

Funções, High-Order Functions, Closures

Linguagens de Programação

8 de abril de 2019

- 1 Introdução
- 2 Funções como argumentos
 - Polimorfismo Paramétrico
 - Funções Anônimas
 - Maps e Filters
 - Retornando Funções
- 3 Escopo Léxico
- 4 Ambiente e Closure
 - Idioms

First-class functions, functions Clousures, Higher-order function

First-class functions

Funções podem ser computadas, passadas como parâmetros, armazenadas, etc, sempre que outros valores podem ser computadas, passadas como parâmetros, armazenadas, etc

Higher-order functions

- Funções que recebem (como argumento) ou retornam outras funções.
- Forma de reutilizar comportamento (Ex., algoritmo de ordenação + função para comparar 2 elementos)

Function clousures

Funções que usam usam variáveis definidas “fora” delas, i.e, no ambiente.

Funções como argumentos

O que a função abaixo faz?

```
fun increment x = x + 1;
```

Quais os resultados das seguintes expressões?

```
(increment 7)
```

```
increment (increment 7)
```

```
increment(increment (increment 7))
```

```
increment (increment(increment (increment 7)))
```

Funções como argumentos

O que a função abaixo faz?

```
fun double x = x + x;
```

Quais os resultados das seguintes expressões?

(double 7)

double (double 7)

double(double (double 7))

double (double(double (double 7)))

Funções como argumentos

Quais os resultados das seguintes expressões?

```
(tl [4,8,12,16])
```

```
tl (tl [4,8,12,16])
```

```
tl (tl (tl [4,8,12,16]))
```

```
tl (tl(tl (tl [4,8,12,16])))
```

O que as seguintes funções fazem?

```
fun increment x = x + 1;  
fun double x = x + x;  
  
fun ntimes(f,n,x) =  
  if n=0  
  then x  
  else f (ntimes(f,n-1,x))
```

Quais os valores das seguintes expressões?

```
val x1 = ntimes(increment,4,7);  
val x2 = ntimes(double,4,7);  
val x3 = ntimes(tl,2,[4,8,12,16]);
```

Parametric Polimorphism

Qual o resultado de?

- `length []`
- `length [10]`
- `length [10.0, 20.1, 30.2]`
- `length [[1], [1,2], [1,2,3], [5]]`

Qual o tipo da função **length**?

'a list → int

O que significa este tipo?

Isto é polimorfismo paramétrico!

Funções Anônimas

```
fun ntimes(f,n,x) =  
  if n=0  
  then x  
  else f (ntimes(f,n-1,x))  
  
(* reduzindo o escopo de triple *)  
fun triple_ntimes (n,x) =  
  let  
    fun triple x = 3*x;  
  in  
    ntimes(triple,n,x)  
  end  
  
(* reduzindo ainda mais o escopo *)  
fun triple_ntimes2(n,x) =  
  ntimes((let fun triple y = 3*y in triple end),n,x);  
  
(* por que dar nome a funcao triple? *)  
fun triple_ntimes3 (n,x) =  
  ntimes((fn y => 3*y),n,x);
```

Maps e Filters

```
fun map (f, xs) =  
  case xs of  
    [ ] => [ ]  
  | x::xs' => (f x)::(map (f,xs'));  
  
fun filter (f, xs) =  
  case xs of  
    [ ] => [ ]  
  | x::xs' => if f x  
               then x::(filter (f,xs'))  
               else filter (f,xs')  
  
val x = map(hd,[ [1,2], [3,4], [5,6] ]);  
  
fun get_all_even_snd xs =  
  filter ( (fn (_,v) => v mod 2 = 0), xs);  
  
val l1 = [ (1,2), (2,3), (3,4), (4,5) ];  
val lev = get_all_even_snd l1;
```

Retornando funções

Funções também podem retornar funções

```
fun doubleOrTriple f =  
  if f 7  
  then fn x => 2*x  
  else fn x => 3*x
```

Escopo Léxico

Muita Atenção!!!

O corpo de uma função é **avaliado** no ambiente (environment) onde a função foi **definida**, **não** no ambiente em que a função foi **invocada** (escopo dinâmico).

```
val x = 1
fun f y = x + y
val x = 2
val y = 3
val z = f(x+y)
```


Ambiente e Closures

Funções

- São valores (“cidadãos de pleno direito”)
- São compostas de 2 partes: código e ambiente em que foi criada
- Variáveis não associadas no código (variáveis livres – free variables), são associadas variáveis do ambiente

Iteradores e Clausuras

```
fun filter (f,xs) =  
  case xs of  
    [ ] => [ ]  
  | x::xs' => if f x  
               then x::(filter(f,xs'))  
               else filter(f,xs')  
  
fun allGreaterThanSeven xs = filter(fn x => x > 7, xs);  
  
fun allGreaterThan(xs,n) = filter (fn x => x > n, xs);
```



Fold

```
fun fold (f,acc,xs) =  
  case xs of  
    [ ] => acc  
  | x::xs' => fold(f,f(acc,x),xs') ;  
  
val l1 = [1,2,3,4,5,6] ;  
val soma1 = fold((fn (x,y) => x + y),0,l1) ;
```

Composição de funções

```
fun compose (f,g) = fn x => f (g x);

fun sqrt_of_abs i = Math.sqrt(Real.fromInt(abs i));
fun sqrt_of_abs2 i = (Math.sqrt ◦ Real.fromInt ◦ abs) i;
val sqrt_of_abs3 = Math.sqrt ◦ Real.fromInt ◦ abs;

infix |>;
fun x |> f = f x;
fun sqrt_of_abs4 i = i |> abs |> Real.fromInt |> Math.sqrt;
```


Currying e Aplicação Parcial

```
fun sorted3 (x,y,z) = z>= y andalso y>=x;

val sorted3' = fn x => fn y => fn z => z>= y andalso y>=x;

val r1 = sorted3 (1,2,3);
val r2 = sorted3 (3,2,1);
val r3 = sorted3 (1,3,2);

val r1' = ((sorted3' 1) 2) 3;
val r2' = sorted3' 3 2 1;

val fnneg = sorted3' 0 0;

val rn1 = fnneg 1;
val rn2 = fnneg ~1;
```