

Inferência em Lógica de Primeira Ordem

Profa. Mercedes Gonzales
Márquez

7 Regras de Inferência da Lógica Proposicional

$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$	Modus Ponens	Da implicação e da premissa infere-se a conclusão
$\frac{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}{\alpha_n}$	Eliminação	Da conjunção infere-se qualquer α_n
$\frac{\alpha_1, \alpha_2, \dots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}$	Introdução da conjunção	De uma lista de sentenças infere-se a sua conjunção
$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \dots \vee \alpha_n}$	Introdução da disjunção	De uma sentença, infere-se sua disjunção com qualquer outra
$\frac{\neg \neg \alpha}{\alpha}$	Negação dupla	De uma negação dupla infere-se uma sentença positiva
$\frac{\alpha \vee \beta, \neg \beta}{\alpha}$	Resolução simples	Se uma das disjunções for falsa, pode-se inferir que a outra é verdade
$\frac{\alpha \vee \beta, \neg \beta \vee \gamma}{\alpha \vee \gamma}$	Resolução	β , não pode ser Verdade e Falso ao mesmo tempo

Regra de Inferência de Resolução Unitária

$$\frac{l_1 \vee \dots \vee l_k, m}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \dots \vee l_k}$$

onde l_i e m são literais complementares (, isto é, l é a negação de m).

Regra de Inferência de Resolução Completa

$$l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_k$$

$$l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \dots \vee l_k \vee m_1 \vee \dots \vee m_{i-1} \vee m_{i+1} \dots m_k$$

onde l_i e m_i são literais complementares.

Em particular, para três literais temos:

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Algoritmo de Resolução

Lembrando da lógica proposicional, o algoritmo de Resolução usa o princípio de prova por contradição:

– para provar que $KB \models \alpha$, ou seja, que α é consequência lógica de KB ($KB \Rightarrow \alpha$), provamos que a negação de ($KB \Rightarrow \alpha$) ou seja $(\neg (\neg KB \vee \alpha)) = (KB \wedge \neg \alpha)$ é não satisfatível (isto é, não existe um modelo que satisfaz a fórmula).

Algoritmo de Resolução

1. $(KB \wedge \neg \alpha)$ é convertida em uma FNC (F. Normal Conjuntiva)
2. Aplica a regra de resolução na FNC resultante Para cada par de cláusulas que contém literais complementares é resolvido para gerar uma nova cláusula que é inserida na KB, se ainda não estiver presente.
3. O passo 2 continua até que:
 - Não exista nenhuma cláusula nova que possa ser adicionada nesse caso $KB \not\models \alpha$ ou
 - Uma aplicação da regra de resolução deriva a cláusula vazia nesse caso $KB \models \alpha$. (uma cláusula vazia é gerada resolvendo-se duas cláusulas unitárias, P e $\neg P$, o que representa uma contradição)

Exemplo de aplicação do Algoritmo de Resolução

$\{p \rightarrow q, \neg p \rightarrow r, q \rightarrow s, \neg s\} \models r$

(1)	$\neg p \vee q$	Δ
(2)	$p \vee r$	Δ
(3)	$\neg q \vee s$	Δ
(4)	$\neg s$	Δ
(5)	$\neg r$	<i>hipótese</i>
(6)	p	$RES(2, 5)$
(7)	q	$RES(1, 6)$
(8)	s	$RES(3, 7)$
(9)	$\square$	$RES(4, 8)$

Inferência na lógica de predicados

Consideremos o argumento:

$\text{homem}(\text{socrates}), \forall X[\text{homem}(X) \rightarrow \text{mortal}(X)] \models \text{mortal}(\text{socrates})$

Normalizando essas fórmulas, obtemos:

$\text{homem}(\text{socrates}), \neg \text{homem}(X) \vee \text{mortal}(X) \models \text{mortal}(\text{socrates}).$

Como a variável X é universal, podemos substituí-la por qualquer constante do domínio ($X = \text{socrates}$), assim obtemos uma nova instância da fórmula $\neg \text{homem}(X) \vee \text{mortal}(X)$ e, assim temos:

Inferência na lógica de predicados

(1)	$homem(socrates)$	Δ
(2)	$\neg homem(socrates) \vee mortal(socrates)$	$\Delta/X=socrates$
(3)	$mortal(socrates)$	$RES(1, 2)$

Outro Exemplo

- (a) “Todos os matriculados em Introdução Inteligência Artificial são estudantes dedicados” e
- (b) “Nicoly está matriculada em Introdução à IA” implicam a conclusão
- (c) “Nicoly é uma estudante dedicada”

Os predicados, tendo como domínio o conjunto de todos os estudantes:

$M(x)$: “x está matriculado em Introdução à IA”

$D(x)$: “x é um estudante dedicado.”

Exemplos

As premissas do argumento são, então:

(a) $\forall x : (M(x) \rightarrow D(x))$ (b) $M(\text{Nicoly})$

A conclusão do argumento é:

(c) $D(\text{Nicoly})$

Derivamos a conclusão a partir das premissas da seguinte forma

- | | | |
|----|---|-------------------------------|
| 1. | $\forall x : (M(x) \rightarrow D(x))$ | Premissa (a) |
| 2. | $M(\text{Nicoly}) \rightarrow D(\text{Nicoly})$ | Instanciação universal de (1) |
| 3. | $M(\text{Nicoly})$ | Premissa (b) |
| 4. | $D(\text{Nicoly})$ | Modus ponens usando (2) e (3) |

Instanciação Universal

O princípio de instanciação universal, que nos permite substituir uma variável por uma constante qualquer do seu domínio, só funciona corretamente para variáveis universais. Considere a sentença “Todo mestre tem um discípulo”:

$$\forall X[\text{mestre}(X) \rightarrow \exists Y [\text{discipulo}(Y, X)]]$$

Sendo X uma variável universal, podemos substituí-la por qualquer constante e a sentença obtida continuará sendo verdadeira. Por exemplo $X = \text{Platão}$ e obter a seguinte instância:

$$\text{mestre}(\text{Platão}) \rightarrow \exists Y [\text{discipulo}(Y, \text{Platão})].$$

Instanciação Existencial

Agora, substituindo a variável existencial, obtemos a instância: $\forall X[\text{mestre}(X) \rightarrow [\text{discipulo}(\text{Platão}, X)]$, que afirma é que “Todo mestre tem um discipulo chamado Platão”. O significado da sentença original foi perdido. Isso acontece porque o valor de Y depende do valor escolhido para X.

Skolemização.

Uma forma de eliminar uma variável existencial , sem alterar o significado da sentença original, é admitir a existência de uma função que representa o valor correto para substituir a variável existencial. Por exemplo, poderíamos substituir a variável existencial Y pela função $\text{seguidor}(X)$, veja:

$$\forall X[\text{mestre}(X) \rightarrow [\text{discipulo}(\text{seguidor}(X), X)]$$

Skolemização.

Skolemize a seguinte fórmula:

- Todo cão é fiel ao seu dono

$$\forall X \text{ cão}(X) \rightarrow \text{fiel}(X, \text{dono}(X))$$

Unificação

Processo de encontrar um conjunto minimal de substituições que torna duas fórmulas idênticas (a fim de que possamos usar resolução). Exemplos:
 $\text{Unifica}(\text{Conhece}(\text{João}, x), \text{Conhece}(x, \text{Maria})) = \text{falha},$
porque x não pode ser igual a João e Maria ao mesmo tempo

- $\text{Unifica}(\text{Conhece}(\text{João}, x_1), \text{Conhece}(x_2, \text{Maria})) = (x_1/\text{Maria}, x_2/\text{João})$
- $\text{Unifica}(\text{Conhece}(x_1, x_2), \text{Conhece}(\text{João}, \text{Maria})) = (x_1/\text{João}, x_2/\text{Maria})$

Algoritmo de Unificação

Para unificar duas fórmulas atômicas (sem variáveis em comum)

1. Compare as fórmulas até achar uma discrepância ou atingir o final de ambas.
2. Ao encontrar uma discrepância: Se nenhum dos elementos envolvidos for uma variável, finalize com fracasso. Caso contrário, substitua todas as ocorrências da variável pelo outro elemento e continue a comparação das fórmulas.
3. Ao atingir o final de ambas as fórmulas atômicas, finalize com sucesso.

Exemplo de Unificação

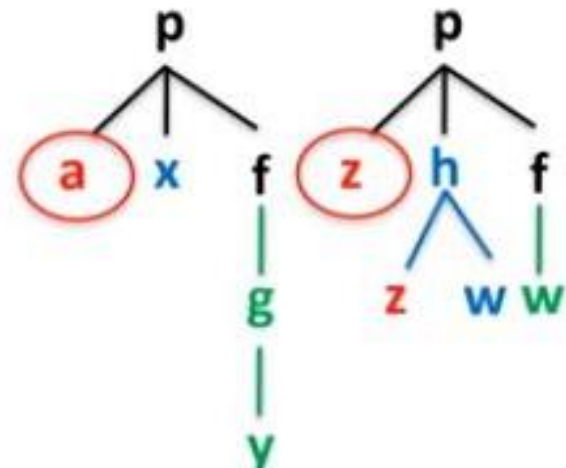
- Unifique (se possível) as fórmulas atômicas a seguir:

$$S = \{ p(a, x, f(g(y))), p(z, h(z, w), f(w)) \}$$

$$\{ p(\textcolor{red}{a}, x, f(g(y))), p(\textcolor{red}{z}, h(\textcolor{red}{z}, w), f(w)) \}$$



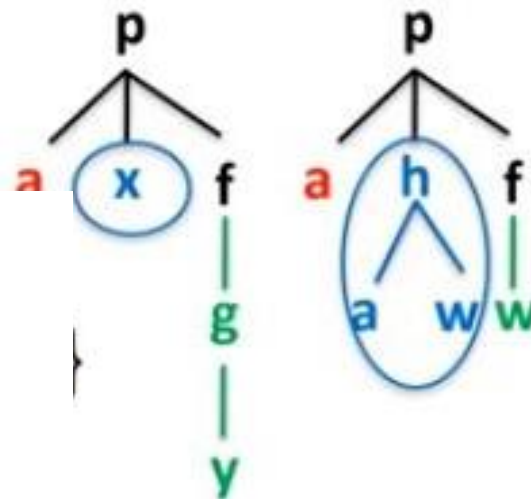
z/a



Exemplo de Unificação

$\{p(a, x, f(g(y))), p(a, h(a, w), f(w))\}$

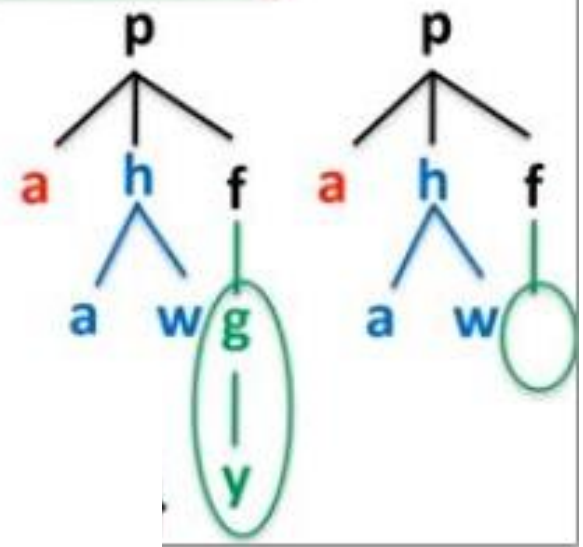
$z/a, x/h(a, w)$



Exemplo de Unificação

$\{p(a, h(a, w), f(\underline{g(y)})), p(a, h(a, w), f(\underline{w}))\}$

$z/a, x/h(z, w), w/g(y)$



Exemplo de Unificação

- Unifique (se possível) as fórmulas atômicas a seguir:
- { $\text{cor}(\text{sapato}(X); \text{branco})$ e $\text{cor}(\text{sapato}(\text{suspeito}); Y)$ }
- { $\text{mora}(X; \text{casa}(\text{mae}(X)))$ e $\text{mora}(\text{joana}; Y)$ }
- { $\text{primo}(X; Y)$ e $\text{prima}(A; B)$ }
- { $\text{ponto}(X; 2; Z)$ e $\text{ponto}(1; W)$ }

Forma Normal Conjuntiva para lógica de primeira ordem

- Todo mundo que ama todos os animais é amado por alguém

$$\forall x [\forall y \text{ Animal}(y) \rightarrow \text{Ama}(x,y)] \rightarrow [\exists z \text{ Ama}(z,x)]$$

- Eliminamos implicações

$$\forall x \neg[\forall y \text{ Animal}(y) \rightarrow \text{Ama}(x,y)] \vee [\exists z \text{ Ama}(z,x)]$$

$$\forall x \neg[\forall y \neg\text{Animal}(y) \vee \text{Ama}(x,y)] \vee [\exists z \text{ Ama}(z,x)]$$

- Movemos $\neg$ para dentro

$$\forall x [\exists y \neg(\neg\text{Animal}(y) \vee \text{Ama}(x,y))] \vee [\exists z \text{ Ama}(z,x)]$$

$$\forall x [\exists y \neg\neg\text{Animal}(y) \wedge \neg\text{Ama}(x,y)] \vee [\exists z \text{ Ama}(z,x)]$$

$$\forall x [\exists y \text{ Animal}(y) \wedge \neg\text{Ama}(x,y)] \vee [\exists z \text{ Ama}(z,x)]$$

Forma Normal Conjuntiva para lógica de primeira ordem

- Skolemização

$$\forall x [\text{Animal}(A) \wedge \neg \text{Ama}(x, A)] \vee [\text{Ama}(B, x)]$$

Remover quantificadores existenciais neste caso dará significado completamente errado, pois afirmaríamos que todo mundo deixa de amar um animal A específico ou é amado por alguma entidade específica B.

Originalmente afirmamos que cada pessoa deixa de amar um animal diferente ou é amada por uma pessoa diferente. Skolemizamos (variáveis que dependem de x)

$$\forall x [\text{Animal}(F(x)) \wedge \neg \text{Ama}(x, F(x))] \vee [\text{Ama}(G(x), x)]$$

Forma Normal Conjuntiva para lógica de primeira ordem

- Descartar quantificadores universais

$$[\text{Animal}(F(x)) \wedge \neg \text{Ama}(x, F(x)))] \vee [\text{Ama}(G(x), x)]$$

- Distribuir $\vee$ sobre $\wedge$

$$[\text{Animal}(F(x)) \vee \text{Ama}(G(x), x)] \wedge [\neg \text{Ama}(x, F(x)) \vee \text{Ama}(G(x), x)]$$

Agora a sentença está em FNC e consiste em duas cláusulas. A função de Skolem $F(x)$ se refere ao animal que é potencialmente não amado por x , enquanto $G(z)$ se refere a alguém que pode amar x .

Exemplo de Aplicação de algoritmo de resolução com lógica de predicados

- O problema é descrito a seguir em linguagem natural.
- Todo mundo que ama todos os animais é amado por alguém.
- Qualquer um que mata um animal não é amado por ninguém.
- João ama todos os animais.
- João ou a Curiosidade matou o gato, que se chama Atum.
- A Curiosidade matou o gato?

Exemplo de Aplicação de algoritmo de resolução com lógica de predicados

- A. $\forall x [\forall y \text{ Animal}(y) \rightarrow \text{Ama}(x,y)] \rightarrow [\exists z \text{ Ama}(z,x)]$
- B. $\forall x [\exists z \text{ Animal}(z) \wedge \text{Mata}(x,z)] \rightarrow [\forall y \neg \text{Ama}(y,x)]$
- C. $\forall x \text{ Animal}(x) \rightarrow \text{Ama}(\text{João},x)$
- D. $\text{Mata}(\text{João}, \text{Atum}) \vee \text{Mata}(\text{Curiosidade}, \text{Atum})$
- E. $\text{Gato}(\text{Atum})$
- F. $\forall x \text{ Gato}(x) \rightarrow \text{Animal}(x)$
- G. $\neg \text{Mata}(\text{Curiosidade}, \text{Atum})$

Convertemos cada sentença na FNC

Exemplo de Aplicação de algoritmo de resolução com lógica de predicados

A1. $\text{Animal}(F(x)) \vee \text{Ama}(G(x), x)$

A2. $\neg \text{Ama}(x, F(x)) \vee \text{Ama}(G(x), x)$

B. $\neg \text{Ama}(y, x) \vee \neg \text{Animal}(z) \vee \neg \text{Mata}(x, z)$

C. $\neg \text{Animal}(x) \vee \text{Ama}(\text{João}, x)$

D. $\text{Mata}(\text{João}, \text{Atum}) \vee \text{Mata}(\text{Curiosidade}, \text{Atum})$

E. $\text{Gato}(\text{Atum})$

F. $\neg \text{Gato}(x) \vee \text{Animal}(x)$

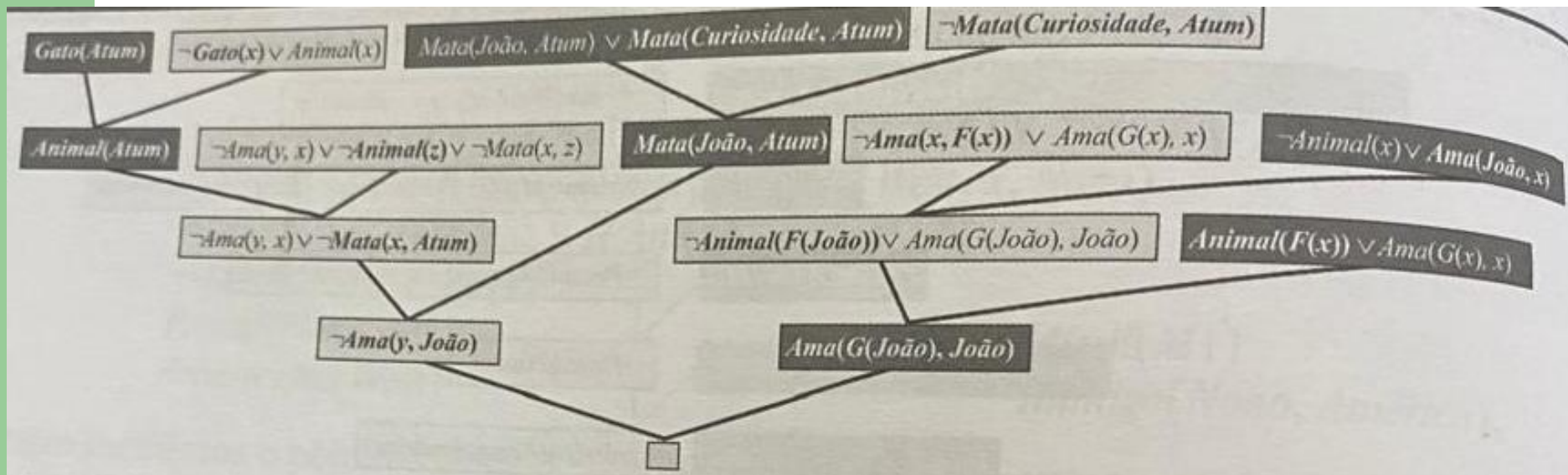
G. $\neg \text{Mata}(\text{Curiosidade}, \text{Atum})$

Exemplo de Aplicação de algoritmo de resolução com lógica de predicados

Em linguagem natural podemos parafrasear da seguinte forma:

Suponha que a Curiosidade não houvesse matado Atum. Sabemos que João ou a Curiosidade o matou; desse modo, João deve ter matado Atum. Agora, Atum é um gato, e gatos são animais, então Atum é um animal. Tendo em vista que qualquer um que mata um animal não é amado por ninguém, sabemos que ninguém ama João. Por outro lado, João ama todos os animais, então alguém o ama; assim, temos a contradição. Portanto, a Curiosidade matou o gato.

Exemplo de Aplicação de algoritmo de resolução com lógica de predicados



Raciocínio Automatizado

Algoritmo de Resolução-SLD

É um algoritmo de inferência usado em linguagens de programação lógica como Prolog.

Na resolução SLD, o objetivo é provar que uma determinada consulta (uma afirmação lógica) decorre de um conjunto de fatos (cláusulas). .

Raciocínio Automatizado

Algoritmo de Resolução-SLD

1. Restringe-se à uma classe de fórmulas denominada cláusulas de Horn.

Cláusulas de Horn são fórmulas da forma

$$p \Leftarrow q_1 \wedge q_2 \wedge \dots \wedge q_n$$

2. Emprega resolução e unificação como regra de inferência.
3. Adota uma estratégia de busca em profundidade para controlar as inferências.

Algoritmo de Resolução SLD

- Este algoritmo controla a aplicação da regra de inferência, realizando uma busca em profundidade.
- Ao derivar a cláusula vazia, o algoritmo sinaliza sucesso e apresenta como solução a composição das substituições efetuadas no caminho percorrido até a cláusula vazia.
- Ao atingir um ponto onde a consulta não pode ser unificada com nenhuma cláusula, o algoritmo sinaliza fracasso e tenta retroceder na busca (backtracking)

BackTracking

Seja a base de dados família. façamos a ela a seguinte consulta:

?- pai(roberto,X),mãe(rossana,X)

O compilador Prolog tenta primeiro satisfazer o primeiro objetivo, isto é pai(roberto,X), desencadeando a seguinte execução *top-down*.

1. Encontra que Roberto é pai de amanda.
2. Instância X com “amanda”.
3. Tenta satisfazer o segundo objetivo com a variável X já instanciada, isto é `mãe(rossana,amanda)`.
4. Como não consegue satisfazer este objetivo, realiza um *backtracking*, isto é volta à consulta original ?- `pai(roberto,X),mãe(gabriela,X)` a partir do último fato casado (`pai(roberto,lúcia)`).
5. Encontra que Roberto é pai de gabriela.
6. Instância X com “gabriela”.
7. Encontra que `mãe(rossana,gabriela)`.
8. Foi bem sucedido, pelo que a resposta é: `X=gabriela`.

Mas como o compilador Prolog acha que Roberto é pai de amanda? fazendo refutação. Primeiro negamos `pai(roberto,X)` e tentamos refutar o conjunto de cláusulas de Horn que compõem o programa Prolog acrescido desta cláusula. Existem basicamente duas refutações

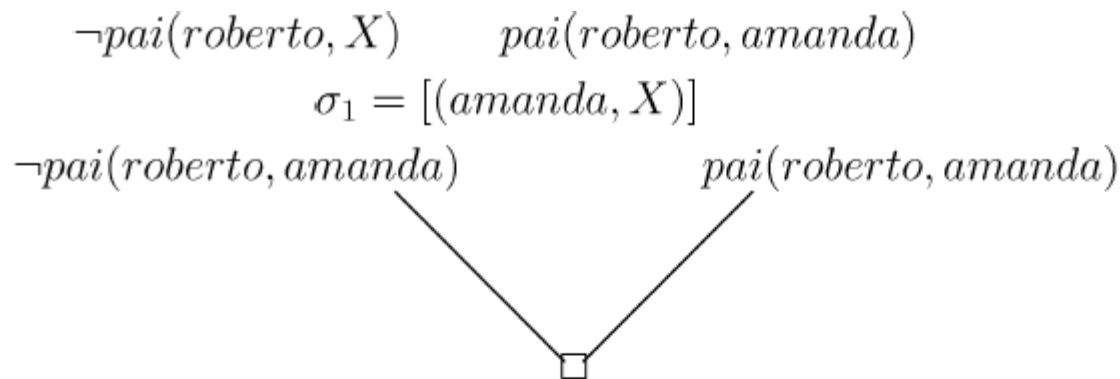


Figura 9.1: Árvore para a refutação da pergunta $?- \text{pai}(\text{roberto}, X)$.

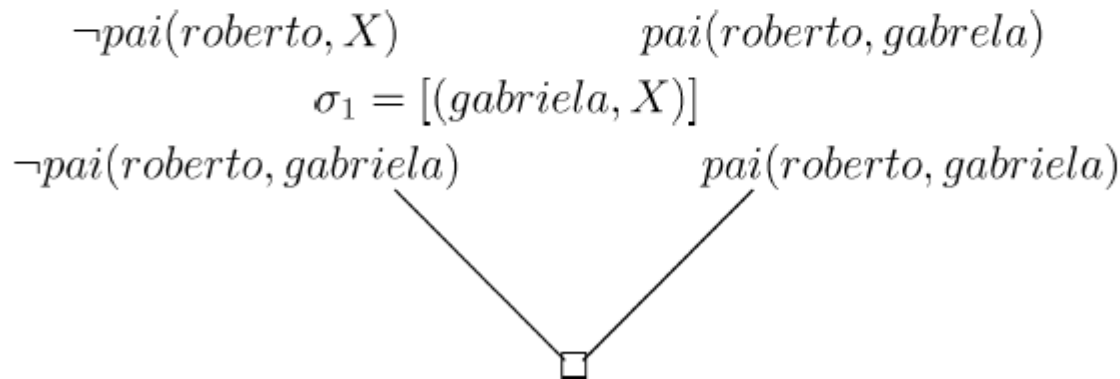


Figura 9.2: Segunda árvore para a refutação da pergunta $?- \text{pai}(\text{roberto}, X)$.

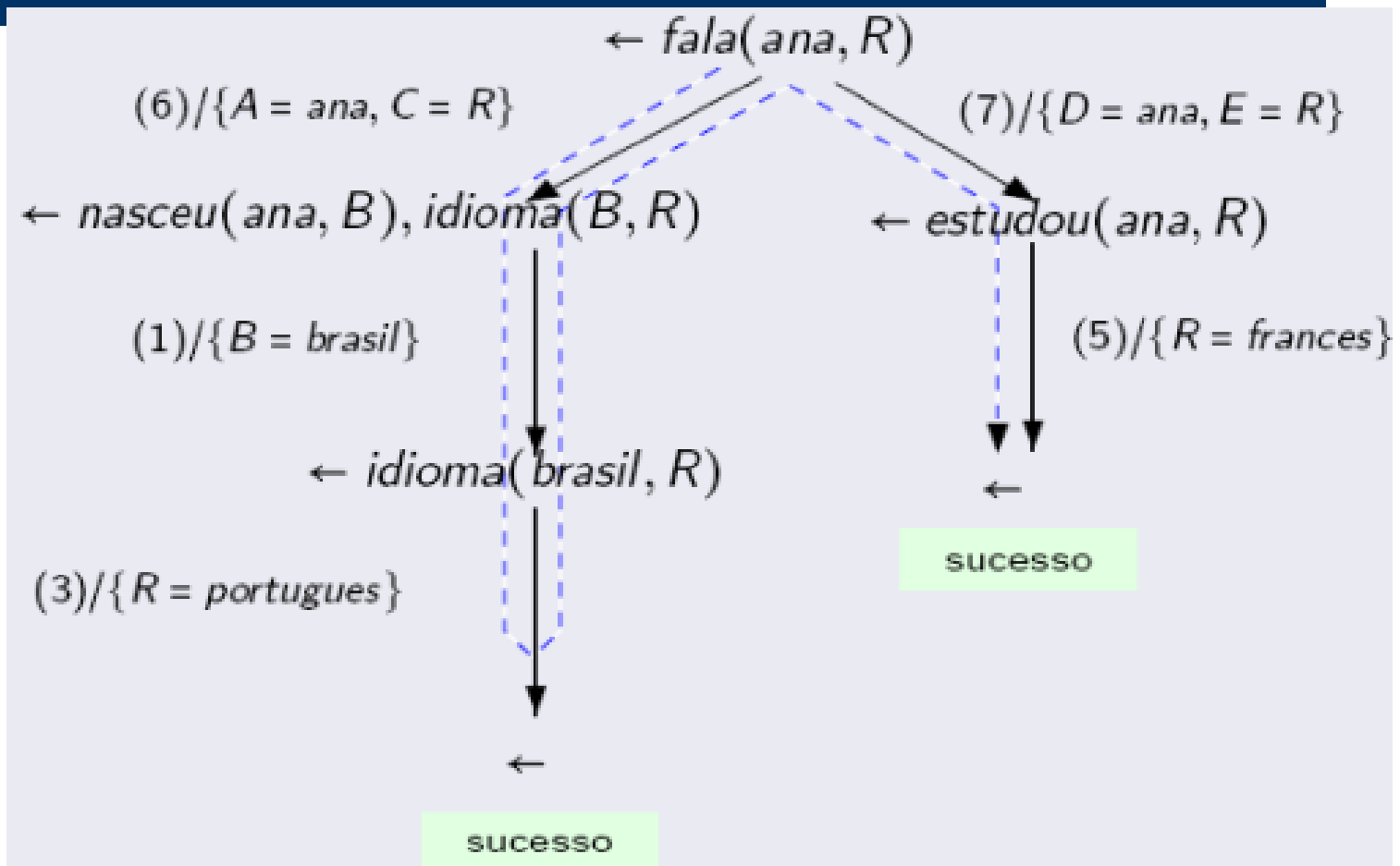
Consulta

Ana fala que idioma

- nasceu(ana, brasil)
- nasceu(marco, frança)
- idioma(brasil, portugues)
- idioma(frança; frances)
- estudou(ana; frances)
- fala(A,C) $\leftarrow$ nasceu(A,B); idioma(B,C)
- fala(D,E) $\leftarrow$ estudou(D,E)

Consulta

Ana fala que idioma



PROLOG

- O Prolog é uma linguagem de declarativa que usa um fragmento da lógica de 1ª ordem (as Cláusulas de Horn) para representar o conhecimento sobre um dado problema.
- Um programa em Prolog é um “conjunto” de axiomas e de regras de inferência (definindo relações entre objetos) que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.

PROLOG

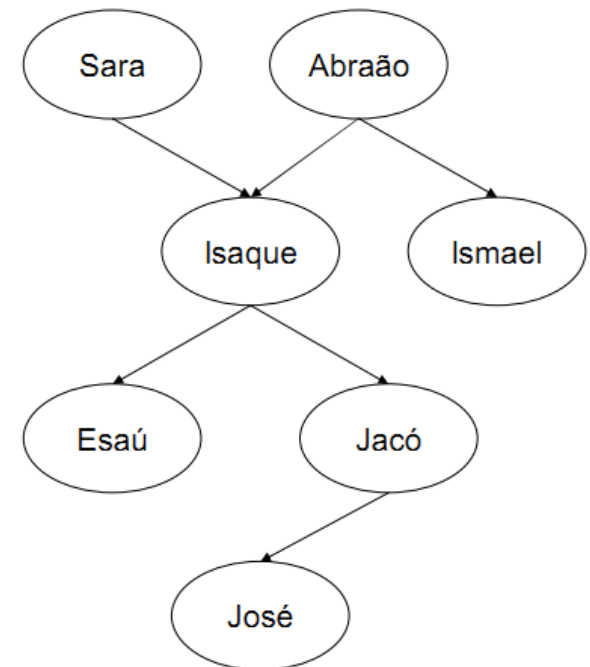
- A execução de um programa em Prolog consiste na dedução de consequências lógicas da base de conhecimento. O utilizador coloca questões e o “motor de inferência” do Prolog pesquisa a base de conhecimento à procura de axiomas e regras que permitam (por dedução lógica) dar uma resposta.
- O motor de inferência faz a dedução aplicando o algoritmo de resolução de 1ª ordem.

PROLOG

- **Programar em Prolog envolve:**
 - Declarar alguns **fatos** a respeito de objetos e seus relacionamentos.
 - Definir algumas **regras** sobre os objetos e seus relacionamentos.
 - Fazer **perguntas (consultas)** sobre os objetos e seus relacionamentos .

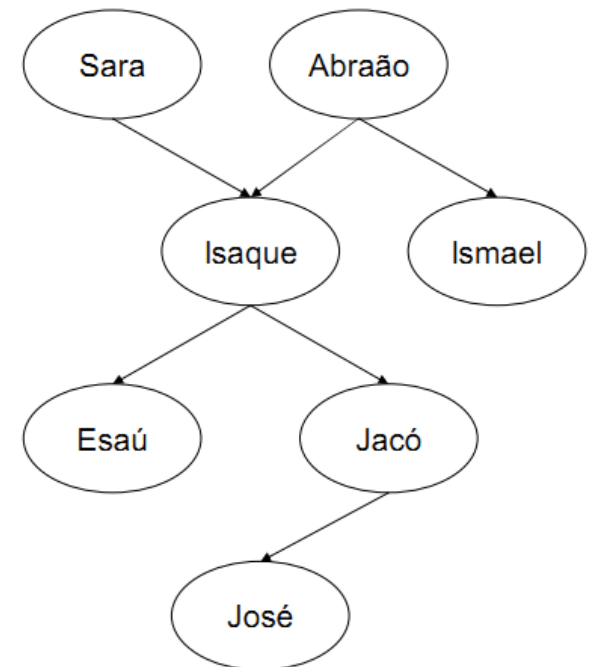
PROLOG

- **Fatos:** Um fato estabelece um relacionamento incondicional entre objetos de um contexto. Um fato é sempre verdadeiro (verdade incondicional).
 - progenitor(sara,isaque).
 - progenitor(abraão,isaque).
 - progenitor(abraão,ismael).
 - progenitor(isaque,esaú).
 - progenitor(isaque,jacó).
 - progenitor(jacó,josé).



PROLOG

- **Fatos:** Um fato estabelece um relacionamento incondicional entre objetos de um contexto. Um fato é sempre verdadeiro (verdade incondicional).
- mulher(sara)
- homem(abraão)
- homem(isaque)
- homem(esaú)
- homem(jacó)



PROLOG

- **Regras** especificam coisas que são verdadeiras se alguma condição é satisfeita.
 - mãe(X,Y) :- mulher(X), progenitor(X,Y).
 $\forall X \forall Y \text{ mãe}(A,B) \Leftarrow \text{mulher}(X) \wedge \text{progenitor}(X,Y)$
 - pai(X,Y) :- homem(X), progenitor(X,Y).
 $\forall A \forall B \text{ pai}(A,B) \Leftarrow \text{homem}(A) \wedge \text{progenitor}(A,B)$
 - avo(X,Y) :- progenitor(X,Z), progenitor(Z,Y).
 $\forall X \forall Y \text{ avo}(X,Y) \Leftarrow \exists Z. \text{progenitor}(X,Z) \wedge \text{progenitor}(Z,Y)$

PROLOG

- Cláusulas de Horn são fórmulas da forma

$$p \Leftarrow q_1 \wedge q_2 \wedge \dots \wedge q_n$$

representadas em Prolog por

$$p :- q_1, q_2, \dots, q_n$$

<cabeça de cláusula>:- <corpo de cláusula>

- Os fatos são cláusulas de Horn com o corpo vazio. As variáveis que aparecem nas cláusulas são quantificadas universalmente e o seu âmbito é toda a cláusula.

PROLOG

- Mas podemos ver as variáveis que ocorrem apenas no corpo da cláusula (mas não na cabeça), como sendo quantificadas existencialmente dentro do corpo da cláusula.
- As consultas são cláusulas de Horn com cabeça vazia e são um meio de extrair informação de um programa. As variáveis que ocorrem nas consultas são quantificadas existencialmente.

Cláusulas de Horn

- Cláusulas de Horn são fórmulas da forma $\phi \leftarrow \phi_1, \dots, \phi_n$, sendo $n \geq 0$, onde ϕ é uma conclusão e $\phi_1, \dots, \phi_n$ são premissas (condições).

- Tipos de cláusulas:

Fato.....: $\phi \leftarrow$

Regra.....: $\phi \leftarrow \phi_1, \dots, \phi_n$

Consulta.....: $\leftarrow \phi_1, \dots, \phi_n$

Contradição.....: $\leftarrow$ (Cláusula vazia)

PROLOG

- Responder a uma consulta é determinar se a consulta é uma consequência lógica do programa. Responder a uma consulta com variáveis é dar uma instânciação da consulta (representada por uma substituição para as variáveis) que é inferível do programa.

PROLOG

- **Consultas:** Pode-se questionar o Prolog sobre a relação progenitor, por exemplo: **Isaque é o pai de Jacó?**

?- progenitor(abraão,ismael).

Ao que Prolog responderá True

?- progenitor(abraão,moisés).

A resposta será false

PROLOG

- Perguntas mais interessantes também podem ser efetuadas:
- **Quem é o progenitor de Ismael?**
- ?- progenitor(X,josé).

A resposta será

X = jacó

PROLOG

- Ao digitar “;” o motor de inferência faz backtracking. Ou seja, construi-se uma outra prova (alternativa) para a consulta que é colocada. Essa nova prova pode dar origem a uma outra substituição das variáveis

PROLOG

- A pergunta “**Quais os filhos de abraão?**” pode ser escrita como:
?- progenitor(abraão,X).
- Neste caso, há **mais de uma resposta possível**.
O Prolog primeiro responde com uma solução:
–X = isaque
- Pode-se requisitar uma **outra solução** (digitando ;) e o Prolog a encontra:
–X = ismael

PROLOG

- Perguntas mais complexas também podem ser efetuadas, tais como: **Quem é o avô de Esaú?**
- Esta pergunta composta pode ser escrita em Prolog como:
- ?- progenitor(Y,esaú), progenitor(X,Y).
- X = abraão
- Y = isaque

PROLOG

- Outras regras
- filho(Y,X) :- progenitor(X,Y).
- irmão(X,Y) :- progenitor(Z,X), progenitor(Z,Y).

PROLOG

- Regras Recursivas
- Para criar uma relação **ancestral** é necessária a criação de uma regra recursiva:

`ancestral(X,Z) :- progenitor(X,Z).`

`ancestral(X,Z) :- progenitor(X,Y), ancestral(Y,Z).`