Introdução ao Paradigma Funcional

Formação: Upskill – ServiceNow (2025/26)
Módulo: Introdução ao Paradigma Funcional
Carga horária total: 35 horas
Docente: Tiago Côrte

Um **programa** pode ser visto como algo que transforma informação

Existem 2 grandes classes de linguagens de programação:

Imperativas - um programa é uma sequências de instruções (ou seja de "ordens"). (ex: Pascal, C, Java, ...)

-- díficil establecer uma relação precisa entre o input e o output e de raciocinar sobre os programas; ...
\+ normalmente mais eficientes; ...

Declarativas - um programa é um conjunto de declarações que descrevem a relação entre o input e o output. (ex: Prolog, ML, Haskell, ...)

\+ facíl de establecer uma relação precisa entre o input e o output e de raciocinar sobre os programas; ...
-- normalmente menos eficientes (mas cada vez mais); ...

Exemplo: A função factorial é descrita matematícamente por

$$0! = 1$$
$$n! = n * (n-1)! \text{ , se } n>0$$

Dois programas que fazem o cálculo do factorial de um número, implementados em:

C

```
int factorial(int n)
{ int i, r;

  i=1;
  r=1;
  while (i<=n) {
    r=r*i;
    i=i+1;
  }
  return r;
}
```

Haskell

```
fact 0 = 1
fact n = n * fact (n-1)
```

Qual é mais facil de entender ?

Na programação (funcional) faremos uma distinção clara entre três grandes grupos de conceitos:

Dados – Que tipo de informação é recebida e como ela se pode organizar por forma a ser processada de forma eficiente.

Operações – Os mecanísmos para manipular os dados. As operações básicas e como contruir novas operações a partir de outras já existentes.

Cálculo – A forma como o processo de cálculo decorre.

A linguagem **Haskell** fornece uma forma rigorosa e precisa de descrever tudo isto.

Programa Resumido

Nesta disciplina estuda-se o paradigma funcional de programação, tendo por base a linguagem de programação ***Haskell***.

- Programação funcional em Haskell.
  - *Conceitos fundamentais:* expressões, tipos, redução, funções e recursividade.
  - *Conceitos avançados:* funções de ordem superior, polimorfismo, tipos indutivos, classes, modularidade e monades.
- Estruturas de dados e algoritmos.
- Tipos abstractos de dados.

Bibliografia

- Fundamentos da Computação, Livro II: Programação Funcional.
  José Manuel Valença e José Bernardo Barros. Universidade Aberta, 1999.

- Introduction to Functional Programming using Haskell.
  Richard Bird. Prentice-Hall, 1998.

- Haskell: the craft of functional programming.
  Simon Thompson. Addison-Wesley.

- A Gentle Introduction to Haskell.
  Paul Hudak, John Peterson and Joseph Fasel.

O Paradigma Funcional de Programação

Haskell

```
fact 0 = 1
fact n = n * fact (n-1)
```

As equações que são usadas na definição da função `fact` são **equações matemáticas**. Elas indicam que o lado esquerdo e direito têm o mesmo valor.

C

```
int factorial(int n)
{ int i, r;

  i=1;
  r=1;
  while (i<=n) {
    r=r*i;
    i=i+1;
  }
  return r;
}
```

Isto é muito diferente do uso do = nas linguagens imperativas.

Por exemplo, a instrução `i=i+1` representa uma **atribuição** (o valor anterior de `i` é destruído, e o novo valor passa a ser o valor anterior mais 1). Portanto `i` é redefinido.

Porque = em Haskell significa ***"é, por definição, igual a"***, e não é possível redefinir, o que fazemos é raciocinar sobre equações matemáticas.
É, portanto, muito mais facil do que raciocinar sobre programas funcionais do que sobre programas imperativos.

O Paradigma Funcional de Programação

- Um **programa** é um conjunto de definições.
- Uma **definição** associa um ***nome*** a um ***valor***.
- **Programar** é definir estruturas de dados e funções para resolver um dado problema.
- O **interpretador** (da linguagem funcional) actua como uma máquina de calcular:

lê uma expressão, calcula o seu valor e mostra o resultado

Exemplo: Um programa para converter valores de temperaturas em graus *Celcius* para graus *Farenheit*, e de graus *Kelvin* para graus *Celcius*.

```
celFar c = c * 1.8 + 32
kelCel k = k - 273
```

Depois de carregar este programa no interpretador Haskell, podemos fazer os seguintes testes:

```
> celFar 25
77.0
> kelCel 0
-273
>
```

- A um conjunto de associações *nome-valor* dá-se o nome de **ambiente** ou **contexto** (ou *programa*).

- As expressões são avaliadas no âmbito de um contexto e podem conter ocorrências dos nomes definidos nesse contexto.

- O interpretador usa as definições que tem no contexto (programa) como regras de cálculo, para simplificar (calcular) o valor de uma expressão.

Exemplo: Este programa define três funções de conversão de temperaturas.

```
celFar c = c * 1.8 + 32
kelCel k = k - 273
kelFar k = celFar (kelCel k)
```

No interpretador ...

```
> kelFar 300
80.6
```

É calculado pelas regras estabelecidas pelas definições fornecidas pelo programa.

```
kelFar 300 ⇨ celFar (kelCel 300)
           ⇨ (kelCel 300) * 1.8 + 32
           ⇨ (300 - 273) * 1.8 + 32
           ⇨ 27 * 1.8 + 32
           ⇨ 80.6
```

Transparência Referencial

- No paradigma funcional, as expressões:
  - são a representação concreta da informação;
  - podem ser associadas a nomes (definições);
  - denotam valores que são determinados pelo interpretador da linguagem.

- No âmbito de um dado contexto, todos os nomes que ocorrem numa expressão têm um valor único e imotável.

- O valor de uma expressão depende *unicamente* dos valores das sub-expressões que a constituem, e essas podem ser substituidas por outras que possuam o mesmo valor.

A esta caracteristica dá-se o nome de **transparência referencial**.

Linguagens Funcionais

- O nome de *linguagens funcionais* advém do facto de estas terem como operações básicas a definição de funções e a aplicação de funções.

- Nas linguagens funcionais as funções são entidades de 1ª classe, isto é, podem ser usadas como qualquer outro objecto: passadas como parâmetro, devolvidas como resultado, ou mesmo armazenadas em estruturas de dados.

Isto dá às linguagens funcionais uma **grande flexibilidade**, **capacidade de abstração** e **modularização do processamento de dados**.

- As linguagens funcionais fornecem um alto nivel de abstração, o que faz com que os programas funcionais sejam mais **concisos**, mais **fáceis de entender / manter** e mais **rápidos de desenvolver** do que programas imperativos.

- No entanto, em certas situações, os programas funcionias podem ser mais penalizadores em termos de eficiência.

www.haskell.org

Valores & Expressões

Os **valores** são as entidades básicas da linguagem Haskell. São os elementos atómicos.

As **expressões** são obtidas aplicando funções a valores ou a outras expressões.

O interpretador Haskell actua como uma *calculadora* (*"read - evaluate - print loop"*):

***lê** uma expressão, **calcula** o seu valor e **apresenta** o resultado.*

Exemplos:

```
>  5
5

>  3.5  +  6.7
10.2

>  2 < 35
True

>  not True
False

>  not ((3.5+6.7) > 23)
True
```

Tipos

Os tipos servem para **classificar** entidades (de acordo con as suas caracteristicas).

Em Haskell *toda a expressão tem um **tipo** associado.*

`e :: T` significa que a expressão `e` *tem* tipo `T`

é do tipo

Informalmente, podemos associar a noção de *"tipo"* à noção de *"conjunto"*, e a noção de *"ter tipo"* à noção de *"pertença"*.

Exemplos:

```
58       :: Int
'a'      :: Char
[3,5,7]  :: [Int]
(8,'b')  :: (Int,Char)
```

Inteiro
Caracter
Lista de inteiros
Par com um inteiro e um caracter

O Haskell é uma linguagem fortemente tipada, com um sistema de tipos muito evoluído (como veremos). Em Haskell, a verificação de tipos é feita durante a compilação.

Tipos Básicos

Bool	Boleanos:	`True, False`
Char	Caracteres:	`'a', 'b', 'A', '1', '\n', '2', ...`
Int	Inteiros de tamanho limitado:	`1, -3, 234345, ...`
Integer	Inteiros de tamanho ilimitado:	`2, -7, 7575685001343468 2, ...`
Float	Números de vírgula flutuante:	`3.5, -6.53422, 51.2E7, 3e4, ...`
Double	Núm. vírg. flut. de dupla precisão:	`3.5, -6.5342, 51.2E7, ...`
()	*Unit*	`()` é o seu único elemento do tipo *Unit.*