de Horn podem ser usadas para a definição de algoritmos eficientes para a verificação de teoremas (representados como uma cláusulas objetivos).:

PROLOG

- Introdução e Histórico
  - Sejam Ai(x1,x2,...xk) e Bi(x1,x2,...xk) fórmulas atômicas, então uma regra do tipo:

    Se A1(x1,x2,...xk) e ... e Am(x1,x2,...xk)

    então B1(x1,x2,...,xk) e ... e Bn(x1,x2,...,xk)

    pode ser escrita como

    ¬ A1(x1,x2,...xk) ou ... ou ¬ Am(x1,x2,...xk) e

    B1(x1,x2,...,xk) ou ... ou Bn(x1,x2,...,xk).
  - No entanto, PROLOG só admite declarações do tipo:
    - Se A1(x1,x2,...xk) e ... e Am(x1,x2,...xk) então B1(x1,x2,...,xk);
    - B1(x1,x2,...,xk);
    - Se A1 e ... e Am então B1; e
    - B1.

PROLOG

- Introdução e Histórico
  - Normalmente, variáveis e constantes são diferenciadas pela primeira letra:
    - Símbolos iniciados por minúscula são constantes; e
    - Símbolos iniciados por letra maiúscula são variáveis.
  - O escopo léxico de nomes de variáveis é apenas uma cláusula.
  - Isto quer dizer que, por exemplo, se o nome de variável X25 ocorre em duas cláusulas diferentes, então ela está representando duas variáveis diferentes.
  - Por outro lado, toda ocorrência de X25 dentro da mesma cláusula quer significar a mesma variável.
  - Esta situação é diferente para as constantes: o mesmo nome sempre significa o mesmo objeto ao longo de todo o programa.

PROLOG

- Introdução e Histórico
  - Em PROLOG, as cláusulas são escritas na forma de regras, com a (única) conclusão no início.
  - B1(X1,...,Xk) :- A1(X1,...,Xk),...,Am(X1,...,Xk).
  - B1(X1,...,Xk).
  - B1 :- A1,..,Am.
  - B1.
  - O único literal positivo de uma cláusula (que aparece antes do símbolo :-) é chamado cabeça da cláusula.
  - Os literais negativos (que aparecem depois do símbolo :-) são chamados corpo da cláusula.

PROLOG

- Resumo da Restrições Implícitas em PROLOG:
  - Para todo predicado p e toda variável X pertencente a p, suponha que $a_1, a_2,...,a_n$ constituam o domínio de X. O interpretador PROLOG, usando unificação, força:

1. O axioma do nome único. Para todos os átomos do domínio $a_i \neq a_j$, a menos que sejam idênticos. Isto implica que átomos com nomes distintos são distintos.
2. O axioma de mundo fechado.

   $p(X) \rightarrow p(a_1) \vee p(a_2) \vee ... \vee p(a_n)$.

   Isto significa que os únicos casos possíveis de uma relação são aqueles implicados pelas cláusulas presentes na especificação do problema.
3. O axioma do fechamento do domínio.

   $(X=a_1) \vee (X=a_2) \vee ... \vee (X=a_n)$.

   Isto garante que os átomos que ocorrem na especificação do problema constituem todos e os únicos átomos.

PROLOG

- Exemplo Introdutório
- progenitor(maria,josé). % Maria é progenitor de José.
- progenitor(joão, josé).
- progenitor(joão, ana).
- progenitor(josé, júlia).
- progenitor(josé, íris).
- progenitor(íris, jorge).
- masculino(joão). % João é do sexo masculino.
- masculino(josé).
- masculino(jorge).
- feminino(maria). % Maria é do sexo feminino.
- feminino(ana).
- feminino(júlia).
- feminino(íris).

PROLOG

- Exemplo Introdutório
- O efeito das entradas anteriores é o armazenamento destas fórmulas atômicas representando fatos em uma base de conhecimentos PROLOG.
- Se o programa for submetido a um sistema Prolog, este será capaz de responder algumas questões sobre a relação ali representada. Por exemplo: "José é o progenitor de Íris?".
  - ?-progenitor(josé, íris).
- Uma outra questão poderia ser: "Ana é um dos progenitores de Jorge?"
  - ?-progenitor(ana, jorge).

PROLOG

- Exemplo Introdutório
- Perguntas mais interessantes podem também ser formuladas, por exemplo: "Quem é progenitor de Íris?"
  - ?-progenitor(X, íris).
- Da mesma forma a questão "Quem são os filhos de José?" pode ser formulada com a introdução de uma variável na posição do argumento correspondente ao filhos de José
  - ?-progenitor(josé, X).
- Uma questão mais geral para o programa seria: "Quem é progenitor de quem?
  - ?-progenitor(X, Y).

PROLOG

- Exemplo Introdutório
- Pode-se formular questões ainda mais complicadas ao programa, como "Quem são os avós de Jorge?". Como nosso programa não possui diretamente a relação avô, esta consulta precisa ser dividida em duas etapas. A saber:

  (1) Quem é progenitor de Jorge? (Por exemplo, Y) e

  (2) Quem é progenitor de Y? (Por exemplo, X)
- Esta consulta em Prolog é escrita como uma seqüência de duas consultas simples, cuja leitura pode ser: "Encontre X e Y tais que X é progenitor de Y e Y é progenitor de Jorge".
  - ?-progenitor(X, Y), progenitor(Y, jorge).
    - X=josé Y=íris

PROLOG

- Pontos Básicos
- Uma relação como progenitor pode ser facilmente definida em Prolog estabelecendo-se as tuplas de objetos que satisfazem a relação;
- O usuário pode facilmente consultar o sistema Prolog sobre as relações definidas em seu programa;
- Um programa Prolog é constituído de cláusulas, cada uma das quais é encerrada por um ponto (.);
- Os argumentos das relações podem ser objetos concretos (como júlia e íris) ou objetos genéricos (como X e Y). Objetos concretos em um programa são denominados átomos, enquanto que os objetos genéricos são denominados variáveis;

PROLOG

- Pontos Básicos
- Consultas ao sistema são constituídas por um ou mais objetivos, cuja seqüência denota a sua conjunção;
- Uma resposta a uma consulta pode ser positiva ou negativa, dependendo se o objetivo correspondente foi alcançado ou não. No primeiro caso dizemos que a consulta foi bem-sucedida e, no segundo, que a consulta falhou;
- Se várias respostas satisfizerem a uma consulta, então o sistema Prolog irá fornecer tantas quantas forem desejadas pelo usuário.

PROLOG

- Exemplo Introdutório
  - A capacidade do PROLOG não se limita à busca em uma base de conhecimentos; é possível armazenar regras.
  - As regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira.

  ? -mae(X,Y) :- progenitor(X,Y), feminino(X).
  ? -pai(X,Y) :- progenitor(X,Y), masculino(X).
  ? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).

  - Com estas definições podemos obter os seguintes resultados:

  ? -mae(X,josé).
  X = maria;

  ? -pai(X,íris).
  X = josé;

PROLOG

- Pontos Básicos
- Programas Prolog podem ser ampliados pela simples adição de novas cláusulas;
- As cláusulas Prolog podem ser de três tipos distintos: fatos, regras e consultas;
- Os fatos declaram coisas que são incondicionalmente verdadeiras;
- As regras declaram coisas que podem ser ou não verdadeiras, dependendo da satisfação das condições dadas;
- Por meio de consultas podemos interrogar o programa acerca de que coisas são verdadeiras;

PROLOG

- Pontos Básicos
- As cláusulas Prolog são constituídas por uma cabeça e um corpo. O corpo é uma lista de objetivos separados por vírgulas que devem ser interpretadas como conjunções;
- Fatos são cláusulas que só possuem cabeça, enquanto que as consultas só possuem corpo e as regras possuem cabeça e corpo;
- Ao longo de uma computação, uma variável pode ser substituída por outro objeto. Dizemos então que a variável está instanciada;
- As variáveis são assumidas como universalmente quantificadas nas regras e nos fatos e existencialmente quantificadas nas consultas

PROLOG

- Exemplo Introdutório
  - Além disso, as definições de regras podem ser recursivas, isto é, uma cláusula de definição de um predicado pode conter este predicado em seu corpo:

```
?  -antepassado(X,Z) :- progenitor(X,Z).
?  -antepassado(X,Z) :- progenitor(X,Y),
                              antepassado(Y,Z).
```

  - Com estas definições podemos obter os seguintes resultados:

```
?  -antepassado(X,jorge).
X =  íris;
X = maria;
X = joão;
X = josé;
```

PROLOG

- Exemplo Introdutório
  - Usa-se o "_" (underscore) para indicar a irrelevância de um objeto

```
? -aniversario(maria,data(25,janeiro,1979)).
? -aniversario(joao,data(5,janeiro,1956)).
? -signo(Pessoa,aquario) :-
   aniversario(Pessoa,data(Dia,janeiro,_)), Dia >=20.
```

  - Com estas definições podemos obter os seguintes resultados:

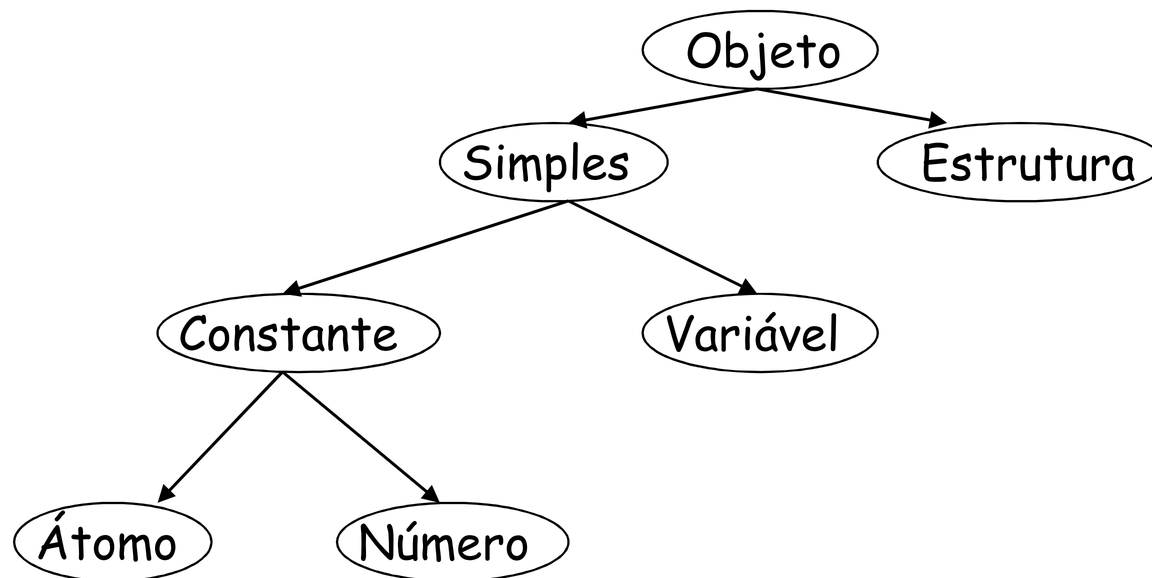
```
? -signo(Pessoa,aquario).
Pessoa =  maria;
no
```

  - Usa-se a "," como operador de conjunção e usa-se o ";" como operador de disjunção (cláusulas começando com o mesmo predicado também indicam a disjunção)

```
? -avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z).
? -amiga(X) :- (X = maria ; X = joana).
```

PROLOG

- Sintaxe
  - O sistema reconhece o tipo de um objeto no programa por meio de sua forma sintática.
  - Isto é possível porque o PROLOG especifica formas diferentes para cada tipo de obejto.



PROLOG

- Sintaxe
  - Átomos e Números
    - No exemplo introdutório viu-se informalmente alguns exemplos de átomos e variáveis. O alabeto básico adotado consiste dos seguintes símbolos:
    - Pontuação: ( ) . ' "
    - Conectivos: , (conjunção)
      ; (disjunção)
      :- (implicação)
    - Letras: a, b, c, ..., z, A, B, C, ..., Z
    - Dígitos: 0, 1, 2, ..., 9
    - Especiais: + - * / < > = ...

PROLOG

- Sintaxe
  - Variáveis
    - Variáveis PROLOG são cadeias de letras, dígitos e do caracter sublinhado (_), devendo iniciar com este ou com uma letra maiúscula.
  - Estruturas
    - Estruturas são objetos que possuem vários componentes.
    - Os próprios componentes, por sua vez, podem também ser estruturas.
    - Para combinar os elementos em uma estrutura é necessário um functor. Um functor é um símbolo funcional (nome de função) que permite agrupar diversos objetos em um único objeto estruturado.
    - data(13, outubro, 1993) - dois inteiros e um átomo.
    - data(Dia, marco, 1996) - um dia qualquer de marco.

PROLOG

- Sintaxe
  - Sintaticamente todos os objetos em PROLOG são denominados termos.
  - O conjunto de termos PROLOG é o menor conjunto que satisfaz às seguintes condições:
    - Toda constante é um termo;
    - Toda variável é um termo;
    - Se t1, t2, ..., tn são termos e f é um átomo, então f(t1,t2,...,tn) também é um termo, onde o átomo f desempenha o papel de um símbolo funcional n-ário. Diz ainda que a expressão f(t1,t2,...tn) é um termo funcional PROLOG.

PROLOG

- Consultas em Prolog
  - Uma consulta em Prolog é sempre uma seqüência composta por um ou mais objetivos. Para obter a resposta, o sistema Prolog tenta satisfazer todos os objetivos que compõem a consulta, interpretando-os como uma conjunção. Satisfazer um objetivo significa demonstrar que esse objetivo é verdadeiro, assumindo que as relações que o implicam são verdadeiras no contexto do programa. Se a questão também contém variáveis, o sistema Prolog deverá encontrar ainda os objetos particulares que, atribuídos às variáveis, satisfazem a todos os sub-objetivos propostos na consulta. A particular instanciação das variáveis com os objetos que tornam o objetivo verdadeiro é então apresentada ao usuário. Se não for possível encontrar, no contexto do programa, nenhuma instanciação comum de suas variáveis que permita derivar algum dos sub-objetivos propostos então a resposta será "não".

PROLOG

- Unificação
  - É a operação mais importante entre dois termos PROLOG.
  - Dados dois termos, diz-se que eles se unificam se:
    - Eles são idênticos, ou
    - As variáveis de ambos os termos podem ser instanciadas com objetos de maneira que, após a substituição das variáveis por estes objetos, os termos se tornam idênticos.
  - Exemplo
    - os termos data(D,M,1994) e data(X,marco,A) unificam. Uma instanciação que torna os dois termos idênticos é:

      D é instanciada com X;

      M é instanciada com marco;

      A é instanciada com 1994.
    - Por outro lado, os termos data(D,M,1994) e data(X,Y,94) não unificam, assim como não unificam data(X,Y,Z) e ponto(X,Y,Z).

PROLOG

- Unificação
  - Se os termos não unificam dizemos, dizemos que o processo FALHA.
  - Se eles unificam, então o processo é bem-sucedido.
  - As regras gerais que determinam se dois termos S e T unificam são:
    - Se S e T são constantes, então S e T unificam somente se ambos representam o mesmo objeto;
    - Se S é uma VARIÁVEL E t É QUALQUER COISA, ENTÃO S E t UNIFICAM COM INSTANCIADA EM t. Inversamente, se T é uma variável, então T é instanciada em S.
    - Se S e T são estruturas, unificam somente se:
      - S e T tem o mesmo functor principal, e
      - todos os seus componentes correspondentes também unificam. A instanciação resultante é determinada pela unificação dos componentes.

PROLOG

- Consultas em Prolog – Exemplo 1
  - antepassado(X, Z) :-   % X é antepassado de Z se
    progenitor(X, Z).% X é progenitor de Z. [pr1]
  - antepassado(X, Z)   % X é antepassado de Z se
    progenitor(X, Y), % X é progenitor de Y e
    antepassado(Y, Z).% Y é antepassado de Z. [pr2]

PROLOG

- Consultas em Prolog
- Um exemplo mais complexo:

?-antepassado(joão, íris).

- Sabe-se que progenitor(josé, íris) é um fato. Usando esse fato e a regra [pr1], podemos concluir antepassado(josé, íris). Este é um fato derivado. Não pode ser encontrado explícito no programa, mas pode ser derivado a partir dos fatos e regras ali presentes. Ou seja:
- "de progenitor(josé, íris) segue, pela regra [pr1] que antepassado(josé, íris)".
- Além disso sabemos que progenitor(joão, josé) é fato. Usando este fato e o fato derivado, antepassado(josé, íris), podemos concluir, pela regra [pr2], que o objetivo proposto, antepassado(joão, íris) é verdadeiro.

PROLOG

- Consultas em Prolog
- Mostrou-se assim o que pode ser uma seqüência de passos de inferência usada para satisfazer um objetivo. Tal seqüência denomina-se seqüência de prova. A extração de uma seqüência de prova do contexto formado por um programa e uma consulta é obtida pelo sistema na ordem inversa da empregada anteriormente.

PROLOG

- Consultas em Prolog
- Ao invés de iniciar a inferência a partir dos fatos, o Prolog começa com os objetivos e , usando as regras, substitui os objetivos correntes por novos objetivos até que estes se tornem fatos.
- Assim,para saber se joão é antepassado de íris, o sistema tenta encontrar uma cláusula no programa a partir da qual o objetivo seja conseqüência imediata. Obviamente, as únicas cláusulas relevantes para essa finalidade são [pr1] e [pr2], que são sobre a relação antepassado, porque são as únicas cujas cabeças podem ser unificadas com o objetivo formulado.
- Tais cláusulas representam dois caminhos alternativos que o sistema pode seguir. Inicialmente o Prolog irá tentar a que aparece em primeiro lugar no programa:

PROLOG

- Consultas em Prolog
- antepassado(X, Z) :- progenitor(X, Z).
- Uma vez que o objetivo é antepassado(joão, íris), as variáveis na regra devem ser instanciadas por X=joão e Y=íris. O objetivo inicial, antepassado(joão, íris) é então substituído por um novo objetivo:

  progenitor(joão, íris)
- Não há, entretanto, nenhuma cláusula no programa cuja cabeça possa ser unificada com progenitor(joão, íris), logo este objetivo falha. Então o Prolog retorna ao objetivo original (backtracking) para tentar um caminho alternativo que permita derivar o objetivo antepassado(joão, íris). A regra [pr2] é então tentada:
- antepassado(X, Z) :-

  progenitor(X, Y),

  antepassado(Y, Z).

PROLOG

- Consultas em Prolog
- Como anteriormente, as variáveis X e Z são instanciadas para joão e íris, respectivamente. A variável Y, entretanto, não está instanciada ainda. O objetivo original, antepassado(joão, íris) é então substituído por dois novos objetivos derivados por meio da regra [pr2]:
- progenitor(joão, Y), antepassado(Y, íris).
- Encontrando-se agora face a dois objetivos, o sistema tenta satisfazê-los na ordem em que estão formulados. O primeiro deles é fácil: progenitor(joão, Y) pode ser unificado com dois fatos do programa: progenitor(joão, josé) e progenitor(joão, ana). Mais uma vez, o caminho a ser tentado deve corresponder à ordem em que os fatos estão escritos no programa. A variável Y é então instanciada com josé nos dois objetivos acima, ficando o primeiro deles imediatamente satisfeito.

PROLOG

- Consultas em Prolog
- O objetivo remanescente é então:
- antepassado(josé, íris).
- Para satisfazer tal objetivo, a regra [pr1] é mais uma vez empregada.
- Essa segunda aplicação de [pr1], entretanto, nada tem a ver com a sua utilização anterior, isto é, o sistema Prolog usa um novo conjunto de variáveis na regra cada vez que esta é aplicada.

PROLOG

- Consultas em Prolog
- A cabeça da regra deve então ser unificada como o nosso objetivo corrente, que é antepassado(josé, íris). A instanciação de X' e Y' fica: X'=josé e Y'=íris e o objetivo corrente é substituído por:
- progenitor(josé, íris)
- Esse objetivo é imediatamente satisfeito, porque aparece no programa como um fato. O sistema encontrou então um caminho que lhe permite provar, no contexto oferecido pelo programa dado, o objetivo originalmente formulado, e portanto responde "sim".

PROLOG

- Consultas em Prolog – Exemplo 2

```
% grafo orientado
caminho(a,b).
caminho(b,c).
/* verifica se existe ligação entre
dois nodos dos grafo */
ligacao(X,Y) :- caminho(X,Y).
ligacao(X,Y) :- caminho(X,Z), ligacao(Z,Y).
```

  - Lembrando:
    - Fatos são cláusula só com cabeça e de corpo vazio.
    - Regras são cláusula com cabeça e corpo não vazio.
    - Consultas são cláusulas de Horn com cabeça vazia.

PROLOG

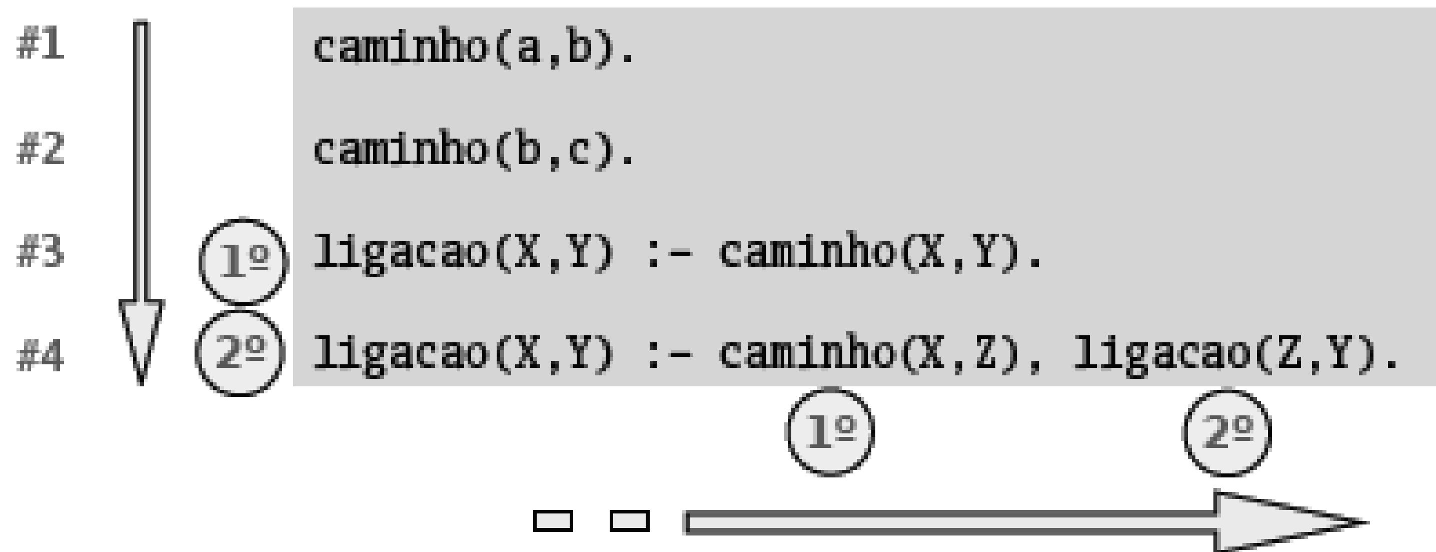
- Consultas em Prolog – Exemplo 1
  - As consultas são um meio de extrair informação de um programa. As variáveis que ocorrem nas questões são quantificadas existencialmente.
  - Responder a uma questão é determinar se ela é uma consequência lógica do programa.
  - Exemplo:

```
| ?- ligacao(a,K).
K = b ?
yes
| ?- ligacao(a,K).
K = b ? ;
K = c ? ;
no
```

o ";" força o backtracking, isto é, pede para serem produzidas novas provas alternativas.

PROLOG

- Estratégia de Procura de Soluções
  - Top Down (de cima para baixo)
  - Depth-First (profundidade máxima antes de tentar um novo ramo)
  - Backtracking (volta a tentar encontrar uma prova alternativa)



PROLOG

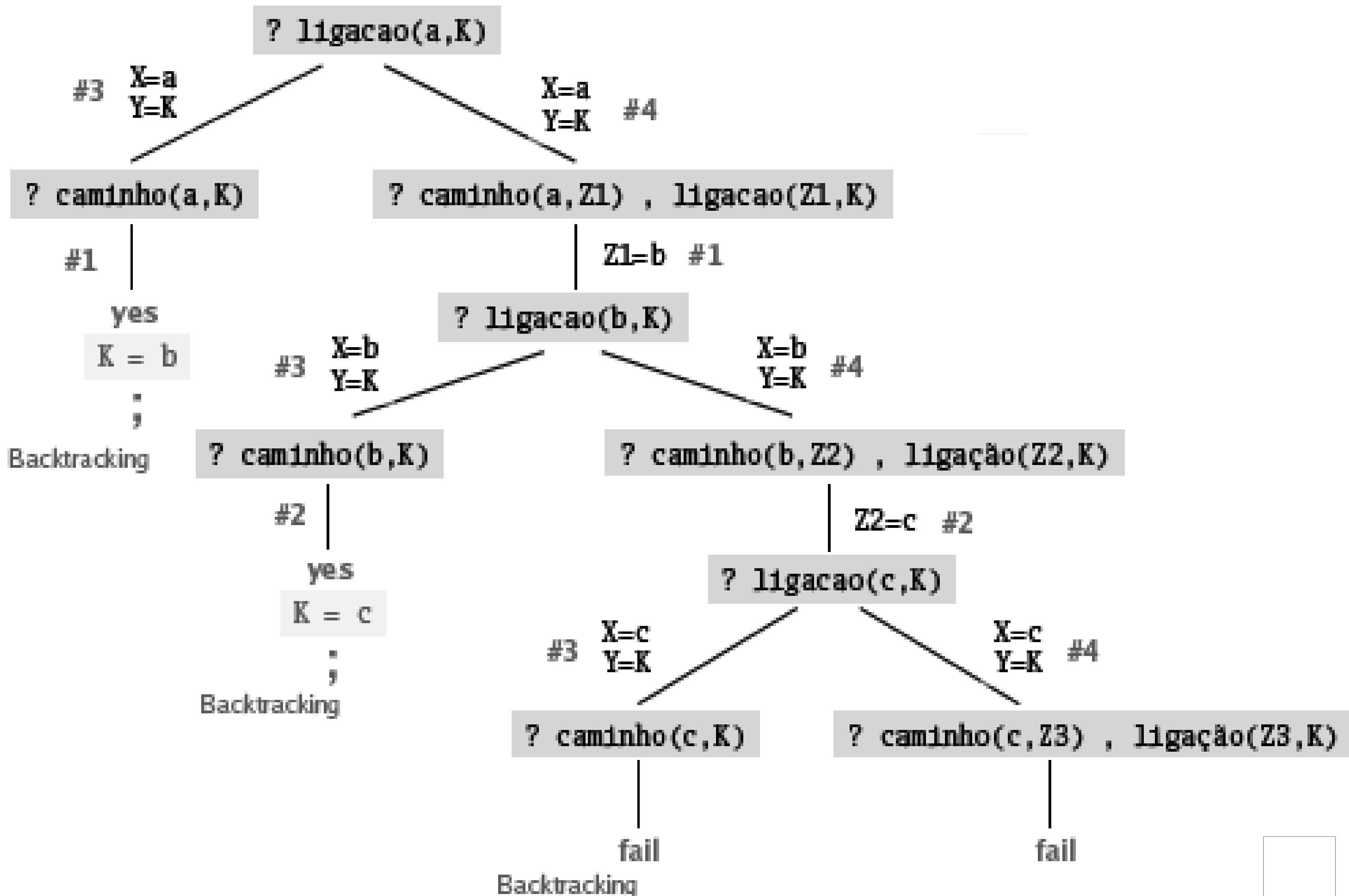
- Estratégia de prova do motor de inferência do Prolog
  - Assuma que o objetivo a provar é: ? G1, G2, ..., Gn
  - O motor de inferência pesquisa a base de conhecimento (de cima para baixo) até encontrar uma regra cuja cabeça unifique com G1. Essa unificação produz uma substituição (o unificador mais geral) Θ
    - Se C :- P1, ..., Pm é a regra encontrada. Θ é tal que C Θ = G1 Θ .

    O novo objetivo a provar é agora ? P1 Θ, ..., Pm Θ, G2 Θ, ..., Gn Θ
    - Se a regra encontrada é um fato F. Θ é tal que F Θ = G1 Θ.

    O novo objetivo a provar é agora ? G2 Θ, ..., Gn Θ
  - A prova termina quando já não há mais nada a provar.
  - O interpretador responde à questão inicial indicando a substituição a que têm que ser sujeitas as variáveis presentes na questão inicial, para produzir a prova.

PROLOG

- Árvore de Busca



PROLOG

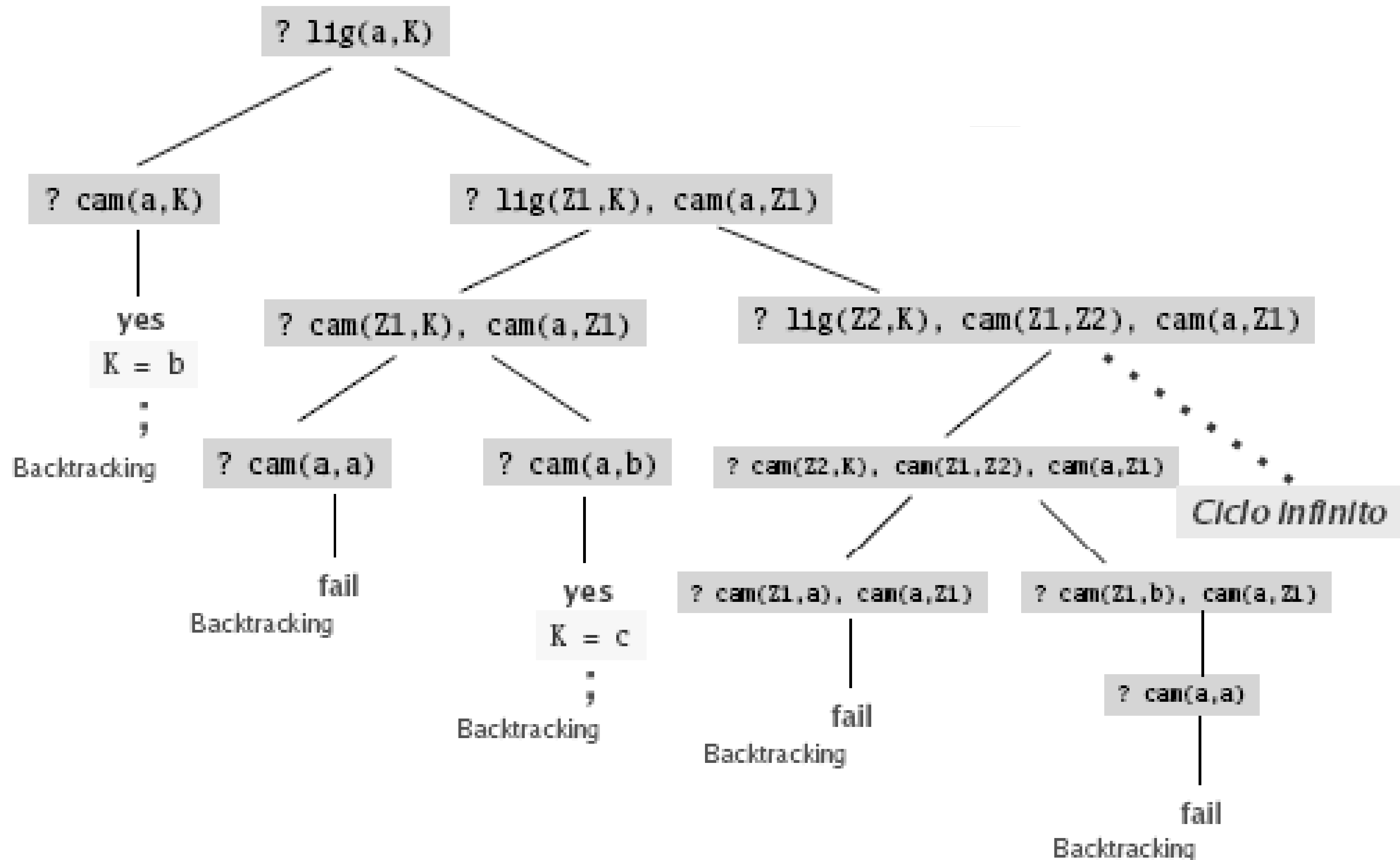
- Observação: Se o programa fosse:

```
% grafo orientado
caminho(a,b).
caminho(b,c).
/* verifica se existe ligação entre
dois nodos dos grafo */
lig(X,Y) :- caminho(X,Y).
lig(X,Y) :- lig(Z,Y),caminho(X,Z).
```

  - Teríamos uma árvore de busca INFINITA

PROLOG

- Árvore de Busca



PROLOG

- Semântica
  - PROLOG tem três semânticas: a DECLARATIVA, a PROCEDURAL e a OPERACIONAL.
  - A semântica DECLARATIVA é aquela em que se escreve e lê o programa PROLOG e que é uma representação do que se conhece da definição do problema a resolver.
  - A semântica declarativa pode ser interpretada de três modos distintos:
    - Para resolver um problema dá-se uma nova cláusula, a pergunta, e o PROLOG tenta verificar se esta cláusula é compatível com o mundo definido;
    - Considera-se que os literais na cabeça e cauda de cada cláusula são objetivos a serem atingidos. Uma pergunta tem resposta afirmativa se o objetivo que ela define é satisfeito usando as regras do programa;
    - Olha as cláusulas como regras de uma gramática, onde cada regra de PROLOG corresponde a uma regra da gramática.

PROLOG

- Semântica
  - Exemplo
    - Seja P :- Q, R

    onde P, Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura declarativa destas cláusulas são:
      - P é verdadeira se Q e R são verdadeiras, e
      - De Q e R, segue P.
  - A semântica declarativa determina se um dado objetivo é verdadeiro e, se for, para que valores de variáveis isto se verifica.
  - Assim, dado um programa e um objetivo G, o significado declarativo nos diz que:
    - Um objetivo G é verdadeiro (isto é, é satisfatível ou segue logicamente do programa) se e somente se há uma cláusula C no programa e uma instância I de C tal que:
      - A cabeça de I é idêntica a G, e
      - Todos os objetivos no corpo de I são verdadeiros.

PROLOG

- Semântica
  - A semântica PROCEDURAL define não apenas o relacionamento lógico existente entre a cabeça e o corpo da cláusula, como também exige a existência de uma ordem na qual os objetivos serão processados.
  - Exemplo
    - Seja P :- Q, R

    onde P, Q e R possuem a sintaxe de termos PROLOG. Duas alternativas para a leitura procedural destas cláusulas são:

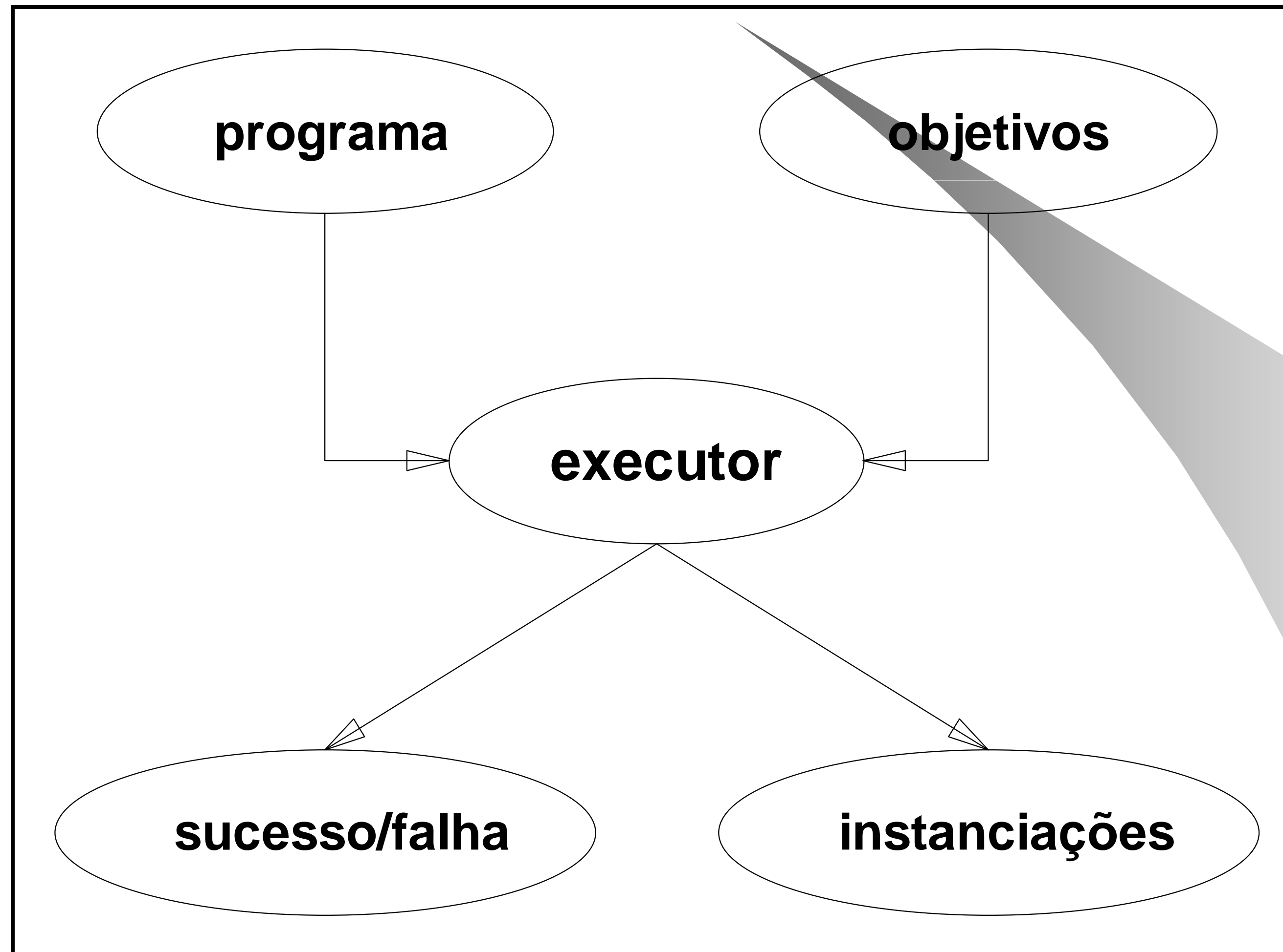
      - Para solucionar o problema P
        primeiro solucione o subproblema Q
        e depois solucione o subproblema R.
      - Para satisfazer P, primeiro satisfaça Q e depois R.

PROLOG

- Semântica
  - A semântica OPERACIONAL define como PROLOG responde a uma pergunta.
  - Seria ótimo se não fosse necessário conhecer esta semântica, pois neste caso, PROLOG seria realmente uma implementação do paradigma de programação em lógica. Entretanto, este não é o caso.
  - PROLOG pesquisa se a pergunta é verdadeira ou falsa construindo a árvore de possibilidades para trás e faz a busca em profundidade. Se a árvore a percorrer é muito grande, o tempo pode se tornar proibitivo e é conveniente restringir o espaço de busca o mais possível, o que só é possível sabendo como o PROLOG vai visitar os nós da árvore.
  - O modo como PROLOG vai visitar os nós da árvore varia se as declarações são apresentadas em ordem diferente. Consequentemente, PROLOG não é realmente uma linguagem declarativa.

PROLOG

- Semântica



PROLOG

- Semântica
  - Suas entradas e saídas são:
    - entrada: um programa e uma lista de objetivos;
    - saída: um indicador de sucesso/falha e instanciações de variáveis.
  - O significado dos resultados de saída do *executor* é o seguinte:
    - O indicador de *sucesso/falha* tem o valor "sim" se os objetivos forem todos satisfeitos e "não" em caso contrário;
    - As *instanciações* são produzidas somente no caso de conclusão bem-sucedida e correspondem aos valores das variáveis que satisfazem os objetivos.

PROLOG

- Semântica (Resumo)
  - A interpretação declarativa de programas escritos em Prolog puro não depende da ordem das cláusulas nem da ordem dos objetivos dentro das cláusulas;
  - A interpretação procedimental depende da ordem dos objetivos e cláusulas. Assim a ordem pode afetar a eficiência de um programa. Uma ordenação inadequada pode mesmo conduzir a chamadas recursivas infinitas;
  - A semântica operacional representa um procedimento para satisfazer a lista de objetivos no contexto de um dado programa. A saída desse procedimento é o valor-verdade da lista de objetivos com a respectiva instanciação de sua variáveis. O procedimento permite o retorno automático (backtracking) para o exame de novas alternativas;

PROLOG

- Backtracking
  - Na execução dos programas Prolog, a evolução da busca por soluções assume a forma de uma árvore - denominada "árvore de pesquisa" ou "search tree" - que é percorrida sistematicamente de cima para baixo (top-down) e da esquerda para direita, segundo o método denominado "depth-first search" ou "pesquisa primeiro em profundidade".

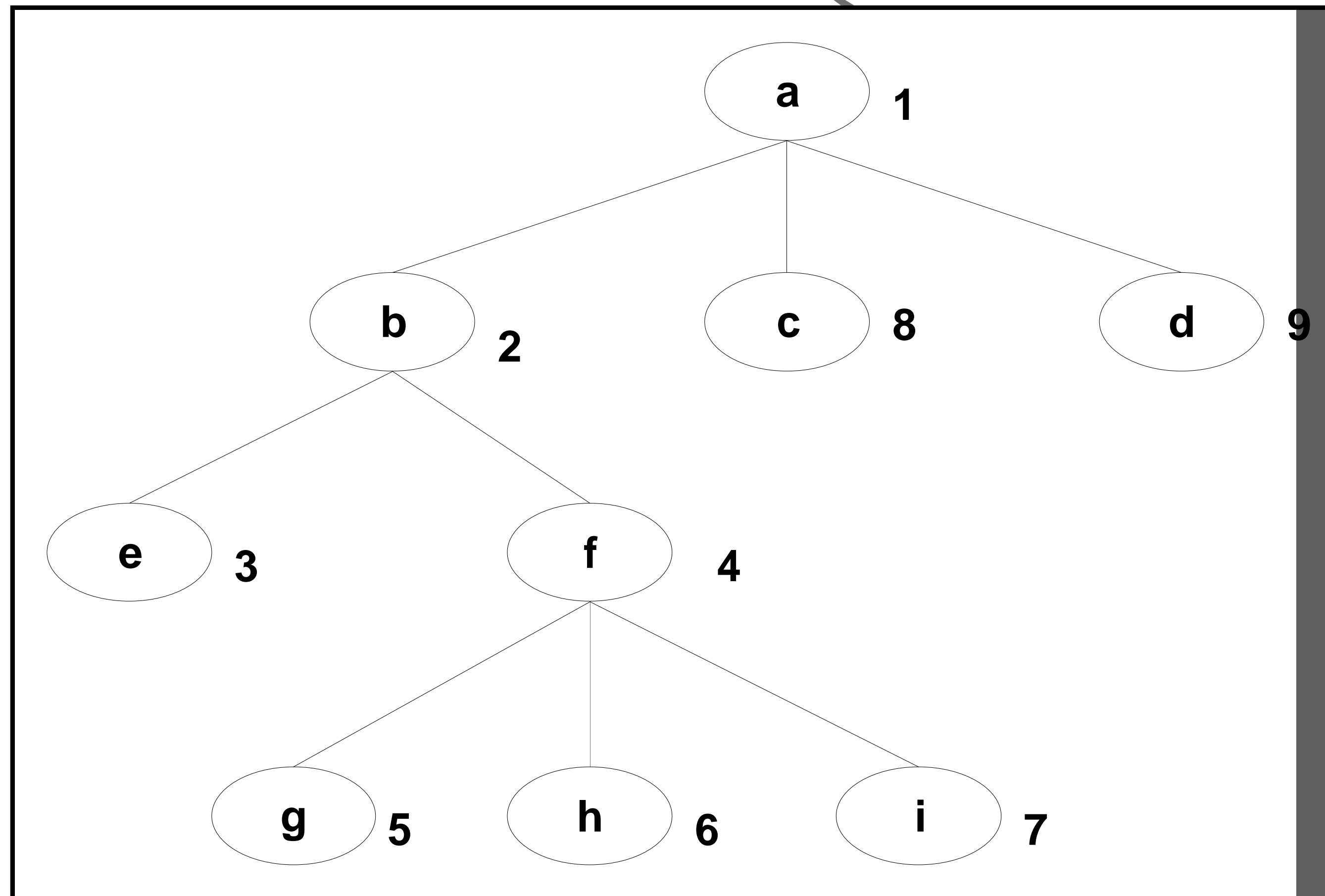
Exemplo

- Sejam a,b,c,etc... termos PROLOG

```
a :- b.
a :- c.
a :- d.
b :- e.
b :- f.
f :- g.
f :- h.
f :- i.
d.
```

PROLOG

- Backtracking
  - Ordem de visita aos nodos da árvore



a, b, e, (b), f, g, (f), h, (f), i, (f), (b), (a), c, (a), d

onde o caminho em backtracking é representado entre parênteses

PROLOG

- Backtracking
  - Como foi visto, os objetivos em um programa Prolog podem ser bem-sucedidos ou falhar.
  - Para um objetivo ser bem-sucedido ele deve ser unificado com a cabeça de uma cláusula do programa e todos os objetivos no corpo desta cláusula devem também ser bem-sucedidos. Se tais condições não ocorrerem, então o objetivo falha.
  - Quando um objetivo falha, em um nodo terminal da árvore de pesquisa, o sistema Prolog aciona o mecanismo de backtracking, retornando pelo mesmo caminho percorrido, na tentativa de encontrar soluções alternativas.
  - Ao voltar pelo caminho já percorrido, todo o trabalho executado é desfeito.

PROLOG

- Backtracking

  Exemplo 2

  - gosta(joão, jazz).
  - gosta(joão, renata).
  - gosta(joão, lasanha).
  - gosta(renata, joão).
  - gosta(renata, lasanha).

  – queremos saber do que ambos, joão e renata, gostam. Isto pode ser formulado pelos objetivos:

  - gosta(joão, X), gosta(renata, X).

  1. Encontra que joão gosta de jazz
  2. Instancia X com "jazz"
  3. Tenta satisfazer o segundo objetivo, determinando se "renata gosta de jazz"
  4. Falha, porque não consegue determinar se renata gosta de jazz
  5. Realiza um backtracking na repetição da tentativa de satisfazer gosta(joão, X), esquecendo o valor "jazz"

PROLOG

- Backtracking

  Exemplo 2

  6. Encontra que joão gosta de renata
  7. Instancia X com "renata"
  8. Tenta satisfazer o segundo objetivo determinando se "renata gosta de renata"
  9. Falha porque não consegue demonstrar que renata gosta de renata
  10. Realiza um backtracking, mais uma vez tentando satisfazer gosta(joão, X), esquecendo o valor "renata"
  11. Encontra que joão gosta de lasanha
  12. Instancia X com "lasanha"
  13. Encontra que "renata gosta de lasanha"
  14. É bem-sucedido, com X instanciado com "lasanha"

PROLOG

- Impurezas de PROLOG
  - O backtracking automático é uma ferramenta muito poderosa e a sua exploração é de grande utilidade para o programador. Às vezes, entretanto, ele pode se transformar em fonte de ineficiência. A seguir se introduzirá um mecanismo para "podar" a árvore de pesquisa, evitando o backtracking quando este for indesejável.
  - Para aumentar a eficiência no percurso da árvore de busca da solução do problema usam-se essencialmente dois operadores: CORTE ("CUT" representado por !) e FALHA ("FAIL").
  - Seu uso deve ser considerado pelas seguintes razões:
    - O programa irá executar mais rapidamente, porque não irá desperdiçar tempo tentando satisfazer objetivos que não irão contribuir para a solução desejada.
    - (ii) Também a memória será economizada, uma vez que determinados pontos de backtracking não necessitam ser armazenados para exame posterior.

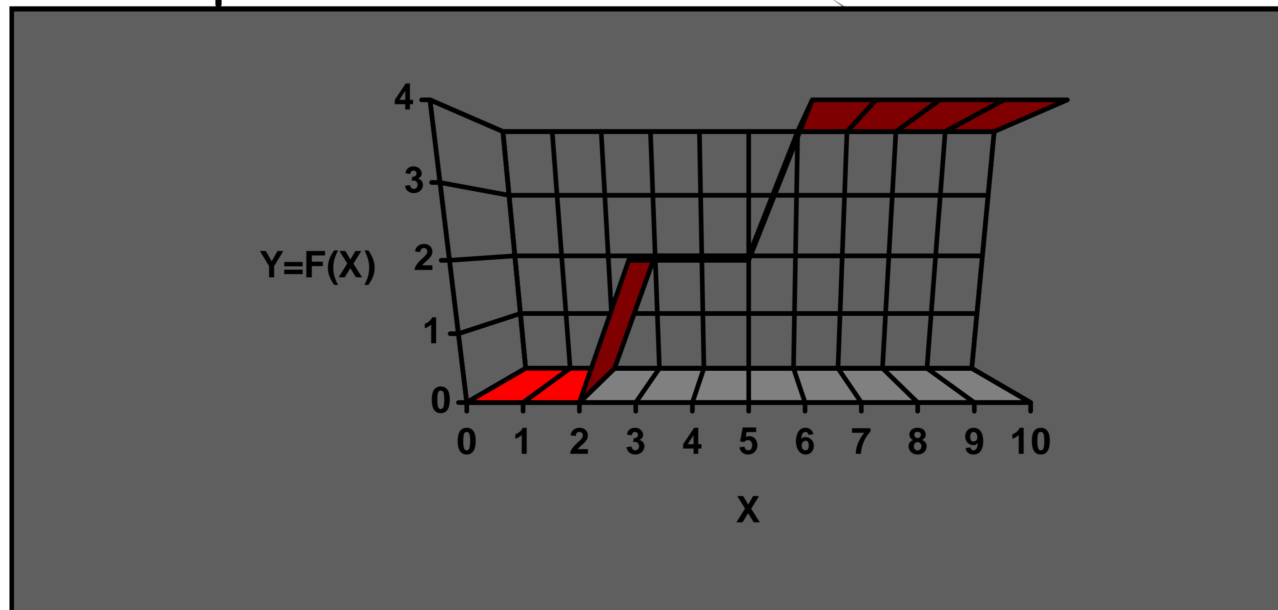
PROLOG

- CUT
  - Algumas das principais aplicações do cut são as seguintes:
    - Unificação de padrões, de forma que quando um padrão é encontrado os outros padrões possíveis são descartados
    - Na implementação da negação como regra de falha
    - Para eliminar da árvore de pesquisa soluções alternativas quando uma só é suficiente
    - Para encerrar a pesquisa quando a continuação iria conduzir a uma pesquisa infinita, etc

PROLOG

- CUT

Exemplo



(1) Se X < 3, então Y = 0
(2) Se 3 >= X e X < 6, então Y = 2
(3) Se 6 >= X, então Y = 4
que podem ser escritas em Prolog como uma relação binária f(X, Y), como se segue:

```
f(X, 0) :- X < 3.
f(X, 2) :- 3 =< X, X < 6.
f(X, 4) :- 6 =< X.
```

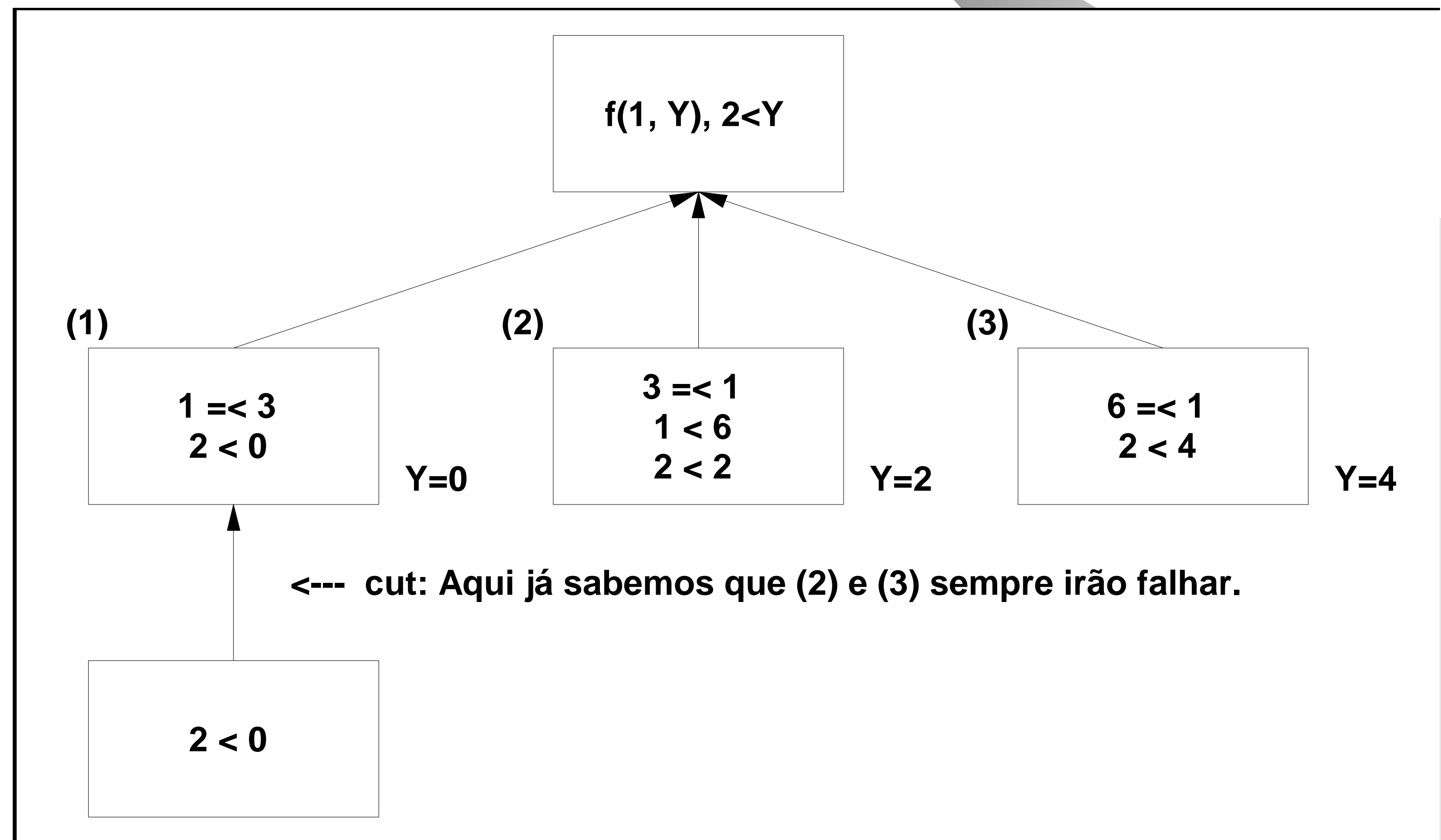
PROLOG

- CUT

  Exemplo

  – Vamos analisar o que ocorre quando a seguinte questão é formulada:

    - ?-f(1, Y), 2 < Y



PROLOG

- CUT

  Exemplo

  – O programa do exemplo, reescrito com cuts assume o seguinte aspecto:

```
f(X, 0)  :- X < 3, !.
f(X, 2)  3 =< X, X < 6, !.
f(X, 4)  6 =< X.
```

- Aqui o símbolo "!" evita o backtracking nos pontos em que aparece no programa.

PROLOG

- FALHA (FAIL)
  - Negação por Falha
  - "Maria gosta de todos os animais, menos de cobras". Como podemos dizer isto em Prolog? É fácil expressar uma parte dessa declaração: Maria gosta de X se X é um animal, isto é:
    - gosta(maria, X) :- animal(X).

    mas é necessário ainda excluir as cobras. Isto pode ser conseguido empregando-se uma formulação diferente:
    - Se X é uma cobra,
      então não é verdade que maria gosta de X
      senão se X é um animal, então maria gosta de X.

PROLOG

- FALHA (FAIL)
  - Podemos dizer que alguma coisa não é verdadeira em Prolog por meio de um predicado pré-definido especial, "fail", que sempre falha, forçando o objetivo pai a falhar. A formulação acima pode ser dada em Prolog com o uso do fail da seguinte maneira:
    - gosta(maria, X) :- cobra(X), !, fail.
    - gosta(maria, X) :-animal(X).
  - Aqui a primeira regra se encarrega das cobras. Se X é uma cobra, então o cut evita o backtracking (assim excluindo a segunda regra) e o fail irá ocasionar a falha da cláusula. As duas regras podem ser escritas de modo mais compacto como uma única cláusula, por meio do uso do conetivo ";":
    - gosta(maria, X) :- cobra(X), !, fail; animal(X).