Lecture 08
Tipificação e compilação de abstrações

Passagem de parâmetros

- Recorde a semântica da chamada de função adotada nas linguagens CALCF e RECF:

Se E é da forma **call(E_1, E_2)** :
if closure(*id***,** *envc*, **B)** = **eval**(E_1, *env*)
then [*envloc* = *envc*.BeginScope();
envloc.Assoc(*id*, **eval**(E_2, *env*));
eval(E,*env*) ≜ **eval**(**B**, *envloc*) **]**
else eval(E,*env*) ≜ error

- A expressão E2 que denota o argumento da função é avaliada antes da função denotada pela expressão E1 ser efetivamente aplicada.

- Qual o valor intuitivo/efetivo do programa seguinte?

```
declrec f = fun x → f(x) end in
     decl g = fun y → 1 end in g(f(1)) end end
```

Passagem de parâmetros por valor

- Qual o valor da expressão seguinte ?

```
declrec f = fun x → f(x) end in
   decl g = fun y → 1 end in g(f(1)) end end
```

- Valor de acordo com a semântica "declarativa":
  - `g` é a função constante que devolve sempre o valor `1` independentemente do valor do argumento.
  - Então, o valor `g(f(1))` deverá ser `1`.
- Valor determinado pela semântica "operacional":
  - A semântica apresentada não define valor para esta expressão, pois a avaliação de `f(1)` não termina!
  - A regra de passagem de argumentos utilizada é a "passagem por valor" (*call-by-value*)

Passagem de parâmetros por “nome”

- Qual o valor da expressão (1 ou nenhum)?

```
declrec f = fun x → f(x) end in
   decl g = fun y → 1 end in g(f(1)) end end
```

- Existe um modo de avaliação que permite sempre encontrar o valor “declarativo” das expressões.

- Consiste em suspender a avaliação dos argumentos das funções até ao momento em que estes se tornam necessários.

- Corresponde em passar a expressão como argumento, em vez do seu resultado.

- A esta regra de avaliação de argumentos chama-se “passagem por nome” (*call-by-name*) (CBN)

- A estratégia de avaliação por omissão do Algol 60.

Passagem de parâmetros por "necessidade"

- Qual o valor da expressão?

```
decl x = var(0) in
  declrec f = fun x → x := !x+1 end in
    decl g = fun y → y+y+!x end in g(f(x)) end end
```

- Se se suspender a avaliação dos argumentos das funções até ao momento em que estes se tornam necessários passando a expressão como argumento, em vez do seu resultado, a utilização repetida dos parâmetros pode levar a resultados não esperados.
- Para avaliar só uma vez cada suspensão é preciso "guardar" o seu resultado após a primeira avaliação.
- A esta regra de avaliação de argumentos chama-se "passagem por necessidade" (*call-by-need*). Utilizada em linguagens com avaliação Lazy (e.g. Haskell)

Semântica de RECF (Lazy)

- Algoritmo eval para calcular a denotação de qualquer expressão E de RECF num ambiente env usando a passagem de parâmetros Lazy:

$$\text{eval} : \text{RECF} \times \text{ENV} \rightarrow \text{RESULT}$$

- É necessário representar a suspensão por um valor mutável:

Se E é da forma **id**(id): val = *env*.Find(id)
if val = **susp** (B, *envsusp*)
then [if val.isUnevaluated()
then val.set(**eval**(B, *envsusp*))
eval(E, *env*) ≜ val.getValue() **]**
else eval(E, *env*) ≜ val