Processamento/Teoria de Linguagens e Compilação
LCC (3ºano) + MEFis (1ºano)

Exame de recurso
31 de Janeiro de 2024 (09h00)

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

Questão 1: Expressões Regulares (5v = 1+1+1+1+1)

Responda a cada uma das alíneas seguintes:

a) Considere as Expressões Regulares $e1$ e $e2$ abaixo e explique, com uma justificação clara, porque é que, apesar de não o parecerem, as ER são equivalentes.

$$e1 \;=\; (\; a^{+} \mid c^{+} \;)\; b\; d^{*}\; (\; d \mid b\; a \;)$$

$$e2 \;=\; (\; c^{+} \mid a^{+} \;)\; (\; b\; d^{*}\; b\; a \mid b\; d^{+} \;)$$

b) Mostre que a frase *ccbd* deriva de $e2$ e apresente justificando uma frase que não pertence à linguagem definida por $e1$.

c) Construa intuitivamente um autómato determinista correspondente a $e3$.

$$e3 \;=\; (b\; a \mid b\; c)^{+}\; (c\; d \mid c\; b)\; d^{*}$$

d) Considere que um Sistema Operativo reconhece e executa os seguintes comandos para gerir ficheiros e diretorias (pastas):

```
'apaga' nomeFich NL
'muda' NL
'mostra'  NL
'move'   NL
```

sendo que o símbolo `NL` representa o *Fim-de-linha* (NewLine) e `nomeFich` é uma palavra formada por letras maiúsculas ou minúsculas, podendo conter algarismos no meio ou no fim.
Desenhe um só **autómato determinista** que reconheça todos os comandos—o autómato pedido só tem um *estado inicial* e terá um *estado final* por cada comando.

e) Uma Junta de Freguesia decidiu identificar os seus fregueses usando um código único formado pelo respetivo nome, código postal e coordenadas geográficas. O resultado desta padronização de identidade pode ser observado nos exemplos seguintes:

```
RangelHenriques.Pedro@4715-012;41,55;-8,45
Silva.Ana.Maria@4715-012;41,55;-9,00
Araujo.?@4715-767;42,05;-9,55
Mota.Carmo@4780-767;40,05;-8,55
```

Depois de observar com atenção os exemplos acima, escreva uma Expressão Regular que especifique o padrão que essas frases seguiram.

Questão 2: módulo re (5v = 1+1+1+1+1)

Recordando o que aprendeu sobre o uso de Expressões Regulares em Python com recurso ao módulo '`re`', resolva os exercícios abaixo:

a) Considere a frase

```
frase = "Ola aaa OLA123ola e tuOla--ola."
```

e as seguintes instruções de um programa Python que importa o módulo 're'

```
res = re.findall( r'[^ ]+?(?i:ola)', frase )
print( len(res) )
```

e responda às alíneas seguintes:

a1) Diga, justificando claramente, qual o valor impresso pelo programa acima quando o mesmo é executado.

a2) Explique o que aconteceria se substituísse a instrução `re.findall` pela instrução `re.search` e depois executasse o programa.

b) Considere o seguinte extrato de um filtro de texto em *Python*, no qual uma expressão regular é usada para decifrar *uma mensagem codificada*[1].

```
import re
codigoSecreto = "4T4UG5H281E60X45L4MQ1T9P25A089M66L1E801D"
mensagem = re.findall(f'[13579](.)[2468]', codigoSecreto)
print("".join(mensagem))
```

e responda às alíneas seguintes:

b1) Apresente o resultado produzido pelo programa.

b2) Apresente um possível código secreto a submeter ao mesmo programa que contenha a mensagem "PLC" escondida.

c) Considere o seguinte extrato de um filtro de texto em Python

```
import re
text  = " --- "
s = re.search(r'\[([aeiou]+|[1-9]+)\]', text)
if ( s ):
  print(s.group())
else:
  print("Falhou")
```

e diga justificando se a resposta produzida pelo programa seria `(1 marca)` caso o texto de entrada fosse

```
text = "LINHA COM ((1) marca) ([de]) SUCESSO[(13)] [a] ou (2)"
```

Questão 3: Gramáticas (3v)

ERDL é uma DSL que serve para descrever modelos de dados do tipo diagramas Entidade-Relação. Para isso, ERDL permite declarar as Entidades do sistema, indicando apenas o seu nome ou definindo ainda o seu conjunto de atributos. Depois podem relacionar-se as Entidades aos pares; por cada relação indica-se se é um para um, um para muitos, ou muitos para muitos. Abaixo dá-se um exemplo de uma frase válida em ERDL.

```
SISTEMA exemplo
ENTIDADES:
     e1, E2, E3 { atr1 : tipo1; atr2: tipo2; atr3: tipo1 }
RELACOES:
   E1 (1-N) e2;  e1 (N-M) E3
   .
```

Neste exercício pede-se que: Escreva, em BNF-puro, uma Gramática Independente de Contexto (GIC) para definir formalmente a linguagem ERDL acima descrita.

[1] Note que o método `sep.join(L)` devolve a string que se obtem concatenando todos os elementos da lista `L` com o separador `sep`.

Questão 4: Compilador (7v = 2+1+1+1+2)

A gramática independente de contexto, G, abaixo escrita em *BNF*, define uma linguagem de domínio específico para descrição de uma *coleção de tuplos de facetas*.
O Símbolo Inicial é `Colecao`. Os Símbolos Terminais '`pal`' e '`string`' representam, respetivamente, *identificadores* (formados só por letras) e `textos livres` (sequências de quaisquer carateres entre aspas). Os demais Símbolos Terminais estão escritos entre apóstrofes e representam os *sinais-de-pontuação* da linguagem. Os restantes símbolos (sempre começados por maiúsculas) serão os Símbolos Não-Terminais.

```
p0:  Colecao     -->  Tuples  '.'
p9:  Tuples      -->  Tuple
p1:                |  Tuples Tuple
p2:  Tuple       -->  TId '(' Faces ')'
p3:  Faces       -->  Faceta
p4:                |  Faces ',' Faceta
p5:  Faceta      -->  Field '=' Value
p6:  Field       -->  pal
p7:  Value       -->  string
p8:  TId         -->  pal
```

Neste contexto e após analisar G, responda às alíneas seguintes:

a) Escreva uma *frase válida* da linguagem gerada por G, apresentando a respetiva *árvore de derivação*.

b) G é recursiva à esquerda e por isso **não é LL(1)**! Identifique as produções de G em que o **Conflito LL(1)** ocorre e para uma dessas situações explique com clareza em que consiste o dito conflito.

c) Transforme G numa gramática equivalente mas recursiva à direita e sem **Conflitos LL(1)**.

d) Escreva em Python usando o módulo '`ply.lex`' um analisador léxico para reconhecer as frases da linguagem definida por G.

e) Escreva em Python usando o módulo '`ply.yacc`' um analisador sintático para reconhecer as frases da linguagem definida por G.

Acrescente Ações Semânticas às produções da gramática para associar ao valor semântico do Axioma (`p[0]` da produção $p0$) a lista com o nome de todas as *facetas* encontradas e para calcular na variável de classe '`parser.tot`' o número total de *tuplos* reconhecidos.