Processamento de Linguagens I
LESI + LMCC (3º ano)

Exame de 1ª Época – 2ª Chamada

Data: 05 de Julho de 2005
Hora: 09:30

Dispõe de 2:30 horas para realizar este exame

A nova linguagem de programação, chamada `lpT`, foi criada com o propósito específico de agilizar o processamento de texto. Para esse efeito, `lpT` tem basicamente variáveis de 2 tipos: caracteres e strings. Mas como é óbvio permite também manipular inteiros (constantes e variáveis), ainda que num contexto mais restrito. Para facilitar a vida ao programador, as variáveis não precisam de ser declaradas, sendo a declaração implícita na sua primeira utilização; o seu tipo será então aí inferido.
A linguagem possui um conjunto grande de operadores especializados no processamento de caracteres e strings, permitindo ainda fazer algumas operações elementares com inteiros. Além disso, admite a definição de funções e, claro, o uso da recursividade; mas sendo uma linguagem de cariz imperativo (ou procedimental), `lpT` tem várias estruturas de controlo do fluxo de execução, condicionais e cíclicas.
A gramática independente de contexto G, abaixo apresentada, é uma parte de $\mathcal{G}_{lpT}$ e define algumas das instruções disponíveis. O Símbolo Inicial é `lpT`, os Símbolos Terminais são escritos em minúsculas (pseudo-terminais), ou em maiúscula (palavras-reservadas), ou entre apostrofes (sinais de pontuação) e a string nula é denotada por `&`; os restantes serão os Símbolos Não-Terminais.

```
p1:  lpT      -->  IdPrg  Progr  Data  Insts
p2:  Insts    -->  Inst
p3:              |  '{'  Inst  LstI  '}'
p4:  LstI     -->  &
p5:              |  ';'  Inst  LstI
p6:  Inst     -->  Leitura
p7:              |  CtrlCond
p8:              |  IdVar  '='  OpersStr
p9:  Leitura  -->  LERCH  IdVar
p10:             |  LERSTR IdVar
p11: CtrlCond -->  CASO  IdVar  SEJA  LstCasos  FCASO
p12: LstCasos -->  &
p13:             |  Caso  LstCasos
p14: Caso     -->  char  ':'  Insts
p15: OpersStr -->  Concat
p16:             |  InsHead
p17: Concat   -->  IdVar  '+'  IdVar
p18: InsHead  -->  char '@'  IdVar
p19: IdPrg    -->  id
p20: Data     -->  str
p21: Progr    -->  str
p22: IdVar    -->  id
```

Questão 1: parsing

Recordando os seus conhecimentos sobre análise sintáctica *Top-Down* e *Bottom-Up*, responda às alíneas seguintes:

a) Mostre que G **é LL(1)**.

b) Explique o raciocínio que se segue para transformar um parser Recursivo-Descendente Genérico guiado por uma Tabela de Decisão em um parser Iterativo LL(1) (se o ajudar a responder, escreva os respectivos algoritmos).

c) Escreva a função, pertencente a um parser Recursivo-Descende Puro, para reconhecer o símbolo Não-Terminal `Inst`.

d) Observando G, é evidente que a GIC dada **não é LR(0)**!
Diga o que nos permite tirar de imediato esta conclusão, explicando a sua resposta.

e) Construa completamente **apenas** o `estado 0` do autómato determinista LR(0) e os `estados` que dele se atingem (em relação a estes faça o `fecho` e indique as `transições`, mas não continue a construção a partir dos novos estado).

f) Supondo que vai analisar a frase

```
PrgExemplo  "PRH"  "2005-06-18"
pal = 'a' @ pal
```

com um parser *Bottom-Up LR*, desenhe a respectiva *Árvore de Parsing* indicando a ordem de redução dos nodos.

Questão 2: TDS e gramática tradutora

Supondo que vai gerar código Assembly para a MSP (resumo das instruções em anexo) escreva as produções duma **gramática tradutora**, GT, (reconhecível pelo yacc) para traduzir as 2 *instruções de leitura*. Em relação à leitura de uma string, assuma que: a leitura termina com o *fim-de-linha* (caracter com o código ASCII 13); e que os caracter ficam todos armazenados sequencialmente a partir de um endereço inicial que está guardado na posição de memória correspondente à variável (do tipo string) a ser lida.
Mostre esquematicamente a forma como iria traduzir a *instrução de controlo condicional CASO*.

Questão 3: TDSem e gramática de atributos

Depois de transformar G numa gramática independente de contexto abstracta (pode reduzir algumas produções que lhe pareçam supérfluas), escreva uma **Gramática de Atributos**, GA, para construir a **Tabela de Identificadores** com todas as variáveis declaradas implicitamente num determinado texto fonte, associando a cada uma o respectivo tipo (caracter, string, ou inteiro).
A sua GA, além das *regras para o cálculo dos atributos necessários* e as *regras de tradução* para apresentar o resultado desejado (gravar num ficheiro a dita Tabela de Identificadores), deve incluir *condições de contexto* para garantir que não haja inconsistências de tipo no caso de variáveis já encontradas na Tabela de Identificadores (o actual contexto de uso sugere um tipo diferente daquele que foi inferido nas utilizações anteriores).
Para facilitar a leitura da sua resposta, reúna numa tabela (no início ou no fim) os **atributos herdados e sintetizados, ou intrínsecos** de cada símbolo (NT ou T) de GA.

Questão 5: sobre o Trabalho Prático 1

Supondo que lhe pedem para fazer uma apresentação a toda a turma sobre o problema que o seu grupo resolveu a nível do 1º TP, esquematize aqui o que diria nessa apresentação. Refira o que é dado, o que é produzido e como está organizado e funciona o seu processador para atingir o resultado em causa.

Questão 6: sobre o Trabalho Prático 2

Relativamente ao 2º TP, diga se o seu grupo optou por gerar Assembly (de uma Máquina Virtual), ou se pelo contrário gerou logo Código-Máquina. Discuta as implicações de uma decisão ou de outra.

Instruções Assembly da Máquina de Stack Virtual MSP

Grupo 1: Manuseamento de Valores e Endereços

```
PUSH  valor  ;coloca o valor inteiro no Topo(Stack)
PSHA  ender  ;coloca o endereço no Topo(Stack)
LOAD
LDA          ;carrega um endereço no Topo(Stack)
STORE
STRA         ;armazena na memória o endereço do Topo(Stack)
IN
INC          ;lê um caracter
OUT
OUTC         ;escreve um caracter
```

Grupo 2: Operações Aritméticas e Lógicas

```
ADD          ;adiciona dois inteiros
SUB
MUL
DIV
ADDA         ;adiciona um inteiro a um endereço
AND
OR
NOT
EQ           ;igual
NE           ;diferente
LT
LE
GT
GE
```

Grupo 3: Instruções de Controlo

```
JUMP  label
JMPF  label  ;salta para a ´label´ se Topo(Stack) for Falso
CALL  label
RET
NOOP         ;não faz nada
HALT         ;termina a execução
```