Lista 2

Sistemas Baseados em Conhecimento: [MAC0444]

Julia Leite - 11221797

17 de outubro de 2021

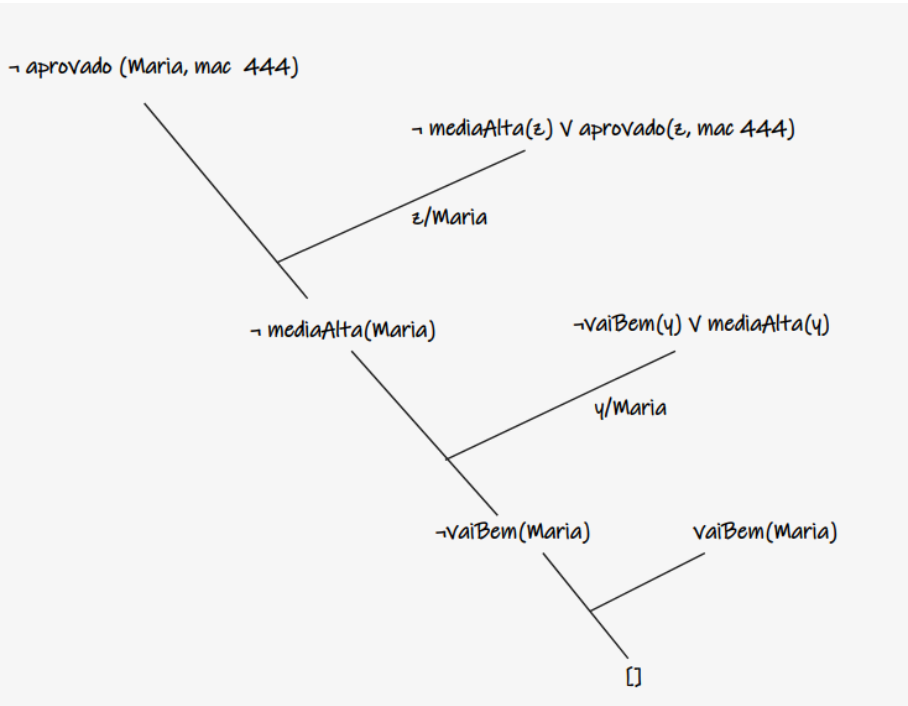
Ex 1

- vaiBem(Maria)
- fezEx(João)
- ∀x (fezEx(x) → vaiBem(x))
- ∀y (vaiBem(y) → mediaAlta(y))
- ∀z (mediaAlta(z) → aprovado(z, mac444))

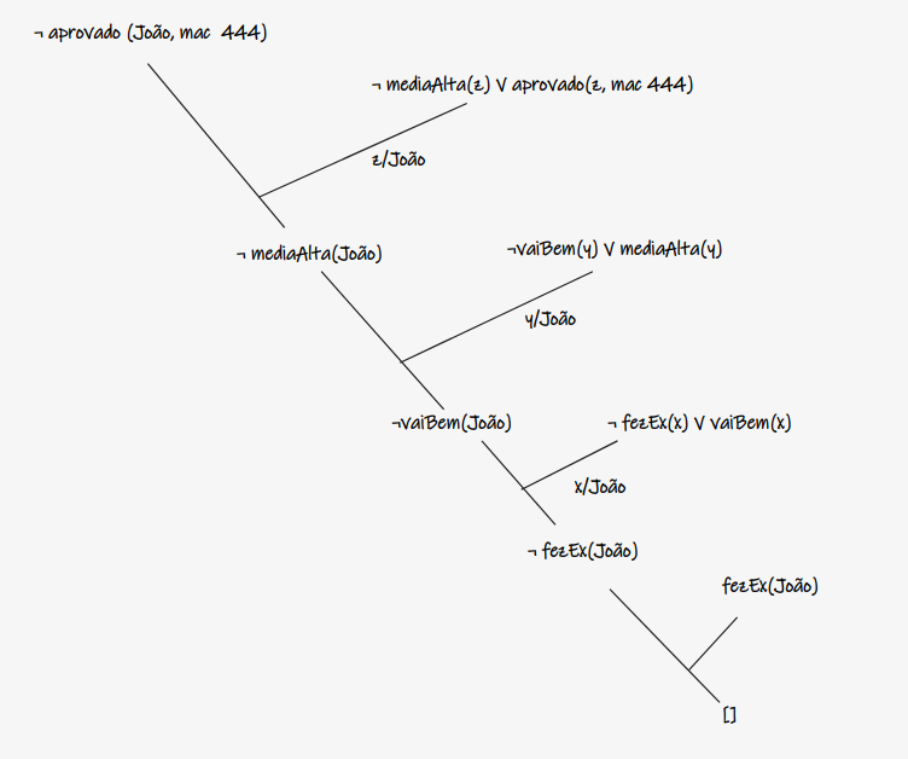
Nosso KB:

- vaiBem(Maria)
- fezEx(João)
- ¬ fezEx(x) ∨ vaiBem(x)
- ¬ vaiBem(y) ∨ mediaAlta(y)
- ¬ mediaAlta(z) ∨ aprovado(z, mac 444)

Primeiro vamos mostrar que Maria foi aprovada em mac 444, mostrando que $KB \vDash \neg aprovado(Maria, mac444)$ é Falso



Agora, analogamente, vamos provar que João foi aprovado em mac 444



Ex 2

- $[\neg A_1(x), \neg A_2(x), P(x)]$
- $[\neg B_1(x), \neg B_2(x), A_1(x)]]$
- $[\neg B_3(x), \neg B_4(x), A_2(x)]$
- $[B_1(\text{a})]$
- $[B_2(\text{a})]$
- $[B_3(\text{a})]$
- $[B_4(\text{a})]$

Vamos mostrar que o backward chaining responde SIM com objetivo P(a)

[P(a)]
[A_1(a), A_2(a)]
[B_1(a), B_2(a), A_2(a)]
[B_2(a), A_2(a)]
[A_2(a)]
[B_3(a), B_4(a)]
[B_4(a)]
[]

Ex 3

Programa:

```
result([ _ , E | L ] , [ E | M ]) : - !  , result(L , M ).
result(_ , []).
```

Consulta:

```
result ([ a , b , c , d , e , f , g ] , X ).
```

a) A resposta do prolog para a consulta seria:

```
X = [b, d, f]
```

b)

Esse programa recebe uma lista e armazena na variável passada como parâmetro (`X`) outra lista, apenas com os elementos de índice par da primeira (ou seja, o 2º elemento, 4º, ...), utilizando recursão.

A cada execução de `result` armazenamos o 2º elemento da lista recebida, e chamamos recursivamente para o restante da lista, como o 1º elemento não é utilizado, uma variável anônima é empregada para representá-lo.

Primeiro verificamos se conseguimos decompor a lista recebida como descrito acima (E, L e M), caso contrário, retornamos uma lista vazia (base da recursão).

A presença do corte, então, garante que apenas a solução com todos os elementos de índice par seja retornada, já que impede o retrocesso do Prolog no ponto de escolha anterior. Sem esse recurso, outras soluções também seriam consideradas, como apenas o 2º elemento, apenas a lista vazia, entre outras.

Ex 4

a) Regra para o predicado: `avof(Mul,Pess)` em que `Mul` seja avó de `Pess`

```
pais(X, Pess) :-
    mae( X, Pess);
    pai( X, Pess).

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

c) Regra para o predicado: `bisavom(Hom,Pess)` que é verdadeiro se `Hom` for bisavô de `Pess`.

```
avofh( Hom, Pess) :-
    pais( X, Pess), pai( Hom, X).

avos( A, Pess) :-
    avof( A, Pess);
    avofh( A, Pess).

bisavom(Hom, Pess) :-
    pai( Hom, A), avos( A, Pess).
```

d) Regra para o predicado: `primo_1(P1, P2)` que é verdadeiro se `P1` e `P2` forem primos em primeiro grau.

```
primo_1(P1, P2) :-
    avos( A, P1), avos( A, P2), not((pais(P, P1), pais(P, P2))).
```

f) Regra para o predicado: `maior_de_idade(Pess)` que é verdadeiro se `Pess` for maior de idade.

```
maior_de_idade(Pess) :-
    idade( Pess, N), N >= 18.
```

g) Regra para o predicado: `pessoas(Lista)` que devolve a `Lista` de todas as pessoas existentes na base de conhecimento

```
is_person(Pess) :-
    mulher(Pess);
    homem(Pess).

pessoas(Lista):-
    findall(X,is_person(X), Lista).
```

h) Regra para o predicado: `mais_velho(Pess)` que retorna a pessoa mais velha que consta na base de conhecimento.

```
mais_velho(Pess):-
    is_person(Pess), not((idade(Pess, I), idade(_, Age), Age > I)), !.
```

i) Regra para o predicado: `lista_pessoas(Lista, Sexo)` que retorna uma `Lista` de todas as pessoas do `Sexo` indicado `(m/f)`, incluindo as suas respectivas idades

```
lista_pessoas(Lista, Sexo) :-
    (Sexo = f, findall([X, I], (mulher(X), idade(X, I)), Lista)), !;
    (Sexo = m, findall([X,I], (homem(X), idade(X, I)), Lista)).
```

j) Regra para o predicado: `adequados(Hom, Mul)` que é verdadeiro se `Hom` for um homem, `Mul` for uma mulher e o homem for (no máximo) 2 anos mais novo do que a mulher ou 10 anos mais velho do que ela e se ambos não tiverem nenhuma relação de parentesco nem nenhum deles for casado.

```
bisavoh(Mul, Pess) :-
    mae( Mul, A), avos( A, Pess).

parente(Hom, Mul):-
    pais(Hom, Mul); pais(Mul, Hom);
    irmaos(Hom, Mul);
    primo_1(Hom, Mul);
    avos(Mul,Hom); avos(Hom,Mul);
    bisavom(Hom,Mul);
    bisavoh(Mul, Hom).

adequados(Hom, Mul) :-
    homem(Hom), mulher(Mul),
    idade(Hom, IH), idade(Mul, IM),
    IH >= IM - 2, IH =< IM + 10,
    not(casados(Hom, _)), not(casados(_, Mul)),
    not(parente(Hom, Mul)).
```

Obs: consideramos parentes pais, irmãos, avó ou avô, bisavô e bisavó e primos de primeiro grau.