Paradigmas de Programação – Lista 2

Parte A - Fundamentos da programação lógica

1. Considere o código Prolog abaixo:

```
time('5am, wood again', 2, 'mg').
time('Amigos do Mortandela', 3, 'mg').
time('C++ ou uma linguagem misteriosa?', 2, 'go').
time('Lone Wolves', 2, 'df').
time('Monkeys', 1, 'go').
time('Teorema de Off', 1, 'df').
time('Teorema do Chinês Viajante', 3, 'df').
time('Torcida Pão de Alho', 1, 'mg').

campeao(X) :- time(X, 1, _).

selecionado(X) :- campeao(X).
selecionado(X) :- time(X, _, 'mg').
```

(a) Identifique os predicados (nome e aridade). ↝ time/3, campeao/1, selecionado/1

(b) Determine o número de fatos, regras e cláusulas.

(c) Observe a saída da consulta ↳ 8 fatos, 3 regras e 11 cláusulas

```
?- selecionado(X).
```

Proponha uma correção para o código para que a saída da consulta liste corretamente os selecionados.

2. Considere os seguintes fatos:

```
p(a, b).
p(a, c).
p(b).

g(a).
g(b).
g(a, b).
```

Determine o resultado das seguintes consultas. Caso o resultado seja verdadeiro, indique também o valor das variáveis lógicas, quando for o caso.

(a) `?- p(b, a).` ↝ false

(b) `?- p(a, b).` ↝ true

(c) `?- g(X).` ↝ true / X = a; X = b.

(d) `?- p(a, Y).` ↝ true / Y = b; Y = c.

3. Considere uma consulta composta de Prolog e as quatro portas que controlam o fluxo da execução. Descreva o cenário no qual a saída da consulta exibirá as variáveis lógicas e seus respectivos valores. Explique quando é possível inserir ponto-e-vírgula após o resultado de uma consulta e qual é o significado desta inserção, em termos do fluxo de controle.

4. Considere o código Prolog abaixo, onde o predicado `matricula/2` indica que o estudante está matriculado na disciplina indicada e o predicado `estudante/3` informa que o estudante do semestre indicado cursa a graduação dada no terceiro argumento:

```
estudante(ana, 3, 'Engenharia de Software').
estudante(beto, 1, 'Engenharias').
estudante(carlos, 4, 'Engenharia de Energia').
estudante(diane, 2, 'Engenharias').
estudante(euler, 1, 'Engenharias').
estudante(fabio, 5, 'Engenharia de Software').
estudante(gustavo, 8, 'Engenharia de Software').
estudante(heitor, 7, 'Engenharia de Energia').
estudante(ian, 3, 'Engenharias').

matricula('Cálculo 1', ana).
matricula('Cálculo 1', fabio).
matricula('Cálculo 1', diane).
matricula('Cálculo 1', euler).
matricula('Cálculo 1', gustavo).
matricula('Cálculo 1', ian).

matricula('IAL', beto).
matricula('IAL', diane).
matricula('IAL', euler).

matricula('APC', carlos).
matricula('APC', fabio).
matricula('APC', gustavo).
matricula('APC', ian).
```

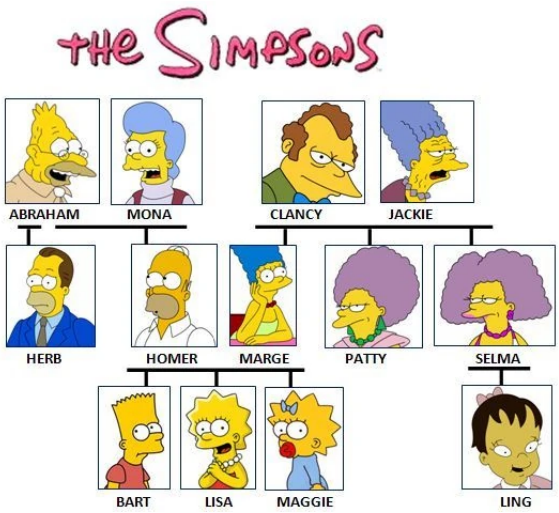
(a) Implemente a regra `tem_calouros/1`, que retorna as disciplinas que tem calouros matriculados.

(b) Implemente a regra `turma_mista/1`, que retorna as disciplinas que tem ao menos um estudante de cada curso.

(c) Implemente a regra `software/0`, que imprime todos os estudantes de Engenharia de Software, um estudante por linha.

Parte B - Regras

5. Considere a figura abaixo, que apresenta a árvore genealógica dos Simpsons:



Implemente os predicados abaixo, declarando a seguir os fatos relativos à família Simpson relacionados a cada predicado.

(a) `male/1` e `female/1`, que indicam os homens e mulheres, respectivamente.

(b) `father/2`, que indica o pai do respectivo membro da família.

(c) `mother/2`, que indica a mãe do respectivo membro da família.

Implemente as regras abaixo e valide sua implementação com consultas na base de fatos declarados.

(a) `uncle/2`, que indica o tio do respectivo membro da família.

(b) `grandmother/2`, que indica a avó do respectivo membro da família.

6. Implemente o predicado `distance/4`, que compute a distância percorrida S, em metros, a partir de um ponto inicial que fica a I metros da origem, com velocidade de V m/s e aceleração igual a A m/s^2.

7. Implemente o predicado `divisors/4`, que computa o número de divisores positivos X de um inteiro positivo N. Utilize o predicado `rem/2`, que retorna o resto da divisão de X por Y.

Parte C - Funções

8. O predicado `is_set/1` recebe como parâmetro uma lista de inteiros e retorna verdadeiro se todos os elemntos são únicos. Descreva o comportamento da consulta

```
?- is_set(1, 2, 3).
```

9. Determine, sem executar, o resultado da consulta:

```
?- X = Y, Y \= Z, Z = W, X = a, W = b.
```

Execute esta consulta no *listener* Prolog e compare com sua expectativa. Caso sua expectativa seja diferente do resultado, como a consulta deve ser modificada para obter o retorno esperado?

10. O código abaixo implementa o predicado `same_parity/2`, que retorna verdadeiro se X e Y são ambos pares ou ambos ímpares.

```
1 same_parity(X, Y) :-
2     rem(X, 2) = rem(Y, 2).
```

Contudo, a consulta

```
?- same_parity(2, 4).
```

retorna falso. Explique o que há de incorreto na implementação e proponha uma correção.

3) Irá entrar pelo call da 1ª consulta, com os valores passados e atar as variáveis coletadas, apenas quando possível. Dando certo, sai da exit para o call da próxima consulta e vai até o final, onde sai pelo exit da último consulta, com true e o/um valor para a variável, caso não ache, sai pelo fail e sem possível valor.
O ";" serve para voltar pelo redo e procurar por mais valores.

8) Temos o caso base:

is_set([])

e chamadas recursivas

is_set([H|T]) :-
  \+ member(H,T), % \+ -> negação
  is_set(T).

então é is checando se H não é membro do Tail, e member é:

member(X,[X|_]).
member(X,[_|Tail]) :-
  member(X,Tail).

Então membro, vai checando X com o que foi passado, pro ver se não tem repetição.

9)

?- X = Y, Y \= Z, Z = W, X = a, W = b.

Preciso comparar Y \= Z no final, após inicializar as variáveis

Correto:

?- X = Y, Z = W, X = a, W = b, Y \= Z

10) O = é unificação, ele tenta unificar rem(X, 2) e rem(Y, 2), mas não consegue/falha, pois são diferentes.

Correção

```
nome_parity(X, Y) :-
    Result1 is rem(X, 2),
    Result2 is rem(Y, 2),
    Result1 = Result2. % Agora unifico os
                       results!
```