# MAC0444 - Sistemas Baseados em Conhecimento
# Lista de Exercícios No. 2

Mateus Agostinho dos Anjos
NUSP 9298191

30 de Setembro de 2019

**1 -**

Predicados:
$fezEx(x)$ = x fez os exercícios
$vaiBem(x)$ = x vai bem na prova
$mediaAlta(x)$ = x fica com media alta
$aprovado(x, y)$ = x é aprovado em y

Formalizando as sentenças do enunciado chegamos em:

$$\forall x\ (fezEx(x) \rightarrow vaiBem(x))$$

$$\forall y\ (vaiBem(y) \rightarrow mediaAlta(y))$$

$$\forall z\ (mediaAlta(z) \rightarrow aprovado(z, mac444))$$

$$fezEx(\text{João})$$

$$vaiBem(\text{Maria})$$

Base de conhecimento (KB):

1. $[\neg fezEx(x),\ vaiBem(x)]$
2. $[\neg vaiBem(y),\ mediaAlta(y)]$
3. $[\neg mediaAlta(z),\ aprovado(z, mac444)]$
4. $[fezEx(\text{João})]$
5. $[vaiBem(\text{Maria})]$
6. $[\neg aprovado(\text{João}, mac444)]$

Veja que inserimos $[\neg aprovado(\text{João}, mac444)]$ na base de conhecimento, pois é a negação do nosso objetivo. Sendo assim, se chegarmos na cláusula vazia a partir desta base de conhecimento estará provado que $aprovado(\text{João}, mac444)$ é consequência lógica das sentenças do enunciado.

Utilizando a **resolução SLD** temos:

$$
\begin{array}{cc}
\neg aprovado(\text{João}, mac444) & \text{(resolve com 3. e z/João)} \\
\downarrow & \\
\neg mediaAlta(\text{João}) & \text{(resolve com 2. e y/João)} \\
\downarrow & \\
\neg vaiBem(\text{João}) & \text{(resolve com 1. e x/João)} \\
\downarrow & \\
\neg fezEx(\text{João}) & \text{(resolve com 4.)} \\
\downarrow & \\
[\ ] &
\end{array}
$$

Sendo assim provamos que: $KB \cup \{\neg aprovado(\text{João}, mac444)\}$ é insatisfazível, portanto $aprovado(\text{João}, mac444)$ é consequência lógica de nossa base de conhecimento.

A **resolução SLD** será semelhante para Maria, portanto temos:
Base de conhecimento (KB):

1. $[\neg fezEx(x),\ vaiBem(x)]$
2. $[\neg vaiBem(y),\ mediaAlta(y)]$
3. $[\neg mediaAlta(z),\ aprovado(z, mac444)]$
4. $[fezEx(\text{João})]$
5. $[vaiBem(\text{Maria})]$
6. $[\neg aprovado(\text{Maria}, mac444)]$

Utilizando a **resolução SLD** temos:

$$
\begin{array}{cc}
\neg aprovado(\text{Maria}, mac444) & \text{(resolve com 3. e z/Maria)} \\
\downarrow & \\
\neg mediaAlta(\text{Maria}) & \text{(resolve com 2. e y/Maria)} \\
\downarrow & \\
\neg vaiBem(\text{Maria}) & \text{(resolve com 5.)} \\
\downarrow & \\
[\,] &
\end{array}
$$

**2 -**

Temos a Base de Conhecimento (KB) reescrita com variáveis renomeadas para evitar confusões na resolução do exercício:

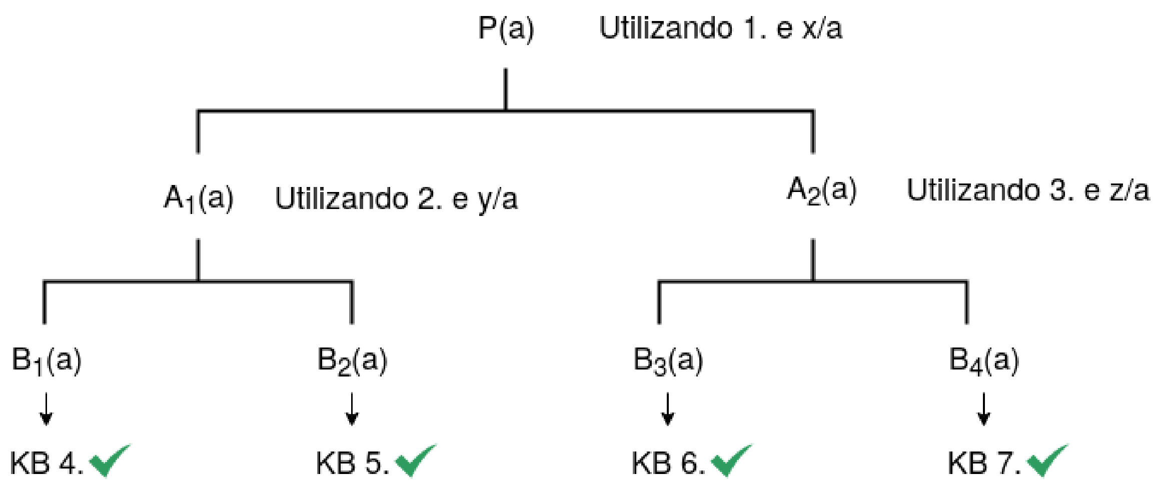
$$
\begin{array}{ll}
1. & [\neg A_1(x), \neg A_2(x), P(x)] \\
2. & [\neg B_1(y), \neg B_2(y), A_1(y)] \\
3. & [\neg B_3(z), \neg B_4(z), A_2(z)] \\
4. & [B_1(a)] \\
5. & [B_2(a)] \\
6. & [B_3(a)] \\
7. & [B_4(a)]
\end{array}
$$

**a)**

Para mostrar o passo a passo do procedimento de encadeamento para trás (backward chaining) devemos começar identificando as implicações da Base de Conhecimento.
Seguindo a ordem acima temos:
(note que utilizamos $\leftarrow$ nas implicações)

$$
\begin{array}{ll}
1. & \forall x(P(x) \leftarrow A_1(x) \wedge A_2(x)) \\
2. & \forall y(A_1(y) \leftarrow B_1(y) \wedge B_2(y)) \\
3. & \forall z(A_2(z) \leftarrow B_3(z) \wedge B_4(z)) \\
4. & B_1(a) \\
5. & B_2(a) \\
6. & B_3(a) \\
7. & B_4(a)
\end{array}
$$

A partir destas implicações o passo a passo pode ser mostrado a partir da figura abaixo, sendo que cada passo gera pelo menos uma sub-árvore.




Todas as folhas da árvore estão na base de conhecimento em uma cláusula unitária, portanto são verdadeiras. Sendo assim podemos marcá-las com um ✓.
A partir disso podemos concluir que $A_1(a)$ e $A_2(a)$ estão provados e depois que $P(a)$ está provado, portanto mostramos que o encadeamento para trás (backward chaining) produz resposta SIM com objetivo P(a).

**b)**

Utilizando a **resolução SLD** com a base de conhecimento definida no início da questão, iniciamos com $\neg P(a)$, que é a negação do nosso objetivo, e buscaremos a cláusula vazia.
Obtemos o seguinte:

$$
\begin{array}{cc}
[\neg P(a)] & \text{(resolve com 1. e x/a)} \\
\downarrow & \\
[\neg A_1(a), \neg A_2(a)] & \text{(resolve com 2. e y/a)} \\
\downarrow & \\
[\neg B_1(y), \neg B_2(y), \neg A_2(a)] & \text{(resolve com 3. e z/a)} \\
\downarrow & \\
[\neg B_1(a), \neg B_2(a), \neg B_3(a), \neg B_4(a)] & \text{(resolve com 4.)} \\
\downarrow & \\
[\neg B_2(a), \neg B_3(a), \neg B_4(a)] & \text{(resolve com 5.)} \\
\downarrow & \\
[\neg B_3(a), \neg B_4(a)] & \text{(resolve com 6.)} \\
\downarrow & \\
[\neg B_4(a)] & \text{(resolve com 7.)} \\
\downarrow & \\
[\;] &
\end{array}
$$

Como chegamos na cláusula vazia a partir de $\neg P(a)$, então está provado que $P(a)$ é consequência desta base de conhecimento.

**3 -**

**a)**

Após ter carregado o programa a resposta do Prolog para a consulta:
$?-result([a, b, c, d, e, f, g], X).$
será:
$X = [b, d, f]$

**b)**

Considerando que a lista é enumerada a partir da posição 1, o programa recebe uma lista e elimina os elementos das posições ímpares, devolvendo apenas os elementos das posições pares.
Veja o exemplo de consulta que os elementos da lista coincidem com o número de sua posição:
$?-result([1, 2, 3, 4, 5, 6], X).$
$X = [2, 4, 6]$
Para fazer isso o programa possui um fato, $result(\_, [\;])$, que cobre os casos em que o primeiro argumento é uma lista vazia ou

uma lista com apenas 1 elemento, pois nestes casos não é possível extrair 2 elementos da lista como a primeira regra exige (o corte impede a utilização do fato quando a lista tem 2 ou mais elementos).
Definida a base do programa a partir deste fato, chamadas recursivas serão feitas tentando equiparar, inicialmente, a lista passada como argumento e a primeira regra, veja a execução do exemplo:

| | |
|---|---|
| Chamada inicial | $result([1, 2, 3, 4, 5, 6], X)$ |
| Casa com | $result([\_, E\|L], [E\|M])$ |
| Com valoração | $\_ = 1,\ E = 2,\ L = [3, 4, 5, 6],\ X = [2\|M_1]$ |
| Faz chamada recursiva | $result(L, M_1)$ |
| | |
| Chamada | $result([3, 4, 5, 6], M_1)$ |
| Casa com | $result([\_, E\|L], [E\|M])$ |
| Com valoração | $\_ = 3,\ E = 4,\ L = [5, 6],\ M_1 = [4\|M_2]$ |
| Faz chamada recursiva | $result(L, M_2)$ |
| | |
| Chamada | $result([5, 6], M_2)$ |
| Casa com | $result([\_, E\|L], [E\|M])$ |
| Com valoração | $\_ = 5,\ E = 6,\ L = [\ ],\ M_2 = [6\|M_3]$ |
| Faz chamada recursiva | $result(L, M_3)$ |
| | |
| Chamada | $result([\ ], M_3)$ |
| Casa com o fato | $result(\_, [\ ])$ |
| Com valoração | $\_ = [\ ],\ M_3 = [\ ]$ |

Após essa execução devemos obter o valor de $X$, para isso temos que reconstruí-lo a partir de $M_3$, $M_2$ e $M_1$, veja:

| | | |
|---|---|---|
| $M_3 =$ | $[\ ]$ | $= [\ \ ]$ |
| $M_2 =$ | $[6\|M_3]$ | $= [6]$ |
| $M_1 =$ | $[4\|M_2]$ | $= [4, 6]$ |
| $X =$ | $[2\|M_1]$ | $= [2, 4, 6]$ |

Esta execução única só é possível, pois o corte (!) na primeira linha do programa faz com que não seja possível criar ramificações para obter diferentes respostas casando as chamadas recursivas intermediárias com o fato, uma vez que já foi utilizado a primeira regra de casamento (que possui a instrução de corte).

Sendo assim, o corte impede a alternativa de resposta em que o programa casa a chamada $result([3,4,5,6], M_1)$ com o fato $result(\_, [\,])$ ($\_ = [3,4,5,6]$ e $M_1 = [\,]$) e obtém a resposta $X = [2]$, por exemplo.
Note que foi utilizado a variável anônima (_), pois não queremos saber qual o valor do elemento que foi atribuído a ela durante o processo de obtenção do valor de $X$, queremos somente que exista um valor possível a ser atribuído.

**4 -**

**a)**

$avof(Mul, Pess) :- mae(Mul, Y),\ mae(Y, Pess).$
$avof(Mul, Pess) :- mae(Mul, Y),\ pai(Y, Pess).$

**b)**

$avom(Hom, Pess) :- pai(Hom, Y),\ mae(Y, Pess).$
$avom(Hom, Pess) :- pai(Hom, Y),\ pai(Y, Pess).$

**c)**

$bisavom(Hom, Pess) :- pai(Hom, Y),\ avom(Y, Pess).$
$bisavom(Hom, Pess) :- pai(Hom, Y),\ avof(Y, Pess).$

**d)**

Primeiro definimos que se X é irmão de Y então Y é irmão de X:

$irmao\_de(X, Y) :- irmaos(X, Y).$
$irmao\_de(X, Y) :- irmaos(Y, X).$

Agora podemos definir primo de primeiro grau se P1 e P2 não forem irmãos e tiverem avô ou avó em comum (P1 e P2 não podem ser iguais):

$primo\_1(P1, P2) :- avom(X, P1),\ avom(X, P2),\ not(irmao\_de(P1, P2)),$
$not(P1 = P2).$
$primo\_1(P1, P2) :- avof(X, P1),\ avof(X, P2),\ not(irmao\_de(P1, P2)),$

$$not(P1 = P2).$$

**e)**

**f)**

**g)**

**h)**

**i)**

**j)**