Processamento de Linguagens – MiEI

Exame de Recurso

01 de Julho de 2016 (9h00)

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

Questão Eliminatória para os alunos sem TPC's

Nota: *esta questão só deve ser respondida pelos alunos que entregaram os 2 TPs e que nos TPCs entregaram menos de 4 trabalhos. Mas para esses* ***a pergunta é eliminatória!***

O parser RD para uma dada linguagem L é mostrado na listagem abaixo.
A partir desse parser escreva a respetiva gramática independente de contexto (GIC), explicando o raciocínio que seguiu.
Mostre como seria a função `main()` para ativar este parser, considerando que `L` é o axioma.

```
RecL(s)
  INICIO
    recTerm(i,s); recEs(s); recTerm(f,s)
  FIM
RecEs(s)
  INICIO
    recE(s); recC(s)
  FIM
RecE(s)
  INICIO
    SE (s==id) ENTAO recTerm(id,s); recTerm(a,s); recNs(s); recTerm(b,s)
    SENAO erro()
  FIM
RecNs(s)
  INICIO
    SE (s==b) ENTAO ;
    SENAO SE (s==n) ENTAO recTerm(n,s); recNs(s)
          SENAO erro()
  FIM
RecC(s)
  INICIO
    SE (s==f) ENTAO ;
    SENAO SE (s==v) ENTAO recTerm(v,s); recEs(s)
          SENAO erro()
  FIM
RecTerm(t,s)
  INICIO
    SE (s==t) ENTAO s = daSimb()
    SENAO erro()
  FIM
```

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

Responda às seguintes alíneas:

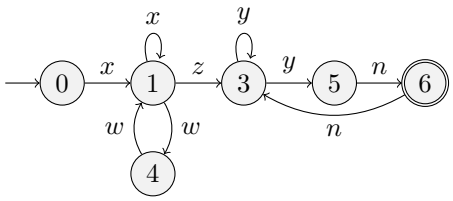
a) Considere as seguintes linguagens L1, e L2:
L1 é definida por:

```
S → a S b
  | c
```

L2 é definida por a^*cb^*.

Indique se L1 e L2 são equivalentes, se uma é um subconjunto da outra, ou se são simplesmente não equivalente (Em caso de diferenças, apresente uma frase que pertença apenas a uma delas).

b) Qual a expressão regular correspondente ao seguinte autómato:



c) Diga, justificando apropriadamente, se as expressões regulares abaixo, escritas em notação do Flex, são equivalentes:

```
(hoje|HOJE)
[hojeHOJE]+
```

d) Desenhe um autómato determinístico correspondente a: $(aab)^+c(d|abf)^*j$

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

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

a) Considere o seguinte ficheiro contendo notas de alunos:

```
Rui Silva:17
Serafim:18
Maria Rita:18
```

onde as notas são inteiros de 0 a 20. Faça uma script awk que calcule o número de ocorrências de cada nota, produzindo uma saída como a que se ilustra a seguir:

```
17   1
18   2
```

b) Escreva um filtro flex que, dada um programa yacc, escreva na saída a lista dos símbolos terminais da sua gramática – tanto os expressamente definidos como tokens (`%token ...`), como os terminais que aparecem protegidos nas produções da gramática (exemplo ','').

Questão 3: Desenho/especificação de uma Linguagem (4v=3+1)

Pretende-se uma linguagem de Domínio Específico que permita descrever o calendário de exames da época especial. Para tal deve-se indicar, para cada dia do período de exames, a data e a lista de UCs a alocar nesse dia. Para cada UC e além da sua sigla, tem de se indicar o respetivo nome e ano do curso, o turmo em que fica (manhã ou tarde) e o número de inscritos, este último opcional. Além disso a linguagem deve permitir que se faça o calendário de vários cursos (identificados por uma sigla e escola a que pertence).

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 Top-Down (4v=1+2+1)

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 Tr o seu axioma ou símbolo inicial.

```
T  = { '.', ';', id, i1, i2 }
NT = { P, Ds, Is, As, I, R }

p1: P  -> Ds Is '.'
p2: Ds -> &
p3:     | id id As
p4: As -> ';' id id As
p5:     | &
p6: Is -> I R
p7: I  -> i1
p8:     | i2
p9: R  -> &
p10:    | ';' Is
```

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

a) Verifique se a frase `ttt av i1 ; i2 .` pertence à linguagem, construindo a respectiva Árvore de Derivação.

b) Construa a Tabela de Parsing LL(1) para mostrar que a gramática é LL(1).
Indique os respetivos *first()*, *follow()* e *lookahead()*.

c) Escreva as funções de um parser RD-puro (recursivo-descendente) para reconhecer o Símbolo `Is` e todos os que dele derivam direta ou indiretamente.

Questão 5: Gramáticas, Tradução e Parsing Bottom-Up (4v=1+1+2)

A gramática independente de contexto, *GIC*, abaixo escrita em *BNF*, define o conceito de atribuição numa tradicional linguagem de programação imperativa.
O Símbolo Inicial é `Lista`.

```
T  = { '=', '+', '*', id, num }
NT = { A, E, T, F }

p1:  A      -->  id '=' E
p2:  E      -->  T
p3:           |  E '+' T
p4:  T      -->  F
p5:           |  T '*' F
p6:  F      -->  num
p7:           |  id
```

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

a) Mostre que esta *GIC* não é LL(1)

b) Após estender a *GIC* dada, construa o respetivo autómato LR(0).

c) Suponha que o analisador léxico passa o valor de cada terminal relevante através da variável global `yylval` e que os identificadores usáveis estão armazenados numa Tabela de Identificadores à qual tem acesso usando as funções `endr(x)` que retorna o endereço da variável `x` e `exist(x)` que verifica se a variável `x` existe na tabela.
Nesse contexto:

c1) Junte à *GIC* dada as ações semânticas necessárias para gerar código Assembly da VM que realize a atribuição.

c2) Junte à *GIC* dada as ações semânticas necessárias para sinalizar erro se for usado um identificador não-declarado.