Lista 02 - LIP

Aluno Gabriel Teixeira Queiroz Damasceno - 565118

Questão 1) `fun {Abs X} if X<0 then ~X else X end end`

A função acima não funciona devido a um erro de tipo ocasionado pela comparação de X - um valor real, isto é, um float - a 0, um número inteiro. Daí, para corrigir a função, deve-se comparar X a um ponto flutuante, da seguinte forma:

```
fun {Abs X} if X<0.0 then ~X else X end end
```

Questão 2) Método de Newton-Raphson para cálculo de raiz cúbica.
-> Código no arquivo oz <-

Questão 3) Método da bisseção para cálculo de raízes de equações.
-> Código no arquivo oz <-

Questão 4) Fatorial iterativo.
-> Código no arquivo oz <-

Questão 5) SumList iterativo.
-> Código no arquivo oz <-

Questão 7)

```
fun {Append Ls Ms}
    case Ms
    of nil then Ls
    [] X|Mr then {Append {Append Ls [X]} Mr}
    end
end
```

O código acima tenta concatenar duas listas arbitrárias Ls e Ms realizando a recursão sobre o segundo argumento, entretanto o programa não funciona devido ao trecho `{Append Ls [X]}`, visto que o segundo argumento [X] nunca é vazio, isto é, nunca chega ao caso base, gerando um loop que ocasiona erro de excesso de

memória. O que acontece é que a chamada recursiva mais interna sempre adiciona a cabeça X de Ms, contida em uma lista unitária, à cauda da lista Ls, de forma que X permanece sempre com o mesmo valor. Uma prova de que o algoritmo acima não funciona diz respeito à quebra da propriedade garantida pela indução: a propriedade do append vale para listas menores do que Ls, sendo que, na versão cuja recursão é feita sobre o segundo argumento, Ls não é diminuído até um caso base, e sim aumentado, o que desrespeita a hipótese indutiva.

Questão 8)

-> Código no arquivo oz <-

Questão 10) `fun {Leaf X} X\=(_|_) end`

Ao utilizar a versão do Leaf mostrada acima, existe o risco de o programa suspender durante a execução, isto porque, caso X seja uma variável não ligada, não será possível realizar a comparação `X \= (_|_)`, uma vez que o operador de igualdade/diferença exige variáveis ligadas a valores. Para contornar esse problema, utiliza-se o case, que compara somente a estrutura das variáveis por meio do pattern matching, sem requisitar seus valores.

Questão 11)

Não é possível realizar o append, isto é, concatenar listas diferença, mais de uma vez. Isso acontece devido às variáveis de atribuição única. Inicialmente, antes de serem concatenadas, as listas diferença são constituídas por um par L|X - que constitui um valor parcial, possibilitado pelo dataflow -, em que L representa uma lista e X representa uma variável não ligada. Para aplicar o append a duas listas diferença, basta ligar a variável não ligada da primeira lista ao início da segunda, fazendo com que X aponte para tal. Dessa maneira, elas são concatenadas. Entretanto, não há como aplicar o append em L|X novamente, visto que X já deixou de ser uma variável não ligada, e está apontando para uma lista.

Questão 14)

Item A)

Considere:

```
fun {Insert Q X}
    case Q of q(N S E) then E1 in E=X|E1 q(N+1 S E1)
    end
end
```

e

```
fun {Delete Q X}
    case Q of q(N S E) then S1 in S=X|S1 q(N-1 S1 E)
    end
end
```

Ao remover um elemento de uma fila vazia, N passa a valer -1, ocasionando a chamada "remoção antecipada", isto é, ao inserir qualquer elemento nessa fila, ele é removido automaticamente, e a fila volta ao estado de fila vazia. Após, ela continua funcionando normalmente. O que acontece de fato é que S, ao realizar a remoção antecipada, é uma variável não ligada, e, devido ao comportamento dataflow, S aguarda um valor. Daí, ao adicionar um elemento Y qualquer, como inicialmente temos S == E, a variável S se liga a Y|E1 e o exclui automaticamente.

Item B) `fun {IsEmpty q(N S E)} S==E end`

Não é possível definir o IsEmpty desta maneira, visto que o operador de igualdade exige que os operandos estejam sempre ligados a valores. Caso contrário, o programa irá suspender a execução. Portanto, o ideal é utilizar a condição N == 0, uma vez que N recebe valor desde a criação da fila (N = 0), garantindo que a comparação possa ser realizada.

Questão 15) Escrever QuickSort usando lista diferença.

-> Código no arquivo oz <-

Questão 16) Convolução com chamada pela cauda.

```
fun {ConvolucaoIter Xs Ys}
    case Xs
    of nil then nil
    [] X|Xr then
        case Ys of Y|Yr then
            (X#Y)|{ConvolucaoIter Xr Yr}
        end
    end
end
fun {Convolucao Xs Ys}
    {ConvolucaoIter Xs {Reverse Ys}}
end
```

Explicação:

Primeiramente, entendamos o objetivo do programa: o algoritmo deve receber duas listas de mesmo tamanho, Xs = [x1 x2 ··· xn] e Ys = [y1 y2 ··· yn], e realizar a convolução delas, de forma que seja gerado o seguinte resultado: [x1#yn x2#yn−1 ··· xn#y1]. Para tal, temos a função ConvolucaoIter, que recebe as duas listas, Xs e Ys, e retorna uma lista de tuplas, na qual cada tupla é um par x#y, x sendo um elemento de Xs e y sendo um elemento de Ys de posição correspondente.

Funcionamento de ConvolucaoIter: caso Xs esteja vazia, então significa que ambas as listas são nil - visto que elas têm o mesmo número de elementos -, e a convolução entre elas também é uma lista vazia. Daí o programa retorna nil. Caso Xs e Ys sejam listas com cabeça e cauda, então o algoritmo junta as cabeças de Xs e Ys em uma tupla (X#Y) e aplica a função ConvolucaoIter à cauda das listas por meio de uma chamada recursiva, de tal maneira que (X#Y) e a chamada recursiva são operandos do operador cons '|', representando cabeça e cauda, respectivamente, da lista de retorno, de forma que o resultado final seja uma lista de tuplas [x1#y1 x2#y2 ··· xn#yn].

Função Convolucao: por fim, para realizar a convolução, a função Convolucao inicializa a função ConvolucaoIter com Xs e o reverso de Ys, de forma que o

resultado final do programa seja a lista de tuplas [x1#yn x2#yn−1 ··· xn#y1], como desejado.