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

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, denonimada 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[1], Wirth[2] 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 sintaxe[3], 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:
  - `PROGRAM = IMPORTS DECLS_GLOBAIS DEF_PROCS_FUNCS DEF_MAIN.`
- Inclusão de bibliotecas:
  - `IMPORTS = { IMPORT }.`
  - `IMPORT = ‘import’ ‘<’ IDENT ‘>’ { ‘,’ ‘<’ 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, que não devem incluir o procedimento principal (chamado main):
  - `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) - para fins de simplificação, a comunicação entre o programa e o ambiente externo deve ser feito através de arquivos, pois não haverá passagem de parâmetros para a função main:
  - `DEF_MAIN = ‘main’ ‘(’ ‘)’ ‘{’ [ BLOCO ] ‘}’.`

3 Exemplos de Programas na Linguagem

3.1 Exemplo Geral

```
1  import <math>;
2  import <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   if (a == 0) {
20       return b;
21   }
22   return a + b;
23 }
24
25 string concatena_chars(int n_chars, char[] caracteres) {
26   string retorno = "";
27   for(int i = 0; i < n_chars; i += 1) {
28       retorno += caracteres[i];
29   }
30   return retorno;
31 }
32
```

```
33 void proc_exemplo (char a, int b, int c, int d) {
34   int tmp;
35   char[32] buff;
36   soma_como_procedimento(b, c, tmp);
37   d = soma_como_funcao(tmp, c) + 5;
38   d = math.exp(SOU_CONSTANTE_INT, 2);
39   io.print(a);
40   io.int_to_str(d, buff);
41   io.print(" gives ");
42   io.print(buff);
43   io.print(" \n pointer to a is: ");
44   buff = a + "caracter";
45   io.print(buff);
46   io.print("bye");
47 }
48
49 void main () {
50   proc_exemplo('x', 3, -6, -15);
51 }
```

3.2 Exemplo Fatorial

```
1  import <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      if (n <= 1) {
9          return 1;
10     }
11     return n * fatorial_recursivo (n - 1);
12 }
13
14 int fatorial_iterativo(int n) {
15     int fatorial = 1;
16     while (n > 0) {
17         fatorial = fatorial * n;
```

```
18          n = n - 1;
19      }
20      return fatorial;
21  }
22
23  void main () {
24      retorno = fatorial_recursivo(fat_10_rec);
25
26      io.print_int(retorno);
27      io.print(" ");
28      io.print_int(fatorial_iterativo(fat_10_iter));
29  }
```

4 Descrição da Linguagem em BNF

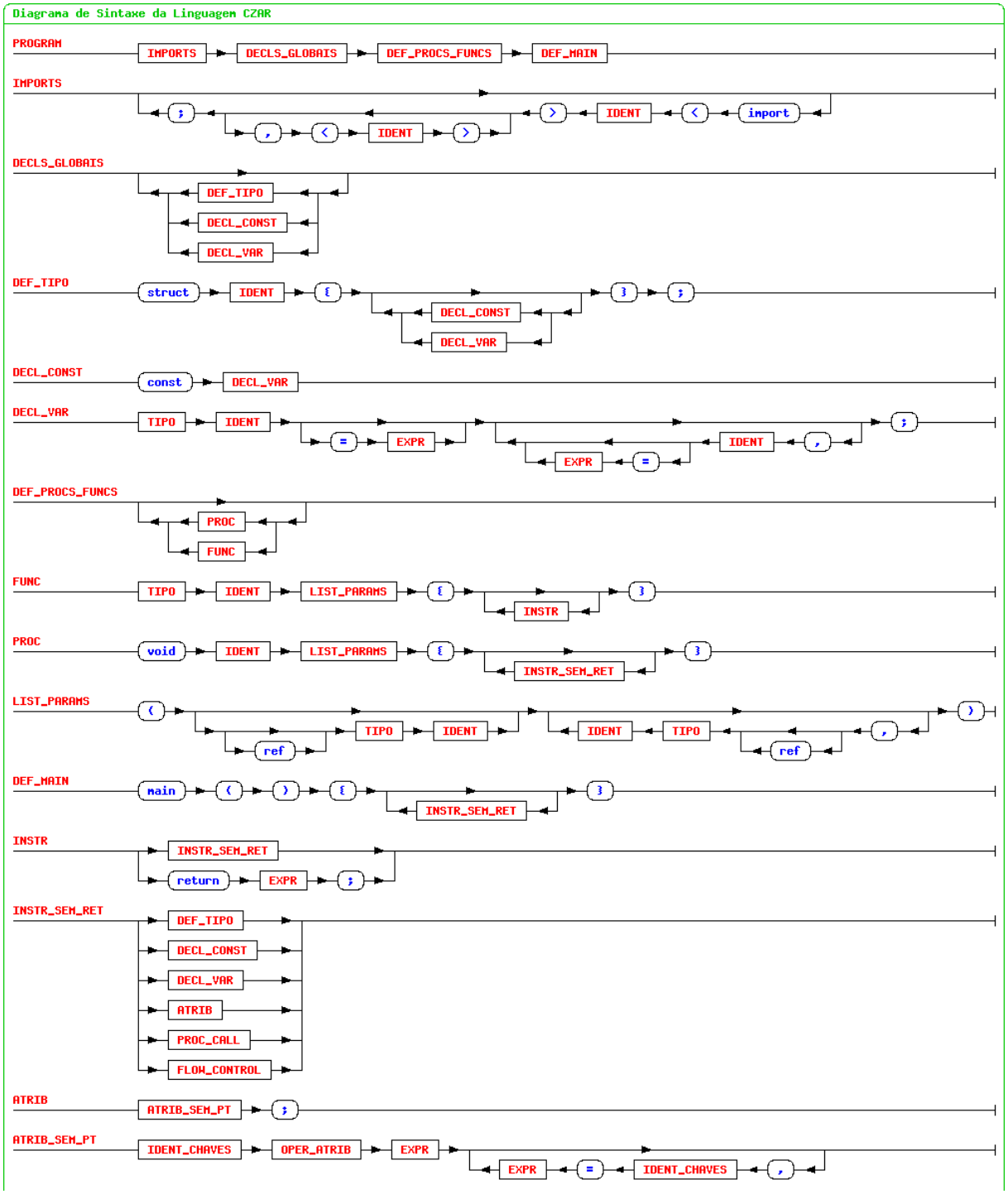
```
1 TODO
```

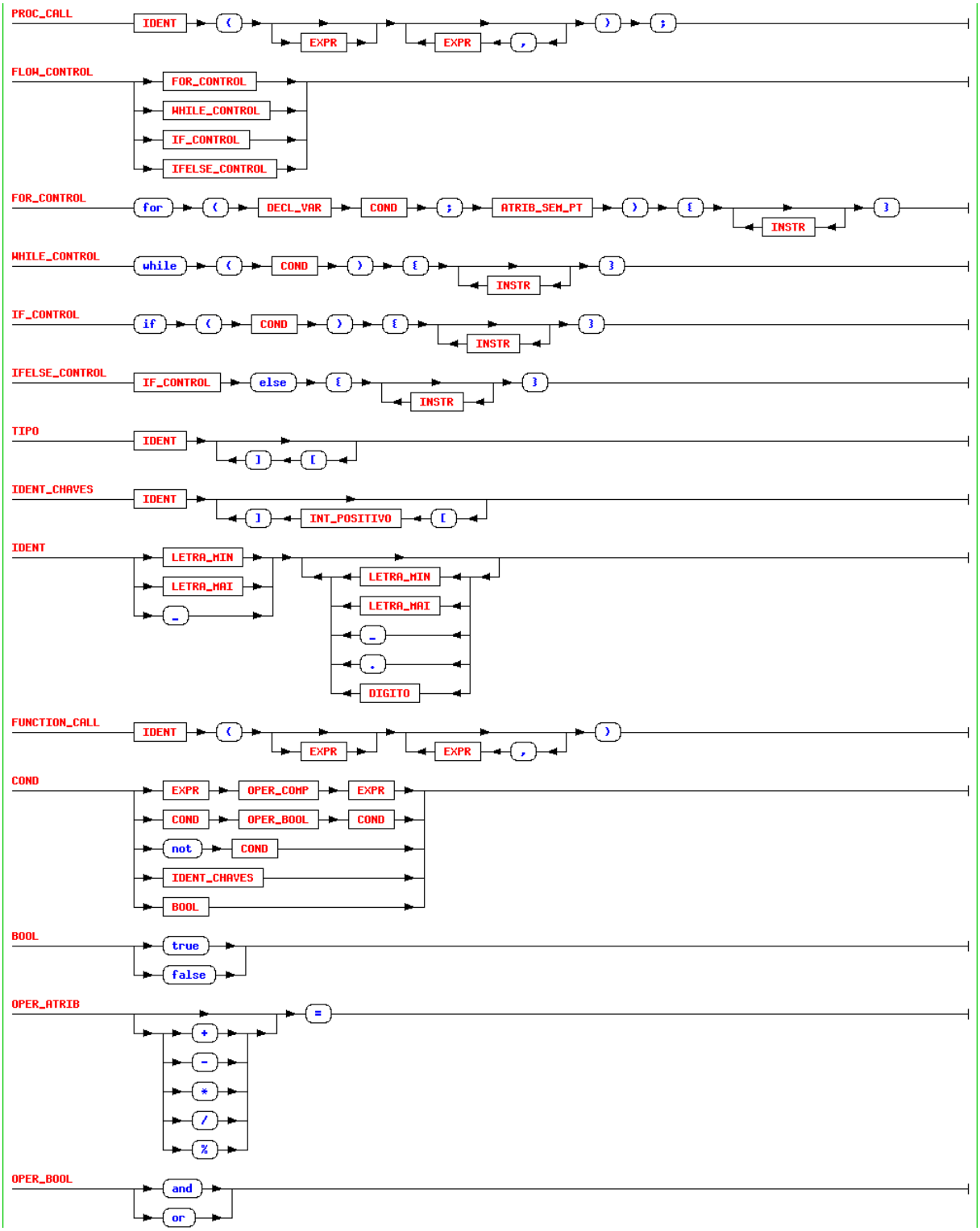
5 Descrição da Linguagem em Wirth

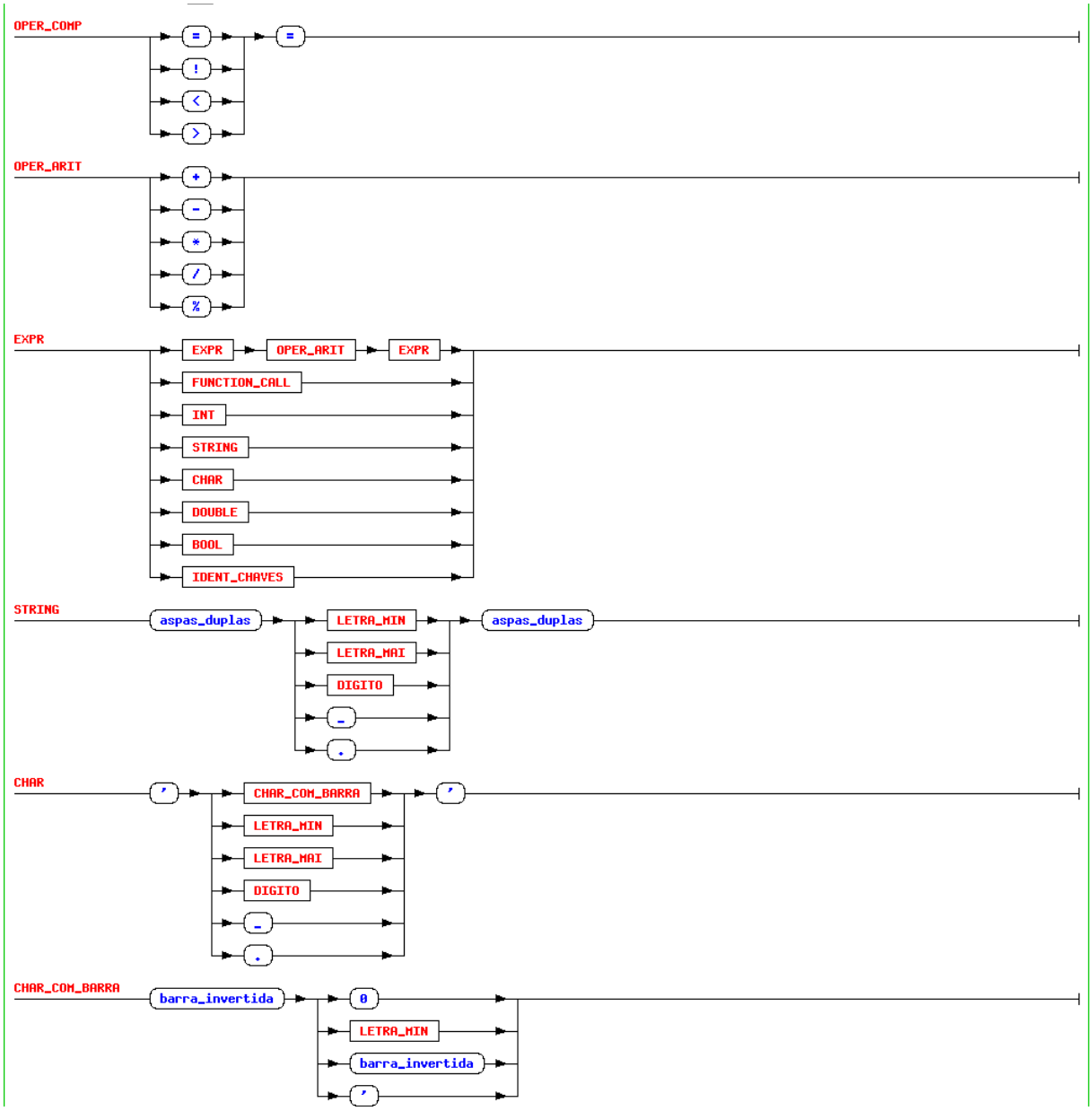
```
1  PROGRAM          = IMPORTS DECLS_GLOBAIS DEF_PROCS_FUNCS DEF_MAIN
       .
2  IMPORTS          = { "import" "<" IDENT ">" { "," "<" IDENT ">" }
       ";" }.
3  DECLS_GLOBAIS    = { DEF_TIPO | DECL_CONST | DECL_VAR }.
4  DEF_TIPO         = "struct" IDENT "{" { DECL_CONST | DECL_VAR }
       "}" ";".
5  DECL_CONST       = "const" DECL_VAR.
6  DECL_VAR         = TIPO IDENT [ "=" EXPR ] { "," IDENT [ "=" EXPR
       ] } ";".
7  DEF_PROCS_FUNCS  = { PROC | FUNC }.
8  FUNC             = TIPO IDENT LIST_PARAMS "{" { INSTR } "}".
9  PROC             = "void" IDENT LIST_PARAMS "{" { INSTR_SEM_RET }
       "}".
10 LIST_PARAMS      = "(" [ [ "ref" ] TIPO IDENT ] { "," [ "ref" ]
       TIPO IDENT } ")".
11 DEF_MAIN         = "main" "(" ")" "{" { INSTR_SEM_RET } "}".
12 INSTR            = INSTR_SEM_RET | ( "return" EXPR ";" ).
13 INSTR_SEM_RET    = DEF_TIPO | DECL_CONST | DECL_VAR | ATRIB |
       PROC_CALL | FLOW_CONTROL.
14 ATRIB            = ATRIB_SEM_PT ";".
15 ATRIB_SEM_PT     = IDENT_CHAVES OPER_ATRIB EXPR { ","
       IDENT_CHAVES "=" EXPR }.
16 PROC_CALL        = IDENT "(" [ EXPR ] { "," EXPR } ")" ";".
17 FLOW_CONTROL     = FOR_CONTROL | WHILE_CONTROL | IF_CONTROL |
       IFELSE_CONTROL.
18 FOR_CONTROL      = "for" "(" DECL_VAR COND ";" ATRIB_SEM_PT ")"
       "{" { INSTR } "}".
19 WHILE_CONTROL    = "while" "(" COND ")" "{" { INSTR } "}".
20 IF_CONTROL       = "if" "(" COND ")" "{" { INSTR } "}".
21 IFELSE_CONTROL   = IF_CONTROL "else" "{" { INSTR } "}".
22 TIPO             = IDENT { "[" "]" }.
23 IDENT_CHAVES     = IDENT { "[" INT_POSITIVO "]" }.
```

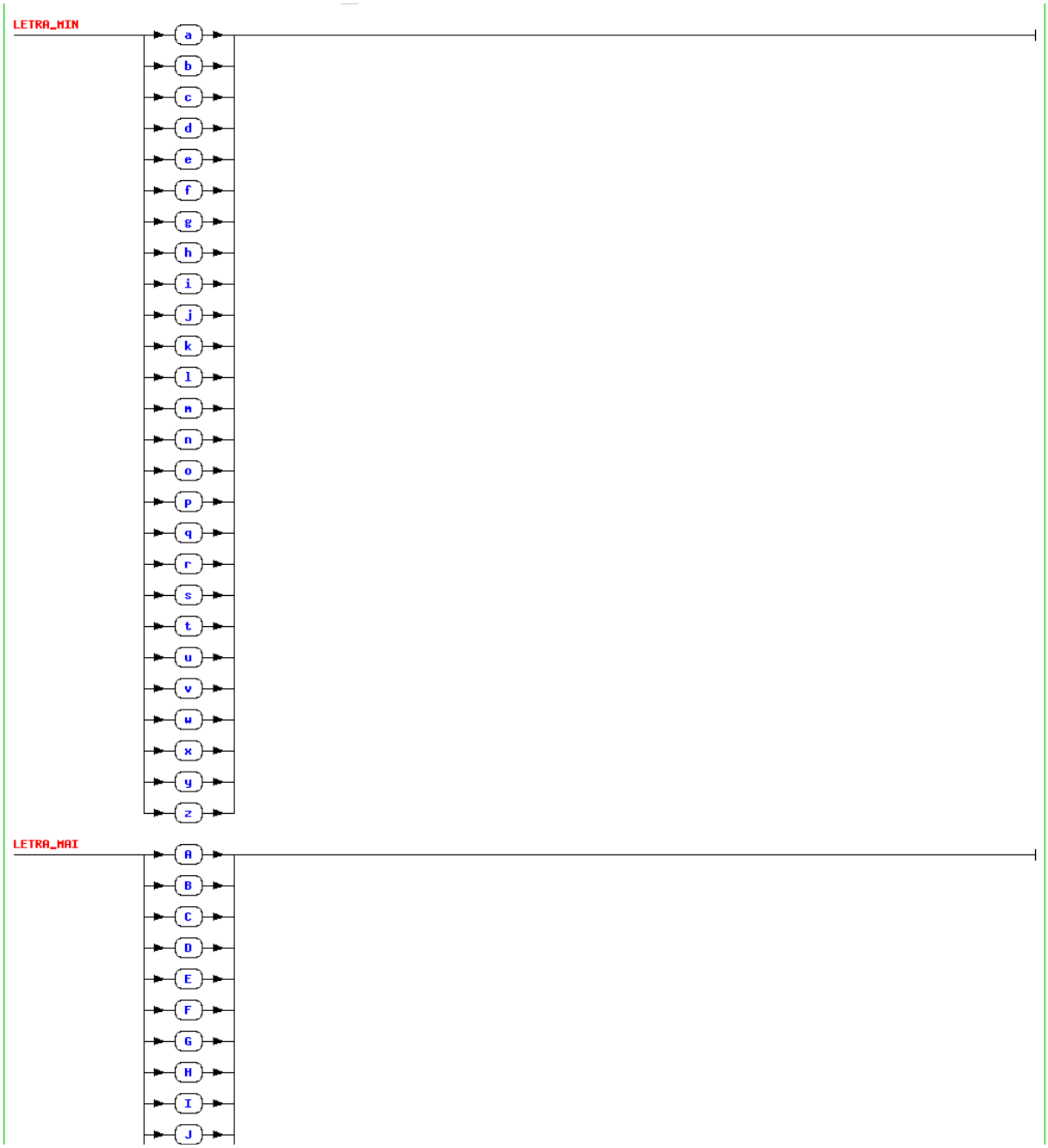
```
24 IDENT            = (LETRA_MIN | LETRA_MAI | "_") {LETRA_MIN |
       LETRA_MAI | "_" | "." | DIGITO}.
25 FUNCTION_CALL    = IDENT "(" [ EXPR ] { "," EXPR } ")".
26 COND             = (EXPR OPER_COMP EXPR) | COND OPER_BOOL COND |
       ("not" COND) | IDENT_CHAVES | BOOL.
27 BOOL             = "true" | "false".
28 OPER_ATRIB       = ["+" | "-" | "*" | "/" | "%"] "=".
29 OPER_BOOL        = "and" | "or".
30 OPER_COMP        = ("=" | "!" | "<" | ">") "=".
31 OPER_ARIT        = "+" | "-" | "*" | "/" | "%".
32 EXPR             = (EXPR OPER_ARIT EXPR) | FUNCTION_CALL | INT |
       STRING | CHAR | DOUBLE | BOOL | IDENT_CHAVES.
33 STRING           = "\"" (LETRA_MIN | LETRA_MAI | DIGITO | "_" |
       ".") "\"".
34 CHAR             = "'" (CHAR_COM_BARRA | LETRA_MIN | LETRA_MAI |
       DIGITO | "_" | ".") "'".
35 CHAR_COM_BARRA   = "\\" ("0" | LETRA_MIN | "\\" | "'").
36 LETRA_MIN        = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h"
       | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
        "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z".
37 LETRA_MAI        = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H"
       | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
        "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z".
38 INT              = [ "-" ] INT_POSITIVO.
39 INT_POSITIVO     = DIGITO { DIGITO }.
40 FLOAT            = [ INT ] "." INT_POSITIVO.
41 DIGITO           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"
       | "8" | "9".
```

6 Diagrama de Sintaxe da Linguagem




PROC_CALL
IDENT
(
EXPR
EXPR
,
)
;
FLOW_CONTROL
FOR_CONTROL
WHILE_CONTROL
IF_CONTROL
IFELSE_CONTROL
FOR_CONTROL
for
(
DECL_VAR
COND
;
ATRIB_SEM_PT
)
{
INSTR
}
WHILE_CONTROL
while
(
COND
)
{
INSTR
}
IF_CONTROL
if
(
COND
)
{
INSTR
}
IFELSE_CONTROL
IF_CONTROL
else
{
INSTR
}
TIPO
IDENT
]
[
IDENT_CHAVES
IDENT
]
INT_POSITIVO
[
IDENT
LETRA_MIN
LETRA_MAI
_
LETRA_MIN
LETRA_MAI
_
.
DIGITO
FUNCTION_CALL
IDENT
(
EXPR
EXPR
,
)
COND
EXPR
OPER_COMP
EXPR
COND
OPER_BOOL
COND
not
COND
IDENT_CHAVES
BOOL
BOOL
true
false
OPER_ATRIB
=
+
-
*
/
%
OPER_BOOL
and
or


OPER_COMP
=
!
<
>
=
OPER_ARIT
+
-
*
/
%
EXPR
EXPR
OPER_ARIT
EXPR
FUNCTION_CALL
INT
STRING
CHAR
DOUBLE
BOOL
IDENT_CHAVES
STRING
aspas_duplas
LETRA_MIN
LETRA_MAI
DIGITO
_
.
aspas_duplas
CHAR
'
CHAR_COM_BARRA
LETRA_MIN
LETRA_MAI
DIGITO
_
.
'
CHAR_COM_BARRA
barra_invertida
0
LETRA_MIN
barra_invertida
'


LETRA_MIN
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
LETRA_MAI
A
B
C
D
E
F
G
H
I
J

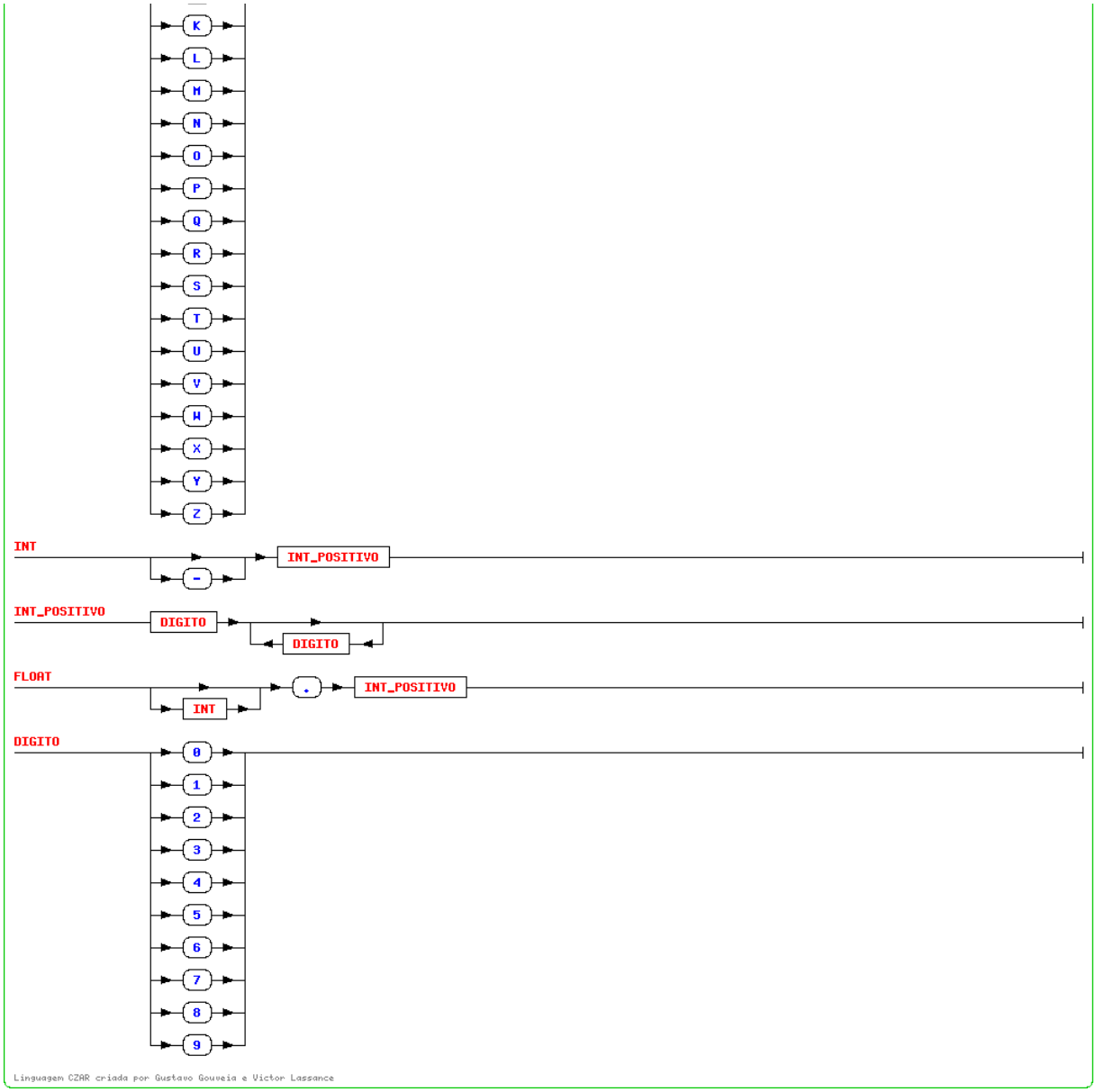


Figura 1 – Diagrama de Sintaxe da Linguagem CZAR

7 Conjunto das Palavras Reservadas

```
1  import
2  const
3  struct
4  ref
5  int
6  float
7  string
8  char
9  bool
10 for
11 while
12 if
13 else
14 and
15 or
16 not
17 true
18 false
19 main
20 return
21 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).