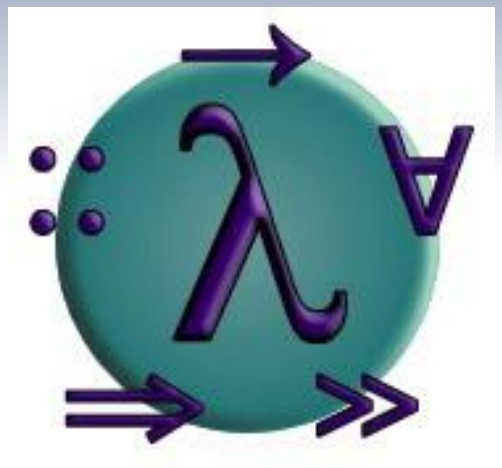

Programação Funcional

Capítulo 2

Primeiros Passos

José Romildo Malaquias

2012.1

Departamento de Computação

Universidade Federal de Ouro Preto

Tópicos

Glasglow Haskell Compiler

- GHC é um compilador e um ambiente interativo livre para a linguagem funcional Haskell.
- GHC suporta toda a linguagem Haskell 2010 mais uma grande variedade de extensões.
- GHC tem suporte particularmente bom para a concorrência e paralelismo.
- GHC gera código rápido, principalmente para programas concorrentes. Dê uma olhada no desempenho GHC em The Computer Language Benchmarks Game.
- GHC funciona em várias plataformas, incluindo Windows, Mac, Linux, a maioria das variedades de Unix, e várias arquiteturas de processadores diferentes.
- GHC tem capacidades de otimização, incluindo otimização entre módulos.

Glasglow Haskell Compiler (cont.)

- GHC compila código Haskell
  - diretamente para código nativo ou
  - pode usar LLVM como um **back-end**, ou
  - pode gerar código C como um código intermediário para portar para novas plataformas.
- O ambiente interativo **GHCi** compila para **bytecode**, e suporta a execução mista de **bytecode** e programas compilados.
- GHC suporta **Profiling**.
- GHC vem com várias bibliotecas, e muitas outras estão disponíveis no Hackage.

Iniciando o ambiente interativo ghci

- **GHCi** é um ambiente interativo do GHC em que expressões Haskell podem ser avaliadas de forma interativa e os programas podem ser interpretados.
- GHCi também tem suporte para carregar código compilado de forma interativa, bem como apoio a todos as extensões de linguagem que GHC oferece.
- GHCi também inclui um depurador interativo.
- O GHCi pode ser iniciado a partir do prompt simplesmente digitando `ghci`. Em um sistema Unix:

```
$ ghci
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

Iniciando o ambiente interativo ghci (cont.)

- O prompt > significa que o sistema ghci está pronto para avaliar expressões.
- À esquerda do prompt é mostrada a lista de módulos abertos.
- Expressões Haskell podem ser digitadas no prompt.
- Por exemplo:

```
Prelude> 2 + 3 * 4
14
Prelude> (2 + 3) * 4
20
Prelude> sqrt (3^2 + 4^2)
5.0
```

Tópicos

A biblioteca padrão (standard prelude)

- O arquivo de biblioteca `Prelude.hs` oferece um grande número de funções definidas no **padrão da linguagem** através do módulo `Prelude`.
- O módulo `Prelude` é importado automaticamente em todos os módulos de uma aplicação Haskell.

A biblioteca padrão (standard prelude) (cont.)

- O módulo `Prelude` oferece várias funções para manipulação de números.
- Normalmente uma **aplicação de função** usa a notação prefixa: escreve-se a função seguida dos argumentos.
- No entanto algumas funções são definidas como operadores binários infixos.

A biblioteca padrão (standard prelude) (cont.)

- O módulo `Prelude` oferece as funções aritméticas familiares, como `+`, `-`, `*`, `div`, `mod`, `/`.
- Exemplos:

```
Prelude> sqrt 2.56
1.6
Prelude> 7 * 8
56
Prelude> 1 + 2 * 3
7
Prelude> (1 + 2) * 3
9
Prelude> div (7*8) 3
18
Prelude> mod (7*8) 3
2
Prelude> (7*8) / 3
18.666666666666668
```

A biblioteca padrão (standard prelude) (cont.)

- Além das funções numéricas familiares, o módulo `Prelude` também oferece muitas funções úteis para a manipulação de **listas** e outras estruturas de dados.

A biblioteca padrão (standard prelude) (cont.)

- `null`: verifica se uma lista é vazia:

```
Prelude> null []
True

Prelude> null [1,2,3,4,5]
False
```

- `head`: seleciona o primeiro elemento de uma lista:

```
Prelude> head [1,2,3,4,5]
1
```

- `tail`: remove o primeiro elemento de uma lista:

```
Prelude> tail [1,2,3,4,5]
[2,3,4,5]
```

A biblioteca padrão (standard prelude) (cont.)

- `lenght`: calcula o tamanho de uma lista:

```
Prelude> length [1,2,3,4,5]
5
```

- `(!!)`: seleciona o *n*-ésimo elemento de uma lista:

```
Prelude> [1,2,3,4,5] !! 2
3
```

A biblioteca padrão (standard prelude) (cont.)

- `take`: seleciona os primeiros *n* elementos de uma lista:

```
Prelude> take 3 [1,2,3,4,5]
[1,2,3]
```

- `drop`: remove os primeiros *n* elementos de uma lista:

```
Prelude> drop 3 [1,2,3,4,5]
[4,5]
```

A biblioteca padrão (standard prelude) (cont.)

- `sum`: calcula a soma dos elementos de uma lista de números:

```
Prelude> sum [1,2,3,4,5]
15
```

- `product`: calcula o produto dos elementos de uma lista de números:

```
Prelude> product [1,2,3,4,5]
120
```

A biblioteca padrão (standard prelude) (cont.)

- `(++)`: concatena duas listas:

```
Prelude> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- `reverse`: inverte uma lista:

```
Prelude> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

Tópicos

Applicação de função

- Em Matemática, **aplicação de função** é denotada usando parênteses, e a multiplicação é muitas vezes denotada usando justaposição ou espaço.
- Exemplo:

$$f(a, b) + cd$$

aplica a função f aos argumentos a e b, e adiciona o resultado ao produto de c e d.

Applicação de função (cont.)

- Em Haskell, **aplicação de função** é denotada usando o espaço, e multiplicação é indicado pelo operador `*`.
- Exemplo:

```
f a b + c * d
```

aplica a função `f` aos argumentos `a` e `b`, e adiciona o resultado ao produto de `c` e `d`.

Applicação de função (cont.)

- Além disso, aplicação de função tem precedência maior do que todos os outros operadores.
- Assim

```
f a + b
```

significa

```
(f a) + b
```

em vez de

```
 f (a + b)
```

Applicação de função (cont.)

- Exemplos:

Matemática	Haskell
$f(x)$	`f x`
$f(x, y)$	`f x y`
$f(g(x))$	`f (g x)`
$f(x, g(y))$	`f x (g y)`
$f(x)g(y)$	`f x * g y`

Tópicos

Scripts Haskell

- Além das funções no prelúdio padrão, o programador também pode definir suas próprias funções.
- Novas funções são definidas dentro de um **script** , um arquivo texto compreendendo uma seqüência de definições.
- Por convenção, **scripts** Haskell normalmente têm a extensão `.hs` em seu nome. Isso não é obrigatório, mas é útil para fins de identificação.

Meu primeiro script

- Ao desenvolver um **script** Haskell, é útil manter duas janelas abertas, uma executando um editor para o script, e outra para o GHCi em execução.
- Inicie um editor de texto, digite as seguintes definições de função, e salve o **script** como `test.hs`:

```
double x = x + x

quadruple x = double (double x)
```

Meu primeiro script (cont.)

- Deixando o editor aberto, em outra janela execute o GHCi com o novo **script**:

```
$ ghci test.hs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( test.hs, interpreted )
Ok, modules loaded: Main.
*Main>
```

Meu primeiro script (cont.)

- Agora, tanto `Prelude.hs` como `test.hs` são carregados, e funções de ambos os **scripts** podem ser usadas:

```
*Main> quadruple 10
40
*Main> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Meu primeiro script (cont.)

- Deixando GHCi aberto, volte para o editor, adicione as seguintes definições, e salve:

```
factorial n = product [1..n]

average ns  = sum ns `div` length ns
```

- Observe que:
  - `div` é colocado entre crases para trás, e não para a frente;
  - `x `f` y` é apenas abreviação sintática para `f x y`.

Meu primeiro script (cont.)

- GHCi não detecta automaticamente que o **script** foi alterado. Assim um comando **reload** deve ser executado antes de as novas definições poderem ser usadas:

```
*Main> :reload
[1 of 1] Compiling Main             ( test.hs, interpreted )
Ok, modules loaded: Main.
```

```
*Main> factorial 10
3628800
*Main> factorial 50
30414093201713378043612608166064768844377641568960512000000000000
*Main> average [1,2,3,4,5]
3
```

Identificadores

- Nomes de **função e variáveis** devem começar com uma letra minúscula.

  Exemplos: `myFun`, `fun1`, `arg_2`, `x'`

- Por convenção, uma lista de elementos normalmente têm um sufixo `s` em seu nome, que indica plural.

  Exemplos: `xs`, `ns`, `nss`

Comentários

- **Comentário de linha**: introduzido por `--` e se extende até o final da linha.
- **Comentário de bloco**: delimitado por `{-` e `-}`. Pode ser aninhado.

Tópicos

A regra de layout

Em uma seqüência de definições, cada definição deve começar precisamente na mesma coluna:

```
a = 10
b = 20
c = 30
```

```
a = 10
  b = 20
c = 30
```

```
  a = 10
b = 20
    c = 30
```

A regra de layout (cont.)

A regra de layout evita a necessidade de uma sintaxe explícita para indicar o agrupamento de definições.

```
{- agrupamento implícito -}
a = b + c
    where
      b = 1
      c = 2
d = a * 2
```

significa

```
{- agrupamento explícito -}
a = b + c
     where
       {b = 1;
        c = 2}
d = a * 2
```

A regra de layout (cont.)

- Para evitar problemas com a regra de **layout**, é recomendado não utilizar caracteres de tabulação para indentação do código fonte, uma vez que um único caracterizar de tabulação pode ser apresentado na tela como vários espaços. O texto do programa vai aparentar estar alinhado na tela do computador, mas na verdade pode não estar devido ao uso do tabulador.
- No notepad++ você deve desabilatar o uso de tabulação. Para tanto marque a opção para substituir tabulações por espaço, acessando o menu *Configurações -> Preferências -> Menu de Linguagens/Configuração de Abas -> Substituir por espaço*.

Tópicos

Comandos úteis do GHCi

comando	abrev	**significado**
`:load` *name*	`:l`	carrega o script `name`
`:reload`	`:r`	recarrega o script atual
`:edit` *name*	`:e`	edita o script `name`
`:edit`	`:e`	edita o script atual
`:type` *expr*	`:t`	mostra o tipo de *expr*
`:info` *name*	`:i`	dá informações sobre *name*
`:browse` *Name*		dá informações sobre o módulo *Name*, se ele estiver carregado
`let` *id* `=` *exp*		associa a variável *id* com o valor da expressão *exp*
`:!` *comando*		executa *comando* do sistema
`:help`	`:h`, `:?`	lista completa dos comandos do GHCi
`:quit`	`:q`	termina o GHCi

Exercícios

1. Experimente os exemplos apresentados nos slides usando o GHCi.
2. Corrija os erros de sintaxe no programa abaixo, e teste sua solução usando o GHCi.

```
N = a 'div' length xs
    where
       a = 10
      xs = [1,2,3,4,5]
```

3. Mostre como a função de biblioteca `last` que seleciona o último elemento de uma lista pode ser definida de acordo com as funções introduzidas nesta aula.
4. Você pode pensar em outra possível definição?
5. Da mesma forma, mostrar como a função de biblioteca `init`, que remove o último elemento de uma lista pode ser definida de duas maneiras diferentes.

Fim