UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA EM PROLOG PARA CONTAGEM DE ÁTOMOS EM LISTAS

Discentes:

Caio L. R. S. Ueno
Gabriel C. P. Mendes

Docente:

Dra. Heloisa de Arruda Camargo

Ciência da Computação

Disciplina: Paradigmas de Linguagem de Programação

São Carlos / SP

18 de dezembro de 2020

1 Código-fonte

Para a realização do projeto, programou-se o código mostrado na Figura 1, esse está disponível no Github.

Figura 1 – Código fonte em Prolog.

```
empty([]).
removeSubList([],[]):- !.
removeSubList([X|Y],R):- is_list(X), not(empty(X)), removeSubList(X,Z),
                         removeSubList(Y,W), append(Z,W,R), !.
removeSubList([X|Y],[X|Z]):- removeSubList(Y,Z).

removeVars([],[]):- !.
removeVars([X|Y],R):- var(X), removeVars(Y,R), !.
removeVars([X|Y],[X|Z]):- removeVars(Y,Z).

belongsTo(Elem, [Elem|_]):- !.
belongsTo(Elem, [_|Y]):- belongsTo(Elem, Y).

sharedElements([], _, []).
sharedElements([A|B], C, [A|D]):- atomic(A), belongsTo(A, C), sharedElements(B, C, D), !.
sharedElements([_|B], C, D):- sharedElements(B, C, D), !.

concatSharedElements(A, B, C):- sharedElements(A, B, X), sharedElements(B, A, Y), append(X, Y, C).

removeAll(_,[],[],0):- !.
removeAll(Elem,[Elem|T],L,N):- removeAll(Elem,T,L,N1), N is N1+1, !.
removeAll(Elem,[X|T1],[X|T2],N):- removeAll(Elem,T1,T2,N).
countElem([],[]):- !.
countElem([X|T],[[X,N]|C]):- removeAll(X,[X|T],L,N), countElem(L,C).

conta_atomos(L1, L2, Lout):- removeSubList(L1,L1rem), removeSubList(L2,L2rem),
                             removeVars(L1rem,L1mod), removeVars(L2rem,L2mod), concatSharedElements(L1mod, L2mod, L), countElem(L,Lout).

conta_atomos_read(L1, L2, Lout):- read(L1), read(L2), conta_atomos(L1, L2, Lout).
```

Fonte – Criada pelos autores.

2 Explicação dos predicados definidos

2.1 removeSubList

Regra cujo resultado consiste na eliminação de listas internas, ou seja, elementos pertencentes às listas internas são colocados no primeiro nível, listas vazias são consideradas como um elemento. Primeiramente, é verificado se a lista passada está vazia, caso esteja, seu resultado é uma lista vazia. Caso contrário, segundamente, a cabeça da lista fornecida é verificada, caso seja uma lista e não seja uma lista vazia (verificada pelo fato *empty*), tanto a cabeça quanto a cauda são passadas –separadamente– para a mesma regra (chamada recursiva) e, após, as listas resultantes são concatenadas. Por fim, caso o elemento da cabeça da lista não seja uma lista, ele é inserido na cabeça da lista resultante e a cauda é passada para a mesma regra (chamada recursiva).

2.2 removeVars

Regra que recebe uma lista e elimina suas variáveis. Primeiramente, é verificado se a lista é vazia, caso seja, não há variáveis a serem removidas. Caso não seja uma lista vazia, é verificado se a cabeça da lista é uma variável, caso seja, a cauda é passada para a mesma regra (chamada recursiva). Por fim, caso a cabeça não seja uma variável, ela é inserida na cabeça da lista resultante e a cauda da lista de entrada é passada para a mesma regra (chamada recursiva).

2.3 belongsTo

Regra para verificar se o primeiro termo (*Elem*) pertence ao segundo termo - uma lista. Para tal, verifica se *Elem* é a cabeça da lista - se for, então ele pertence -, caso contrário, verifica se *Elem* pertence a cauda da lista.

2.4 sharedElements

Dado dois termos - duas listas - verifica para cada elemento da primeira lista se este pertence a segunda, e também seguindo o critério de ser atômico. Caso o elemento não faça parte da segunda lista, chama recursivamente o predicado, mas agora com a cauda da primeira lista e a segunda lista intacta. O terceiro termo será uma lista com os elementos da primeira lista - com repetição - que também estão na segunda lista.

2.5 concatSharedElements

Utilizando-se do predicado anterior, identifica quais elementos da primeira lista estão na segunda e vice-versa, concatenando-as ao final. Com isso, o terceiro termo será uma lista com as repetições dos elementos que ambas as listas possuem.

2.6 removeAll

Regra para remover todas as ocorrências de um dado elemento em uma dada lista, contando quantas remoções ocorreram. Para tal, primeiramente, é verificado se a lista está vazia, caso esteja, a lista resultante é vazia e a contagem dos elementos removidos é 0 (zero). Segundamente, caso a lista não seja vazia, é verificado se a cabeça da lista é o elemento que busca-se remover, caso seja, há a chamada recursiva da regra passando a cauda da lista e incermenta-se o contador de remoções. Por fim, caso o elemento não seja o mesmo da cabeça da lista, então, ele é concatenado à lista resultante e a cauda é passada na chamada recursiva da regra.

2.7 countElem

Regra para criar uma lista, cujos elementos consistem em uma sub-lista do tipo: [*Elemento*, *Quantidade de ocorrências*], essa recebe como entrada a lista contendo apenas os elementos que ocorrem em ambas as listas de entrada do programa e em quantidade igual a soma das ocorrências das duas listas.

2.8 conta_atomos

Predicado que implementa a lógica do trabalho utilizando dos predicados auxiliares definidos anteriormente.

2.9 conta_atomos_read

Predicado auxiliar que permite o usuário digitar as listas em tempo de execução do predicado.

3 Exemplos

A Figura 2 ilusta três exemplos de uso do programa feito.

Figura 2 – Exemplos quanto ao Funcionamento do Programa.

```
?- conta_atomos([a, b, Z,[a, w], [5,x], [x], [],par(c,d)],  [4.6, 5, w, [a,5], F, [], par(c,d), [], par(1,2)], L).
L = [[a, 3], [w, 2], [5, 3], [[], 3]].

?- conta_atomos([X, M, ['Maria', [b], b], 'Maria'], [1, [b, [b, [b]], 'Maria']], L).
L = [['Maria', 3], [b, 5]].

?- conta_atomos([[[], [], [], [[],[]]], [[], [[],[]]], L).
L = [[[], 8]].
```

Fonte – Criada pelos autores.