EXERCÍCIOS PROLOG

Lógica – Prof. Tacla (UTFPR/Curitiba)
arquivo: ExProlog01.docx

1. Introdução

Basicamente, um programa PROLOG é constituído por fatos acerca do domínio e regras – que são sentenças ou fórmulas.

Por exemplo, a fórmula abaixo em LPO

$$\forall x \, \forall y \, (p(x) \wedge q(y) \rightarrow r(x, y))$$

fica, em PROLOG

```
r(X, Y):- p(X), q(Y).
```

*r(X, Y) é chamado de **cabeça de regra***
*p(X), q(Y) é o **corpo da regra**.*
não esqueça do ponto ao final da fórmula

Já um fato em LPO que é representado por

$$p(a)$$

em PROLOG fica

```
p(a).
```

onde **a** é um símbolo de constante nas duas linguagens.

2. Exemplo Prático

Objetivo: entender como se representam predicados com constantes (**fatos**) e fórmulas quantificadas universalmente com implicação material (**regras**).

Execute o Prolog:

1. Abra um **xterm**
2. No xterm, tecle **swipl**
3. No ambiente PROLOG tecle **emacs**

Entenda o programa e faça queries:

4. Baixe e abra o programa **ex005-00-regras-apr.pl** em http://www.dainf.ct.utfpr.edu.br/~tacla/logica/prolog/
5. Faça **file>edit>ex005-00-regras-apr.pl**

```
% FATOS
nota_freq(alberto, logica, 9.0, 60.0).
nota_freq(juca, logica, 7.0, 95.0).
nota_freq(maria, logica, 4.0, 75.0).
```

```
% REGRA
aprovado(X, Y, Z, W) :- nota_freq(X, Y, Z, W),Z>=7,W>=75.
```

O fato marcado em amarelo é um predicado quaternário que representa:

alberto, na disciplina de lógica, tirou nota 9.0 e teve frequência de 60.0%. Observe que só há constantes: alberto, logica, 9.0 e 60.

A regra em azul significa, em LPO:

$\forall x \forall y \forall z \forall w$ (nota_freq(x, y, z, w) $\wedge$ z >= 7 $\wedge$ z >= 75 $\rightarrow$ aprovado(x, y, z, w)

Traduzindo para português:

Todo aluno de qualquer disciplina que tirar nota maior ou igual a 7 e tiver frequência maior ou igual a 75% está aprovado.

6. Agora, no prompt do PROLOG execute as queries abaixo, observando que o motor de inferência prossegue para encontrar outros valores de variáveis que satisfaçam a query quando você tecla ‘;’

```
For help, use ?- help(Topic). or ?- ap

1 ?- aprovado(X, logica, Y, Z).
X = juca,
Y = 7.0,
Z = 95.0 ;
false.

2 ?- aprovado(X, Y, Z, W).
X = juca,
Y = logica,
Z = 7.0,
W = 95.0 ;
false.

3 ?- nota_freq(X, _, Z, _).
X = alberto,
Z = 9.0 ;
X = juca,
Z = 7.0 ;
X = maria,
Z = 4.0.

4 ?-
```

Pergunta pelos aprovados em **lógica**, sendo X o nome, Y a nota e Z a frequência.

Pergunta pelos aprovados em qualquer disciplina, sendo X o nome, Y a disciplina, Z a nota e W a frequência.

Consulta o predicado not_freq retornando apenas o nome (em X) e a nota (em X). O *underscore* faz com que o PROLOG ignore os argumentos (neste caso, a disciplina e a frequência).

Entenda o trace que mostra os passos do motor de inferência:

1. No prompt, digite trace para ligá-lo.
2. Faça a query: **`aprovado(X, Y, Z, W)`** e observe como PROLOG trabalha.

a. Note que você apenas declarou fatos e escreveu fórmulas (regras); o PROLOG executa uma busca para encontrar os valores de variáveis que podem satisfazer a regra aprovado.
b. Note que testa todas as substituições possíveis para X, Y, Z e W.
c. Note que a ordem em que ele busca: para demonstrar que uma quádrupla (X, Y, Z, W) satisfaz o predicado aprovado ele tem que provar que:
    i. há um fato/cabeça de regra que casa com nota_freq(X, Y, Z, W) e
    ii. Z >= 7 e
    iii. W >= 75.
    iv. Repetir i, ii e iii para todos fatos/cabeças de regra que casam com nota_freq(X, Y, Z, W)

```
[trace] 9 ?- aprovado(X, Y, Z, W).
Call: (6) aprovado(_G1155, _G1156, _G1157, _G1158) ? creep // tecle c
   Call: (7) nota_freq(_G1155, _G1156, _G1157, _G1158) ? creep
   Exit: (7) nota_freq(alberto, logica, 9.0, 60.0) ? creep
   Call: (7) 9.0>=7 ? creep     // verifica a subfórmula da nota
   Exit: (7) 9.0>=7 ? creep
   Call: (7) 60.0>=75 ? creep // verifica a subfórmula da freq.
   Fail: (7) 60.0>=75 ? creep
Redo: (7) nota_freq(_G1155, _G1156, _G1157, _G1158) ? creep
   Exit: (7) nota_freq(juca, logica, 7.0, 95.0) ? creep
   Call: (7) 7.0>=7 ? creep
   Exit: (7) 7.0>=7 ? creep
   Call: (7) 95.0>=75 ? creep
   Exit: (7) 95.0>=75 ? creep
   Exit: (6) aprovado(juca, logica, 7.0, 95.0) ? creep
      X = juca,
      Y = logica,
      Z = 7.0,
      W = 95.0 ;    // tecle ; para prosseguir
Redo: (7) nota_freq(_G1155, _G1156, _G1157, _G1158) ? creep
   Exit: (7) nota_freq(maria, logica, 4.0, 75.0) ? creep
   Call: (7) 4.0>=7 ? creep
   Fail: (7) 4.0>=7 ? creep
   Fail: (6) aprovado(_G1155, _G1156, _G1157, _G1158) ? creep
false.
```

3. EXERCÍCIO: REGRAS APROVADO

Modifique o programa **ex005-00-regras-apr.pl** incluindo uma fórmula (regra) que permita determinar os alunos reprovados em qualquer disciplina.

Solução em ex005-01-regras-apr-rep-sol.pl

4. EXERCÍCIO: MUNDO DE TARSKI

Abra o programa **ex008-00-mundo-tarski.pl** e faça o que se pede nos comentários iniciais do mesmo (construir fórmulas que correspondam ao que foi codificado no programa prolog e um mundo que satisfaça todas as fórmulas).

Soluções: *ex008-01-mundo-tarski.wld* e *ex008-02-mundo-tarski.flm*

Nota: para rodar o software tarski leia o documento disponível em http://www.dainf.ct.utfpr.edu.br/~tacla/logica/Exercicios/ExPredicados03-tarski-world.pdf

5. EXERCÍCIO: NACIONALIDADE DE COMIDAS

Abra o arquivo **ex010-00-comida.pl** e inclua uma regra que represente que *toda pessoa gosta de uma certa comida se a pessoa e a comida forem do mesmo país (de outra forma, toda pessoa gosta das comidas de seu país).*

Solução em ex010-01-comida-sol.pl (não muito boa) e em ex010-02-comida-sol.pl

6. EXERCÍCIO: JOGO DA VELHA

Ainda sobre regras e negação em PROLOG

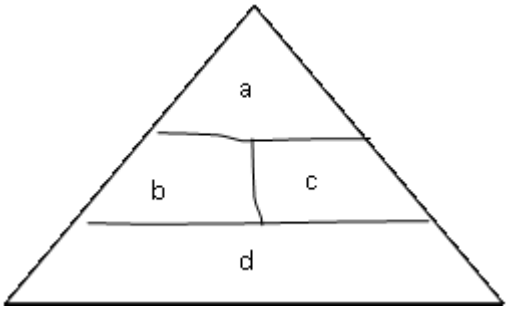
Faça um programa PROLOG que com o estado de um tabuleiro do jogo da velha diga se há vencedor e quem é este vencedor (cruz ou bola).

Solução em ex180-00-jogo-da-velha.pl

7. EXERCÍCIO: COLORAÇÃO DE MAPAS

Ainda sobre regras...

Dado um mapa com quatro regiões sendo que somente 3 delas podem ser adjacentes fazer um programa que escolha a cor para cada região de forma que duas regiões fronteiriças não tenham a mesma cor.



Solução em ex190-00-coloracao-mapa: esta solução com menos predicados (mais difícil de alterar)

Solução em ex190-10-coloracao-mapa: esta solução é dependente de estrutura de adjacência das regiões acima.

8. EXERCÍCIO: MODELO E TEORIA EM PROLOG

A figura ao abaixo representa um mundo formado por pessoas e cachorros bem como pelas relações apresentadas entre estes objetos do domínio. Construa uma teoria em LPO que represente

- pessoas e cachorros como objetos distintos;
- a relação Dono somente se dá entre pessoas e cachorro;
- melhor amigo: alguém só tem um melhor amigo
- Somente pessoas são donas de algo.
- Somente cachorros podem pertencer a outrem.
- Há alguém cujo melhor amigo é um cão.

Implementar a teoria construída (se possível!) em PROLOG. Verificar se o programa pode responder às queries:

- Quem é o dono de X?
- Quais são os cachorros?
- Quais são as pessoas?
- Quem é o melhor amigo de X?

Responda:

1) Explique como implementou a função melhor amigo;
2) Sabe-se que a semântica em LPO depende do modelo construído sobre os conjuntos P e F. Pergunta-se: onde está o modelo em um programa PROLOG?
3) Quais são os limites que você percebeu no PROLOG em relação à LPO?

Solução em *ex170-00-teoria-modelo.pl*

9. EXERCÍCIO: RECURSIVIDADE NO FATORIAL

Abra o arquivo ex060-00-fatorial.pl e observe a implementação da recursividade. Há dois predicados, um é o caso base, **fatorial(0,1)**, que significa que o fatorial de 0 é um. O outro – **fatorial(X,Y)**, é o passo de redução que leva em algum momento ao caso base. No entanto, o programa tal como está implementado apresenta um problema de término. Entenda e descreva o problema e conserte o programa.

Solução em *ex060-10-fatorial-sol.pl*

10. EXERCÍCIO: EXPRESSIVIDADE SIMPSONS

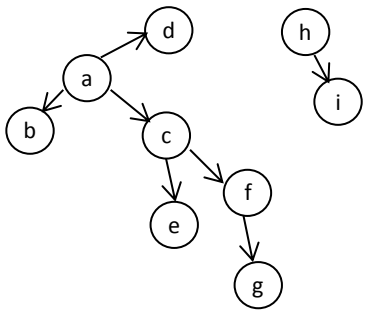
Abra o arquivo **ex030-00-expressividade-simpsons.pl** e observe que estamos no caso da lógica proposicional (o programa não usa variáveis, somente constantes). Problema: expressar a regra de antepassado sem variáveis gera inúmeros fatos – uma para cada caso. Pergunta-se, como podemos criar uma regra geral? A solução passa por recursividade.

Solução está em ex040-00-simpsons-recursividade.pl

11. EXERCÍCIO: MAPA ACÍCLICO

Exercício sobre recursividade de busca de caminhos em árvores

1. Represente um mapa de estradas que interconectam cidades (por meio de uma ou mais árvores). Dada uma cidade, deseja-se saber todas as cidades alcançáveis a partir dela ou dadas duas cidades deseja-se saber se há conexão ou não entre elas.
2. Desenhe na forma de árvore como o PROLOG resolve a query **?con(c1,Y)**



?con(a, Y) {b, c, d, f, e}
?con(h, g) false
?con(a, g) true

Solução em ex200-00-mapaAciclico-sol.pl

12. EXERCÍCIO: RESOLUÇÃO

Pré-requisito: aula 01-UnificacaoEmProlog.pptx

- ex140-00-arvoreBusca.pl - fazer a árvore de busca e comparar com trace.
- ex060-10-unificacao-recursividade.pl
- ex060-10-unificacao-recursividade-interminavel.pl