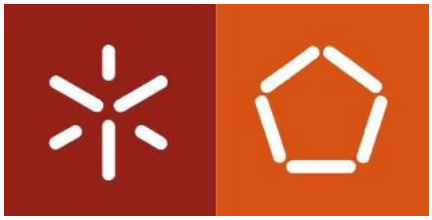

Escola de Engenharia

Departamento de Informática

Licenciatura em Engenharia Informática

Laboratórios de Informática I

Projecto – Haskell CSV Parser

Joao Monteiro

Conteúdo

Jornadas de Informática

As Jornadas de Informática são um evento académico realizado todos anos no Departamento de Informática da Universidade do Minho em que são convidados investigadores desta área para a realização de palestras e onde também são realizados *jobshops* envolvendo empresas associadas e patrocinadoras deste evento.

Para se inscrever nas Jornadas de Informática, preenche-se um formulário *on-line*, que gera um ficheiro CSV (*Comma Separated Values*) com os dados dos participantes.

Este projecto tem como objectivo fazer o reconhecimento (*Parsing*) deste ficheiro, validando as inscrições dos participantes e distinguindo-as pelos seus três tipos: aluno, universitário e empresa.

Também se pretende gerar crachás em LaTeX para todos os participantes, que servirão de acesso ao recinto onde se realizará o evento. Partindo de um ponto de vista económico, uma folha deverá conter o máximo de crachás possível.

Adicionalmente, devem ser feitos gráficos e tabelas multi-dimensionais, para uma posterior análise estatística acerca da comunidade envolvida neste evento.

O projecto será realizado com recurso à linguagem funcional *Haskell* e terá como principal *output* ficheiros LaTeX com os crachás, gráficos e tabelas multi-dimensionais.

1 Parsing

Para o reconhecimento dos ficheiros CSV, em primeiro lugar, e sabendo que a cada linha[1] correponde uma inscrição, separa-se o ficheiro em inscrições e, posteriormente, cada inscrição é separada pelos seus campos.

Então, usa-se a função *lines* já pré-definida e depois a função *virgula* auxiliada de *splitstringx*, de forma a acautelar as situações em que os campos têm vírgulas ou aspas. Note-se que *virgula* utiliza, de certa forma, *splitstringx* como um *map*.

```
splitstringx :: String -> Int -> [String]
splitstringx [] _ = [[]]
splitstringx (c:cs) x = let (s:ss) = splitstringx cs x
                        in if (c==',') && (even x) then []:(s:ss)
                  else if (c=='\"') then (splitstringx cs (x+1))
                                    else (c:s):ss

virgula :: [String] -> [[String]]
virgula [ ] = [ ]
virgula (x:xs) = (splitstringx x 0) : (virgula xs)
```

Assim, o ficheiro recebido, que é do tipo String, passa a ser do tipo [String]. Aqui, é eliminada a primeira linha do ficheiro, pois apenas tem a designação de cada campo. De seguida, passa a ser do tipo [[String]], onde cada elemento desta lista corresponde a um campo de uma inscrição.

A partir desta lista, cria-se uma lista do tipo [(Inscrição)] com a função *tuplar*, que é uma lista de tuplos em que cada tuplo corresponde a uma Inscrição = (Nome, Email, Curso, Numero, Univ, Curso1, Filiacao, Jantar, Almoco, Join, Outro, Data) e o tipo de cada um destes campos continua a ser String. Considera-se o campo *Email* a chave da inscrição.

Nesta etapa, são validadas as inscrições, com as funções *validacao* e *repetidos*, em que as inscrições repetidas ou inválidas vao gerar um documento LaTeX. No caso de haver inscrições repetidas, mantém-se a mais recente.

```
validacao :: [(Inscricao)] -> String
validacao [] = ""
validacao ((n,e,c,nm,u,c1,f,j,a,jo,o,d):r)
           | (n /= "") && (e /= "") && (c /= "") && (nm /= "")
             && (u == "") && (c1 == "") && (f == "") = validacao r
           | (n /= "") && (e /= "") && (c == "") && (nm == "")
             && (u /= "") && (c1 /= "") && (f == "") = validacao r
           | (n /= "") && (e /= "") && (c == "") && (nm == "")
             && (u == "") && (c1 == "") && (f /= "") = validacao r
           | otherwise = "O registo com o" ++ show e ++ " tem a inscricao
             invalida \r\n" ++ (validacao r)
```

[1] Uma nova linha corresponde ao caracter *newline*

Por fim, através da função *diferentesx*, já sem as inscrições inválidas e/ou repetidas, o ficheiro é ordenado pelas inscrições de alunos, universitários e empresas.

Note-se que: uma inscrição é do tipo *aluno* se *Nome*,*Email*,*Curso*,*Numero* estão preenchidos e *Univ*,*Curso1*,*Filiacao* estão vazios; uma inscrição é do tipo *universitário* se *Nome*,*Email*,*Univ*,*Curso1* estão preenchidos e *Curso*,*Numero* e *Filiacao* estão vazios; uma inscrição é do tipo *empresa* se *Nome*,*Email*,*Filiacao* estão preenchidos e *Curso*,*Numero*,*Univ*,*Curso1* estão vazios.

Uma inscrição é válida se é de algum destes tipos. Assim, a partir deste ponto, apenas se utilizam inscrições válidas.

2 Crachás

De início, substitui-se na lista do tipo [(Inscricao)], o campo *Nome* pelo primeiro e último nome, com a função *nomes*, visto que para os crachás apenas estes são necessários. Então, divide-se a lista em três listas do tipo [(Inscricao)] através das funções *isaluno*,*isexterno* e *isempresa*, uma para cada um dos tipos de participantes.

```
isaluno :: [(Inscricao)] -> [(Inscricao)]
isaluno [] = []
isaluno ((n,e,c,nm,u,c1,f,j,a,jo,o,d):r)
        | (n /= "") && (e /= "") && (c /= "") && (nm /= "") && (u == "")
         && (c1 == "") && (f == "") = (n,e,c,nm,u,c1,f,j,a,jo,o,d) : (isaluno r)
        |otherwise = isaluno r
```

Depois, geram-se os crachás de cada tipo, separadamente, com as funções *texalunos*,*texexterno* e *texempresa* e, como há uma parte comum a cada um destes documentos, usa-se a função *tex*, que contém esta parte comum, para gerar os três documentos.

```
texalunos :: [(Inscricao)] -> String
texalunos [] = ""
texalunos ((n,e,c,nm,u,c1,f,j,a,jo,o,d):r) ="\\AddToShipoutPicture{\\BackgroundPic
{design/background-participante}}\n\n\\includegraphics{design/logo}\\\\\n\n\\
addvspace{5mm}\n\n\\begin{center}\n\t\\huge{NAME}\n\t\\scriptsize{\n\t\\begin
{tabular*}{0.75\\textwidth}{c}\n\t"++ n ++" \\\\\n\t\t\\hline\n\t\\end{tabular*}}
\\\\\n\tCOMPANY\\\\ Universidade do Minho - " ++ c ++ "\n\t\t\n\\end{center}\n\n\\
begin{flushright}\n\t\\begin{tabular}{r l l}\n%\t\t\\normalsize{Participante}& &\n
\t\t\\normalsize{Participante} & &\n\t\\end{tabular}\n\\end{flushright}\n\n\\
pagebreak" ++ (texalunos r)
```

Posteriormente, estes documentos passam para outros documentos LaTeX só que agora cada folha contém dez crachás em vez de apenas um, assim, poupar-se-à papel no momento da impressão dos crachás. Para este efeito, usa-se *mpc*.

3 Estatísticas

De modo a analisar estatisticamente os dados acerca dos participantes, é necessário agrupar esta informação. Os dados escolhidos são relativos aos alunos nos cursos de Informática da Universidade do Minho, aos participantes que preferiram almoçar e/ou jantar na cantina e também às razões para se inscreveram na Jornadas de Informática.

Para saber quais os alunos nos cursos de Informática, usa-se a função *statsc*, que retorna uma lista com tamanho igual ao número de cursos, em que um elemento dessa lista é 1 se o aluno pertence ao curso e 0, caso contrário.

```
statsc :: (Inscricao) -> [Int]
statsc (n,e,c,nm,u,c1,f,j,a,jo,o,d) | (c == "LEI") = [1,0,0,0,0,0]
                                    | (c == "LCC") = [0,1,0,0,0,0]
                                    | (c == "MEI") = [0,0,1,0,0,0]
                                    |  (c == "MI") = [0,0,0,1,0,0]
                                    |  (c == "MERSCOM") = [0,0,0,0,1,0]
                                    |  (c == "MBIO") = [0,0,0,0,0,1]
                                    | otherwise = [0,0,0,0,0,0]
```

Mas esta função apenas serve para uma inscrição, portanto, é necessário recorrer a um *map* aplicado a todas as inscrições. De seguida, contam-se os alunos pertencentes a cada curso com a função *zipall*, tendo *(+)* e *(map(statsc))* como argumentos.

```
zipall::(a->a->a)->[[a]]->[a]
zipall _ [] = []
zipall _ (a:[]) = a
zipall f (l:ls) = zipWith f l (zipall f ls)
```

Finalmente, cria-se um ficheiro DAT com estes dados através da função *csdat* e recorre-se ao *gnuplot* para mostrar as estatisticas em formato LaTeX com a função *unitable1*.

```
csdat :: [Int] -> String
csdat (a:b:c:d:e:f:r) = ("1\t" ++ show a ++ "\n2\t " ++ show b ++ "\n3\t "
                        ++  show c ++ "\n4\t " ++ show d ++ "\n5\t " ++ show e
                        ++ "\n6\t" ++ show f)
```

De modo análogo, usam-se as funções *statsaj* para contar os participantes que vão almoçar e/ou jantar na cantina, *aljndat* para criar o ficheiro DAT e *unitable3* para gerar o LaTeX. Para analisar as razões que levam os participantes a virem às Jornadas de Informática, utilizam-se as funções *statsi*, *indat* e *unitable2*.

4 Análises Multi-dimensionais

Nesta secção, optou-se por cruzar os dados obtidos na tarefa anterior. Então, pretende-se saber quantos elementos de cada curso vão almoçar e/ou jantar na cantina, e ainda que razões os levam a inscrever-se nas Jornadas de Informática.

Assim, filtra-se da lista de inscrições as listas com as inscrições que contêm os elementos de cada curso. Foi necessário definir a função *filterx* pois com *filter* não se conseguiu fazer o *pattern matching* necessário.

```
filterx :: (String -> Bool) -> [(Inscricao)] -> [(Inscricao)]
filterx _ [] = []
filterx p (((n,e,c,nm,u,c1,f,j,a,jo,o,d):r))
                          | p c = ((n,e,c,nm,u,c1,f,j,a,jo,o,d) : (filterx p r))
                          | otherwise = filterx p r
```

De seguida, no caso dos almoços/jantares, constrói-se um lista do tipo [(String,[Int])]. O primeiro elemento do tuplo é o nome do curso, o segundo é uma lista com dois elementos: um com o número de pessoas desse curso que vão ao almoço e o segundo ao jantar.

Com esta função, *analiseaj*, simplesmente agrupam-se os dados obtidos com as funções da secção anterior, neste caso, das funções *zipall (+) (map statsaj)*.

```
analise2 :: [(Inscricao)] -> String -> (String,[Int])
analise2 l s |((filterx (s==) l) == []) = (s,[0,0])
             |otherwise = (s,(zipall (+) (map statsaj (filterx (s==) l))))

analiseaj :: [(Inscricao)] -> [(String,[Int])]
analiseaj l = [(analise2 l "LEI"),(analise2 l "LCC"),(analise2 l "MEI"),
               (analise2 l "MI"),(analise2 l "MERSCOM"),(analise2 l "MBIO")]
```

Tal como na secção anterior, cria-se um ficheiro DAT com estes dados através da função *analiseajdat* e recorre-se ao *gnuplot* para mostrar as tabelas com *multitable2*.

```
analiseajdat :: [(String,[Int])] -> String
analiseajdat [] = " "
analiseajdat ((c,(h:t:z)):r) =  ("\"" ++ c ++ "\" " ++ show h ++ " "
                                 ++ show t ++ "\n") ++ analiseajdat r
```

Procedendo de modo análogo, criam-se as tabelas com os dados referentes ao cruzamento entre as razões para a participação nas Jornadas de Informática e os alunos dos cursos de Informática, recorrendo às funções *analiseint*, *analiseintdat* e *multitable1*.

Note-se que o output das secções 3 e 4 vai para um único ficheiro LaTeX com a função *juntatex*.

5 Main

O encadeamento de todo o projecto é feito desta forma.

```
main =do
--Parsing
   putStrLn "+++++++++++++++++++++++++++++++++++++++++++"
   putStrLn   "++**   JOIN   PARSER   3.4beta      **++"
   putStrLn "++++** Insira o nome do ficheiro csv **++++"
   putStrLn "++++_____________________________________++++"
   csvname <- getLine
   x <- readFile (csvname ++".csv")
   y <- return ((drop 1 (lines x)) :: [String])
   z <- return ((virgula y) :: [[String]])
   w <- return ((map tuplar z))
   c <- return ((validacao w)++(repetidos w) ++ (validnames w))

--Crachás
   a <- return (diferentes w)
   as <- return (diferentesx a)
   n <- return (nomes as)
   parsinglatex <- return( parsingtex (c))
   alunos <- return (isaluno n)
   externos <- return (isexterno n)
   empresas <- return (isempresa n)
   crashaaluno <- return (tex (texalunos alunos))
   crashaexterno <- return (tex (texexterno externos))
   crashaempresa <- return(tex (texempresa empresas))
   multipagescrashaaluno <- return (mpc ("cards_alunos.pdf"))
   multipagescrashaexterno <- return (mpc ("cards_externos.pdf"))
   multipagescrashaempresa <- return (mpc ("cards_empresas.pdf"))

--Estatística e Análise Multi-dimensional
   cscount <- return (zipall (+) (map statsc (as)))
   csdatcount <- return (csdat (cscount))
   aljncount <- return (zipall (+) (map statsaj (as)))
   aljndatcount <- return (aljndat (aljncount))
   incount <- return (zipall (+) (map statsi (as)))
   indatcount <- return (incsv (incount))
   analiseintmontar <- return(analiseint (n))
   analiseajmontar <- return(analiseaj (n))
   analisecsvint <- return(analiseintdat (analiseintmontar))
   analisecsvaj <- return(analiseajdat (analiseajmontar))
   tabela1 <- return(unitable1 (cscount))
   tabela2 <- return(unitable2 (incount))
   tabela3 <- return(unitable3 (aljncount))
   tabela4 <- return(multable1 (analiseintmontar))
   tabela5 <- return(multable2 (analiseajmontar))
   statslatex <- return(juntatex (tabela1) (tabela2) (tabela3) (tabela4) (tabela5))
   graficos <- readFile "gnustats/graficos.tex"
```

```
--Criçao de Ficheiros
   writeFile "CRASHAS_ALUNOS.tex" multipagescrashaaluno
   writeFile "CRASHAS_EXTERNOS.tex" multipagescrashaexterno
   writeFile "CRASHAS_EMPRESAS.tex" multipagescrashaempresa
   writeFile "cards_alunos.tex" crashaaluno
   writeFile "cards_externos.tex" crashaexterno
   writeFile "cards_empresas.tex" crashaempresa
   writeFile "int.dat" incsvcount
   writeFile "cs.dat" cscsvcount
   writeFile "aj.dat" aljncsvcount
   writeFile "Estatisticas.tex" statslatex
   writeFile "ESTATS.tex" statslatex
   writeFile "analise1.dat" analisecsvint
   writeFile "analise2.dat" analisecsvaj
   writeFile "Parsing.tex" parsinglatex
   writeFile "Graficos.tex" graficos

-- Foi necessária a criaçao de uma Shell Script para estabelecer a sequência
  de execuçao dos comandos desta secçao
   runCommand "sh scrip.sh"
```

Considerações Finais

Os objectivos principais propostos para este projecto foram alcançados:

- os ficheiros CSV são reconhecidos e devidamente validados;
- os crachás são produzidos de acordo com o tipo de participante;
- os dados acerca dos inscritos são representados graficamente;
- as tabelas multi-dimensionais são também representadas.

Com a realização do projecto a sensação de que a linguagem funcional Haskell é muito eficiente cresceu.

As maiores dificuldades sentidas foram, no in´ıcio, em saber qual seria a melhor forma de obter uma solução prática para os problemas propostos e, numa fase final, a geração de tabelas e gráficos com o *gnuplot*.

Contudo, considera-se o resultado final muito positivo quando comparado com as expectativas iniciais.