Paradigmas de Linguagens de Programação
Lista 1
Prof. Sergio D. Zorzo

---

1. Expresse cada uma das seguintes afirmações usando uma sentença escrita em lógica de primeira ordem, utilizando apenas os símbolos de predicados “pai”, “mãe”, “progenitor”, “mulher”, “filho”
   Obs: faça uso de quaisquer quantificadores que julgar necessários
   a. “Se X é um pai ou X é uma mãe então X é progenitor de alguém”

```
Resposta=(∀X)((∃Y)progenitor(X, Y) se pai(X) ou mãe(X))
```

   b. “Se X é um progenitor e X é uma mulher então X é mãe de alguém”
   c. “Se X é um pai ou X é uma mãe então alguém é progenitor de alguém”
   d. “Se todo mundo é um progenitor então alguém é um filho”

2. Expresse cada uma das seguintes afirmações usando lógica de primeira ordem em forma clausal, utilizando apenas as palavras grifadas como símbolos de predicado
   a. “Nem tudo que **reluz** é **ouro**”
   b. “Tudo **está bem** quando **termina bem**”

```
estáBem(X) se terminaBem(X)
```

   c. “Nenhum **homem** é uma **ilha**”
   d. “Os **fins justificam** os **meios**”

3. Encontre os unificadores mais gerais para os seguintes pares de termos (se possível)
   a. p(a, X) e p(Y, b)

```
{Y/a, X/b}
```

   b. p(X, X) e p(Y, Z)
   c. p(X, Y) e p(Y, X)
   d. p(t(X, t(X, b))) e p(t(a, Z))
   e. p(t(X, t(X, b))) e p(t(a, t(Z, Z)))
   f. p(X, f(Y)) e p(f(Y), X)
   g. p(X, f(X)) e p(f(Z), f(Z))

4. Faça a árvore de busca para o seguinte programa lógico, considerando a consulta

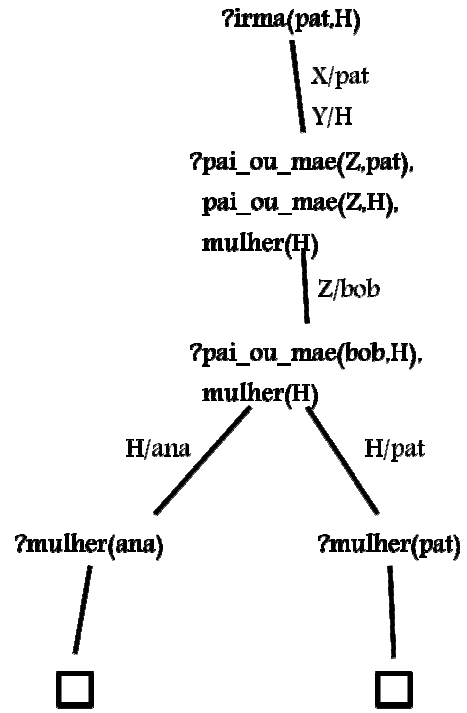
?sub(l(a,X), l(a,l(b,nil)))

   S1: sub(X, Y) se pre(X, Y)
   S2: sub(X, l(U,Y)) se sub(X, Y)
   P1: pre(nil, Y)
   P2: pre(l(U,X), l(U,Y)) se pre(X, Y)

5. Considere o seguinte programa Prolog

```
pai_ou_mae(tom,bob).
pai_ou_mae(tom,liz).
pai_ou_mae(bob,ana).
pai_ou_mae(bob,pat).
pai_ou_mae(pat,jim).
mulher(ana).
mulher(pat).
irma(X,Y) :- pai_ou_mae(Z,X),
             pai_ou_mae(Z,Y),
             mulher(Y).
```

Faça a árvore de busca para a consulta `irma(pat,X)`, para demonstrar como pat se torna irmã dela mesma nessa versão do programa.



6. Considere o seguinte programa Prolog

```
proc(X,Z) :- proc(Y,Z), a(X,Y).
proc(X,Z) :- a(X,Z).
a(a,b).
a(b,c).
```

a. Faça a árvore de busca para a consulta `proc(a,Z)`, para demonstrar como é importante conhecer o mecanismo de inferência na escrita dos programas.
b. O que acontece?
c. Como resolver o problema?

7. Um programa Prolog tem as seguintes cláusulas

```
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)).
```

8. 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]

9. Dê o resultado das seguintes consultas:

```
?- gosta_de(maria,X) = gosta_de(X,joao).
?- 3+5 >= 5+3.
?- 3+5 @>= 5+3
?- 3+(5*2-1)/4 =:= 5-2.
?- 3+(5*2-1)/4 = 5-2.
?- 3+(5*2-1)/4 == 5-2.
?- X=4, X is (X-2)*2.
?- X is X+1.
?- X=1, X is X + 1.
```

10. Constua uma base de dados sobre livros com pelo menos cinco estruturas do tipo:

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

onde a lista de palavras chave pode ter entre três e seis elementos.

a. 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

b. 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 dados um de cada vez.

11. Defina os predicados n_par(Lista) e n_impar(Lista) para determinar se uma dada lista tem número par ou ímpar de elementos, respectivamente.

```
n_par([]).
n_par([_,_|L]) :- n_par(L).

n_impar([X]) :- atom(X).
n_impar([_,_|L]) :- n_impar(L).

%%%%%% outra solução %%%%%%

comprimento([ ], 0) :- !.
comprimento([_|L],N) :- comprimento(L,N1), N is N1 + 1.
n_par2(L) :- comprimento(L,N), N mod 2 =:= 0.
n_impar2(L) :- comprimento(L,N), N mod 2 =\= 0.
```

12. Defina a relação shift(Lista1, Lista2) tal que Lista2, seja a Lista1 com uma rotação à esquerda. Por exemplo:

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

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

```
?- traduz([1, 2, 4], L).
L = [um, dois, quatro]
```

Dica: use a relação auxiliar: t(0, zero), t(1, um), t(2, dois), ...t(9, nove)

14. Defina a relação subset(Set, Subset) onde Set e Subset são duas listas representando conjuntos, tal que seja possível verificar se Subset é subconjunto de Set. Por exemplo:

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

15. Defina a relação max(X, Y, Max) tal que Max é o maior entre os números X e Y.

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

17. Defina o predicado between(N1, N2, X) tal que, para dois números inteiros dados N1 e N2, gere todos os inteiros que satisfaçam a restrição N1 ≤ X ≤ N2.

18. Considere o seguinte programa Prolog

```
del(X, [X|L], L).
del(X, [Y|L], [Y|L2]) :- del(X, L, L2).
```

Este programa, quando utilizado da forma del(X, L1, L2), declara que L2 contém todos os elementos de L1 menos uma ocorrência de X. Se L1 tiver mais do que uma ocorrência de X, cada passo da execução irá apresentar L2 removendo uma ocorrência diferente. Faça o teste com a consulta: `del(a,[a,b,c,a],L).`

Modifique o programa para que em uma única execução, todas as ocorrências de X sejam removidas de L1 ao apresentar L2. Dica: utilize o operador X \== Y para indicar que as váriáveis X e Y representam instâncias distintas.

```
del(_, [], []).
del(X, [X|L], L2) :- del(X, L, L2).
del(X, [Y|L], [Y|L2]) :- X \== Y, del(X, L, L2).
```

19. Considere os seguintes programas Prolog

P1:

```
membro(X,[X|_]).
membro(X,[_|Z]) :- membro(X,Z).
```

P2:

```
membro(X,[X|_]).
membro(X,[Y|Z]) :- X\==Y, membro(X,Z).
```

P3:

```
membro(X,[Y|_]) :- X==Y.
membro(X,[_|Z]) :- membro(X,Z).
```

P4:

```
membro(X,[Y|_]) :- X==Y.
membro(X,[Y|Z]) :- X\==Y, membro(X,Z).
```

a. Qual é o resultado destes programas para consultas do tipo membro(a,L) (ou seja, quando se deseja saber se a pertence a L)? Todos retornam as mesmas respostas? Qual é mais eficiente?
b. Qual é o resultado destes programas para consultas do tipo membro(X,[a,b,c]) (ou seja, quando se deseja listar em X os membros de [a,b,c])? Todos retornam as mesmas respostas? Qual é mais eficiente?

20. O seguinte programa Prolog conta o número de elementos de uma lista.

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

a. Qual o resultado da consulta conta([a,[b,c],d],X)? Por que?
b. Modifique o programa para que o mesmo consiga contar elementos recursivamente. Por exemplo, na consulta do item a, ele deve retornar 4.
Dica: utilize os predicados de negação e teste de lista, descritos abaixo.
Predicado not (Obs: SWI-Prolog já possui o operador "\+" equivalente ao not)

```
not(X) :- call(X), !, fail.
not(X).
```

Predicado is_list(+Termo) retorna true caso Termo seja uma lista, e false caso contrário

21. Faça um programa Prolog que, dada uma lista, separa em duas, colocando os números negativos em uma, os positivos em outra, e descartando o zero. Ex:

```
?- separa_sz([1,3,-5,0,-64,37,0,19,-53],P,N).
P = [1,3,37,19] ,
N = [-5,-64,-53] ;
```

22. Escreva um programa Prolog que possa responder a pergunta 'Eu sou meu próprio avô?', criando um conjunto de relações que representem a seguinte situação:

"Eu me casei com uma viúva (W) que tem uma filha adulta (D). Meu pai (F), que nos visitava freqüentemente, apaixonou-se por minha enteada e casou-se com ela. Logo, meu pai se tornou meu enteado e minha enteada tornou-se minha madrasta. Alguns meses depois, minha mulher teve um filho (S1), que se tornou cunhado do meu pai, assim como meu tio. A mulher do meu pai, isto é, minha enteada, também teve um filho (S2)."

23. 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.

24. 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 tem apenas números, sem repetições. Os elementos da interseção podem aparecer em qualquer ordem na lista resultante.

25. Construir um programa Prolog para verificar se um dado elemento pertence a uma lista, em qualquer nível.

```
pertence(X,[X|_]).
pertence(X,[_|Cauda]) :- pertence(X,Cauda).
pertence(X,[Cabeca|_]) :- is_list(Cabeca),
                          pertence(X,Cabeca).
```

26. Construir um programa Prolog para, dada uma lista numérica, fazer a ordenação dos elementos dessa lista.

27. Escreva um programa Prolog para resolver o seguinte problema: Existem dois baldes de água, um com capacidade para 3 galões e outro com capacidade para 5 galões de água. É possível encher os baldes totalmente com água de uma torneira, esvaziá-los totalmente, despejar água de um para o outro até que o que recebe a água esteja cheio ou até que aquele do qual se despeja a água esteja totalmente vazio. Defina uma seqüência de ações que deixe 4 galões de água no balde maior.

28. Escreva programas em Prolog para resolver os problemas:

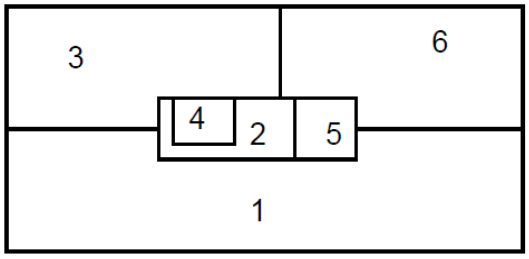
a. Missionários e Canibais: Na margem esquerda de um rio há três missionários e três canibais, que querem atravessá-lo para a outra margem. Há um bote que pode transportar no máximo duas pessoas ao mesmo tempo sendo que uma delas será sempre um missionário. Os missionários não podem ficar em menor número que os canibais em qualquer margem, caso contrário serão devorados. Como fazer para que todos cheguem a salvo na outra margem do rio? Dica: representar o estado por (Me, Ce, be) onde Me é o número de missionários na margem esquerda, Ce é o número de canibais na margem esquerda e be é 1 se o bote está na margem esquerda e 0 se o bote está na margem direita.

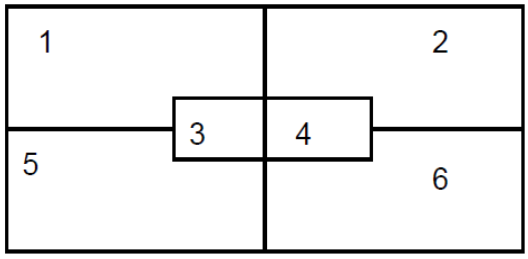
```
% Neste programa, cada travessia é representada
% Pelo predicado "travessia".
% Os argumentos M1 e C1 representam o número
% de missionários e canibais na margem esquerda
% antes da travessia e os argumentos M2 e C2
% representam o número de missionários e canibais
% na margem esquerda depois da travessia.
% O argumento B1 representa a posição do bote antes
% da travessia (esquerda ou direita), e o
% argumento B2 representa a posição do bote
% depois da travessia

% Há seis possibilidades de travessia:
% T1: Um missionário da esquerda para a direita
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
        M1 \== 0,
        B1 == esquerda,
        M2 is M1 - 1,
        C2 is C1,
        B2 = direita,
        margemEsquerdaValida(M2,C2),
        Movimento = 'M >'.

% T2: Um missionário da direita para a esquerda
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
        M1 \== 3,
        B1 == direita,
        M2 is M1 + 1,
        C2 is C1,
        B2 = esquerda,
        margemDireitaValida(M2,C2),
        Movimento = 'M <'.

% T3: Um missionário e um canibal da esquerda para a direita
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
```

```
        M1 \== 0,
        C1 \== 0,
        B1 == esquerda,
        M2 is M1 - 1,
        C2 is C1 - 1,
        B2 = direita,
        margemEsquerdaValida(M2,C2),
        Movimento = 'MC >'.

% T4: Um missionário e um canibal da direita para a esquerda
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
        M1 \== 3,
        C1 \== 3,
        B1 == direita,
        M2 is M1 + 1,
        C2 is C1 + 1,
        B2 = esquerda,
        margemDireitaValida(M2,C2),
        Movimento = 'MC <'.

% T5: Dois missionários da esquerda para a direita
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
        M1 >= 2,
        B1 == esquerda,
        M2 is M1 - 2,
        C2 is C1,
        B2 = direita,
        margemEsquerdaValida(M2,C2),
        Movimento = 'MM >'.

% T6: Dois missionários da direita para a esquerda
travessia(M1, C1, B1, M2, C2, B2, Movimento) :-
        M1 < 2,
        B1 == direita,
        M2 is M1 + 2,
        C2 is C1,
        B2 = esquerda,
        margemDireitaValida(M2,C2),
        Movimento = 'MM <'.

% Verifica se as margens estão válidas (ou seja, se não há mais canibais
% do que missionários
margemEsquerdaValida(0,_) :- !.
margemEsquerdaValida(M,C) :- M >= C.

margemDireitaValida(3,_) :- !.
margemDireitaValida(M,C) :- 3 - M >= 3 - C.

sequencia(M1,C1,B1,M2,C2,B2,[Movimento],_) :-
        travessia(M1,C1,B1,M2,C2,B2,Movimento).

sequencia(M1,C1,B1,M2,C2,B2,[Mov|OutrosMov],Contador) :-
             Contador > 1,
             NovoContador is Contador - 1,
             travessia(M1,C1,B1,TempM,TempC,TempB,Mov),
             sequencia(TempM,TempC,TempB,M2,C2,B2,OutrosMov,NovoContador).

resolve(MaxMov) :- resolve1(Mov,MaxMov), imprime(Mov), nl.

resolve1(Mov,MaxMov) :- sequencia(3,3,esquerda,0,0,direita,Mov,MaxMov).

imprime([]).
imprime([X|L]) :- write('['), write(X), write('] '), imprime(L).

% Exemplo de consulta
% ?- resolve(11).
```

b. Fazendeiro: Um fazendeiro está na margem norte de um rio com uma cabra, um lobo e um repolho. Existe um bote através do qual o fazendeiro pode transportar um elemento por vez para a outra margem do rio. Se o lobo e a cabra ficarem sozinhos numa margem, o lobo comerá a cabra. Se a cabra e o repolho ficarem sozinhos, a cabra comerá o repolho. Como transportar a salvo os três elementos, considerando o repolho um passageiro?

c. Colorir figuras: Dadas as figuras:





colorir cada região numerada com uma de quatro cores sem que duas regiões contíguas possuam a mesma cor.