

Gustavo P. Gouveia (6482819), Victor Lassance (6431325)

**Relatório de Compiladores**  
**Terceira Etapa**  
**Implementação do Reconhecedor Sintático**  
**Linguagem de programação CZAR**

Texto apresentado à Escola Politécnica da Universidade de São Paulo como requisito para a aprovação na disciplina Linguagens e Compiladores no quinto módulo acadêmico do curso de graduação em Engenharia de Computação, junto ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS).

Universidade de São Paulo

Escola Politécnica

Engenharia de Computação - Curso Cooperativo

Professor: Ricardo Luis de Azevedo da Rocha

São Paulo

2013

# Resumo

Este trabalho descreve a concepção e o desenvolvimento de um compilador utilizando a linguagem C. O escopo do compilador se limita a casos mais simples, porém simbólicos, e que servem ao aprendizado do processo de criação e teste de um compilador completo. A estrutura da linguagem escolhida para ser implementada se assemelha a própria estrutura do C, por facilidade de compreensão, porém com algumas peculiaridades trazidas de outras linguagens.

**Palavras-chaves:** Linguagens, Compiladores, Implementação do Reconhecedor Sintático.

# Sumário

<b>Sumário</b>	<b>3</b>
<b>1 Introdução</b>	<b>4</b>
<b>2 Descrição Atualizada da Linguagem em Wirth</b>	<b>5</b>
<b>3 Lista de Submáquinas do APE</b>	<b>6</b>
3.1 Lista de Transições	6
3.2 Lista de Autômatos	12
<b>4 Exemplos Atualizados de Programas na Linguagem CZAR</b>	<b>22</b>
4.1 Exemplo Geral	22
4.2 Exemplo Fatorial	23
<b>5 Comentários sobre a Implementação do Reconhecedor Sintático</b>	<b>24</b>
<b>Referências</b>	<b>25</b>

# 1 Introdução

Este projeto tem como objetivo a construção de um compilador de um só passo, dirigido por sintaxe, com analisador e reconhecedor sintático baseado em autômato de pilha estruturado.

Em um primeiro momento, foi definida uma linguagem de programação e identificados os tipos de átomos. Para cada átomo foi escrito uma gramática linear representativa da sua lei de formação e um reconhecedor para o átomo. Desse modo, as gramáticas assim escritas foram unidas e convertidas em um autômato finito, o qual foi transformado em um transdutor e implementado como sub-rotina, dando origem ao analisador léxico propriamente dito. Também foi criada uma função principal para chamar o analisador léxico e possibilitar o seu teste.

Durante a segunda etapa, a sintaxe da linguagem, denominada por nós de CZAR, foi definida formalmente a partir de uma definição informal e de exemplos de programas que criamos, misturando palavras-chave e conceitos de diferentes linguagens de programação. As três principais definições foram escritas na notação BNF<sup>1</sup>, Wirth<sup>2</sup> e com diagramas de sintaxe.

Nessa etapa, implementamos o módulo referente à parte sintática para a nossa linguagem. O papel do analisador sintático é obter uma cadeia de *tokens* proveniente do analisador léxico, e verificar se a mesma pode ser gerada pela gramática da linguagem e, com isso, construir a árvore sintática (ALFRED; SETHI; JEFFREY, 1986).

Como material de consulta, além de sites sobre o assunto e das aulas ministradas, foi utilizado o livro indicado pelo professor no começo das aulas (NETO, 1987), para pesquisa de conceitos e possíveis implementações.

O documento apresenta a seguir as respostas às questões propostas para a terceira etapa.

---

<sup>1</sup> Ver [http://en.wikipedia.org/wiki/Backus\\_Naur\\_Form](http://en.wikipedia.org/wiki/Backus_Naur_Form)

<sup>2</sup> Ver [http://en.wikipedia.org/wiki/Wirth\\_syntax\\_notation](http://en.wikipedia.org/wiki/Wirth_syntax_notation)

## 2 Descrição Atualizada da Linguagem em Wirth

```

1 PROGRAM          = IMPORTS DECLS_GLOBAIS DEF_PROCS_FUNCS "main" DEF_MAIN.
2 IMPORTS          = { "<" IDENT ">" } .
3
4 DECLS_GLOBAIS    = { "struct" IDENT "{" { [ "const" ] IDENT { "[" INT "]" } IDENT
   [ "=" EXPR ] { "," IDENT [ "=" EXPR ] } ";" } "}" | [ "const" ] IDENT { "["
   INT "]" } IDENT [ "=" EXPR ] { "," IDENT [ "=" EXPR ] } ";" } "meth".
5
6 DEF_PROCS_FUNCS  = { "void" IDENT LIST_PARAMS "{" { INSTR_SEM_RET } "}" | IDENT {
   "[" INT "]" } IDENT LIST_PARAMS "{" { INSTR_SEM_RET } "return" EXPR [ ";" ]
   "}" }.
7 LIST_PARAMS      = "(" [ [ "ref" ] IDENT { "[" INT "]" } IDENT { "," [ "ref" ]
   IDENT { "[" INT "]" } IDENT } ] ")" .
8
9 DEF_MAIN         = "(" ")" "{" { INSTR_SEM_RET } "}" .
10
11 INSTR_SEM_RET    = IDENT ( "[" ( [ "+" | "-" ] ( "(" EXPR ")" | ( INT | FLOAT |
   IDENT ( "(" [ EXPR { "," EXPR } ] ")" | { "[" EXPR "]" } { "." VARIDENT } ) )
   ) | STR | CHAR | BOOL { "*" | "/" | "%" ATOMO } { ( "+" | "-" ) TERM } "]" {
   "[" EXPR "]" } { "." VARIDENT } [ "+" | "-" | "*" | "/" | "%" ] "=" EXPR {
   "," VARIDENT OPER_ATRIB EXPR } ";" | INT "]" { "[" INT "]" } IDENT [ "="
   EXPR ] { "," IDENT [ "=" EXPR ] } ";" ) | IDENT [ "=" EXPR ] { "," IDENT [
   "=" EXPR ] } ";" | { "." VARIDENT } [ "+" | "-" | "*" | "/" | "%" ] "=" EXPR
   { "," VARIDENT OPER_ATRIB EXPR } ";" | "(" [ EXPR { "," EXPR } ] ")" ";" ) |
   "for" "(" IDENT { "[" INT "]" } IDENT [ "=" EXPR ] { "," IDENT [ "=" EXPR ]
   } ";" COND ";" VARIDENT OPER_ATRIB EXPR { "," VARIDENT OPER_ATRIB EXPR } ")"
   " {" { INSTR_SEM_RET } "}" | "while" "(" COND ")" "{" { INSTR_SEM_RET } "}"
   | "if" "(" COND ")" "{" { INSTR_SEM_RET } "}" ["else" "{" { INSTR_SEM_RET }
   "}" ] .
12
13 VARIDENT         = IDENT { "[" EXPR "]" } { "." IDENT { "[" EXPR "]" } } .
14
15 FUNCTION_CALL    = IDENT "(" [ EXPR { "," EXPR } ] ")" .
16
17 BOTH = IDENT ( { "[" EXPR "]" } { "." IDENT { "[" EXPR "]" } } | "(" [ EXPR {
   "," EXPR } ] ")" ) .
18
19 COND             = COND_TERM { ("and" | "or") COND_TERM } .
20 COND_TERM        = "(" COND ")" | ATOMO_COND { ("==" | "!=" | "<=" | ">=")
   ATOMO_COND } .
21 ATOMO_COND       = VARIDENT | BOOL | INT | "not" ATOMO_COND .
22
23 OPER_ATRIB       = "+=" | "-=" | "*=" | "/=" | "%=" | "=".
24
25 EXPR             = [ "+" | "-" ] TERM { ( "+" | "-" ) TERM } .
26 TERM             = "(" EXPR ")" | ATOMO { ( "*" | "/" | "%" ) ATOMO } .
27 ATOMO            = ( [ "+" | "-" ] ( BOTH | INT | FLOAT ) ) | STR | CHAR | BOOL .

```

## 3 Lista de Submáquinas do APE

### 3.1 Lista de Transições

**Autômato Finito Determinístico Mínimo da submáquina *ATOMO-COND*:**

```

1 initial: 0
2 final: 1
3 (0, VARIDENT) -> 1
4 (0, BOOL) -> 1
5 (0, INT) -> 1
6 (0, "not") -> 2
7 (2, ATOMO_COND) -> 1

```

**Autômato Finito Determinístico Mínimo da submáquina *ATOMO*:**

```

1 initial: 0
2 final: 2
3 (0, "+") -> 1
4 (0, "-") -> 1
5 (0, BOTH) -> 2
6 (0, INT) -> 2
7 (0, FLOAT) -> 2
8 (0, STR) -> 2
9 (0, CHAR) -> 2
10 (0, BOOL) -> 2
11 (1, BOTH) -> 2
12 (1, INT) -> 2
13 (1, FLOAT) -> 2

```

**Autômato Finito Determinístico Mínimo da submáquina *BOTH*:**

```

1 initial: 0
2 final: 1, 7, 8
3 (0, IDENT) -> 1
4 (1, "[") -> 2
5 (1, ".") -> 3
6 (1, "(") -> 4
7 (2, EXPR) -> 5
8 (3, IDENT) -> 8
9 (4, EXPR) -> 6
10 (4, ")") -> 7
11 (5, "]"") -> 8
12 (6, ",") -> 9
13 (6, ")") -> 7
14 (8, "[") -> 2
15 (8, ".") -> 3
16 (9, EXPR) -> 6

```

**Autômato Finito Determinístico Mínimo da submáquina *COND-TERM*:**

```

1 initial: 0
2 final: 2, 5
3 (0, "(" -> 1
4 (0, ATOMO_COND) -> 2
5 (1, COND) -> 3
6 (2, "==" -> 4
7 (2, "!=" -> 4
8 (2, "<=" -> 4
9 (2, ">=" -> 4
10 (3, ")") -> 5
11 (4, ATOMO_COND) -> 2

```

#### Autômato Finito Determinístico Mínimo da submáquina *COND*:

```

1 initial: 0
2 final: 1
3 (0, COND_TERM) -> 1
4 (1, "and") -> 0
5 (1, "or") -> 0

```

#### Autômato Finito Determinístico Mínimo da submáquina *DECLS-GLOBAIS*:

```

1 initial: 0
2 final: 4
3 (0, "struct") -> 1
4 (0, IDENT) -> 2
5 (0, "const") -> 3
6 (0, "meth") -> 4
7 (1, IDENT) -> 5
8 (2, IDENT) -> 9
9 (2, "[" -> 10
10 (3, IDENT) -> 2
11 (5, "{" -> 6
12 (6, IDENT) -> 7
13 (6, "const") -> 8
14 (6, "}") -> 0
15 (7, IDENT) -> 12
16 (7, "[" -> 13
17 (8, IDENT) -> 7
18 (9, "=") -> 15
19 (9, ",") -> 16
20 (9, ";") -> 0
21 (10, INT) -> 11
22 (11, "]" -> 2
23 (12, "=") -> 17
24 (12, ",") -> 18
25 (12, ";") -> 6
26 (13, INT) -> 14
27 (14, "]" -> 7
28 (15, EXPR) -> 19
29 (16, IDENT) -> 9
30 (17, EXPR) -> 20
31 (18, IDENT) -> 12
32 (19, ",") -> 16
33 (19, ";") -> 0
34 (20, ",") -> 18

```

35 | (20, ";") -> 6

### Autômato Finito Determinístico Mínimo da submáquina *DEF-MAIN*:

```

1 initial: 0
2 final: 4
3 (0, "(") -> 1
4 (1, ")") -> 2
5 (2, "{") -> 3
6 (3, INSTR_SEM_RET) -> 3
7 (3, "}") -> 4

```

### Autômato Finito Determinístico Mínimo da submáquina *DEF-PROCS-FUNCS*:

```

1 initial: 0
2 final: 9
3 (0, "void") -> 1
4 (0, IDENT) -> 2
5 (1, IDENT) -> 3
6 (2, IDENT) -> 4
7 (2, "[") -> 5
8 (3, LIST_PARAMS) -> 6
9 (4, LIST_PARAMS) -> 10
10 (5, INT) -> 7
11 (6, "{") -> 8
12 (7, "]" ) -> 2
13 (8, INSTR_SEM_RET) -> 8
14 (8, "}") -> 9
15 (9, "void") -> 1
16 (9, IDENT) -> 2
17 (10, "{") -> 11
18 (11, INSTR_SEM_RET) -> 11
19 (11, "return") -> 12
20 (12, EXPR) -> 13
21 (13, "}") -> 9
22 (13, ";") -> 14
23 (14, "}") -> 9

```

### Autômato Finito Determinístico Mínimo da submáquina *EXPR*:

```

1 initial: 0
2 final: 2
3 (0, "+") -> 1
4 (0, "-") -> 1
5 (0, TERM) -> 2
6 (1, TERM) -> 2
7 (2, "+") -> 1
8 (2, "-") -> 1

```

### Autômato Finito Determinístico Mínimo da submáquina *FUNCTION-CALL*:

```

1 initial: 0
2 final: 4

```



```

3 (0, IDENT) -> 1
4 (1, "(") -> 2
5 (2, EXPR) -> 3
6 (2, ")") -> 4
7 (3, ",") -> 5
8 (3, ")") -> 4
9 (5, EXPR) -> 3

```

### Autômato Finito Determinístico Mínimo da submáquina *IMPORTS*:

```

1 initial: 0
2 final: 3
3 (0, "<") -> 1
4 (1, IDENT) -> 2
5 (2, ">") -> 3
6 (3, "<") -> 1

```

### Autômato Finito Determinístico Mínimo da submáquina *INSTR-SEM-RET*:

```

1 initial: 0
2 final: 15, 19, 32, 35, 62, 66
3 (0, IDENT) -> 1
4 (0, "for") -> 2
5 (0, "while") -> 3
6 (0, "if") -> 4
7 (1, IDENT) -> 5
8 (1, "[") -> 6
9 (1, "+") -> 7
10 (1, "-") -> 7
11 (1, "(") -> 8
12 (1, ".") -> 9
13 (1, "*") -> 7
14 (1, "/") -> 7
15 (1, "%") -> 7
16 (1, "=") -> 10
17 (2, "(") -> 28
18 (3, "(") -> 23
19 (4, "(") -> 11
20 (5, ",") -> 53
21 (5, "=") -> 54
22 (5, ";") -> 15
23 (6, IDENT) -> 32
24 (6, "+") -> 33
25 (6, "-") -> 33
26 (6, "(") -> 34
27 (6, INT) -> 35
28 (6, FLOAT) -> 15
29 (6, STR) -> 15
30 (6, CHAR) -> 15
31 (6, BOOL) -> 36
32 (7, "=") -> 10
33 (8, EXPR) -> 25
34 (8, ")") -> 26
35 (9, VARIDENT) -> 22
36 (10, EXPR) -> 12

```

```

37 (11, COND) -> 13
38 (12, ",") -> 14
39 (12, ";") -> 15
40 (13, ")") -> 16
41 (14, VARIDENT) -> 18
42 (16, "{") -> 17
43 (17, INSTR_SEM_RET) -> 17
44 (17, "}") -> 19
45 (18, OPER_ATRIB) -> 10
46 (19, "else") -> 20
47 (20, "{") -> 21
48 (21, INSTR_SEM_RET) -> 21
49 (21, "}") -> 15
50 (22, "+") -> 7
51 (22, "-") -> 7
52 (22, ".") -> 9
53 (22, "*") -> 7
54 (22, "/") -> 7
55 (22, "%") -> 7
56 (22, "=") -> 10
57 (23, COND) -> 24
58 (24, ")") -> 20
59 (25, ")") -> 26
60 (25, ",") -> 27
61 (26, ";") -> 15
62 (27, EXPR) -> 25
63 (28, IDENT) -> 29
64 (29, IDENT) -> 30
65 (29, "[") -> 31
66 (30, ",") -> 41
67 (30, "=") -> 42
68 (30, ";") -> 43
69 (31, INT) -> 37
70 (32, "[") -> 59
71 (32, "(") -> 60
72 (32, ".") -> 61
73 (33, IDENT) -> 32
74 (33, "(") -> 34
75 (33, INT) -> 15
76 (33, FLOAT) -> 15
77 (34, EXPR) -> 67
78 (35, "]"") -> 56
79 (36, "+") -> 38
80 (36, "-") -> 38
81 (36, "]"") -> 39
82 (36, "*") -> 36
83 (36, "/"") -> 36
84 (36, "%") -> 40
85 (37, "]"") -> 29
86 (38, TERM) -> 52
87 (39, "[") -> 46
88 (39, "+") -> 7
89 (39, "-") -> 7
90 (39, ".") -> 9
91 (39, "*") -> 7
92 (39, "/"") -> 7
93 (39, "%") -> 7

```

```

94 (39, "=") -> 10
95 (40, ATOMO) -> 36
96 (41, IDENT) -> 30
97 (42, EXPR) -> 50
98 (43, COND) -> 44
99 (44, ";") -> 45
100 (45, VARIDENT) -> 47
101 (46, EXPR) -> 51
102 (47, OPER_ATRIB) -> 48
103 (48, EXPR) -> 49
104 (49, ")") -> 20
105 (49, ",") -> 45
106 (50, ",") -> 41
107 (50, ";") -> 43
108 (51, "]"") -> 39
109 (52, "+"") -> 38
110 (52, "-"") -> 38
111 (52, "]"") -> 39
112 (53, IDENT) -> 5
113 (54, EXPR) -> 55
114 (55, ",") -> 53
115 (55, ";") -> 15
116 (56, IDENT) -> 5
117 (56, "["") -> 57
118 (57, INT) -> 58
119 (58, "]"") -> 56
120 (59, EXPR) -> 65
121 (60, EXPR) -> 63
122 (60, ")") -> 15
123 (61, VARIDENT) -> 62
124 (62, "."") -> 61
125 (63, ")") -> 15
126 (63, ",") -> 64
127 (64, EXPR) -> 63
128 (65, "]"") -> 66
129 (66, "["") -> 59
130 (66, "."") -> 61
131 (67, ")") -> 15

```

### Autômato Finito Determinístico Mínimo da submáquina *LIST-PARAMS*:

```

1 initial: 0
2 final: 4
3 (0, "(") -> 1
4 (1, "ref") -> 2
5 (1, IDENT) -> 3
6 (1, ")") -> 4
7 (2, IDENT) -> 3
8 (3, IDENT) -> 5
9 (3, "["") -> 6
10 (5, ",") -> 7
11 (5, ")") -> 4
12 (6, INT) -> 8
13 (7, "ref") -> 2
14 (7, IDENT) -> 3
15 (8, "]"") -> 3

```

**Autômato Finito Determinístico Mínimo da submáquina *OPER-ATRIB*:**

```

1 initial: 0
2 final: 1
3 (0, "+=") -> 1
4 (0, "-=") -> 1
5 (0, "*=") -> 1
6 (0, "/=") -> 1
7 (0, "%=") -> 1
8 (0, "=") -> 1

```

**Autômato Finito Determinístico Mínimo da submáquina *PROGRAM*:**

```

1 initial: 0
2 final: 5
3 (0, IMPORTS) -> 1
4 (1, DECLS_GLOBAIS) -> 2
5 (2, DEF_PROCS_FUNCS) -> 3
6 (3, "main") -> 4
7 (4, DEF_MAIN) -> 5

```

**Autômato Finito Determinístico Mínimo da submáquina *TERM*:**

```

1 initial: 0
2 final: 2, 5
3 (0, "(") -> 1
4 (0, ATOMO) -> 2
5 (1, EXPR) -> 3
6 (2, "*") -> 4
7 (2, "/" ) -> 4
8 (2, "%") -> 4
9 (3, ")") -> 5
10 (4, ATOMO) -> 2

```

**Autômato Finito Determinístico Mínimo da submáquina *VARIDENT*:**

```

1 initial: 0
2 final: 1
3 (0, IDENT) -> 1
4 (1, "[" ) -> 2
5 (1, "." ) -> 0
6 (2, EXPR) -> 3
7 (3, "]" ) -> 1

```

## 3.2 Lista de Autômatos

- ATOMO-COND:

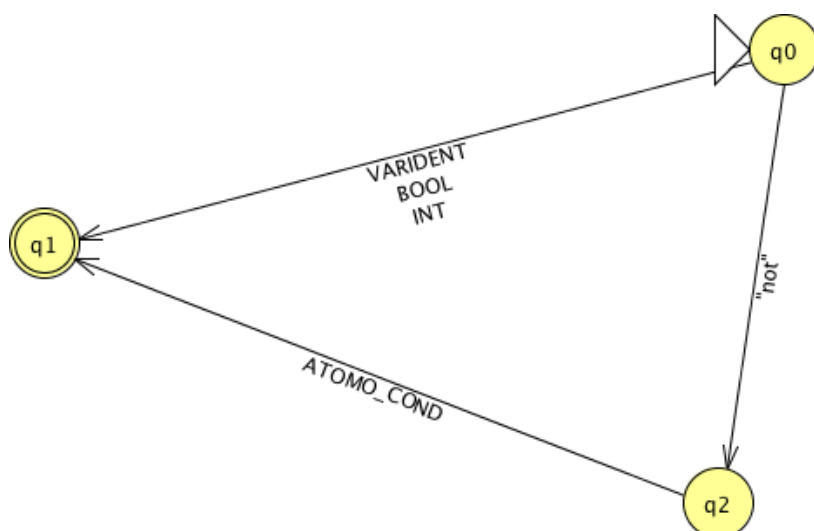


Figura 1 – Autômato ATOMO-COND

- ATOMO:

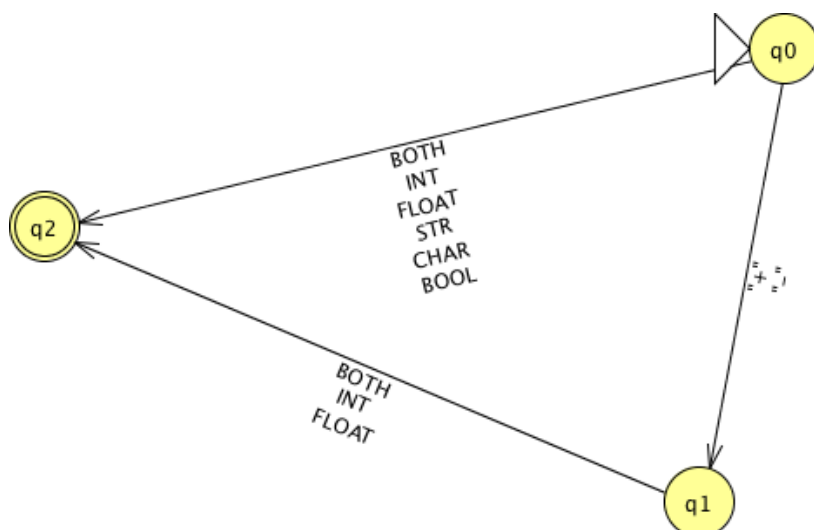


Figura 2 – Autômato ATOMO

- BOTH:

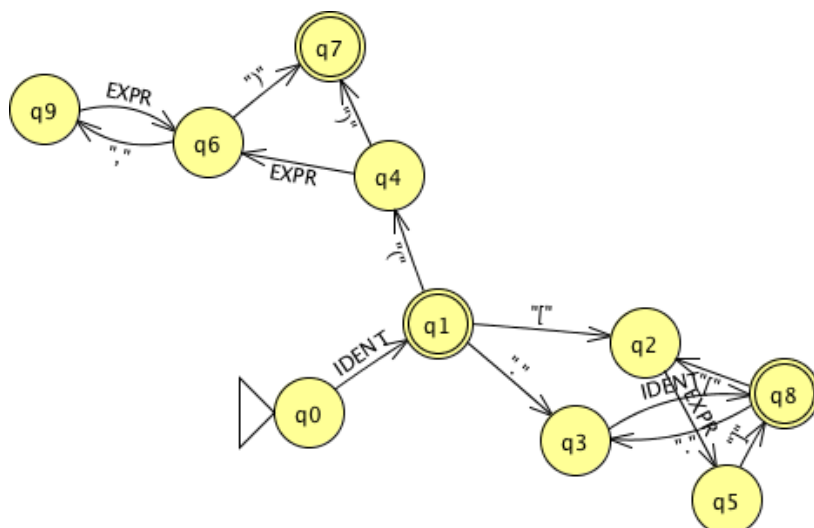


Figura 3 – Autômato BOTH

- COND-TERM:

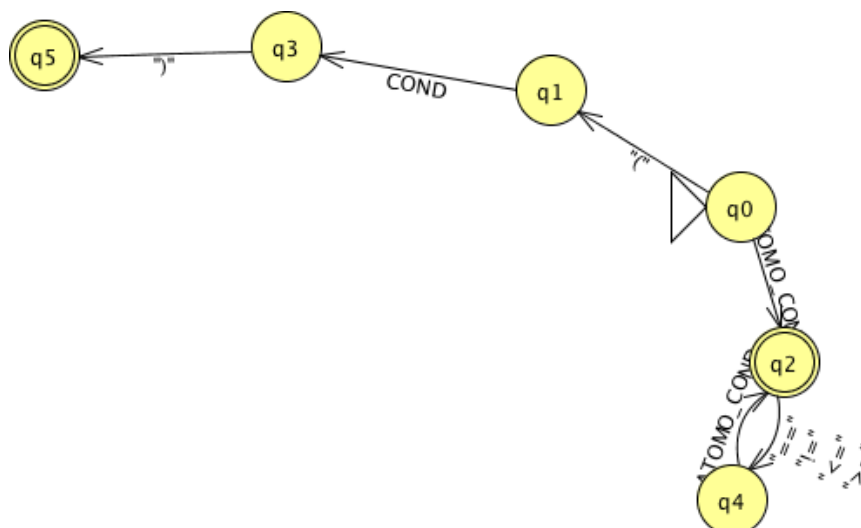


Figura 4 – Autômato COND-TERM

- COND:

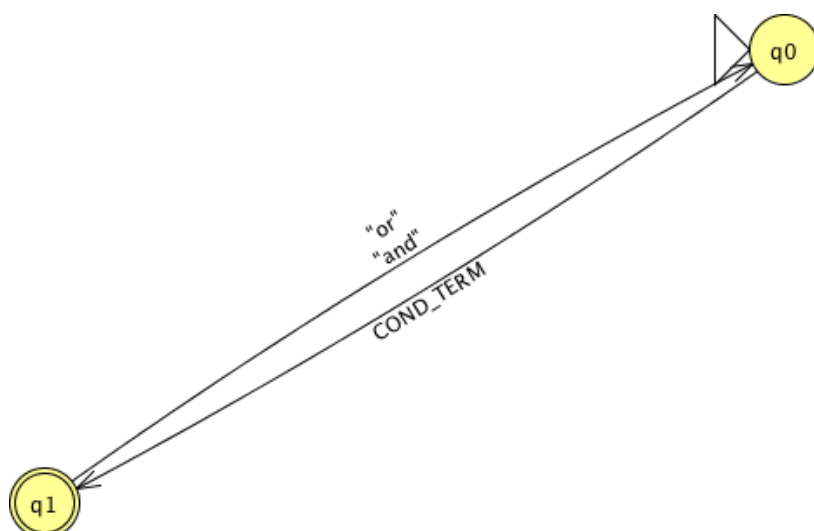


Figura 5 – Autômato COND

- DECLS-GLOBAIS:

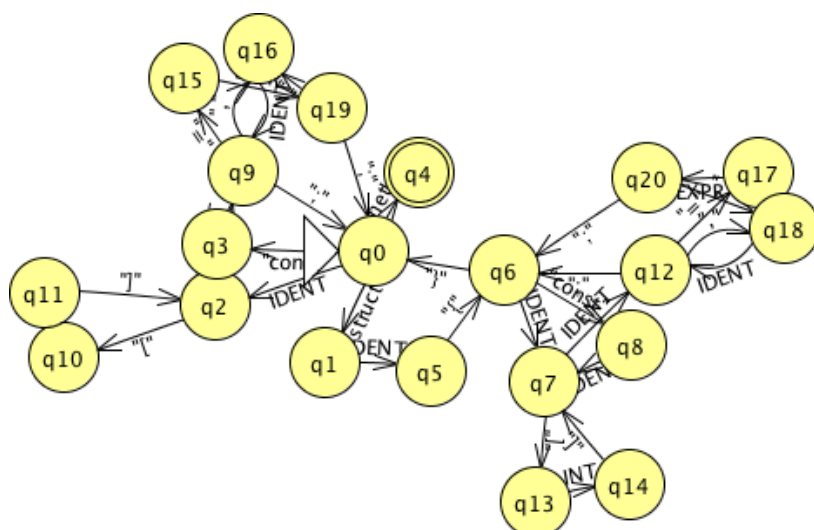


Figura 6 – Autômato DECLS-GLOBAIS

- DEF-MAIN:

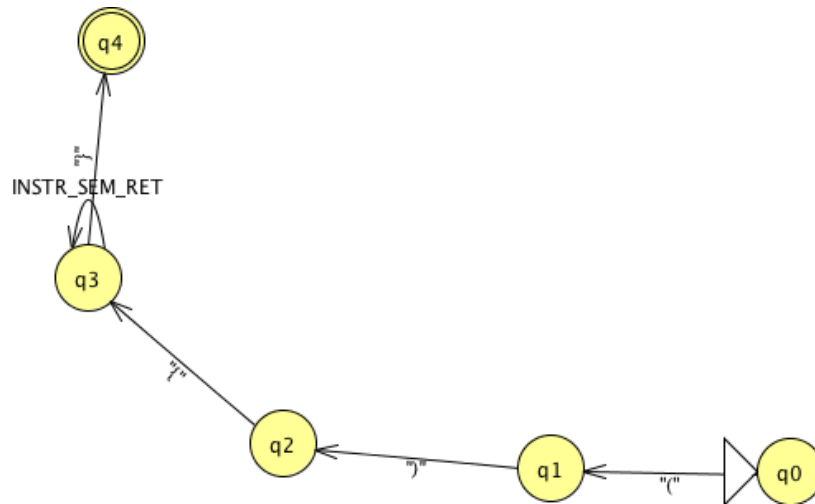


Figura 7 – Autômato DEF-MAIN

- DEF-PROCS-FUNCS:

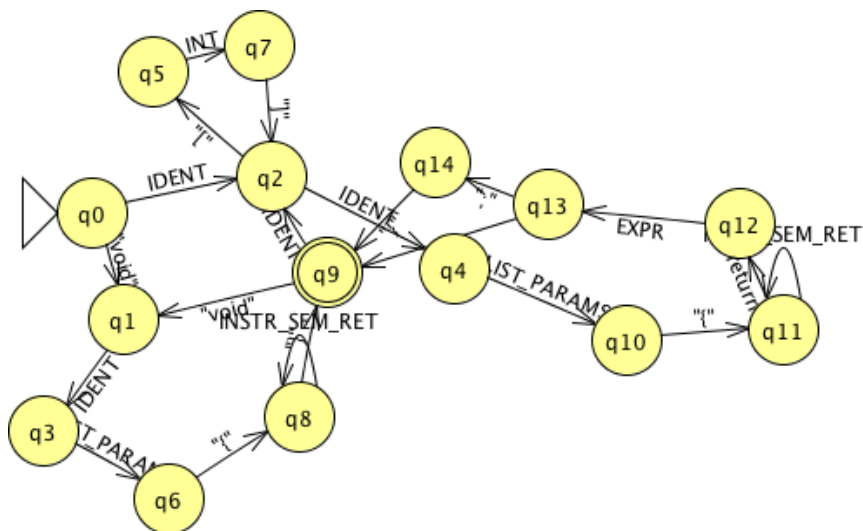


Figura 8 – Autômato DEF-PROCS-FUNCS

- EXPR:



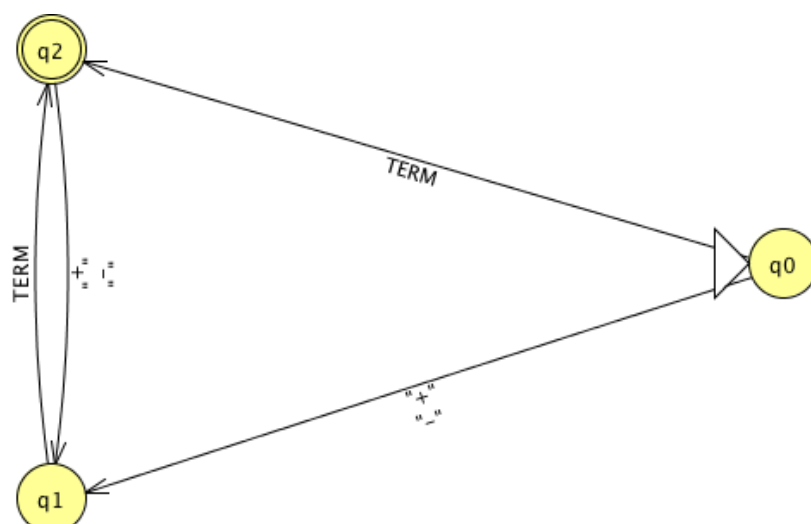


Figura 9 – Autômato EXPR

- FUNCTION-CALL:

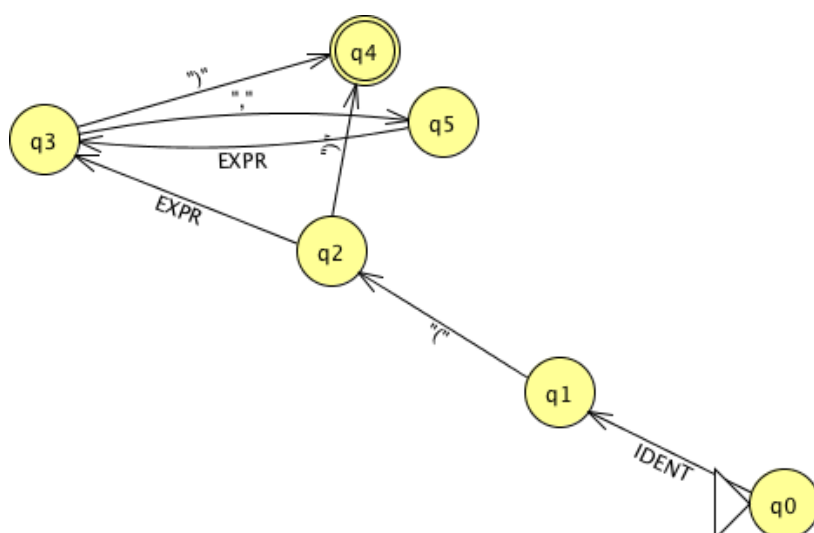


Figura 10 – Autômato FUNCTION-CALL

- IMPORTS:

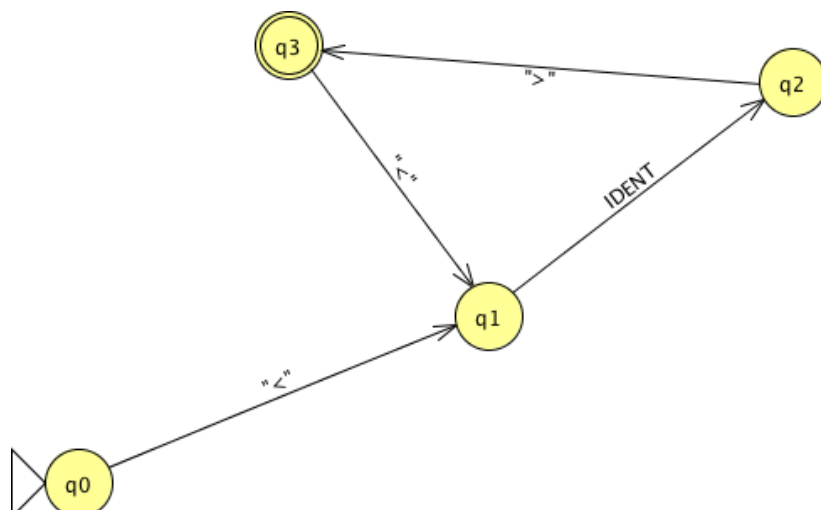


Figura 11 – Autômato IMPORTS

- INSTR-SEM-RET:

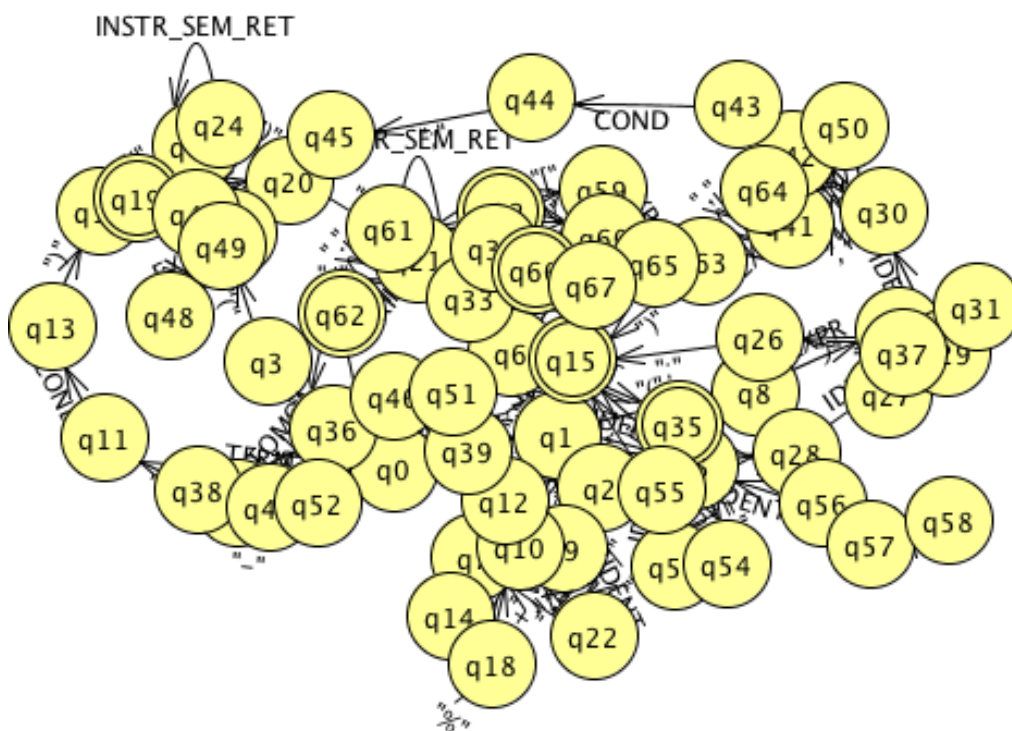


Figura 12 – Autômato INSTR-SEM-RET

- LIST-PARAMS:

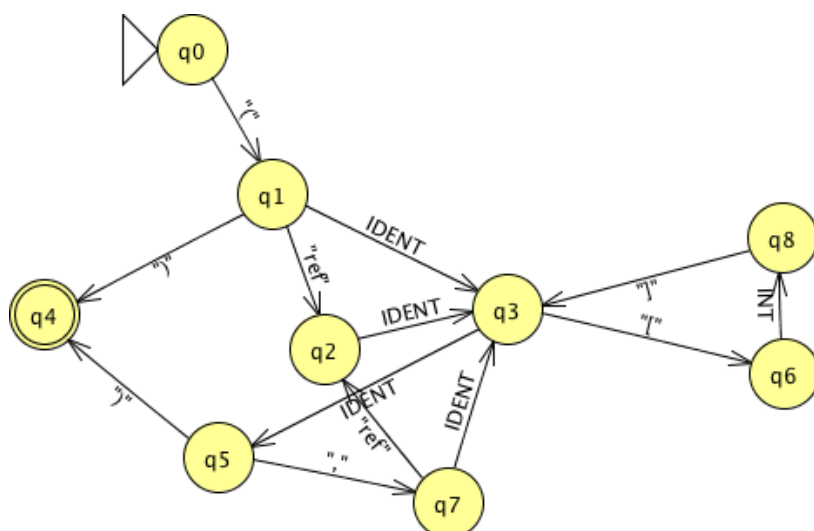


Figura 13 – Autômato LIST-PARAMS

- OPER-ATRIB:

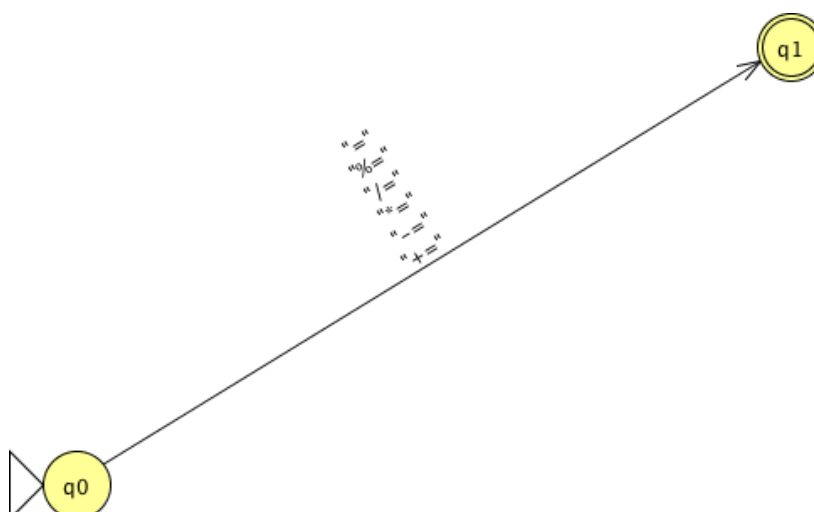


Figura 14 – Autômato OPER-ATRIB

- PROGRAM:

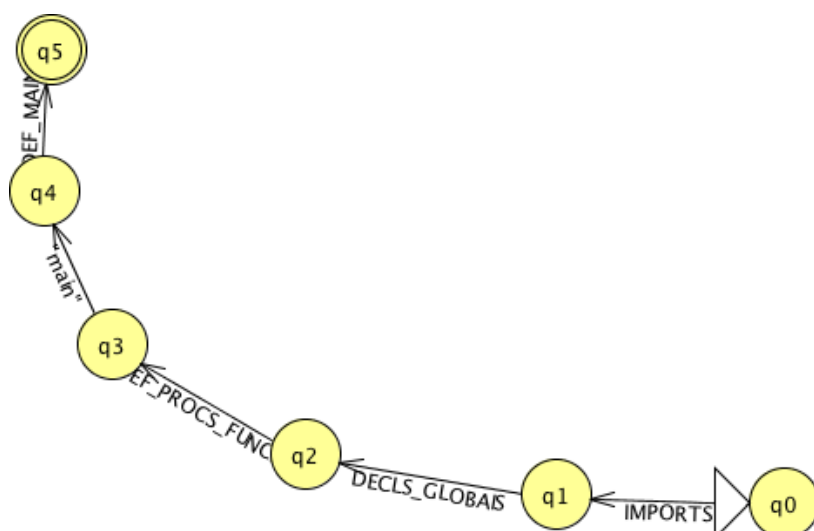


Figura 15 – Autômato PROGRAM

- TERM:

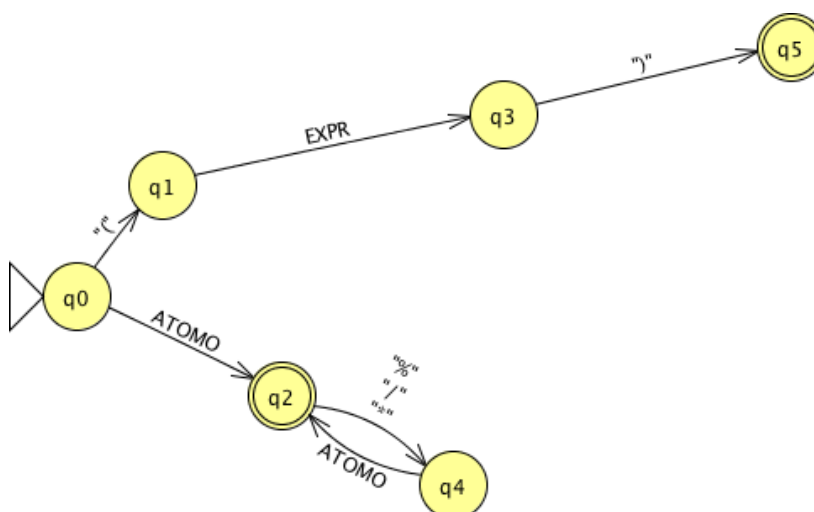


Figura 16 – Autômato TERM

- VARIDENT:

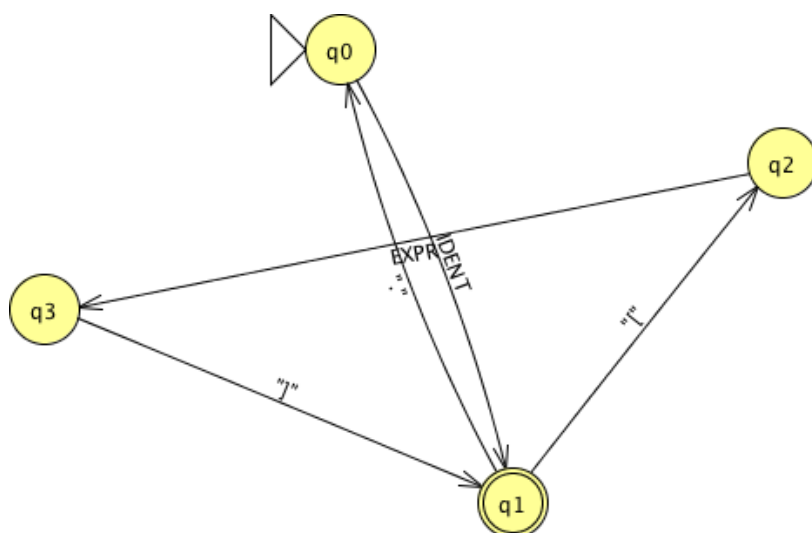


Figura 17 – Autômato VARIDENT

## 4 Exemplos Atualizados de Programas na Linguagem CZAR

### 4.1 Exemplo Geral

```

1  <math>
2  <io>
3
4  struct nome_struct {
5      nome_struct eu_mesmo;
6      int a;
7      char b;
8  }
9
10 const int SOU_CONSTANTE_INT = 10;
11 const string SOU_CONSTANTE_STRING = "CONSTANTE_STRING";
12 string sou_variavel = "valor inicial da variavel";
13
14 meth
15
16 void soma_como_procedimento (int a, int b, ref int soma) {
17     soma = a + b;
18 }
19
20 int soma_como_funcao (int a, int b) {
21     return a + b;
22 }
23
24 void proc_exemplo (char a, int b, int c, int d) {
25     int tmp;
26     char[32] buff;
27     soma_como_procedimento(b, c, tmp);
28     d = soma_como_funcao(tmp, c) + 5;
29     d = math_exp(SOU_CONSTANTE_INT, 2);
30     io_print(a);
31     io_int_to_str(d, buff);
32     io_print(" gives ");
33     io_print(buff);
34     io_print(" \n pointer to a is: ");
35     buff = a + "caracter";
36     io_print(buff);
37     io_print("bye");
38 }
39
40 main () {
41     proc_exemplo('x', 3, -6, -15);
42 }

```

## 4.2 Exemplo Fatorial

```
1 <io>
2
3 const int fat_10_rec = 10;
4 const int fat_10_iter = 10;
5 int retorno;
6
7 meth
8
9 int fatorial_recursivo(int n) {
10     int retorno = 1;
11     if (n >= 1) {
12         retorno = n * fatorial_recursivo (n - 1);
13     }
14     return retorno;
15 }
16
17 int fatorial_iterativo(int n) {
18     int fatorial = 1;
19     while (n >= 0) {
20         fatorial = fatorial * n;
21         n = n - 1;
22     }
23     return fatorial;
24 }
25
26 main () {
27     retorno = fatorial_recursivo(fat_10_rec);
28
29     io_print_int(retorno);
30     io_print(" ");
31     io_print_int(fatorial_iterativo(fat_10_iter));
32 }
```

## 5 Comentários sobre a Implementação do Reconhecedor Sintático

TODO: Gustavo explicar como o analisador sintático foi implementado e integrado ao compilador



## Referências

ALFRED, V.; SETHI, R.; JEFFREY, D. *Compilers: principles, techniques and tools*. [S.l.]: Addison-Wesley, 1986.

NETO, J. J. *Introdução à Compilação*. [S.l.]: LTC, 1987. (ENGENHARIA DE COMPUTAÇÃO).