Capítulo 1

Introdução

A linguagem *Haskell*[2] [1, 2, 3, 4, 6, 7] é uma linguagem funcional[3] (pura) com tipos de dados bem definidos [1], ou seja, os modelos para as estruturas de informação são baseados na Teoria dos Conjuntos, e as operações são descritas através de funções entre conjuntos.

Os programas mais divulgados para se trabalhar com a linguagem Haskell são o interpretador *Hugs*[4] [5] e o compilador *GHC – the Glasgow Haskell Compiler*[5] sendo que este último providencia também um interpretador, o *GHCI*, com um modo de utilização idêntica ao *Hugs*.

A diferença mais visível entre um compilador e um interpretador é dada pela interacção com o utilizador. No caso de um compilador é necessário utilizar um editor de texto para escrever um programa que é depois transformado pelo compilador de uma forma automática num executável. No caso do interpretador não só é possível escrever programas e avaliá-los através do próprio interpretador (no caso do *Hugs* e do *GHCI* só é possível calcular o valor de expressões), como no caso de se utilizar um editor externo para a escrita do programa é depois necessário ter o interpretador em memória de forma a executar o programa. Os interpretadores permitem uma testagem dos programas de uma forma interactiva o que no caso de uma linguagem funcional (funções independentes umas das outras) é bastante interessante.

De seguida vai-se centrar a atenção na utilização dos interpretadores pois será esta a forma mais usual de desenvolver programas em Haskell.

1.1 Utilização dos Interpretadores

Ao invocar-se o interpretador este carrega de imediato o módulo Prelude.hs, que constitui o módulo básico da linguagem *Haskell*.

Toda a interacção com o programa *Hugs* desenvolve-se no âmbito de um dado módulo, tal facto é posto em evidência na linha de comando do programa:

```
Prelude>
```

[1] (Versão 1.3)
[2] http://www.haskell.org/
[3] http://www.mat.uc.pt/~pedro/cientificos/Funcional.html
[4] http://www.haskell.org/hugs/
[5] http://www.haskell.org/ghc/

Se se invocar o interpretador conjuntamente com o nome de um ficheiro, já previamente construído, e que contenha um módulo, por exemplo `Fact.hs`, contendo o módulo `Fact`, a linha de comando do interpretador passará a ser:

```
Fact>
```

A interacção com o interpretador limitar-se-á a:

1. Avaliação de expressões. As expressões têm de estar contidas numa só linha. Por exemplo:

   ```
   Prelude> 6*4
   24
   Prelude>
   ```

2. Avaliação de comandos. Entre estes temos: "`:q`", para abandonar o interpretador; "`:?`", para obter uma lista dos comandos disponíveis; e "`:load` <nome_de_ficheiro>" para carregar um novo módulo. Um comando interessante numa fase de aprendizagem é o comando `set +t`. Este comando `set`, com o argumento `+t`, modifica o comportamento do interpretador de molde a que este passe a mostrar a informação acerca do tipo dos resultados dos cálculos da expressões. Por exemplo

   ```
   Prelude> :set +t
   Prelude> 6*3.0
   18.0 :: Double
   Prelude>
   ```

1.2 Escrita de Programas em Haskell

Para a escrita de programas, isto é funções, é necessário usar um editor exterior. As novas definições podem ser incorporadas na definição de um novo módulo *Haskell*, ou então escritas de uma forma solta, caso em que serão automaticamente incorporadas no módulo `Main`. Posteriormente as novas definições podem ser carregado no interpretador de *Haskell* através do comando `:load` <*nome_do_ficheiro*>.

Comentários Em *Haskell* os comentários são introduzidos pela sequência "`--`", à direita da qual todo o restante conteúdo da linha é ignorado.

Podemos ter comentários que se extendem por várias linhas bastando para tal colocar a sequência referida no princípio de cada linha.

Para poder colocar toda uma secção de texto como comentário tem-se ainda a possiblidade de definir um bloco de comentários através das sequências {`-`, `-`}, abrir e fechar bloco respectivamente. Podemos definir vários blocos de comentários aninhados uns nos outros.

Indentação É importante notar que a forma como as várias definições são escritas em *Haskell* não é "inocente", isto é, em *Haskell* a indentação é significativa, sendo que a forma como as definições estão escritas afecta a avaliação das mesmas.

Ao contrário de muitas outras linguagens (*Pascal*, *C*, *Fortran*, ...) em que a indentação, embora aconselhada, não é significativa, sendo somente uma forma de organizar o texto para permitir uma leitura mais fácil, em *Haskell* a indentação é significativa. As regras precisas são facilmente assimiláveis pois que se tratam das regras que são aconselhadas para a escrita de programas nas outras linguagens. Temos então:

1. Se a linha começa mais à frente do que começou a linha anterior, então ela deve ser considerada uma continuação da linha anterior. Assim por exemplo, escrever

   ```
   quadrado x
     = x*x
   ```

   é o mesmo que escrever

   ```
   quadrado x = x*x
   ```

2. Se uma linha começa na mesma coluna que a anterior, então elas são consideradas definições independentes.

3. Se uma linha começa mais atrás do que a anterior, então essa linha não pertence à mesma lista de definições.

Se quisermos separar, de forma explícita, duas definições podemos usar o separador ";". Por exemplo:

```
dobro x = 2*x ; triplo x = 3*x
```

1.2.1 A utilização do *(X)emacs*

O desenvolvimento de programas em *Haskell* é muito facilitado caso se use o editor *(X)emacs*, caso em que se pode usar o *haskell-mode*[6], um modo de edição do *(X)emacs* especializado para o desenvolvimento de programas em *Haskell*. De seguida transcreve-se a informação (`C-h m`) existente para este modo.

```
Haskell mode:
Major mode for editing Haskell programs.  Last adapted for Haskell 1.4.
Blank lines separate paragraphs, comments start with `-- '.

M-; will place a comment at an appropriate place on the current line.
C-c C-c comments (or with prefix arg, uncomments) each line in the region.

Literate scripts are supported via `literate-haskell-mode'.  The
variable `haskell-literate' indicates the style of the script in the
current buffer.  See the documentation on this variable for more
details.

Modules can hook in via `haskell-mode-hook'.  The following modules
are supported with an `autoload' command:
```

[6] http://www.haskell.org/haskell-mode/

```
   ‘haskell-font-lock’, Graeme E Moss and Tommy Thorn
     Fontifies standard Haskell keywords, symbols, functions, etc.

   ‘haskell-decl-scan’, Graeme E Moss
     Scans top-level declarations, and places them in a menu.

   ‘haskell-doc’, Hans-Wolfgang Loidl
     Echoes types of functions or syntax of keywords when the cursor is idle.

   ‘haskell-indent’, Guy Lapalme
     Intelligent semi-automatic indentation.

   ‘haskell-simple-indent’, Graeme E Moss and Heribert Schuetz
     Simple indentation.

   ‘haskell-hugs’, Guy Lapalme
     Interaction with Hugs interpreter.

Module X is activated using the command ‘turn-on-X’.  For example,
‘haskell-font-lock’ is activated using ‘turn-on-haskell-font-lock’.
For more information on a module, see the help for its ‘turn-on-X’
function.  Some modules can be deactivated using ‘turn-off-X’.  (Note
that ‘haskell-doc’ is irregular in using ‘turn-(on/off)-haskell-doc-mode’.)

Use ‘haskell-version’ to find out what version this is.

Invokes ‘haskell-mode-hook’ if not nil.

----

Font-Lock minor mode (indicator Font):
Toggle Font Lock Mode.
With arg, turn font-lock mode on if and only if arg is positive.

When Font Lock mode is enabled, text is fontified as you type it:

 - Comments are displayed in ‘font-lock-comment-face’;
 - Strings are displayed in ‘font-lock-string-face’;
 - Documentation strings (in Lisp-like languages) are displayed in
   ‘font-lock-doc-string-face’;
 - Language keywords ("reserved words") are displayed in
   ‘font-lock-keyword-face’;
 - Function names in their defining form are displayed in
   ‘font-lock-function-name-face’;
 - Variable names in their defining form are displayed in
   ‘font-lock-variable-name-face’;
 - Type names are displayed in ‘font-lock-type-face’;
```

```
 - References appearing in help files and the like are displayed
   in ‘font-lock-reference-face’;
 - Preprocessor declarations are displayed in
  ‘font-lock-preprocessor-face’;

   and

 - Certain other expressions are displayed in other faces according
   to the value of the variable ‘font-lock-keywords’.

Where modes support different levels of fontification, you can use the variable
‘font-lock-maximum-decoration’ to specify which level you generally prefer.
When you turn Font Lock mode on/off the buffer is fontified/defontified, though
fontification occurs only if the buffer is less than ‘font-lock-maximum-size’.
To fontify a buffer without turning on Font Lock mode, and regardless of buffer
size, you can use M-x font-lock-fontify-buffer.

See the variable ‘font-lock-keywords’ for customization.

----

Column-Number minor mode (indicator):
Toggle Column Number mode.
With arg, turn Column Number mode on iff arg is positive.
When Column Number mode is enabled, the column number appears
in the mode line.

----

Line-Number minor mode (indicator):
Toggle Line Number mode.
With arg, turn Line Number mode on iff arg is positive.
When Line Number mode is enabled, the line number appears
in the mode line.

----

Haskell-Indent minor mode (indicator Ind):
‘‘intelligent’’ Haskell indentation mode that deals with
the layout rule of Haskell.  tab starts the cycle
which proposes new possibilities as long as the TAB key is pressed.
Any other key or mouse click terminates the cycle and is interpreted
except for RET which merely exits the cycle.
Other special keys are:
    C-c = inserts an =
    C-c | inserts an |
    C-c o inserts an | otherwise =
these functions also align the guards and rhs of the current definition
```

```
    C-c w inserts a where keyword
    C-c . aligns the guards and rhs of the region.
    C-c > makes the region a piece of literate code in a literate script

Note: M-C-\ which applies tab for each line of the region also works
but it stops and ask for any line having more than one possible indentation.
Use TAB to cycle until the right indentation is found and then RET to go the
next line to indent.

Invokes ‘haskell-indent-hook’ if not nil.

----

Haskell-Doc minor mode (indicator Doc):
Enter haskell-doc-mode for showing fct types in the echo area (see variable docstring).
```

Antes de tratar da escrita de módulos em *Haskell*, vejamos quais são os tipos de dados disponíveis no módulo `Prelude` e como escrever expressões nesse módulo.

Capítulo 2

Tipos de Dados

O *Haskell* é uma linguagem de programação com uma disciplina de tipos rigorosa ("strongly typed"), quer isto dizer que toda a entidade num programa em *Haskell* tem um, e um só, tipo, sendo sempre possível determinar o tipo de uma determinada entidade. O contra-ponto deste tipo de linguagem são as linguagens sem tipos, ou melhor linguagens em que todas as entidades partilham um mesmo tipo, um exemplo deste tipo de linguagens é a linguagem Lisp. Existem argumentos a favor e contra cada uma destas duas aproximações, uma das grandes vantagens das linguagens com uma disciplina de tipos rigorosa reside na possibilidade de constatar a correcção dos programas através da verificação dos tipos, as linguagens deste tipo são também as que melhor suportam a modularidade e a abstracção.

Em termos práticos temos que um dado elemento tem sempre um tipo bem definido, de igual modo toda e qualquer função tem um domínio e um co-domínio bem definidos.

Em *Haskell* temos então um conjunto de tipos de base, pré-definidos, e um conjunto de construtores que permitem ao programador criar novos tipos a partir dos tipos de base.

2.1 Tipos de Base

O *Haskell* possui os seguintes tipos de base:

- Tipo Unitário ($\equiv \mathbf{1}$);
- Tipo Lógico, `Bool`;
- Tipo Caracter, `Char`;
- Tipos Numéricos:
  - `Int`;
  - `Integer`;
  - `Float`;
  - `Double`;

[1] (Versão 1.3)