Processamento de Linguagens – MiEI

Teste

05 de Julho de 2017 (9h00)

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

Questão 1: Expressões Regulares e Autómatos (4v)

Responda às seguintes alíneas:

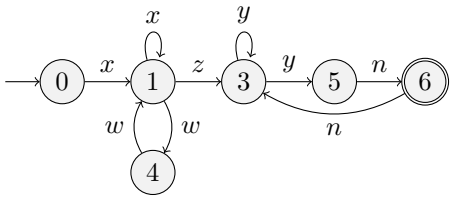
a) Considere as seguintes linguagens L1, e L2:
L1 é definida por:

```
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:

b) Escreva um filtro usando o Flex para ler um texto anotado em XML e transferir esse texto para a saída capitalizando todos os nomes de elementos (tags) usados nas marcas.
Além disso, se as marcas de abertura contiverem atributos depois do nome do elemento, esses pares *atributo-valor (AV)* devem ser retirados ficando só mesmo o elemento, pretendendo-se que no fim indique quantas dessas situações ocorreram e então faça uma listagem dos pares *(AV)* afetos a cada elemento.

Questão 3: Desenho/especificação de uma Linguagem (3v=2+1)

Pretende-se uma linguagem de Domínio Específico que permita descrever as tarefas urgentes de cada funcionário de uma secretaria.

Para tal deve-se indicar, no início, a lista de funcionários, indicando para cada um o nome completo e o respetivo código, bem como a função. Depois então surge a lista de tarefas agrupadas por funcionário (agora já só identificado pelo respetivo código). Por cada tarefa indique o dia, hora, a prioridade (normal, urgente ou baixa), e a descrição da tarefa.

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 (3v=1+1+1)

Considere a gramática independente de contexto, G, abaixo apresentada, que gera frases com igual número de 'a' ou de 'b' por qualquer ordem, atendendo a que os símbolos terminais T e não-terminais NT são definidos antes do conjunto de produções P, sendo Z o seu axioma ou símbolo inicial.

```
T  = { '.', a, b }
NT = { Z, S }

p0: Z  -> S '.'
p1: S  -> a S b S
p2:     | b S a S
p3:     | &
```

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

a) Gere uma frase válida da linguagem definida por G de comprimento 4, 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 os 2 símbolos não-terminais.

Questão 5: Gramáticas, Tradução e Parsing Bottom-Up (6v=.5+1+.5+4)

Considere a gramática independente de contexto, G, abaixo apresentada, que permite declarar uma ou mais variáveis definindo o seu tipo e permite executar instruções de dois tipos sobre essas variáveis. Note ainda que os símbolos terminais T e não-terminais NT estão definidos antes do conjunto de produções P, sendo S o seu axioma (ou símbolo inicial).

```
T  = { '{', '}', ';',  id, opA, opB }
NT = { S, Ds, Is, As, I, Tip, Var }
P = {
p1: S  -> Ds '{' Is '}'
p2: Ds -> &
p3:     | Tip Var As
p4: As -> ';' Tip Var As
p5:     | &
p6: Is -> I
p7:     | Is  ';' I
p8: I  -> opA Var
p9:     | opB Var Var
p10: Tip -> id
p11: Var -> id
}
```

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

a) Verifique se a frase `Ta varA; Tb varB { opA varA ; opB varB }` **pertence à linguagem**, construindo a respectiva Árvore de Derivação.

b) Após estender a *GIC* dada, construa o respetivo **autómato LR(0)** e identifique todas as **situações de conflito** que eventualmente ocorram.

c) Reescreva a gramática (basta mostrar as produções que modifica ou acrescenta) de modo a eliminar a recursividade à direita e ter só recursividade à esquerda e também para obrigar todas as instruções a terminar com ';'

d) Usando notação do Yacc (e todas as facilidades oferecidas pelo par de ferramenta Lex/Yacc) transforme a *GIC* dada numa **gramática tradutora** (*GT*) (juntando-lhe ações semânticas) para:

- **d1)** contar o número de instruções de cada tipo.
- **d2)** sinalizar erro se for usado nas instruções uma variável não-declarada.
- **d3)** gerar código da VM para reservar memória para as variáveis declaradas.
- **d4)** gerar código da VM para implementar a instrução 'opA' supondo que se trata de ler um inteiro para a varável 'Var'.