UFSC / CTC / INE
Disciplina: Paradigmas de Programação

CCO: INE5416 / SIN:INE5636

Prof. Dr. João Dovicchi*

Roteiro 4 - Cálculo-λ

0.1 Bases do Cálculo-λ

O cálculo-λ é um formalismo algébrico para representar abstração de funções e argumentos na lógica matemática descrito por Alonso Church com base na lógica combinatória de Schönfinkel e na teoria da recursividade de Stephen Kleene.

Parte 1:

1. Encontre os artigos originais de Church que descrevem as bases do cálculo-λ: **1.** Church, Alonzo "An unsolvable problem of elementary number theory", *American Journal of Mathematics*, 58 (1936) pp 345-363 e, também: **2.** Church, Alonzo "The Calculi of Lambda-Conversion", 1941.

2. Procure por referências sobre o cálculo-λ, principalmente para definições e exemplos.

3. Compare com a notação matemática tradicional e tente entender o que muda. Qual a diferença entre variável e argumento?

4. Quais são os tipos de redução do cálculo-λ, encontre exemplos e tente compreender como funcionam.

*http://www.inf.ufsc.br/~dovicchi --- joao.dovicchi@ufsc.br

0.2 Linguagens e λ-calculus

O cálculo-λ é a base das linguagens funcionais.

Em Python, por exemplo, podemos escrever:

```
def square(x):
   return lambda x:x*x
```

A linguagem HASKELL é baseada no cálculo λ. Entretanto, uma vez que o construtor "." (ponto) tem o significado de compositor de funções (veremos isto mais tarde quando estudarmos a sintaxe da linguagem Haskell), a sintaxe do cálculo-λ tem algumas particularidades.

- A letra grega λ é substituída pela contrabarra "\";
- O ponto "." é substituído por `->`.

Por exemplo, $\lambda x.x^2$ é notado, em HASKELL como: `\x->x^2` ou, ainda, `\x->x*x`.

0.3 O interpretador

O compilador Haskell (GHC) pode ser utilizado de forma interativa (GHCi) como um interpretador da linguagem. Para carregar o interpretador, digite, na linha de comando:

```
$ ghci
```

Feito isto, o aluno se encontrará em um novo *shell*, ou seja, na linha de comando do interpretador:

```
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.6.1, for Haskell 98.
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
Prelude>
```

Neste ambiente, pode-se digitar comandos, desde que iniciados por ":". Por exemplo, para obter o *help* usa-se o ponto de interrogação como comando:

```
Prelude> :?
```

Use ":q" ou ":quit" para sair do interpretador.

0.4 O expressões lambda em Haskell

Usando o GHCi, experimente os seguintes comandos:

```
Prelude> let quad = \x->x*x
Prelude>
```

Experimente usar a função `quad` com uma aplicação numérica.
Defina agora:

```
Prelude> let expr = \x->x^2+2*x+3
Prelude> let raiz = \x->(sqrt x)
```

Teste as funções com argumentos (aplicações) numéricas.
No interpretador GHCi[1], observe as seguintes declarações:

```
Prelude> map (f x = x*x) [1..10]
<interactive>:1:9: parse error on input `='
Prelude>
```

A função `map` serve para mapear uma função em uma lista de argumentos. Aparentemente, não há nada de errado, mas o interpretador retorna um erro de *parsing*. No entanto, se declararmos a função como uma expressão λ,

```
Prelude> map (\x->x*x) [1..10]
[1,4,9,16,25,36,49,64,81,100]
Prelude>
```

o interpretador retorna o resultado esperado. Observe que, se a função for declarada antes, o resultado também é correto.

```
Prelude> let f x = x*x
Prelude> map f [1..10]
[1,4,9,16,25,36,49,64,81,100]
Prelude>
```

Uma expressão λ pode receber uma função como argumento, por exemplo:

```
Prelude> let fac n = if n==1 then 1 else (n*fac(n-1))
Prelude> map (\x->(fac x)) [1..10]
[1,2,6,24,120,720,5040,40320,362880,3628800]
Prelude>
```

[1]Nem todos os exemplos são aceitos no HUGS, pois o ele é um interpretador de programas em HASKELL e não um interpretador de comandos como o GHCi.

Parte 2:

A função `deleteBy` remove o primeiro elemento da lista que casa com uma declaração, por exemplo:

```
Prelude> :module List
Prelude List> deleteBy (\x y -> y*x == 48) 6 [6,8,10,12]
[6,10,12]
```

1. Usando uma expressão λ remova da lista `[5..10]` primeiro elemento que casa com a declaração "divisível por 3".
2. Usando uma expressão λ remova da lista `[4..19]` todos os elementos não divisíveis por 4.
3. Qual o valor da expressão:

   ```
   [ x | x <- [1..4], y <- [x..5], (x+y) `mod` 2 == 0 ]
   ```