Programação III - Trabalho Prático

Autores:
Filipe Alfaiate - l 43315
Vasco Barnabé - l 42819

31 de Janeiro de 2021

1 Introdução

Este projeto consiste na implementação de um programa que receba 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 duas das suas possíveis interpretações. Este programa foi desenvolvido para resolver um conflito que surge quando duas palavras distintas são portadoras do mesmo código(sequência de bits).

Para este problema foi utilizada a linguagem OCaml(Programação Funcional) uma vez que considerámos ser esta a linguagem(em comparação com Prolog) com maior grau de semelhança relativamente às linguagens de programação mais comuns, com as quais temos maior familiarização.

2 Implementação

Para resolver o problema em questão, é necessário verificar constantemente a presença de ambiguidades, partindo sempre da comparação de tuplos da lista inicialmente fornecida e avançando concatenando os vários elementos da lista entre si. Por fim, o objetivo é devolver a mensagem codificada mais curta, que pode ser interpretada de forma ambígua e duas das suas possíveis interpretações.

Assim, para apresentar o resultado final pretendido, foram implementadas várias funções:

- **converterLista lista** - esta função converte uma lista inteira de tuplos;
- **converte (lista, arr)** - esta função converte um caracter de um tuplo para uma string;
- **recverificacao lista** - esta função verifica em toda a lista se existem ambiguidades. Caso não haja, produz novas concatenações para verificar de novo sobre a existência de ambiguidades;
- **semrepetir lista** - esta função concatena a lista com as concatenações já efetuadas sem repetir tuplos iguais;

- **loopexterior lista1 lista2** - esta função envia o elemento que tem de ser concatenado;
- **loopinterior lista1 cabeça lista2** - esta função é responsável pela concatenação de um elemento de um elemento da lista com todos os elementos da lista;
- **ambiguoexterior lista** - esta função vai percorrendo a lista recebida como argumento e a cada elemento da lista, envia este para a função ambiguointerior onde se vai verificar se existe algum tuplo igual. Além disso, envia também a lista completa(que vai ser percorrida para comparar elementos) e um array vazio onde vão ser guardados os valores iguais ao tuplo enviado;
- **ambiguointerior lista1 cabeça lista2** - esta função verifica se um certo tuplo recebido tem um tuplo igual (é necessário verificar se a sequência de bits é igual e os caracteres correspondentes são diferentes). Se estas condições se verificarem, se a lista com tuplos guardados estiver vazia, o tuplo em questão é guardado nessa lista. De seguida, se existir mesmo um tuplo igual verifica-se qual dos dois tem a sequência menor, para se guardar o tuplo com a menor sequência. Se, tendo em conta a condição mais recentemente apresentada, o tuplo não tiver sido substituído, vai-se verificar se os comprimentos dos caracteres dos tuplos são iguais e se algum tem a sequência com menor comprimento. Se sim, guarda-se o tuplo com a sequência de menor comprimento, uma vez que o objetivo deste programa é, no fim, devolver a mensagem codificada mais curta. Caso as condições acima não se verifiquem, avança-se na lista para efetuar estas mesmas verificações sobre o próximo tuplo.
- **juntartuplo (caracter1, sequencia1)(caracter2, sequencia2)** - esta função executa a concatenação de dois tuplos, ex: (caracter1+caracter2), (sequencia1+sequencia2);
- **retornaprimeiralista** - esta função retorna o primeiro elemento da lista recebida
- **retornaultimolista lista** - função responsável por retornar o último elemento da lista recebida;
- **final[a;b]** - função que coloca os tuplos iguais no formato desejado e devolve assim, o output pretendido.

Foram também implementadas algumas funções auxiliares para dividir os tuplos, para que deste modo fosse possível isolar o primeiro caracter ou a sequência de bits(segundo elemento do tuplo) para trabalhar:

- **dividirtuplosequencia** - devolve a sequência(segundo elemento do tuplo);
- **dividirtuplocaracter** - devolve o caracter(primeiro elemento do tuplo).

Além destas, foram criadas mais funções auxiliares, estas com a responsabilidade de apresentar o output num formato específico:

- **printlistint lista** - função responsável por apresentar uma lista de inteiros (utilizada para apresentar a sequência de bits);
- **printliststring lista** - função responsável por apresentar uma lista de strings(utilizada para apresentar as interpretações devolvidas);
- **printlista lista** - função responsável por apresentar uma lista;
- **printfinal (a,b,c)** - esta função apresenta, por fim, o output no formato pretendido, um triplo contendo a mensagem codificada encontrada(a) e duas das suas possíveis interpretações(b e c).

3 Limitações do Programa

Este programa apresenta algumas limitações de funcionamento, tais como:

1. As listas(código fornecido) recebidas estão implementadas no programa, não sendo estas fornecidas como um input usual através do terminal;

2. Não foi criada uma função **ambíguo** que receba como argumento o código e devolva um triplo contendo a mensagem codificada encontrada e duas das suas possíveis interpretações. Em vez disso, foram criadas outras funções para desempenhar estas funções, tendo sido criada a função **printfinal (a,b,c)** que devolve a mensagem encontrada (a) e duas das suas possíveis interpretações (b e c).

4 Conclusão

Por fim, consideramos que o desenvolvimento deste programa permitiu colocar em prática várias funcionalidades da Programação Funcional, mais especificamente em OCaml, principalmente no que diz respeito à manipulação de listas e do seu conteúdo, tendo sido atingido o objetivo deste projeto, devolver a mensagem codificada mais curta que pode ser interpretada de forma ambígua e duas das suas possíveis interpretações, após a receção de um código ambíguo. Ainda que com as limitações já apresentadas, o programa consegue resolver os dois exemplos fornecidos, tendo sido o objetivo deste projeto cumprido.