Métodos de Programação III
LESI + LMCC (3^o ano)

Trabalho Prático n^o 2

(**Representação Gráfica de Autómatos Finitos e Minimização do Número de Estados**)

Ano lectivo 2002/2003

Grupo:

nnnnn Lxxx 1.nome ult.nome

nnnnn Lxxx 1.nome ult.nome

nnnnn Lxxx 1.nome ult.nome

O trabalho deve ser realizado em grupo, e deve ser entregue acompanhado deste relatório até 13 de Dezembro de 2002 ao fim da tarde

Conteúdo

1 Introdução **1**

1.1 Objectivos e Organização 2

2 Representação Gráfica de Autómatos Finitos **2**

3 Representação Gráfica de Expressões Regulares **2**

4 Minimização de Estados de Autómato Finitos **3**

5 A Implementação em Haskell **3**

5.1 A função `dfa2graph` 4

5.2 A função `ndfa2graph` 4

5.3 A função `re2graph` 5

5.4 A função `beautifyDfa` 5

5.5 A função `minimizeDfa` 5

6 Execução do Programa Desenvolvido **6**

6.1 Representação Gráfica dos Autómatos 6

7 Conclusão 6

A Extras ao trabalho prático 7

A.1 Desenho dos Autómatos 7

A.2 Função de equivalência de Expressões Regulares 7

A.3 Desempenho dos Vários Reconhecedores de Linguagens Regulares 7

A.4 Função Standard de Minimização de Estados 7

1 Introdução

Este texto está escrito em **Literate Haskell** e pretende ser, além do enunciado do 2º Trabalho Práctico da disciplina de MP3, um esboço (ou esquema geral) do relatório que cada grupo tem de apresentar aquando da discussão em computador do projecto realizado. De acordo com o princípio de *Literate Programming*, o código a desenvolver deve aparecer naturalmente inserido no meio do documento. Assim, o documento presente pode ser interpretado como um documento LaTeX ou como um puro programa na linguagem Haskell.
O documento é obtido "compilando" este ficheiro em LaTeX:

```
latex mp302tp2.lhs
```

O programa Haskell incluído neste ficheiro é compilado/interpretado directamento pelos compiladores/interpretadores de Haskell, executando por exemplo o comando:

```
hugs mp302tp2.lhs
```

Para isso o código Haskell deve estar delimitado pelos comandos `\begin{code}` e `\end{code}`.

A ideia essencial do trabalho em causa é gerar código para uma ferramenta de desenho de grafos, o GraphViz, de modo a poder-se visualizar directamente autómatos finitos (deternimistas ou não).
Dessa forma e recorrendo às funcões de conversão desenvolvidas nas aulas teórico-prácticas, será possível desenhar graficamente a linguagem definida por uma dada expressão regular.
Complementarmente, pede-se que desenvolva uma função para minimizar o número de estados de um autómato dado.
Na resolução desta proposta de trabalho deve ser reutilizado todo o conjunto de funções criadas ao longo do semestre.

1.1 Objectivos e Organização

Neste trabalho prático pretende-se que os alunos implementem um programa em Haskell para efectuar as seguintes tarefas:

- Representação Gráfica de Autómatos Finitos.

- Representação Gráfica de Expressões Regulares.
- Minimização de Estados de Autómatos Finitos.

Para isso a folha propõe um enunciado único que tem de ser resolvido por todos os grupos, sugerindo-se em apêndice alguns extras que os grupos poderão, ou não, concretizar.

Os alunos devem ler este enunciado atentamente e completá-lo de modo a obter o relatório do trabalho prático. Para tal, deverão preencher as secções deixadas em branco, incluir o código desenvolvido, completar os parágrafos que estão identificados com ..., apagar partes do enunciado que não façam sentido como documentação do trabalho, etc.

2 Representação Gráfica de Autómatos Finitos

Os autómatos finitos são compreendidos mais facilmente se forem representados numa notação gráfica. Dado um autómato $\mathcal{A} = (\mathcal{V}, \mathcal{Q}, \mathcal{S}, \mathcal{Z}, \delta)$ uma representação gráfiva é construída de acordo com as seguintes regras:

- ...

Por exemplo, o seguinte autómato definido formalmente a seguir

...

induz de acordo com as regras anteriores o seguinte grafo:

3 Representação Gráfica de Expressões Regulares

...

4 Minimização de Estados de Autómato Finitos

A minimização do número de estado dos autómatos finitos é uma tarefa muito importante devido aos seguintes aspectos: ....

- ...
- ...

5 A Implementação em Haskell

Para a realização deste trabalho foram executadas das seguintes tarefas:

- A primeira tarefa consistiu em completar os módulos `Dfa` e `Ndfa` (desenvolvidos nas aulas teórico-práticas). Assim, inclui-se as funções `show` para os tipos `Dfa` e `Ndfa`, de modo aos autómatos finitos serem vizualizados textualmente tal como são escritos. Foi ainda útil definir uma função para gravar os autómatos finitos em ficheiro. O tipo desta função é:

  ```
  faIO :: Show fa => fa -> String -> IO ()
  ```

  Para tal, os tipos `Dfa` e `Ndfa` são instâncias da classe `Show`.

- A segunda tarefa consistiu em utilizar o sistema GraphViz para a vizualização de grafos. O sistema GraphViz está a ser desenvolvido nos laboratórios da *AT&T* e é de domínio publico:

  `http://www.research.att.com/sw/tools/graphviz/`

  Assim, depois de fazermos o *download* deste software, estudamos a melhor forma de representar autómatos finitos neste sistema.

  Nota: Este sistema processa grafos representados (em ficheiro) numa notação textual.

- Após compreendermos a notação textual dos grafos do sistema GraphViz, desenvolvemos uma nova função `show` que representa os autómatos finitos deterministas nessa notação. Definimos ainda uma função que grava em ficheiro os autómatos. Este ficheiro é depois passado ao GraphViz que o apresenta graficamente em vários formatos (postscript, gif, etc).

  Repetimos esta tarefa para os autómatos finitos não deterministas. Nestes autómatos tivemos de ter em atenção a representação das transições espontâneas (i.e., transições-ϵ). A forma como tratamos estas transições é descrita na secção 5.2.

- A terceira tarefa consistiu na representação gráfica de expressões regulares. Mais concretamente, vizualizamos expressões regulares como o autómatos finitos deterministas (equivalentes, i.e., que definem a mesma linguagem).

- A última tarefa consistiu na construção de uma função que efectua a minimização de estados de um autómato finito deterministas. Após termos construído esta função, re-escrevemos a função que representa expressões regulares como autómatos finitos deterministas de modo a utilizar o autómato de estados minimos.

As funções construídas foram incluídas num novo módulo chamado `DrawFa` que importa os módulos: `RegExp`, `Dfa`, `Ndfa`, `RegExp2Ndfa`, `Ndfa2Dfa`, todos eles já desenvolvidos nas aulas teórico-práticas.

Código Haskell

```
--
-- Módulo de Minimização de Estados e Representação Gráfica de
--            Autómatos Finitos em Haskell
--
-- Métodos de Programação III
-- Universidade do Minho
-- 2001/2002
--


module DrawFa where

import RegExp
import Dfa
import Ndfa
import RegExp2Ndfa
import Ndfa2Dfa
```

5.1 A função `dfa2graph`

Código Haskell

```
-- dfa2graph :: (Show st, Show sy) => Dfa st sy -> [Char] -> [Char]
```

5.2 A função `ndfa2graph`

Código Haskell

```
-- ndfa2graph ::
```

5.3 A função `re2graph`

Código Haskell

```
-- re2DiGraph :: RegExp -> [Char] -> [Char]
```

5.4 A função `beautifyDfa`

A função de conversão de um autómato não determinista em determinista constrói um autómato determinista em que os identificadores dos estado são listas. Dependendo da complexidade do autómato, estas listas poderão ter comprimentos razoáveis. Este facto origina que as representações gráficas fiquem pouco claras uma vez que os nomes dos estados ficam enormes. Assim, construímos uma função `beautifyDfa` que "substitui" esses identificadores de estados por um número apenas. O tipo e a definição desta função apresenta-se a seguir:

Código Haskell

```
-- beautifyDfa :: Eq st => Dfa st sy -> Dfa Int sy
```

tendo esta função definida, podemos então re-escrever a função `dfa2graph` de modo a utilizar esta representação:

Código Haskell

```
-- dfa2graph' = dfa2graph . beautifyDfa
```

5.5 A função `minimizeDfa`

A função de minimização dos estados de um autómato finito determinista completo baseia-se no seguinte algoritmo (apresentado nas aulas teóricas):

1. Primeiro, ...
2. Segundo, ...
3. Terceiro, ...
4. Finalmente, ...

Em Haskell esta função é definida muito simplesmente do seguinte modo:

Código Haskell

```
-- minimizeDfa :: (Eq sy, Ord st) => Dfa st sy -> Dfa [[st]] sy
```

6 Execução do Programa Desenvolvido

Para ilustrar o bom funcionamento do nosso programa vamos apresentar nesta secção alguns exemplos obtidos com a sua execução.
Considere-se a seguinte linguagem regular definida pela expressão regular:

$$er =$$

utilizando a função `regExp2Ndfa` obtém-se o seguinte autómato finito não determinista:

...

E utilizando a função de conversão para deterministas obtém-se

...

Agora minimizando os estados com a função `minimizeDfa` obtém-se

...

E podemos agora utilizar as funções de aceitação `matches`, `ndfaaccept` e `dfaaccept` para aceitar a frase ...

6.1 Representação Gráfica dos Autómatos

Nesta secção incluimos alguns grafos produzidos pelo sistema GraphViz a partir de autómatos produzidos pelas funções `dfa2graph` e `dfa2graph`.

7 Conclusão

...sintese do conteúdo do relatório...
...estado final do trabalho: requisitos atingidos; requisitos não atingidos; extras...
...trabalho futuro....

A Extras ao trabalho prático

Como trabalho complementar ao conjunto de funções que foram já desenvolvidas, construímos ainda funções extra para realizar as seguintes tarefas[1]:

[1] Os alunos devem apagar deste relatório os extra que não realizaram.

1. Melhoramos a representação gráfica dos autómatos finitos através da seguinte estratégia: quando do mesmo estado se transita para um outro estado por mais que um símbolo, em vez de se adicionar ao grafo uma seta por cada uma dessas transições, inclui-se apenas uman transição em que a estiqueta são os vários símbolos separados por vírgula. Esta optimização tem sido utilizada inúmeras vezes quando se desenham os autómatos à mão.

2. Desenvolvemos uma função que prova que duas expressões regulares são equivalentes ou não. A prova da equivalência de expressões regulares foi feita na aula teórico-prática nº2 utilizando para tal a álgebra de expreessões regulares. A função que desenvolvemos efectua a prova através da comparação dos autómatos finitos deterministas minimos que são gerados a partir de cada uma das expressões regulares. Se estes autómatos forem iguais, as expressões regulares são equivalentes, caso contrário as expressões regulares não são equivalentes.

3. Analisamos da complexidade dos vários reconhecedores de linguagens regulares. Isto é a complexidade dos vários reconhecedores de linguagens regulares baseados em expressões regulares, autómatos finitos deterministas e não deterministas e autómatos com um número minimo de estados. Para tal consideramos uma dada linguagem e uma frase dessa mesma linguagem, modelamos essa linguagem nos vários modelos apresentados e utilizamos o número de reduções e células usadas pelo interpretador `Hugs` na aceitação da frase. Obviamente, incluimos neste relatório uma análise e um comentário aos resultados obtidos pelo `Hugs`.

4. Implementamos o algoritmo "*standard*" de minimização de estados de um autómato finito determinista.

5. Por último, incluimos ainda a resolução de todas as fichas teórico-práticas que abordam a matéria envolvida neste trabaçho prático.

A.1 Desenho dos Autómatos

A.2 Função de equivalência de Expressões Regulares

A.3 Desempenho dos Vários Reconhecedores de Linguagens Regulares

A.4 Função Standard de Minimização de Estados