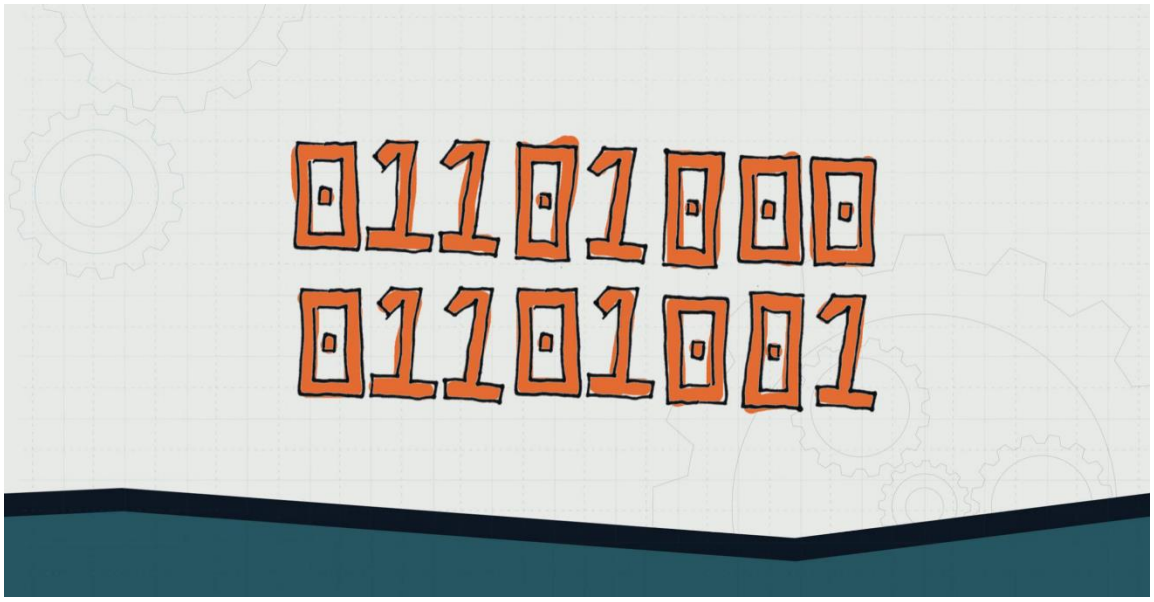

Relatório Códigos Ambíguos

Trabalho Prático de Programação III

Docentes:

Prof Pedro Patinho

Prof Salvador Abreu

Trabalho de:

Denis Lapuste nº42616

Marcelo Feliz nº38073

Índice:

Introdução:

Este trabalho consiste em escrever um programa em *Prolog* que recebe um código ambíguo e devolva a mensagem codificada mais curta (caso haja várias, a primeira por ordem lexicográfica*) que pode ser interpretada de forma ambígua e de duas das suas possíveis interpretações.

Como para a realização do trabalho foi escolhida a linguagem *Prolog*, mesmo tendo vários predicados o predicado final (ambíguo/4), que recebe no primeiro argumento o código e devolve no segundo a mensagem codificada encontrada e nos dois restantes duas das suas possíveis interpretações.

*No desenvolvimento explica a ordem lexicográfica usada.

Desenvolvimento:

Para a realização do programa, optou-se começar pela ordenação dos códigos de cada letra ordenados por tamanho e caso o este seja idêntico em dois ou mais códigos, a ordenação leva em consideração a ordenação crescente de valores binários, ou seja, para “1001” e “1100” o segundo é considerado maior. Para a criação desta função assumimos que não existem códigos iguais para letras diferentes, pois se isto acontecesse não iria de encontro com o objetivo do trabalho.

Primeiramente, foi criado um predicado “*indexOf*/3” que dado um elemento retorna a sua posição na lista recebida, ou retorna o elemento dada a posição.

Foi criado também o predicado “*len*/2” que dada uma lista retorna o seu tamanho.

Dois predicados auxiliares criados: “*equalOB*/2” e “*bigger*/2” que dados dois *arrays* e como o nome indica, a primeira retorna *no* se o primeiro argumento for menor e *yes* caso contrário e a segunda retorna *no* se o primeiro argumento for menor ou igual e *yes* caso contrário.

O “*first*/2” decide qual é maior ou menor dados dois elementos, levando em consideração a ordem lexicográfica mencionada anteriormente.

O “*calc*/2” aplica o predicado anterior para elementos do tipo (a, [0,1,1,0]), de forma à comparação só ser feita com os códigos.

“*isort*/2” foi o predicado dado nas aulas de ordenação, adaptado pelo grupo para ordenação da forma desejada.

Seguidamente foi implementado o predicado “*fuse*/2” que devolve no terceiro argumento a lista gerada pela fusão das duas

listas dadas como argumentos e o "*combinations*/3" que dado um elemento (seja ele único ou não) faz a combinação com todos os elementos da lista dada como segundo argumento.

"*cb*/3" irá usar o predicado anterior de forma a criar todas as combinações só com os elementos das duas listas dadas recursivamente, o "*equals*/2" que como o nome indica diz se dois elementos são iguais ou não, e "*finalCombin*/4" que faz todas as combinações possíveis com n elementos (sendo n <= 4, este valor pode ser alterado) recursivamente.

"*ex*/2", e "*convert*/2" servem ambos para guardar só o código e descartar a letra, a diferença é que a primeira é usada só para um elemento, a segunda para a lista, "*ex2*/2", e "*convert2*/2", têm o mesmo propósito que as anteriores, mas guarda a letra e descarta o código.

"*addPar*/2", e "*addPar2*/2" foram 2 predicados adicionais para corrigir um output.

"*calcCobs_NC*/2", e "*saveAll_NC*/2" servem para calcular todas as combinações de letras apagando os parêntesis indesejados com o "*flatten*/2" (predicado interno do prolog) e guardar todas as respostas com o "*findall*/3", respetivamente.

"*calcCobs*/2", e "*saveAll*/2" servem para calcular todas as combinações de códigos apagando os parêntesis indesejados com o "*flatten*/2" (predicado interno do prolog) e guardar todas as respostas com o "*findall*/3", respetivamente.

"*findDups*/3", encontra os duplicados todos usando "*select*/3" e "*member*/2" (predicados internos do prolog), "*interupt*/3" que pára quando encontra o primeiro duplicado, "*returnEv*/4" que retorna ambas as posições na lista que são iguais e têm índice diferente.

E finalmente "*ambiguo*/4" (explicação na introdução).

Problemas encontrados:

Um dos problemas encontrados inicialmente foi depois de calculadas todas as possibilidades estas ficavam formatadas de forma errada e com demasiados parenteses.

O nosso output estava a ser possibilidades infinitas de combinações, e percebemos depois que não podemos estar a testar combinações infinitas, portanto para resolver esta parte decidimos limitar as combinações até palavras de tamanho 4 (alterável).

Depois destes deparámo-nos com a dificuldade de tentar encontrar 2 códigos iguais nas respostas que eram retornadas, decidimos, portanto, usar um predicado para guardar tudo numa lista e depois comparar os resultados entre si, ficando a saber os indexes das posições iguais ficou fácil aceder-lhes depois.

Outro problema menos relevante era o *output* do trabalho não ser igualzinho para o primeiro exemplo, já que como "c" não era uma lista os parênteses retos não apareciam como no enunciado do trabalho era mostrado, os predicados "*addPar*/2", e "*addPar2*/2" corrigiram este problema.

Finalmente, antes da existência dos predicados "*calcCobs_NC*/2" e "*calcCobs*/2" existia só um predicado que fazia ambas, mas como o grau de complexidade era quadrático, e por isso o trabalho ia ser mais lento e menos eficiente, decidiu-se separar ambas de forma à complexidade ser 2x linear em vez de quadrática.

Conclusão:

Com a realização deste trabalho pudemos aprender um pouco mais sobre como funciona a programação em Prolog e a lógica por detrás da linguagem, em especial a criação de combinações. Com isto achamos que todos os objetivos do trabalho foram bem alcançados.