Exercício para Avaliação n.º 3

Bruno Azevedo*and Miguel Costa†

Módulo Engenharia Gramatical,
UCE30 Engenharia de Linguagens,
Mestrado em Engenharia Informatica,
Universidade do Minho

13 de Março de 2012

Resumo

Este documento apresenta as resoluções dos Exercícios Práticos n.º 3 e n.º 4 do módulo de Engenharia Gramatical. O exercício está relacionado com a geração automática de Processadores de Linguagens a partir de Gramáticas.

Para o exercício n.º 3 era pretendido utilizar a gramática `Genea` j´a utilizada nas aulas para calcular algumas estatísticas relacionadas com ela. Para o exercício 4 o objectivo era criar uma linguagem para fazer movimentar um `Robo` num Terreno e depois criar um processador para as frases da linguagem com algumas funcionalidades.

*Email: `azevedo.252@gmail.com`
†Email: `miguelpintodacosta@gmail.com`

Conteúdo

1 Ambiente de Trabalho **3**

2 Exercício n.º 3 - Genea **3**
2.1 Descrição do problema . . . 3
2.2 Resolução do Problema . . . 4
2.3 Resultado Final . . . 6
2.4 Gramática Final . . . 6

3 Exercício n.º 4 - Robo **11**
3.1 Descrição do problema . . . 11
3.2 Criação da linguagem . . . 11
3.3 Implementação . . . 14
3.4 Decisões Tomadas . . . 14
3.4.1 Classes . . . 14
3.5 Gramática Final . . . 15
3.6 Resultado Final . . . 17

4 Conclusões **21**

5 Anexos **22**
5.1 Classes em Java . . . 22
5.1.1 Robo.java . . . 22
5.1.2 Terreno.java . . . 35
5.1.3 Movimento.java . . . 38
5.1.4 Matrix.java . . . 40

1 Ambiente de Trabalho

Foi necessário usar um Gerador de Compiladores para gerar o nosso próprio compilador, por isso usámos o AnTLR que é também usado nas aulas. Para facilitar o processo de debugging durante a resolução do problema dado, usámos a ferramenta AnTLRWorks, que tem uma interface bastante agradável e simpática para ajudar a resolver problemas desta natureza.

A linguagem de programação adoptada foi o JAVA. De forma a tornar a nossa solução mais legível e estruturada, criámos classes com o auxílio do IDE NetBeans que nos ajudou no desenvolvimento do código JAVA e ainda na criação da sua documentação (javadoc).

2 Exercício n.º 3 - Genea

2.1 Descrição do problema

O pretendido para este exercício era calcular algumas estatísticas a partir de uma frase válida para a linguagem do `Genea`, os cálculos efectuados foram:

- Total de Famílias
- Total de Progenitores
- Total de Filhos
- Total de filhos de cada família
- Média de filhos por família

Além das estatísticas é verificado ainda se as datas são válidas, verificámos se:

- a data de casamento é posterior à data de nascimento do casal;
- a data de morte, nascimento e casamento são datas e não uma string que possa representar outra coisa.

Relembrando uma frase válida para o Genea:

```
PROGENITORES ( 28-02-1988 )
    PAI Antonio, Costa 09-03-1961
    MAE Maria, Costa 21-07-1962
    FILHOS
        Miguel 28-03-1990,
        Pedro 06-04-1992,
        Cristina 02-01-1997
```

que tem como árvore de derivação:

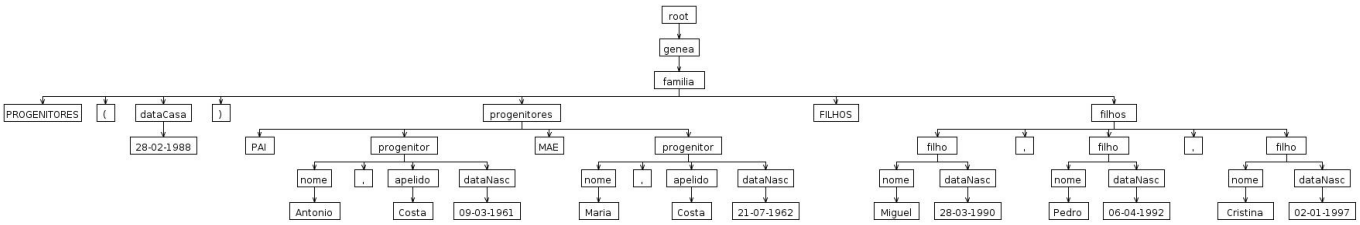

Figura 1: Árvore de derivação de uma frase da Gramática Genea

2.2 Resolução do Problema

Para calcular e verificar o que era pretendido, criámos variáveis globais (na secção members). Como variáveis temos então:

- `int total_progenitores;` // variável que conta o total de progenitores
- `int total_filhos;` // variável que conta o total de filhos.
- `Integer fil_temp;` // variável temporário que contêm o total de filhos de uma família, quando começa uma nova família é coloca novamente a 0.
- `Integer total_familias;` // conta o total de famílias existentes na frase.
- `Integer media_filhos;` // indica o número médio de filhos que as famílias tem.
- `ArrayList<Integer> filhos = new ArrayList<Integer>();` // Array que contêm o número de filhos que cada família tem.

Para o tratamento das datas foi necessário ter as variáveis:

- `GregorianCalendar dataCasa_tmp;` // variável que tem a data de casamento dos progenitores.
- `GregorianCalendar dataNasc1_tmp;` // data de nascimento de um dos progenitores.
- `GregorianCalendar dataNasc2_tmp;` // data de nascimento de um dos progenitores
- `boolean vez = false;` // para controloar se a data a ser lida é a de casamento ou de nascimento.

Para além das variáveis, tivemos ainda de criar algumas funções para verificar as datas:

Listing 1: Funções para as datas

```
/**
* Verifica se uma data é válida.
*/
public String verificaData(String data){

        try{
            String[] valores;
            String delimiter = "-";
            valores = data.split(delimiter);

            Integer a = Integer.parseInt(valores[2]);
            if(a< 1000 || a > 2100){
                return "Ano Invalido";
            }

            Integer m = Integer.parseInt(valores[1]);
            if(m < 1 || m > 12){
                return "Mes Invalido";
            }

            Integer d = Integer.parseInt(valores[0]);
            if(d<1 || d > 31){
                return "Dia invalido";
            }
        } catch(Exception e){
            System.out.println("Erro ao validar data.");
        }
```

```
28          return "";
29      }
30
31      /**
32      * Dada uma string extrai o ano
33      */
34      public Integer getAno(String data){
35          String[] valores;
36          String delimiter = "-";
37          valores = data.split(delimiter);
38          return Integer.parseInt(valores[2]);
39      }
40
41      /**
42      * Dada uma string extrai o mes
43      */
44      public Integer getMes(String data){
45          String[] valores;
46          String delimiter = "-";
47          valores = data.split(delimiter);
48          return Integer.parseInt(valores[1]);
49      }
50
51      /**
52      * Dada uma string extrai o dia
53      */
54      public Integer getDia(String data){
55          String[] valores;
56          String delimiter = "-";
57          valores = data.split(delimiter);
58          return Integer.parseInt(valores[0]);
59      }
```

Depois de criadas todas as funções e variáveis necessárias tivemos de adicionar nas produções as regras de cálculo.

- `int total_progenitores;`
  Para calcular o total de progenitores tivemos de adicionar na produção:
  `progenitor : nome ',' apelido dataNasc dataMorte? ;`
  a instrução:
  `total_progenitores++;`

- `int total_filhos;`
  No cálculo do total de filhos, o comportamento é semelhante ao total de progenitores, mas neste caso, adicionámos na secção init da produção:
  `filho : nome dataNasc dataMorte? ;`
  a instrução:
  `total_filhos++;`

- `Integer total_familias;`
  O total de famílias funciona da mesma forma que as anteriores, em que na secção da produção:
  `familia : 'PROGENITORES' '(' dataCasa ')' progenitores 'FILHOS' filhos? ;`
  foi adicionada a instrução:
  `total_familias++;`

- `ArrayList<Integer> filhos = new ArrayList<Integer>();`
  O array que contém os filhos de cada família, utiliza a variável temporária `Integer fil_temp;`, que conta

os filhos de uma família e é colocada novamente a 0 quando é encontrada uma nova família, para no final (secção after) da produção `filhos : filho (',' filho)*;` ser adicionado mais um índice ao array com o valor correspondente ao número de filhos da família.

- `Integer media_filhos;`
  A média de filhos é calculada no final de toda a frase ter sido reconhecida, ou seja, na secção after da produção `genea : familia+;`. A instrução usada apenas pega no total de filhos e divide pelo total de famílias que já estão calculados (`media_filhos = total_filhos/total_familias;`).

Relativamente às datas, a verificação é feita na produção de cada uma das datas, por exemplo, na produção `dataCasa : DATA ;` foram adicionadas as instruções:
`String data = verificaData(DATA.text); System.out.println(data);` em que a string data guarda algum erro caso não seja válida.

Para verificar se a data de casamento é posterior à data de nascimento dos progenitores, na produção da `familia` são colocadas as instruções para guardar na variável temporária `dataCasa_tmp` a data de casamento do casal:
`String d = dataCasa.text;dataCasa_tmp = new GregorianCalendar(getAno(d), getMes(d), getDia(d));`
As datas de nascimento dos progenitores são criadas da mesma forma que a data do casamento, mas de forma a ir para variáveis diferentes o `PAI` e a `MAE`, é usada a flag `vez`.

Depois de termos as datas todas, no final da produção `familia` é verificado se as datas estão correctas:

```
if(dataNasc1_tmp.after(dataCasa_tmp) || dataNasc2_tmp.after(dataCasa_tmp)){
    System.out.println("Data de Casamento inválida.");
}else {
    System.out.println("Data de casamento válida.");
}
```

2.3 Resultado Final

Depois de correr a gramática em que o input é a frase dada como exemplo anterioremente, obtemos como output:

```
Data de casamento válida.
Total de familias: 1
Total de progenitores: 2
Média de filhos por familia: 3
Filhos da familia 1: 3
```

2.4 Gramática Final

No final de termos a nossa Gramática defenida e com todas as instruções necessários para atingir os objectivos, ficamos com:

Listing 2: Linaguagem Final

```
1  grammar genea;
2
3  @header{
4      import java.util.ArrayList;
5      import java.util.GregorianCalendar;
6  }
7
8  @members{
9      private int total_progenitores = 0;
10     private int total_filhos = 0;
```

```
11      private Integer fil_temp = 0;
12      private Integer total_familias = 0;
13      private Integer media_filhos = 0;
14      private ArrayList<Integer> filhos = new ArrayList<Integer>();
15      // para verificar se as dastas estao correctas
16      private GregorianCalendar dataCasa_tmp;
17      private GregorianCalendar dataNasc1_tmp;
18      private GregorianCalendar dataNasc2_tmp;
19      private boolean vez = false; // para ver se vai para a segunda data ou ainda para
            a primeira
20
21      public String verificaData(String data){
22          try{
23              String[] valores;
24              String delimiter = "-";
25              valores = data.split(delimiter);
26              Integer a = Integer.parseInt(valores[2]);
27              if(a< 1000 || a > 2100){
28                  return "Ano Invalido";
29              }
30              Integer m = Integer.parseInt(valores[1]);
31              if(m < 1 || m > 12){
32                  return "Mes Invalido";
33              }
34              Integer d = Integer.parseInt(valores[0]);
35              if(d<1 || d > 31){
36                  return "Dia invalido";
37              }
38          } catch(Exception e){
39              System.out.println("Erro ao validar data.");
40          }
41          return "";
42      }
43
44      public Integer getAno(String data){
45          String[] valores;
46          String delimiter = "-";
47          valores = data.split(delimiter);
48          return Integer.parseInt(valores[2]);
49      }
50
51      public Integer getMes(String data){
52          String[] valores;
53          String delimiter = "-";
54          valores = data.split(delimiter);
55          return Integer.parseInt(valores[1]);
56      }
57
58      public Integer getDia(String data){
59          String[] valores;
60          String delimiter = "-";
61          valores = data.split(delimiter);
62          return Integer.parseInt(valores[0]);
63      }
64
65      }
66
67  genea
68  @init   {
69      total_progenitores = 0;
70      total_familias = 0;
```

```
71      total_filhos = 0;
72  }
73  @after  {
74      System.out.println("Total de familias: "+total_familias);
75      System.out.println("Total de progenitores: "+total_progenitores);
76      media_filhos = total_filhos/total_familias;
77      System.out.println("Media de filhos por familia: "+media_filhos);
78      int i = 1;
79      for(Integer n : filhos){
80          System.out.println("Filhos da familia "+i +": "+n);
81          i++;
82      }
83  }
84      : familia+
85      ;
86
87  familia
88  @init   {
89      total_familias++;
90  }
91  @after  {
92      if(dataNasc1_tmp.after(dataCasa_tmp) || dataNasc2_tmp.after(dataCasa_tmp)){
93          System.out.println("Data de Casamento invalida.");
94      }else {
95          System.out.println("Data de casamento valida.");
96      }
97  }
98      : 'PROGENITORES' '(' dataCasa ')'   {
99                          String d = $dataCasa.text;
100                         dataCasa_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
101                         vez = false;
102                         }
103      progenitores 'FILHOS' filhos?
104     ;
105
106 progenitores
107     : 'PAI' progenitor 'MAE' progenitor
108     | 'MAE' progenitor 'PAI' progenitor
109     ;
110
111 progenitor
112     : nome ',' apelido dataNasc     {
113                     total_progenitores++;
114                     if(!vez){
115                         String d = $dataNasc.text;
116                         dataNasc1_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
117                         vez = true;
118                     }else{
119                         String d = $dataNasc.text;
120                         dataNasc2_tmp = new GregorianCalendar(getAno(d), getMes(d),
                                getDia(d));
121                     }
122                     }
123
124     dataMorte?
125     ;
126
127 filhos
128 @init   {
```

```
129     fil_temp = 0;
130 }
131 @after  {
132     filhos.add(fil_temp);
133 }
134     : filho (',' filho)*
135     ;
136
137 filho
138 @init   {
139     fil_temp++;
140     total_filhos++;
141 }
142     : nome dataNasc dataMorte?
143     ;
144
145 nome    : ID
146     ;
147
148 apelido
149     : ID
150     ;
151
152 dataCasa
153     : DATA  {
154         String data = verificaData($DATA.text);
155         System.out.println(data);
156         }
157     ;
158
159 dataNasc
160     : DATA {
161         String valido = verificaData($DATA.text);
162         System.out.println(valido);
163         }
164     ;
165
166 dataMorte
167     : DATA {
168         String valido = verificaData($DATA.text);
169         System.out.println(valido);
170         }
171     ;
172
173
174 DATA    : INT '-' INT '-' INT
175     ;
176
177 ID  :   ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
178     ;
179
180 INT :   '0'..'9'+
181     ;
182
183 COMMENT
184     :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
185     |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
186     ;
187
188 WS  :   ( ' '
189         | '\t'
```

```
190         | '\r'
191         | '\n'
192         ) {$channel=HIDDEN;}
193     ;
```

3 Exercício n.º 4 - Robo

3.1 Descrição do problema

Imaginemos um robo com a função de aspirar um terreno de forma retangular. Este terreno tem uma área que é conhecida pelo robo e que acaba por limitar o raio de ação dele.

O robo pode ter definida uma posição inicial e os seus movimentos podem ser em quatro direções diferentes (norte, sul, este e oeste) com um peso associado que representa a distância que se vai deslocar (por exemplo `NORTE 4`, desloca-se 4 unidades para norte). Tem ainda a opção de estar ligado ou desligado que define se está ativo ou não para aspirar.

Com base na descrição do robo, era pedido:

1. Criar uma linguagem que conseguisse descrever uma rotina possível para o robo. Esta linguagem deve permitir ainda que tenha no início certas definições como a dimensão do terreno e a posição inicial do robo.
2. Depois de definida a linguagem, tínhamos de criar um processador para as possíveis frases que podiam ser geradas com as seguintes funcionalidades:
   - Verificar que o robo não se movimenta para fora da área de limpeza.
   - Calcular a distância (em cm) que o robo percorreu durante a sua rotina.
   - Determinar quantas mudanças de direção foram feitas pelo robo.
   - Determinar a distância média que o robo se desloca por cada movimento.

3.2 Criação da linguagem

Analisando o que era pretendido para descrever a rotina do robo, tentámos criar uma linguagem com uma sintaxe de fácil leitura e sem ambiguidades. Depois de analisar várias alternativas, definimos a linguagem com a seguinte estrutura:

Listing 3: Estrutura da gramática

```
ASPIRADOR
{
DEFINICOES
{
definicao1; definicao2;
}
MOVIMENTOS
instrucao1;
instrucao2;
....
}
```

Uma linguagem tem de ter símbolos terminais e neste caso definimos os símbolos:

- DIM
- POS
- LIGAR
- DESLIGAR
- NORTE
- SUL

- ESTE
- OESTE
- ID
- INT

Definindo formalmente a gramática para representar os eventos possíveis do robo, obtemos:

Listing 4: Gramática

```
grammar robot;

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

robot
@init {
    terreno = new Terreno();
    robo = new Robo(terreno);
}
@after {
    System.out.println(terreno.toString());
    System.out.println(robo.toString());
    System.out.println(robo.toStringEstatisticas());

    Matrix m = new Matrix(robo, terreno);
    m.setVisible(true);
}
    : 'ASPIRADOR' '{' corpo '}'
    ;

corpo
    : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
    ;

definicoes
    : '{' dimensao (posicao)? '}'
    | '{' (posicao)? dimensao '}'
    ;
dimensao
    :DIM '=' '(' INT ',' INT ')' ';'
    ;
posicao
    :POS '=' '(' INT ',' INT ')' ';'
    ;

movimentos
    : movimento (movimento)*
    ;

movimento
    : LIGAR ';'
    | DESLIGAR ';'
    | NORTE INT ';'
    | SUL INT ';'
    | ESTE INT ';'
    | OESTE INT ';'
    ;

/*------------------------------------------------------------------
```

```
52  * LEXER RULES
53  *------------------------------------------------------------------*/
54
55  DIM      : ('d'|'D')('i'|'I')('m'|'M');
56  POS      : ('p'|'P')('o'|'O')('s'|'S');
57
58  LIGAR : ('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
59  DESLIGAR :  ('d'|'D')('e'|'E')('s'|'S')('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
60
61  NORTE    : ('n'|'N')('o'|'O')('r'|'R')('t'|'T')('e'|'E');
62  SUL      : ('s'|'S')('u'|'U')('l'|'L');
63  ESTE     : ('e'|'E')('s'|'S')('t'|'T')('e'|'E');
64  OESTE    : ('o'|'O')('e'|'E')('s'|'S')('t'|'T')('e'|'E');
65
66  ID   :   ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
67       ;
68
69  INT  :   '0'..'9'+
70       ;
71
72  COMMENT
73       :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
74       |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
75       ;
76
77  WS   :   ( ' '
78           | '\t'
79           | '\r'
80           | '\n'
81           ) {$channel=HIDDEN;}
82       ;
```

Depois de gerada a gramática, uma frase que se pode gerar é:

Listing 5: Frase gerada 1

```
ASPIRADOR
{
    DEFINICOES
    {
        dim = (100 , 150) ; pos = (0 , 0) ;
    }
    MOVIMENTOS
        LIGAR;
        NORTE 2 ;
        DESLIGAR ;
        SUL 10;
    }
```

Para provar que era uma frase válida, fizemos a sua árvore de derivação:

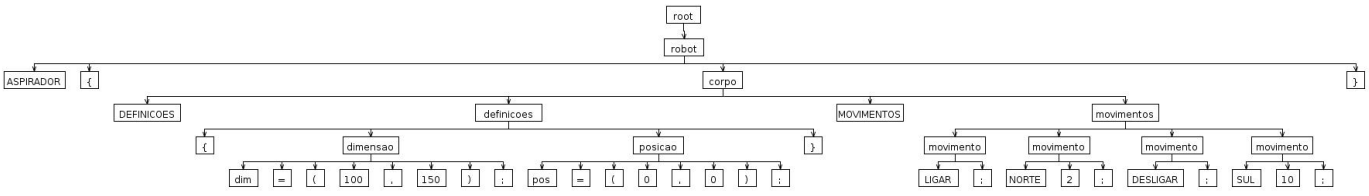

Figura 2: Árvore de derivação

Analisando a árvore gerada, verificámos que o elemento raiz é `robot` e o parser terá de encontrar, no início, a palavra `ASPIRADOR` seguida de um `corpo` que se encontra dentro de chavetas.

O corpo está dividio em 2 partes: `definicoes` e `movimentos`. Nas `definicoes` podemos configurar a `dimensao` do terreno e ainda a `posicao` inicial do robo.

Quanto aos `movimentos`, este podem ser de 2 tipos, os que fazem realmente movimentar o robo (por exemplo `NORTE 2`) e os que ligam (`LIGAR`) ou desligam (`DESLIGAR`) o robo.

3.3 Implementação

De forma a estruturar melhor todo o exercício, criámos classes em java que nos facilitassem o cálculo de todas as estatísticas e todas as restrições que eram necessárias.

3.4 Decisões Tomadas

Como seria de esperar, há pormenores que tinham de ser decididos para colocar o robo no terreno e para o cálculo das estatísticas, algumas decisões tomadas foram:

- Caso não esteja definida a posição inicial do Robo no terreno, é assumido que esta é (0,0), que corresponde ao canto superior esquerdo do terreno.
- Inicialmente, o Robo é colocado no terreno sem direção, assim, apenas depois do primeiro movimento, ele tem a direção definida e é possível contar para efeitos estatísticos a mudança de direção.
- Apenas quando o Robo está no modo ligado é que ele se movimenta, caso contrário ignora todas as instruções que receber, excepto a de `LIGAR`.

3.4.1 Classes

As classes criadas foram:

- `Robo`
- `Terreno`
- `Movimento`
- `Matrix`

A classe `Robo` é a responsável por guardar o estado, a posicao atual, a direção atual, todos os movimentos executados pelo robo e por gerar as estatísticas relacionadas com os mesmos. Esta classe contém 4 `ArrayList<Integer>` para guardar inteiros com o valor que foi deslocado em cada uma das direções possíveis e, ainda, um `TreeMap<Integer,Movimento>` em que a `key` corresponde ao número em que o `Movimento` ocorreu, este `value` é do tipo `Movimento` que contém apenas 3 variáveis de instância:

- `Integer num` - número em que o movimento ocorreu.
- `Direcao direcao` - direção em que o movimento foi feito.
- `Integer distancia` - a distancia percorrida nesse movimento.

Este `TreeMap<Integer,Movimento>` é usado apenas para na animação sabermos a ordem em que os movimentos foram feitos e que tipo de deslocação foi feita pelo robo, enquanto que as estatísticas são todas calculadas a partir dos `ArrayList<Integer>` para ser mais eficiente e não termos que estar sempre a percorrer a estrutura em árvore.

`Terreno` é a classe que contém o valor, em cm, de uma unidade de movimento, as dimensões do terreno onde o robo se vai movimentar e verifica se o robo não se quer deslocar para fora dele. Para confirmar visualmente que tudo o que era pedido ao Robo se concretizava, criámos uma interface onde é possível ver a deslocação, passo a passo, do Robo e ainda as estatísticas geradas. Esta interface corresponde à classe `Matrix` que recorre ao Java SWING para criar a animação.

Em anexo está o código java de cada classe.

3.5 Gramática Final

Depois de criadas as classes em Java, foi necessário adaptar a nossa gramática de forma a realizar o que era pretendido, e instanciámos as três classes `Robo`, `Terreno` e (`Matrix`).

Resultando em:

Listing 6: Gramática Final do Robo

```
grammar robot;

options {
     language = Java;
}

@header{
    import Robot.Robo;
    import Robot.Terreno;
    import Robot.Matrix;
}

@members{
    private Robo robo;
    private Terreno terreno;
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

robot
@init {
    terreno = new Terreno(); // instancia o terreno
    robo = new Robo(terreno); // instancia o robo
}
@after {
    System.out.println(terreno.toString()); //Imprime o valor da unidade (em cm), a
        largura e altura do terreno
    System.out.println(robo.toString()); //Imprime a posicao inicial e a posicao
        final, o estado final, a direcao final, os movimentos executados por direcao,
         o numero de vezes que mudou de direcao, toda a sequencia de movimentos
        executada e o total de movimentos executados
    System.out.println(robo.toStringEstatisticas()); // Imprime, por direcao, o
        numero de deslocacoes realizadas, a distancia percorrida (em cm) e a
        distancia media percorrida por movimentacao. Imprime tambem o numero total de
         deslocacoes, a distancia total percorrida, a distancia media percorrida por
        movimentacao e o numero total de mudancas de direcao

    Matrix m = new Matrix(robo, terreno); // instancia a matrix
    m.setVisible(true);
}
    : 'ASPIRADOR' '{' corpo '}'
    ;

```

```
38 corpo
39     : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
40     ;
41
42 definicoes
43     : '{' dimensao (posicao)? '}'
44     | '{' (posicao)? dimensao '}'
45     ;
46 dimensao
47     :DIM '=' '('    x=INT{terreno.setLarg(Integer.parseInt(x.getText()));} ',' // define a largura do terreno
48          y=INT{terreno.setAlt(Integer.parseInt(y.getText()));} // define a altura do terreno
49        ')' ';'
50     ;
51 posicao
52     :POS '=' '('x=INT { if (terreno.validaPosX(Integer.parseInt(x.getText()))){ robo.setPosX(x.getText()); robo.setPosXini(x.getText());} // se a posicao inicial do robo no eixo X for valida (ou seja, esta dentro dos limites do terreno) entao define a posicao inicial e atual do robo nesse eixo
53                 else System.out.println("Posicao inicial invalida.");
54               }
55        ','
56           y=INT { if (terreno.validaPosY(Integer.parseInt(y.getText()))) { robo.setPosY(y.getText()); robo.setPosYini(y.getText());} // se a posicao inicial do robo no eixo Y for valida entao define a posicao inicial e atual do robo nesse eixo
57                 else System.out.println("Posicao inicial invalida.");
58               }
59        ')' ';'
60     ;
61
62 movimentos
63     : movimento (movimento)*
64     ;
65
66 movimento
67     : LIGAR ';'     {robo.setEstado("LIGADO");} // define o estado do robo como Ligado
68     | DESLIGAR ';'      {robo.setEstado("DESLIGADO");} // define o estado do robo como Desligado
69     | NORTE INT ';'     { if (terreno.validaPosY(robo.getPosY() - Integer.parseInt($INT.text))) {robo.movNorte(Integer.parseInt($INT.text));} // se a posicao final for valida, entao movimenta o robo para essa posicao
70                 else {System.out.println("Movimento NORTE "+ $INT.text +" invalido por ultrapassar os limites da area de limpeza!");}
71               }
72     | SUL INT ';'       {if (terreno.validaPosY(robo.getPosY() + Integer.parseInt($INT.text))) {robo.movSul(Integer.parseInt($INT.text));} // se a posicao final for valida, entao movimenta o robo para essa posicao
73                 else {System.out.println("Movimento SUL "+ $INT.text +" invalido por ultrapassar os limites da area de limpeza!");}
74               }
75     | ESTE INT ';'      { if (terreno.validaPosX(robo.getPosX() + Integer.parseInt($INT.text))) {robo.movEste(Integer.parseInt($INT.text));} // se a posicao final for valida, entao movimenta o robo para essa posicao
76                 else {System.out.println("Movimento ESTE "+ $INT.text +" invalido por ultrapassar os limites da area de limpeza!");}
77               }
78     | OESTE INT ';'     { if (terreno.validaPosX(robo.getPosX() - Integer.parseInt($INT.text))) {robo.movOeste(Integer.parseInt($INT.text));} // se a posicao
```

```
        final for valida, entao movimenta o robo para essa posicao
79                  else {System.out.println("Movimento OESTE "+ $INT.text +" invalido
                        por ultrapassar os limites da area de limpeza!");}
80                }
81    ;
82
83 /*------------------------------------------------------------------
84  * LEXER RULES
85  *------------------------------------------------------------------*/
86
87 DIM      : ('d'|'D')('i'|'I')('m'|'M');
88 POS      : ('p'|'P')('o'|'O')('s'|'S');
89
90 LIGAR : ('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
91 DESLIGAR :  ('d'|'D')('e'|'E')('s'|'S')('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
92
93 NORTE    : ('n'|'N')('o'|'O')('r'|'R')('t'|'T')('e'|'E');
94 SUL      : ('s'|'S')('u'|'U')('l'|'L');
95 ESTE     : ('e'|'E')('s'|'S')('t'|'T')('e'|'E');
96 OESTE    : ('o'|'O')('e'|'E')('s'|'S')('t'|'T')('e'|'E');
97
98 ID   :   ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
99     ;
100
101 INT :   '0'..'9'+
102     ;
103
104 COMMENT
105     :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
106     |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
107     ;
108
109 WS  :   ( ' '
110         | '\t'
111         | '\r'
112         | '\n'
113         ) {$channel=HIDDEN;}
114     ;
```

3.6 Resultado Final

Depois de criada a linguagem, se testarmos com o input:

```
ASPIRADOR
{
    DEFINICOES
    {
    dim = (15 , 15) ; pos = (7 , 7) ;
    }
    MOVIMENTOS
        LIGAR;
        NORTE 2 ;
        ESTE 150 ;
        ESTE 3 ;
        ESTE 2 ;
        SUL 1 ;
        OESTE 5 ;
        SUL 5;
```

```
        DESLIGAR ;
        SUL 0;
        NORTE 10;
        LIGAR;
        OESTE 4;
}
```

vamos obter dois tipos de output, um na consola e outro gráfico.

Output em texto:

```
Movimento ESTE 150 inválido por ultrapassar os limites da área de limpeza!

Terreno{uni=25, larg=15, alt=15}

Robo{posx=3, posy=11, posx_ini=7, posy_ini=7,
     estado=LIGADO, dir=OESTE,
     norte=[2], sul=[1, 5], este=[3, 2], oeste=[5, 4], mud_dir=5,
movs={0=Movimento{num=0, direcao=NORTE, distancia=2},
     1=Movimento{num=1, direcao=ESTE, distancia=3},
     2=Movimento{num=2, direcao=ESTE, distancia=2},
     3=Movimento{num=3, direcao=SUL, distancia=1},
     4=Movimento{num=4, direcao=OESTE, distancia=5},
     5=Movimento{num=5, direcao=SUL, distancia=5},
     6=Movimento{num=6, direcao=OESTE, distancia=4}},
totalMovs=7}

ESTATISTICAS
    Norte:
        Total deslocações: 1
        Total distancia percorrida: 50
        Media de distancia percorrida por cada movimentacao: 50.0
    Sul:
        Total deslocações: 2
        Total distancia percorrida: 150
        Media de distancia percorrida por cada movimentacao: 75.0
    Este:
        Total deslocações: 2
        Total distancia percorrida: 125
        Media de distancia percorrida por cada movimentacao: 62.5
    Oeste:
        Total deslocações: 2
        Total distancia percorrida: 225
        Media de distancia percorrida por cada movimentacao: 112.0
    TOTAL:
        Total deslocações: 7
        Total distancia percorrida: 550
        Media de distancia percorrida por cada movimentacao: 78.57143
        Total mudancas direcao: 5
```

Analisando este output, o que é impresso primeiro é o movimento inválido, depois quando o programa chega ao fim faz o toString das classes `Terreno` e `Robo`, seguidas das estatísticas.

Interface gráfico:

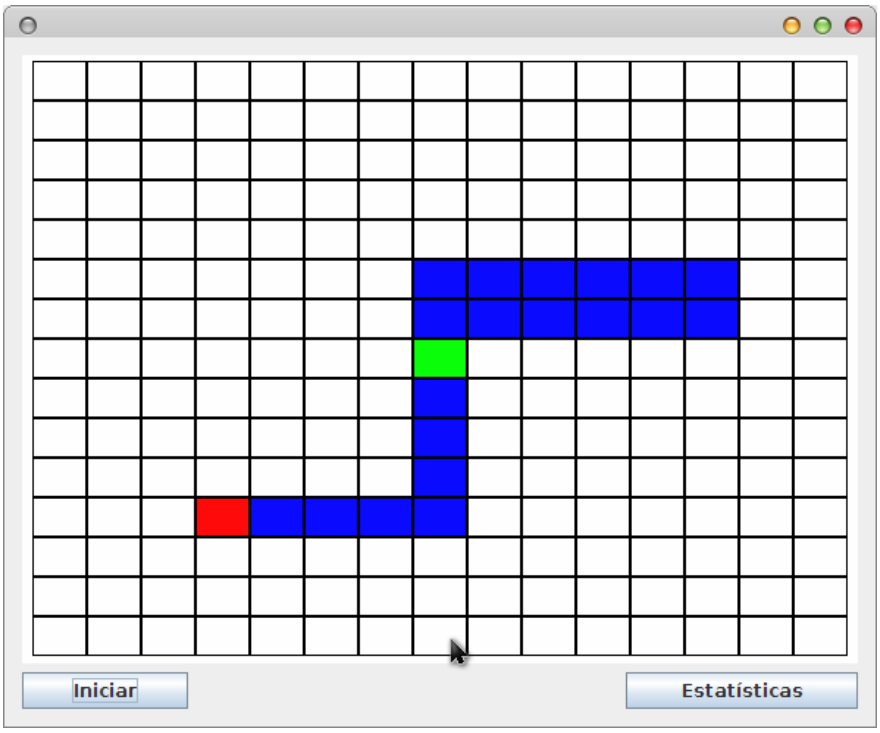

Figura 3: Terreno percorrido pelo Robo

A célula preenchida de cor verde corresponde à posição inicial em que o Robo foi colocado e a célula vermelha à posição final.

Analisando a frase fornecida como input, podemos concluir que as células pintadas corresponde ao trajecto introduzido.

Temos ainda a opção de clicar no botão estatísticas que nos apresenta a seguinte informação:

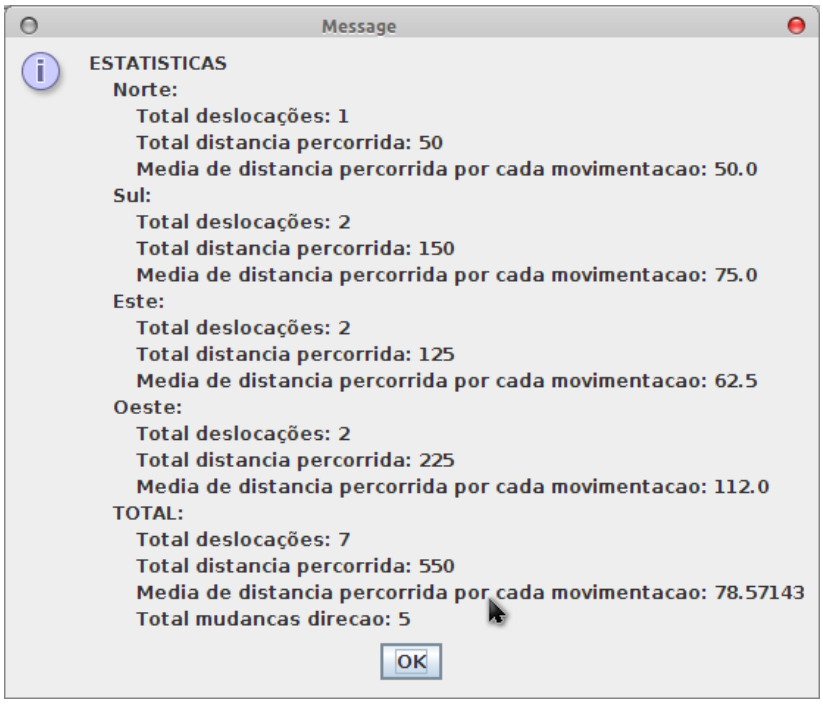

Figura 4: Estatísticas do Robo

4 Conclusões

A resolução deste exercício permitiu perceber melhor a forma como as linguagens podem ser úteis para gerar um programa, que dependo do input que irá receber, o resultado final seja o esperado sem ter de estar a alterar o código do programa que é automaticamente gerado.

Umas das dificuldades foi perceber como o Antlr fazia o parser das frases de forma a não haver ambiguidade e conseguir na mesma produção termos acesso ao valor de dois símbolos terminais, tal como acontece, por exemplo, quando queremos saber a dimensão do terreno, em que a solução foi inserir labels para o compilador saber qual o valor pertendido.

Serviu de consolidação da matéria dada até agora no módulo de Engenharia de Linguagens, tendo em conta que conseguimos resolver os exercícios com sucesso.