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

Exame de 1ª Época – 1ª Chamada

Data: 21 de Janeiro de 2004
Hora: 14:00

Dispõe de 2:30 horas para realizar este exame

1 Questão: Definição de ERs e Conversão para Autómatos

a) Considere um alfabeto definido por $\{a, b, c\}$. Escreva uma expressão regular que defina uma linguagem cujas frases podem ser sequências de 'a' ou de 'b', jamais misturadas, ou então sequências de 'c' desde que sempre emparelhados com 'a' (antes ou depois) e terminando num 'b'.

b) Considere o alfabeto $\{a, b, c\}$. Mostre que a expressão regular

$$r \;=\; c\,(a\;b^*\;+\;\epsilon)\;c$$

e a Gramática Regular GR abaixo

```
X  -->  c  Y
X  -->  c  W
Y  -->  a  Z
Z  -->  b  Z
    |   c
W  -->  c
```

definem exactamente a mesma linguagem.

c) Desenhe informalmente, ou directamente, o autómato determinista que define a mesma linguagem que a expressão regular r.

d) Considerando o alfabeto definido por $\{0, 1\}$, reescreva a seguinte expressão regular p, eliminando os operadores + (fecho transitivo) e ? (opcional):

$$p \;=\; 0\;+(0\;1?\;+\;1\;0)^+$$

e) Converta a ER p num autómato não-determinista, mostrando cada passo formal do processo de conversão.

2 Questão: ERs, Sistemas de Produção e Lex

Determinadas linguagens, ditas de *scripting*, suportam a utilização directa de expressões regulares.
Considere o seguinte programa Perl, onde se usam os parêntesis, '(' e ')', para agrupar sub-expressões que depois podem ser referenciadas por `$1`, `$2`, etc., onde o número indica a ordem pela qual são encontradas no texto que concorda com o padrão definido pela ER em causa.

```
#!/usr/bin/perl -w
while ( <STDIN> ) {
    $_ =~ s/<page n="([0-9]+)" id="([A-Za-z]+)">/Pag. $1, Cap. $2/g;
    print $_;     }
```

O mesmo mecanismo de *substituição de strings* é suportado pelo editor vi, no qual a mesma operação poderia ser efectuada com o comando:

```
:%s/<page n="\([0-9]+\)" id="\([A-Za-z]+\)">/Pag. \1, Cap. \2/g
```

Escreva um programa em Lex, equivalente ao programa Perl e ao comando vi apresentado.

3 Questão: Modelação com Autómatos

Desenhe um Autómato Determinista Reactivo (com acções associadas às transições) que modele o sistema que controla o acesso reservado a um parque automóvel.
O sistema começa por receber um sinal de despertar, quando detecta a aproximação de uma viatura; então pode receber um sinal interno de que está cheio e, nesse caso, acende a luz vermelha, não aceitando a identificação e permanecendo nesse estado até receber um sinal de saída de viatura (nessa altura passa a comporta-se como se indica a seguir, na situação em que há lugares). Se receber o sinal interno de que há lugares, acende a luz verde e aceita a entrada do identificador da viatura. A seguir recebe outro sinal interno de acesso permitido ou negado; no primeiro caso, abre a cancela e gera outro sinal interno de entrada de viatura, voltando ao estado inicial de adormecido; no segundo caso acende a luz vermelha e volta ao ponto de partida. Em qualquer outra situação, a recepção de um sinal de saída de viatura é aceite, mas não provoca mudança de estado

4 Questão: ANDs e ADs

Sobre autómatos e conversão entre eles, responda às seguintes alíneas:

a) Considere o autómato não-determinista, `ndfa04`, definido em Haskell como se segue:

```
ndfa04 = Ndfa ['a','b'] [0,1,2,3] [0] [0,1] delta04
 delta04 0 (Just 'a') = [1,2]
 delta04 1 (Just 'a') = [1,2]
 delta04 2 (Just 'b') = [1,3]
 delta04 3 (Just 'a') = [1,2]
 delta04 _   _        = []
```

Considere ainda que, do processo de conversão para determinista, resultou um autómato `dfa05`, calculado com base na seguinte tabela:

`dfa05` =

estado	'a'	'b'
0,1,2	1,2	1,3
1,2	1,2	1,3
1,3	1,2	1,3

1. Apresente uma frase que não pertença simultaneamente às linguagens definidas pelos dois autómatos `ndfa04` e `dfa05`.
2. Indique erros de cálculo encontrados na tabela.
3. Recalcule, então, correctamente o autómato determinista `dfa05`. Apresente a tabela de conversão e desenhe o autómato, indicando o estado inicial e os estados finais.

b) Escreva uma função Haskell `verifica()`, que recebe dois autómatos deterministas, $\mathcal{A}_1$ e $\mathcal{A}_2$, e uma frase $\mathcal{F}$ como parâmetros, e verifica se essa frase pertence à linguagem $\mathcal{L}(\mathcal{A}_1) \bigcap \mathcal{L}(\mathcal{A}_2)$.

5 Questão: escrita de uma gramática

Uma empresa recém-formada pretende fornecer serviços na área da investigação genealógica. Como uma das primeiras tarefas, há que desenvolver uma linguagem para o registo da informação que se vai colectando sobre um determinado indivíduo, ou seja, sobre a sua árvore genealógica.

Como os elementos da empresa não percebem muito de informática decidiram contratá-lo para desenvolver a dita linguagem e as ferramentas necessárias para trabalhar com ela.

Neste contexto, responda às seguintes alíneas:

a) Especifique, com uma gramática independente de contexto (GIC), a linguagem para a definição de árvores genealógicas que deverá obedecer aos seguintes requisitos:

1. Uma árvore genealógica é uma lista de indivíduos;
2. Cada indivíduo é representado por um registo de informação com os seguintes componentes: identificador, nome, data de nascimento, data de óbito (que é opcional), uma referência para o indivíduo do qual é filho (esta referência para o identificador do pai é opcional), uma segunda referência (também opcional) para o indivíduo que representa a sua mãe.

b) Apresente uma instância de uma árvore genealógica, utilizando a linguagem que definiu; mostre a respectiva *árvore de derivação* para nos convencer da correcção do exemplo apresentado.

c) Especifique, em C, a estrutura de dados que seria necessária para suportar com eficiência o cálculo dos seguintes invariantes:

1. O identificador de cada indivíduo é único, ou seja, não há dois indivíduos com o mesmo identificador.
2. Tem que existir um indivíduo com um identificador igual ao usado na referência de um pai ou de uma mãe, ou seja, todas as referências apontam para indivíduos que existem na árvore genealógica.

6 Questão: Parsing

Considere a seguinte função C, que generaliza o reconhecimento dos terminais e recorre a uma função `yylex()`, gerada pelo Lex.

```
#include "lex.yy.c"
(...)
int reconhece_terminal(int desejado, int proximopedaco) {
    if (desejado == proximopedaco)
       return (proximopedaco = yylex());
    else {
       fprintf(stderr, "Símbolo inesperado na linha %d: em vez de %d apareceu %d\n", yylineno, desejado, p
       exit(1); }
    }
```

A função `yylex()` poderia ter sido gerada a partir da seguinte especificação:

```
%{
#define INTEIRO 1
%}
 (...)
%%
[0-9]+  return(INTEIRO);
```

Faça as alterações necessárias para passar, do analisador léxico ao reconhecedor, o valor numérico das sequências de algarismos .

7 Questão: Gramáticas, Linguagens e Parsing

A listagem abaixo é um extracto da GIC (a operação de derivação está representada pelo símbolo ':') escrita no contexto de um projecto de *Data-Warehousing*, onde foi necessário definir uma linguagem para *selecção de fontes e dados*.
O Símbolo Inicial é `Linguagem`, os símbolos escritos em minúsculas são pseudo-terminais (classes terminais), as palavras-reservadas estão escritas em maiúsculas e os sinais estão entre apóstrofes.

```
Linguagem   :  Cabecalho  Seleccao '.'
Cabecalho   :  PROJ id ';' AUTOR str ';' VERSAO DE ':' data '.'
Seleccao    :  SelFnts SelEnts SelAtribs SelAmostras
SelFnts     :  SelFnt
            |  SelFnts ';' SelFnt
SelFnt      :  SelBD
            |  SelDoc
SelBD       :  BD '=' NomeBD User Alias
NomeBD      :  IdMaq ':' Path
Path        :  str
IdMaq       :  id
User        :  USER '=' IdUser
IdUser      :  id
Alias       :  ALIAS '=' id
SelDoc      :  DOC '=' idMaq ':' path Alias ;
..............
```

Responda, então, às alíneas seguintes:

a) Explique, por palavras suas, qual é a linguagem gerada por essa GIC.

b) A Gramática não é `LL(1)`, isto é, tem *lookaheads* em conflito ao reconhecer certos símbolos não-terminais com regras de derivação alternativas.
Identifique as produções conflituosas e transforme a gramática de modo a resolver essas situações.

c) Escreva, na linguagem `C`, as funções de um parser recursivo descendente para reconhecer o símbolo não-terminal `SelFnt`.

d) Escreva, na linguagem `Haskell`, a função de um parser recursivo descendente para reconhecer o símbolo não-terminal `NomeBD`.