

## Inteligência Artificial

### Lista de Exercícios – Programação Lógica - Profa. Heloisa

#### Recomendações

- Baixe o ambiente de programação SWI-Prolog e implemente os exemplos da aula e os exercícios desta lista para se certificar que entendeu o funcionamento da linguagem;
- Para aprender a programar em uma linguagem diferente é necessário construir um número expressivo de programas; o número de exercícios da lista é grande para que a repetição possa levar à compreensão do raciocínio utilizado na programação da linguagem;
- Prolog é uma linguagem com fundamento bem diferente das linguagens mais conhecidas; preocupe-se em entender a lógica da programação sem procurar fórmulas que repitam o que é feito em outras linguagens;
- Lembre-se: Prolog não tem comando de atribuição; não tente fazer atribuição com o predicado IS;
- Uma lista pode conter sublistas como elementos; quando o exercício pede que uma operação seja feita no primeiro nível da lista a operação deve considerar as listas internas como um único elemento; quando o exercício pede que a operação seja feita em todos os níveis da lista, a operação deve ser feita com os elementos das listas internas.
- Não pode ter espaço em branco entre o nome do predicado e o parêntesis – causa erro.
- As aspas, quando usadas para definir uma constante formada por sequências em que apareçam brancos ou comecem com letras maiúsculas podem ser “duplas” ou ‘simples’. Entretanto, o mesmo tipo de aspas deve ser usado na definição do predicado e na consulta.

1) Um programa Prolog tem as seguintes cláusulas:

```
% Base de Dados Geometria
vertical(seg(ponto(X,Y),ponto(X,Y1))).
horizontal(seg(ponto(X,Y),ponto(X1,Y))).
```

Dê os resultados das consultas:

```
?- vertical(seg(ponto(1,1),ponto(1,2))).
?- vertical(seg(ponto(1,1),ponto(2,Y))).
?- horizontal(seg(ponto(1,1),ponto(2,Y))).
?- vertical(seg(ponto(2,3),P)).
```

2) Na programação Lógica, a operação de casamento de padrão é efetuada pela unificação, que no Prolog é denotada pelo operador =. Dê o resultado das instanciações de variáveis nas operações de casamento de padrão abaixo:

```
a) point(A,B) = point(1,2).
b) point(A,B) = point(X, X, Z).
c) [a,b,c,[ab], [], [[a], c]] = [X, Y | Z].
d) [p, [seg], t, q] = [X,[Y],Z | S].
e) triangle(point(-1,0),P2, P3) = triangle(P1, point(1,0), point(0,Y)).
f) [X,Y | Z] = [2,[3,4]].
g) [lista, de, exercicios, de , IA] = [X, de | W].
```

3) Dê o resultado das seguintes consultas:

```
?- gosta_de(maria,X) = gosta_de(X,joao).
false
?- 3+5 >= 5+3.
true
?- 3+(5*2-1)/4 =:= 5-2.
true % As expressões são calculadas e os resultados são comparados
?- 3+(5*2-1)/4 = 5-2.
false % Esse é o operador de unificação. Os termos não unificam
?- 3+(5*2-1)/4 == 5-2.
false % Esse operador verifica se os termos são idênticos. Nesse caso, não são.
?- X=4, X is (X-2)*2.
X=4 % Na segunda parte da consulta o valor de X não é alterado pelo is mas tem valor
% igual ao do resultado da expressão aritmética, portanto, unifica.
?- X=4, X is (X-1)*2.
false % Na segunda parte da consulta o valor de X não é alterado pelo is. A expressão
% tem valor 6, diferente do valor de X. Portanto, não unifica.
?- X is X+1.
Erro % Variável não instanciada não pode ser usada em expressões aritméticas
?- X=1, X is X + 1.
false. % X instancia com 1. Na segunda parte da consulta, o valor de X não é alterado pelo is
```

4) Construa uma base de dados sobre livros com:

a) pelo menos cinco estruturas do tipo exemplificado abaixo, onde a lista de palavras-chave pode ter entre três e seis elementos.

Exemplo de estrutura:

```
livro(nome('C completo e total'), autor('Schildt'), pal_chave([linguagemc, programacao,
computacao])).
```

```
livro(nome('Paradigmas de LP'), autor('Schildt'), pal_chave([linguagemc, programacao,
computacao])).
```

```
livro(nome('LISP'), autor('Winston, P. ; Horn, B.'), pal_chave([lisp, programacao, list])).
livro(nome('Introdução à Programação Orientada a Objetos usando JAVA'), autor('Santos, R.'), pal_chave(['POO', classes, objetos])).
```

b) Escreva consultas para encontrar:

- nome do autor, dado o nome do livro
- nome do livro, dado o nome do autor
- as palavras-chave, dado o nome do livro
- nome do autor e o nome do livro, dada uma palavra-chave

```
> livro(nome('Paradigmas de LP'), autor(X),_).
> livro(nome(X), autor('Schildt'), _).
> livro(nome('Paradigmas de LP'), _, pal_chave(L)).
> livro(nome(Nome), autor(Autor), pal_chave(L)), pertence(lisp, L).
% O predicado pertence deve estar definido e presente na base
```

c) Escreva um programa para, dada uma lista de palavras-chave, encontrar os livros (nome e autor) que tem pelo menos uma das palavras-chave fornecidas. Os livros encontrados devem ser retornados um de cada vez.

```
busca_livro(Lista, Nome, Autor):-
    livro(nome(Nome), autor(Autor), pal_chave(Pal_chave)),
    tem_palavra(Lista,Pal_chave).

tem_palavra([X | Y], Lista):-
    pertence(X, Lista), !.
tem_palavra([X | Y], Lista):-
    tem_palavra(Y, Lista).
```

5) Defina os predicados `n_par(Lista)` e `n_impar(Lista)` para determinar se uma dada lista tem número par ou ímpar de elementos, respectivamente. (A linguagem tem o predicado pré-definido `length(L,N)`, que poderia ter sido usado nesse exercício em vez de definir o predicado `comprimento`).

```
comprimento([ ], 0) :- !.
comprimento([X|L],N) :-
    comprimento(L,N1), N is N1 + 1.
n_par(L) :-
    comprimento(L,N), N mod 2 =:= 0.
n_impar(L):-
    comprimento(L,N), N mod 2 =\= 0.
```

6) Defina a relação `shift_esq(Lista1, Lista2)` tal que `Lista2` seja a `Lista1` com uma rotação à esquerda (deslocamento de todos os elementos à esquerda em uma posição e adicionando o primeiro elemento como último). Por exemplo,

```
?- shift_esq([1,2,3,4,5], L1).
L1 = [2,3,4,5,1]
```

```
%usando o predicado add_ultimo definido na aula
shift_esq([ ],[ ]):- !.
shift_esq([X], [X]):- !.
```

```
shift_esq([X|Y], L) :- add_ultimo(X,Y,L).
```

```
add_ultimo(X,[ ], [X]) :- !.
```

```
add_ultimo(X,[E|Y],[E|Z]) :- add_ultimo(X,Y,Z).
```

- 7) Defina a relação `shift_dir(Lista1, Lista2)` tal que `Lista2` seja a `Lista1` com uma rotação à direita. Por exemplo,

```
?- shift_dir([1,2,3,4,5], L1).
```

```
L1 = [5,1,2,3,4]
```

- 8) Defina a relação `traduz(L1, L2)` para traduzir uma lista de dígitos entre 0 e 9 para a palavra correspondente. Por exemplo:

```
?- traduz([1, 2, 4], L).
```

```
L = [um, dois, quatro]
```

Sugestão: Use a relação auxiliar: `t(0, zero)`, `t(1, um)`, `t(2, dois)`, .....`t(9, nove)`

% o predicado `t` deve estar definido

```
traduz([ ], [ ]):- !.
```

```
traduz([X|Y], [Z|W]) :- t(X,Z), traduz(Y,W).
```

- 9) Defina a relação `subconjunto(Set, Subset)` onde `Set` e `Subset` são duas listas representando conjuntos, tal que seja possível verificar se `Subset` é subconjunto de `Set`. (O predicado `member(X,L)` é um predicado pré-definido da linguagem que é verdadeiro quando `X` pertence a lista `L`). Por exemplo:

```
?- subconjunto([1,b,c], [1,c]).
```

```
true
```

```
subconjunto([ ], [ ]) :- !.
```

```
subconjunto(Set, [X|Y]) :- member(X, Set), !, subconjunto(Set,Y).
```

- 10) Defina a relação `max(X, Y, Max)` para encontrar o maior entre os números `X` e `Y`.

```
max(X,Y,X):- X >=Y, !.
```

```
max([ ], Y,Y).
```

- 11) Defina o predicado `maxlista(Lista, Max)` tal que `Max` seja o maior número na lista de números `Lista`.

Versão 1:

```
maxlista([X],X) :- !.
```

```
maxlista([X|Y],X) :- maxlista(Y,M), X >=M, !.
```

```
maxlista([X|Y], M) :- maxlista(Y,M).
```

Versão 2:

```
maxlista([X],X) :- !.
```

```
maxlista([X|Y],M) :- maxlista(Y,N), max(X,N,M).
```

- 12) Defina o predicado `entre(N1, N2, X)` tal que, para dois números inteiros dados `N1` e `N2`, gere uma lista com todos os inteiros que satisfaçam a restrição  $N1 \leq X \leq N2$ .

```

entre(N1,N2,L):- N1 <=N2, entre1(N1, N2, L).
entre1(N,N,[N]):- !.
entre1(N1, N2, [N1|X]) :- Nnovo is N1 + 1, entre1(Nnovo,N2,X).

```

- 13) A relação a seguir classifica números em três classes: positivo, negativo e zero:

```

classe (N, positivo) :- N > 0.
classe (0, zero).
classe (N, negativo) :- N < 0.

```

Defina esse procedimento de maneira mais eficiente usando o corte.

```

classe (N, positivo) :- N > 0, !.
classe (0, zero):- !.
classe (N, negativo).

```

- 14) Defina o procedimento `split(Numeros, Positivos, Negativos)` que divide uma lista de números em duas listas: uma contendo os números positivos (incluindo o zero) e outra contendo os negativos. Proponha duas versões: uma usando corte e outra sem uso do corte.

```

split([ ], [ ], [ ]):- !.
split([X|Cauda], [X|Y], Z) :- X>=0, split(Cauda,Y,Z), !.
split([X|Cauda], Y, [X|Z]) :- split(Cauda, Y, Z).

```

- 15) Listas podem ser usadas para representar conjuntos. Implemente procedimentos em prolog para realizar as operações básicas entre conjuntos: união, interseção e diferença. Note que quando listas são usadas para representar conjuntos, elementos duplicados não podem aparecer nas listas. Para resolver o exercício, assuma que os elementos só aparecem uma vez na lista. O predicado `not(S)` inverte o valor verdade da sentença S.

```

uniao([ ], Y, Y):- !.
uniao([X|Cauda], Y, [X|Z]) :- not(member(X,Y)), uniao(Cauda,Y,Z),!.
uniao([_|Cauda], Y, Z) :- uniao(Cauda,Y,Z).

```

```

inter([ ], _, [ ]):- !.
inter([X|Y], Z, [X|W]) :- member(X,Z), inter(Y,Z,W), !.
inter([_|Y], Z, W):- inter(Y,Z,W).

```

```

difer([ ], _, [ ]):- !.
difer([X|Y], L, [X|Z]) :- not(member(X,L)), difer(Y,L,Z),!.
difer([_|Y], L, Z) :- difer(Y,L,Z).

```

- 16) Fazer um programa Prolog para, dadas 3 listas representando conjuntos de números, construir a interseção dessas listas: lista contendo somente os elementos que aparecem nas três listas dadas. Assumir que as listas dadas têm apenas números, sem repetições. Os elementos da interseção podem aparecer em qualquer ordem na lista resultante. Usar o corte (!) para omitir condições.

```

inter_tres(L1, L2, L3, L) :- inter(L1,L2,Linter), inter(Linter,L3, L).
inter([ ], _, [ ]):- !.
inter([X|Y], Z, [X|W]) :- member(X,Z), inter(Y,Z,W), !.
inter([_|Y], Z, W):- inter(Y,Z,W).

```

- 17) Construir um programa Prolog para verificar se um dado elemento pertence a uma lista, em qualquer nível. Use o predicado `is_list(X)`, que retorna true se o seu argumento for lista.

```
encontra_atomo(X, [X]) :- !.  
encontra_atomo(X, [_|_]) :- !.  
encontra_atomo(X, [Y|_]) :- is_list(Y), encontra_atomo(X,Y), !.  
encontra_atomo(X, [_|Z]) :- encontra_atomo(X,Z).
```

- 18) Escreva um programa Prolog para, dada uma lista numérica em ordem crescente e um número, inserir o número dado na posição correta para manter a ordenação.

```
insere_em_ordem(E, [], [E]) :- !.  
insere_em_ordem(E, [X|Y], [E,X|Y]) :- E <= X, !.  
insere_em_ordem(E, [X|Y], [X|Z]) :- insere_em_ordem(E, Y, Z), !.
```

- 19) Escreva um programa Prolog para, dada uma lista L e dois elementos, E1 e E2, substituir todas as ocorrências de E1 por E2 em L, no primeiro nível da lista.
- 20) Escreva um programa Prolog para, dada uma lista L e dois elementos, E1 e E2, verificar se E1 e E2 aparecem consecutivamente (na ordem dada, sem elementos entre eles) na lista L, no primeiro nível da lista.
- 21) Escreva um programa Prolog `caminho(X, L)` para verificar se existe um caminho entre os nós X e Y em um grafo. Suponha que existe um predicado `arco(X,Y)` que é verdadeiro quando os nós X e Y estão conectados por um arco.

```
caminho(X, Y) :- arco(X, Y).  
caminho(X, Y) :- arco(X, Z), caminho(Z, Y).
```

- 22) Supondo que a base de conhecimento tem, além das cláusulas do exercício anterior, os fatos `arco(a, b)` e `arco(b, c)`, qual a resposta para as consultas:

```
?- caminho(a, c).  
?- caminho(a, X).  
?- caminho(X, Y).
```

- 23) Supondo que a base de conhecimento tem, além das cláusulas do exercício anterior, os fatos `arco(a, d)` e `arco(d, b)`, encontre as respostas para as consultas:

```
?- caminho(a, c).  
?- caminho(a, X).
```

Comente os resultados obtidos nas consultas de confirmação e de recuperação usando as regras do programa `caminho(X,Y)` definidas no exercício 21.