INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus Birigui	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA** **Campus Birigui** **Bacharelado em Engenharia de Computação**

Disciplina: Inteligência Artificial	**Lista 4**
Professor: Prof. Dr. Murilo Varges da Silva	**Data:** 23/10/2023
Nome do Aluno: Henrique Akira Hiraga	**Prontuário:** BI300838X

LISTA 4 – PROLOG

1. Defina o predicado último(L,U), que determina o último item U numa lista L. Por exemplo, último([a,b,c],U), resulta em U=c.

```
ultimo([X],X).
ultimo([_|Y],X) :- ultimo(Y,X).
```

```
?- ultimo([a,b,c],X).
X = c ;
false.
```

2. Defina o predicado tam(L,N), que determina o número de itens N existente numa lista L. Por exemplo, tam([a,b,c],N), resulta em N=3.

```
tam([],0).
tam([_|X],N) :- tam(X,N1), N is N1+1.
```

```
?- tam([a,b,c],N).
N = 3.
```

3. Defina o predicado soma(L,S) que calcula a soma S dos itens da lista L. Por exemplo, soma([4,9,1],S) resulta em S=14.

```
soma([],0).
soma([X|Y],S) :- soma(Y,S1), S is X+S1.
```

```
?- soma([4,9,1],S).
S = 14.
```

4. Defina o predicado máx(L,M) que determina o item máximo M na lista L. Por exemplo, máx[4,9,1],M) resulta em M=9.

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

```
?- max([4,9,1],M).
M = 9 ;
false.
```

5. Usando o predicado anexa, defina o predicado inv(L,R) que inverte a lista L. Por exemplo, inv([b, c, a], R) resulta em R=[a,c,b].

```
anexa([],B,B).
anexa([X|A],B,[X|C]) :- anexa(A,B,C).

inv([],[]).
inv([X|A],B) :- inv(A,R), anexa(R,[X],B).
```

```
?- inv([b,c,a],R).
R = [a, c, b].
```

6. Usando o predicado inv, defina o predicado sim(L) que verifica se uma lista é simétrica. Por exemplo, sim([a,r,a,r,a]) resulta em yes.

```
inv([],[]).
inv([X|A],B) :- inv(A,R), anexa(R,[X],B).

sim([]).
sim([X]) :- true,!.
sim([X|A]) :- inv([X|A],[X|A]).
```

```
?- sim([a,r,a,r,a]).
true.
```

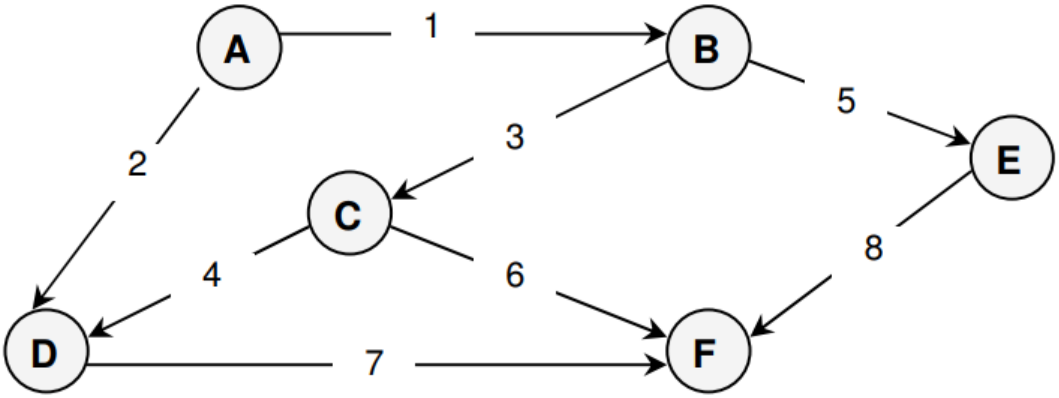
7. Usando a tabela d(0,zero), d(1,um), ..., d(9,nove), defina o predicado txt(D,P) que converte uma lista de dígitos numa lista de palavras. Por exemplo, txt([7,2,1],P) resulta em P=[sete,dois,um].

```
d(0,'zero').
d(1,'um').
d(2,'dois').
d(3,'três').
d(4,'quatro').
d(5,'cinco').
d(6,'seis').
d(7,'sete').
d(8,'oito').
d(9,'nove').

txt([],[]).
txt([X|Y],[X1|Y1]) :- d(X,X1), txt(Y,Y1).
```

```
?- txt([7,2,1],P).
P = [sete, dois, um].
```

8. O grafo a seguir representa um mapa, cujas cidades são representadas por letras e cujas estradas são representados por números.



   a. Usando o predicado estrada(Número,Origem,Destino), crie um programa para representar esse mapa.
   b. Defina o predicado rota(A,B,R), que determina uma rota R (lista de estradas) que leva da cidade A até a cidade B.

```
estrada(1,a,b).
estrada(2,a,d).
estrada(3,b,c).
estrada(4,c,d).
estrada(5,b,e).
estrada(6,c,f).
estrada(7,d,f).
estrada(8,e,f).

rota(X,Y,[R]) :- estrada(R,X,Y).
rota(X,Y,[R1|R2]) :- estrada(R1,X,Z), rota(Z,Y,R2).
```

```
?- rota(a,f,X).
X = [1, 3, 6] ;
X = [1, 3, 4, 7] ;
X = [1, 5, 8] ;
X = [2, 7] ;
false.
```

9. Um retângulo é representado pela estrutura retângulo(A,B,C,D), cujos vértices são A, B, C e D, nesta ordem.
    a. Defina o predicado regular(R) que resulta em yes apenas se R for um retângulo cujos lados sejam verticais e horizontais.
    b. Defina o predicado quadrado(R) que resulta em yes apenas se R for um retângulo cujos lados têm as mesmas medidas.

```
% Definir uma relação para verificar se uma linha é vertical
vertical(linha(ponto(X,Y1), ponto(X,Y2))) :- Y1 =\= Y2.

% Definir uma relação para verificar se uma linha é horizontal
horizontal(linha(ponto(X1,Y), ponto(X2,Y))) :- X1 =\= X2.

% Definir uma relação para verificar se um retângulo é regular (quadrado)
regular(retangulo(A, B, C, D)) :-
    vertical(linha(A, B)),
    horizontal(linha(B, C)),
    vertical(linha(C, D)),
    horizontal(linha(D, A)),
    A \== B,
    B \== C,
    C \== D.

% Definir uma relação para verificar se um retângulo é um quadrado
quadrado(R) :-
    regular(R),
    vertical(linha(A, C)),
    horizontal(linha(B, D)),
    A \== B,
    B \== C,
    C \== D.
```

```
?- regular(retângulo(ponto(0,0),ponto(1,0),ponto(1,1),ponto(0,1))).
true.
```

10. Supondo que a base de dados esteja inicialmente vazia, indique qual será o seu conteúdo após terem sido executadas as seguintes consultas.

```
?- asserta( metal(ferro) ).
?- assertz( metal(cobre) ).
?- asserta( metal(ouro) ).
?- assertz( metal(zinco) ).
?- retract( metal(X) ).
```

Os metais serão adicionados para a lista, porém após utilizar o comando retract(metal(x)), toda a lista será apagada.

11. Implemente os predicados liga, desliga e lâmpada para que eles funcionem conforme indicado pelos exemplos a seguir:

```
?- liga, lâmpada(X).
X = acessa
Yes
?- desliga, lâmpada(X).
X = apagada
Yes
```

```
:- dynamic lâmpada/1.

lâmpada(apagada).

liga :-
    retract(lâmpada(_)),
    asserta(lâmpada(acesa)).

desliga :-
    retract(lâmpada(_)),
    asserta(lâmpada(apagada)).
```

```
?- lâmpada(X).
X = apagada.

?- liga, lâmpada(X).
X = acesa.

?- lâmpada(X).
X = acesa.

?- desliga, lâmpada(X).
X = apagada.

?- lâmpada(X).
X = apagada.
```

12. O predicado asserta adiciona um fato à base de dados, incondicionalmente, mesmo que ele já esteja lá. Para impedir essa redundância, defina o predicado memorize, tal que ele seja semelhante a asserta, mas só adicione à base de dados fatos inéditos.

```
memorize(X) :-
    retractall(X), % Remove um fato se já houver um fato igual na base de dados
    asserta(X).    % Acrescenta o novo fato
```

```
?- memorize(boss(gundyr)).
true.

?- memorize(boss(artorias)).
true.

?- memorize(boss(gundyr)).
true.

?- listing(boss).
:- dynamic boss/1.

boss(gundyr).
boss(artorias).

true.
```

13. Suponha um robô capaz de andar até um certo local e pegar ou soltar objetos. Além disso, suponha que esse robô mantém numa base de dados sua posição corrente e as respectivas posições de uma série de objetos. Implemente os predicados pos(Obj,Loc), ande(Dest), pegue(Obj) e solte(Obj), de modo que o comportamento desse robô possa ser simulado, conforme exemplificado a seguir:

```
?- pos(O,L).
O = robô
L = garagem ;
O = tv
L = sala ;
No
?- pegue(tv), ande(quarto), solte(tv), ande(cozinha).
anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
Yes
```

```
:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, sala).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2),  %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define
posição atual
    format('anda de ~w até ~w\n',[L1,L2]), % define deslocamento
    format('pega ~w\n',[Obj]). % define objeto que o robô pegou

ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo,Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n', [L1,Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj,Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n', [Obj]). %define robô soltando objeto em posse
```

```
?- pos(O,L).
O = robo,
L = garagem ;
O = tv,
L = sala.

?- pegue(tv),ande(quarto),solte(tv),ande(cozinha).
anda de garagem até sala
pega tv
anda de sala até quarto
solta tv
anda de quarto até cozinha
false.
```

14. Modifique o programa desenvolvido no exercício anterior de modo que, quando for solicitado ao robô pegar um objeto cuja posição é desconhecida, ele pergunte ao usuário onde está esse objeto e atualize a sua base de dados com a nova informação. Veja um exemplo:

```
?- pos(O,L).
O = robô
L = cozinha ;
O = tv
```

```
L = quarto ;
No
?-     pegue(lixo),     ande(rua),     solte(lixo),
ande(garagem).
Onde está lixo? quintal
anda de cozinha até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
Yes
?- pos(O,L).
O = robô
L = garagem ;
O = lixo
L = rua ;
O = tv
L = quarto ;
No
```

```
:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, quarto).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2),  %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define posição atual
    format('anda de ~w até ~w\n',[L1,L2]), % define deslocamento
    format('pega ~w\n',[Obj]). % define objeto que o robô pegou

pegue(Obj) :-
    format('Onde está ~w? ',[Obj]), %Pergunta qual a localização do objeto
    gets(L2), % Guarda local do objeto em L2
    asserta(pos(Obj,L2)), %Define posição do objeto cuja localização era desconhecida
    pos(robo,L1), %Posição do robô
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %Pega novo objeto
    retract(pos(robo,L1)), %Remove posição original do robô
    asserta(pos(robo,L2)), %Define posição atual do robô
    format('anda de ~w até ~w\n', [L1,L2]), %Define deslocamento
    format('pega ~w\n', [Obj]). %Define objeto que o robô pegou

ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo,Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n', [L1,Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj,Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n', [Obj]). %define robô soltando objeto em posse

gets(S) :-
    read_line_to_codes(user_input,C),
    name(S,C).
```

```
?- pegue(lixo),ande(rua),solte(lixo),ande(garagem).
Onde está lixo? quintal
anda de garagem até quintal
pega lixo
anda de quintal até rua
solta lixo
anda de rua até garagem
false.

?- pos(O,L).
O = robo,
L = garagem ;
O = lixo,
L = rua ;
O = tv,
L = quarto.
```

15. Acrescente também ao programa do robô o predicado leve(Obj,Loc), que leva um objeto até um determinado local. Por exemplo:

```
?- leve(tv,sala).
```

```
anda de garagem até quarto
pega tv
anda de quarto até sala
solta tv
Yes
```

```
:- dynamic pos/2.
:- dynamic posse/1.

% pos(Objeto, Local)
pos(robo, garagem).
pos(tv, quarto).
posse([]). %objeto que o robo esta segurando, inicialmente indefinido

pegue(Obj) :-
    pos(robo, L1), %pos(robo, L1) = posição do robô
    pos(Obj, L2),  %pos(Obj, L2) = posição do objeto
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %pega objeto
    retract(pos(robo,_)), asserta(pos(robo,L2)), %apaga posição original e define posição atual
    format('anda de ~w até ~w\n',[L1,L2]), % define deslocamento
    format('pega ~w\n',[Obj]). % define objeto que o robô pegou

pegue(Obj) :-
    format('Onde está ~w? ',[Obj]), %Pergunta qual a localização do objeto
    gets(L2), % Guarda local do objeto em L2
    asserta(pos(Obj,L2)), %Define posição do objeto cuja localização era desconhecida
    pos(robo,L1), %Posição do robô
    retract(posse(_)), %solta objeto em posse, se tiver algum
    asserta(posse(Obj)), %Pega novo objeto
    retract(pos(robo,L1)), %Remove posição original do robô
    asserta(pos(robo,L2)), %Define posição atual do robô
    format('anda de ~w até ~w\n', [L1,L2]), %Define deslocamento
    format('pega ~w\n', [Obj]). %Define objeto que o robô pegou

ande(Dest) :-
    pos(robo,L1), %pos do robô
    retract(pos(robo,_)), %remove posicao original do robô
    asserta(pos(robo,Dest)), %define nova posicao do robô
    format('anda de ~w até ~w\n', [L1,Dest]), %define deslocamento
    posse(Obj), %verifica objeto em posse do robô
    retract(pos(Obj,_)), %remove posição original do objeto em posse do robô
    asserta(pos(Obj,Dest)). %atualiza posição do objeto do robô

solte(Obj) :-
    posse(Obj), %verifica objeto em posse do robô
    retract(posse(_)), %remove objeto em posse do robô
    asserta(posse([])), %define que não há nenhum objeto em posse do robô
    format('solta ~w\n', [Obj]). %define robô soltando objeto em posse

gets(S) :-
    read_line_to_codes(user_input,C),
    name(S,C).

leve(Obj,Loc) :-
    pegue(Obj),
    ande(Loc),

    solte(Obj).
```

```
?- pos(O,L).
O = robo,
L = garagem ;
O = lixo,
L = rua ;
O = tv,
L = quarto.

?- leve(tv,sala).
anda de garagem até quarto
pega tv
anda de quarto até sala
solta tv
true ;
```