Data: 04/06/09

Programação Funcional e a linguagem Haskell

João Paulo Pizani Flor

joaopizani@inf.ufsc.br

PET COMPUTAÇÃO UFSC

Tópicos

- Definição relâmpago
- Programação Imperativa vs. Funcional
- Conceitos fundamentais
- Conseqüências (boas e “ruins”)
- Por quê Haskell?
- Tour da linguagem e ferramentas
- Exemplos de uso
- Por onde começar a aprender

Definição Relâmpago

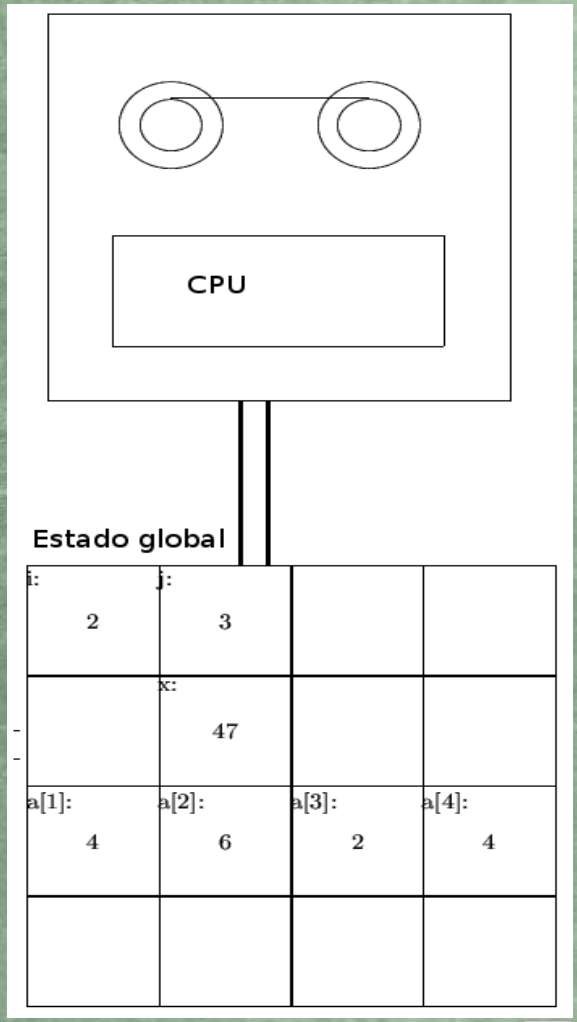
"Um paradigma de programação que trata a computação como a avaliação de funções e *evita a* noção de estado e de dados variáveis."

Imperativa x Funcional

PET COMPUTAÇÃO UFSC

Recapitulando Prog. Imperativa

- Na programação imperativa:
  - Variável → Célula de memória
  - *Statement* → Transição de estado
  - Estruturas de controle → Instruções “test and jump”

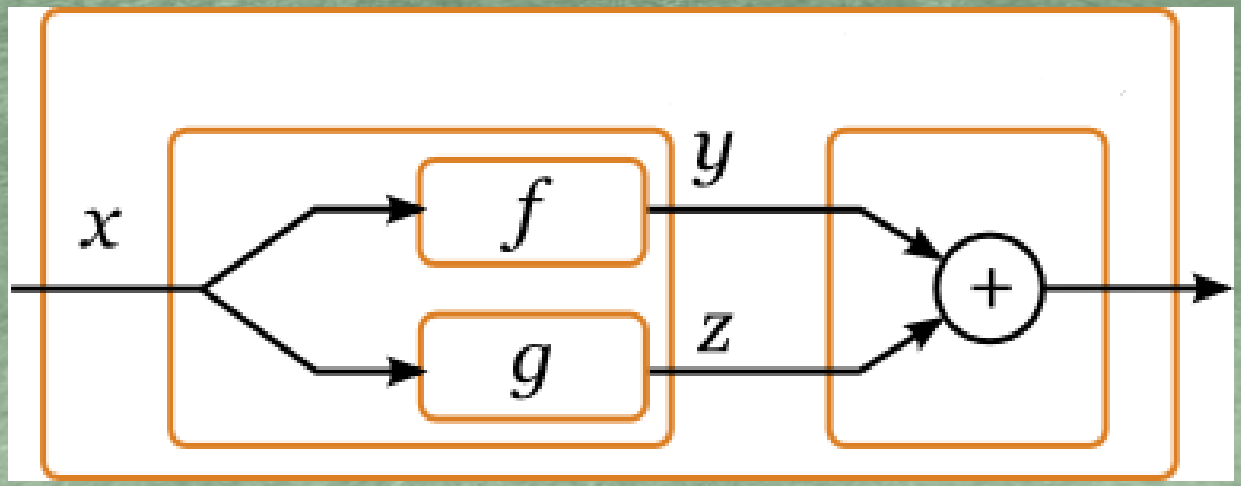


Ou seja...

- Semântica da linguagem ligada à idéia de máquina de estados.
- Existe um escopo global
- Funções, além de produzirem um valor, podem ter efeitos colaterais

Já em Programação Funcional...

- Não existe a noção de estado
  - Não existe comando de atribuição
  - Não existem variáveis!
- Cada função é definida como a composição e aplicação de outras funções (ou da própria)



O que é Programação Funcional?

“Funcional” vem de funções???

- Frase comum: “Prog. Funcional é programar usando funções matemáticas...”
  - Mas C/C++/Java/Python/etc também têm funções!

- Grande diferença: Em um programa funcional, todas as funções têm uma propriedade:

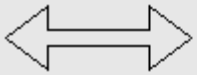
TRANSPARÊNCIA REFERENCIAL

Transparência Referencial

- Duas condições bem simples:
  - O **único** efeito de chamar uma função é o seu valor de retorno
  - Os **únicos** fatores que podem influenciar o valor de uma função são seus **parâmetros.**

- Def. alternativa: “Like can be replaced by like”

(3 * F(a,b) + b) * (3 * F(a,b) - c)



```
let t be 3 * F(a,b)
in (t + b) *  (t - c)
end
```

X

```
F(a,b: integer): integer is
do
  c := c + 1;
  result := 2 * (a + b)
end
```

Quem garante?

- Em linguagens convencionais → Programador
- Em linguagens funcionais →Compilador

- É dessa garantia/obrigação que vêm todas as vantagens de performance
  - E também as “bizarrices” no jeito de programar

Conseqüências

Vantagens

- Compilador pode aplicar várias otimizações
  - Common Subexpression Elimination
  - Avaliação preguiçosa

```
tam = length([3,2,7,0/0]) //não ocorre erro e tam vale 4
```

  - Paralelização automática

```
res1 = operacaoBemLonga1(s1) // não tem efeitos colaterais!
res2 = operacaoBemLonga2(s2) // também é transparente
print(res1+res2)
```

Para o programador

- Programas são mais legíveis
  - Código fica mais conciso (exemplo: quicksort)
  - Não é preciso executar mentalmente para entender
- Gerenciamento automático de memória
  - “No more SIGSEGV's”!
- Facilidade em *encapsular* abstrações
  - Map, reduce (fold), compose e cia.

“Quando C é melhor?”

- Há *áreas* em que a Prog. Funcional não é *tão* adeqüada *ainda*:
  - Programação de sistemas
  - Drivers, SO
  - Aplicações onde performance é necessária a *todo custo*
- Isso pelo maior controle sobre a maneira como a computação é realizada

Haskell

PET COMPUTAÇÃO UFSC

Por quê Haskell?

- Puramente funcional
- Fortemente e estaticamente tipada
- Avaliação preguiçosa *(lazy evaluation)*
- Classes de tipos
- MUITAS bibliotecas úteis
- Comunidade bastante ativa

Tipagem

- Cada expressão tem um tipo definido
- Tipos checados em tempo de compilação
  - Elimina grande parte dos erros de programação
- Tipos das expressões são *inferidos*
  - *Programador não precisa declarar tipo de nada!*
  - *O melhor de dois mundos*

Avaliação preguiçosa

- “Uma expressão só será avaliada quando seu valor for necessário.”
- Exemplos:

```
isInfixOf :: Eq a => [a] -> [a] -> Bool
isInfixOf x y = any (isPrefixOf x) (tails y)
// incrivelmente ineficiente em uma ling. imperativa

g ← getStdGen
take 3 (randomRs (0,100) g)
// lista infinita de randoms
```

Classes de tipos

- Permitem que funções tomem parâmetros de tipos *genéricos*
  - Similar ao generics de Java / templates de C++
  - Há diferenças...
- Têm semelhanças com *interfaces* Java
- Exemplo:

```
class Eq a where
     (==) :: a -> a -> Bool
     (/=) :: a -> a -> Bool
     x == y = not (x /= y)
     x /= y = not (x == y)
```

```
head :: (Eq a) => [a] → a
head (x:xs) = x
```

Tour das ferramentas

Compilador

- GHC – Glasgow Haskell Compiler
  - http://www.haskell.org/ghc
  - Mais popular e suportado de todos
- GHCi – GHC interativo (interpretador)

Cabal

- Sistema para a automatização de compilação e empacotamento de software Haskell
  - Similar ao GNU Autoconf/Automake
  - http://www.haskell.org/cabal/

HackageDB

- Repositório centralizado de bibliotecas Haskell
  - Similar aos repositórios de distros Linux
  - Pacotes no formato cabal
  - http://hackage.haskell.org/
  - Ferramenta "`cabal install`" funciona como se fosse o apt-get do Haskell
- Hoogle
  - Pesquisa de API (nome e tipos)
  - http://www.haskell.org/hoogle/

Exemplos de uso

Haskell na indústria

- Alguns exemplos:
  - AT&T
  - Barclays
  - Credit Suisse
  - Galois
  - Qualcomm
- Para empregos consulte :)
  - http://haskell.org/haskellwiki/Jobs

Software em Haskell

- Happstack
  - Servidor de aplicações web
  - http://happstack.com/
- Darcs
  - Controle de versão distribuído
  - http://darcs.net/
- Xmonad
  - Gerenciador de janelas para X11
  - http://xmonad.org/

PET COMPUTAÇÃO UFSC

Software em Haskell

- Frag
  - First-person shooter based on Quake 3 Arena
  - http://haskell.org/haskellwiki/Frag
- Lava Hardware Description Language
  - De Haskell p/ FPGA's Xilinx
  - http://raintown.org/lava/

Software mais humilde :)

- Trabalho Final INE5416
  - Separar um bitmap em suas 3 camadas RGB
- Ninety-nine Haskell problems
  - Envolvem estruturas de dados, grafos, matemática, lógica, etc.
  - http://www.haskell.org/haskellwiki/H-99:_Ninety-Nine_Haskell_Problems

Por onde começar

Baixe a plataforma

- Haskell Platform - “batteries included”
  - http://hackage.haskell.org/platform
  - Para Windows, Unix/Linux e MacOS

Tutoriais

- Básico
  - Learn you a Haskell for Great Good!
  - http://learnyouahaskell.com/
- Nem tão básico
  - Haskell wikibook
  - http://en.wikibooks.org/wiki/Haskell
- Avançado
  - Write yourself a Scheme in 48 hours
  - http://en.wikibooks.org/wiki/Write_Yourself_a_Scheme_in_48_Hours

Livros

- Real World Haskell
  - Versão online gratuita (www.realworldhaskell.org)
  - O'Reilly – US$49,90

- The Haskell School of Expression
  - Aprendendo programação funcional através desenvolvendo multimídia
  - http://www.haskell.org/soe/

Dúvidas?

Site: http://pet.inf.ufsc.br

Email: pet@inf.ufsc.br