Universidade de Évora

Curso de Engenharia Informática

Relatório do Trabalho Prático de Programação III

Códigos Ambíguos

Leonardo Catarro, 43025

Diogo Solipa, 43071

Linguagem Escolhida

Para o desenvolvimento deste trabalho foi-nos dada a opção de escolha entre as linguagens Prolog e OCaml. Nós decidimos escolher Prolog, visto que foi a linguagem mais abortada e que mais tempo ocupou na unidade curricular de Programação III.

Organização do Código

O nosso trabalho é constituído por 16 cláusulas.

```
ambiguo(Input, M, T1, T2):-
    findDuplicate(1,4, Input, _, _, _, _, _,_, Out2),
    Out2 = [(H1,H2), (H3,H2)],
    append([], H2, M),
    append([], H3, T1),
    append([], H1, T2), !.
```

Recebe a lista de tuplos, em Input, e após a execução da claúsula findDuplicate(), retorna no formato de Out2, a mensagem codificada M em H2, e as duas representações ambíguas possíveis T1, T2 em H1 e H3.

```
findDuplicate(Start, End, Set, L, CodesList, SymbolList, CodeFlatList, TupleOfLists, Out, Out2) :-
    findall(TupleOfLists, allPerms(Start, End, Set, L, CodesList, SymbolList, CodeFlatList, TupleOfLists), Out),
    iterFindRepeat2(Out, Out, Out2).

allPerms(Start, End, Set, L, CodesList, SymbolList, CodeFlatList, TupleOfLists):-
    isort(Set, Sortedset),
    loop(Start, End, Index),
    perm(Index, Sortedset, L, CodesList, SymbolList, CodeFlatList, TupleOfLists).
```

O findDuplicate(), recorre a uma clausula do Prolog findall() e ao predicado allPerms(), que após ordenação pelo tamanho dos códigos representativos da lista de input fornecida nos retorna uma lista com todas as combinações possíveis de tamanho passado em Start até tamanho passado em End.

Recorrendo de seguida ao iterFindRepeat2() que “chama” iterFindRepeat(), e estas 2 clausulas simulam um Nested For para a verificação da existência de códigos repetidos(ambíguos) na lista de combinações existentes.

```
iterFindRepeat([], [], []).
iterFindRepeat(X, [H|_], Out) :- cmpTupleList(X,H), append([], [H,X], Out), !.
iterFindRepeat(X, [_|T], Out) :- iterFindRepeat(X, T, Out).

iterFindRepeat2([], [], []).
iterFindRepeat2([H|_], L, Out) :- iterFindRepeat(H, L, Out).
iterFindRepeat2([_|T], L, Out) :- iterFindRepeat2(T, L, Out).
```

perm(). Esta clausula é a responsável por nos dar as permutações possíveis de tamanho N.

```
eval([],_).
eval([H|T],Set):- member(H,Set), eval(T,Set).

perm(N,Set,L, CodesList, SymbolList, CodeFlatList, TupleOfLists):-
    length(L,N),
    eval(L,Set),
    listsCopyAndUnite(L, CodesList, SymbolList, CodeFlatList, TupleOfLists).
```

isort(). Para a ordenação das lista, tal como foi acima mencionado, usamos o algoritmo isort(), sort por inserção. Essa inserção era definir atráves da clausula cmptuple(), clausula essa que compara o tamanho das listas dos 2 tuplos passados, retornando o tuplo com lista de menor tamanho.

```
isort(I, S) :- isort(I, [], S).
isort([], S, S).
isort([X|Xs], SI, SO) :- insord(X, SI, SX), isort(Xs, SX, SO).

insord(X, [], [X]).
insord(X, [A|As], [X,A|As]) :- cmptuple(X,A,X).
insord(X, [A|As], [A|AAs]) :- \+cmptuple(X,A,X), insord(X, As, AAs).
```

```
cmptuple(X, Y, X) :- secondpostuplelen(X, Xlen), secondpostuplelen(Y, Ylen), Xlen<Ylen.
```

Foram ainda usadas algumas clausulas auxiliares como:

→unite(), que recebe um simbolo e um código, unindo-os num tuplo.

```
unite([], [], []).
unite(L1, L2, (L1,L2)).
```

→symbolCopy(), listCodeCopy(), clausulas que copiam listas, a primeira usada para copiar apenas as primeiras posições dos tuplos, posição dos símbolos e a segunda para copiar apenas as listas representativas, segunda posição dos tuplos.

```
symbolCopy([], []).
symbolCopy([(H,_)|T1], [H|T2]) :- symbolCopy(T1,T2).
```

```
listCodeCopy([], []).
listCodeCopy([(_,H)|T1], [H|T2]) :- listCodeCopy(T1,T2).
```

→cmplists() e cmpTupleLists(), clausulas para comparar listas de códigos e p ara comparar listas de tuplos, respetivamente.

```
cmplists([],[]).
cmplists([H|T], [H1|T1]) :- H=H1, cmplists(T,T1).

cmpTupleList([], []).
cmpTupleList((L,H), (L1,H1)) :- L\=L1, cmplists(H, H1).
```

listlen(), clausula que retorna o tamanho de uma lista

```
listlen([],0).
listlen([_|T], Len) :- listlen(T, Y), Len is Y+1.
```

Limites de funcionamento

Relativamente aos dois exemplos fornecidos pelo professor, não obtivemos quaisquer limites, visto que o nosso código permite restringir o tamanho das combinações, não originando um ciclo infinito.

Daí que, os limites de funcionamento do trabalho são os limites por nós impostos.