

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

Relatório de Compiladores
Segunda Etapa
Definição formal da sintaxe da 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, Definição formal da Sintaxe.

Sumário

| | |
|---|-----------|
| Sumário | 3 |
| 1 Introdução | 4 |
| 2 Descrição Informal da Linguagem | 5 |
| 3 Exemplos de Programas na Linguagem | 7 |
| 3.1 Exemplo Geral | 7 |
| 3.2 Exemplo Fatorial | 8 |
| 4 Descrição da Linguagem em BNF | 10 |
| 5 Descrição da Linguagem em Wirth | 14 |
| 6 Diagrama de Sintaxe da Linguagem | 15 |
| 7 Conjunto das Palavras Reservadas | 17 |
| 8 Considerações Finais | 18 |
| Referências | 19 |

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.

Nesta 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 BNF¹, Wirth² e com diagramas de sintaxe.

Como material de consulta, além de sites sobre o assunto, como por exemplo um que permite verificar a definição em Wirth e criar os diagramas de sintaxe³, 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 segunda etapa, assim como as considerações finais.

¹ Ver http://en.wikipedia.org/wiki/Backus_Naur_Form

² Ver http://en.wikipedia.org/wiki/Wirth_syntax_notation

³ Site: <http://karmin.ch/ebnf/index>

2 Descrição Informal da Linguagem

O programa é composto por quatro partes, explicadas abaixo de forma simplificada, pois a linguagem será definida de forma completa nos capítulos 4 e 5 nas notações BNF e Wirth, respectivamente:

- Definição do programa:

Um programa em **czar** possui em ordem obrigatória, a importação de bibliotecas, declaração de variáveis globais, definição de funções e procedimentos. O programa deve terminar obrigatoriamente pela declaração da função principal **main**.

– PROGRAM = IMPORTS DECLS_GLOBAIS DEF_PROCS_FUNCS DEF_MAIN.

- Inclusão de bibliotecas:

– IMPORTS = { '<' IDENT '>' }.

- Declaração de tipos, variáveis e constantes de escopo global:

– DECLS_GLOBAIS = { DEF_TIPO | DECL }.

– DEF_TIPO = 'struct' IDENT '{' { DECL } '}'.

– DECL = ['const'] TIPO IDENT ['=' EXPR]
 { ',', IDENT ['=' EXPR] } ';'.

- Definição dos procedimentos e funções do programa, As funções não devem incluir o procedimento principal (chamado **main**). Estas também possuem retorno final único e obrigatório.

– DEF_PROCS_FUNCS = { PROC | FUNC }.

– FUNC = TIPO IDENT LIST_PARAMS
 '{' { INSTR_SEM_RET } "return" EXPR [";"] '}'.

– PROC = 'void' IDENT LIST_PARAMS '{' { INSTR_SEM_RET } '}'.

– LIST_PARAMS = '(' [['ref'] TIPO IDENT]
 { ',', ['ref'] TIPO IDENT } ')'

- Definição do procedimento principal (chamado **main**):

Não existe passagem explícita de parâmetros para a função **main**, ou seja, a passagem de valores para a mesma deve ocorrer por meio de arquivos ou pela utilização de

uma função incluída por alguma biblioteca *built-in* a ser feita, permitindo o acesso dos parâmetros em todas as partes do código.

– `DEF_MAIN = 'main' '(' ' ' ')' '{' [BLOCO] '}'`.

3 Exemplos de Programas na Linguagem

3.1 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 void soma_como_procedimento (int a, int b, ref int soma) {
15     soma = a + b;
16 }
17
18 int soma_como_funcao (int a, int b) {
19     return a + b;
20 }
21
22 string concatena_chars(int n_chars, char[] caracteres) {
23     string retorno = "";
24     for(int i = 0; i < n_chars; i += 1) {
25         retorno += caracteres[i];
26     }
27     return retorno;
28 }
29
30 void proc_exemplo (char a, int b, int c, int d) {
31     int tmp;
32     char[32] buff;
```

```
33 soma_como_procedimento(b, c, tmp);
34 d = soma_como_funcao(tmp, c) + 5;
35 d = math_exp(SOU_CONSTANTE_INT, 2);
36 io_print(a);
37 io_int_to_str(d, buff);
38 io_print(" gives ");
39 io_print(buff);
40 io_print(" \n pointer to a is: ");
41 buff = a + "character";
42 io_print(buff);
43 io_print("bye");
44 }
45
46 main () {
47     proc_exemplo('x', 3, -6, -15);
48 }
```

3.2 Exemplo Fatorial

```
1 <io>
2
3 const int fat_10_rec = 10;
4 const int fat_10_iter = 10;
5 int retorno;
6
7 int fatorial_recursivo(int n) {
8     int retorno = 1;
9     if (n > 1) {
10         retorno = n * fatorial_recursivo (n - 1);
11     }
12     return retorno;
13 }
14
15 int fatorial_iterativo(int n) {
16     int fatorial = 1;
17     while (n > 0) {
18         fatorial = fatorial * n;
19         n = n - 1;
20     }
```



```
21     return fatorial;
22 }
23
24 main () {
25     retorno = fatorial_recursivo(fat_10_rec);
26
27     io_print_int(retorno);
28     io_print(" ");
29     io_print_int(fatorial_iterativo(fat_10_iter));
30 }
```

4 Descrição da Linguagem em BNF

Apesar de termos visto em aula que a sintaxe BNF não costuma ter os não-terminais explicitados entre aspas, preferimos colocar aspas simples para facilitar a leitura, que também é uma forma válida de sintaxe BNF¹.

```

1  <PROGRAM>                ::= <IMPORTS> <DECLS_GLOBAIS> <DEF_PROCS_FUNCS> <DEF_MAIN>
2
3  <IMPORTS>                 ::= ε
4                          | '<' <IDENT> '>' <IMPORTS>
5
6  <DECLS_GLOBAIS>          ::= ε
7                          | <DEF_TIPO> <DECLS_GLOBAIS>
8                          | <DECL_CONST> <DECLS_GLOBAIS>
9                          | <DECL_VAR> <DECLS_GLOBAIS>
10
11 <DEF_TIPO>                ::= 'struct' <IDENT> '{' <DEF_INSTR_TIPO> '}'
12
13 <DEF_INSTR_TIPO>          ::= ε
14                          | <DECL_CONST>
15                          | <DECL_VAR>
16
17 <DECL_CONST>              ::= 'const' <DECL_VAR>
18
19 <DECL_VAR>                ::= <TIPO> <IDENT> <DECL_VAR_CONT> ';'
20                          | <TIPO> <IDENT> '=' <EXPR> <DECL_VAR_CONT> ';'
21
22 <DECL_VAR_CONT>           ::= ε
23                          | ',' <IDENT> <DECL_VAR_CONT>
24                          | ',' <IDENT> '=' <EXPR> <DECL_VAR_CONT>
25
26 <DEF_PROCS_FUNCS>         ::= ε
27                          | <PROC> <DEF_PROCS_FUNCS>
28                          | <FUNC> <DEF_PROCS_FUNCS>
29
30 <FUNC>                    ::= <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
31                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
32                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
33                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
34                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
35                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
36                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
37                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
38                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
39                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
40                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'
41                          | <TIPO> <IDENT> <LIST_PARAMS> '{' <INSTRUcoes> 'return'

```

¹ Ver <http://www.cs.man.ac.uk/~pjj/bnf/bnf.html>

```

42 <LIST_PARAMS_CONT> ::= ε
43                       | ',' <TIPO> <IDENT> <LIST_PARAMS_CONT>
44                       | ',' 'ref' <TIPO> <IDENT> <LIST_PARAMS_CONT>
45
46 <DEF_MAIN> ::= 'main' '(' ')' '{' <INSTRUcoes> '}'
47
48 <INSTR_SEM_RET> ::= <DECL_VAR>
49                  | <ATRIB>
50                  | <PROC_CALL>
51                  | <FLOW_CONTROL>
52
53 <ATRIB> ::= <ATRIB_SEM_PV> ';'
54
55 <ATRIB_SEM_PV> ::= <VARIDENT> <OPER_ATRIB> <EXPR> <ATRIB_SEM_PV_CONT>
56
57 <ATRIB_SEM_PV_CONT> ::= ε
58                      | ',' <VARIDENT> <OPER_ATRIB> <EXPR> <ATRIB_SEM_PV_CONT>
59
60 <PROC_CALL> ::= <IDENT> '(' ')' ';'
61              | <IDENT> '(' <EXPR> <PROC_CALL_CONT> ')' ';'
62
63 <PROC_CALL_CONT> ::= ε
64                  | ',' <EXPR> <PROC_CALL_CONT>
65
66 <FLOW_CONTROL> ::= <FOR_CONTROL>
67                  | <WHILE_CONTROL>
68                  | <IF_CONTROL>
69
70 <FOR_CONTROL> ::= 'for' '(' <DECL_VAR> <COND> ';' <ATRIB_SEM_PV> ')' '{'
71                <INSTRUcoes> '}'
72
73 <WHILE_CONTROL> ::= 'while' '(' <COND> ')' '{' <INSTRUcoes> '}'
74
75 <IF_CONTROL> ::= 'if' '(' <COND> ')' '{' <INSTRUcoes> '}'
76               | 'if' '(' <COND> ')' '{' <INSTRUcoes> '}' 'else' '{' <
77               INSTRUcoes> '}'
78
79 <TIPO> ::= <IDENT> <TIPO_CONT>
80
81 <TIPO_CONT> ::= ε
82              | '[' <INT> ']' <TIPO_CONT>
83
84 <IDENT_COLCHETES> ::= <IDENT> <IDENT_COLCH_CONT>
85
86 <IDENT_COLCH_CONT> ::= ε
87                    | '[' <EXPR> ']' <IDENT_COLCH_CONT>
88
89 <VARIDENT> ::= <IDENT_COLCHETES> <VARIDENT_CONT>
90
91 <VARIDENT_CONT> ::= ε
92                 | '.' <VARIDENT> <VARIDENT_CONT>
93
94 <FUNCTION_CALL> ::= <IDENT> '(' ')'
95                  | <IDENT> '(' <EXPR> <FUNCTION_CALL_CONT> ')'
96
97 <FUNCTION_CALL_CONT> ::= ε
98                      | ',' <EXPR> <FUNCTION_CALL_CONT>

```

```

97
98 <COND>                ::= <COND_TERM>
99                        | <COND_TERM> <OPER_BOOL> <COND_TERM>
100
101 <COND_TERM>           ::= '(' <COND> ')',
102                        | <ATOMO_COND>
103                        | <ATOMO_COND> <OPER_COMP> <COND_TERM>
104
105 <ATOMO_COND>          ::= <VARIDENT>
106                        | 'true'
107                        | 'false'
108                        | <INT>
109                        | 'not' <ATOMO_COND>
110
111 <OPER_ATRIB>          ::= '+='
112                        | '-='
113                        | '*='
114                        | '/='
115                        | '%='
116                        | '=='
117
118 <OPER_BOOL>           ::= 'and'
119                        | 'or'
120
121 <OPER_COMP>           ::= '=='
122                        | '!='
123                        | '<='
124                        | '>='
125
126 <OPER_ARIT>           ::= '+'
127                        | '-'
128
129 <OPER_TERM>           ::= '*'
130                        | '/'
131                        | '%'
132
133 <EXPR>                ::= <TERM>
134                        | <TERM> <OPER_ARIT_TERM_ARR>
135                        | <OPER_ARIT_TERM_ARR>
136
137 <OPER_ARIT_TERM_ARR> ::= <OPER_ARIT> <TERM>
138                        | <OPER_ARIT> <TERM> <OPER_ARIT_TERM_ARR>
139
140 <TERM>                ::= '(' <EXPR> ')',
141                        | <ATOMO>
142                        | <ATOMO> <OPER_TERM_ATOMO_ARR>
143
144 <OPER_TERM_ATOMO_ARR> ::= <OPER_TERM> <ATOMO>
145                        | <OPER_TERM> <ATOMO> <OPER_TERM_ATOMO_ARR>
146
147 <ATOMO>               ::= <FUNCTION_CALL>
148                        | <OPER_ARIT> <FUNCTION_CALL>
149                        | <INT>
150                        | <OPER_ARIT> <INT>
151                        | <STRING>
152                        | <CHAR>
153                        | <FLOAT>

```

| | |
|-----|------------------------|
| 154 | <OPER_ARIT> <FLOAT> |
| 155 | <BOOL> |
| 156 | <VARIDENT> |
| 157 | <OPER_ARIT> <VARIDENT> |

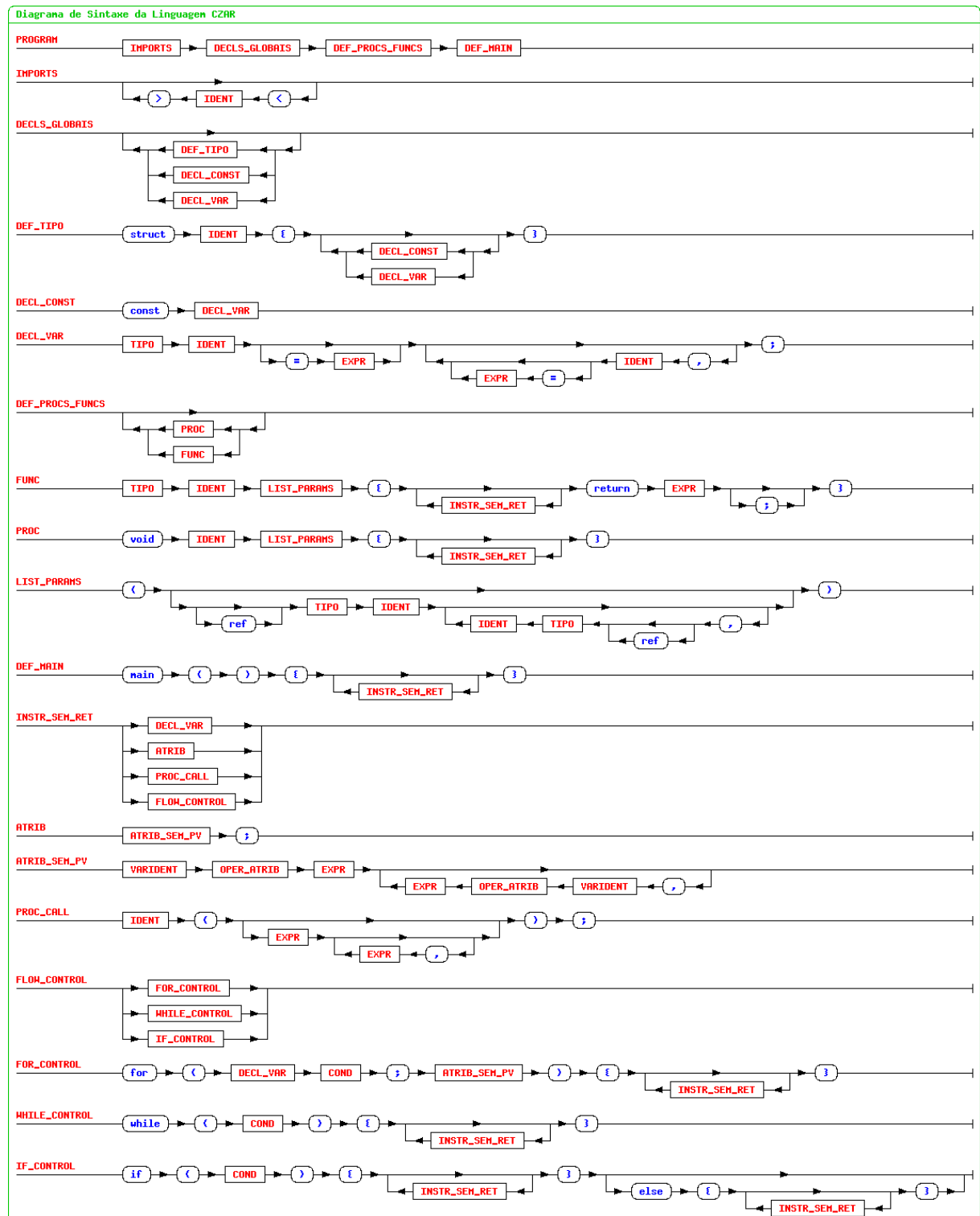
5 Descrição da Linguagem em Wirth

```

1 PROGRAM      = IMPORTS DECLS_GLOBAIS DEF_PROCS_FUNCS DEF_MAIN.
2
3 IMPORTS      = { "<" IDENT ">" }.
4
5 DECLS_GLOBAIS = { DEF_TIPO | DECL_CONST | DECL_VAR }.
6 DEF_TIPO     = "struct" IDENT "{" { DECL_CONST | DECL_VAR } "}".
7 DECL_CONST   = "const" DECL_VAR.
8 DECL_VAR     = TIPO IDENT [ "=" EXPR ] { "," IDENT [ "=" EXPR ] } ";".
9
10 DEF_PROCS_FUNCS = { PROC | FUNC }.
11 FUNC          = TIPO IDENT LIST_PARAMS "{" { INSTR_SEM_RET } "return" EXPR [
    ";" ] "}".
12 PROC          = "void" IDENT LIST_PARAMS "{" { INSTR_SEM_RET } "}".
13 LIST_PARAMS    = "(" [ [ "ref" ] TIPO IDENT { "," [ "ref" ] TIPO IDENT } ] ")".
14
15 DEF_MAIN      = "main" "(" ")" "{" { INSTR_SEM_RET } "}".
16
17 INSTR_SEM_RET = DECL_VAR | ATRIB | PROC_CALL | FLOW_CONTROL.
18 ATRIB         = ATRIB_SEM_PV ";".
19 ATRIB_SEM_PV  = VARIDENT OPER_ATRIB EXPR { "," VARIDENT OPER_ATRIB EXPR }.
20 PROC_CALL     = IDENT "(" [ EXPR { "," EXPR } ] ")" ";".
21 FLOW_CONTROL  = FOR_CONTROL | WHILE_CONTROL | IF_CONTROL.
22 FOR_CONTROL   = "for" "(" DECL_VAR COND ";" ATRIB_SEM_PV ")" "{" {
    INSTR_SEM_RET } "}".
23 WHILE_CONTROL = "while" "(" COND ")" "{" { INSTR_SEM_RET } "}".
24 IF_CONTROL    = "if" "(" COND ")" "{" { INSTR_SEM_RET } "}" ["else" "{" {
    INSTR_SEM_RET } "}" ].
25
26 TIPO          = IDENT { "[" INT "]" }.
27 IDENT_COLCHETES = IDENT { "[" EXPR "]" }.
28 VARIDENT      = IDENT_COLCHETES { "." VARIDENT }.
29
30 FUNCTION_CALL = IDENT "(" [ EXPR { "," EXPR } ] ")".
31
32 COND          = COND_TERM { OPER_BOOL COND_TERM }.
33 COND_TERM     = "(" COND ")" | ATOMO_COND { OPER_COMP ATOMO_COND }.
34 ATOMO_COND    = VARIDENT | BOOL | INT | "not" ATOMO_COND.
35 BOOL         = "true" | "false".
36
37 OPER_ATRIB    = ["+" | "-" | "*" | "/" | "%"] "=" .
38 OPER_BOOL     = "and" | "or".
39 OPER_COMP     = ("=" | "!=" | "<" | ">") "=" .
40 OPER_ARIT     = "+" | "-".
41 OPER_TERM     = "*" | "/" | "%".
42
43 EXPR          = [ OPER_ARIT ] TERM { OPER_ARIT TERM }.
44 TERM          = "(" EXPR ")" | ATOMO { OPER_TERM ATOMO }.
45 ATOMO         = [ OPER_ARIT ] FUNCTION_CALL | [ OPER_ARIT ] INT | STRING |
    CHAR | [ OPER_ARIT ] FLOAT | BOOL | [ OPER_ARIT ] VARIDENT.

```

6 Diagrama de Sintaxe da Linguagem



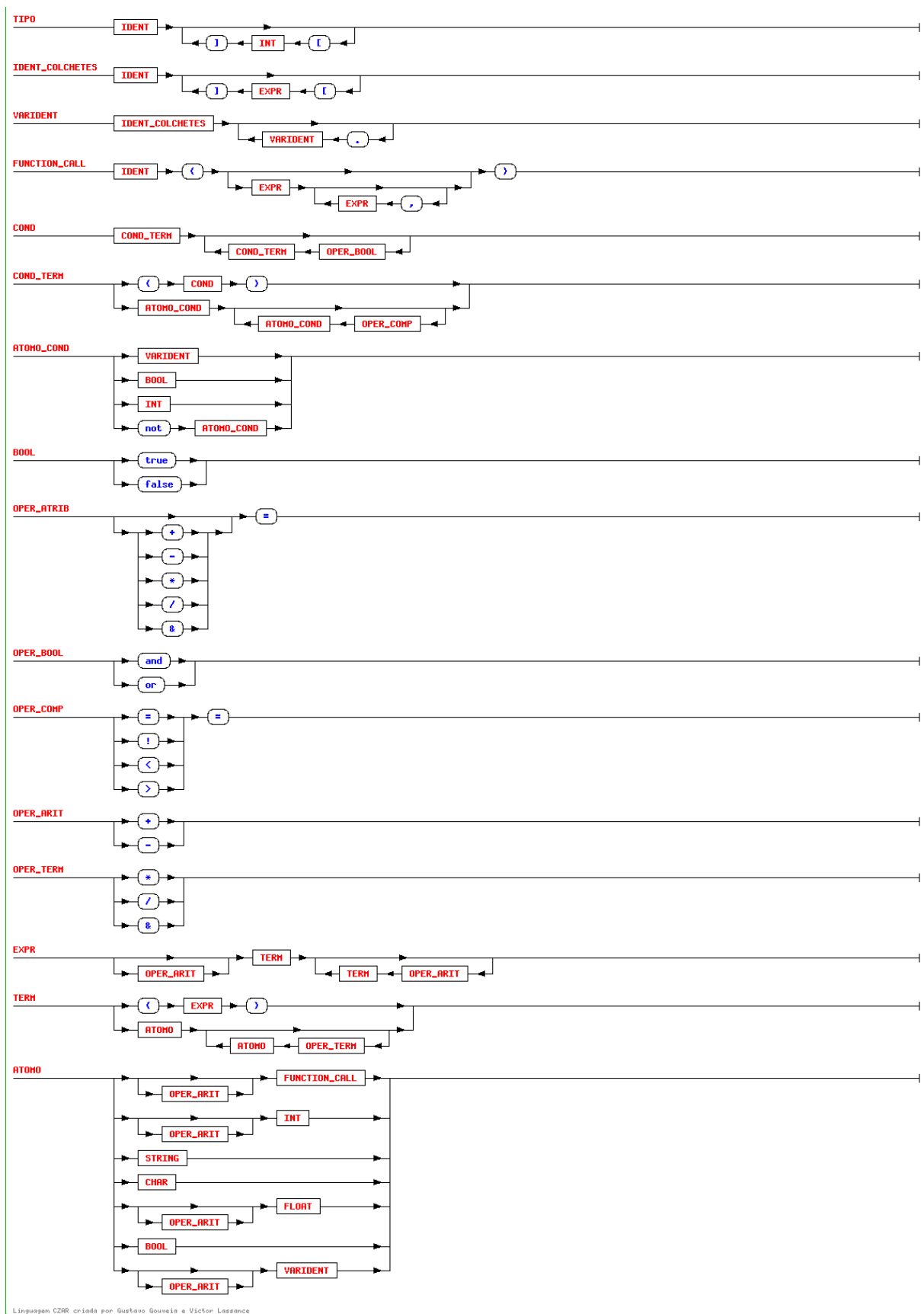


Figura 1 – Diagrama de Sintaxe da Linguagem CZAR

7 Conjunto das Palavras Reservadas

```
1  const
2  struct
3  ref
4  int
5  float
6  string
7  char
8  bool
9  for
10 while
11 if
12 else
13 and
14 or
15 not
16 true
17 false
18 main
19 return
20 void
```

8 Considerações Finais

O projeto do compilador é um projeto muito interessante, porém complexo. Desta forma, a divisão em etapas bem estruturadas permite o aprendizado e teste de cada uma das etapas. Em um primeiro momento, o foco foi no analisador léxico, o que permitiu realizar o *parse* do código e transformá-lo em tokens. Para a realização do analisador, tentamos pensar em permitir o processamento das principais classes de tokens, com o intuito de entender o funcionamento de um compilador de forma prática e didática.

Já na segunda etapa, começamos definindo a linguagem de forma mais livre e geral, partindo para a criação de exemplos de códigos escritos na nossa linguagem com todos os conceitos que deveriam ser implementados. A partir da definição informal e dos exemplos de código, criamos a definição formal na notação BNF, Wirth e com Diagramas de Sintaxe, além de atualizar a lista de palavras-chave. Essa etapa nos fez refletir sobre diversos detalhes de implementação que teremos que definir para o projeto, sendo, portanto, uma etapa crucial no desenvolvimento de um compilador.

Para as próximas etapas, espera-se continuar a atualizar o código e as definições descritas nesse documento quando for necessário, visando agregar os ensinamentos das próximas aulas.

Referências

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