Lista de Linguagens de Programação – 21

Nome: _____ Matrícula: _____

1. Consideremos a funo `resolution` que vimos em aula:

   - function `resolution(clause, goals)`
     - seja `sub` = MGU de `head(clause)` e `head(goals)`
     - retorna (`(sub(tail(clause)` concatenado com `tail(goals))`)

   Com base neste interpretador, qual é o MGU usado nas resoluções abaixo, e qual é a nova lista de metas que será criada?

   (a) `resolution([p, q, r], [p, x, y])`

   (b) `resolution([p(a)], [p(X)])`

   (c) `resolution([p(X, b), q(X)], [p(a, Y), r(Y)])`

   (d) `resolution([append(X, Y, [1, 2]), [append([], B, B)])`

2. Considere o predicado `append` abaixo:

```
append([], B, B)
append([Head|TailA], B, [Head|TailC]) :-
  append(TailA, B, TailC).
```

Mostre um traço de resolução para a chamada `solve([append(X, Y, [1, 2])])`. Não pare após o primeiro sucesso – mostre todas as chamadas de `solve`. Lembre-se de renomear as variáveis antes de cada chamada da clausula `append`.

3. Cúmulo é um joguinho de sorte e raciocínio rápido. Dado uma lista de números L, e um *alvo* (M, N), o objetivo do jogo é encontrar uma combinação de M números de L cuja soma mais se aproxime de N. As combinações não podem conter elementos repetidos. Normalmente cúmulo é jogado entre um grupo de participantes. A cada rodada, participantes acumulam pontos, onde os pontos de um participante são a distância de sua combinação de M números até o objetivo N. A distância é o valor absoluto da diferença entre N e a soma da combinação escolhida. Ao final de um número de rodadas, ganha aquele participante que houver acumulado o menor número. Nesta questão faremos um programa em Prolog para resolver uma rodada de cúmulos. O objetivo é criar um predicado `cumulo(L, M, N, Result, Distance)` que é verdadeiro quando `Result` é a combinação de M elementos tomados de L mais próxima de N, e `Distance` é a distância de `Result` até N. O nosso predicado está escrito abaixo:

```
cumulo(L, M, N, Result, Distance) :-
  allCombs(L, M, ListCombinations),
  findClosest(ListCombinations, N, Result),
  distance(Result, N, Distance).
```

Por exemplo:

```
?- cumulo([7, 11, 13, 17, 19, 23, 29], 4, 52, Result, Distance).
Result = [7, 11, 13, 19],
Distance = 2
```

Parte do exercício já está resolvida. O problema agora é preencher as lacunas:

(a) Crie um predicado `sumList(L, S)`, que seja verdadeiro quando `S` for a soma de todos os números da lista `L`.

(b) Crie um predicado `distance(L, N, D)` que seja verdadeiro quando `D` for o valor absoluto da diferença entre a soma dos elements de `L` e `N`.

(c) Nós precisamos de um predicado `allCombs(L, M, ListResults)` que seja verdadeiro quando *ListResults* for uma lista contendo todas as possíveis combinações de M elementos tomados de L. Este predicado é definido assim:

```
allCombs(L, M, ListResults) :- findall(C, comb(M, L, C), ListResults).
```

`allCombs` usa um predicado `comb(N, L, C)`, que é verdadeiro quando `C` é uma combinação de N elementos tomados da lista L. Defina `comb`.

(d) Finalmente, precisamos de um predicado `findClosest(L, N, X)`, que seja verdadeiro quando X for a lista de menor distância de N dentre as listas contidas em L. Escreva este predicado.

4. Considere o predicado `reverse` abaixo:

```
reverse([], [])
reverse([Head|Tail], X) :-
  reverse(Tail, Y), append(Y, [Head], X).
```

Desenhe a árvore de prova para a busca `reverse(X, [1, 2])`. Como nos *slides* do livro, cada novo *solve* com um termo `reverse` como a primeira meta deve ter dois filhos, um para cada uma das cláusulas `reverse`. Cada nodo *solve* que tem um termo `append` como a primeira chamada terá somente um filho: ou um nodo *nothing*, se o termo textttappend não puder ser resolvido, ou um nodo *solve*, com o textttappend resolvido, e o restante das metas substituídas a contento.

Lembre-se de renomear variáveis antes de cada cláusula do programa. A árvore para `reverse(X, [1, 2])` é uma árvore infinita. Assim, quando você ver que um ramo da árvore não vai ter fim, você pode cortá-lo, explicando isto. Com base nesta árvore de prova, explique a seguinte chamada:

```
?- reverse(X, [1, 2, 3, 4]).

X = [4, 3, 2, 1]

Action (h for help) ? a
% Execution Aborted
?-
```