Aluno: Carlos Victor Martins da Silva

Matrícula: 476516

| **Questões feitas** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **7** | **8** | **9** | **11** |
| X | X | X | X | X | X | X | X | X |

1. **Absolute value of real numbers.** We would like to define a function *Abs* that calculates the absolute value of a real number. The following definition does not work:

fun {Abs X} if X<0 then ~X else X end end

Why not? How would you correct it? Hint: the problem is trivial.

Resposta:

| A definição está incorreta pois na linguagem Oz não é possível comparar Integers com Floats através de operadores sem realizar a conversão, sendo assim, impossível a função receber Floats no parâmetro X e seguir o comportamento esperado. Uma implementação correta para a função *Abs* seria:  fun {Abs X}  % if X < 0 then ~X - Oz não permite comparar Floats e Ints.  if X < 0.0 then ~X % Para corrigir a implementação basta substituir "0" (int) por "0.0" (float)  else X end  end  {Browse {Abs ~1.1}} % Resultado esperado: 1.1  {Browse {Abs 0.0}} % Resultado esperado: 0.0  {Browse {Abs 1.1}} % Resultado esperado: 1.1 |
| --- |

2. **Cube roots.** This chapter uses Newton’s method to calculate square roots. The method can be extended to calculate roots of any degree. For example, the following method calculates cube roots. Given a guess **g** for the cube root of **x**, an improved guess is given by **(x/g² + 2g)/3**. Write a declarative program to calculate cube roots using Newton’s method.

Resposta:

| fun {CubeRoot X}  fun {Abs X}  if X < 0.0 then ~X  else X end  end  % “Chute”  Guess = 1.0    % Checa se o variação é próxima o suficiente para que se retorne resultado  fun {CloseEnough Guess}  ({Abs X - (Guess \* Guess \* Guess)}) / X < 0.000001 % Margem de erro aceita  end    % Aproxima o “chute”  fun {Approximate Guess}  ( X / (Guess \* Guess) + 2.0 \* Guess ) / 3.0  end  fun {CubeRootIter Guess}  if {CloseEnough Guess} then Guess  else {CubeRootIter {Approximate Guess}} end  end  in  {CubeRootIter Guess}  end |
| --- |

3. **The half-interval method.** ²¹The half-interval method is a simple but powerful technique for finding roots of the equation **f(x) = 0**, where **f** is a continuous real function. The idea is that, if we are given points **a** and **b** such that **f(a) < 0 < f(b)**, then **f** must have at least one root between **a** and **b**. To locate a root, let **x = (a + b)/2** and compute **f(x)**. If **f(x) > 0** then f must have a root between **a** and **x**. If **f(x) < 0** then **f** must have a root between **x** and **b**. Repeating this process will define smaller and smaller intervals that converge on a root. Write a declarative program to solve this problem using the techniques of iterative computation.

Resposta:

| Antes de construirmos a função é necessário destrinchar a descrição feita no enunciado da questão. Para encontrarmos o f(x) = 0 de uma função pode ser encontrado (por aproximação) se dando dois valores a e b que definem um intervalo [a, b] onde o método irá fazer a busca. Lembrando que deve existir pelo menos uma raiz dentro desse intervalo. A cada execução um novo valor de x será calculado com a equação x = (a + b) / 2 para reduzir o intervalo seguindo a seguinte regra:  Se x > 0, o novo intervalo é [a, x]  Se não, o novo intervalo é [x, b]  Ao repetirmos esse processo, iremos reduzir cada vez mais o espaço do intervalo até encontrar o valor aproximado ou exato da raiz.  Uma implementação interativa para esse problema é:  fun {BisectionMethod F A B}  % Checa se já está próximo o suficiente  fun {CloseEnough V}  {Abs V} < 0.00001 % Determina o nível de precisão do método  end  % Faz a aproximação  fun {Approximate A B V X}  if V > 0.0 then A # X  else X # B end  end  % O método descrito no enunciado da questão  % é chamado de Método da Bisseção  fun {BisectionMethodIter A B}  X = (A + B) / 2.0  V = {F X}  in  if {CloseEnough V} then X  else A1 B1 in  A1 # B1 = {Approximate A B V X}  {BisectionMethodIter A1 B1}  end  end  in  {BisectionMethodIter A B}  end |
| --- |

4. **Iterative factorial.** This chapter gives a definition of factorial whose maximum stack depth is proportional to the input argument. Give another definition of factorial which results in an iterative computation. Use the technique of state transformations from an initial state, as shown in the IterLength example.

Resposta:

| fun {Fact N}  fun {FactIter N L}  if N == 1 then L  else {FactIter N-1 N \* L} end  end  in  {FactIter N 1}  end |
| --- |

5. **An iterative SumList.** Rewrite the function *SumList* of Section 3.4.2 to be iterative using the techniques developed for Length.

Resposta:

| fun {SumList Xs}  fun {SumListIter Ys N} case Ys of nil then N [] Y|Yr then {SumListIter Yr N + Y} end end  in  {SumListIter Xs 0}  end |
| --- |

7. Checking if something is a list. Section 3.4.3 defines a function *LengthL* that calculates the number of elements in a nested list. To see whether X is a list or not, *LengthL* uses the function Leaf defined in this way:

fun {Leaf X} case X of \_|\_ then false else true end end

What happens if we replace this by the following definition:

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

What goes wrong if we use this version of *Leaf*?

Resposta:

| É possível reescrever a 2° função Leaf como:  fun {Leaf X}  if X == (\_ | \_) then false  else true end  end  A implementação da função Leaf pode ser bloqueada pois embora mantenha a mesma estrutura condicional que a primeira (que utiliza um case), na hora de checar a inclusão de elementos na lista usando o operador “\=” que tenta executar a comparação do valor X ao indefinido gera um bloqueio na execução da função e, consequentemente, um loop infinito na execução da função. |
| --- |

8. **Another append function.** Section 3.4.2 defines the Append function by doing recursion on the first argument. What happens if we try to do recursion on the second argument? Here is a possible solution:

fun {Append Ls Ms}

case Ms

of nil then Ls

[] X|Mr then {Append {Append Ls [X]} Mr}

end

end

Is this program correct? Does it terminate? Why or why not?

Resposta:

| O programa não está correto, ao executá-lo é fácil notar que ele entra em uma recursão infinita pois o segundo argumento da função nunca será nil, por consequência, a função continuará sua execução até estourar a memória alocada na máquina virtual do ambiente Mozart.  Uma implementação correta para essa função append é:  fun {AppendFix Ls Ms}  case Ls  of nil then Ms  [] L | Lr then % Basta trocar a posição dos parâmetros  L | {AppendFix Lr Ms} % e “desencapsular” o 2° parâmetro para fora da nova lista gerada  end  end |
| --- |

9. **An iterative append.** This exercises explores the expressive power of dataflow variables. In the declarative model, the following definition of append is iterative:

fun {Append Xs Ys}

case Xs

of nil then Ys

[] X|Xr then X|{Append Xr Ys}

end

end

We can see this by looking at the expansion:

proc {Append Xs Ys ?Zs}

case Xs

of nil then Zs = Ys

[] X|Xr then Zr in

Zs = X|Zr

{Append Xr Ys Zr}

end

end

This can do a last call optimization because the unbound variable **Zr** can be put in the list **Zs** and bound later. Now let us restrict the computation model to calculate with values only. How can we write an iterative append? One approach is to define two functions:

(1) an iterative list reversal

(2) an iterative function that appends the reverse of a list to another.

Write an iterative append using this approach.

Resposta:

| % (1) Implementação do método de inversão de lista interativo  declare Reverse  fun {Reverse Xs}  fun {ReverseIter Rs Ys}  case Ys  of nil then Rs  [] Y|Yr then {ReverseIter Y|Rs Yr}  end  end  in  {ReverseIter nil Xs}  end  % (2) Uma função interativa que une o inverso de uma lista a outra.  declare ReverseAppend  fun {ReverseAppend Rs Ys}  proc {AppendIter Xs Ys ?Zs}  case Xs  of nil then Zs = Ys  [] X|Xr then Zr in  Zs = X|Zr  {AppendIter Xr Ys Zr}  end  end  in  {AppendIter {Reverse Ys} Rs}  end |
| --- |

11. **Limitations of difference lists.** What goes wrong when trying to append the same difference list more than once?

Resposta:

| Unir uma lista a outra é basicamente pegar a ponta final da lista e apontar para a outra lista que se deseja unir de forma a gerar uma lista maior, quando se faz isso com uma lista de diferencia por uma vez se gera uma lista de diferencia apontando para a mesma lista, quando se é feito pela 2° vez acontece um erro de ligação, ao final, a lista fica quebrada e desconectada. |
| --- |