

Processamento de Linguagens e Compiladores

LMCC, Universidade do Minho

Ano lectivo 2005/2006

João Saraiva

Ficha Teórico-Prática N°4

Este texto está escrito em **literate Haskell**. Isto é, pode ser interpretado como um documento \LaTeX ou como um puro programa na linguagem Haskell. Responda às perguntas sobre Haskell neste próprio ficheiro para assim produzir o programa e a sua documentação.

Esta ficha contém a resolução do trabalho prático nº1 realizado pelos alunos no ano lectivo 2000/2001.

1 Conversão de Autómatos Finitos Não Deterministas em Deterministas

A ideia geral do método de conversão de um autómato finito não determinista completo num autómato finito determinista completo é a seguinte: a cada estado do AFD corresponde um conjunto de estados do AFND. Isto acontece porque o reconhecimento de uma frase por um AFND pode ser feito utilizando caminhos diferentes (e conseqüentemente seqüências de estados diferentes). Assim, cada estado de um AFD contém a informação (i.é., os estados) que se podem alcançar a partir de um estado do AFND transitando por um dado símbolo do vocabulário.

Considerando o autómato finito não-determinista $\mathcal{A}_{nd} = (\mathcal{V}, \mathcal{Q}, \mathcal{S}, \mathcal{Z}, \delta)$ e o autómato finito determinista $\mathcal{A}_d = (\mathcal{V}', \mathcal{Q}', \mathcal{S}', \mathcal{Z}', \delta')$ a conversão de \mathcal{A}_{nd} em \mathcal{A}_d é feita do seguinte modo:

- $\mathcal{V}' = \mathcal{V}$
- $\mathcal{S}' = \epsilon\text{-closure } \delta \mathcal{S}$
- \mathcal{Q}' define-se recursivamente por:
 - $\mathcal{S}' \in \mathcal{Q}'$

– $qs \in \mathcal{Q}' \Rightarrow (\delta' qs v) \in \mathcal{Q}'$, para todo $v \in \mathcal{V}$

- $\delta' qs v = walk \delta qs \langle v \rangle$
- $\mathcal{Z}' = \{q \in \mathcal{Q}' \mid q \cap \mathcal{Z} \neq \emptyset\}$

em que as funções ϵ -closure e walk foram definidas formalmente na ficha teórico-prática nº4. Informalmente estas regras dizem o seguinte: o vocabulário do autómato finito determinista \mathcal{V}' é o mesmo que o do não determinista. O seu estado inicial \mathcal{S}' é o conjunto formado pelos estados iniciais do AFND e todos os que se alcançam transitando por ϵ (i.e. sem consumir nenhum símbolo). O conjunto de estados \mathcal{Q}' do AFD é constituído pelo seu estado inicial e ainda por todos os estados que se alcançam em transições de um estado de \mathcal{Q}' por um símbolo do vocabulário v . O conjunto de estados finais \mathcal{Z}' é formado por todos estados do AFD que incluem algum estado final do AFND.

1.1 Considere o seguinte autómato finito não determinista $\mathcal{A}_1 = (\mathcal{V}_1, \mathcal{Q}_1, \mathcal{S}_1, \mathcal{Z}_1, \delta_1)$, com $\mathcal{V}_1 = \{ 'a', 'b' \}$, $\mathcal{Q}_1 = \{ A, B, C \}$, $\mathcal{S}_1 = \{ A \}$, $\mathcal{Z}_1 = \{ D \}$ e a função de transição é

δ_1	A	$'a'$	$=$	$\{ B, C \}$
δ_1	A	ϵ	$=$	$\{ D \}$
δ_1	B	$'a'$	$=$	$\{ C \}$
δ_1	C	$'b'$	$=$	$\{ D \}$
δ_1	D	ϵ	$=$	$\{ C \}$
δ_1	-	-	$=$	$\{ \}$

Cálcule um autómato determinista equivalente.

1.1 Construção de uma Tabela de Conversão

Uma técnica bastante mais sistemática e estruturada para efectuar a conversão entre um AFND e um AFD consiste na utilização de uma *tabela* que auxilia o processo de conversão. Esta tabela que chamamos *tabela de conversão*, é constituída por várias colunas. A primeira coluna está associada aos estados do AFD e as restantes colunas estão associadas a cada um dos símbolos do vocabulário. Cada linha da tabela define as transições associadas ao estado do DFA que consta da sua primeira coluna.

Esta tabela é preenchida do seguinte modo: a primeira coluna da primeira linha preenche-se com o estado inicial do AFD (calculado através do ϵ -closure dos estados iniciais do AFND). As restantes colunas da primeira linha da tabela preenchem-se com o conjunto de estados que se alcançam caminhando no AFND a partir do conjunto de estados da primeira coluna e consumindo a frase constituída por um caracter: o símbolo do vocabulário associado à coluna. Cada um destes conjuntos de símbolos encontrados corresponderá um novo estado do AFD. Para cada novo estado proceder-se-á de igual modo, i.e., acrescentando uma linha na tabela, pondo esse estado na primeira coluna e determinando as suas transições. O processo termina quando os estados encontrados já estiverem todos considerados. Isto é, o processo convergir.

1.2 Um Exemplo

Consideremos a definição do seguinte autômato finito não determinista \mathcal{A}_1 . Utilizando a técnica definida anteriormente obtemos a seguinte tabela auxiliar de conversão:

$Q' \setminus T'$	'a'	'b'
$S' = \epsilon\text{-closure } \delta_1 S_1$ $= \{A, D, C\}$	$walk \delta_1 \{A, D, C\} ['a']$ $= \{B, C\}$	$walk \delta_1 \{A, D, C\} ['b']$ $= \{C, D\}$
$\{B, C\}$	$walk \delta_1 \{B, C\} ['a']$ $= \{C\}$	$walk \delta_1 \{B, C\} ['b']$ $= \{C, D\}$
$\{C, D\}$	$walk \delta_1 \{C, D\} ['a']$ $= \{\}$	$walk \delta_1 \{C, D\} ['b']$ $= \{C, D\}$
$\{C\}$	$walk \delta_1 \{C\} ['a']$ $= \{\}$	$walk \delta_1 \{C\} ['b']$ $= \{C, D\}$
$\{\}$	$\{\}$	$\{\}$

1.2 Considere as seguintes linguagens regulares modeladas pelas seguintes expressões regulares:

1. $\mathcal{L} = (a + b)^*abb$
2. $\mathcal{L}_{reais} = ('+' '+' -')? d^* ('!)? d^+$

Para cada uma destas linguagens apresente um autômato finito não determinista que as definem. Calcule os autômatos deterministas equivalentes (utilize as regras de conversão de autômatos não deterministas em deterministas).

2 Conversão de Autômatos Finitos Não-Deterministas em Deterministas em Haskell

Vamos agora construir em Haskell a função de conversão de autômatos não deterministas completos para autômatos deterministas completos. Começamos por definir um novo módulo chamado Ndfa2Dfa:

Código Haskell

```

--
-- Modulo de Conversao de Automatos Finitos Nao-Deterministas
--     em Automatos Finitos Deterministas em Haskell
--
--
-- Processamento de Linguagens e Compiladores
-- 2005/2006
--
--

module Ndfa2Dfa where

import List
import RegExp
import Dfa
import Ndfa

```

Para tornar os tipos, e conseqüentemente as funcoes, mais legiveis vamos definir um tipo para os estados do automato determinista que se obtem apos a conversao, (isto e, o tipo `StDfa` que e uma lista de estados do AFND) e o tipo da tabela de conversao (tipo `CT`):

Codigo Haskell

```

type StDfa st = [st]
type CT      st = [( StDfa st, [StDfa st])]

```

Para desenvolver a funcao de conversao vamos necessitar de algumas funcoes auxiliares. Note que a funcao `epsilon_closure` foi ja definida na ficha teorica anterior, portanto a primeira coluna da primeira linha da tabela pode ser calculada facilmente por esta funcao. Vamos entao definir uma funcao que preenche as restantes colunas da tabela. Assim, definimos a funcao `OneRow` que dada a funcao de transicao de estado, o estado do AFD que esta na primeira coluna, o alfabeto, calcula (atraves da funcao ja definida `ndfawalk`) a lista de estados (do AFD) para cada uma das restantes colunas da tabela:

Codigo Haskell

```

oneRow :: Ord st
        => (st -> Maybe sy -> [st]) -> StDfa st -> [sy] -> [StDfa st]
oneRow delta sts alfabet = map ( v -> sort (ndfawalk delta sts [v])) alfabet

```

2.1 *Considere de novo o automato \mathcal{A}_2 . Modele este automato em Haskell e utilize as funcoes anteriores e a sua execucao no sistema Hugs de modo a calcular:*

1. *Calcular (ϵ -closure $\delta_2 \mathcal{S}_2$). Isto e, calcule a primeira coluna da primeira linha da tabela de conversao.*

2. Calcular o valor das restantes colunas da primeira linha da tabela usando a função `oneRow`.

Código Haskell

Uma linha da tabela introduz (potencialmente) novos estados do autómato finito determinista. Cada um destes estados induz, por sua vez, uma nova linha na tabela. Seguidamente definimos uma função `consRows` que constrói linhas completas da tabela. Esta função tem como argumentos a função de transição de estados, um conjunto de estados (do AFD) e o alfabeto. A função constrói uma tabela de transição de estados.

Código Haskell

```
consRows :: Ord st => (st -> Maybe sy -> [st]) -> [StDfa st] -> [sy] -> CT st
consRows delta [] v = []
consRows delta (q:qs) v = nub ((q , oneRow delta q v) : (consRows delta qs v))
```

2.2 No exercício 2.1.2, introduziram-se dois novos estados no autómato finito determinista. Calcule, com auxílio da função `consRows`, as novas linhas da tabela de conversão que esses estados induzem.

Código Haskell

Até este momento, temos apenas funções que trabalham com linhas da tabela de conversão. Vamos agora definir uma função que dado uma tabela de conversão parcialmente preenchida contrói uma nova tabela de conversão "totalmente"preenchida. Por uma tabela parcialmente preenchida entedemos uma tabela em que existem estados nas colunas do lado direito que ainda não foram considerados, isto é, ainda não deram origem a uma linha.

Código Haskell

```
ndfa2CtStep :: Ord st => (st -> Maybe sy -> [st]) -> [sy] -> CT st -> CT st
ndfa2CtStep delta v [] = []
ndfa2CtStep delta v ((s,ss):rs) =
    (s,ss):(consRows delta ss v) 'union' (ndfa2CtStep delta v rs)
```

2.3 Considere a primeira linha da tabela de conversão:

```
[("ACD", ["BC", "CD"])]
```

Cálcule a tabela de conversão que se obtém com a função `ndfa2CtStep`. A tabela obtida está totalmente preenchida?

Código Haskell _____

Como a alínea anterior prova a tabela que se obtém está de novo parcialmente preenchida uma vez que os estados que ainda não tinham sido considerados induziram novos estados que não estão a ser considerados nesta. Portanto, precisamos de repetir a construção da tabela até que o processo convirja, isto é, todos os estados estejam considerados.

2.4 Considere de novo a tabela de construção do autómato deterministas.

1. Execute de novo a função `ndfa2CtStep` em que a tabela de conversão passada como argumento é a tabela construída no exercício 2.3.
2. Execute de novo a função `ndfa2CtStep` em que a tabela de conversão passada como argumento é a tabela construída na alínea anterior.

Código Haskell _____

Isto é, a função convergiu após 3 invocações. Em Haskell usamos a função `limit` para definir o limite de uma função. A função `ndfa2ct` utiliza a função `limit` (da função `ndfa2CtStep`) para calcular a tabela de conversão. Como tabela inicial é passada uma tabela que contém unicamente a sua primeira linha preenchida.

Código Haskell _____

```
ndfa2ct :: Ord st => Ndfa st sy -> CT st
ndfa2ct (Ndfa v q s z delta) = limit (ndfa2CtStep delta v) ttFstRow
  where ttFstRow = consRows delta [epsilon_closure delta s] v
```

2.5 Considere de novo o autómato A_2 . Utilize a função `ndfa2ct` para calcular a tabela de conversão.

Código Haskell _____

2.6 *Considere os autómatos finitos não deterministas definidos na exercício 1.2. Utilize a implementação em Haskell para provar que a tabela de conversão obtidas estão correctas.*

A construção da tabela de conversão é apenas um passo intermédio no processo de conversão de um AFND em AFD. A seguir apresenta-se a função `ndfa2dfa` que recebe como argumento um AFND e constrói um AFD. Analise com atenção esta função e o seu tipo.

Código Haskell

```
ndfa2dfa :: (Ord st, Eq sy) => Ndfa st sy -> Dfa [st] sy
ndfa2dfa ndfa@(Ndfa v q s z delta) = (Dfa v' q' s' z' delta')
  where tt = ndfa2ct ndfa
        v' = v
        q' = map fst tt
        s' = fst (head tt)
        z' = finalStatesDfa q' z
        delta' st sy = lookupCT st sy tt v

finalStatesDfa :: Eq st => [StDfa st] -> [st] -> [StDfa st]
finalStatesDfa [] z = []
finalStatesDfa (q:qs) z | (q 'intersect' z /= []) = q : finalStatesDfa qs z
                        | otherwise                = finalStatesDfa qs z

lookupCT :: (Eq st, Eq sy) => [st] -> sy -> CT st -> [sy] -> StDfa st
lookupCT st sy ct v = qs !! col
  where Just qs = lookup st ct
        Just col = elemIndex sy v
```

2.7 *Considere de novo o exercício 1.2-2 que define a linguagem dos números reais.*

1. *Modele a expressão regular, o autómato finito não determinista e o autómato determinista (obtido) em Haskell.*

Código Haskell

2. Prove que "153.148" $\in \mathcal{L}_{\text{reais}}$. Utilize o sistema Hugs para comparar a complexidade de cada uma das funções de aceitação implementados nas aulas teórico-práticas, i.e., as funções *matches'*, *dfaaccept* e *ndfaaccept*, para reconhecer essa frase. Qual o número de reduções efectuado pelo interpretador para cada uma das funções reconhecer a frase "153.148"? (o número de reduções do Hugs é activado fazendo `:set +s`)¹. Qual o número de reduções efectuado quando uma função é invocada duas vezes consecutivas com os mesmos argumentos? A que se deve essa diferença?

2.8 Considere a seguinte tabela de conversão de um autómato finito não determinista em determinista.

$Q \setminus T'$	'a'	'b'
{A}	{A, B, C}	{C}
{A, B, C}	{B, C}	{B, C}
{B, C}	{B, C}	{C}
{C}	{}	{}

Diga justificando se esta tabela está correctamente construída ou não.

¹Obtenha os resultados para cada uma das funções, em diferentes execuções do interpretador.