

CC7711

# Inteligência Artificial e Robótica

Prof. Dr. Flavio Tonidandel



The background of the slide features a complex network diagram. It consists of numerous circular nodes of varying sizes, connected by thin, light blue lines. The nodes are distributed across the entire frame, with a higher density on the left side, creating a sense of depth and connectivity. The overall color palette is light blue and white.

# Representação do Conhecimento

**Logica de Primeira Ordem (LPO)**

# LP x LPO – Linguagem de Primeira ordem

A lógica proposicional (LP) assume mundos compostos somente por  **fatos**.

- Isto a torna uma linguagem com poder de expressão limitado do mundo real

A lógica de primeira ordem – LPO, assim como a linguagem natural, assume mundos compostos por

- Objetos, Relações e Funções
- Ela é suficientemente expressiva para representar nosso conhecimento comum



# Lógica de Primeira Ordem (LPO)

Além dos objetos e fatos da LP, também possui:

- Variáveis
- Relações:
  - Unárias: propriedades de um objeto. Ex: vermelho, redondo, falso, Jogador (Kaká)
  - *n-árias: relacionam grupos de objetos. Ex: irmão de, maior que, interior a, parte de... pai (Fulano, Ciclano)*
- Funções:
  - um objeto está relacionado a exatamente um objeto. Ex: pai de, melhor amigo de, uma unidade maior que...Sqrt (25) ou mesmo PernaDireitaDe(Beltrano)
- Linguagem da lógica de primeira ordem é elaborada em torno de objetos e relações

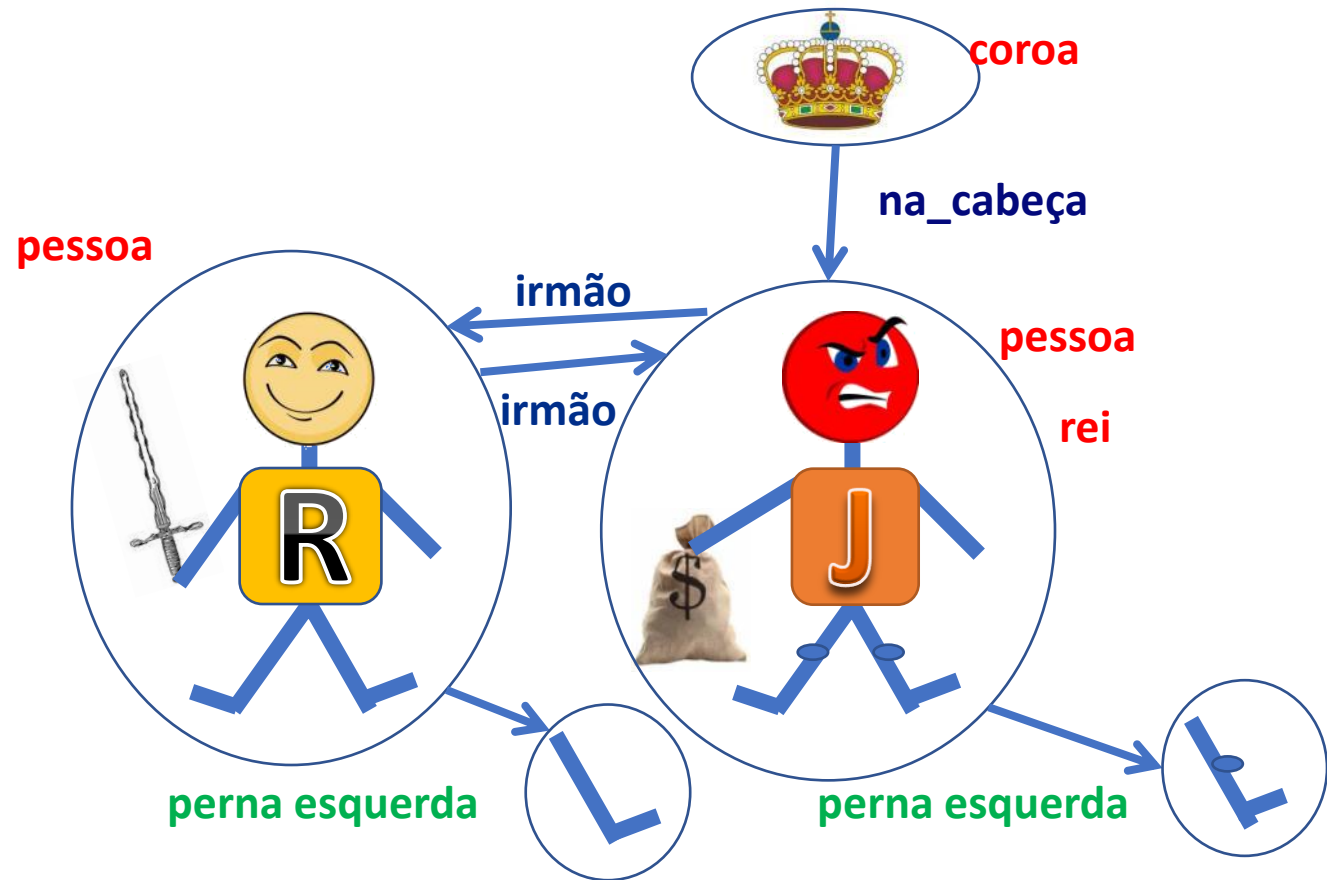
# Exemplo de um Modelo em LPO

Ricardo Coração de Leão, rei da Inglaterra de 1189 a 1199 e seu irmão mais jovem, o perverso rei João, que governou de 1199 a 1215

**Relações unárias  
(propriedades)**

**Relações binárias  
(pares de objetos)**

**Funções  
(relacionado a um objeto)**



# Símbolos

Começam com letras maiúsculas. Podem ser de três tipos:

- Símbolos de **constantes**: representam objetos
  - Exemplo: *Ricardo e João*
- Símbolos de **predicados**: representam relações
  - Exemplo: *Irmão, NaCabeça, Pessoa*
- Símbolos de **funções**: representam funções
  - Exemplo: *PernaEsquerda*

# Relações

**Relação:** conjunto de tuplas de objetos inter-relacionados

- Tuplas: objetos agrupados em uma certa ordem

“Ricardo e João são irmãos”

Irmão(Ricardo, João)

*Ordem de leitura.... Ricardo é irmão de João*

- “A coroa está na cabeça do rei João”
  - Na\_cabeça(Coroa,João)
  - A relação “na cabeça” contém a tupla: <coroa, João>
- Perna esquerda: cada pessoa tem uma perna esquerda, então o modelo tem uma função unária “perna esquerda” que inclui os
- seguintes mapeamentos: <Ricardo Coração de Leão> → perna esquerda de Ricardo ; < Rei João > → perna esquerda de João



# Funções

- Uma função é uma relação binária tal que não há dois membros distintos para um único primeiro elemento. Em outras palavras, se  $F$  é uma função:  $\langle x, y \rangle \in F$  e  $\langle x, z \rangle \in F \Rightarrow y = z$
- Se  $\langle x, y \rangle \in F$ , então:
  - $x$  é um argumento de  $F$ ;  $y$  é o valor de  $F$  para o argumento  $x$ ;
  - Logo  $y$  é a imagem de  $x$  para  $F$ .
- $F(x)$  designa o objeto  $y$  tal que  $y = F(x)$ .
- Exemplo:  $Mae(Andrea, Carlos)$ . Não existe outra Mãe para Carlos. Assim, Mãe pode virar uma função:  $Mae(Carlos) = Andrea$
- Qual a vantagem?
  - Posso usar a função como termo. Um predicado não.
  - $Adulto(Mae(Eduardo)) \equiv Adulto(Silvia)$



# Quantificadores

Expressam propriedades de grupos de objetos

**Quantificador universal ( $\forall$ ):** “Para todo...”

$\forall x P$ , onde  $P$  é qualquer expressão lógica, afirma que  $P$  é verdadeira para todo objeto  $x$ . Exemplo:

$\forall x \text{ Rei}(x) \Rightarrow \text{Pessoa}(x)$

Todo Rei é uma pessoa

**Quantificador existencial ( $\exists$ ):** “Para algum...”

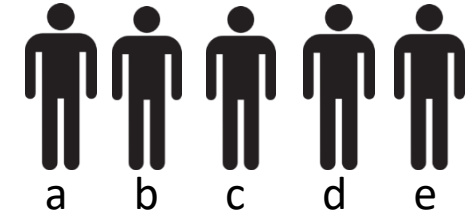
$\exists x P$  afirma que  $P$  é verdadeira para pelo menos um  $x$ .  
Exemplo:

$\exists x \text{ Coroa}(x) \wedge \text{NaCabeça}(x, \text{João})$

Existe uma coroa que está na cabeça de João

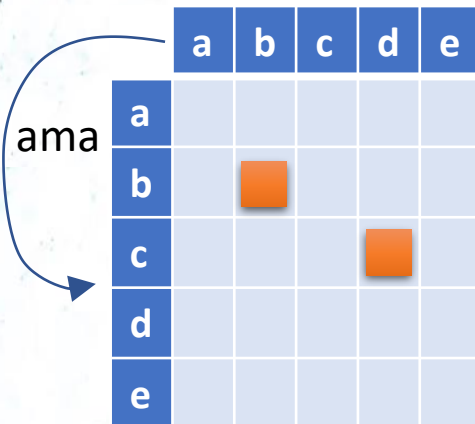
# Entendendo os Quantificadores

Consideraremos 5 pessoas:



$$\exists x \exists y \text{ Ama}(x,y)$$

Alguém ama alguém

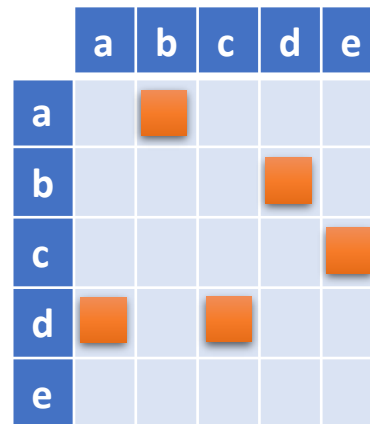


	a	b	c	d	e
a		■	■		
b			■		
c				■	
d					■
e					

**Matriz não é vazia  
nem cheia**

$$\forall x \exists y \text{ Ama}(x,y)$$

Todos amam alguém

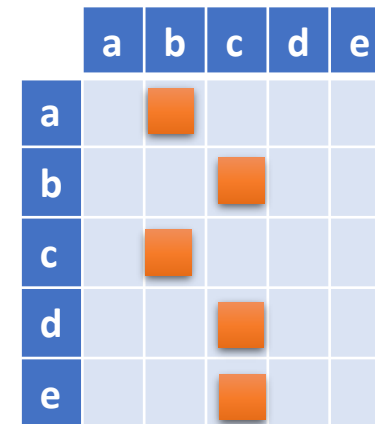


	a	b	c	d	e
a		■			
b				■	
c					■
d	■		■		
e					

**Nenhuma coluna vazia**

$$\forall x \exists y \text{ Ama}(y,x)$$

Todos são amados p/ alguém

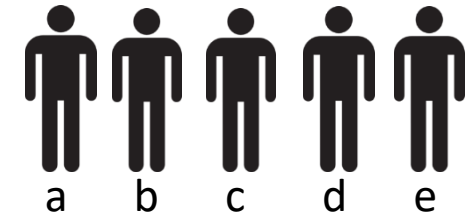


	a	b	c	d	e
a		■			
b			■		
c	■				
d			■		
e			■		

**Nenhuma linha vazia**

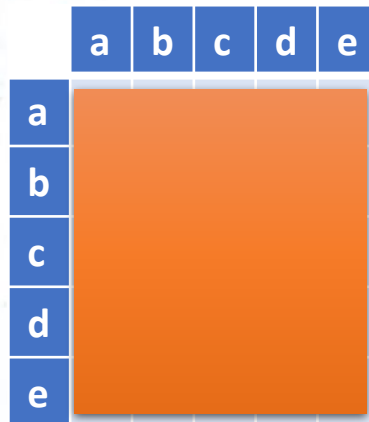
# Entendendo os Quantificadores

Consideraremos 5 pessoas:



$$\forall x \forall y \text{ Ama}(x,y)$$

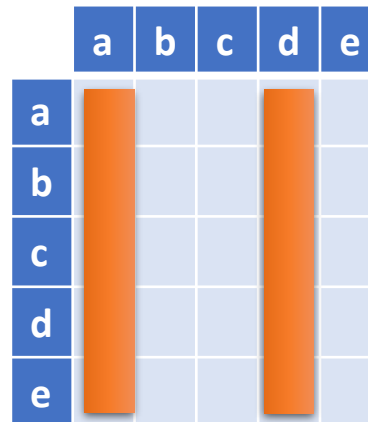
todos amam todos



Matriz cheia

$$\exists x \forall y \text{ Ama}(x,y)$$

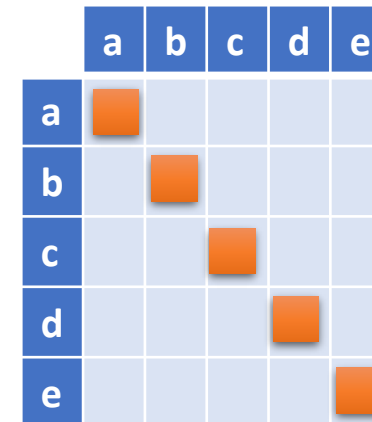
Alguém ama todos



Algumas colunas inteiras

$$\forall x \text{ Ama}(x,x)$$

Todos amam a si mesmos



Diagonal cheia



# Alguns propriedades dos quantificadores

- $\forall x \forall y$  é o mesmo que  $\forall y \forall x$  : pode ser expresso como  $\forall x, y$
- $\exists x \exists y$  é o mesmo que  $\exists y \exists x$  : também pode ser expresso  $\exists x, y$
- $\exists x \forall y$  **não** é o mesmo que  $\forall y \exists x$  sempre:
  - $\exists x \forall y \text{ Ama}(x,y)$ 
    - alguém ama todo mundo
  - $\forall y \exists x \text{ Ama}(x,y)$ 
    - Todo mundo é amado por alguém

Isto ocorre porque  $\text{Ama}(x,y)$  não é simétrico à  $\text{Ama}(y,x)$ .

# Uso dos quantificadores

- Todo irmão é parente

$$\forall x \forall y \text{ Irmão}(x,y) \rightarrow \text{Parente}(x,y)$$

- Todo mundo ama alguém

$$\forall x \exists y \text{ Ama}(x,y)$$

- “irmão” é simétrico

$$\forall x,y \text{ irmão}(x,y) \Leftrightarrow \text{irmão}(y,x)$$

- Avô é pai de alguém

$$\forall x,y \text{ Avô}(x,y) \Leftrightarrow \exists p \text{ Pai}(p,x) \wedge \text{Pai}(p,y)$$

$$\forall x,y \text{ Avô}(x,y) \Leftrightarrow \exists p \text{ Pai}(x,p) \wedge \text{Pai}(p,y)$$

*Cuidado ! A ordem dos argumentos devem respeitar a interpretação dada ao predicado  $\text{Pai}(x,y)$  significa “x é pai de y”*

# Equívocos comuns

- Tipicamente:
  - $\Rightarrow$  é o principal conectivo para ser usado com  $\forall$
  - $\wedge$  é o principal conectivo para  $\exists$
- *Equívocos:*
- *usar  $\wedge$  como o principal conectivo com  $\forall$ :*
  - $\forall x \text{ Rei}(x) \wedge \text{Pessoa}(x)$ , i.e. “Todos mundo é rei e pessoa”
- *usar  $\Rightarrow$  como o principal conectivo com  $\exists$ :*
  - $\exists x \text{ Coroa}(x) \Rightarrow \text{NaCabeça}(x, \text{João})$
  - Existir uma coroa implica que ela está na cabeça de João
  - i.e., não pode existir outra coroa na cabeça de João. E inclusive a sentença seria também verdadeira se x não for coroa e estiver na cabeça de João 😞



# Relação entre os quantificadores $\forall$ e $\exists$

- Estão intimamente conectados um ao outro por meio da negação.

Exemplos:

- $\forall x \neg \text{Gosta}(x, \text{Cenouras}) \equiv \neg \exists x \text{Gosta}(x, \text{Cenouras})$ 
  - “todo mundo detesta cenouras”  $\equiv$  “não existe alguém que goste de cenouras”
- $\forall x \text{Gosta}(x, \text{Sorvete}) \equiv \neg \exists x \neg \text{Gosta}(x, \text{Sorvete})$ 
  - “todo mundo gosta de sorvete”  $\equiv$  “não existe alguém que não goste de sorvete”

- Obedecem à regra de de Morgan:

$$\forall x \neg P \equiv \neg \exists x P$$

$$\neg \forall x P \equiv \exists x \neg P$$

$$\forall x P \equiv \neg \exists x \neg P$$

$$\exists x P \equiv \neg \forall x \neg P$$

# Igualdade

- Em LPO, pode-se usar o símbolo de igualdade para fazer declarações afirmando que dois termos se referem ao mesmo objeto
- $term_1 = term_2$  é verdade em uma interpretação se e somente se  $term_1$  e  $term_2$  referem ao mesmo objeto.

- E.g., definição de *Irmão*:

$$\exists h \forall x, y \text{ Pai}(h, x) \wedge \text{Pai}(h, y) \Rightarrow \text{Irmão}(x, y)$$

*se ficar assim, Irmão(Fulano, Fulano) é verdadeiro. O certo seria:*

$$\exists h \forall x, y \text{ Pai}(h, x) \wedge \text{Pai}(h, y) \wedge \neg(x = y) \Rightarrow \text{Irmão}(x, y)$$

\* e.g. É uma abreviatura em Latim de *exempli gratia* que significa “por exemplo”.

## Exercício 8.6 Livro Russel e Norvig

- Alguns alunos tiveram aulas de francês em 2010
- Todos os alunos que tiveram aulas de francês passaram
- Somente um aluno teve aula de grego no ano de 2010
- A melhor nota em grego é sempre mais alta que a melhor nota em francês



# Inferência direta em LPO

Uma forma de inferência é transformar LPO em LP !

Quantificando as variáveis e tornando todas as fórmulas “ground” (sem variáveis).

Mas isso pode ser um problemaço !

Como posso fazer inferência em LPO sem transformá-la em LP ?



- **Precisamos:**
  - Uma nova regra Modus Ponens
  - Algum processo p/ substituir variáveis por objetos

# Modus Ponens Generalizado (MPG)

- Seja a base:
  - $\forall x \text{ Rei}(x) \wedge \text{Ganancioso}(x) \Rightarrow \text{Malvado}(x)$
  - $\text{Rei}(\text{João})$
  - $\forall y \text{ Ganancioso}(y)$
- Regra geral para a inferência: encontre algum  $x$  que seja rei e ganancioso, e depois deduza que  $x$  é malvado
- Aplicando M.P.G.:
  - $p1'$  é  $\text{Rei}(\text{João})$   $p1$  é  $\text{Rei}(x)$
  - $p2'$  é  $\text{Ganancioso}(y)$   $p2$  é  $\text{Ganancioso}(x)$
  - $\theta$  é  $\{x/\text{João}, y/\text{João}\}$   $q$  é  $\text{Malvado}(x)$
  - $\text{SUBST}(\theta, q)$  é  $\text{Malvado}(\text{João})$
- Como determinar a substituição de variáveis ?
  - Pelo processo de Unificação !

# Unificação

Podemos realizar a inferência assim que nós encontrarmos uma substituição  $\theta$  para as variáveis.

p	q	$\theta$
Knows(John,x)	Knows(John,Jane)	$\{x/Jane\}$
Knows(John,x)	Knows(y,Maria)	$\{x/Maria, y/John\}$
Knows(John,x)	Knows(y,Mother(y))	$\{y/John, x/Mother(John)\}$
Knows(John,x)	Knows(x,Carlos)	$\{fail\}$

Para unificar  $Knows(John,x)$  e  $Knows(y,z)$ :

$\theta_1 = \{y/John, x/z\}$  ou  $\theta_2 = \{y/John, x/John, z/John\}$  O  $\theta_1$  é mais geral que o  $\theta_2$ .

Existe SEMPRE um único **Unificador Mais Geral (MGU)** capaz de renomear as variáveis:

**MGU =  $\{y/John, x/z\}$**



# Resolução para LPO

- Estender a Resolução da LP para a LPO
- Primeira coisa a fazer:
  - Transformar a BC para a forma CNF (forma normal conjuntiva)
  - Procedimento parecido ao da LP, o que difere é que temos que eliminar os quantificadores Universal ( $\forall$ ) e Existencial ( $\exists$ )
- Vamos verificar o método de conversão de uma sentença para CNF usando um exemplo:

**Todo mundo que ama os animais é amado por alguém**

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

Perceba que o y após a implicação é diferente do y antes da implicação, visto que cada um está dentro de um colchete.

# Transformando para CNF

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

- 1ª. Etapa: eliminar implicações ( $a \rightarrow b \equiv \neg a \vee b$ )
  - $\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Ama}(x,y)] \vee [\exists y \text{Ama}(y,x)]$
- 2ª. Etapa: Mover  $\neg$  para dentro:
  - Onde:  $\neg \forall x p \equiv \exists x \neg p$  e  $\neg \exists x p \equiv \forall x \neg p$
  - $\forall x [\neg \forall y \neg \text{Animal}(y) \vee \text{Ama}(x,y)] \vee [\exists y \text{Ama}(y,x)]$
  - $\forall x [\exists y \neg (\neg \text{Animal}(y) \vee \text{Ama}(x,y))] \vee [\exists y \text{Ama}(y,x)]$
  - $\forall x [\exists y \neg \neg \text{Animal}(y) \wedge \neg \text{Ama}(x,y)] \vee [\exists y \text{Ama}(y,x)]$
  - $\forall x [\exists y \text{Animal}(y) \wedge \neg \text{Ama}(x,y)] \vee [\exists y \text{Ama}(y,x)]$

# Transformando para CNF

- 3ª. Etapa: Padronizar variáveis (eliminar [ ])
  - $\forall x \exists y \text{ Animal}(y) \wedge \neg \text{Ama}(x,y) \vee \exists z \text{ Ama}(z,x)$
- 4ª. Etapa: “Skolemizar” p/ eliminar  $\exists$ 
  - Uma opção:  $\forall x \text{ Animal}(A) \wedge \neg \text{Ama}(x,A) \vee \text{Ama}(B,x)$ 
    - Mas a sentença mudaria de significado. Antes era algo como:
      - Todo mundo é Amado por alguém ou não gosta de Animais.
      - Todo mundo é Amado por B ou não gosta do Animal A.
    - Queremos que cada x não goste de algum animal (que será diferente para cada x, ou seja, o gosto é em função de x). O mesmo para ser amado.
    - Deste modo, o mais natural é definir uma **função de Skolem**:
$$\forall x \text{ Animal}(F(x)) \wedge \neg \text{Ama}(x,F(x)) \vee \text{Ama}(G(x),x)$$
    - Onde  $F(x)$  = animal não-amado por X e  $G(x)$  = alguém que ama x



# Transformando para CNF

- 5ª. Etapa: Descartar quantificadores Universais

Nesta etapa, sendo todas as variáveis universais, o quantificador pode ser eliminado.

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

- 6ª. Etapa: Distribuir  $\wedge$  sobre  $\vee$

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

Temos duas clausulas para a resolução resolver !!!

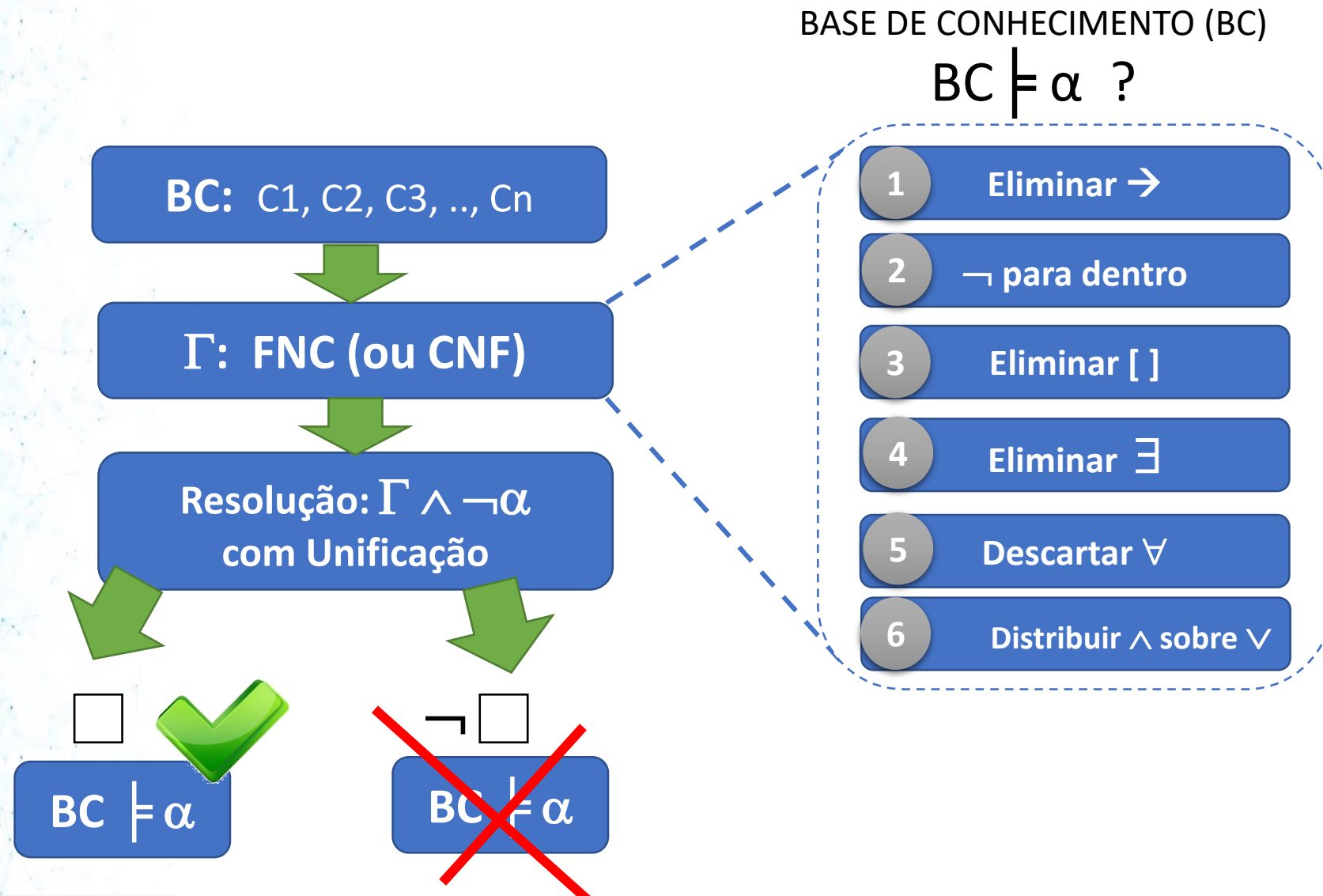
$$\boxed{\text{Animal}(F(x)) \vee \text{Ama}(G(x), x)} \wedge \boxed{\neg \text{Ama}(x, F(x)) \vee \text{Ama}(G(x), x)}$$

# Resolução

Unificar

$$\frac{[\text{Animal}(F(x)) \vee \text{Ama}(G(x), x)] \quad [\neg \text{Ama}(u, v) \vee \neg \text{Mata}(u, v)]}{[\text{Animal}(F(x)) \vee \neg \text{Mata}(G(x), x)] \quad \text{MGU} = \{u/G(x), v/x\}}$$

# Esquema Geral



# Exemplo Resolução em LPO

Exemplo retirado de: <https://www.javatpoint.com/ai-resolution-in-first-order-logic>



1. John likes all kind of food.
2. Apple and vegetable are food
3. Anything anyone eats and not killed is food.
4. Anil eats peanuts and still alive
5. Harry eats everything that Anil eats.

Duas fórmulas adicionadas



- a.  $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
- b.  $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c.  $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
- d.  $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$ .
- e.  $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
- f.  $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$
- g.  $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$ .

Para relacionar vivo (alive) e morto (killed)

Sem estas fórmulas o agente inteligente nunca irá saber que vivo é o contrário de morto.



# Exemplo Resolução em LPO

Exemplo retirado de: <https://www.javatpoint.com/ai-resolution-in-first-order-logic>



- a.  $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
- b.  $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- c.  $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
- d.  $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- e.  $\forall x: \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
- f.  $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$
- g.  $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$

FNC

Base de Conhecimento ( $\Gamma$ )

C1:  $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$

C2:  $\text{food}(\text{Apple})$

C3:  $\text{food}(\text{vegetables})$

C4:  $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$

C5:  $\text{eats}(\text{Anil}, \text{Peanuts})$

C6:  $\text{alive}(\text{Anil})$

C7:  $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$

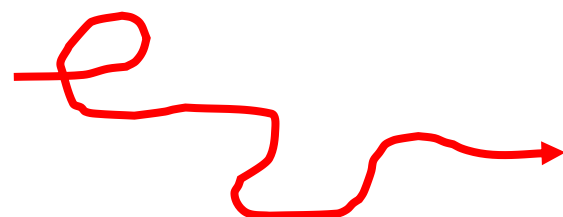
C8:  $\text{killed}(g) \vee \text{alive}(g)$

C9:  $\neg \text{alive}(k) \vee \neg \text{killed}(k)$

**C10:  $\neg \text{likes}(\text{John}, \text{Peanuts})$**

Pergunta:

$\Gamma \models \text{likes}(\text{John}, \text{Peanuts})$



# Exemplo Resolução em LPO

Exemplo retirado de: <https://www.javatpoint.com/ai-resolution-in-first-order-logic>



Base de Conhecimento ( $\Gamma$ )

C1:  $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$

C2:  $\text{food}(\text{Apple})$

C3:  $\text{food}(\text{vegetables})$

C4:  $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$

C5:  $\text{eats}(\text{Anil}, \text{Peanuts})$

C6:  $\text{alive}(\text{Anil})$

C7:  $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$

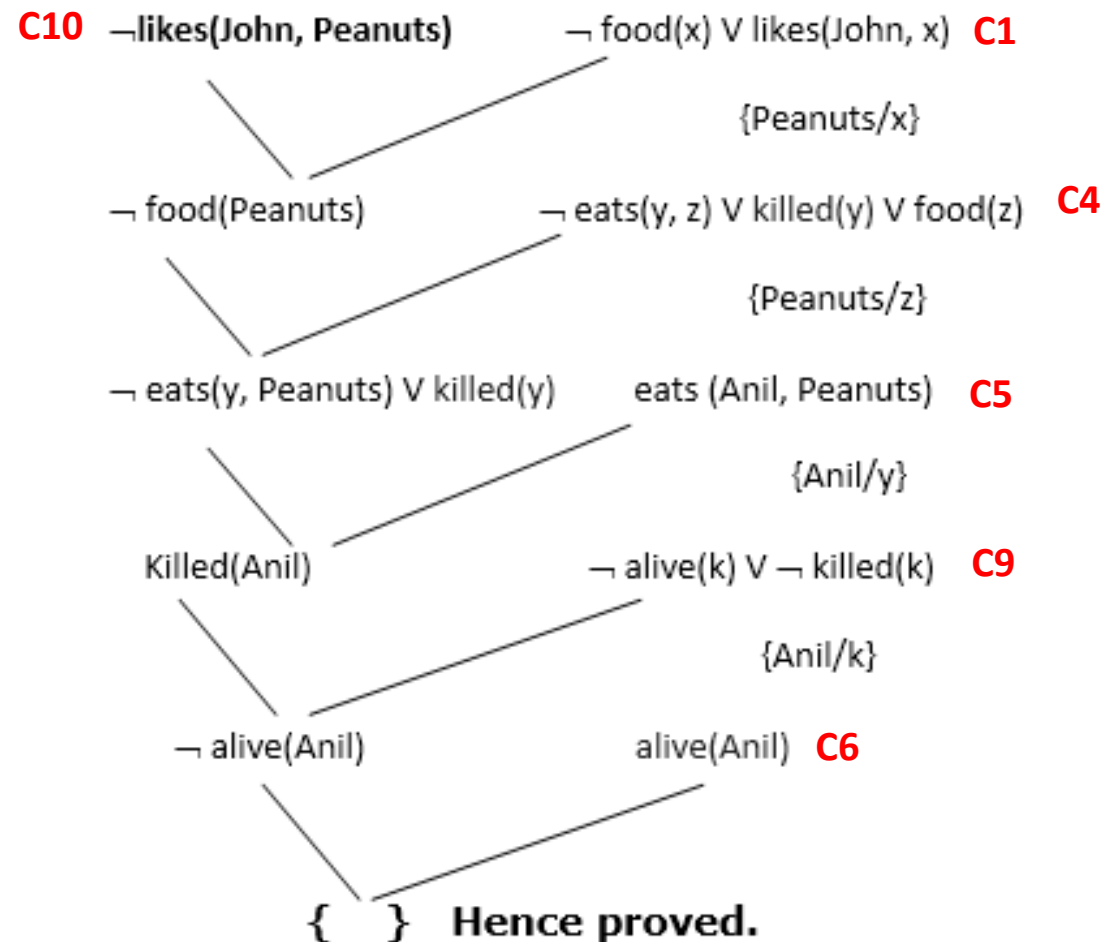
C8:  $\text{killed}(g) \vee \text{alive}(g)$

C9:  $\neg \text{alive}(k) \vee \neg \text{killed}(k)$

**C10:  $\neg \text{likes}(\text{John}, \text{Peanuts})$**

Pergunta:

$\Gamma \models \text{likes}(\text{John}, \text{Peanuts})$



# Clausulas de Horn

- Assim como na LP, em LPO se for possível definir as sentenças em Clausulas de Horn, podemos aplicar a Modus Ponens generalizada e inferir sentenças de modo mais rápido que a resolução.
- Método usado na **Programação em Lógica**
  - Algoritmo = Lógica + Controle
  - Base: backward chaining com Horn clauses + parâmetros de controle
  - Programa = conjunto de cláusulas
    - $\text{head} \text{ :- literal}_1, \dots \text{literal}_n.$
  - Busca em profundidade;
  - Hipótese de mundo fechado ("negation as failure")

# Funcionamento do PROLOG

- Seja a questão: ?- `sister(X,Y)`.
- Após localizada na BC a regra:  
`sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).`
- o PROLOG tenta satisfazê-la da esquerda para a direita e observando a ordem em que as cláusulas aparecem na BC
- Faz Unificação.
- Backward Chaining (começa da conclusão)





# SWI Prolog

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```





# SWI Prolog

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Hipótese:**

X = -

Y = -

Z = -



**SWI Prolog**

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=gaea**

**Hipótese:**

X = gaea

Y = -

Z = -



**SWI Prolog**

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (Z,gaea)**  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Z=chaos**

**X=gaea**

**Hipótese:**

X = gaea

Y = -

Z = chaos





**SWI Prolog**

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (chaos,Y)**

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Z=chaos**

**Y= gaea**

**X=gaea**

**Hipótese:**

X = gaea

Y = gaea

Z = chaos



SWI Prolog

Backtracking



BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Z=chaos**

**Y=gaea**

**Falha!**

**X=gaea**

**Hipótese:**

X = gaea

Y = gaea

Z = chaos



SWI Prolog

Backtracking



parent (chaos,Y)

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Z=chaos

X=gaea

Hipótese:

X = gaea

Y = -

Z = chaos

Y= Falha!





SWI Prolog

Backtracking



parent (Z,gaea)

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

X=gaea

Hipótese:

X = gaea

Y = -

Z = -

Z=Falha!





**SWI Prolog**

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=rhea**

**Hipótese:**

X = rhea

Y = -

Z = -



SWI Prolog

Backtracking



parent (Z,rhea)

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Falha!

X=rhea

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

Hipótese:

X = rhea

Y = -

Z = -



**SWI Prolog**

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=leto**

**Hipótese:**

X = leto

Y = -

Z = -





# SWI Prolog

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (Z,leto)**  
sister(X,Y) :- female(X), **parent(Z,X)**, parent(Z,Y), not(X=Y).

**Z=coeus**

**X=leto**

**Hipótese:**

X = leto

Y = -

Z = coeus





SWI Prolog

BC

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Z=coeus

Y=leto

X=leto

Hipótese:

X = leto

Y = leto

Z = coeus



SWI Prolog

BC

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Z=coeus

Y=leto

X=leto

Falha!

Backtracking

← ← ←

Hipótese:

X = leto

Y = leto

Z = coeus



SWI Prolog

Backtracking



parent (coeus,Y)

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Z=coeus

X=leto

Hipótese:

X = leto

Y = -

Z = coeus

Y= Falha!





SWI Prolog

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (Z,leto)**  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Z=phoebe**

**X=leto**

**Hipótese:**

X = leto

Y = -

Z = phoebe





SWI Prolog

Backtracking

← ← ←

parent (phoebe,Y)

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Z=phoebe

X=leto

Hipótese:

X = leto

Y = -

Z = phoebe

Y= Falha!



SWI Prolog

Backtracking



parent (Z,leto)

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

X=leto

Z = Falha !

Hipótese:

X = leto

Y = -

Z = -



**SWI Prolog**

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=hera**

**Hipótese:**

X = hera

Y = -

Z = -





**SWI Prolog**

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=phoebe**

**Hipótese:**

X = phoebe

Y = -

Z = -





**SWI Prolog**

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=tethys**

**Hipótese:**

X = tethys

Y = -

Z = -



**SWI Prolog**

**BC**

```
parent(chaos,gaea).  
parent(coeus,leto).  
parent(phoebe,leto).  
parent(leto,artemis).  
parent(leto,apollo).  
sister(X,Y) :- ...  
female(gaea).  
female(rhea).  
female(leto).  
female(hera).  
female(phoebe).  
female(tethys).  
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**X=artemis**

**Hipótese:**

X = artemis

Y = -

Z = -



SWI Prolog

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (Z,artemis)**  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

**Z=leto**

**X=artemis**

**Hipótese:**

X = artemis

Y = -

Z = leto



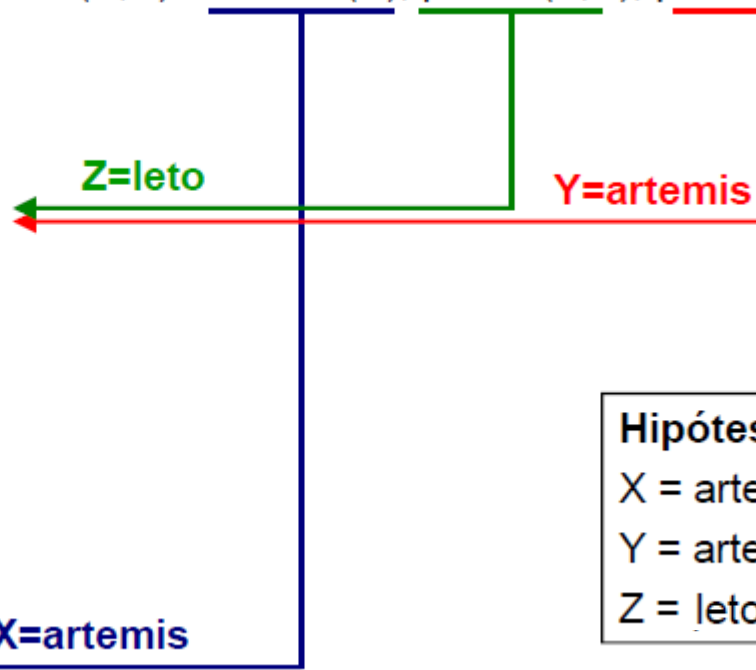


**SWI Prolog**

**BC**

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (leto,Y)**  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).



**Hipótese:**

X = artemis  
Y = artemis  
Z = leto





SWI Prolog

Backtracking

← ← ←

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).

Falha!

Z=leto

Y=artemis

X=artemis

Hipótese:

X = artemis

Y = artemis

Z = leto

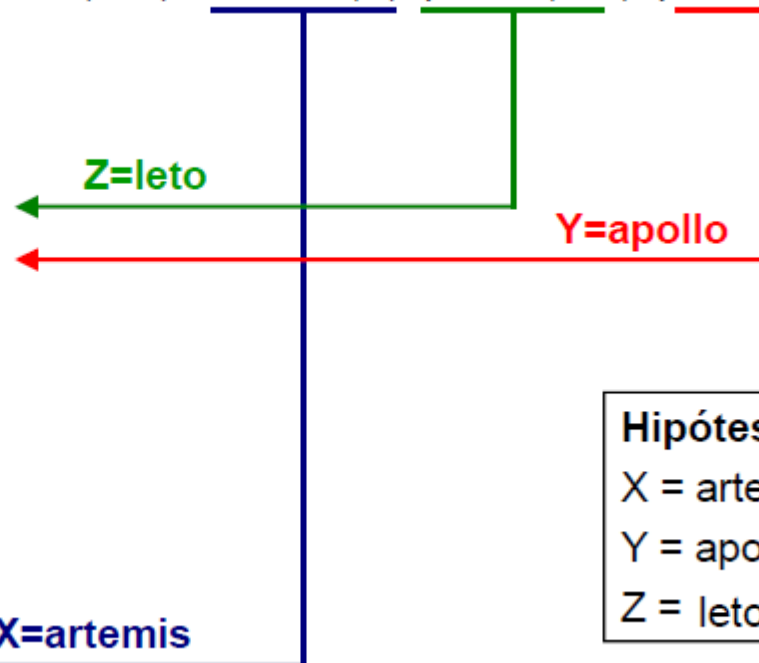


SWI Prolog

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

**parent (leto,Y)**  
sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).



**Hipótese:**

X = artemis

Y = apollo

Z = leto

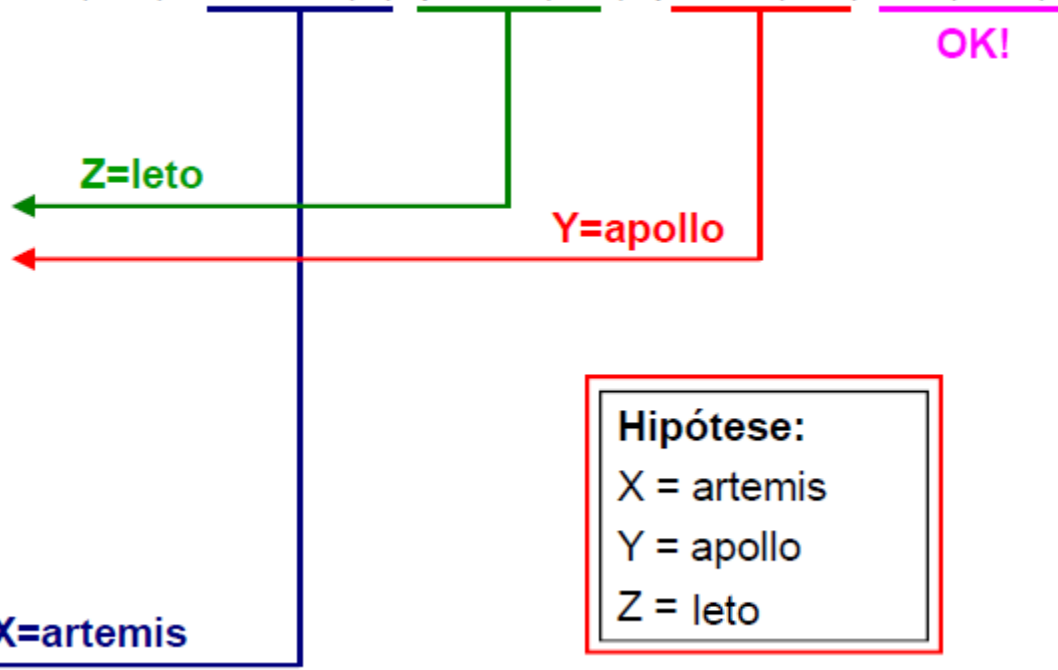


SWI Prolog

BC

```
parent(chaos,gaea).
parent(coeus,leto).
parent(phoebe,leto).
parent(leto,artemis).
parent(leto,apollo).
sister(X,Y) :- ...
female(gaea).
female(rhea).
female(leto).
female(hera).
female(phoebe).
female(tethys).
female(artemis).
```

sister(X,Y) :- female(X), parent(Z,X), parent(Z,Y), not(X=Y).



# PROLOG

Percorre a base de conhecimento procurando satisfazer o primeiro elemento que encontra na BC que casa com a consulta.

Se falhar, busca outra opção fazendo BACKTRACKING (voltando a clausula anterior e procurando outro valor para ela)

- Este método é chamado de BUSCA EM PROFUNDIDADE



# Exemplo Resolução x PROLOG

Fala(Angela,Portugues)

Fala(Jose,Ingles)

Fala(Viola,Portugues)

Fala(Viola,Espanhol)

Fala(Carlos,Ingles)

Fala(Sonia,Ingles)

Fala(Wilson,Ingles)

$Fala(x,z) \wedge Fala(y,z) \rightarrow Falam\_a\_mesma\_lingua(x,y)$

$BC \models Falam\_a\_mesma\_lingua(Viola,Angela) \text{ ?}$

# Exemplo Resolução x PROLOG

C1: Fala(Angela,Portugues)

C2: Fala(Jose,Ingles)

C3: Fala(Viola,Portugues)

C4: Fala(Viola,Espanhol)

C5: Fala(Carlos,Ingles)

C6:Fala(Sonia,Ingles)

C7:Fala(Wilson,Ingles)

C8:

$$\text{Fala}(x,z) \wedge \text{Fala}(y,z) \rightarrow \text{Falam\_a\_mesma\_lingua}(x,y)$$

$$\neg(\text{Fala}(x,z) \wedge \text{Fala}(y,z)) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$$

$$\neg \text{Fala}(x,z) \vee \neg \text{Fala}(y,z) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$$

$$\text{BC} \models \text{Falam\_a\_mesma\_lingua}(\text{Viola},\text{Angela}) \text{ ?}$$

# Exemplo Resolução x PROLOG

C1: Fala(Angela,Portugues)

C2: Fala(Jose,Ingles)

C3: Fala(Viola,Portugues)

C4: Fala(Viola,Espanhol)

C5: Fala(Carlos,Ingles)

C6: Fala(Sonia,Ingles)

C7: Fala(Wilson,Ingles)

C8:  $\neg$  Fala(x,z)  $\vee$   $\neg$  Fala(y,z)  $\vee$  Falam\_a\_mesma\_lingua(x,y)

C9:  $\neg$  Falam\_a\_mesma\_lingua(Viola,Angela)

# Exemplo Resolução x PROLOG

C1: Fala(Angela,Portugues)

C2: Fala(Jose,Ingles)

C3: Fala(Viola,Portugues)

C4: Fala(Viola,Espanhol)

C5: Fala(Carlos,Ingles)

C6: Fala(Sonia,Ingles)

C7: Fala(Wilson,Ingles)

C8:  $\neg \text{Fala}(x,z) \vee \neg \text{Fala}(y,z) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$

C9:  $\neg \text{Falam\_a\_mesma\_lingua}(\text{Viola},\text{Angela})$

C8 com C9:

$\neg \text{Fala}(x,z) \vee \neg \text{Fala}(y,z) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$        $\neg \text{Falam\_a\_mesma\_lingua}(\text{Viola},\text{Angela})$

$\neg \text{Fala}(\text{Viola},z) \vee \neg \text{Fala}(\text{Angela},z)$        $\theta = \{x/\text{Viola}, y/\text{Angela}\}$



# Exemplo Resolução x PROLOG

C1: Fala(Angela,Portugues)

C2: Fala(Jose,Ingles)

C3: Fala(Viola,Portugues)

C4: Fala(Viola,Espanhol)

C5: Fala(Carlos,Ingles)

C6: Fala(Sonia,Ingles)

C7: Fala(Wilson,Ingles)

C8:  $\neg \text{Fala}(x,z) \vee \neg \text{Fala}(y,z) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$

C9:  $\neg \text{Falam\_a\_mesma\_lingua}(\text{Viola},\text{Angela})$

C10:  $\neg \text{Fala}(\text{Viola},z) \vee \neg \text{Fala}(\text{Angela},z)$

**C10 com C3:**

$\neg \text{Fala}(\text{Viola},Z) \vee \neg \text{Fala}(\text{Angela},Z)$

$\neg \text{Fala}(\text{Angela},\text{Portugues})$

$\text{Fala}(\text{Viola},\text{Portugues})$

$\theta = \{z/\text{Portugues}\}$

# Exemplo Resolução x PROLOG

C1: Fala(Angela,Portugues)

C2: Fala(Jose,Ingles)

C3: Fala(Viola,Portugues)

C4: Fala(Viola,Espanhol)

C5: Fala(Carlos,Ingles)

C6: Fala(Sonia,Ingles)

C7: Fala(Wilson,Ingles)

C8:  $\neg \text{Fala}(x,z) \vee \neg \text{Fala}(y,z) \vee \text{Falam\_a\_mesma\_lingua}(x,y)$

C9:  $\neg \text{Falam\_a\_mesma\_lingua}(\text{Viola},\text{Angela})$

C10:  $\neg \text{Fala}(\text{Viola},z) \vee \neg \text{Fala}(\text{Angela},z)$

C11:  $\neg \text{Fala}(\text{Angela},\text{Portugues})$

**C11 com C1:**

$\neg \text{Fala}(\text{Angela},\text{Portugues})$

$\text{Fala}(\text{Angela},\text{Portugues})$



# Exemplo Resolução x PROLOG

Fala(Angela,Portugues)

Fala(Jose,Ingles)

Fala(Viola,Portugues)

Fala(Viola,Espanhol)

Fala(Carlos,Ingles)

Fala(Sonia,Ingles)

Fala(Wilson,Ingles)

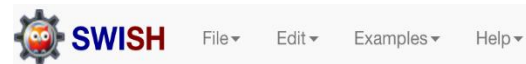
$Fala(x,z) \wedge Fala(y,z) \rightarrow Falam\_a\_mesma\_lingua(x,y)$

$BC \models Falam\_a\_mesma\_lingua(Viola,Angela) \text{ ?}$

**SIM !!!**

# SWI-Prolog – Versão WEB

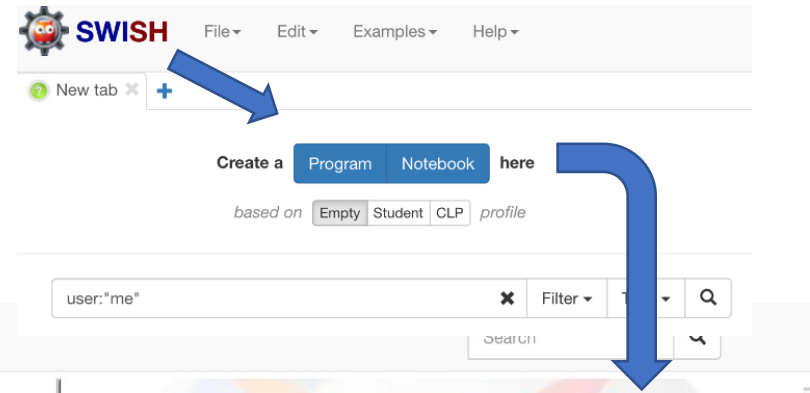
<http://swish.swi-prolog.org/>



Program

1 Your Prolog rules and facts go here ...

**FATOS E REGRAS**



**RESPOSTAS**

**PERGUNTAS**



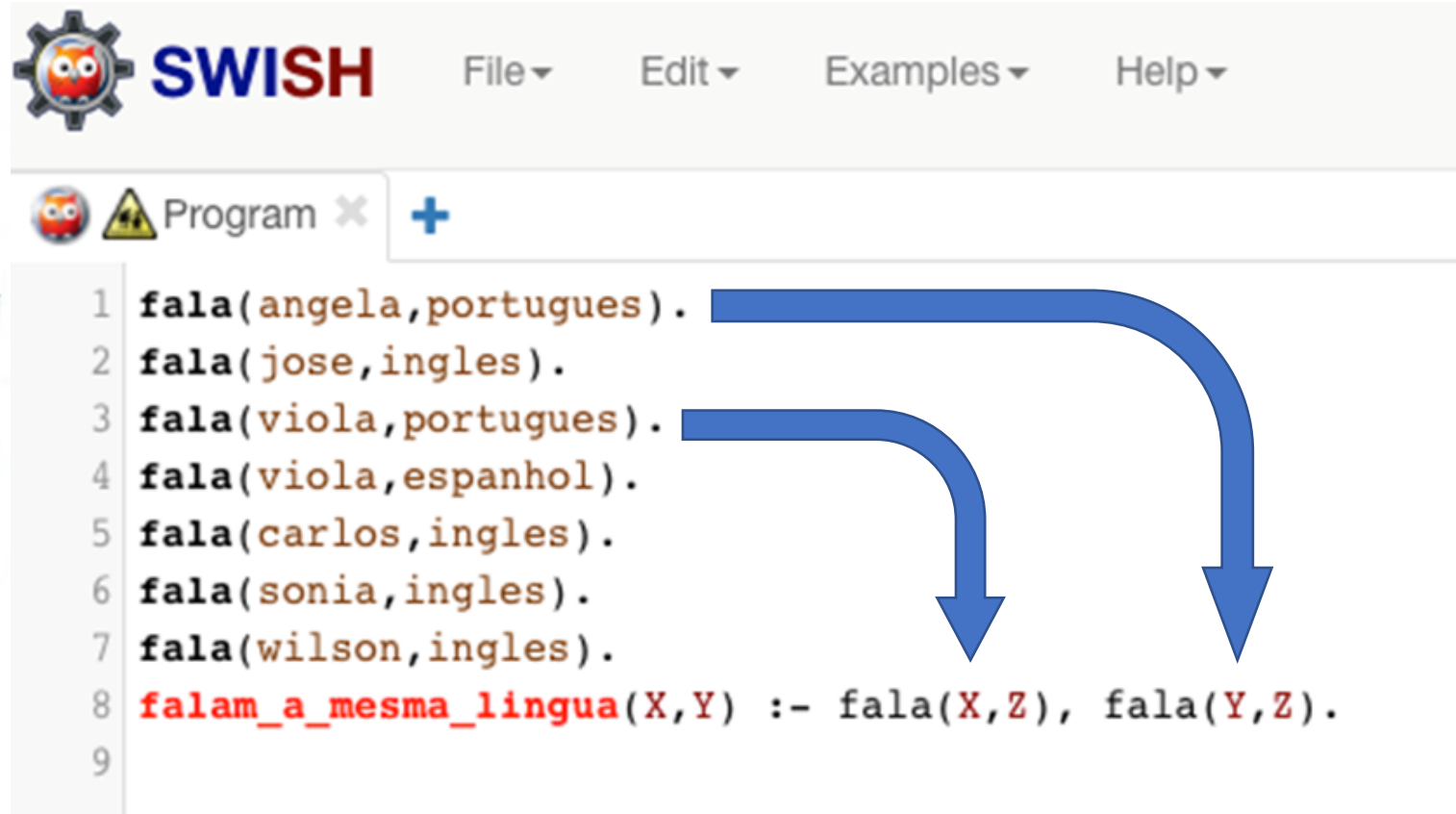
# Exemplo Resolução x PROLOG

**SWISH** File Edit Examples Help  
Program x +  

```
1 fala(angela,portugues).  
2 fala(jose,ingles).  
3 fala(viola,portugues).  
4 fala(viola,espanhol).  
5 fala(carlos,ingles).  
6 fala(sonia,ingles).  
7 fala(wilson,ingles).  
8 falam_a_mesma_lingua(X,Y) :- fala(X,Z), fala(Y,Z).  
9
```



# Exemplo Resolução x PROLOG

A screenshot of the SWISH Prolog environment. The window title is 'SWISH' with a gear icon. The menu bar includes 'File', 'Edit', 'Examples', and 'Help'. Below the menu is a tab labeled 'Program' with a warning icon and a '+' button. The code editor contains the following Prolog code:

```
1 fala(angela,portugues).
2 fala(jose,ingles).
3 fala(viola,portugues).
4 fala(viola,espanhol).
5 fala(carlos,ingles).
6 fala(sonia,ingles).
7 fala(wilson,ingles).
8 falam_a_mesma_lingua(X,Y) :- fala(X,Z), fala(Y,Z).
9
```

Two blue arrows originate from the code: one from line 1 pointing to the first 'fala' in line 8, and another from line 3 pointing to the second 'fala' in line 8.

```
? falam_a_mesma_lingua(viola,angela)
YES !
```

# Podemos descobrir mais....



Se o José  
fala Inglês

SWISH File Edit Examples Help 143 users online

Program

```
1 fala(angela,portugues).
2 fala(jose,ingles).
3 fala(viola,portugues).
4 fala(viola,espanhol).
5 fala(carlos,ingles).
6 fala(sonia,ingles).
7 fala(wilson,ingles).
8 falam_a_mesma_lingua(X,Y) :- fala(X,Z), fala(Y,Z).
9
```

Resposta → `fala(jose,ingles)`  
true

Pergunta → `?- fala(jose,ingles)`

Quem fala Inglês

1ª. Resposta →

`fala(A,ingles)`  
A = jose  
Next 10 100 1,000 Stop

Pergunta → `?- fala(A,ingles)`

`fala(A,ingles)`  
A = jose  
A = carlos  
A = sonia  
A = wilson

`?- fala(A,ingles)`



# Pode ser mais divertido

Ele pode te ensinar a resolver a torre de Hanoi



```
Program x +
1  move(1,X,Y,_):-
2    write('Mova disco superior '),
3    write(X),
4    write(' para '),
5    write(Y),
6    nl.
7  move(N,X,Y,Z):-
8    N>1,
9    M is N-1,
10   move(M,X,Z,Y),
11   move(1,X,Y,_),
12   move(M,Z,Y,X).
13
```

```
move(4,esquerda,direita,centro).
Mova disco superior esquerda para centro
Mova disco superior esquerda para direita
Mova disco superior centro para direita
Mova disco superior esquerda para centro
Mova disco superior direita para esquerda
Mova disco superior direita para centro
Mova disco superior esquerda para centro
Mova disco superior esquerda para direita
Mova disco superior centro para direita
Mova disco superior centro para esquerda
Mova disco superior direita para esquerda
Mova disco superior centro para direita
Mova disco superior esquerda para centro
Mova disco superior esquerda para direita
Mova disco superior centro para direita
true
Next 10 100 1,000 Stop
?- move(4,esquerda,direita,centro).
```



# Existem outros Métodos de Inferência ?

Existem !

- Mas não são consequências lógicas !
  - Porém permitem tomadas de decisão
  - Permitem aprendizado
  - Permitem raciocínio
- São eles:
  - Abdução
  - Indução

# Abdução

$$\frac{\forall x: \text{bebado}(x) \rightarrow \text{anda\_torto}(x), \quad \text{anda\_torto}(\text{joão})}{\text{bebado}(\text{joão}).} \quad \frac{\beta \Rightarrow \alpha, \alpha}{\beta}$$

- A abdução nos leva a uma inferência plausível.
- Porém não nos leva a uma consequência lógica, mas sim a uma hipótese !
- Mas pode levar a erros: *João poderia estar com dor na perna ao invés de bebado.*
- A abdução não gera uma conclusão, como a dedução, mas sim uma hipótese !
- Abdução permite encontrar mais de uma resposta.
- Para decidir por apenas uma hipótese, deve-se procurar heurísticas mais prováveis
- Ex de aplicação: **Diagnóstico Médico.**

# Indução

- Inicialmente → quantidade grande de observações  $\alpha_1, \alpha_2, \alpha_3, \dots$   
 $\alpha_i$  p/ todo  $i$
- Examinando as observações → descobre-se regras para prever o comportamento de um certo fenômeno.
- EX:
  - Chevette a mais de 200 Km/h capota na imigrantes
  - Fusca a mais de 200 Km/h capota na imigrantes
  - Gurgel BR 800 a mais de 200 Km/h capota na imigrantes
  - ...
  - por indução: qualquer carro a mais de 200 Km/h capota na imigrantes
- Se o número de observações for insuficientes ou os dados relevantes forem mal escolhidos, o método leva a conclusões erradas.
- Uma amostragem de exemplos ampla e representativa elimina, ou diminui muito, a chance de erro na conclusão.

# Referências desta Aula

- Russel & Norvig (Artificial Intelligence):
  - Capítulos 6 a 9
- Rezende, Solange (Sistemas Inteligentes):
  - Capítulos 2 e 3
- Silva, F. S., Finger, M.; Melo, Ana C.V. (Lógica p/ a Computação):
  - Capítulos 1, 2 e 3
- Alguns slides desta aula foram baseados nos slides:
- Paulo Eduardo Santos: “Fundamentos da Inteligência Artificial”, FEI, 2005
- Alexandre da Silva Simões: “Agentes Lógicos – Aula 7”, UNESP, 2010
- Paulo Eduardo Santos: “Inferência em LPO”, FEI, 2005
- Alexandre da Silva Simões: “Lógica de Primeira Ordem – Aula 8”, UNESP, 2010
- Alexandre da Silva Simões: “Programação em Lógica – Aula 9”, UNESP, 2010
- Salvatore J. Stolfo: Inference in first-order logic, Columbia University – USA