Implementação de um compilador. Etapa 2: Analisador sintático

**Marcelo Fonseca Faraj, Rodrigo Rodrigues do Carmo**

02 de maio de 2016

Trabalho entregue como parte das atividades da disciplina de Compiladores, do Centro Federal de Educação Tecnológica de Minas Gerais , ministrada pela professora Kécia Marques

# **Introdução e metodologia**

Um analisador sintático, também chamado de *parser* é uma peça de software responsável por receber um fluxo de tokens e verificar se estes se apresentam em uma ordem tal que respeite as regras de uma dada gramática. Em outros termos, podemos dizer que se trata de um programa que verifica se um dado fluxo de tokens constitui uma sequência de tokens que pode ser gerada pela gramática em questão.

O processo de verificação se um dado texto pertence à linguagem gerada por uma gramática é realizado por máquinas abstratas chamadas autômatos. No caso de uma análise léxica, um autômato finito determinístico é suficiente para os propósitos da maioria das linguagens. Entretanto, para a análise sintática, tona-se necessário uma quantidade maior de poder computacional, razão pela qual autômatos de pilha são geralmente considerados mais indicados para esta finalidade.

Dentre as diversas possibilidades de *parsers*, podemos dividi-los em duas categorias, sendo cada uma constituída por duas linhas de implementação. Quanto à direção em que é construída a derivação, *parsers* podem ser ascendentes, se a derivação for construída dos tokens ou folhas para a variável de partida ou raíz, ou descendentes, se a derivação for feita a partir da raíz em direção às folhas. Quanto à árvore de derivação gerada, podemos ter as de derivação mais à direita e as de derivação mais à esquerda.

O algoritmo que utilizamos para realizar nossa implementação é conhecido como LL(1) e consiste em um parser preditivo, ou seja, descendente, que analisa o texto fonte da esquerda para a direita e que gera uma árvore de derivação mais à esquerda. Trata-se de um parser com estrutura simples, elegante e com bom desempenho, o que o coloca entre as melhores opções sempre que seu uso é uma possibilidade. Entretanto, cabe observar que o requisito para que uma dada gramática possa ser reconhecida por um *parser* LL(1) é que sua tabela de ações, construída com base em suas tabela First-Follow não contenha qualquer ambiguidade.

Para a garantir a ausência de ambiguidades da nossa gramática, a primeira etapa que realizamos foi a remoção de fatoração ou recursão à esquerda presente em algumas produções da gramática original. São estas:

*if-stmt ::= if condition then stmt-list end | if condition then stmt-list else stmt-list end*

*expression ::= simple-expr | simple-expr relop simple-expr*

*decl-list ::= decl ";" { decl ";"}*

*ident-list ::= identifier {"," identifier}*

*stmt-list ::= stmt ";" { stmt ";"}*

*simple-expr ::= term | simple-expr addop term*

*term ::= factor-a | term mulop factor-a*

Realizando as expansões e criações necessárias de novos símbolos não terminais, substituímos as construções acima pelas que se seguem:

*if-stmt ::= if condition then stmt-list if-stmt’*

*if-stmt’ ::= end | else stmt-list end*

*expression ::= simple-expr expression’*

*expression’ ::= λ | relop simple-expr*

*decl-list ::= decl ";" decl-list’*

*decl-list’ ::= λ | decl-list*

*ident-list ::= identifier ident-list’*

*ident-list’ ::= λ | "," ident-list*

*stmt-list ::= stmt ";" stmt-list’*

*stmt-list’ ::= λ | stmt-list*

*simple-expr ::= term simple-expr’*

*simple-expr’ ::= λ | addop simple-expr*

*term ::= factor-a term’*

*term’ ::= λ | mulop term*

Feito isto, discriminamos e numeramos cada instrução específica da gramática obtida, números estes que terão uso posterior na construção da tabela de decisão do *parser*. Seguem as 58 (cinquenta e oito) instruções gramaticais de origem sintática obtidas:

1. program ::= **var** decl-list **begin** stmt-list **end**
2. program ::= **begin** stmt-list **end**
3. decl-list ::= decl **";"** decl-list’
4. decl-list’ ::= λ
5. decl-list’ ::= decl-list
6. decl ::= ident-list **is** type
7. ident-list ::= **identifier** ident-list’
8. ident-list’ ::= λ
9. ident-list’ ::= **","** ident-list
10. type ::= **int**
11. type ::= **string**
12. stmt-list ::= stmt **";"** stmt-list’
13. stmt-list’ ::= λ
14. stmt-list’ ::= stmt-list
15. stmt ::= assign-stmt
16. stmt ::= if-stmt
17. stmt ::= do-stmt
18. stmt ::= read-stmt
19. stmt ::= write-stmt
20. assign-stmt ::= **identifier** **":="** simple\_expr
21. if-stmt ::= **if** condition **then** stmt-list if-stmt’
22. if-stmt’ ::= **end**
23. if-stmt’ ::= **else** stmt-list **end**
24. condition ::= expression
25. do-stmt ::= **do** stmt-list stmt-suffix
26. stmt-suffix ::= **while** condition
27. read-stmt ::= **in** **"("** identifier **")"**
28. write-stmt ::= **out** **"("** writable **")"**
29. writable ::= simple-expr
30. expression ::= simple-expr expression’
31. expression’ ::= λ
32. expression’ ::= relop simple-expr
33. simple-expr ::= term simple-expr’
34. simple-expr’ ::= λ
35. simple-expr’ ::= addop simple-expr
36. term ::= factor-a term’
37. term’ ::= λ
38. term’ ::= mulop term
39. factor-a ::= factor
40. factor-a ::= **not** factor
41. factor-a ::= **"-"** factor
42. factor ::= **identifier**
43. factor ::= constant
44. factor ::= **"("** expression **")"**
45. relop ::= **"="**
46. relop ::= **">"**
47. relop ::= **">="**
48. relop ::= **"<"**
49. relop ::= **"<="**
50. relop ::= **"<>"**
51. addop ::= **"+"**
52. addop ::= **"-"**
53. addop ::= **or**
54. mulop ::= **"\*"**
55. mulop ::= **"/"**
56. mulop ::= **and**
57. constant ::= **integer\_const**
58. constant ::= **literal**

A etapa seguinte consiste em determinar a tabela *first-follow* da gramática. Esta tabela indica quais *tokens* são esperados no topo da expansão de cada símbolo não terminal bem como quais são os tokens que podem ocorrer imediatamente após cada um dos símbolos não terminais.

Tabela *First-Follow*

|  |  |  |
| --- | --- | --- |
|  | **FIRST** | **FOLLOW** |
| **program** | var, begin | $ |
| **decl-list** | identifier | begin |
| **decl-list’** | λ, identifier | begin |
| **decl** | identifier | “;” |
| **ident-list** | identifier | is |
| **ident-list’** | λ, “,” | is |
| **type** | int, string | “;” |
| **stmt-list** | identifier, if, do, in, out | end, else, while |
| **stmt-list’** | λ, identifier, if, do, in, out | end, else, while |
| **stmt** | identifier, if, do, in, out | “;” |
| **assign-stmt** | identifier | “;” |
| **if-stmt** | if | “;” |
| **if-stmt’** | end, else | “;” |
| **condition** | identifier, literal, integer\_const, “(”, not, “-” | then, “;” |
| **do-stmt** | do | “;” |
| **stmt-suffix** | while | “;” |
| **read-stmt** | in | “;” |
| **write-stmt** | out | “;” |
| **writable** | identifier, literal, integer\_const, “(”, not, “-” | “)” |
| **expression** | identifier, literal, integer\_const, “(”, not, “-” | “)”, then, “;” |
| **expression’** | λ, "=" , ">" , ">=" , "<" , "<=", "<>" | “)”, then, “;” |
| **simple-expr** | identifier, literal, integer\_const, “(”, not, “-” | "=", ">", ">=", "<", "<=", "<>", “)”, then, “;” |
| **simple-expr’** | λ, “+”, “-”, or | "=", ">", ">=", "<", "<=", "<>", “)”, then, “;” |
| **term** | identifier, literal, integer\_const, “(”, not, “-” | "=", ">", ">=", "<", "<=", "<>", “+”, “-”, or, “)”, then, “;” |
| **term’** | “\*”, “/”, and | "=", ">", ">=", "<", "<=", "<>", “+”, “-”, or, “)”, then, “;” |
| **factor-a** | identifier, literal, integer\_const, “(”, not, “-” | “\*”, “/”, and, "=", ">", ">=", "<", "<=", "<>", “+”, “-”, or, “)”, then, “;” |
| **factor** | identifier, integer\_const, literal, “(” | “\*”, “/”, and, "=", ">", ">=", "<", "<=", "<>", “+”, “-”, or, “)”, then, “;” |
| **relop** | "=" , ">" , ">=" , "<" , "<=", "<>" | identifier, integer\_const, “(”, not, “-” |
| **addop** | “+”, “-”, or | identifier, integer\_const, “(”, not, “-” |
| **mulop** | “\*”, “/”, and | identifier, integer\_const, “(”, not, “-” |
| **constant** | integer\_const, literal | “\*”, “/”, and, "=", ">", ">=", "<", "<=", "<>", “+”, “-”, or, “)”, then, “;” |

Por fim, esta tabela serve de base para a construção do que chamamos tabela do parser. Fizemos esta tabela em um software de planilhas e o arquivo correspondente é intitulado **TableParser.xlsx**. Além disso, também anexamos a tabela do parser no fim deste relatório, para fins de simplificação.

# **Experimentos e resultados**

Realizamos experimentos com os mesmos 8 códigos fonte utilizados no teste do analisador léxico. Desta vez, utilizamos o analisador sintático para realizar todas as correções necessárias para que a análise sintática se dê sem qualquer erro. Seguem nos códigos e suas respectivas mensagens de erro, quando há:

|  |  |
| --- | --- |
| Código Fonte | Erro indicado pelo compilador |
| teste1.txt  var  a, b, c is int;  result is int  begin  in (a);  in (c);  b := 10;  result := (a \* c)/(b + 5 - 345);  out(result);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Syntax Error in line 4:0 in non\_terminal "decl\_list1"  decl\_list structure must be:  decl ";" {decl ";"} |
| teste1.txt – corrigido  var  a, b, c is int;  result is int;  begin  in (a);  in (c);  b := 10;  result := (a \* c)/(b + 5 - 345);  out(result);  end |  |
| teste2.txt  a, \_ is int;  b is int;  nome is string;  begin  in (a);  in (nome);  b := a \* a;  b := b + a/2 \* (3 + 5);  out (nome);  out(b);  end. | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Syntax Error in line 1:1 in non\_terminal "program"  Program structure must be:  ["var" decl\_list] "begin" stmt\_list "end" |
| teste2.txt - corrigido  var  a, \_a is int;  b is int;  nome is string;  begin  in (a);  in (nome);  b := a \* a;  b := b + a/2 \* (3 + 5);  out (nome);  out(b);  end//. |  |
| teste3.txt  var  \_cont is int;  media, altura, soma\_ is int;  begin  \_cont := 5;  soma = 0;  do  write({Altura: });  in (altura);  soma := soma altura;  \_cont := \_cont - 1;  while(\_cont);  out({Media: });  out (soma / qtd);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Error: Invalid identifier name "\_" in line 3:25 |
| teste3.txt - corrigido  var  \_cont is int;  media, altura, soma is int;  begin  \_cont := 5;  soma := 0;  do  out({Altura: });  in (altura);  soma := soma + altura;  \_cont := \_cont - 1;  while(\_cont);  out({Media: });  out (soma / qtd);  end |  |
| teste4.txt  var  i, j, k, @total, 1soma is int;  begin  i := 4 \* (5-3 \* 50 / 10;  j := i \* 10;  k := i\* j / k;  k := 4 + a $;  out(i);  out(j);  out(k);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Error: Character : '@' in line 2:14 does not belong to language Ke$ia |
| teste4.txt corrigido  var  i, j, k, total, soma is int;  begin  i := 4 \* (5-3 \* 50 / 10);  j := i \* 10;  k := i\* j / k;  k := 4 + a ;  out(i);  out(j);  out(k);  end |  |
| teste5.txt  var  j, k is int;  a, j real;  begin  read(j);  read(K);  if (k <> 0)  result := j/k  else  result := 0  write({Altura: });  in (altura);  soma := soma altura;  \_cont := \_cont - 1;  end;  out(result);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Syntax Error in line 3:13 in non\_terminal "ident\_list2"  ident\_list structure must be:  identifier {"," ident\_list} |
| teste5.txt corrigido  var  j, k is int;  a, j is int;  begin  in(j);  in(K);  if k <> 0 then  result := j/k;  else  result := 0;  end;  out({Altura: });  in (altura);  soma := soma + altura;  \_cont := \_cont - 1;  end;  out(result);  end |  |
| teste6.txt  var  a, b, c, maior is int; nomecompletodoalunodeposgraduacao is string; start  read(a);  read(b);  read(c;  maior := 0;  if(a>banda>c )then maior := a;  else  if (b>c) then  maior := b;  else  maior := c;  end  end  Out({Maior idade: });  out(maior);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Error: Identifier's name "nomecompletodoalunodeposgraduacao" in line 2:57 exceeds maximum length |
| teste6.txt corrigido  var  a, b, c, maior is int; nomecompleto%doalunodeposgraduacao% is string;  begin  in(a);  in(b);  in(c);  maior := 0;  if a>banda then  if banda >c then  maior := a;  end;  else  if (b>c) then  maior := b;  else  maior := c;  end;  end;  out({Maior idade: });  out(maior);  end |  |
| teste7.txt  var  i, j, k, target is int  begin  read(target);  i:=j:=k:=1;  cont=2;  if (target>=3)  do  i:=j;  j:=k;  k:=i+j;  cont:=cont+1;  while(cont<target)  out(k);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Syntax Error in line 3:0 in non\_terminal "decl\_list1"  decl\_list structure must be:  decl ";" {decl ";"} |
| teste7.txt corrigido  var  i, j, k, target is int;  begin  in(target);  i:=j;j:=k;k:=1;  cont:=2;  if (target>=3) then  do  i:=j;  j:=k;  k:=i+j;  cont:=cont+1;  while(cont<target);  out(k);  end;  end |  |
| teste8.txt  % Exemplo de comentario  bla bla bla %  var //variaveis  i, j, k, target is int  begin  read(target);  i:=j:=k:=1;  cont=2;  if (target>=3)  do  i:=j;  j:=k;  k:=i+j;  cont:=cont+1;  while(cont<target)  out(k);  end | SYNTAX ANALYZER 1.0 -> Marcelo e Rodrigo  Syntax Error in line 6:0 in non\_terminal "decl\_list1"  decl\_list structure must be:  decl ";" {decl ";"} |
| teste8.txt corrigido  % Exemplo de comentario  bla bla bla %  var //variaveis  i, j, k, target is int;  begin  in(target);  i:=j;j:=k;k:=1;  cont:=2;  if target>=3 then  do  i:=j;  j:=k;  k:=i+j;  cont:=cont+1;  while cont<target;  end;  out(k);  end |  |

# ANEXO A: Tabela do parser LL(1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Var** | **Begin** | **Identifier** | **Int** | **String** | **If** | **Do** | **In** | **Out** | **End** | **Else** | **Literal** | **“(“** | **")"** | **integer-const** | **Not** | **while** | **“;”** | **then** | **“=”** | **“>”** | **“>=”** | **“<”** | **‘<=”** | **“<>”** | **“+”** | **“-“** | **"\*"** | **"/"** | **or** | **is** | **“,”** | **and** |
| **program** | **1** | **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **decl-list** |  |  | **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **decl-list'** |  | **4** | **5** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **decl** |  |  | **6** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **ident-list** |  |  | **7** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **ident-list'** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **8** | **9** |  |
| **type** |  |  |  | **10** | **11** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **stmt-list** |  |  | **12** |  |  | **12** | **12** | **12** | **12** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **stmt-list'** |  |  | **14** |  |  | **14** | **14** | **14** | **14** | **13** | **13** |  |  |  |  |  | **13** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **stmt** |  |  | **15** |  |  | **16** | **17** | **18** | **19** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **assign-stmt** |  |  | **20** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **if-stmt** |  |  |  |  |  | **21** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **if-stmt’** |  |  |  |  |  |  |  |  |  | **22** | **23** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **condition** |  |  | **24** |  |  |  |  |  |  |  |  | **24** | **24** |  | **24** | **24** |  |  |  |  |  |  |  |  |  |  | **24** |  |  |  |  |  |  |
| **do-stmt** |  |  |  |  |  |  | **25** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **stmt-suffix** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **26** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **read-stmt** |  |  |  |  |  |  |  | **27** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **write-stmt** |  |  |  |  |  |  |  |  | **28** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **writable** |  |  | **29** |  |  |  |  |  |  |  |  | **29** | **29** |  | **29** | **29** |  |  |  |  |  |  |  |  |  |  | **29** |  |  |  |  |  |  |
| **expression** |  |  | **30** |  |  |  |  |  |  |  |  | **30** | **30** | **30** | **30** | **30** |  |  |  |  |  |  |  |  |  |  | **30** |  |  |  |  |  |  |
| **expression’** |  |  |  |  |  |  |  |  |  |  |  |  |  | **31** |  |  |  | **31** | **31** | **32** | **32** | **32** | **32** | **32** | **32** |  |  |  |  |  |  |  |  |
| **simple-expr** |  |  | **33** |  |  |  |  |  |  |  |  | **33** | **33** |  | **33** | **33** |  |  |  |  |  |  |  |  |  |  | **33** |  |  |  |  |  |  |
| **simple-expr'** |  |  |  |  |  |  |  |  |  |  |  |  |  | **34** |  |  |  | **34** | **34** | **34** | **34** | **34** | **34** | **34** | **34** | **35** | **35** |  |  | **35** |  |  |  |
| **term** |  |  | **36** |  |  |  |  |  |  |  |  | **36** | **36** |  | **36** | **36** |  |  |  |  |  |  |  |  |  |  | **36** |  |  |  |  |  |  |
| **term'** |  |  |  |  |  |  |  |  |  |  |  |  |  | **37** |  |  |  | **37** | **37** | **37** | **37** | **37** | **37** | **37** | **37** | **37** | **37** | **38** | **38** | **37** |  |  | **38** |
| **factor-a** |  |  | **39** |  |  |  |  |  |  |  |  | **39** | **39** |  | **39** | **40** |  |  |  |  |  |  |  |  |  |  | **41** |  |  |  |  |  |  |
| **factor** |  |  | **42** |  |  |  |  |  |  |  |  | **43** | **44** |  | **43** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **relop** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **45** | **46** | **47** | **48** | **49** | **50** |  |  |  |  |  |  |  |  |
| **addop** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **51** | **52** |  |  | **53** |  |  |  |
| **mulop** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **54** | **55** |  |  |  | **56** |
| **constant** |  |  |  |  |  |  |  |  |  |  |  | **58** |  |  | **57** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |