Universidade do Minho
Mestrado em Engenharia Informática
Engenharia de Linguagens
Engenharia Gramatical - Grupo 1

Resolução das Fichas 3 e 5

Ano Lectivo de 2012/2013

pg22820 - **António Silva**
pg22781 - **Rui Brito**

18 de Novembro de 2012

Conteúdo

1 Introdução

Este primeiro trabalho de avaliação *Engenharia Gramatical* da Unidade Curricular de Especialização *Engenharia de Linguagens*, consiste na realização das fichas 3 e 5 disponiblizadas no Blackboard. Apresentamos neste documento a resolução das mesmas.

2 Ficha 3

2.1 Composição do Corpo

Para escrever uma gramática tradutora, foi necessário completar a informação sobre o corpo. Assim, incialmente, o corpo da factura era um conjunto de linhas em que cada linha era *'(' codartigo ',' designacao ',' pvu ',' quantidade ')'*. Depois para suportar o pedido da alínea c), cada linha passou a ser somente *'(' codartigo ',' quantidade ')'*.

2.2 Código Java

Para conseguirmos saber os totais dos produtos, com várias facturas (sendo que cada factura possuia um id alfanumérico), foi criado um *HashMap* para associar a cada id de factura uma lista de valores, que era o total de cada linha da factura. No final, é possível apresentar o total de cada linha em cada factura e ainda o total de cada factura (que mais não é do que a soma dos totais das linhas). Para se obter o Preço Unitário, que a pedido da alínea c) deveria já ter sido indicado no ínicio, foi também criada um *HashMap* com a correspondência entre o código do produto e os seus atributos (guardados numa classe). Assim, por cada linha só era necessário obter o PVU através do código do artigo, e multiplicá-lo pela quantidade.

2.3 Exemplo de Input

```
a1 "xpto" 3.6 50
a2 "outro" 1 60.5
a3 "mais um" 4.99 4
---

f1
"Nome 1" "NIF 1" "Morada 1" "NIB 1"
"Nome 2" "NIF 2" "Morada 2"
(a1,5) (a3,2)
;

f2
"Nome 3" "NIF 3" "Morada 3" "NIB 3"
"Nome 4" "NIF 4" "Morada 4"
(a2,9.5) (a1,5) (a3,2)
;

f3
"Nome 5" "NIF 5" "Morada 5" "NIB 5"
```

```
"Nome 6" "NIF 6" "Morada 5"
(a2,2.25)
.
```

2.4 Gramatica Tradutora

Segue abaixo a Gramática Tradutora usado para a resolução do problema:

```
grammar ficha3;

options{
language=Java;
k=1;
}

@header{
import java.util.HashMap;
import java.util.Map;

}

@members{
class Artigos{
public String codigo;
public String designacao;
public Double preco;
public Double stock;
public Artigos(String c, String d, Double p, Double s)
            {this.codigo = c; this.designacao = d; this.preco = p; this.stock = s;}
}

private HashMap<String,ArrayList<Double>> _totais;
private HashMap<String,Artigos> _artigos;
private String _fct;
}

facturas
@init{
    this._totais = new HashMap<String, ArrayList<Double>>();
    this._artigos = new HashMap<String,Artigos>();
}
@after{
    int i;
    Double soma;
    for (Map.Entry<String, ArrayList<Double>> entry : this._totais.entrySet()) {
     System.out.println("Factura " + entry.getKey());
     i = 0;
     soma = 0.0;
```

```
     for(Double d : entry.getValue()){
     i++;
     soma += d;
     System.out.println("\tLinha " + i + ": " + d.toString());
     }
     System.out.println("Total factura: " + soma);
     System.out.println("----------------------");
    }
    System.out.println("");
    System.out.println("==============================");
    System.out.println("");
    for (Artigos a : this._artigos.values()){
     System.out.println("Artigo " + a.codigo);
     System.out.println("\tDesignacao: " + a.designacao);
     System.out.println("\tPreço: " + a.preco);
     System.out.println("\tStock: " + a.stock);
     System.out.println("----------------------");
    }
}
: descs+ '---' factura (';' factura)* '.'
;

descs
:  codartigo designacao pvu stock {
String c = $codartigo.text;
String d = $designacao.text;
Double p = Double.parseDouble($pvu.text);
Double s = Double.parseDouble($stock.text);
this._artigos.put(c, new Artigos(c, d, p, s));
}
;

factura
: cabec corpo
;

cabec
: idfact {
            this._fct = $idfact.text;
            this._totais.put(this._fct, new ArrayList<Double>());
            } ide idr
;

corpo
: linha+
;
```

```
linha
: '(' codartigo ',' quantidade ')'  {
Double qt = Double.parseDouble($quantidade.text);
Artigos ag = this._artigos.get($codartigo.text);
this._totais.get(this._fct).add(ag.preco * qt);
ag.stock -= qt;
}
;

codartigo
: ID;

designacao
: STR;

pvu : FLOAT;

stock
: FLOAT;

quantidade
: FLOAT;


idfact
: numfact
;

numfact
: ID
;


ide : nome nif morada nib
;
nome: STR;
nif : STR;
morada
: STR;
nib : STR;
idr : nome nif morada
;


FLOAT
    :   ('0'..'9')+ '.' ('0'..'9')* EXPONENT?
```

```
    |   '.' ('0'..'9')+ EXPONENT?
    |   ('0'..'9')+ EXPONENT?
    ;

fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;


WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) {$channel=HIDDEN;}
    ;

STR
    :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
    ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

3 Ficha 5

3.1 Exemplo de frase válida e Árvore de Derivação

Um exemplo rudimentar de frase válida seria:

```
[ REGISTO id1122 LIVRO "nome_livro" ("nome_autor") "nome_editora" 2010 BGUM
EXISTENCIAS LOCAL "Braga" ( cb54433231 ; PERMANENTE ) ]
```

A árvore de derivação corresponde seria como se segue:

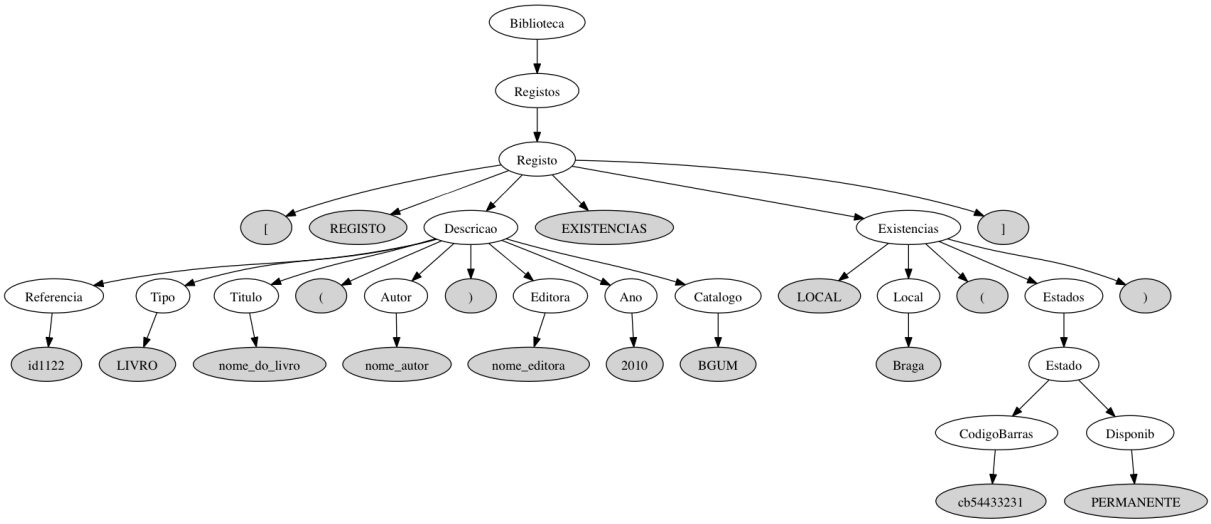

Figura 1: Árvore de Derivação

Imagem em maior detalho na figura 4

3.2 Lista de Autores

Para permitir uma lista de autores ao contrário de apenas um, basta extender a produção dez. Ficaria, então, algo como segue abaixo:

```
p10: Autores --> Autor
p11:           | Autores ',' Autor
```

3.3 Recursividade à esquerda para Recursividade à direita

Recursividade à esquerda:

```
p1: Registos --> Registo
p2:            | Registos ',' Registo
```

O que transformado para recursividade à direita (assumindo a notação eBNF):

```
p1: Registos --> Registo
p2:            | Registo ',' Registos
```

As diferenças quando à árvore de derivação são apresentadas a seguir:

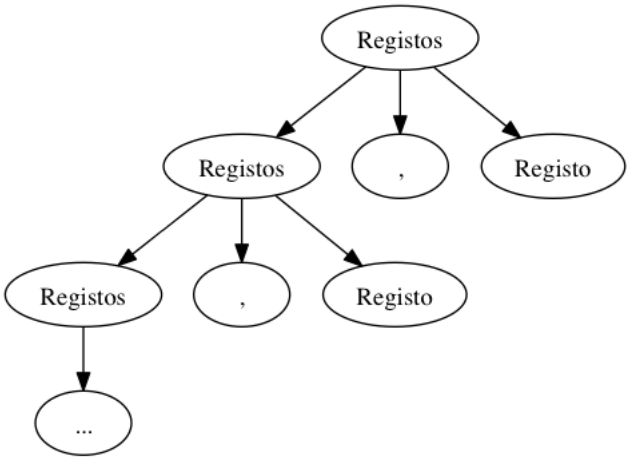

Figura 2: Recursivo à esquerda

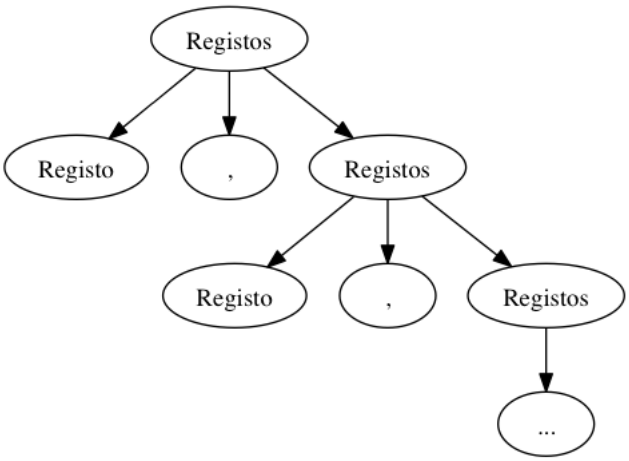

Figura 3: Recursivo à direita

4 Gramática Tradutora

Os pontos pedidos foram os seguintes:

1. calcular e imprimir: o número de registos; e o número de livros existentes para cada registo.
2. o número total de livros com estado RESERVADO/PERMNENTE/ESTANTE.
3. identificar e listar por ordem alfabética os títulos dos livros.
4. verificar se não existem registos com a mesma referência.

Para resolver o ponto 1, simplesmente foram criados contadores e os mesmo incrementados nos pontos apropriados, como se pode ver abaixo.
A resolução do ponto 2 não difere muito da do ponto 1, mais uma vez, foi feita com contadores.
Para o ponto 3, criou-se uma variável booleana que é posta a *true* sempre que o literal 'LIVRO' é encontrado, posteriormente o valor é inserido num *TreeSet*, estrutura esta que garante a ordenação do output.

Quanto ao ponto 4, mais uma vez tiramos partido das propriedades da estrutura *TreeSet* sendo que esta não permite elementos repetidos, no caso da inserção falhar, o valor de referência actual é impresso para o ecrã.
Abaixo pode ser visto o código integral.

```
grammar ficha5;

options{
k=2;
language=Java;
}

@header{
import java.util.TreeSet;
}

@members{
int numRegistos = 0;
int numLivros = 0;
String refReg = null;

int reservados = 0;
int permanentes = 0;
int estante = 0;

Boolean isBook = false;

TreeSet<String> bookNames = new TreeSet<String>();
TreeSet<String> refs = new TreeSet<String>();
}

biblioteca
@after{
System.out.println ("Livros Reservados: " + reservados);
System.out.println ("Livros Permanentes: " + permanentes);
System.out.println ("Livros em Estante: " + estante);

System.out.println("=== Livros ===");
for (String s : bookNames) {
System.out.println(s);
}

}
: registos
;

registos
: registo{numRegistos++;} (',' registo{numRegistos++;})*
;

registo
@after {
System.out.println(refReg + ": " + numLivros);
numLivros = 0;
```

```
refReg = null;
}
: '[ REGISTO ' descricao ' EXISTENCIAS ' existencias ']'
;

descricao
: referencia {refReg = $referencia.text;
           if(!refs.add($referencia.text)){
                    System.out.println("Referencia ja existente: " + $referencia.text);}
                    }
                    tipo titulo {if(isBook) {
                          bookNames.add($titulo.text);isBook = false;}
                          } '(' autores ')' editora ano catalogo
;

autores
: autor (',' autor)*
;

referencia
: ID
;

tipo: 'LIVRO' {numLivros++; isBook = true;}
| 'CDROM'
| 'OUTRO'
;

titulo
: STRING
;

autor
: STRING
;

editora
: STRING
;

ano :  INT
;
catalogo
: 'BGUM'
| 'ALFA'
| 'OUTRO'
;

existencias
: 'LOCAL ' local '(' estados ')'
;

local
```

```
: STRING
;

estados
: estado (',' estado)*
;

estado
: codigoBarras disponib
;

codigoBarras
: ID
;
disponib
: 'ESTANTE' {estante++;}
| 'PERMANENTE'{permanentes++;}
| 'EMPRESTADO' {reservados++;}dataDev
;

dataDev
: ano '-' mes '-' dia
;
mes : INT;
dia : INT;

ID  : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
    ;

INT : '0'..'9'+
    ;

FLOAT
    :   ('0'..'9')+ '.' ('0'..'9')* EXPONENT?
    |   '.' ('0'..'9')+ EXPONENT?
    |   ('0'..'9')+ EXPONENT
    ;

WS  :   ( ' '
        | '\t'
        | '\r'
        | '\n'
        ) {$channel=HIDDEN;}
    ;

STRING
    :  '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
    ;

fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment
```

```
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   UNICODE_ESC
    |   OCTAL_ESC
    ;

fragment
OCTAL_ESC
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UNICODE_ESC
    :   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

A Imagens

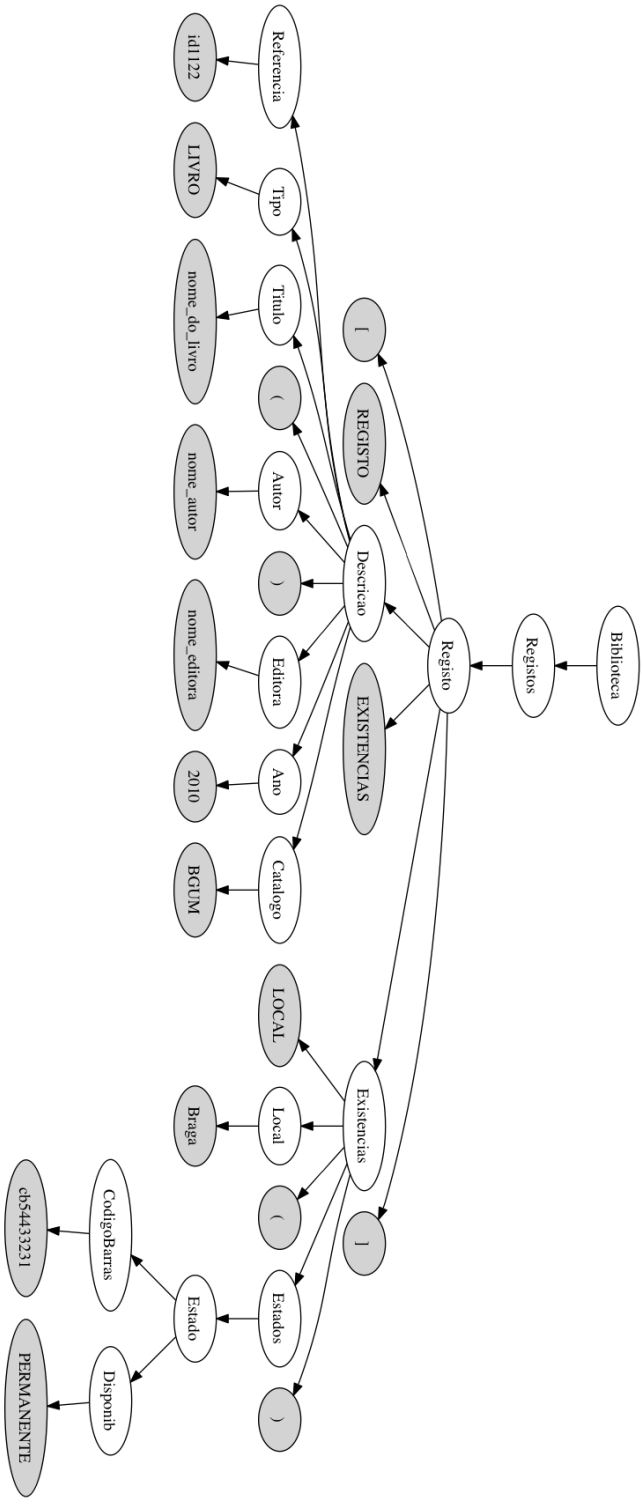

Figura 4: Árvore de Derivação