Processamento de Linguagens – LEI

Exame de Recurso

10 de Julho de 2015 (16h00)

Dispõe de **2:00 horas** para realizar este teste.

Questão 1: Expressões Regulares e Autómatos (4v = 1+1+1+1)

Considere as seguintes ERs:

$$e1 \;=\; a\ b \;+\; (c\ d\ e^*)$$
$$e2 \;=\; a\ (b \;+\; c\ d)\ e^*$$
$$e3 \;=\; (a\ b \;+\; c\ d)\ e^+$$

Responda, então, às seguintes questões:

a) explique a linguagem gerada por $e2$ e, usando a respectiva *cadeia de derivação*, diga se a frase "`abee`"pertence a essa linguagem.

b) construa informalmente o Autómato Determinista equivalente a $e1$ e depois, aplicando as regras ensinadas nas aulas construa o Autómato Não-Determinista correspondente.

c) diga, justificando, se $e2$ e $e3$ são equivalentes.

d) Escreva uma expressão regular para definir o *caminho para um ficheiro num sistema de diretorias*. Seguem-se alguns exemplos de caminhos válidos: "/", "/dir1", "dirA/dirB/", "/dir/dir1/dirA/fich.ext", etc;

Questão 2: Filtros de Texto em Flex e GAWK (4v = 1+1+1+1)

Especifique filtros de texto com base em expressões regulares e regras de produção (padrão - ação) para resolver as seguintes alíneas:

a) Especifique em Flex um filtro que dado um texto de entrada contendo uma lista de palavras (sequências de letras podendo conter digitos) seguidas por uma sigla em maiúsculas e entre parêntesis curvos, separadas por virgulas, produz um texto de saída em que cada palavra aparece em 1 linha começada pela sigla seguida por ":"e depois por uma lista com todos os carateres da palavra dentro de parêntesis retos na forma minúscula e maiúscula. Todos os outros caracteres devem ser eliminados.
Exemplo: dado o seguinte texto de entrada

```
abril (ABR) , Janeiro (JAN), junHo (JUN) .
```

O resultado gerado na saída seria:

```
ABR: [aA][bB][rR][iI][lL]
JAN: [jJ][aA][nN][eE][iI][rR][oO]
JUN: [jJ][uU][nN][hH][oO]
```

O restante texto deverá ser copiado para a saída sem alterações.

b) Explique cuidadosamente o que faz a especificação Flex abaixo:

```
%{
#include <strings.h>
int c=1, ls=0, rs=0;
%}

%x CIT
%%
\\chapter\{[^}]+       { yytext=toupper(yytext+9); printf("%d. %s",c++,yytext); }
\\part\{[^}]+\}        { ; }
\\label\{              { ls++; ECHO; }
```

```
\\ref\{                 { rs++; ECHO; }
\\cite\{                { printf("["); BEGIN CIT; }
<CIT>[a-zA-Z0-9]+       { printf("%s",geraChaves(yytext)); }
<CIT>\}                 { printf("]"); BEGIN INITIAL; }
%%
int yywrap()
{ return(1); }

int main()
{ yylex();
   if (ls != rs) { printf("Aviso: ...\n"); }
  return 0; }
```

e, supondo que a função `'geraChaves()'` retorna sempre a string `"KK"`, concretize a sua explicação, indicando qual o texto que seria produzido à saída ao ler o seguinte texto de entrada:

```
\part{Abertura} \label{cI}
\chapter{Inicio}
Aqui esta um exemplo interessante!
\Section{sec.1} \label{sum}
 \enum
   + primeiro caso: 2+3-1
 \ee
\part{Fecho}
\chapter{Conclusao} \label{sdois}
conclui-se que um  \Chapter{Exemplo} faz falta, bem como uma nova \section{Final}.
Este capítulo \ref{cI} fala de ... e por isso \cite{arl1} ou \cite{PRHas}.
Para terminar em beleza \cite{aa/1eum}.
```

c) Suponha que tem um ficheiro correspondente ao dump em ASCII de uma tabela de uma base de dados (1 registo por linha) com 6 colunas separadas por ";"como se mostra a seguir:

```
elisa1;grego, troiano, historia;nada;qq coisa estranha;nada;PT
elisa2;camoes, historia, lusiadas;SIM;outro lixo;nada;PT
xico1;historia, poetas, camoes, pessoa;nada;bla-bla;PT;FR
```

Escreva um programa em GAWK para:

- inserir no início de cada linha da saída um novo campo que será um contador único;
- eliminar o sufixo numérico do 1ª campo;
- substituir todos os campos que sejam apenas "nada"por "NULL"
- eliminar a 4ª coluna

Por exemplo, a saída gerada para o ficheiro mostrado acima seria

```
1;elisa;grego, troiano, historia;NULL;NULL;PT
2;elisa;camoes, historia, lusiadas;SIM;NULL;PT
3;xico;historia, poetas, camoes, pessoa;NULL;PT;FR
```

d) Supondo que lhe é fornecido um ficheiro muito grande (com mais de 10000 linhas) com o formato seguinte por cada linha:

```
(Conceito,Relacao,Predicado)
```

desenvolva em GAWK um filtro que conte (e imprima no fim) as ocorrências de cada *conceito* distinto e faça uma lista de todas as *relações* mencionadas.

Questão 3: Desenho/especificação de uma Linguagem (4v)

Uma apresentação oral costuma ser apoiada por uma sequência de diapositivos (vulgo, slides). Além de darem um bom suporte à comunicação, os slides são muito simples de produzir porque têm uma forma sistemática. Cada slide tem uma cabeçalho, um rodapé e um corpo. No cabeçalho leva um título, eventualmente o subtítulo e no rodapé, que pode ser vazio, pode levar o número do slide, o nome do evento e a data, devendo dizer-se para cada um se deve ser colocado ao centro à esquerda ou à direita. Quanto ao corpo pode ser de 2 tipos: texto livre e nesse caso indica-se o texto, ou *itemized* e nesse caso

fornece-se a lista de items. A sequência de slides começa sempre por um slide de abertura que tem o título da comunicação e a lista de autores.

Sendo assim é possível criar uma linguagem muito simples que permita especificar os slides de uma apresentação. O que se lhe pede neste exercício é que escreva então uma Gramática Independente de Contexto, *GIC*, que especifique a Linguagem pretendida (note que o estilo da linguagem (mais ou menos verbosa) e o seu desenho são da sua responsabilidade).

Especifique em Flex um Analisador Léxico para reconhecer todos os símbolos terminais da sua linguagem e devolver os respetivos códigos.

Questão 4: Gramáticas e Parsing (5v)

Considere a gramática independente de contexto, *GIC*, abaixo apresentada, atendendo a que os símbolos terminais T e não-terminais NT são definidos antes do conjunto de produções P, sendo LP o seu axioma ou símbolo inicial.

```
T  = { id, f, k, pv, v, t }
NT = { LP, C, D }

p0:  LP  -->  C D f
p1:  C   -->  k id
p2:  D   -->  Dcl D1
p3:  D1  -->  &
p4:       |   pv Dcl D1
p5:  Dcl -->  t Lstid
p6:  Lstid --> id Co
p7:  Co  --> v Lstid
p8:       |  &
```

Neste contexto e após analisar a *GIC* dada, responda às alíneas seguintes.

a) Mostre que a frase `"k id t id id"` pertence à linguagem, construindo a respectiva Árvore de Derivação, e diga qual o conteúdo da Stack de Parsing de um parser LL(1) após reconhecer o prefixo `"k id"`.

b) Calcule o `lookahead()` das 9 produções.

c) Verifique se a *GIC* dada é LL(1), construindo a respetiva Tabela de Parsing, `tabLL(1)`.

d) Escreva as funções de um parser RD-puro (recursivo-descendente) para reconhecer os Símbolos D, $D1$ e Dcl.

e) Após estender a *GIC* dada, construa o estado inicial do autómato LR(0) e os estados que dele saem (identifique esses estado e mostre ainda as ações a partir de cada um deles).

Questão 5: Gramáticas e Tradução (2v)

A gramática independente de contexto, *GIC*, abaixo escrita em *BNF*, define uma linguagem de domínio específico para descrição de listas.
O Símbolo Inicial é `Listas`, os Símbolos Terminais são escritos só em minúsculas (terminais-variáveis) ou entre apostrofes (sinais-de-pontuação), e a string nula é denotada por `&`; os restantes (sempre começados por maiúsculas) serão os Símbolos Não-Terminais.

```
p0:  Listas      -->  Lst
p1:               |   Listas Lst
p2:  Lst         -->  '(' Args ')'
p3:  Args        -->  Arg
p4:               |   Args ',' Arg
p5:  Arg         -->  num
p6:               |   pal
```

Transforme a *GIC* dada numa **gramática tradutora**, *GT*, reconhecível pelo Yacc, para:

a) calcular e imprimir o número total de listas e o número de argumentos de cada lista e total.

b) garantir que se verificam as seguintes regras semânticas:

- se o primeiro argumento for um `id` a seguir devem aparecer 3 números;
- se o primeiro argumento for um `num` a seguir devem aparecer tantos argumentos quantos o valor desse número.

Questão 6: Compilação (1v)

Supondo que no seu programa em LPIS surge a seguinte atribuição

```
a = b + 3* c;
```

e assumindo que os endereços de `a, b, c` são respetivamente `0, 1, 2`
diga justificando qual dos fragmentos de código Assembly da VM (abaixo) seria gerado

```
   (a)          (b)          (c)
PUSHG 1      PSHA 0       PUSHG 1
PUSHI 3      PUSHG 1      PUSHI 3
PUSHG 2      PUSHI 3      ADD
MUL          PUSHG 2      PUSHG 2
ADD          MUL          MUL
STOREG 0     ADD          STOREG 0
             STOREN
```