Nome: Luís Felipe de Melo Costa Silva
Nº USP: 9297961
MAC0444 – Sistemas Baseados em Conhecimento

Lista 2

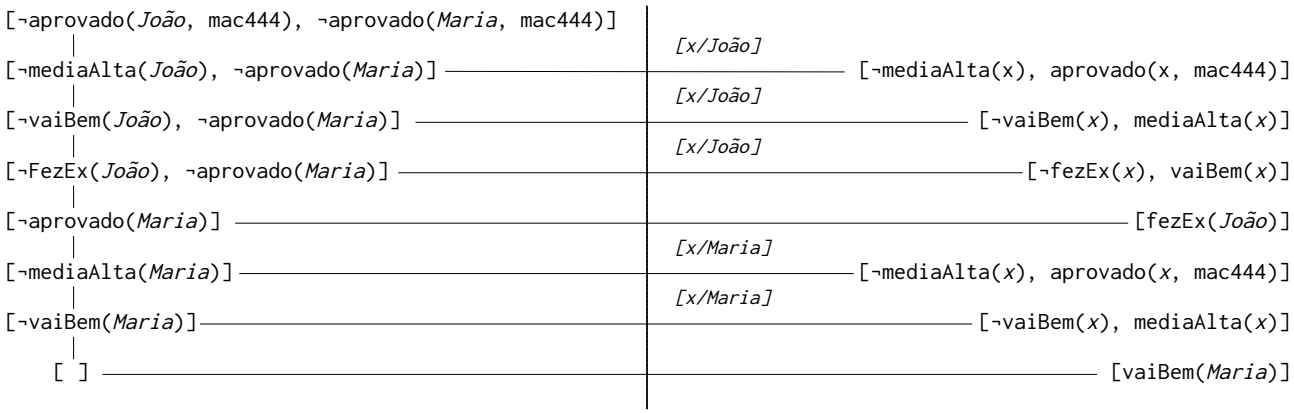
Questão 1.

A nossa base de conhecimento (KB) será formada por:

Em cláusulas de Horn:

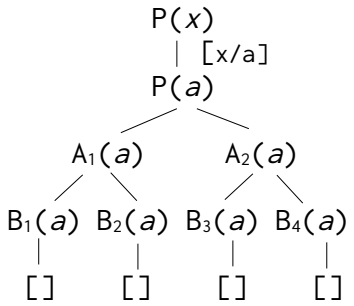
[¬fezEx(*x*), vaiBem(*x*)]
[¬vaiBem(*x*), mediaAlta(*x*)]
[¬mediaAlta(*x*), aprovado(*x*, mac444)]
[fezEx(*João*)]
[vaiBem(*Maria*)]

Para mostrar que João e Maria foram aprovado em mac444, temos que mostrar que KB ∪ {[¬aprovado(*João*, mac444), ¬aprovado(*Maria*, mac444)]} é insatisfazível. Usando **resolução SLD**:

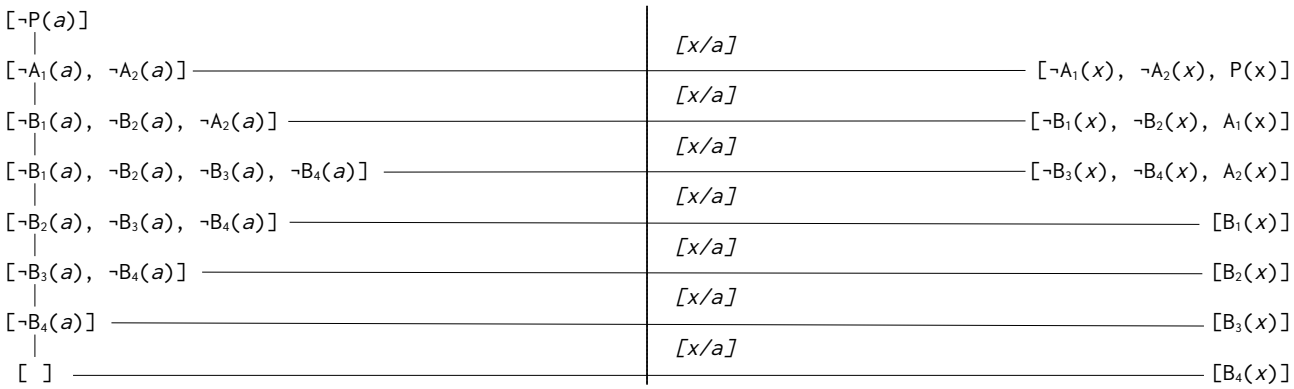


Questão 2.

a) Usando *backward-chaining*:



b) Usando resolução SLD, temos que mostrar que KB ∪ {[¬P(*a*)]} é insatisfazível.



Questão 3.

a) O resultado dessa consulta é `X = [b, d, f]`.

b) Quando chamamos `result/2` como no item **a)** estamos passando uma lista e uma variável. O programa devolve uma nova lista, baseada na que foi passada, apenas com os elementos de índices pares (se a indexação começar em 1). Ele trabalha assim: tentando casar a variável até que ela corresponda ao predicado. O programa procura o primeiro fato que satisfaz o objetivo inicial e tenta instanciar as variáveis recursivamente, até chegar a um fato do programa.

Em nossa consulta, o fato dado corresponde ao segundo caso, instanciando `X` como `[]` e `_` como `[a, b, c, d, e, f, g]`.

A chamada resultante (`result([a, b, c, d, e, f, g], [])`) cobre o primeiro caso, instanciando _ como `[a]`, `E` como `[b]` e `L` como `[c, d, e, f, g]`.

A próxima chamada é `result([c, d, e, f, g], M)`. Seguindo esse processo, chegamos em `X = .(b, .(f, .(f, [])))`, que é a lista `X = [b, d, f]`.

A variável anônima utilizada serve para fazermos instanciações cujo o valor não nos interessa, e o programa funciona do mesmo jeito, só não mostra o valor dessas instanciações. O corte utilizado faz que a primeira instanciação usada tenha que ser coberta pela segunda regra, para que não utilizemos uma lista que já tenha elementos.

Questão 4.

```
a) avof(Mul, Pess) :- mae(Mul, X), pai(X, Pess).
   avof(Mul, Pess) :- mae(Mul, X), mae(X, Pess).

b) avom(Hom, Pess) :- pai(Hom, X), pai(X, Pess).
   avom(Hom, Pess) :- pai(Hom, X), mae(X, Pess).

c) bisavom(Hom, Pess) :- pai(Hom, X), avom(X, Pess).
   bisavom(Hom, Pess) :- pai(Hom, X), avof(X, Pess).
```

d)
```
sao_irmaos(X,Y) :- irmaos(X, Y).
sao_irmaos(X,Y) :- irmaos(Y, X).

primo_1(P1, P2) :- pai(X, P1), pai(Y, P2), sao_irmaos(X, Y).
primo_1(P1, P2) :- pai(X, P1), mae(Y, P2), sao_irmaos(X, Y).
primo_1(P1, P2) :- mae(X, P1), pai(Y, P2), sao_irmaos(X, Y).
primo_1(P1, P2) :- mae(X, P1), mae(Y, P2), sao_irmaos(X, Y).
```

e)
```
primo(P1, P2) :- primo_1(P1,P2).

primo(P1, P2) :- pai(X, P1), pai(Y, P2), primo(X, Y).
primo(P1, P2) :- pai(X, P1), mae(Y, P2), primo(X, Y).
primo(P1, P2) :- mae(X, P1), pai(Y, P2), primo(X, Y).
primo(P1, P2) :- mae(X, P1), mae(Y, P2), primo(X, Y).
```

f)
```
maior_de_idade(Pess) :- idade(Pess, X), X > 17.
```

g)
```
pessoa(X) :- homem(X).
pessoa(X) :- mulher(X).
pessoas(Lista) :- findall(X, pessoa(X), Lista).
```

h)
```
mais_velho(Pess) :- idade(Pess, X), \+ (idade(_, Y), Y > X).
```

i)
```
seleciona_pessoas(Pess, I, Sexo) :- Sexo = m, homem(Pess), idade(Pess, I).
seleciona_pessoas(Pess, I, Sexo) :- Sexo = f, mulher(Pess), idade(Pess, I).
lista_pessoas(Lista, Sexo) :- findall([Pess, I], seleciona_pessoas
                                      (Pess, I,Sexo), Lista).
```

j)
```
parentes(Hom, Mul) :- sao_irmaos(Hom, Mul).
parentes(Hom, Mul) :- pai(Hom, Mul).
parentes(Hom, Mul) :- mae(Mul, Hom).
parentes(Hom, Mul) :- avom(Hom, Mul).
parentes(Hom, Mul) :- avof(Mul, Hom).
parentes(Hom, Mul) :- bisavom(Hom, Mul).
parentes(Hom, Mul) :- primo(Hom, Mul).
adequados(Hom, Mul) :- homem(Hom), mulher(Mul), idade(Hom, X), idade(Mul, Y),
                       Y < X+3, X < Y+11, \+ (parentes(Hom, Mul)),
                       \+ (casados(Hom, _)), \+ (casados(_, Mul)).
```