Processamento de Linguagens – MiEI

Teste

03 de Junho de 2016 (8h00)

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 tem nota inferior a 4 pontos (em 6). Mas para esses* ***a pergunta é eliminatória!***

Explique com clareza o algoritmo de Parsing LL(1) listado abaixo e diga qual a sua principal diferença relativamente ao parser RD:

```
ParserLL1()
  INICIO
    push(Sigma,$); push(Sigma,S);
    t = getSimb();
    accao = er;
    REPETIR
      X = top(Sigma); pop(Sigma)
      SE ((X==$) E (t=$)) ENTAO accao = ac;
      SENAO SE (X in T) ENTAO SE (t==X) ENTAO t = getSimb();
                              SENAO accao = er;
            SENAO accao = TabLL1( X,t );
                  SE (accao!=er) ENTAO PushProd(Sigma,RHS(p));
    ATE ((accao==ac) OU (accao==er))
  FIM
```

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

Responda às seguintes alíneas:

a) O comando `sed`, disponível el Linux, permite (entre outras coisas) fazer substituições de uma expressão regular, `expReg`, por uma `string` em todas as ocorrências desse padrão numa dado ficheiro de texto `file`, sendo usado do seguinte modo

```
sed -e 's/expReg/string/g' file
```

Construa, então, um comando `sed` que usando uma única substituição retire as etiquetas html de um ficheiro (deixando o resto inalterado).

b) Considere a seguinte gramática:

```
A → a b A
  | B c
B → B d
  | d
```

Escreva uma expressão regular equivalente.

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

```
(0[1-9]|1[0-2])
[0-1][0-9]
```

d) Desenhe um autómato determinístico correspondente a $a(abcd|abf|gh^{+}i)^{*}j$

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) Temos um ficheiro contendo milhares de linhas e pretendemos, usando Gawk, substituir cada linha que já tenha aparecido pelo número da linha do original. Exemplo:

```
Joaquim                          Joaquim
João da Cunha                    João da Cunha
Manuel Fagundes                  Manuel Fagundes
João da Cunha            →       =2
Manuel Ribeiro                   Manuel Ribeiro
João da Cunha                    =2
```

Relembra-se que em AWK, NR dá o número do registo corrente.

b) Escreva um filtro `flex` que, dada um programa yacc, escreva na saída apenas a gramática nele definida.

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

Considere uma situação em que se pretende descrever através de uma linguagem apropriada um álbum fotográfico. O álbum é constituído por um título (ou tema), um período de tempo, um proprietário e uma sequência de páginas. Em cada página há várias fotografias de tamanho pequeno, normal ou grande. Cada fotografia é identificada por um título (que pode ser vazio), uma descrição de quem está na foto, o local onde foi tirada e a data (sendo estes dois últimos elementos opcionais). As páginas podem ser agrupadas em sub-álbuns, cada um subordinados a um tema.

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)

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  = { t, ft, id, pe, pd, n }
NT = { Tr, As, A, Ns, Rt }

p1:  Tr  -->  t As ft
p2:  As  -->  A Rt
p3:  A   -->  id pe Ns pd
p4:  Ns  -->
p5:       |   n Ns
p6:  Rt  -->  As
p7:       |
```

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

a) Partindo dos `first()` e `follow()` de cada símbolo, calcule o `lookahead()` das 7 produções.

b) Construa a Tabela de Parsing LL(1) que indica para cada símbolo NT (5 linhas) e para cada símbolo T (6+1 colunas) qual a produção a usar para continuar o reconhecimento (ou terminar com erro, se o terminal não for aceite nesse momento).
Mostre que a gramática é LL(1).

c) Suponha que num dado momento do parsing LL(1) a sua *stack de parsing* (ou *stack de objetivos*) contém os símbolos (da base para o topo): \$, ft, As.
Diga o que é que isso significa e indique quais são os únicos terminais válidos que podem ser aceites nesse momento.

d) Escreva a função de um parser RD-puro (recursivo-descendente) para reconhecer o Símbolo Ns.

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

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

```
p1:  Lista      -->  Elems '.'
p2:  Elems      -->  Elem
p3:                | Elems ',' Elem
p4:  Elem       -->  num
```

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

a) Construa a árvore de derivação que prova que a sequência de terminais "$12, 3, 5.$" é uma frase válida da linguagem.

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

c) Transforme a *GIC* dada numa **gramática tradutora**, *GT*, reconhecível pelo Yacc, para gerar código Assembly da VM que calcule e imprima a soma dos elementos da lista. Suponha para isso que o analisador léxico passa o valor de cada número no campo `yylval.valn`