PLP: Módulo 2

Programação Lógica e
a Linguagem PROLOG

Ementa

- Introdução à Programação Lógica (PROLOG)

Referências de PROLOG

Livro

The Art of Prolog, Second Edition: Advanced Programming Techniques (Logic Programming) by Leon Sterling and Ehud Shapiro

Material

Vamos adotar SWI Prolog

Interpretador – SWI Prolog
http://www.swi-prolog.org/

Manual
http://www.swi-prolog.org/pldoc/refman/

Tutorial (muito bom)
http://lpn.swi-prolog.org

Conceitos Básicos

Características do Prolog

- Um programa Prolog é um coleção de procedimentos (simples).
- Dados são procedimentos simples – fatos ou regras.
- Regras suportam recursividade.
- Prolog não tem comandos de atribuição.
- Prolog não faz distinção entre parâmetros reais e parâmetros formais
- Prolog tem tipos dinâmicos
- O escopo de uma variável é a linha (regra ou fato) que a contém.
- A idéia em Prolog é dizer ao computador o quê queremos computar e não como fazer (DECLARATIVA).
- O algoritmo de unificação (solução) é sempre o mesmo.

Programação em Lógica

Programa é um conjunto de axiomas (fatos e regras)

A computação é uma prova construtiva a partir do programa de uma meta estabelecida pelo usuário.

A solução depende do Programa e da Meta.

Fatos

Fatos são as declarações mais simples do PROLOG

Fatos é uma meio de declarar relações que existem entre objetos:

```
cidade(salvador).
cidade(brasilia).
pais(brasil).
estado(bahia).
capital(brasilia,brasil).
capital(salvador, bahia).
```

Variáveis

As relações podem conter variáveis.

```
gosta(vinho,X).
```

No exemplo, a relação indicam que qualquer `objeto` gosta de `vinho`. Por exemplo:

`manoel` gosta de vinho.

`3` gosta de vinho.

`corona_virus` gosta de vinho.

Variáveis (1)

- Variáveis em Prolog têm escopo na declaração (fato ou regra).
- Elas representam uma instancia não especificada de uma única entidade.
- Não há o conceito de ocupação de uma posição de memória.

Variáveis (2)

- Prolog tem tipagem dinâmica.
- Variáveis começam com uma letra maiúscula.
- Elas podem ser usadas em consultas, em fatos e regras.

Outros exemplos

```
pais(X).

bonita(X).

capital(X, brasil).

capital(brasilia, X).

gemeos_univitelineos(X,X).
```

X tem uma instância diferente em cada caso. Daria no mesmo usar cinco variáveis diferentes.

~~Relação~~ => Termos

Termos é a estrutura de dados básica do Prolog. Eles são generalização do conceito de constante, variável e relação visto anteriormente. Eles têm a forma:

Variáveis e constantes são termos.

`functor(termo`$_1$`,termo`$_2$`,...,termo`$_N$`)`
também é um termo.

Notem que a definição é recursiva.

Assista:
https://www.youtube.com/watch?v=TUjQqvCTwjQ

Termos

```
quente(leite).

capital(brasil,X).

tree(tree(nil,3,nil),5,R).
```

Regras

Regras permitem a definição de novas relações a partir de relações existentes.

Elas têm a forma:

B :- $A_1, A_2, \ldots, A_N$.

Indicando que a meta *B*, a cabeça da regra, é verdade se a conjunção de metas $A_1, A_2, \ldots, A_N$, o corpo da regra, for verdadeira. Ou seja:

Se $A_1 \wedge A_2 \wedge \ldots \wedge A_N$ então B

Regras

As relações, e consequentemente as regras, podem conter variáveis.

Regras

```
progenitor(X,Y) :- pai(X,Y).
progenitor(X,Y) :- mae(X,Y).

filho(Y,X) :- progenitor(X,Y), homem(Y).
filha(Y,X) :- progenitor(X,Y), mulher(Y).

irmaos(X,Y) :-
progenitor(Z,X),progenitor(Z,Y), X\==Y.

irmao(X,Y) :- irmaos(X,Y), homem(X).
irma(X,Y) :- irmaos(X,Y), mulher(X).
```

Consulta

A computação em Prolog é disparada por uma consulta.

Consultas parecem muito com fatos, elas têm o mesmo formato, e são distinguidas pelo contexto:

```
?- estado(bahia).
```

Vamos brincar um pouco

Crie uma tabela com fatos sobre descendência, e a regras que vimos antes. Adicione por exemplo:

```
homem(manoel).
homem(renato).
mulher(conca).
pai(renato, manoel).
mae(conca, Manoel).
```

Tente fazer algumas consultas básicas:

```
?- homem(X).
?- filho(X,renato).
```

Consulta

Uma consulta estabelece uma meta inicial, que o algoritmo de unificação tentará resolver por substituição de variáveis nos fatos e nas regras.

Durante a unificação, regras podem demandar a criação de nova metas. Por exemplo, se minha meta inicial requerer a solução do termo `B`. A regra abaixo criará metas para solucionar os termos A_1 , A_2 e A_3.

```
B :- A1,A2,A3.
```

Consulta Conjuntivas

Consultas podem ser conjuntivas. Por exemplo, a consulta “quais países têm capitais bonitas?” pode ser escrita na forma:

```
?- pais(X),capital(Y,X),bonita(Y).
```

Resolvendo Consultas

Consultas são resolvidas pela máquina Prolog por unificação ou substituição de termos.

A consulta é uma meta a ser resolvida utilizando o programa como ponto de partida para uma prova construtiva.

Resolvendo a Consulta

Para resolver uma consulta conjuntiva $A_1, A_2, \ldots, A_N$
sobre um programa *P*

a máquina Prolog procura por substituições Θ

que levem a instâncias $A_1\Theta, A_2\Theta, \ldots, A_N\Theta$

a fatos em *P*.

Substituição / Unificação

Uma substituição é um conjunto finito (possivelmente vazio) de pares da forma *Xi=ti*, aonde *Xi* é uma variável e *ti* é um termo, e *Xi≠Xj* para todo *i≠j*, e *Xi* não ocorre em *tj*, para qualquer *i* e *j*.

Por exemplo, aplicação da substituição `{X,salvador}`

ao termo `capital(X, brasil)`

produz o termo `capital(salvador, brasil)`

Um Interpretador Simples

Entrada: Uma meta **M** e um Programa **P**
Saída: **true** se **M** é uma conseqüência lógica de **P**, **false** caso contrário.

Algoritmo:

inicialize o resolvente com M
while o resolvente não for vazio ***do***
pegue uma meta A do resolvente
pegue uma instância de uma cláusula
A :- $B_1, B_2, \ldots, B_N$ de P e substitua
A por $B_1, B_2, \ldots, B_N$ no resolvente
saia do loop se não houver mais metas
ou cláusulas
if resolvente está vazio retorne true
else retorne false

Revisitando as Características do Prolog

- Um programa Prolog é um coleção de procedimentos (simples).
- Dados são procedimentos simples – fatos ou regras.
- Regras suportam recursividade.
- Prolog não tem comandos de atribuição.
- Prolog não faz distinção entre parâmetros reais e parâmetros formais
- Prolog tem tipo dinâmicos
- O escopo de uma variável é a linha (regra ou fato) que a contém.
- A idéia em Prolog é dizer ao computador o quê queremos computar e não como fazer (DECLARATIVA).
- O algoritmo de unificação (solução) é sempre o mesmo.

Executando Prolog

Um Exemplo - Fatos

```
homem(john).                % F1
homem(jim).                 % F2
homem(jerry).               % F3
mulher(sue).                % F4
mulher(carol).              % F5
mulher(alice).              % F6

pai(john, jerry).           % F7
pai(john, carol).           % F8
mae(alice, jerry).          % F9
mae(alice, carol).          % F10
mae(sue, ted).              % F11
pai(jerry, ted).            % F12
mae(carol, bob).            % F13
pai(jim, bob).              % F14
```

Regras

```
progenitor(X,Y) :- pai(X,Y).                          % R1
progenitor(X,Y) :- mae(X,Y).                          % R2

filho(Y,X) :- progenitor(X,Y), homem(Y).              % R3
filha(Y,X) :- progenitor(X,Y), mulher(Y).             % R4

irmaos(X,Y) :-
      progenitor(Z,X),progenitor(Z,Y), X\==Y.         % R5

irmao(X,Y) :- irmaos(X,Y), homem(X).                  % R6
irma(X,Y) :- irmaos(X,Y), mulher(X).                  % R7
```

Note o condicional para diferente \==

Consulta

```
?- irmao(jerry, carol).
```

O quê acontece?

```
?- irmao(jerry, carol).
```

Consulta

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).              % R1

progenitor(X,Y) :- mae(X,Y).              % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                   % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                  % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                     % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                   % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                   % R7
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).
```

REGRA 6
{jerry/X, carol/Y}

```
?- irmaos(jerry, carol),
   homem(jerry).
```

```
?- irmao(jerry, carol).        REGRA 6

?- irmaos(jerry, carol),
   homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5 (jerry/X, carol/Y)
   homem(jerry).

?- progenitor(Z, jerry),
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),         REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),         REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),         REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                 FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(john,jerry),
   progenitor(john, carol),
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),         REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                 FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).          REGRA 6

?- irmaos(jerry, carol),         REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),         REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                 FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- progenitor(john, carol),
   jerry \== carol,
   homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).   REGRA 6

?- irmaos(jerry, carol),  REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),  REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),          FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),   REGRA 1 {john/X, carol/Y}
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).   REGRA 6

?- irmaos(jerry, carol),  REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),  REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),          FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),   REGRA 1 {john/X, carol/Y}
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),
   jerry \== carol,
   homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).                REGRA 6

?- irmaos(jerry, carol),               REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),               REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                       FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),            REGRA 1
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),                   FATO 8
   jerry \== carol,
   homem(jerry).

?- true,
   jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).              REGRA 6

?- irmaos(jerry, carol),             REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),             REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                     FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),          REGRA 1
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),                 FATO 8
   jerry \== carol,
   homem(jerry).

?- jerry \== carol,
   homem(jerry).
```

```
?- irmao(jerry, carol).                REGRA 6

?- irmaos(jerry, carol),               REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),               REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                       FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),            REGRA 1
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),                   FATO 8
   jerry \== carol,
   homem(jerry).

?- jerry \== carol,                    OPERAÇÃO PRIMITIVA
   homem(jerry).

?- true,
   homem(jerry).
```

```
?- irmao(jerry, carol).              REGRA 6

?- irmaos(jerry, carol),             REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),             REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                     FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),          REGRA 1
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),                 FATO 8
   jerry \== carol,
   homem(jerry).

?- jerry \== carol,                  OPERAÇÃO PRIMITIVA
   homem(jerry).

?- homem(jerry).
```

Consulta Fatos e Regras

```
homem(john).          % F1
homem(jim).           % F2
homem(jerry).         % F3

mulher(sue).          % F4
mulher(carol).        % F5
mulher(alice).        % F6

pai(john, jerry).     % F7
pai(john, carol).     % F8
pai(jerry, ted).      % F9
pai(jim, bob).        % F10

mae(alice, jerry).    % F11
mae(alice, carol).    % F12
mae(sue, ted).        % F13
mae(carol, bob).      % F14
```

```
progenitor(X,Y) :- pai(X,Y).                % R1

progenitor(X,Y) :- mae(X,Y).                % R2

filho(Y,X) :- progenitor(X,Y),
              homem(Y).                     % R3

filha(Y,X) :- progenitor(X,Y),
              mulher(Y).                    % R4

irmaos(X,Y) :- progenitor(Z,X),
               progenitor(Z,Y),
               X\==Y.                       % R5

irmao(X,Y) :- irmaos(X,Y),
              homem(X).                     % R6

irma(X,Y) :- irmaos(X,Y),
             mulher(X).                     % R7
```

```
?- irmao(jerry, carol).              REGRA 6

?- irmaos(jerry, carol),             REGRA 5
   homem(jerry).

?- progenitor(Z, jerry),             REGRA 1
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- pai(Z,jerry),                     FATO 7 {john/Z}
   progenitor(Z, carol),
   jerry \== carol,
   homem(jerry).

?- true,
   progenitor(john, carol),          REGRA 1
   jerry \== carol,
   homem(jerry).

?- pai(john, carol),                 FATO 8
   jerry \== carol,
   homem(jerry).

?- jerry \== carol,                  OPERAÇÃO PRIMITIVA
   homem(jerry).

?- homem(jerry).                     FATO 3
```

True.

Acompanhando a Execução com *o trace*

```
?- trace.
true.

?- [trace] irmaos(jerry,carol).
   Call: (10) irmaos(jerry, carol) ? creep
   Call: (11) progenitor(_9800, jerry) ? creep
   Call: (12) pai(_9844, jerry) ? creep
   Exit: (12) pai(john, jerry) ? creep
   Exit: (11) progenitor(john, jerry) ? creep
   Call: (11) progenitor(john, carol) ? creep
   Call: (12) pai(john, carol) ? creep
   Exit: (12) pai(john, carol) ? creep
   Exit: (11) progenitor(john, carol) ? creep
   Call: (11) jerry\==carol ? creep
   Exit: (11) jerry\==carol ? creep
   Exit: (10) irmaos(jerry, carol) ? creep
true
```

trace ou *spy* começa

notrace e *nodebug* para.

Acompanhando a Execução com *spy*

```
?- spy(irmaos).
% Spy point on irmaos/2
true.

[debug]  ?- irmaos(jerry,carol).
 * Call: (10) irmaos(jerry, carol) ? creep
   Call: (11) progenitor(_11384, jerry) ? creep
   Call: (12) pai(_11428, jerry) ? creep
   Exit: (12) pai(john, jerry) ? creep
   Exit: (11) progenitor(john, jerry) ? creep
   Call: (11) progenitor(john, carol) ? creep
   Call: (12) pai(john, carol) ? creep
   Exit: (12) pai(john, carol) ? creep
   Exit: (11) progenitor(john, carol) ? creep
   Call: (11) jerry\==carol ? creep
   Exit: (11) jerry\==carol ? creep
 * Exit: (10) irmaos(jerry, carol) ? creep
true
```

trace ou *spy* começa

notrace e *nodebug* para.

Continuando a busca

Lembram que tivemos duas opções para progenitor (por duas vezes)? Isto significa que ainda temos três caminhos de solução ainda não explorados.

Para encerrar a busca por uma solução utilizamos no **ponto “.”** no console.

Para continuar explorando os caminhos não explorados utilizamos o **ponto e vírgula “;”** no console.

Algumas buscas falham, outras são bem sucedidas.

Tente a mesma consulta novamente e use o ponto e vírgula, com e sem *trace*.

Aritmética e Atribuições

Aritmética e Atribuições

```
media(A, B, C) :-
        C is (A+B)/2.
```

Note o uso de atribuição temporária is

Fatorial Exemplos Numéricos

```
fatorial1(0,1).
fatorial1(N,F) :-
   N>0,
   N1 is N-1,
   fatorial1(N1,F1),
   F is N * F1.
```

Fatorial Exemplos Numéricos

```
fatorial1(0,1).
fatorial1(N,F) :-
   N>0,
   N1 is N-1,
   fatorial1(N1,F1),
   F is N * F1.

fatorial2(0,F,F).
fatorial2(N,A,F) :-
    N>0,
    A1 is N*A,
    N1 is N -1,
    fatorial2(N1,A1,F).
```

Qual a diferença?

fatorial2(4,1,F)

Fatorial Exemplos Numéricos

```
fatorial1(0,1).
fatorial1(N,F) :-
   N>0,
   N1 is N-1,
   fatorial1(N1,F1),
   F is N * F1.

fatorial2(0,F,F).
fatorial2(N,A,F) :-
    N>0,
    A1 is N*A,
    N1 is N -1,
    fatorial2(N1,A1,F).
```

```
fatorial3(N,F) :-
      factorial(N,1,F).
```

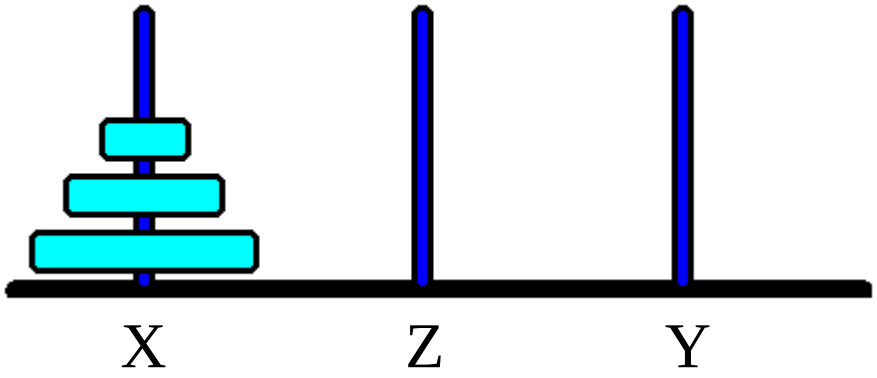
Recursividade Fatorial Simbólico

```
fatorial3(0, 1).

fatorial3(N, N*Z) :-
            N>0,
            U is N-1,
            fatorial3(U, Z).
```

Buscas Interessantes

Torre de Hanoy



Passe os discos da torre da esquerda para a torre da direita sem jamais colocar um disco sobre um menor que ele.

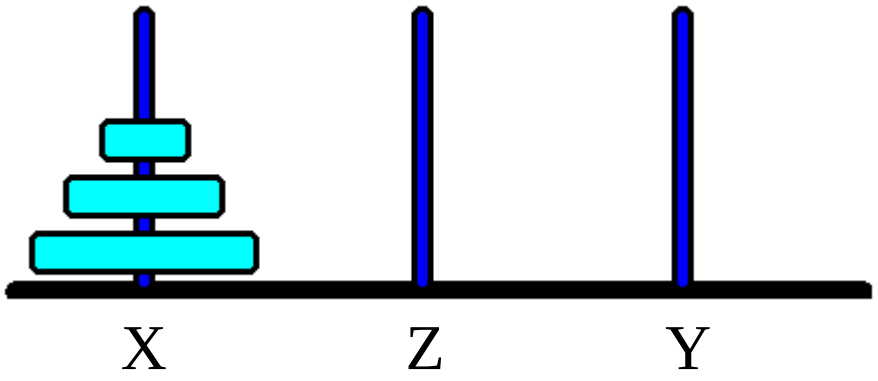
Qual a solução recursiva?

Torre de Hanoy Solução

Mova(N discos, de X, para Y, passando por Z)

Se N=1,
Mova(1 disco, de X, para Y, passando por ---).

Se N>1,
Mova(N-1 discos, de X, para Z, passando por Y),
Mova(1 disco, de X, para Y, passando por ---),
Mova(N-1 discos, de Z, para Y, passando por X).



Torre de Hanoy

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
```

- Note o uso da impressão.

- Underscore casa com qualquer estrutura

  Elas são variáveis anônimas ou “don’t care”

Torre de Hanoy

```
move(1,X,Y,_) :-
     write('Move top disk from '),
     write(X),
     write(' to '),
     write(Y),
     nl.

move(N,X,Y,Z) :-
     N>1,
     M is N-1,
     move(M,X,Z,Y),
     move(1,X,Y,_),
     move(M,Z,Y,X).
```

- Note o uso da impressão.

- Underscore casa com qualquer estrutura

  Elas são variáveis anônimas ou “don’t care”

Torre de Hanoy

```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.

move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Que consulta deve ser usada para obter a solução desejada ?

Torre de Hanoy – Qual é a Consulta?

Note como a consulta é importante para obter a solução:

Torre de Hanoy – Qual é a Consulta?

Note como a consulta é importante para obter a solução:

```
mova(4,esquerda,direita,centro)
```

Colorindo Mapas

Considere o problema de colorir um mapa de forma que nenhuma área vizinha tenha as mesmas cores.

Estabeleça as Cores como Fatos

```
cor(vermelha).
cor(verde).
cor(azul).
cor(amarela).
```

Estabeleça as Vizinhanças como Regras de Cores Diferentes

```
viz(CorX, CorY) :- cor(CorX), cor(CorY), CorX \= CorY.
```

```
viz(EstX, EstY) :- cor(EstX), cor(EstY), EstX \= EstY.
```

Estabeleça o Mapa
dizendo quem é Vizinho de quem


El Salvador
International Boundary
Department Boundary
Road
River
National Capital
Department Capital
City or Town
0 10 20 30 KM
0 10 20 30 Miles
© 2007 Geology.com
GUATEMALA
HONDURAS
EL SALVADOR
NICARAGUA
Pacific Ocean
Golfo de Fonseca
El Progreso
Chiquimula
Santa Rosa de Copan
Cucuyagua
Gracias
Jalapa
Ipala
Esquipulas
Monjas
Nueva Ocotepeque
San Sebastian
San Juan
Citala
Asuncion Mita
Metapan
Lago de Guija
Jutiapa
Tejutla
Embalse Cerron Grande
Chalatenango
Marcala
Lepaterique
Texistepeque
Santa Cruz
Chalchuapa
Santa Ana
Aguilares
Victoria
Ahuachapan
Lago de Coatepeque
Suchitoto
Sensuntepeque
La Hachadura
Quezaltepeque
Ilobasco
Lislique
Apopa
Sonsonate
Armenia
Gojutepeque
Sesori
San Francisco
Nueva San Salvador
San Salvador
San Vicente
Santa Rosa de Lima
Acajutla
Lago de Ilopango
El Triunfo
Nacaome
La Libertad
Comalapa
Zacatecoluca
San Miguel
San Lorenzo
La Herradura
El Carmen
Usulutan
La Union
La Canoa
Puerto El Triunfo
Laguna de Olomega
El Cuco
Intipuca
Rio Lempa
Rio Ostua
Rio Paz
Rio Sumpul
Rio Guarajambala
Rio Mejocote
Rio Torola
Rio Goascoran
Rio Jiboa
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14

Estabeleça o Mapa
dizendo quem é Vizinho de quem

Estabeleça o Mapa dizendo quem é Vizinho de quem

```
elSalvador(S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14) :-
       viz(S1,S2), viz(S1,S3),
       viz(S2,S3), viz(S2,S4),
       viz(S3,S4), viz(S3,S5),
       viz(S4,S5), viz(S4,S6), viz(S4,S7),
       viz(S5,S6), viz(S5,S10),
       viz(S6,S7), viz(S6,S8),
       viz(S7,S8), viz(S7,S9),
       viz(S8,S9), viz(S8,S10),
       viz(S9,S10), viz(S9,S11), viz(S9,S12),
       viz(S10,S12),
       viz(S11,S12),
       viz(S12,S13), viz(S12,S14),
       viz(S13,S14). /* S14 não tem mais nenhum vizinho */
```

Qual é a consulta?

Note como a consulta é importante para obter a solução:

Qual é a consulta?

Note como a consulta é importante para obter a solução:

```
elSalvador(S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14).
```

Exercício 4

Revise as regras anteriores para colorir o Brasil.

Use as siglas dos estados como variáveis.

Acompanhando
a Derivação de Regras

Considere o Programa PROLOG

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */

q(X) :- s(X).                          /* #4 */

r(a).                                  /* #5 */
r(b).                                  /* #6 */

s(a).                                  /* #7 */
s(b).                                  /* #8 */
s(c).                                  /* #9 */

u(d).                                  /* #10 */
```

O quê este programa responde para a consulta “?- P(X)” ?

:- P(X)

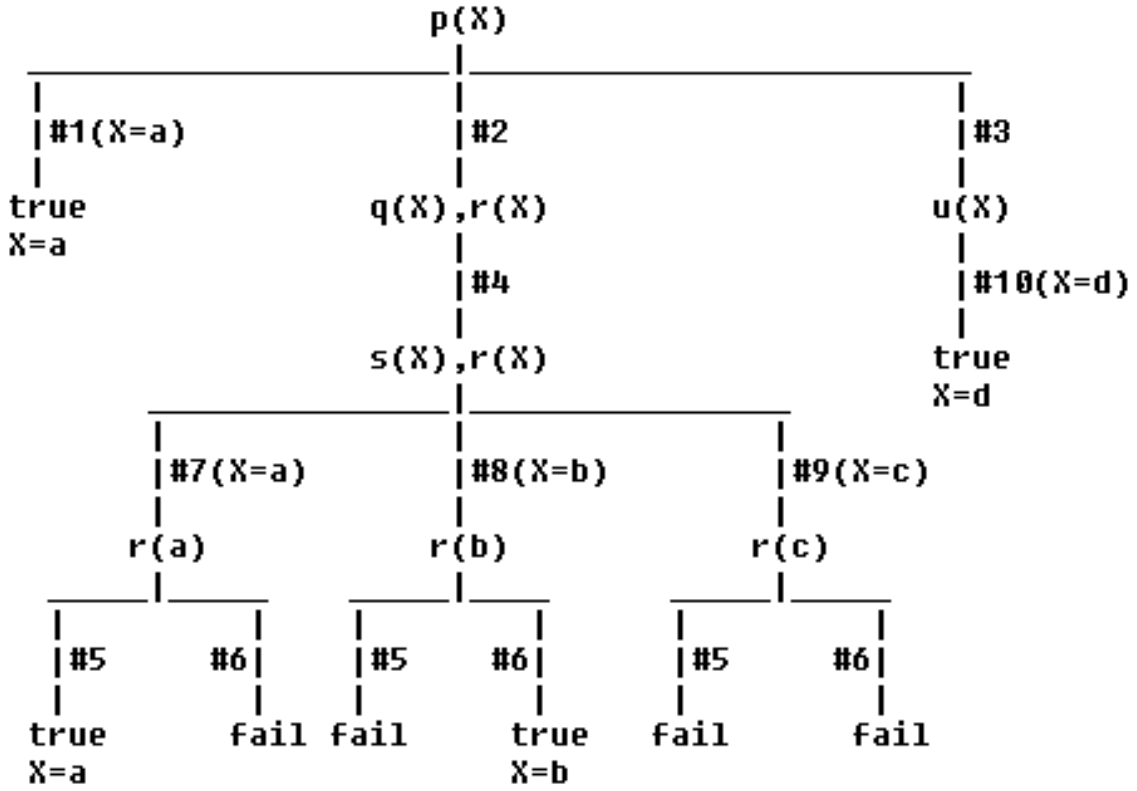
```
/* Consulta */

?- p(X).

X = a ;
X = a ;
X = b ;
X = d.
```

Qual a derivação?

Árvore de Derivação



Só saber a árvore é suficiente?

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */
```

Qual a primeira derivação?

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */
```

```
p(X) => X=a => p(a) => true.
```

Qual a segunda derivação?

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */

q(X) :- s(X).
```

```
p(X) => X=a => p(a) => true.
p(X) => q(X),r(X) => s(X),r(X) => X=a => s(a),r(a) => true,r(a) => true.
```

Qual a próxima derivação?

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */

q(X) :- s(X).                          /* #4 */

r(a).                                  /* #5 */
r(b).                                  /* #6 */
```

```
p(X) => X=a => p(a) => true.
p(X) => q(X),r(X) => q(X),r(X) => X=a => s(a),r(a) => true,r(a) => true.
p(X) => q(X),r(X) => q(X),r(X) => X=b => s(b),r(b) => true,r(b) => true.
```

Qual as próximas derivações?

```
/* programa P                          Clausula    */

p(a).                                  /* #1 */
p(X) :- q(X), r(X).                    /* #2 */
p(X) :- u(X).                          /* #3 */

q(X) :- s(X).                          /* #4 */

r(a).                                  /* #5 */
r(b).                                  /* #6 */

p(X) => X=a => p(a) => true.
p(X) => q(X),r(X) => q(X),r(X) => X=a => s(a),r(a) => true,r(a) => true.
p(X) => q(X),r(X) => q(X),r(X) => X=b => s(b),r(b) => true,r(b) => true.
p(X) => q(X),r(X) => q(X),r(X) => X=c => s(c),r(c) => true,r(c) => fail.
P(X) => u(X) => X=d => u(d) => true.
```

O quê acontece se invertermos q e r na regra #2?

Resolva agora

```
p(a).                          /* #1 */
p(X) :- q(X), r(X).            /* #2 */
p(X) :- q(X).
p(X) :- u(X).                  /* #3 */

q(X) :- s(X).                  /* #4 */
q(X) :- t(X).                  /* #4.1 */
q(X) :- u(X).                  /* #4.2 */

r(a).                          /* #5 */
r(b).

s(a).                          /* #7 */
s(b).                          /* #8 */
s(c).                          /* #9 */

u(z).                          /* #10 */

t(e).
t(f).
t(b).
```

Resolva para Q(X) e então para P(X)

Backtracking

• Toda vez que uma regra falha ou não há mais opções para um dado termo, a máquina Prolog retorna na árvore de derivação para tentar a próxima regra possível.
• A árvore derivação PROLOG pode crescer rapidamente se existir muitas regras para a mesma cabeça.

```
p(a).                          /* #1 */
p(X) :- q(X), r(X).            /* #2 */
p(X) :- u(X).                  /* #3 */

q(X) :- s(X).                  /* #4 */
q(X) :- t(X).                  /* #4.1 */
q(X) :- u(X).                  /* #4.2 */

r(a).                          /* #5 */
r(b).                          /* #6 */
s(a).                          /* #7 */
s(b).                          /* #8 */
s(c).                          /* #9 */
u(d).                          /* #10 */
t(e).
t(f).
t(b).
```

Usando o Cut

A cláusula CUT (!) elimina as próximas opções na árvore de derivação.

```
/* tente as consultas */

?- p(X),!.

?- r(X),!,s(Y).

?- r(X), s(Y), !.
```

O quê acontece?

Usando o Cut

A cláusula CUT (!) é muito usado no corpo de regras de programas PROLOG para elimina sub-metas.

```
p(a).                          /* #1 */
p(X) :- q(X), r(X).            /* #2 */
p(X) :- u(X).                  /* #3 */

q(X) :- s(X),!.                /* #4 */
q(X) :- t(X).                  /* #4.1 */
q(X) :- u(X).                  /* #4.2 */

r(a).                          /* #5 */
r(b).                          /* #6 */
s(a).                          /* #7 */
s(b).                          /* #8 */
s(c).                          /* #9 */
u(d).                          /* #10 */
t(e).
t(f).
t(b).
```

O quê acontece?

Usando o Cut

O cut é um recurso procedural para controlar que uma meta, ou sub-meta, foi satisfeita.

```
vermelho(marcador).
preto(lapis).
cor(P,vermelho) :- vermelho(P),!.
cor(P,preto) :- preto(P),!.
cor(P,desconhecida).
```

O quê acontece?

Usando o Cut

- O cut pode afetar o significado dos programas.
- Este é o caso do programa anterior.
- Isto pode ser muito ruim.

- O melhor uso para o cut é como uma ferramenta para melhorar eficiência, para evitar computação adicional que não é requerida. Este tipo de uso é chamado de ***corte verde*** ou ***green cut***.

Listas e Sequencias

Listas e Sequencias

- Prolog utiliza colchetes para construir listas.

`[a,b,c,d,e]` é uma lista em Prolog.

- A notação [X | Y] é utilizada para se referir à cabeça e resto da lista. As declarações abaixo definem as operações do LISP para cabeça, cauda e composição de listas:

```
car([X|Y],X).
cdr([X|Y],Y).
cons(X,R,[X|R]).
```

Exemplos

Escreva as regras que definem quando um elemento é membro de uma lista.

Membro

```
membro(X,[X|R]).
membro(X,[Y|R]) :- membro(X,R).
```

O quê `membro(c,[a,b,c,d,e])` retorna?

Membro

```
?- membro(c,[a,b,c,d,e]).
true
.
```

O quê `membro(X,[a,b,c,d,e])` retorna?

Continuação: em caso de existirem várias respostas, o Prolog retorna a primeira e espera você entrar ENTER ou ponto e vírgula.

Membro

```
?- membro(X,[a,b,c,d,e]).
X = a ;
X = b ;
X = c ;
X = d ;
X = e.
```

O quê `membro(c, X)` retorna?

Membro

```
?- membro(c,X).
X = [c|_G848] ;
X = [_G847, c|_G851] ;
X = [_G847, _G850, c|_G854] ;
X = [_G847, _G850, _G853, c|_G857] ;
X = [_G847, _G850, _G853, _G856, c|_G860] ;
X = [_G847, _G850, _G853, _G856, _G859, c|_G863] ;
X = [_G847, _G850, _G853, _G856, _G859, _G862, c|_G866] ;
X = [_G847, _G850, _G853, _G856, _G859, _G862, _G865, c|_G869] ;
X = [_G847, _G850, _G853, _G856, _G859, _G862, _G865, _G868, c|...] ;
X = [_G847, _G850, _G853, _G856, _G859, _G862, _G865, _G868, _G871|...] ;
X = [_G847, _G850, _G853, _G856, _G859, _G862, _G865, _G868, _G871|...] .
```

Ou seja, todas as listas que contêm `c`. Note o uso de variáveis anônimas (*underscore*) para denotar qualquer elemento.

Membro Versão Anônima

```
membro(X,[X|R]).
membro(X,[Y|R]) :- membro(X,R).
```

Falando em variável anônima, você consegue alterar a definição de *membro* para usá-las?

Quais nomes de variáveis (se algum) não importam nas regras acima.

Membro Versão Anônima

```
membroV2(X,[X|_]).
membroV2(X,[_|R]) :- membroV2(X,R).
```

Não ter que amarrar valores para as variáveis anônimas economiza um pouco de espaço e tempo de execução.

Consultas Interessantes: Membro

```
?- membro(X,[23,45,67,12,222,19,9,6]),
   Y is X*X,
   Y < 100.
```

O quê isto retorna?

Consultas Interessantes: Membro

```
?- membro([3,Y],[[1,a],[2,m],[3,z],[4,v],[3,p]]).
```

O quê isto retorna?

Exercício

Escreva a função

```
quadradoMenor(X,L1,L2)
```

que retorna em L2 todos os elementos de L1 cujo quadrado é menor que X.

Escreva a função

```
quadradoMenor2(X,L,E)
```

que retorna todos os elementos E de L cujo quadrado é menor que X.

Remova

Escreva as regras que removem elementos de uma lista.

Remova

```
remova(X,[],[]).
remova(X,[X|Y],W) :- remova(X,Y,W).
remova(X,[Z|Y], [Z|W]) :- X \== Z,
                           remova(X,Y,W).
```

Remova

```
remova(X,[X|R],R).
remova(X,[F|R],[F|S]) :- remova(X,R,S).
```

O quê acontece se rodarmos as consultas a seguir:

```
?- remova(a,[a,b,c],X).
?- remova(mm,[mm,a,mm,b,mm,c],X).
?- remova(mm,X,[a,b,c]).
```

Insere

Escreva uma rotina que insere um elemento em todas as posições possíveis de uma lista.

Insere

```
insere1(X,L,[X|L]).
insere1(X,[Y|R],[Y|R2]) :- insere1(X,R,R2).
```

Remova x Insere

Baseado no último resultado. Escreva a regra *insere* baseado nas regras de *remova*.

Remova x Insere

Baseado no última resultado. Escreva a regra *insere* baseado nas regras de *remova*.

```
insere(X,L,R) :- remova(X,R,L).
```

Exemplos Interessantes

União de Dois Conjuntos

```
uniao([],L,L).

uniao([X|E],L,W) :- member(X,L), uniao(E,L,W),!.

uniao([X|E],L,[X|W]) :- uniao(E,L,W).


?- uniao([2,3,4,5],[4,5,6,7],L).
L = [2, 3, 4, 5, 6, 7].
```

Exercício

Escreva a função

```
interseccao(C1,C2,I)
```

que retorna em I a intersecção dos conjuntos C1 e C2.

União de uma Lista de Conjuntos

```
uniaoll([X],X) :-!.
uniaoll([X,Y|R],WW) :- uniaoll([Y|R],W),
                       uniao(X,W,WW).
```

```
?- uniaoll([[1,2,3],[3,4,5],[6,2]],L).
L = [1, 3, 4, 5, 6, 2]
```

Removendo Recursividade Interna de *uniaoll*

```
uniaoll2([X],X) :- !.
uniaoll2([X,Y|R],WW) :- uniao(X,Y,W),
                        uniaoll2([W|R],WW).


?- uniaoll2([[1,2,3],[3,4,5],[6,2]],L).
L = [1, 3, 4, 5, 6, 2].
```

Ordenação por Inserção v1

```
isort([],[]).
isort([X|Y],Z) :- isort(Y,W),coloque(X,W,Z).
```

Ordenação por Inserção v1

```
isort([],[]).
isort([X|Y],Z) :- isort(Y,W),coloque(X,W,Z).

coloque(X,[],[X]).
coloque(X,[Y|W1],[X,Y|W1]) :- X=<Y.
coloque(X,[Y|W1],[Y|W2]) :- X>Y, coloque(X,W1,W2).
```

Ordenação por Inserção v2

```
isort([],[]).
isort([X|Y],Z) :- isort(Y,W),coloque(X,W,Z).

coloque(X,[],[X]).

% coloque(X,[Y|W1],[X,Y|W1]) :- X=<Y.
% coloque(X,[Y|W1],[Y|W2]) :- X>Y, coloque(X,W1,W2).

coloque(X,[Y|W1],[X,Y|W1]) :- X=<Y , !.
coloque(X,[Y|W1],[Y|W2]) :- coloque(X,W1,W2).
```

O quê o cut faz?

Exercício 6

Escreva a rotina merge(L1,L2,L), onde L1, L2, L são listas ordenadas.

Regras de entrega são as mesmas de antes.

Exercício 7

Escreva a rotina mergeSort(L,LO), onde L é uma lista e LO é uma lista ordenada através do algoritmo de ordenação por mesclagem dos elementos de L.

Exercício 8

Escreva o resto de QuickSort(particione e apende)em Prolog

```
quicksort([], []).
quicksort([X], [X]).
quicksort([X, Y | Z], W) :-
            particione([X, Y | Z], U, V),
            quicksort(U, Q), quicksort(V, R),
            apende(Q, R, W).
```

Equivalência Dado Programa

Manipulando a Base de Dados de Fatos e Regras

Base de Dados

Prolog mantém uma base de dados de fatos e regras. A base atual pode ser listada com o comando:

```
?- listing.
```

Predicados Dinâmicos

Existem uma série de predicados para alterar a lista de cláusulas sendo processadas por um programa Prolog.

Através deles pode-se dinamicamente incluir e remover cláusulas na base de cláusulas. São eles:

```
assert, asserta, assertz
retract, abolish
```

A cláusula `dynamic(predicado/n)` deve ser usada para tornar um predicado dinâmico.

Incluindo Cláusulas

`asserta/1` declara uma cláusula como a primeira cláusula do predicado

`assertz/1` declara uma cláusula como a última cláusula do predicado.

Por exemplo, no prompt do Prolog, sobre a base intro.pl faça:

```
assertz(pai(manoel,bia)).

assertz(mulher(bia)).

assertz((avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z),
homem(X))).
```

Removendo Cláusulas

`abolish(+Name, +Arity)` remove todas as cláusulas do predicado `Name` com dimensão `Arity`.

`retract(+Term)` remove a cláusula `Term` da base.

Por exemplo, no prompt do Prolog, sobre a base faça:

```
retract(pai(manoel,bia)).

retract((avo(X,Z) :- progenitor(X,Y), progenitor(Y,Z),
homem(X))).

abolish(pai,2).
```

Exemplo Dinâmico

A solução abaixo realiza um armazenamento dinâmico de uma solução (possivelmente custosa). Desta forma consultas futuras não precisam recomputar a solução.

```
:- dynamic olha_resultado/3.

resolucao_complicada(X,Y,Res):-
   olha_resultado(X,Y,Res), !. % olha resultado

resolucao_complicada(X,Y,Res):-
   Res is (X+Y)*(X+Y),                % resolve
   assert(olha_resultado(X,Y,Res)). % armazena
```

Equivalência Dado Programa

Dados como Metas

Dados como Metas

Dados podem ser chamados como metas para criar cláusulas de mais alta ordem. Isto é feito com:

`call(:Goal, +ExtraArg1, ...)` que chama a meta `Goal` com os argumentos passados.

`apply(:Goal, +ListArg)` que chama a meta `Goal` com a lista de argumentos passados.

Dados como Metas - Exemplo

Suponha que queremos deixar nosso programa de ordenação por inserção mais genérico de forma que ele possa, ordenar uma lista qualquer de itens.

Para isto temos que:

1 – utilizar uma função qualquer para comparar os itens listados

2 – identificar as chaves de comparação nos elementos listados

Ordenação por Inserção v2

Vamos começar por:

1 – utilizar uma função qualquer para comparar os itens listados

```
isort([],[]).
isort([X|Y],Z) :- isort(Y,W),coloque(X,W,Z).

coloque(X,[],[X]) :-!.
coloque(X,[Y|W1],[X,Y|W1]) :- X=<Y , !.
coloque(X,[Y|W1],[Y|W2]) :- coloque(X,W1,W2),!.
```

Ordenação por Inserção v3

Considere agora:

```
iSortG([],_,[]).
iSortG([X|Y],C,Z) :-
                    iSortG(Y,C,W),insereG(X,W,C,Z).

insereG(X,[],_,[X]) :- !.
insereG(X,[Y|R],C,[X,Y|R]) :- comparador(C,X,Y),!.
insereG(X,[Y|R],C,[Y|W]) :- insereG(X,R,C,W),!.

comparador(direta,X,Y) :- X=<Y.
comparador(inversa,X,Y) :- X>Y.
```

Ordenação por Inserção v3

Execute:

```
iSortG([[3,1,2],direta,L).

iSortG([[3,1,2],inversa,L).
```

Note que o comparador torna a função de comparação parametrizável, permitindo ordenar a lista de números na ordem direta e na ordem inversa.

Ordenação por Inserção v4

Considere agora:

```
iSortP([],_,[]).
iSortP([X|Y],CC,Z) :-
iSortP(Y,CC,W),insereP(X,W,CC,Z).

insereP(X,[],_,[X]) :- !.
insereP(X,[Y|R],CC,[X,Y|R]) :- call(CC,X,Y),!.
insereP(X,[Y|R],CC,[Y|W]) :- insereP(X,R,CC,W),!.
```

O uso do `call` torna a comparação completamente parametrizável.

Ordenação por Inserção v4

Execute:

```
iSortP([[3,1,2],<,L).

iSortP([[3,1,2],>=,L).
```

Os predicados de comparação agora são passados como parâmetros.

Ordenação por Inserção v5

Vamos agora estender isort para:

2 - identificar as chaves de comparação nos elementos listados

Para isto vamos passar um outro predicado para extrair a chave dos itens da lista antes das comparações.

Vamos aproveitar o esforço e criar cláusulas que tratam os casos defaults se não passarmos os predicados de comparação e de extração de chaves.

Ordenação por Inserção v5

```
iSortPP([],_,_,[]).
iSortPP([X|Y],CC,Key,Z) :-
                  iSortPP(Y,CC,Key,W),inserePP(X,W,CC,Key,Z).

inserePP(X,[],_,_,[X]) :- !.
inserePP(X,[Y|R],CC,Key,[X,Y|R]) :- call(Key,X,XC),
                                    call(Key,Y,YC),
                                    call(CC,XC,YC),!.

inserePP(X,[Y|R],CC,Key,[Y|W]) :- inserePP(X,R,CC,Key,W),!.
```

Ordenação por Inserção v5

```
% Casos Default

% nenhum predicado de comparação e extração de chave
% defaults são =< e identidade

iSortPP(L,LO) :- iSortPP(L,=<,identidade,LO).

% somente a função de comparação é passada
% default de chave é a identidade

iSortPP(L,CC,LO) :- iSortPP(L,CC,identidade,LO).

identidade(X,X).
```

Ordenação por Inserção v5

Execute as consultas:

```
iSortPP([4,7,9,5,1,2,0,8,6],L).

iSortPP([4,7,9,5,1,2,0,8,6],>,L).

iSortPP([[4,7,9],[5,1],[2]],=<,length,L).

iSortPP([[4,7,9],[5,1],[2]],>,length,L).


asserta(first([X|R],X)).
iSortPP([[4,7,9],[5,1],[2]],=<,first,L).
```

Lista 1

Todos as cláusulas em um único arquivo:

lista01.pl