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

21 de Fevereiro de 2012

Resumo

Este documento apresenta a resolução do Exercício Prático n.º 3 do módulo de Engenharia Gramatical. O exercício está relacionado com a geração automática de Processadores de Linguagens a partir de Gramáticas.

Era pretendido 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

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 linguem de programação adoptada foi o JAVA. De forma 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 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 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 1: Estrutura da gramática

```
1  ASPIRADOR
2  {
3  DEFINICOES
4  {
5  definicao1; definicao2;
6  }
7  MOVIMENTOS
8  instrucao1;
9  instrucao2;
10 ....
11 }
```

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 2: Gramática

```
PLTNgrammar 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)*
        ;
```

```
41
42 movimento
43         : LIGAR ';'
44         | DESLIGAR ';'
45         | NORTE INT ';'
46         | SUL INT ';'
47         | ESTE INT ';'
48         | OESTE INT ';'
49         ;
50
51 /*------------------------------------------------------------------
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 3: Frase gerada 1

```
1  ASPIRADOR
2  {
3          DEFINICOES
4          {
5                  dim = (100 , 150) ; pos = (0 , 0) ;
6          }
7          MOVIMENTOS
8                  LIGAR;
9                  NORTE 2 ;
10                 DESLIGAR ;
11                 SUL 10;
12         }
```

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

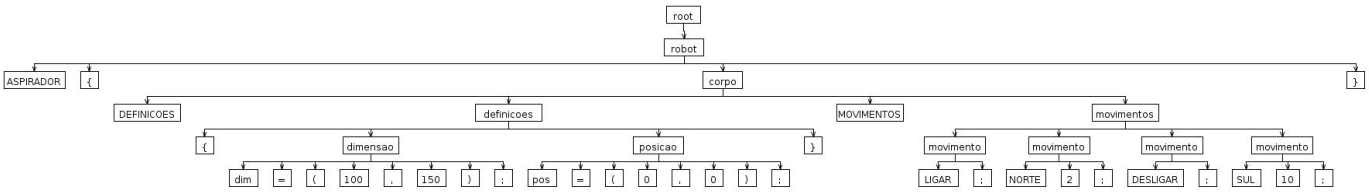

Figura 1: Árvore de derivação

Analisando a gramática a as frases geradas a partir dela, 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.

4 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.

4.1 Decisões Tomadas

Como seria de esperar, há promenores 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 que a de `LIGAR`.

4.2 Classes

As classes criadas foram:

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

A classe `Robo` é a responsável por guardar o estado, a posicao atual, a direção atual e todos os movimentos executados pelo robo e por gerar as estatísticas relacionadas com os mesmos. `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.

4.3 Linguagem

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 4: Linaguem Final

```
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();
        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 '=' '('    x=INT{terreno.setLarg(Integer.parseInt(x.getText()));} ','
                        y=INT{terreno.setAlt(Integer.parseInt(y.getText()));}
                  ')' ';'
        ;
posicao
        :POS '=' '('x=INT { if (terreno.validaPosX(Integer.parseInt(x.getText()))){
           robo.setPosX(x.getText()); robo.setPosXini(x.getText());}
                            else System.out.println("PosiÃ■Ã■o inicial invÃ¡lida.");
```

```
54                          }
55                  ','
56                     y=INT { if (terreno.validaPosY(Integer.parseInt(y.getText()))) {
                           robo.setPosY(y.getText()); robo.setPosYini(y.getText());}
57                             else System.out.println("PosiÃ■Ã■o inicial invÃ¡lida.");
58                          }
59                  ')' ';'
60          ;
61
62  movimentos
63          : movimento (movimento)*
64          ;
65
66  movimento
67          : LIGAR ';'             {robo.setEstado("LIGADO");}
68          | DESLIGAR ';'          {robo.setEstado("DESLIGADO");}
69          | NORTE INT ';'         { if (terreno.validaPosY(robo.getPosY() - Integer.
              parseInt($INT.text))) {robo.movNorte(Integer.parseInt($INT.text));}
70                                    else {System.out.println("Movimento NORTE "+ $INT.
                                          text +" invÃ¡lido por ultrapassar os limites da
                                           Ã¡rea de limpeza!");}
71                                  }
72          | SUL INT ';'           {if (terreno.validaPosY(robo.getPosY() + Integer.
              parseInt($INT.text))) {robo.movSul(Integer.parseInt($INT.text));}
73                                   else {System.out.println("Movimento SUL "+ $INT.text
                                        +" invÃ¡lido por ultrapassar os limites da
                                       Ã¡rea de limpeza!");}
74                                  }
75          | ESTE INT ';'          { if (terreno.validaPosX(robo.getPosX() + Integer.
              parseInt($INT.text))) {robo.movEste(Integer.parseInt($INT.text));}
76                                    else {System.out.println("Movimento ESTE "+ $INT.
                                          text +" invÃ¡lido por ultrapassar os limites da
                                           Ã¡rea de limpeza!");}
77                                  }
78          | OESTE INT ';'         { if (terreno.validaPosX(robo.getPosX() - Integer.
              parseInt($INT.text))) {robo.movOeste(Integer.parseInt($INT.text));}
79                                    else {System.out.println("Movimento OESTE "+ $INT.
                                          text +" invÃ¡lido por ultrapassar os limites da
                                           Ã¡rea 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
```

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

COMMENT
    :   '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    |   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
    ;

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

5 Conclusões

A resolução deste exercício permitiu perceber melhor a forma como as linguagens de estrutura para a resolução de determinados problemas. Depois de definida a GIC e criando a GA, conseguimos realizar os cálculos que eram pretendidos para a soma.

Apesar de serem dois exercícios para calcular um resultado de forma diferente, deu para perceber que o reaciocínio para resolver é idêntifo com ambos os casos.

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