UNIVERSIDADE FEDERAL DO CEARÁ – CAMPUS SOBRAL

CURSO DE ENGENHARIA DA COMPUTAÇÃO

DISCIPLINA: PARADIGMAS DE LINGUAGEM DE PROGRAMAÇÃO

PROFESSOR: IÁLIS CAVALCANTE DE PAULA JUNIOR

Trabalho Final

FRANCISCO EVANDRO RIBEIRO MARTINS FILHO	**385143**
FRANCISCO JEFFERSON MARQUES DE SOUSA	**475486**
FRANCISCO RENATO GRACIANO FREIRE	**428131**
GABRIEL ALBUQUERQUE ARAÚJO	**427418**
IONARA BRANDÃO SANT'ANNA	**389107**
YURI KAUÊ ARAÚJO SAMPAIO	**418905**

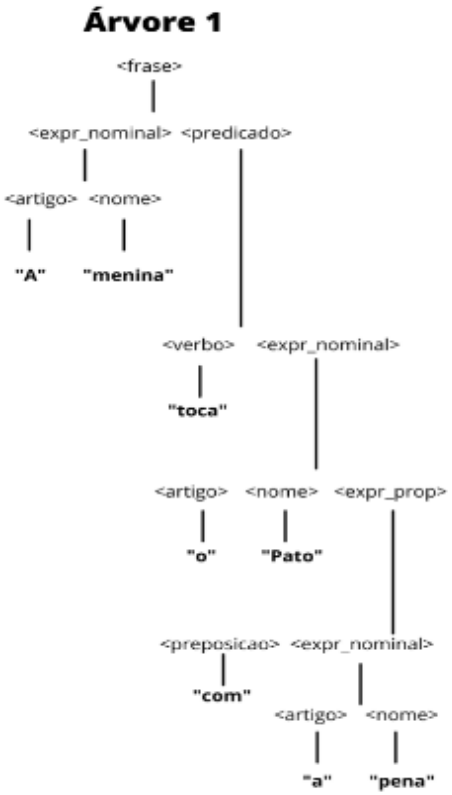
SOBRAL-CE
2021

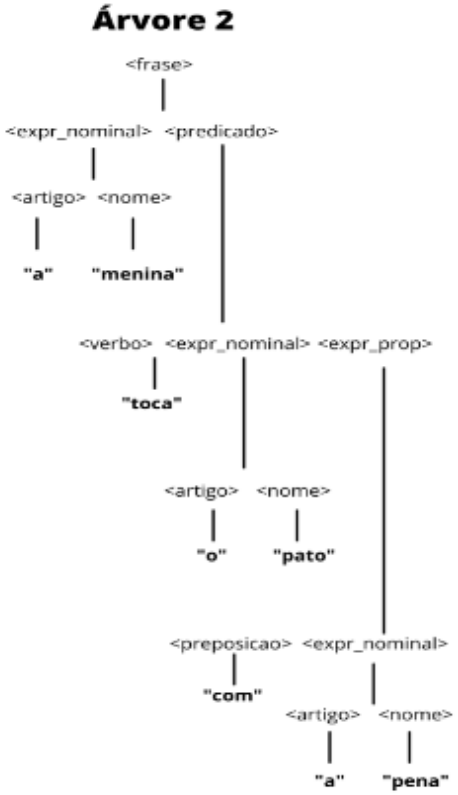
I-1) Esta questão refere-se à gramática abaixo:

```
<frase> ::= <expr_nominal> <predicado>
<expr_nominal> ::= <artigo> <nome>
                 | <artigo> <nome> <expr_propos>
<predicado> ::= <verbo>
              | <verbo> <expr_nominal>
              | <verbo> <expr_nominal> <expr_propos>
<expr_prepos> ::= <preposicao> <expr_nominal>
<nome> ::= "menino" | "menina" | "pato" | "telescopio" | "musica" |
"pena"
<preposicao> ::= "com" | "ate"
<verbo> ::= "viu" | "esta" | "e" | "canta" | "surpreende" | "toca"
<artigo> ::= "um" | "uma" | "o" | "a"
```

Uma gramática é ambígua quando uma mesma sentença possui duas ou mais árvores de derivação diferentes. Construa duas árvores de derivação distintas para a seguinte sentença: "A menina toca o pato com a pena" e verifique se a mesma é ambígua ou não.

I-1) Resolução: As duas árvores são ambíguas, pois possuem duas árvores de derivação diferentes na sentença "predicado".





I-2) Considere a seguinte gramática:

<S> → a <S>c<B> | <A> | b

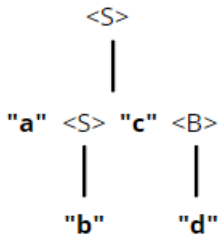
<A> → c <A> | c

<B> → d | <A>

Qual das seguintes sentenças está na linguagem gerada por essa gramática? Mostre a sua derivação (se possível).

a) **abcd**
b) **acccbcc**

abcd



Resolução: (a) É a sentença gerada pela gramática do item I-2). Não é possível fazer sua derivação.

II - Desenvolvimento com Linguagem Lambda Cálculo e Haskell

II-1) Compute as expressões lambda:

a) **((+ 3) 2)**

$+\ 3\ 2\ =\ 5$

b) **(λx.(λy. – y x)) 4 5**

$(\lambda x.\ (\ \lambda y.\ -\ y\ x))\ 4\ 5 \rightarrow^{\beta} (\lambda y. -\ y\ 4)5 \rightarrow^{\beta} -\ 5\ 4\ =\ 1$

c) **(λx. λy. + x((λy. – 3) y)) 5 6**

$(\lambda x.\ \lambda y. +\ x((\lambda y. - 3)\ y))\ 5\ 6 \rightarrow^{\beta} (\lambda x. \lambda y. +\ x((-\ 3)\ y))\ 5\ 6 \rightarrow^{\beta} (\lambda x. +\ x(-\ 3)\ 6) \rightarrow^{\beta} 5(-\ 3)\ 6\ =$

d) **(λxy. x2 + y) 4 3 (relembrando que: λxy ≡ λx λy)**

$(\lambda xy.\ \ x^2\ +\ \ y)\ 4\ 3 \rightarrow (\lambda x \lambda y.\ x^2 + y)\ 4\ 3 \rightarrow (\lambda x.\ x^2 + 3)4 \rightarrow 4^2 + 3 = 19$

e) (λyx. x2 + y) 4 3

$$(\lambda yx.\ x2\ +\ y)\ 4\ 3\ \rightarrow^{\beta} (\lambda x\lambda y.\ x^{2}\ +\ y)\ 4\ 3\ \rightarrow^{\beta} (\lambda y.\ 4^{2}\ +\ y)3 \rightarrow^{\beta} 4^{3}\ +\ 3\ =\ 19$$

II-2) Aplique as reduções (α, β e η) nas abstrações lambda:

a) (λf. f 3)(λx. + x 1)

$$(\lambda f.\ f3)((\lambda x.+\ x1)\ \rightarrow^{\beta} (\lambda x.\ \ +\ \ x\ 1)3 \rightarrow^{\beta}\ +\ \ 3\ 1\ \ =\ 4$$

b) (λx. x (λy. x a))

$$(\lambda x.\ x\ (\lambda y.\ x\ a)) \rightarrow^{\alpha} (\lambda x.\ x\ (\lambda y.\ x\ a) \rightarrow^{\eta} \lambda y.\ x\ a$$

c) ((λx. λy. (y x) z) w)

$$((\lambda x.\ \lambda y.\ (y\ x)\ z)\ w)\ \ =\ \ (\lambda x.\ \lambda y.\ y\ x\ z)\ w \rightarrow^{\eta} (\lambda x.\ \lambda y)w \rightarrow^{\beta} w\ z$$

d) (λx. λy. (y x) y)

$$((\lambda x.\ \lambda y.\ (y\ x)\ y)\ \ =\ \ (\lambda x.\ \lambda y.\ y\ x\ y) \rightarrow^{\eta} (\lambda y.\ y\ y)$$

II-3) Reduza as expressões avaliando-as e descrevendo conforme suas definições.

a) (λ x.2*x + 1) 3

$\rightarrow (2\ *\ 3\ +\ 1)$

$\rightarrow 6\ \ +\ \ 1$

$\rightarrow 7$

b) (λ xy.x+y) 5 7

$\rightarrow (\lambda xy.\ x\ +\ y)\ 5\ 7\ \rightarrow\ Projeção\ (\lambda xy.\ x)$

$\rightarrow (\lambda x.\ \lambda y.\ x\ +\ y)\ 5\ 7$

$\rightarrow (\lambda y.\ 5\ +\ y)\ \ 7$

$\rightarrow 5\ +\ 7$

$\rightarrow 12$

c) (λ yx.x+y) 5 7

$\rightarrow (\lambda\ yx.\ x\ +\ y)\ 5\ 7\ \ ->\ \ Projeção\ (\lambda yx.\ x)$

$\rightarrow (\lambda y\lambda x.\ x\ +\ y)\ 5\ 7$

$\rightarrow (\lambda x.\ x\ +\ 5)\ 7$

$\rightarrow 7\ +\ 5$

$\rightarrow 12$

d) (λ xy.x-y) (λz.z/2)

$\rightarrow (\lambda\ xy.\ x\ -\ y)\ (\lambda z.\ z/2)$

$\rightarrow (\lambda x.\ \lambda y.\ x\ -\ y)\ (\lambda z.\ z/2)\ Incompleto$

$\rightarrow (\lambda y.\ (\lambda z.\ z/2)\ -\ y)$

e) (λ x.xx) (λ y.y)

$\rightarrow (\lambda\ x.\ xx)\ (\lambda\ y.\ y)$

$\rightarrow (\lambda\ y.\ y)(\lambda\ y.\ y)$

$\rightarrow \lambda y.\ y$

$\rightarrow y$

f) (λ x.λ y. x + ((λ x.8) - y)) 5 6

$\rightarrow \lambda y.\ 5\ +\ ((\lambda x.8)\ -\ y)6$

$\rightarrow 5\ +\ ((\lambda x.8)\ -\ 6)$

$\rightarrow 5\ +\ (8\ -\ 6)$

$\rightarrow 5\ +\ 2$

$\rightarrow 7$

g) (λ x.λ y. – x y) 9

$\rightarrow (\lambda x.\lambda y.\ -x\ y)\ 9$

$\rightarrow (\lambda y.\ -9\ y)$

$\rightarrow -9\ y$

h) (λ x.xx) (λ x.xx)

$\rightarrow (\lambda x.xx)\ (\lambda x.xx)\ Se\ repete$

→...

i) (λ x.xxy) (λ x.xxy)

$\rightarrow ((\lambda x.xxy)(\lambda x.xxy)y)$

$\rightarrow (\lambda x.xxy)(\lambda x.xxy)yy$

$\rightarrow (\lambda x.xxy)(\lambda x.xxy)yyy$

$\rightarrow (\lambda x.xxy)(\lambda x.xxy)yyyy$

II-4) Defina a função *transpõe* que recebe uma tabela de 2 dimensões (x, y) de tamanho arbitrário e retorna outra tabela que corresponde à transposição da tabela recebida em argumento.

```
transpoe :: [[a]] -> [[a]]
transpoe ([]:_) = []
transpoe mat = (map head mat) :transpoe (map tail mat)
```

II-5) Um palíndromo é uma lista que tem a mesma sequência de elementos quando é lida tanto da esquerda para a direita quanto da direita para a esquerda. Defina uma função *palindromop*, que recebe uma lista para ver se é um palíndromo. Se for, retorna TRUE; caso contrário, retorna NIL.

```
palindromop :: Eq a => [a] -> Bool
palindromop lista
        | (lista == reverse lista) = True
        | otherwise = False

{- reverse é uma operação padrão de listas e
serve para inverter uma lista -}
```

II-7) Defina uma função que calcule o resultado da exponenciação inteira, xy, sem recorrer a funções pré-definidas.

```
exponenciacao :: Int -> Int -> Int
exponenciacao a 0 = 1
exponenciacao a b = a * exponenciacao a (b-1)
```

II-8) Defina uma função que:

a) dados quatro números inteiros, determine a sua média aritmética;

```
media_ari :: Int -> Int -> Int -> Int -> Int
media_ari a b c d = div (a +b + c + d) 4

main = do
    print(media_ari 8 10 1 2)
```

b) dados três números inteiros retorne um par em que o primeiro elemento é o maior dos números e o segundo elemento é o menor dos números;

```
maior :: Int -> Int -> Int -> Int
maior a b c
    | a>b && a>c = a
    | b>a && b>c = b
    | c>a && c>b = c

menor :: Int -> Int -> Int -> Int
menor a b c
    | a<b && a<c = a
    | b<a && b<c = b
    | c<a && c<b = c

compara :: Int -> Int -> Int -> [Int]
compara a b c = [maior a b c, menor a b c]

main = do
    print(compara 8 10 1)
```

c) dado um triplo de números inteiros retorna um triplo em que os mesmos números inteiros estão ordenados por ordem crescente.

```
min1 :: [Int] -> Int
min1 [x] = x
min1 (x:y:ys) | x <= y = min1 (x:ys)
              | otherwise = min1 (y:ys)
```

```
rmv :: Int -> [Int] -> [Int]
rmv x [] = []
rmv x (y:ys) | x == y = ys
             | otherwise = y:(rmv x ys)

ssort :: [Int] -> [Int]
ssort [] = []
ssort xs = m:(ssort ys)
         where m = min1 xs
               ys = rmv m xs

main = do
    print (ssort[5,3,4])
```

II-9) Defina uma função que repete que recebe um inteiro n e uma string s, e retorna uma string que é a repetição n vezes da string de entrada.

```
repete ::  Int -> String -> String
repete 0 s = []
repete 1 s = s
repete n s = s ++ repete (n-1) s
```

II-10) Defina uma função unico :: [Int] −> [Int] que retorna a lista de números que ocorrem exatamente uma vez em uma lista. Por exemplo, unico [2,4,2,1,4] = [1].

```
unico :: [Int] -> [Int]
unico [] = []
unico (a:x)
        | elem a x = [y | y <- (unico x), y /= a]
        | otherwise = a :(unico x)
```

II-11) Pesquise sobre a função map e resolva o seguinte problema: dados um elemento e uma lista, intercale o elemento na lista, em todas as posições. Exemplo:
Main> intercala 1 [2,3]
[[1,2,3],[2,1,3],[2,3,1]]

```
intercala :: a -> [a] -> [[a]]
intercala x[] = [[x]]
intercala x (l:ls) = [x:l:ls] ++ map(l:)(intercala x ls)
```

II-12) Defina uma função que, dada uma lista de inteiros, retorna o número de elementos de valor superior a 10.

```
superior10 :: [Int]->Int
superior10 [] = 0
superior10(h : t) | (h <= 10) =  superior10 t
                  | otherwise = 1+superior10 t
-- h representa  a cabeca da lista e t a calda;
lista = [10,10,11,11] -- lista de teste
main = do
  print(superior10 lista) -- 2
```

II-13) Defina uma função que, dada uma lista de inteiros, retorna outra lista que contém apenas de elementos de valor superior a 10.

```
super10 :: [Int] -> [Int]
super10 [] = []
super10 (a:x)
        | (a > 10) = a:super10 x
        | otherwise = super10 x
```

II-14) Escreva uma função *multiplica_listas* que recebe duas listas de inteiros e produz uma lista de listas em que cada lista corresponde à multiplicação de um elemento da primeira lista por todos os elementos da segunda.

```
multiplicar :: Int -> [Int] -> [Int]
multiplicar a lista = [a*l | l <- lista]

multiplica_listas :: [Int] -> [Int] -> [[Int]]
multiplica_listas [] [] = [[]]
multiplica_listas listaA listaB = [multiplicar a listaB | a <- listaA]
```