Lógica Matemática
Joseluce de Farias Cunha

PROLOG

Alexandre de Andrade Barbosa
aab@dsc.ufcg.edu.br

Introdução

Os aspectos teóricos relacionados à lógica matemática já foram estudados ao longo do curso.

Esta parte do curso busca apresentar aspectos práticos relacionados à lógica, especificamente à programação em lógica, através da linguagem Prolog.

Espera-se que após esta parte do curso, o aluno seja capaz de:

- compreender e executar programas Prolog;
- modificar programas Prolog;
- escrever programas Prolog básicos e intermediários.

Bibliografia


▶ Judith L. Gersting.
Fundamentos matemáticos para à Ciência da Computação.
LTC, 2001.


▶ Ivan Bratko.
Prolog Programming for Artificial Intelligence.
Addison-Wesley, 1986.


Marco A. Casanova, Fernando Giorno & Antônio L. Furtado.
Programação em lógica e a linguagem Prolog.
Edgar Blucher, 1987.


L. Sterling e E. Shapiro.
The Art of Prolog.
MIT Press, 1986.

PARTE I

Programação em Lógica e Prolog

Existem diversos paradigmas de programação, tais como:

- procedimental (Java, C, C++, Pascal, ...);
- imperativo (LISP);
- descritivo ou declarativo (Prolog, Gödel, ...).

No paradigma descritivo o programador implementa uma descrição do problema e não as instruções para sua resolução.

Prolog é uma linguagem para programação em lógica, ou seja, é uma linguagem descritiva.

Um programa Prolog constitui-se de uma coleção de fatos e regras que são utilizadas por um "motor de inferência" para checar se uma consulta pode ser deduzida desta coleção.

Programação em Lógica e Prolog

Qual é o melhor paradigma???

Você está em uma competição, se ganhar receberá uma certa quantia em dinheiro. Para participar escolha um dos veículos:

Figura: Jeep Rubicon X Ferrari F50

Programação em Lógica e Prolog

Qual é o melhor paradigma???

Você está em uma competição, se ganhar receberá uma certa quantia em dinheiro. Para participar escolha um dos veículos:

Figura: Jeep Rubicon X Ferrari F50

Sua tarefa é ganhar um rali...

Programação em Lógica e Prolog

Qual é o melhor paradigma???

Você está em uma competição, se ganhar receberá uma certa quantia em dinheiro. Para participar escolha um dos veículos:

Figura: Jeep Rubicon X Ferrari F50

Sua tarefa é ganhar um rali...

Mas se sua tarefa for ganhar uma corrida em interlagos...

Programação em Lógica e Prolog

Qual é o melhor paradigma???

Você está em uma competição, se ganhar receberá uma certa quantia em dinheiro. Para participar escolha um dos veículos:

Figura: Jeep Rubicon X Ferrari F50

Sua tarefa é ganhar um rali...

Mas se sua tarefa for ganhar uma corrida em interlagos...

Não existe "o" melhor paradigma ou linguagem, existe o mais adequado para resolução de determinados problemas.

Programação em Lógica e Prolog

Apesar das linguagens procedimentais serem as mais utilizadas, linguagens descritivas ou imperativas são mais adequadas para resolver certos problemas.

Prolog foi criada em 1972 por Colmerauer e Roussel, e é mais adequada para problemas onde é necessário descrever conhecimento, por exemplo:

- em aplicações que realizem computação simbólica;
- na compreensão de linguagem natural;
- em sistemas especialistas.

Implementações Prolog

Existem diversas implementações de Prolog, algumas das mais conhecidas são:

- **Win-Prolog**;
- **Ciao Prolog**;
- **YAP Prolog**;
- **SWI Prolog + SWI-Prolog-Editor** (recomendado);
  - `www.swi-prolog.org`
  - `www.dsc.ufcg.edu.br/~aab/prolog`
- **SICStus Prolog**.

Sintaxe SWI-Prolog

Os dados representados em Prolog podem ser um dos seguintes tipos:

- **variáveis** - iniciadas com maiúsculas ou *underscore* (_), seguidos de qualquer caractere alfanumérico. Somente *underscore* define uma variável anônima. Ex.: X, Y1, _Nome, ...;
- **átomos** - são constantes, devem ser iniciadas com minúsculas seguidas de qualquer caractere alfanumérico ou qualquer seqüência entre ‘ ’ (aspas simples). Ex.: joao, ‘João’, ‘16’, ...;
- **inteiros** - qualquer seqüência numérica que não contenha ponto (.). Caracteres ASCII entre “ ” (aspas duplas) são tratados como listas de inteiros. Ex.: 1, 6, -3, “a”, ...;
- **floats** - números com um ponto (.) e pelo menos uma casa decimal. Ex.: 5.3 (correto), 7. (incorreto);
- **listas** - seqüência ordenada de elementos entre [ ] e separados por vírgulas. Ex.: [a, b, c], [a | b, c].

Sintaxe SWI-Prolog

Os comandos *write* e *read*, escrevem e lêem sobre os arquivos padrão (monitor e teclado).
Para escrever bastar utilizar o comando *write(+termo)*:

- *write('Teste de impressão.').* (Correto)
- *write(Teste de impressão.).* (Errado)
- *write(X).* (Correto)
- *write(joao).* (Correto)

Para ler deve-se usar o comando *read(+var)*:

- *read(X).* (Correto)
- *read(x).* (Errado)
- *read(Joao).* (Correto)
- *read(joao).* (Errado)

Sintaxe SWI-Prolog

Alguns caracteres são especiais para impressão, são eles:

- nl, \n, \l - nova linha.
- \r - retorna ao início da linha;
- \t - tabulação;
- \% - imprime o símbolo %;

Existem dois tipos de comentários em Prolog, são eles:

- % - todo texto existente na mesma linha após o símbolo é considerado comentário;
- /* */ - todo o texto entre os símbolos é considerado comentário.

Exemplo

/* Descrição dos precidados homem e mulher. */
homem(pedro). **% representa o fato de que pedro é homem.**
mulher(teresa). **% representa o fato de que teresa é mulher.**

Usando o SWI-Prolog

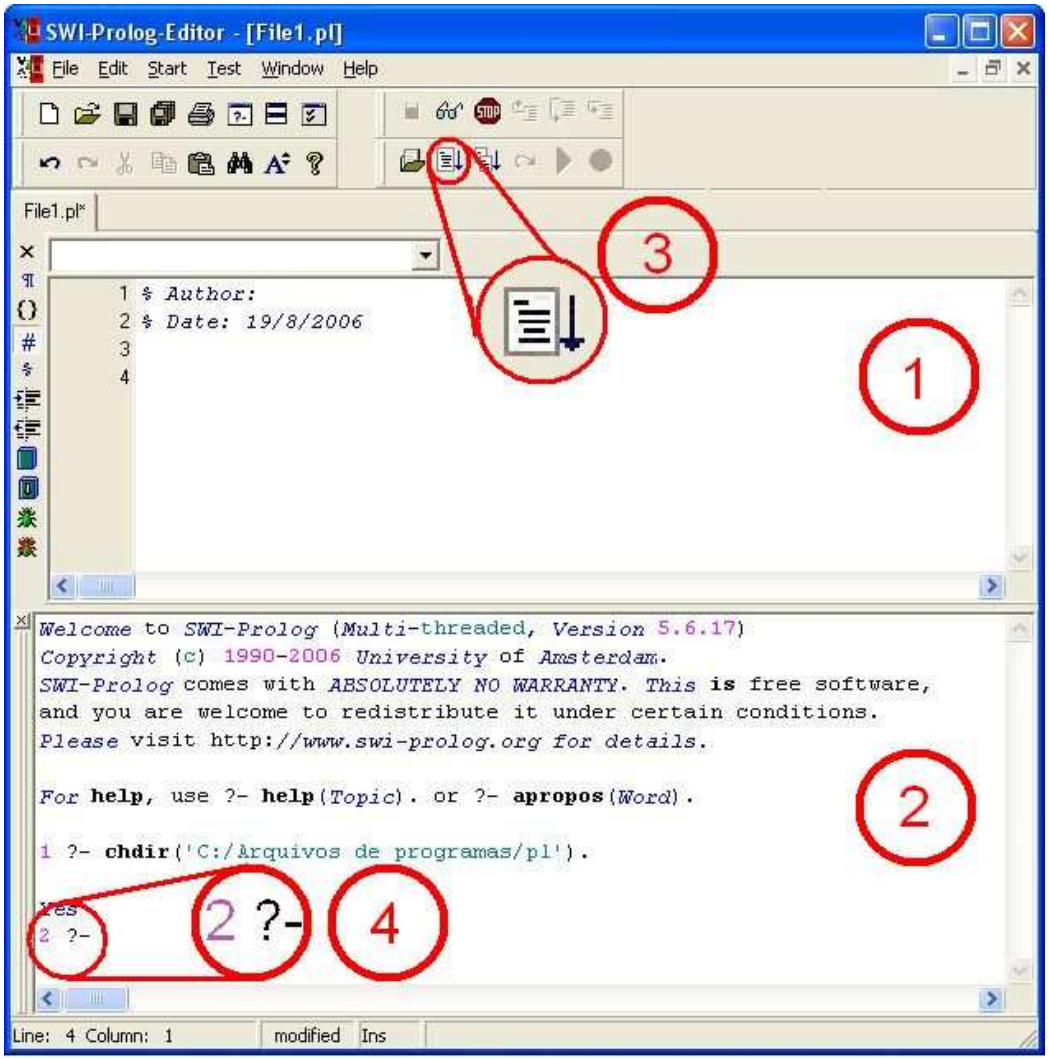

Figura: Tela inicial do SWI-Prolog Editor

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*
   - *a) vINCENT* — *átomo*
   - *b) 23* — *inteiro*
   - *c) Variable2000* — *variável*
   - *d) [a, b, c]* — *lista*
   - *e) variable23* — *átomo*
   - *f) aulas de lógica* — *Inválido*
   - *g) 'Joao'* — *átomo*
   - *h) [1, [2, 3], 4]* — *lista*
   - *i) Footmassage* — *variável*
   - *j) 65.* — *Inválido*
   - *k) 23.0* — *float*
   - *l) _* — *variável anônima*
   - *m) 'aulas de lógica'* — *átomo*
   - *n) _Var* — *variável*
   - *o) "a"* — *lista*
   - *p) []* — *lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

Usando o SWI-Prolog

Exercício

1. *Classifique os termos abaixo como átomo, variável, número ou lista:*

a) vINCENT	*átomo*	*i) Footmassage*	*variável*
b) 23	*inteiro*	*j) 65.*	*Inválido*
c) Variable2000	*variável*	*k) 23.0*	*float*
d) [a, b, c]	*lista*	*l) _*	*variável anônima*
e) variable23	*átomo*	*m) 'aulas de lógica'*	*átomo*
f) aulas de lógica	*Inválido*	*n) _Var*	*variável*
g) 'Joao'	*átomo*	*o) "a"*	*lista*
h) [1, [2, 3], 4]	*lista*	*p) []*	*lista*

- Para checar as respostas utilize os comandos:
  - **atom(+termo)** - checa se termo é um átomo;
  - **var(+termo)** - checa se termo é uma variável;
  - **number(+termo)** - checa se termo é um número (inteiro ou float);
  - **is_list(+termo)** - checa se termo é uma lista;

Usando o SWI-Prolog

Todos os comandos devem ser finalizados com (.), alguns exemplos:

Exemplo

```
start(_) :-
  write('Digite o valor de X: '), nl,
  read(X), nl,
  write(X), nl.
```

Exemplo

```
?- atom(vINCENT).
?- var(X).
?- is_list([a, b, c]).
?- number(23).
?- start(X).
|: 1 .
```

PARTE II

Fatos

Um programa Prolog é uma coleção de fatos e regras.

Fatos são sempre verdadeiros, mas as regras precisam ser avaliadas.

Como criar um fato em uma base Prolog:

- homem(x). - significa que "x é um homem";
- genitor(x, y). - significa que "x é genitor de y" ou "y é genitor de x";

É responsabilidade do programador definir os predicados corretamente.

Fatos

Durante toda a "PARTE II" iremos descrever as seguintes relações:

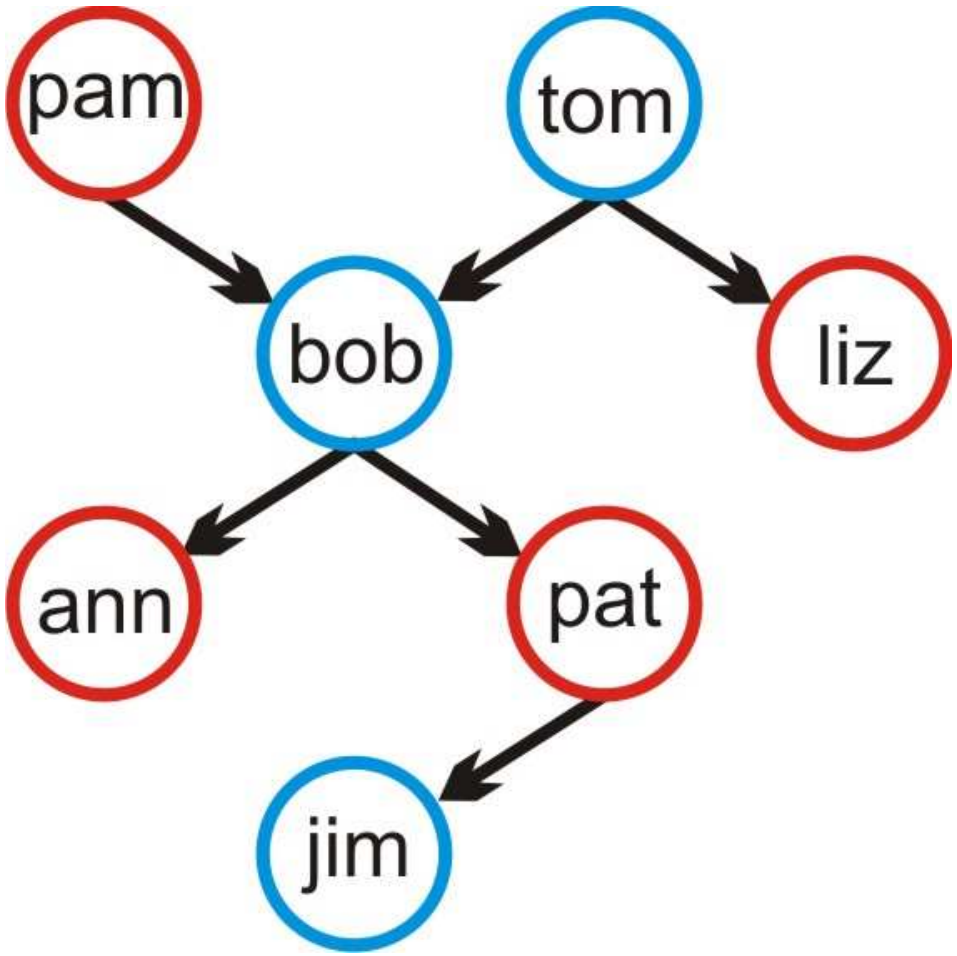

Figura: Árvore genealógica

Fatos

Definidos os predicados:

- homem(x) - significando que "x é um homem";
- mulher(x) - significando que "x é um mulher";
- genitor(x, y) - significando que "x é genitor de y";

Usando apenas fatos, descreva as seguintes relações:

- tom, bob, jim, são homens;
- pam, liz, ann e pat, são mulheres;
- tom e pam são os pais de bob;
- bob é pai de ann e pat;
- pat é a mãe de jim.

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
  mulher(pam).          % pam é mulher
  homem(tom).           % tom é homem
  homem(bob).           % bob é homem
  mulher(liz).          % liz é mulher
  mulher(pat).          % pat é mulher
  mulher(ann).          % ann é mulher
  homem(jim).           % jim é homem
  genitor(pam, bob).    % pam é genitora de bob
  genitor(tom, bob).    % tom é genitor de bob
  genitor(bob, ann).    % bob é genitor de ann
  genitor(bob, pat).    % bob é genitor de pat
  genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
  mulher(pam).          % pam é mulher
  homem(tom).           % tom é homem
  homem(bob).           % bob é homem
  mulher(liz).          % liz é mulher
  mulher(pat).          % pat é mulher
  mulher(ann).          % ann é mulher
  homem(jim).           % jim é homem
  genitor(pam, bob).    % pam é genitora de bob
  genitor(tom, bob).    % tom é genitor de bob
  genitor(bob, ann).    % bob é genitor de ann
  genitor(bob, pat).    % bob é genitor de pat
  genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
  mulher(pam).          % pam é mulher
  homem(tom).           % tom é homem
  homem(bob).           % bob é homem
  mulher(liz).          % liz é mulher
  mulher(pat).          % pat é mulher
  mulher(ann).          % ann é mulher
  homem(jim).           % jim é homem
  genitor(pam, bob).    % pam é genitora de bob
  genitor(tom, bob).    % tom é genitor de bob
  genitor(bob, ann).    % bob é genitor de ann
  genitor(bob, pat).    % bob é genitor de pat
  genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exemplo

```
/* A ordem das regras é importante, deve-se também agrupar os predicados */
mulher(pam).          % pam é mulher
homem(tom).           % tom é homem
homem(bob).           % bob é homem
mulher(liz).          % liz é mulher
mulher(pat).          % pat é mulher
mulher(ann).          % ann é mulher
homem(jim).           % jim é homem
genitor(pam, bob).    % pam é genitora de bob
genitor(tom, bob).    % tom é genitor de bob
genitor(bob, ann).    % bob é genitor de ann
genitor(bob, pat).    % bob é genitor de pat
genitor(pat, jim).    % pat é genitora de jim
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

a) animal(urso). *% urso é um animal*
b) animal(peixe). *% peixe é um animal*
c) planta(alga). *% alga é um planta*
d) planta(grama). *% grama é um planta*
e) come(urso, peixe). *% urso come peixe*
f) come(coelho, leão). *% ???coelho come leão???*
g) menor(formiga, tamanduá). *% formiga é menor que o tamanduá*
h) menor(elefante, cavalo). *% ???elefante é menor que o cavalo???*
i) proximo('JP', 'CG'). *% JP é próxima de CG*
j) proximo(Brasil, Japao). *% não diz -> BR é próximo de JP.*

Fatos

Exercício

2. O que os fatos abaixo descrevem:

a) animal(urso). *% urso é um animal*
b) animal(peixe). *% peixe é um animal*
c) planta(alga). *% alga é um planta*
d) planta(grama). *% grama é um planta*
e) come(urso, peixe). *% urso come peixe*
f) come(coelho, leão). *% ???coelho come leão???*
g) menor(formiga, tamanduá). *% formiga é menor que o tamanduá*
h) menor(elefante, cavalo). *% ???elefante é menor que o cavalo???*
i) proximo('JP', 'CG'). *% JP é próxima de CG*
j) proximo(Brasil, Japao). *% não diz -> BR é próximo de JP.*

Fatos

Exercício

2. O que os fatos abaixo descrevem:

a) animal(urso). *% urso é um animal*

b) animal(peixe). *% peixe é um animal*

c) planta(alga). *% alga é um planta*

d) planta(grama). *% grama é um planta*

e) come(urso, peixe). *% urso come peixe*

f) come(coelho, leão). *% ???coelho come leão???*

g) menor(formiga, tamanduá). *% formiga é menor que o tamanduá*

h) menor(elefante, cavalo). *% ???elefante é menor que o cavalo???*

i) proximo('JP', 'CG'). *% JP é próxima de CG*

j) proximo(Brasil, Japao). *% não diz -> BR é próximo de JP.*

Fatos

Exercício

2. O que os fatos abaixo descrevem:

a) animal(urso). *% urso é um animal*

b) animal(peixe). *% peixe é um animal*

c) planta(alga). *% alga é um planta*

d) planta(grama). *% grama é um planta*

e) come(urso, peixe). *% urso come peixe*

f) come(coelho, leão). *% ???coelho come leão???*

g) menor(formiga, tamanduá). *% formiga é menor que o tamanduá*

h) menor(elefante, cavalo). *% ???elefante é menor que o cavalo???*

i) proximo('JP', 'CG'). *% JP é próxima de CG*

j) proximo(Brasil, Japao). *% não diz -> BR é próximo de JP.*

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                    % urso é um animal
b) animal(peixe).                   % peixe é um animal
c) planta(alga).                    % alga é um planta
d) planta(grama).                   % grama é um planta
e) come(urso, peixe).               % urso come peixe
f) come(coelho, leão).              % ???coelho come leão???
g) menor(formiga, tamanduá).        % formiga é menor que o tamanduá
h) menor(elefante, cavalo).         % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').             % JP é próxima de CG
j) proximo(Brasil, Japao).          % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                  % urso é um animal
b) animal(peixe).                 % peixe é um animal
c) planta(alga).                  % alga é um planta
d) planta(grama).                 % grama é um planta
e) come(urso, peixe).             % urso come peixe
f) come(coelho, leão).            % ???coelho come leão???
g) menor(formiga, tamanduá).      % formiga é menor que o tamanduá
h) menor(elefante, cavalo).       % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').           % JP é próxima de CG
j) proximo(Brasil, Japao).        % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

a) animal(urso). *% urso é um animal*
b) animal(peixe). *% peixe é um animal*
c) planta(alga). *% alga é um planta*
d) planta(grama). *% grama é um planta*
e) come(urso, peixe). *% urso come peixe*
f) come(coelho, leão). *% ???coelho come leão???*
g) menor(formiga, tamanduá). *% formiga é menor que o tamanduá*
h) menor(elefante, cavalo). *% ???elefante é menor que o cavalo???*
i) proximo('JP', 'CG'). *% JP é próxima de CG*
j) proximo(Brasil, Japao). *% não diz -> BR é próximo de JP.*

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                % urso é um animal
b) animal(peixe).               % peixe é um animal
c) planta(alga).                % alga é um planta
d) planta(grama).               % grama é um planta
e) come(urso, peixe).           % urso come peixe
f) come(coelho, leão).          % ???coelho come leão???
g) menor(formiga, tamanduá).    % formiga é menor que o tamanduá
h) menor(elefante, cavalo).     % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').         % JP é próxima de CG
j) proximo(Brasil, Japao).      % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                % urso é um animal
b) animal(peixe).               % peixe é um animal
c) planta(alga).                % alga é um planta
d) planta(grama).               % grama é um planta
e) come(urso, peixe).           % urso come peixe
f) come(coelho, leão).          % ???coelho come leão???
g) menor(formiga, tamanduá).    % formiga é menor que o tamanduá
h) menor(elefante, cavalo).     % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').         % JP é próxima de CG
j) proximo(Brasil, Japao).      % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                % urso é um animal
b) animal(peixe).               % peixe é um animal
c) planta(alga).                % alga é um planta
d) planta(grama).               % grama é um planta
e) come(urso, peixe).           % urso come peixe
f) come(coelho, leão).          % ???coelho come leão???
g) menor(formiga, tamanduá).    % formiga é menor que o tamanduá
h) menor(elefante, cavalo).     % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').         % JP é próxima de CG
j) proximo(Brasil, Japao).      % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                % urso é um animal
b) animal(peixe).               % peixe é um animal
c) planta(alga).                % alga é um planta
d) planta(grama).               % grama é um planta
e) come(urso, peixe).           % urso come peixe
f) come(coelho, leão).          % ???coelho come leão???
g) menor(formiga, tamanduá).    % formiga é menor que o tamanduá
h) menor(elefante, cavalo).     % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').         % JP é próxima de CG
j) proximo(Brasil, Japao).      % não diz -> BR é próximo de JP.
```

Fatos

Exercício

2. O que os fatos abaixo descrevem:

```
a) animal(urso).                % urso é um animal
b) animal(peixe).               % peixe é um animal
c) planta(alga).                % alga é um planta
d) planta(grama).               % grama é um planta
e) come(urso, peixe).           % urso come peixe
f) come(coelho, leão).          % ???coelho come leão???
g) menor(formiga, tamanduá).    % formiga é menor que o tamanduá
h) menor(elefante, cavalo).     % ???elefante é menor que o cavalo???
i) proximo('JP', 'CG').         % JP é próxima de CG
j) proximo(Brasil, Japao).      % não diz -> BR é próximo de JP.
```

Então o que diz o "fato" **j**? **j** é um fato realmente?

Consultas

A cláusula *proximo(Brasil, Japao).* é uma consulta Prolog, pois, "Brasil" e "Japao" são variáveis.

Para responder consultas Prolog utiliza:

- **matching** - checa se determinado padrão está presente, para saber quais fatos e regras podem ser utilizados;
- **unificação** - substitui o valor de variáveis para determinar se a consulta é satisfeita pelos fatos ou regras da base (programa);
- **resolução** - verifica se uma consulta é conseqüência lógica dos fatos e regras da base (programa);
- **recursão** - utiliza regras que chamam a si mesmas para realizar demonstrações;
- **backtracking** - para checar todas as possibilidades de resposta.

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).     <- tom é genitor de quem (X)?
X = bob ;               <- Unifica [X/bob]
No

?- genitor(X, bob).     <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;              <- genitor(tom, bob). (matching)
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;              <- busca outras soluções (backtrack).
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No                     <- Nenhum outro fato satisfaz a consulta.

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;              <- Unifica [X/tom]
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;               <- genitor(tom, bob). (matching)
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;                <- busca outras soluções (backtrack).
X = pam ;
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;              <- Unifica [X/pam]
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;              <- genitor(pam, bob). (matching)
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;              <- busca outras soluções (backtrack).
No
```

Consultas

Exemplo

```
homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato
```

Exemplo

```
?- genitor(tom, X).    <- tom é genitor de quem (X)?
X = bob ;
No

?- genitor(X, bob).    <- Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;
X = pam ;
No                     <- Nenhum outro fato satisfaz a consulta.
```

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– usa regra 3*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– Unifica [X/a] e [Y/e]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– Nova meta chama(a,e)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,e) ? *<– Falha, não existe o fato*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,e) ? *<– busca outras soluções (backtrack)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– usa regra 4*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– Unifica [X/a] e [Y/e]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ? *<– Nova meta usa(a,e)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

usa(a,e) ? *<– Falha, não existe o fato*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

usa(a,e) ? *<– busca outras soluções (backtrack)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– usa regra 5*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– Unifica [X/a] e [Y/e]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? *<– Nova meta chama(a,Z)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,Z) ? *<– Unifica [Z/b]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,Z) ? *<– chama(a,b) (matching)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,b) ? OK *<– Nova meta depende(b, e)* **(recursão)**

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ? *<– usa regra 3*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ? *<– Unifica [X/b] e [Y/e]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ? *<– Nova meta chama(b,e)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ?
chama(b, e) ? *<– Falha, não existe o fato*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ?
chama(b, e) ? *<– busca outras soluções (backtrack)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ? *<– usa regra 4*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ? *<– Unifica [X/b] e [Y/e]*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,b) ? OK

depende(b, e) ? *<– Nova meta usa(b,e)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*
depende(a, e) ?
chama(a,b) ? OK
depende(b, e) ?
usa(b,e) ? *<– usa(b,e) (matching)*

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ?

chama(a,b) ? OK

depende(b, e) ? OK

Consultas

Exemplo

1. *chama(a,b). % fato*
2. *usa(b,e). % fato*
3. *depende(X, Y) :- chama(X,Y). % regra*
4. *depende(X,Y) :- usa(X, Y). % regra*
5. *depende(X, Y) :- chama(X, Z), depende(Z, Y). % regra recursiva*

Exemplo

?- depende(a,e). *<– 'a' depende de 'e'?*

depende(a, e) ? OK

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

3. O que querem dizer as seguintes consultas:

```
?- animal(X).          % Quem (X) são animais?
?- animal(peixe).      % peixe é um animal?
?- planta(alga).       % alga é uma planta?
?- planta(x).          % x é uma planta?
?- come(urso, peixe).  % urso come peixe?
?- come(X, peixe).     % Quem (X) come peixe?
?- come(X, Y).         % Quais animais (X) comem quais (Y)?
?- come(raposa, _).    % A raposa come algum animal?
?- come(_, coelho).    % Algum animal come o coelho?
```

Consultas

Exercício

4. Dado o banco abaixo, quais as respostas para as consultas:

a)animal(urso).	*h)animal(guaxinim).*	*o)come(peixe,peixinho).*
b)animal(peixe).	*i)planta(alga).*	*p)come(peixinho,alga).*
c)animal(peixinho).	*j)planta(grama).*	*q)come(guaxinim,peixe).*
d)animal(lince).	*k)come(urso,peixe).*	*r)come(raposa,coelho).*
e)animal(raposa).	*l)come(lince,veado).*	*s)come(coelho,grama).*
f)animal(coelho).	*m)come(urso,raposa).*	*t)come(veado,grama).*
g)animal(veado).	*n)come(urso,veado).*	*u)come(urso,guaxinim).*

?- planta(X). *X = alga; X = grama; No*
?- come(raposa,_). *Yes*
?- come(_, _). *Yes*
?- come(X, grama). *X = coelho ; X = veado; No*

Consultas

Exercício

4. Dado o banco abaixo, quais as respostas para as consultas:

a)animal(urso).	*h)animal(guaxinim).*	*o)come(peixe,peixinho).*
b)animal(peixe).	*i)planta(alga).*	*p)come(peixinho,alga).*
c)animal(peixinho).	*j)planta(grama).*	*q)come(guaxinim,peixe).*
d)animal(lince).	*k)come(urso,peixe).*	*r)come(raposa,coelho).*
e)animal(raposa).	*l)come(lince,veado).*	*s)come(coelho,grama).*
f)animal(coelho).	*m)come(urso,raposa).*	*t)come(veado,grama).*
g)animal(veado).	*n)come(urso,veado).*	*u)come(urso,guaxinim).*

?- planta(X).	*X = alga; X = grama; No*
?- come(raposa,_).	*Yes*
?- come(_, _).	*Yes*
?- come(X, grama).	*X = coelho ; X = veado; No*

Consultas

Exercício

4. Dado o banco abaixo, quais as respostas para as consultas:

a)animal(urso).	*h)animal(guaxinim).*	*o)come(peixe,peixinho).*
b)animal(peixe).	*i)planta(alga).*	*p)come(peixinho,alga).*
c)animal(peixinho).	*j)planta(grama).*	*q)come(guaxinim,peixe).*
d)animal(lince).	*k)come(urso,peixe).*	*r)come(raposa,coelho).*
e)animal(raposa).	*l)come(lince,veado).*	*s)come(coelho,grama).*
f)animal(coelho).	*m)come(urso,raposa).*	*t)come(veado,grama).*
g)animal(veado).	*n)come(urso,veado).*	*u)come(urso,guaxinim).*

?- planta(X).	*X = alga; X = grama; No*
?- come(raposa,_).	*Yes*
?- come(_, _).	*Yes*
?- come(X, grama).	*X = coelho ; X = veado; No*

Consultas

Exercício

4. Dado o banco abaixo, quais as respostas para as consultas:

a)animal(urso).	*h)animal(guaxinim).*	*o)come(peixe,peixinho).*
b)animal(peixe).	*i)planta(alga).*	*p)come(peixinho,alga).*
c)animal(peixinho).	*j)planta(grama).*	*q)come(guaxinim,peixe).*
d)animal(lince).	*k)come(urso,peixe).*	*r)come(raposa,coelho).*
e)animal(raposa).	*l)come(lince,veado).*	*s)come(coelho,grama).*
f)animal(coelho).	*m)come(urso,raposa).*	*t)come(veado,grama).*
g)animal(veado).	*n)come(urso,veado).*	*u)come(urso,guaxinim).*

?- planta(X).	*X = alga; X = grama; No*
?- come(raposa,_).	*Yes*
?- come(_, _).	*Yes*
?- come(X, grama).	*X = coelho ; X = veado; No*

Consultas

Exercício

4. Dado o banco abaixo, quais as respostas para as consultas:

a)animal(urso).	*h)animal(guaxinim).*	*o)come(peixe,peixinho).*
b)animal(peixe).	*i)planta(alga).*	*p)come(peixinho,alga).*
c)animal(peixinho).	*j)planta(grama).*	*q)come(guaxinim,peixe).*
d)animal(lince).	*k)come(urso,peixe).*	*r)come(raposa,coelho).*
e)animal(raposa).	*l)come(lince,veado).*	*s)come(coelho,grama).*
f)animal(coelho).	*m)come(urso,raposa).*	*t)come(veado,grama).*
g)animal(veado).	*n)come(urso,veado).*	*u)come(urso,guaxinim).*

?- planta(X).	*X = alga; X = grama; No*
?- come(raposa,_).	*Yes*
?- come(_, _).	*Yes*
?- come(X, grama).	*X = coelho ; X = veado; No*

Regras

Regras facilitam a execução de consultas e tornam um programa muito mais expressivo.

Uma cláusula Prolog é equivalente à uma fórmula em lógica de 1ª, então, em Prolog, existem os conectivos:

- :- = se, equivalente à implicação;
- , = e, equivalente à conjunção;
- ; = ou, equivalente à disjunção.

Exemplo

A fórmula: $A(x) \rightarrow B(x) \vee (C(x) \wedge D(x))$,
seria escrita em Prolog como: $a(x) :- b(X); (c(x), d(x))$.

Prolog não utiliza quantificadores explicitamente, porém, trata todas as regras como se elas estivessem universalmente quantificadas e usa $\sim EU$ (eliminação do universal).

Regras

Exemplo

Tem-se a base:

mulher(pam).	*homem(jim).*	*genitor(pam, bob).*
homem(tom).	*mulher(ann).*	*genitor(tom, bob).*
homem(bob).	*genitor(pat, jim).*	*genitor(tom, liz).*
mulher(liz).	*genitor(bob, pat).*	*genitor(bob, ann).*
mulher(pat).		

Como podemos especificar uma regra prole(x, y)., significando que "x é prole (filho ou filha) de y".

Prole (filho ou filha) é a relação inversa de Genitor (pai ou mãe), então: "x é prole de y, se y é genitor de x".

Regras

Exemplo

Tem-se a base:

mulher(pam).	*homem(jim).*	*genitor(pam, bob).*
homem(tom).	*mulher(ann).*	*genitor(tom, bob).*
homem(bob).	*genitor(pat, jim).*	*genitor(tom, liz).*
mulher(liz).	*genitor(bob, pat).*	*genitor(bob, ann).*
mulher(pat).		

Como podemos especificar uma regra prole(x, y)., significando que "x é prole (filho ou filha) de y".

Prole (filho ou filha) é a relação inversa de Genitor (pai ou mãe), então: "x é prole de y, se y é genitor de x".

Assim:

prole(X,Y) :- genitor(Y,X).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

```
prole(X,Y) :-
  genitor(Y,X).
```

Exemplo

```
?- prole(pam, bob).
```

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

```
prole(X,Y) :-
  genitor(Y,X).
```

Exemplo

```
?- prole(pam, bob). consulta...
```

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

prole(X,Y) :- *match*
genitor(Y,X).

Exemplo

?- prole(pam, bob).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

```
prole(X,Y) :-        Unifica [X/pam] e [Y/bob]
  genitor(Y,X).
```

Exemplo

```
?- prole(pam, bob).
```

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

prole(X,Y) :-
genitor(Y,X). *nova meta*

Exemplo

?- prole(pam, bob).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

prole(X,Y) :-
genitor(Y,X). *Unifica [X/pam] e [Y/bob]*

Exemplo

?- prole(pam, bob).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

prole(X,Y) :-
genitor(Y,X). *Encontra o fato genitor(pam,bob).*

Exemplo

?- prole(pam, bob).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

prole(X,Y) :-
genitor(Y,X). *Satisfaz a consulta.*

Exemplo

?- prole(pam, bob).

Regras

Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos.

Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma: CONCLUSÃO(+ARG) :- CONDIÇÃO1(+ARG) CONECTIVO CONDIÇÃO2(+ARG) ...

Utilizando *matching* Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta. Cada vez que um *matching* ocorre a satisfação da regra passa a ser a meta atual.

Exemplo

```
prole(X,Y) :-
  genitor(Y,X).
```

Exemplo

```
?- prole(pam, bob).
```

Regras

Exercício

5. Tem-se a base:

a)mulher(pam). *f)homem(jim).* *k)genitor(pam, bob).*
b)homem(tom). *g)mulher(ann).* *l)genitor(tom, bob).*
c)homem(bob). *h)genitor(pat, jim).* *m)genitor(tom, liz).* *a*
d)mulher(liz). *i)genitor(bob, pat).* *n)genitor(bob, ann).*
e)mulher(pat). *j)prole(X,Y) :- genitor(Y,X). % regra*

Qual a resposta para as consultas?

?- prole(_ , tom).

Yes

?- prole(tom, X).

X = bob ;

X = liz ;

No

Regras

Exercício

5. Tem-se a base:

a)mulher(pam). *f)homem(jim).* *k)genitor(pam, bob).*
b)homem(tom). *g)mulher(ann).* *l)genitor(tom, bob).*
c)homem(bob). *h)genitor(pat, jim).* *m)genitor(tom, liz).* *a*
d)mulher(liz). *i)genitor(bob, pat).* *n)genitor(bob, ann).*
e)mulher(pat). *j)prole(X,Y) :- genitor(Y,X). % regra*

Qual a resposta para as consultas?

?- prole(_ , tom).
Yes
?- prole(tom, X).
X = bob ;
X = liz ;
No

Regras

Exercício

5. Tem-se a base:

a)mulher(pam). *f)homem(jim).* *k)genitor(pam, bob).*
b)homem(tom). *g)mulher(ann).* *l)genitor(tom, bob).*
c)homem(bob). *h)genitor(pat, jim).* *m)genitor(tom, liz).*
d)mulher(liz). *i)genitor(bob, pat).* *n)genitor(bob, ann).*
e)mulher(pat). *j)prole(X,Y) :- genitor(Y,X). % regra*

Qual a resposta para as consultas?

?- prole(_ , tom).
Yes
?- prole(tom, X).
X = bob ;
X = liz ;
No

Regras

Exercício

5. Tem-se a base:

a)mulher(pam). *f)homem(jim).* *k)genitor(pam, bob).*
b)homem(tom). *g)mulher(ann).* *l)genitor(tom, bob).*
c)homem(bob). *h)genitor(pat, jim).* *m)genitor(tom, liz).* *a*
d)mulher(liz). *i)genitor(bob, pat).* *n)genitor(bob, ann).*
e)mulher(pat). *j)prole(X,Y) :- genitor(Y,X). % regra*

Qual a resposta para as consultas?

?- prole(_ , tom).
Yes
?- prole(tom, X).
X = bob ;
X = liz ;
No

Regras

Exercício

5. Tem-se a base:

a)mulher(pam).	*f)homem(jim).*	*k)genitor(pam, bob).*
b)homem(tom).	*g)mulher(ann).*	*l)genitor(tom, bob).*
c)homem(bob).	*h)genitor(pat, jim).*	*m)genitor(tom, liz).*
d)mulher(liz).	*i)genitor(bob, pat).*	*n)genitor(bob, ann).*
e)mulher(pat).	*j)prole(X,Y) :- genitor(Y,X). % regra*	

Qual a resposta para as consultas?

?- prole(_ , tom).

Yes

?- prole(tom, X).

X = bob ;

X = liz ;

No

Regras

Exemplo

Dada a base:

a)mulher(pam).
b)homem(tom).
c)homem(bob).
d)mulher(liz).
e)mulher(pat).
f)homem(jim).
g)mulher(ann).
h)genitor(pat, jim).
i)genitor(bob, pat).
j)prole(X,Y) :- genitor(Y,X). % regra
k)genitor(pam, bob).
l)genitor(tom, bob).
m)genitor(tom, liz).
n)genitor(bob, ann).

Descreva regras para as seguintes relações:

- *mae(x,y), significando "x é mãe de y";*
- *avos(X,Z), significando "x é avô/avó de y".*

mae(X,Y) :- genitor(X,Y), mulher(X).
avos(X,Z) :- genitor(X,Y), genitor(Y,Z).

Regras

Exemplo

Dada a base:

a)mulher(pam). *f)homem(jim).* *k)genitor(pam, bob).*
b)homem(tom). *g)mulher(ann).* *l)genitor(tom, bob).*
c)homem(bob). *h)genitor(pat, jim).* *m)genitor(tom, liz).*
d)mulher(liz). *i)genitor(bob, pat).* *n)genitor(bob, ann).*
e)mulher(pat). *j)prole(X,Y) :- genitor(Y,X). % regra*

Descreva regras para as seguintes relações:

- *mae(x,y), significando "x é mãe de y";*
- *avos(X,Z), significando "x é avô/avó de y".*

mae(X,Y) :- genitor(X,Y), mulher(X).
avos(X,Z) :- genitor(X,Y), genitor(Y,Z).

Regras

Exemplo

Dada a base:

a)mulher(pam).
b)homem(tom).
c)homem(bob).
d)mulher(liz).
e)mulher(pat).
f)homem(jim).
g)mulher(ann).
h)genitor(pat, jim).
i)genitor(bob, pat).
j)prole(X,Y) :- genitor(Y,X). % regra
k)genitor(pam, bob).
l)genitor(tom, bob).
m)genitor(tom, liz).
n)genitor(bob, ann).

Descreva regras para as seguintes relações:

- *mae(x,y), significando "x é mãe de y";*
- *avos(X,Z), significando "x é avô/avó de y".*

mae(X,Y) :- genitor(X,Y), mulher(X).
avos(X,Z) :- genitor(X,Y), genitor(Y,Z).

Regras

Exercício

6. Tem-se a base:

a)animal(urso).
b)animal(peixe).
c)animal(peixinho).
d)animal(lince).
e)animal(raposa).
f)animal(coelho).
g)animal(veado).
h)animal(guaxinim).
i)planta(alga).
j)planta(grama).
k)come(urso,peixe).
l)come(lince,veado).
m)come(urso,raposa).
n)come(urso,veado).
o)come(peixe,peixinho).
p)come(peixinho,alga).
q)come(guaxinim,peixe).
r)come(raposa,coelho).
s)come(coelho,grama).
t)come(veado,grama).
u)come(urso,guaxinim).

a

Descreva uma regra para determinar quais animais são presas.

presa(X) :- come(_, X), animal(X).

Regras

Exercício

6. Tem-se a base:

a)animal(urso).
b)animal(peixe).
c)animal(peixinho).
d)animal(lince).
e)animal(raposa).
f)animal(coelho).
g)animal(veado).
h)animal(guaxinim).
i)planta(alga).
j)planta(grama).
k)come(urso,peixe).
l)come(lince,veado).
m)come(urso,raposa).
n)come(urso,veado).
o)come(peixe,peixinho).
p)come(peixinho,alga).
q)come(guaxinim,peixe).
r)come(raposa,coelho).
s)come(coelho,grama).
t)come(veado,grama).
u)come(urso,guaxinim).

a

Descreva uma regra para determinar quais animais são presas.

presa(X) :- come(_, X), animal(X).

Regras recursivas

A recursão é um dos elementos mais importantes da linguagem Prolog, este conceito permite a resolução de problemas significativamente complexos de maneira relativamente simples.

Um regra é recursiva se sua condição depende dela mesma, tal como: $a(X) :- b(X), a(X)$.

A importância do uso de recursão pode ser ilustrada na implementação da relação *descendente(x, y)*, significando que "x é descendente de y".

Um conjunto de regras é denominado **procedimento**.

Regras recursivas

Exemplo

Implementação 1 (sem recursão):

```
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...
```

Exemplo

Implementação 2 (com recursão):

```
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).
```

Regras recursivas

Exemplo

Implementação 1 (sem recursão):

```
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...
```

Exemplo

Implementação 2 (com recursão):

```
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).
```

Regras recursivas

Exemplo

Implementação 1 (sem recursão):

```
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...
```

Exemplo

Implementação 2 (com recursão):

```
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).
```

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

A implementação 1 é limitada e trabalhosa.

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

A implementação 1 é limitada e trabalhosa.

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

A implementação 1 é limitada e trabalhosa.

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

Regras recursivas

Exemplo

Implementação 1 (sem recursão):
descendente(X,Z) :- genitor(X,Z). % caso base
descendente(X,Z) :- genitor(X,Y), genitor(Y,Z).
descendente(X,Z) :- genitor(X,Y), genitor(Y,W), genitor(W,Z).
...

A implementação 1 é limitada e trabalhosa.

Exemplo

Implementação 2 (com recursão):
descendente(X,Z) :- genitor(X,Z).
descendente(X,Z) :- genitor(X,Y), descendente(Y,Z).

A implementação 2 é não tem limites e utiliza bem menos regras que 1.

Regras recursivas

Exercício

7. Tem-se a base:

a)animal(urso).
b)animal(peixe).
c)animal(peixinho).
d)animal(lince).
e)animal(raposa).
f)animal(coelho).
g)animal(veado).
h)animal(guaxinim).
i)planta(alga).
j)planta(grama).
k)come(urso,peixe).
l)come(lince,veado).
m)come(urso,raposa).
n)come(urso,veado).
o)come(peixe,peixinho).
p)come(peixinho,alga).
q)come(guaxinim,peixe).
r)come(raposa,coelho).
s)come(coelho,grama).
t)come(veado,grama).
u)come(urso,guaxinim).

a

Descreva uma regra para determinar quais animais pertencem a cadeia alimentar de outro.

```
cadeia-alimentar(X, Z) :- come(X, Z). % caso base
cadeia-alimentar(X, Z) :- come(X, Y), cadeia-alimentar(Y, Z).
```

Regras recursivas

Exercício

7. Tem-se a base:

a)animal(urso).
b)animal(peixe).
c)animal(peixinho).
d)animal(lince).
e)animal(raposa).
f)animal(coelho).
g)animal(veado).
h)animal(guaxinim).
i)planta(alga).
j)planta(grama).
k)come(urso,peixe).
l)come(lince,veado).
m)come(urso,raposa).
n)come(urso,veado).
o)come(peixe,peixinho).
p)come(peixinho,alga).
q)come(guaxinim,peixe).
r)come(raposa,coelho).
s)come(coelho,grama).
t)come(veado,grama).
u)come(urso,guaxinim).

a

Descreva uma regra para determinar quais animais pertencem a cadeia alimentar de outro.

```
cadeia-alimentar(X, Z) :- come(X, Z). % caso base
cadeia-alimentar(X, Z) :- come(X, Y), cadeia-alimentar(Y, Z).
```

PARTE III

Listas

Uma lista é uma seqüência ordenada de elementos de qualquer tipo de dados de Prolog.

Os elementos contidos em uma lista devem ser separados por vírgulas, e precisam estar entre colchetes. Existem notações alternativas, porém, esta é a mais simples.

Exemplo

São exemplos de listas:
a) *[pam, liz, pat, ann, tom, bob, jim]*
b) *[1, 2, 3, 4, 5]*
c) *[a, [b, c], d, e], onde [b,c] é o segundo elemento da lista*

Listas

Listas podem ser de dois tipos:

- vazias - quando não contém nenhum elemento, representadas por [];
- não-vazias - quando contém ao menos um elemento.

Listas não vazias possuem duas partes, são elas:

- cabeça - corresponde ao primeiro elemento da lista;
- cauda - corresponde aos elementos restantes da lista.

Exemplo

a) *[pam, liz, pat, ann, tom, bob, jim]*
pam é a cabeça, [liz, pat, ann, tom, bob, jim] é a cauda.
b) *[1, 2, 3, 4, 5]*
1 é a cabeça, [2, 3, 4, 5] é a cauda
c) *[a, [b, c], d, e]*
a é a cabeça, [[b, c], d, e] é a cauda

Podemos utilizar a notação [Cabeca | Cauda] para separar as partes de uma lista.

Listas

Para se trabalhar com uma lista deve-se utilizar a notação [Cabeca | Cauda] para buscar um elemento ou propriedade através de recursão.

Exemplo

Verifique se um elemento X pertence a uma lista L:
pertence(X, [X | _]). % caso base
pertence(X, [_ | L]) :- pertence(X, L). (regra recursiva)

Exercício

8. Implemente regras que verifiquem se um elemento é o último elemento de uma lista.
final(X, [X]). % caso base
final(X, [_ | L]) :- final(X, L).

Listas

Para se trabalhar com uma lista deve-se utilizar a notação [Cabeca | Cauda] para buscar um elemento ou propriedade através de recursão.

Exemplo

Verifique se um elemento X pertence a uma lista L:
pertence(X, [X | _]). % caso base
pertence(X, [_ | L]) :- pertence(X, L). (regra recursiva)

Exercício

8. Implemente regras que verifiquem se um elemento é o último elemento de uma lista.
final(X, [X]). % caso base
final(X, [_ | L]) :- final(X, L).

Aritmética

Prolog é mais indicada para resolução de problemas simbólicos, mas também oferece suporte à aritmética.

Podemos utilizar duas notações para representar expressões em Prolog:

- Infixa: $2 * a + b * c$
- Prefixa: $+(*(2, a), *(b, c))$

Prolog trata as representações de forma equivalente, pois, internamente utiliza árvores para representar expressões. Assim, basta mudar a ordem de caminhamento para obter uma ou outra forma.

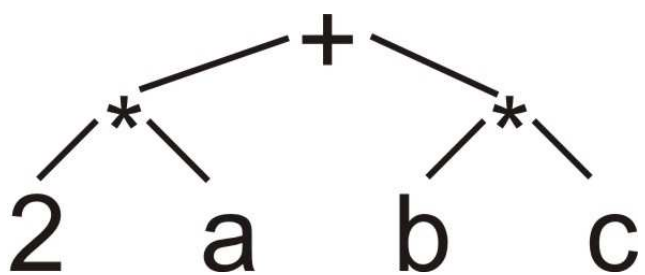

Figura: Árvore para a expressão $2 * a + b * c$

Aritmética

São oferecidos diversos predicados para operações aritméticas, alguns destes são:

- $+, -, *, /$, para realizar soma, subtração, multiplicação e divisão, respectivamente;
- **is**, atribui uma expressão numérica à uma variável;
- mod, para obter o resto da divisão;
- $\widehat{}$, para potenciação;
- cos, sin, tan, função cosseno, seno e tangente, respectivamente;
- exp, exponencial;
- ln, log, logaritmo natural e logaritmo;
- sqrt, raiz quadrada.

Predicados de conversão, tais como:

- integer(X), converte X para inteiro;
- float(X), converte X para ponto flutuante.

Aritmética

Prolog também possui predicados para comparação, os operadores são:

- $>$, maior que;
- $<$, menor que;
- $>=$, maior ou igual;
- $=<$, menor ou igual;
- $=:=$, igual;
- $=/=$, diferente.

Os operadores $=$ e $=:=$, realizam diferentes tipos de comparação:

- $=$, checa se os "objetos" são iguais, ou atribui valores para as variáveis;
- $=:=$, avalia se os valores são iguais.

Exemplo

?- 1 + 2 = 2 + 1. *No*
?- 1 + 2 =:= 2 + 1. *Yes*
?- 1 + A = B + 2. *A = 2 B = 1 ; No*
?- A =:= 1. *ERROR*

Aritmética

Prolog também possui predicados para comparação, os operadores são:

- $>$, maior que;
- $<$, menor que;
- $>=$, maior ou igual;
- $=<$, menor ou igual;
- $=:=$, igual;
- $=/=$, diferente.

Os operadores $=$ e $=:=$, realizam diferentes tipos de comparação:

- $=$, checa se os "objetos" são iguais, ou atribui valores para as variáveis;
- $=:=$, avalia se os valores são iguais.

Exemplo

?- 1 + 2 = 2 + 1. *No*
?- 1 + 2 =:= 2 + 1. *Yes*
?- 1 + A = B + 2. *A = 2 B = 1 ; No*
?- A =:= 1. *ERROR*

Aritmética

Prolog também possui predicados para comparação, os operadores são:

- $>$, maior que;
- $<$, menor que;
- $>=$, maior ou igual;
- $=<$, menor ou igual;
- $=:=$, igual;
- $=/=$, diferente.

Os operadores $=$ e $=:=$, realizam diferentes tipos de comparação:

- $=$, checa se os "objetos" são iguais, ou atribui valores para as variáveis;
- $=:=$, avalia se os valores são iguais.

Exemplo

?- 1 + 2 = 2 + 1. *No*
?- 1 + 2 =:= 2 + 1. *Yes*
?- 1 + A = B + 2. *A = 2 B = 1 ; No*
?- A =:= 1. *ERROR*

Aritmética

Prolog também possui predicados para comparação, os operadores são:

- $>$, maior que;
- $<$, menor que;
- $>=$, maior ou igual;
- $=<$, menor ou igual;
- $=:=$, igual;
- $=/=$, diferente.

Os operadores $=$ e $=:=$, realizam diferentes tipos de comparação:

- $=$, checa se os "objetos" são iguais, ou atribui valores para as variáveis;
- $=:=$, avalia se os valores são iguais.

Exemplo

?- 1 + 2 = 2 + 1. No
?- 1 + 2 =:= 2 + 1. Yes
?- 1 + A = B + 2. A = 2 B = 1 ; No
?- A =:= 1. ERROR

Aritmética

Prolog também possui predicados para comparação, os operadores são:

- $>$, maior que;
- $<$, menor que;
- $>=$, maior ou igual;
- $=<$, menor ou igual;
- $=:=$, igual;
- $=/=$, diferente.

Os operadores $=$ e $=:=$, realizam diferentes tipos de comparação:

- $=$, checa se os "objetos" são iguais, ou atribui valores para as variáveis;
- $=:=$, avalia se os valores são iguais.

Exemplo

?- 1 + 2 = 2 + 1. No
?- 1 + 2 =:= 2 + 1. Yes
?- 1 + A = B + 2. A = 2 B = 1 ; No
?- A =:= 1. ERROR

Aritmética

Exemplo

Implemente um programa para calcular o fatorial de um número:

```
fatorial(0,1). % caso base
fatorial(N,F) :-
  N > 0,
  N1 is N-1,
  factorial(N1,F1),
  F is N * F1.
```

Exercício

9. O que o programa abaixo faz?

```
checagem(N, propriedade1) :- N > 0.
checagem(N, propriedade2) :- N < 0.
checagem(0, propriedade3).
```

Checa se N é positivo (propriedade1), negativo (propriedade2) ou zero (propriedade3)

Aritmética

Exemplo

Implemente um programa para calcular o fatorial de um número:

```
fatorial(0,1). % caso base
fatorial(N,F) :-
  N > 0,
  N1 is N-1,
  factorial(N1,F1),
  F is N * F1.
```

Exercício

9. O que o programa abaixo faz?

```
checagem(N, propriedade1) :- N > 0.
checagem(N, propriedade2) :- N < 0.
checagem(0, propriedade3).
```

Checa se N é positivo (propriedade1), negativo (propriedade2) ou zero (propriedade3)

Aritmética

Exemplo

Implemente um programa para calcular o fatorial de um número:

```
fatorial(0,1). % caso base
fatorial(N,F) :-
  N > 0,
  N1 is N-1,
  factorial(N1,F1),
  F is N * F1.
```

Exercício

9. O que o programa abaixo faz?

```
checagem(N, propriedade1) :- N > 0.
checagem(N, propriedade2) :- N < 0.
checagem(0, propriedade3).
```

Checa se N é positivo (propriedade1), negativo (propriedade2) ou zero (propriedade3)

Aritmética

Exemplo

Implemente um programa para calcular o fatorial de um número:

```
fatorial(0,1). % caso base
fatorial(N,F) :-
  N > 0,
  N1 is N-1,
  factorial(N1,F1),
  F is N * F1.
```

Exercício

9. O que o programa abaixo faz?

```
checagem(N, propriedade1) :- N > 0.
checagem(N, propriedade2) :- N < 0.
checagem(0, propriedade3).
```

Checa se N é positivo (propriedade1), negativo (propriedade2) ou zero (propriedade3)

Corte

Prolog realiza busca exaustiva para responder uma consulta, ou seja, todas as cláusulas existentes são analisadas.

- Pró: todas as soluções possíveis são obtidas.
- Contra: uma busca pode ser extremamente ineficiente.

Para minimizar os casos de ineficiência, o operador de corte (!) pode ser utilizado. Este operador impede que busca posteriores sejam realizadas.

Exemplo

Sejam fornecidos dois números x e y, implemente regras que informem qual destes números é o máximo:

maximo(X,Y,X) :- X >= Y.

maximo(X,Y,Y) :- X < Y.

Porém, as duas regras serão testadas sempre...

maximo(X,Y,X) :- X >= Y, !. % impede que a regra seguinte seja utilizada em caso de sucesso.

maximo(X,Y,Y) :- X < Y.

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- a(X), b(X).
2.p(X) :- c(X).
?- p(1).
?- p(2).
```

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- a(X), b(X).
2.p(X) :- c(X).
?- p(1).  Yes
?- p(2).
```

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- a(X), b(X).
2.p(X) :- c(X).
?- p(1).  Yes
?- p(2).  No
```

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- c(X).
2.p(X) :- a(X), b(X).
?- p(1).
?- p(2).
```

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- c(X).
2.p(X) :- a(X), b(X).
?- p(1).  Yes
?- p(2).
```

Corte

O operador de corte é extremamente útil, porém, deve ser utilizado com cuidado.

Um programa sem cortes pode ter a ordem das cláusulas alterada e este irá realizar a mesma computação. Já um programa com cortes pode ter seu significado alterado caso a ordem das regras seja modificada.

Exemplo

O programa abaixo, sem cortes, não teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1.p(X) :- c(X).
2.p(X) :- a(X), b(X).
?- p(1). Yes
?- p(2). No
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- a(X) , ! , b(X).
2. p(X) :- c(X).
?- p(1).
?- p(2).
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0., !.
checagem(N, negativo) :- N < 0., !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- a(X) , ! , b(X).
2. p(X) :- c(X).
?- p(1). No
?- p(2).
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0. , !.
checagem(N, negativo) :- N < 0. , !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- a(X) , ! , b(X).
2. p(X) :- c(X).
?- p(1). No
?- p(2). No
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0., !.
checagem(N, negativo) :- N < 0., !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- c(X).
2. p(X) :- a(X) , ! , b(X).
?- p(1).
?- p(2).
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0., !.
checagem(N, negativo) :- N < 0., !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- c(X).
2. p(X) :- a(X) , ! , b(X).
?- p(1).  Yes
?- p(2).
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0., !.
checagem(N, negativo) :- N < 0., !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- c(X).
2. p(X) :- a(X) , ! , b(X).
?- p(1). Yes
?- p(2). No
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0. , !.
checagem(N, negativo) :- N < 0. , !.
checagem(0, zero).
```

Corte

Exemplo

Porém, utilizando cortes, o programa abaixo, teria seu significado alterado caso a ordem das regras fosse modificada.

```
a(1).
b(2).
c(1).
1. p(X) :- c(X).
2. p(X) :- a(X) , ! , b(X).
?- p(1). Yes
?- p(2). No
```

Exercício

10. Melhore o programa abaixo tornando-o mais eficiente.

```
checagem(N, positivo) :- N > 0, !.
checagem(N, negativo) :- N < 0, !.
checagem(0, zero).
```

Sumário

- Fatos, consultas...
- Regras, regras recursivas...
- Listas...
- Aritmética...
- Corte...