

Exercício para Avaliação n.º 3

Bruno Azevedo* and Miguel Costa†

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

21 de Fevereiro de 2012

Resumo

Este documento apresenta a resolução do Exercício Prático n.º 3 do módulo de Engenharia de Linguagens. 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	3
2	Descrição do problema	3
3	Criação da linguagem	3
4	Implementação	6
4.1	Classes	6
4.2	Linguagem	6
5	Conclusões	9

1 Ambiente de Trabalho

Foi necessário usar um Gerador de Compiladores para gerar o nosso próprio compilador, por isso usamos o ANTLR que é também usado nas aulas. Para facilitar o debug durante a resolução do problema que era dada, usamos a ferramenta ANTLRWorks, que tem um interface bastante agradável e simpático para problemas desta natureza.

A linguagem de programação adoptada foi o JAVA. De forma tornar a nossa solução mais legível e estruturada, criamos classes com o auxílio do IDE NetBeans que nos ajuda 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 pela 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 uma 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 inicio certas definições como a dimensão do terreno e a posição inicial do robo.
2. Depois de definida a linguagem tinhamos de criar um processador para as possíveis frases que podiam ser geradas com as funcionalidades de:
 - 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, tentamos 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 simbolos terminais e neste caso nós definimos os símbolos:

- DIM
- POS
- LIGAR
- DESLIGAR
- NORTE
- SUL
- ESTE
- OESTE
- ID
- INT

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

Listing 2: Gramática

```

1  PLTNgrammar robot;
2
3  /*-----
4  * _PARSER RULES
5  *-----*/
6
7  robot
8  @init {
9      terreno = new Terreno();
10     robo = new Robo(terreno);
11 }
12 @after {
13     System.out.println(terreno.toString());
14     System.out.println(robo.toString());
15     System.out.println(robo.toStringEstatisticas());
16
17     Matrix m = new Matrix(robo, terreno);
18     m.setVisible(true);
19 }
20 : 'ASPIRADOR' '{' corpo '}'
21 ;
22
23 corpo
24 : 'DEFINICOES' definicoes 'MOVIMENTOS' movimentos
25 ;
26
27 definicoes
28 : '{' dimensao (posicao)? '}'
29 | '{' (posicao)? dimensao '}'
30 ;
31 dimensao
32 : DIM '=' '(' x=INT ',' y=INT ')' ';'
33 ;
34 posicao
35 : POS '=' '(' x=INT ',' y=INT ')' ';'
36 ;
37
38 movimentos
39 : movimento (movimento)*
40 ;

```

```

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, algumas das frases que se podem gerar, são:

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:

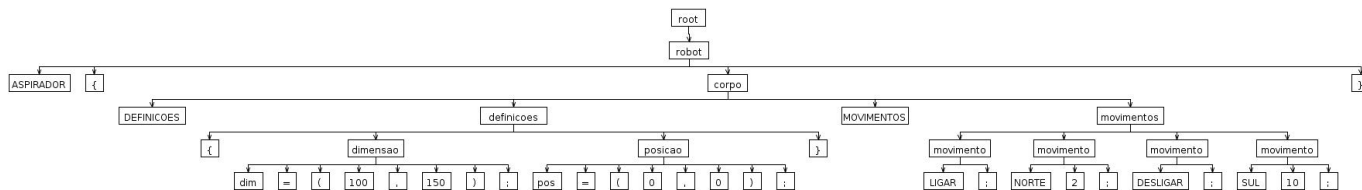


Figura 1: Árvore de derivação

Analisando a gramática e as frases geradas a partir dela, verificamos que o elemento raiz é **robot** e o parser terá de encontrar no início a palavra **ASPIRADOR** seguido de um **corpo** que se encontra dentro de chavetas.

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

Quanto aos **movimentos**, estes podem ser de 2 tipos, os que fazem realmente mover 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, criamos em classes em java que nos facilitassem o cálculo de todas as estatísticas e todas as restrições que eram necessárias.

4.1 Classes

As classes criadas foram:

- Robo
- Terreno
- Matrix

A classe **Robo** é a responsável por guardar todos os movimentos e por gerar as estatísticas. **Terreno** é a classe que contém as dimensões do território em que o Robo se pode deslocar e verifica se este não se quer deslocar para fora dele. Para confirmar se tudo o que era pedido ao Robo se concretizava, criamos um pequeno interface em que é possível ver a deslocação passo a passo do Robo e ainda as estatísticas. Este interface corresponde à classe **Matrix** que recorre ao Java SWING para criar a animação.

Em anexo está o código java de cada classe.

4.2 Linguagem

Depois de criadas as classes em Java, foi necessário adaptar a nossa gramática de forma a realizar o que era pretendido, adicionamos variáveis para o Robo, o **Terreno** e para o interface (**Matrix**).

Acabou por ficar:

Listing 4: Linguagem Final

```

1 grammar robot;
2
3 options {
4     language = Java;
5 }
6
7 @header{

```

```

8      import Robot.Robo;
9      import Robot.Terreno;
10     import Robot.Matrix;
11 }
12
13 @members{
14     private Robo robo;
15     private Terreno terreno;
16 }
17
18 /*-----
19  * PARSER RULES
20  *-----*/
21
22 robot
23 @init {
24     terreno = new Terreno();
25     robo = new Robo(terreno);
26 }
27 @after {
28     System.out.println(terreno.toString());
29     System.out.println(robo.toString());
30     System.out.println(robo.toStringEstatisticas());
31
32     Matrix m = new Matrix(robo, terreno);
33     m.setVisible(true);
34 }
35 : 'ASPIRADOR' '{' corpo '}'
36 ;
37
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()));} ','
48             y=INT{terreno.setAlt(Integer.parseInt(y.getText()));}
49             ')' ',' ';'
50 ;
51 posicao
52 :POS '=' '(' x=INT { if (terreno.validaPosX(Integer.parseInt(x.getText()))){
53     robo.setPosX(x.getText()); robo.setPosXini(x.getText());}
54     else System.out.println("PosiÃ§Ã£o inicial invÃ¡lida.");
55     }
56     ','
57     y=INT { if (terreno.validaPosY(Integer.parseInt(y.getText())) {
58     robo.setPosY(y.getText()); robo.setPosYini(y.getText());}
59     else System.out.println("PosiÃ§Ã£o inicial invÃ¡lida.");
60     }
61     ')' ',' ';'
62 ;
63
64 movimentos
65 : movimento (movimento)*
66 ;
67 movimento

```

```

67 : LIGAR ';' {robo.setEstado("LIGADO");}
68 | DESLIGAR ';' {robo.setEstado("DESLIGADO");}
69 | NORTE INT ';' { if (terreno.validaPosY(robo.getPosY() - Integer.
70     parseInt($INT.text))) {robo.movNorte(Integer.parseInt($INT.text));}
        else {System.out.println("Movimento NORTE "+ $INT.
            text +" inválido por ultrapassar os limites da
            área de limpeza!");}
71     }
72 | SUL INT ';' {if (terreno.validaPosY(robo.getPosY() + Integer.
73     parseInt($INT.text))) {robo.movSul(Integer.parseInt($INT.text));}
        else {System.out.println("Movimento SUL "+ $INT.text
            +" inválido por ultrapassar os limites da
            área de limpeza!");}
74     }
75 | ESTE INT ';' { if (terreno.validaPosX(robo.getPosX() + Integer.
76     parseInt($INT.text))) {robo.movEste(Integer.parseInt($INT.text));}
        else {System.out.println("Movimento ESTE "+ $INT.
            text +" inválido por ultrapassar os limites da
            área de limpeza!");}
77     }
78 | OESTE INT ';' { if (terreno.validaPosX(robo.getPosX() - Integer.
79     parseInt($INT.text))) {robo.movOeste(Integer.parseInt($INT.text));}
        else {System.out.println("Movimento OESTE "+ $INT.
            text +" inválido por ultrapassar os limites da
            área de limpeza!");}
80     }
81 ;
82
83 /*-----
84 * LEXER RULES
85 *-----*/
86
87 DIM : ('d'|'D')('i'|'I')('m'|'M');
88 POS : ('p'|'P')('o'|'O')('s'|'S');
89
90 LIGAR : ('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
91 DESLIGAR : ('d'|'D')('e'|'E')('s'|'S')('l'|'L')('i'|'I')('g'|'G')('a'|'A')('r'|'R');
92
93 NORTE : ('n'|'N')('o'|'O')('r'|'R')('t'|'T')('e'|'E');
94 SUL : ('s'|'S')('u'|'U')('l'|'L');
95 ESTE : ('e'|'E')('s'|'S')('t'|'T')('e'|'E');
96 OESTE : ('o'|'O')('e'|'E')('s'|'S')('t'|'T')('e'|'E');
97
98 ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
99 ;
100
101 INT : '0'..'9'+
102 ;
103
104 COMMENT
105 : '//' ~('\n'|\r)* '\r'? '\n' {$channel=HIDDEN;}
106 | '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
107 ;
108
109 WS : ( ' '
110 | '\t'
111 | '\r'
112 | '\n'
113 ) {$channel=HIDDEN;}
114 ;

```


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 raciocínio para resolver é idêntico 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.