



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO

Construção de Compiladores
Av. João Naves de Ávila 2121, Campus Santa Mônica



Construção de Compiladores Portugol para CLR

Alunos: Eduardo Costa de Paiva/Fausto H. N. Silva

Matrícula: 11221BCC012/11211BCC030

Email: eduardocspv@gmail.com/faustohenriquens@hotmail.com

Prof^o: Alexsandro Santos Soares

Sumário

1	Introdução	4
2	Configuração do Ambiente	4
2.1	OCaml	4
2.2	CLR	4
2.3	CIL	5
2.4	Mono	5
2.4.1	Instalação Mono	6
2.4.2	Compilando no Mono	6
2.4.3	Executando no Mono	6
2.4.4	Gerar assembly a partir de um executável no Mono . .	6
2.4.5	Gerar executável a partir de um assembly no Mono . .	7
2.5	Portugol	8
3	Códigos	8
3.1	Exemplo de código em C # e seu equivalente em assembly . .	8
3.2	Nano 01	10
3.3	Nano 02	11
3.4	Nano 03	11
3.5	Nano 04	12
3.6	Nano 05	13
3.7	Nano 06	13
3.8	Nano 07	14
3.9	Nano 08	15
3.10	Nano 09	16
3.11	Nano 10	17
3.12	Nano 11	18
3.13	Nano 12	19
3.14	Micro 01	21
3.15	Micro 02	22
3.16	Micro 03	24
3.17	Micro 04	25
3.18	Micro 05	27
3.19	Micro 06	29
3.20	Micro 07	31

SUMÁRIO

3.21	Micro 08	33
3.22	Micro 09	34
3.23	Micro 10	36
3.24	Micro 11	38
4	Analizador Léxico	42
4.1	Autômato Finito Determinístico	42
4.2	Implementação em Ocaml	44
4.3	Execução	49
4.4	Analizador Léxico para linguagem Portugol	49
4.4.1	Execução	55
5	Analizador sintático preditivo	59
5.1	Códigos	60
6	Analizador sintático para a linguagem Portugol	64
6.1	Instalação Menhir	64
6.2	Códigos do Parser e da Árvore Sintática	65
6.3	Compilando e Executando	73
7	Tratamento de Erros	75
7.1	Gerando arquivo de mensagens de erro	75
7.2	Arquivo completo com as mensagens de erro	75
7.3	Gerando arquivo .ml com as mensagens de erro	139
7.4	Construindo analisador sintático com mensagens de erro	139
7.5	Exemplo de mensagem de erro	140
8	Analizador Semântico	140
8.1	Árvore sintática	140
8.2	Definição de uma expressão para árvore sintática	142
8.3	Definição de uma expressão para árvore semântica	143
8.4	Definição do Ambiente	143
8.5	Definição da tabela de símbolos	144
8.6	Definição do analisador semântico	145
9	Interpretador	156
9.1	Código do interpretador	156
9.2	Ambiente	166

SUMÁRIO

10 Bibliografia	168
------------------------	------------

1 Introdução

Esse relatório tem como objetivo a apresentação das tecnologias que serão utilizadas para a construção de um compilador para a linguagem Portugal utilizando a máquina virtual Common Language Runtime(CLR) da plataforma Microsoft .NET.

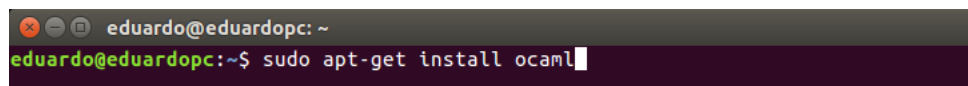
O sistema operacional que será utilizado é o Ubuntu 16.04 e a linguagem para o desenvolvimento será Ocaml.

2 Configuração do Ambiente

Nesta seção, serão apresentadas as ferramentas que serão utilizadas e como configurá-las para o experimento.

2.1 OCaml

Para instalar a linguagem OCaml no Ubuntu, basta digitar no terminal:

A screenshot of a terminal window with a dark background. The prompt is 'eduardo@eduardopc: ~'. The command 'sudo apt-get install ocaml' is entered and followed by a cursor. The terminal has standard window control buttons in the top left corner.

```
eduardo@eduardopc: ~  
eduardo@eduardopc:~$ sudo apt-get install ocaml
```

Figura 1: Instalando Ocaml no Ubuntu

Nas demais distribuições do Linux o procedimento é semelhante. Recomenda-se ler a documentação do Ocaml.

2.2 CLR

A Common Language Runtime (CLR) é a máquina virtual do framework Microsoft .NET, que gerencia a execução dos programas .NET. Através de um processo de compilação Just In Time (JIT), a CLR converte o código compilado para instruções de máquina que possam ser executadas pela CPU do computador.

2 CONFIGURAÇÃO DO AMBIENTE

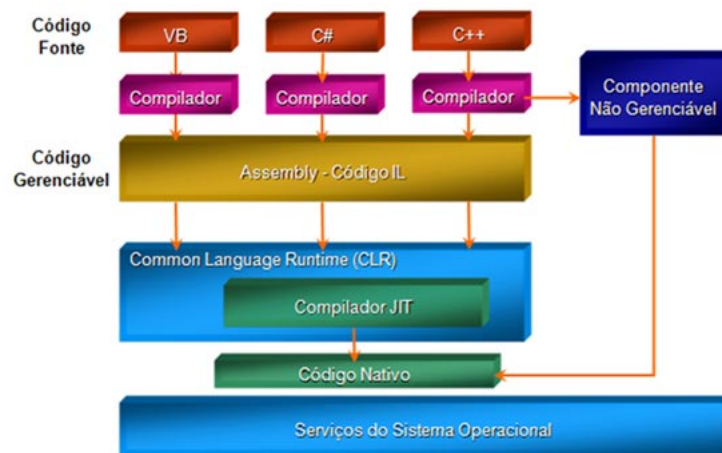


Figura 2: Processo de Compilação utilizando CLR

Algumas características da CLR são:

- Garbage Collector
- Facilidade para utilizar componentes desenvolvidos em outras linguagens

2.3 CIL

A Common Intermediate Language (CIL), anteriormente chamada de Microsoft Intermediate Language (MSIL), é uma linguagem de programação de baixo nível definida pela Common Language Infrastructure (CLI), que é utilizada pelo Framework .NET e Mono.

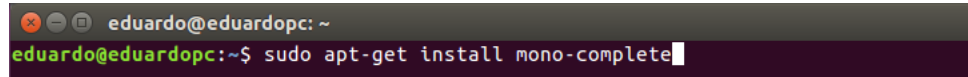
2.4 Mono

Mono é uma plataforma de desenvolvimento Open Source baseada no framework .NET que permite o desenvolvimento de aplicações multiplataforma. A implementação do Mono segue os padrões ECMA para C# e Common Language Infrastructure (CLI).

Como neste projeto o sistema operacional utilizado é o Ubuntu 16.04 e o framework .NET é uma ferramenta exclusiva da Microsoft, iremos utilizar o Mono.

2 CONFIGURAÇÃO DO AMBIENTE

2.4.1 Instalação Mono



```
eduardo@eduardopc: ~  
eduardo@eduardopc:~$ sudo apt-get install mono-complete
```

Figura 3: Instalando Mono no Ubuntu

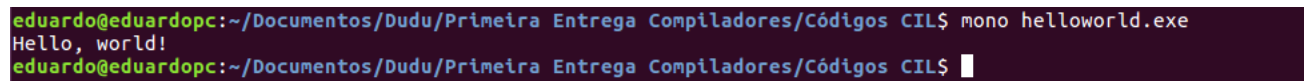
2.4.2 Compilando no Mono



```
eduardo@eduardopc: ~  
eduardo@eduardopc:~$ mcs helloworld.cs
```

Figura 4: Compilando Mono no Ubuntu

2.4.3 Executando no Mono



```
eduardo@eduardopc:~/Documentos/Dudu/Primeira Entrega Compiladores/Códigos CIL$ mono helloworld.exe  
Hello, world!  
eduardo@eduardopc:~/Documentos/Dudu/Primeira Entrega Compiladores/Códigos CIL$
```

Figura 5: Executando Mono no Ubuntu

2.4.4 Gerar assembly a partir de um executável no Mono

2 CONFIGURAÇÃO DO AMBIENTE

```
eduardo@eduardopc:~/Documentos/Dudu/Primeira Entrega Compiladores/Códigos CIL$ monodis nano01.exe
.assembly extern mscorlib
{
    .ver 0:0:0:0
}
.assembly 'nano01'
{
    .hash algorithm 0x00000000
    .ver 0:0:0:0
}
.module nano01.exe // GUID = {02F22A31-7189-4E36-8C90-B4388D1E2030}

    // method line 1
    .method private scope static
        default void Main () cil managed
    {
        // Method begins at RVA 0x2050
        .entrypoint
        // Code size 1 (0x1)
        .maxstack 8
        IL_0000: ret
    } // end of global method Main
```

Figura 6: Gerando assembly a partir de um executável no Mono.

2.4.5 Gerar executável a partir de um assembly no Mono

```
eduardo@eduardopc:~$ ilasm
Mono ILasm compiler
ilasm [options] source-files
  --about          About the Mono ILasm compiler
  --version        Print the version number of the Mono ILasm compiler
  /output:file_name Specifies output file.
  /exe             Compile to executable.
  /dll             Compile to library.
  /debug           Include debug information.
  /key:keyfile     Strongname using the specified key file
  /key:@container  Strongname using the specified key container
Options can be of the form -option or /option
```

Figura 7: O comando ilasm

```
eduardo@eduardopc:~/Documentos/Dudu/Primeira Entrega Compiladores/Códigos CIL$ ilasm nano01.il
Assembling 'nano01.il' , no listing file, to exe --> 'nano01.exe'
Operation completed successfully
```

Figura 8: Gerando executável a partir de um assembly .il no Mono

3 CÓDIGOS

2.5 Portugol

Portugol é uma linguagem de programação totalmente em português, muito usada para ensinar iniciantes em computação. Por ser uma linguagem com comandos em português o aluno não necessita de saber o inglês para programar, dessa forma o aluno foca na lógica de programação e não se perde nos comandos.

3 Códigos

3.1 Exemplo de código em C # e seu equivalente em assembly

Listing 1: Atribuição de duas operações aritméticas sobre inteiros a uma variável(C#)

```
1 using System;
2
3 class nano09{
4
5     public static void Main()
6     {
7         int n;
8         n = 1+1/2;
9         if(n==1)
10        {
11            Console.WriteLine(n);
12        }
13        else
14        {
15            Console.WriteLine(0);
16        }
17
18
19 }
20
21 }
```

Listing 2: Atribuição de duas operações aritméticas sobre inteiros a uma variável(Código em CIL gerado pelo comando monodis)

```
1 .assembly extern mscorlib
```

3 CÓDIGOS

```
2 {
3   .ver 4:0:0:0
4   .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'nano09'
7 {
8   .custom instance void class [mscorlib]System.Runtime.
      CompilerServices.RuntimeCompatibilityAttribute::.ctor
      '() = (
9     01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78    // ....
      T..WrapNonEx
10    63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01        ) //
      ceptionThrows.
11
12   .hash algorithm 0x00008004
13   .ver 0:0:0:0
14 }
15 .module nano09.exe // GUID = {B11A0372-0BF1-40E0-B512-30
      D05E235D9B}
16
17
18 .class private auto ansi beforefieldinit nano09
19   extends [mscorlib]System.Object
20 {
21
22   // method line 1
23   .method public hidebysig specialname rtspecialname
24     instance default void '.ctor' () cil managed
25   {
26     // Method begins at RVA 0x2050
27     // Code size 7 (0x7)
28     .maxstack 8
29     IL_0000: ldarg.0
30     IL_0001: call instance void object::.ctor'()
31     IL_0006: ret
32   } // end of method nano09::.ctor
33
34   // method line 2
35   .method public static hidebysig
36     default void Main () cil managed
37   {
38     // Method begins at RVA 0x2058
39     .entrypoint
40     // Code size 27 (0x1b)
41     .maxstack 2
```

3 CÓDIGOS

```
42 .locals init (
43     int32 V_0)
44 IL_0000: ldc.i4.1
45 IL_0001: stloc.0
46 IL_0002: ldloc.0
47 IL_0003: ldc.i4.1
48 IL_0004: bne.un IL_0014
49
50 IL_0009: ldloc.0
51 IL_000a: call void class [mscorlib]System.Console::
    WriteLine(int32)
52 IL_000f: br IL_001a
53
54 IL_0014: ldc.i4.0
55 IL_0015: call void class [mscorlib]System.Console::
    WriteLine(int32)
56 IL_001a: ret
57 } // end of method nano09::Main
58
59 } // end of class nano09
```

3.2 Nano 01

Listing 3: Módulo mínimo que caracteriza um programa (Portugol)

```
1 algoritmo "nano01"
2 var
3 inicio
4 fim_algoritmo
```

Listing 4: Módulo mínimo que caracteriza um programa (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano01 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     ret
8 }
```

Podemos notar que o código em CIL inicia chamando a biblioteca `mscorlib` (**M**icrosoft **C**ommon **O**bject **R**untime). Essa biblioteca é a principal biblioteca ao utilizar o framework `.NET`. O comando `.entrypoint` marca o

3 CÓDIGOS

início do programa e o comando `ret` marca o fim.

3.3 Nano 02

Listing 5: Declaração de uma variável (Portugol)

```
1 algoritmo "nano02"
2 var
3   n: inteiro
4 inicio
5 fim_algoritmo
```

Listing 6: Declaração de uma variável (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano02 {}
3
4 .method static void Main() cil managed
5 {
6   .entrypoint
7   .maxstack 1
8   .locals init ( int32 )
9   ret
10 }
```

3.4 Nano 03

Listing 7: Atribuição de um inteiro a uma variável (Portugol)

```
1 algoritmo "nano03"
2 var
3   n: inteiro
4 inicio
5   n <- 1
6 fim_algoritmo
```

Listing 8: Atribuição de um inteiro a uma variável (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano03 {}
3
4 .method static void Main() cil managed
5 {
6   .entrypoint
7   .maxstack 1
```

3 CÓDIGOS

```
8  .locals init ( int32 )
9  ldc.i4 1
10 stloc.0
11 ret
12 }
```

O comando **ldc.i4** coloca o número 1 do tipo **int32** na pilha. O comando **stloc.0** tira da pilha e guarda na variável local 0.

3.5 Nano 04

Listing 9: Atribuição de uma soma de inteiros a uma variável (Portugol)

```
1 algoritmo "nano04"
2 var
3   n: inteiro
4 inicio
5   n <- 1 + 2
6 fim_algoritmo
```

Listing 10: Atribuição de uma soma de inteiros a uma variável (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano04 {}
3
4 .method static void Main() cil managed
5 {
6   .entrypoint
7   .maxstack 3
8   .locals init (int32, int32, int32)
9   ldc.i4 1
10  stloc.0
11  ldc.i4 2
12  stloc.1
13
14  ldloc.0
15  ldloc.1
16  add
17  stloc.2
18
19  ret
20 }
```

O comando **ldloc** carrega uma determinada variável local na pilha. O comando **add** retira as duas variáveis da pilha, executa a soma e retorna o

3 CÓDIGOS

resultado para a pilha. O comando **stloc.2** carrega o resultado na variável local 2.

3.6 Nano 05

Listing 11: Inclusão do comando de impressão (Portugol)

```
1 algoritmo "nano05"
2 var
3   n : inteiro
4 inicio
5   n <- 2
6   escreva(n)
7 fim_algoritmo
```

Listing 12: Inclusão do comando de impressão (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano05 {}
3
4 .method static void Main() cil managed
5 {
6   .entrypoint
7   .maxstack 1
8   .locals init (int32)
9   ldc.i4 2
10
11   call void [mscorlib]System.Console::WriteLine(int32)
12
13   ret
14 }
```

3.7 Nano 06

Listing 13: Atribuição de uma subtração de inteiros a uma variável (Portugol)

```
1 algoritmo "nano06"
2 var
3   n : inteiro
4 inicio
5   n <- 1 - 2
6   escreva(n)
7 fim_algoritmo
```

3 CÓDIGOS

Listing 14: Atribuição de uma subtração de inteiros a uma variável (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano06 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32, int32, int32)
9     ldc.i4 1
10    stloc.0
11    ldc.i4 2
12    stloc.1
13
14    ldloc.0
15    ldloc.1
16    sub
17    stloc.2
18
19    ldloc.2
20    call void [mscorlib]System.Console::WriteLine(int32)
21
22    ret
23 }
```

3.8 Nano 07

Listing 15: Inclusão do comando condicional (Portugol)

```
1 algoritmo "nano07"
2 var
3     n : inteiro
4 inicio
5     n <- 1
6     se n = 1 entao
7         escreva(n)
8     fim_se
9 fim_algoritmo
```

Listing 16: Inclusão de comando condicional (CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano07 {}
3
```

3 CÓDIGOS

```
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 2
8     .locals init (int32, int32)
9     ldc.i4 1
10    stloc.0
11    ldc.i4 1
12    stloc.1
13    ldloc.0
14    ldloc.1
15    beq IGUAL
16    br NAOIGUAL
17    IGUAL:
18        ldloc.0
19        call void [mscorlib]System.Console::WriteLine(int32)
20    NAOIGUAL:
21        ret
22 }
```

3.9 Nano 08

Listing 17: Inclusão do comando condicional com parte senão(Portugol)

```
1 algoritmo "nano08"
2 var
3     n : inteiro
4 inicio
5     n <- 1
6     se n = 1 entao
7         escreva(n)
8     senao
9         escreva(0)
10    fim_se
11 fim_algoritmo
```

Listing 18: Inclusão do comando condicional com parte senão(CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano08 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 2
```


3 CÓDIGOS

```
8  .locals init (int32, int32)
9  ldc.i4 1
10 stloc.0
11 ldloc.0
12
13 ldc.i4 1
14 beq IGUAL
15 br NAOIGUAL
16 IGUAL:
17  ldloc.0
18  call void [mscorlib]System.Console::WriteLine(int32)
19  br FIM
20 NAOIGUAL:
21  ldc.i4 0
22  call void [mscorlib]System.Console::WriteLine(int32)
23  FIM:
24  ret
25 }
```

O programa acima funciona da seguinte maneira: Inicialmente usamos o comando **ldc** para colocar o número 1 na pilha e guardamos na variável local 0, **stloc.0**, para representar nossa variável **n**. Após isso carregamos essa variável na pilha e colocamos o valor 1 também nesta pilha para fazermos uma comparação com o comando **beq** (branch if equal). Se esses dois valores forem iguais printamos o que está em **ldloc.0**, que é nossa variável **n**. Se não forem iguais printamos o valor 0, pois colocamos esse valor na pilha com o comando **ldc.i4 0**.

3.10 Nano 09

Listing 19: Atribuição de duas operações aritméticas sobre inteiros a uma variável(Portugol)

```
1 algoritmo "nano09"
2 var
3   n : inteiro
4 inicio
5   n <- 1 + 1 / 2
6   se n = 1 entao
7     escreva(n)
8   senao
9     escreva(0)
10  fim_se
11 fim_algoritmo
```

3 CÓDIGOS

Listing 20: Atribuição de duas operações aritméticas sobre inteiros a uma variável(CIL)

```
1
2 .assembly extern mscorlib {}
3 .assembly nano09 {}
4
5 .method static void Main() cil managed
6 {
7     .entrypoint
8     .maxstack 2
9     .locals init (float32, float32)
10    ldc.r4 1
11    ldc.r4 2
12    div
13    ldc.r4 1
14    add
15
16    stloc.0
17    ldloc.0
18
19    ldc.r4 1
20    beq IGUAL
21    br NAOIGUAL
22    IGUAL:
23        ldloc.0
24        call void [mscorlib]System.Console::WriteLine(float32)
25        br FIM
26    NAOIGUAL:
27        ldc.r4 0
28        call void [mscorlib]System.Console::WriteLine(float32)
29    FIM:
30        ret
31 }
```

3.11 Nano 10

Listing 21: Atribuição de variáveis inteiras(Portugol)

```
1 algoritmo "nano10"
2 var
3     n, m : inteiro
4 inicio
5     n <- 1
6     m <- 2
7     se n = m entao
```

3 CÓDIGOS

```
8     escreva(n)
9     senao
10     escreva(0)
11     fim_se
12 fim_algoritmo
```

Listing 22: Atribuição de variáveis inteiras(CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano10 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 2
8     .locals init (int32, int32)
9     ldc.i4 1
10    stloc.0
11    ldc.i4 2
12    stloc.1
13    ldloc.0
14    ldloc.1
15
16    beq IGUAL
17    ldc.i4 0
18    call void [mscorlib]System.Console::WriteLine(int32)
19    br FIM
20    IGUAL:
21    ldloc 0
22    call void [mscorlib]System.Console::WriteLine(int32)
23    FIM:
24    ret
25
26 }
```

3.12 Nano 11

Listing 23: Introdução do comando de repetição enquanto(Portugol)

```
1 algoritmo "nano11"
2 var
3     n, m, x : inteiro
4 inicio
5     n <- 1
6     m <- 2
```

3 CÓDIGOS

```
7  x <- 5
8  enquanto x > n faça
9      n <- n + m
10     escreva(n)
11     fim_enquanto
12 fim_algoritmo
```

Listing 24: Introdução do comando de repetição enquanto(CIL)

```
1  .assembly extern mscorlib {}
2  .assembly nano11 {}
3
4  .method static void Main() cil managed
5  {
6      .entrypoint
7      .maxstack 3
8      .locals init (int32,int32,int32)
9      ldc.i4 1 //n
10     stloc.0
11     ldc.i4 2 //m
12     stloc.1
13     ldc.i4 5 //x
14     stloc.2
15     LOOP:
16         ldloc.2
17         ldloc.0
18         ble FIM
19
20         ldloc.0
21         ldloc.1
22         add
23         stloc.0
24         ldloc.0
25         call void [mscorlib]System.Console::WriteLine (int32)
26         br LOOP
27     FIM:
28         ret
29 }
```

3.13 Nano 12

Listing 25: Comando condicional aninhando em um comando de repetição(Portugol)

```
1 algoritmo "nano12"
```

3 CÓDIGOS

```
2 var
3     n, m, x : inteiro
4 inicio
5     n <- 1
6     m <- 2
7     x <- 5
8     enquanto x > n faca
9         se n = m entao
10             escreva(n)
11         senao
12             escreva(0)
13         fim_se
14     x <- x - 1
15 fim_enquanto
```

Listing 26: Comando condicional aninhando em um comando de repetição(CIL)

```
1 .assembly extern mscorlib {}
2 .assembly nano12 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32,int32,int32)
9     ldc.i4 1
10    stloc.0
11    ldc.i4 2
12    stloc.1
13    ldc.i4 5
14    stloc.2
15    LOOP:
16        ldloc.2
17        ldloc.0
18        ble FIM
19
20        ldloc.0
21        ldloc.1
22        beq SE_N_IGUAL_M
23        ldc.i4 0
24        call void [mscorlib]System.Console::WriteLine (int32)
25        br DECREMENTAR_X
26
27    SE_N_IGUAL_M:
```

3 CÓDIGOS

```
28     ldloc.0
29     call void [mscorlib]System.Console::WriteLine (int32)
30     br DECREMENTAR_X
31
32     DECREMENTAR_X:
33     ldloc.2
34     ldc.i4 1
35     sub
36     stloc.2
37     br LOOP
38     FIM:
39     ret
40 }
```

3.14 Micro 01

Listing 27: Converte graus Celsius para Fahrenheit(Portugol)

```
1 algoritmo "micro01"
2 /*
3 Funcao: Ler uma temperatura em graus Celsius e apresenta-la
4 convertida em graus Fahrenheit. A formula de conversao e:
5 F=(9*C+160) / 5,
6 sendo F a temperatura em Fahrenheit e C a temperatura em
7 Celsius.
8 */
9 var
10     cel, far: real
11 inicio
12     escreval("Tabela de conversao: Celsius -> Fahrenheit")
13     escreva("Digite a temperatura em Celsius: ")
14     leia(cel)
15     far <- (9*cel+160)/5
16     escreval("A nova temperatura e: ",far," F")
17 fim_algoritmo
```

Listing 28: Converte graus Celsius para Fahrenheit(CIL)

```
1 .assembly extern mscorlib {}
2 .assembly micro01 {}
3
4 .method static void Main() cil managed
5 {
6
7     .entrypoint
```

3 CÓDIGOS

```
8  .maxstack 10
9  .locals init (float32,float32)
10
11  ldstr "Tabela de Conversao: Celsius -> Fahrenheit."
12  call void [mscorlib]System.Console::WriteLine (string)
13  ldstr "Escreva a temperatura em Celsius: "
14  call void [mscorlib]System.Console::WriteLine (string)
15  call string [mscorlib]System.Console::ReadLine ()
16  call float32 [mscorlib]System.Single::Parse(string)
17  stloc.0
18  ldloc.0
19  ldc.r4 9
20  mul
21  ldc.r4 160
22  add
23  ldc.r4 5
24  div
25  stloc.1
26  ldstr "A nova temperatura eh: "
27  call void [mscorlib]System.Console::WriteLine (string)
28  ldloc.1
29  call void [mscorlib]System.Console::WriteLine (float32)
30  ret
31 }
```

3.15 Micro 02

Listing 29: Ler dois inteiros e decide qual é maior(Portugol)

```
1 algoritmo "micro02"
2
3 var
4   num1, num2: inteiro
5 inicio
6   escreva("Digite o primeiro numero: ")
7   leia(num1)
8   escreva("Digite o segundo numero: ")
9   leia(num2)
10  se num1 > num2 entao
11    escreva("O primeiro numero ",num1," e maior que o segundo
12    ",num2)
13  senao
14    escreva("O segundo numero",num2," e maior que o primeiro"
15    ,num1)
16 fim_se
```

3 CÓDIGOS

15 fim_algoritmo

Listing 30: Ler dois inteiros e decide qual é maior(CIL)

```
1 .assembly extern mscorlib {}
2 .assembly micro02 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 2
8     .locals init (int32,int32)
9     ldstr "Escreva o primeiro numero: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13    stloc.0
14    ldstr "Escreva o segundo numero: "
15    call void [mscorlib]System.Console::WriteLine (string)
16    call string [mscorlib]System.Console::ReadLine ()
17    call int32 [mscorlib]System.Int32::Parse(string)
18    stloc.1
19    ldloc.0
20    ldloc.1
21    bgt PRIMEIROMAIORSEGUNDO
22    ldstr "Segundo numero eh maior que o primeiro!"
23    call void [mscorlib]System.Console::WriteLine (string)
24    ldstr "Primeiro numero: "
25    call void [mscorlib]System.Console::WriteLine (string)
26    ldloc.0
27    call void [mscorlib]System.Console::WriteLine (int32)
28    ldstr "Segundo numero: "
29    call void [mscorlib]System.Console::WriteLine (string)
30    ldloc.1
31    call void [mscorlib]System.Console::WriteLine (int32)
32    br FIM
33    PRIMEIROMAIORSEGUNDO:
34    ldstr "Primeiro numero eh maior que o segundo!"
35    call void [mscorlib]System.Console::WriteLine (string)
36    ldstr "Primeiro numero: "
37    call void [mscorlib]System.Console::WriteLine (string)
38    ldloc.0
39    call void [mscorlib]System.Console::WriteLine (int32)
40    ldstr "Segundo numero: "
41    call void [mscorlib]System.Console::WriteLine (string)
```


3 CÓDIGOS

```
42     ldloc.1
43     call void [mscorlib]System.Console::WriteLine (int32)
44     FIM:
45     ret
46 }
```

3.16 Micro 03

Listing 31: Lê um número e verifica se ele está entre 100 e 200(Portugol)

```
1 algoritmo "micro03"
2
3 var
4     numero: inteiro
5 inicio
6     escreva("Digite um numero: ")
7     leia(numero)
8     se numero >= 100 entao
9         se numero <= 200 entao
10            escreval("O numero esta no intervalo entre 100 e 200")
11        senao
12            escreval("O numero nao esta no intervalo entre 100 e
13                200")
14        fim_se
15    senao
16        escreval("O numero nao esta no intervalo entre 100 e 200"
17            )
18    fim_se
19 fim_algoritmo
```

Listing 32: Lê um número e verifica se ele está entre 100 e 200(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro03 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32)
9     ldstr "Escreva um numero: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13    stloc.0
```

3 CÓDIGOS

```
14  ldloc.0
15  ldc.i4 100
16  bge MAIORIGUALACEM
17  br FORAINTERVALO
18  MAIORIGUALACEM:
19  ldloc.0
20  ldc.i4 200
21  ble DENTROINTERVALO
22  br FORAINTERVALO
23  DENTROINTERVALO:
24  ldstr "O numero esta no intervalo entre 100 e 200"
25  call void [mscorlib]System.Console::WriteLine (string)
26  br FIM
27  FORAINTERVALO:
28  ldstr "O numero nao esta no intervalo entre 100 e 200"
29  call void [mscorlib]System.Console::WriteLine (string)
30  br FIM
31  FIM:
32  ret
33 }
```

3.17 Micro 04

Listing 33: Lê números e informa quais estão entre 10 e 150(Portugol)

```
1 algoritmo "micro04"
2
3 var
4   x, num, intervalo: inteiro
5 inicio
6   para x de 1 ate 5 faca
7     escreva("Digite um numero: ")
8     leia(num)
9     se num >= 10 entao
10      se num <= 150 entao
11        intervalo <- intervalo + 1
12      fim_se
13    fim_se
14  fim_para
15  escreval("Ao total, foram digitados ",intervalo," numeros
16  no
17  intervalo entre 10 e 150")
18 fim_algoritmo
```

Listing 34: Lê números e informa quais estão entre 10 e 150(CIL)

3 CÓDIGOS

```
1 .assembly extern mscorlib{}
2 .assembly micro04 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 5
8     .locals init (int32,int32,int32)
9     ldc.i4 0
10    stloc.2 //intervalo
11    ldc.i4 1
12    stloc.0 //variavel pro loop que comeca com x=1
13    COMECALOOOP:
14        ldloc.0 //x
15        ldc.i4.5
16        bgt ESCREVEQUANTIDADE
17        ldstr "Digite um numero: "
18        call void [mscorlib]System.Console::WriteLine (string)
19        call string [mscorlib]System.Console::ReadLine ()
20        call int32 [mscorlib]System.Int32::Parse(string)
21        stloc.1 //num
22        ldloc.0 //x
23        ldc.i4.1
24        add
25        stloc.0
26        ldloc.1 //num
27        ldc.i4 10
28        bge MAIORQUE10
29        br COMECALOOOP
30
31    MAIORQUE10:
32        ldloc.1 //num
33        ldc.i4 150
34        ble MENORQUE150
35        br COMECALOOOP
36
37    MENORQUE150:
38        ldloc.2 //intervalo
39        ldc.i4.1
40        add
41        stloc.2
42        br COMECALOOOP
43
44    ESCREVEQUANTIDADE:
45        ldstr "Quantidade de numeros digitados no intervalo
```

3 CÓDIGOS

```
        entre 10 e 150: "  
46      call void [mscorlib]System.Console::WriteLine (string)  
47      ldloc.2  
48      call void [mscorlib]System.Console::WriteLine (int32)  
49      br FIM  
50  FIM:  
51      ret  
52 }
```

3.18 Micro 05

Listing 35: Lê strings e caracteres(Portugol)

```
1 algoritmo "micro05"  
2  
3 var  
4     nome, sexo: caractere  
5     x, h, m: inteiro  
6 inicio  
7     para x de 1 ate 5 faca  
8         escreva("Digite o nome: ")  
9         leia(nome)  
10        escreva("H - Homem ou M - Mulher: ")  
11        leia(sexo)  
12        escolha sexo  
13            caso 'H'  
14                h <- h + 1  
15            caso 'M'  
16                m <- m + 1  
17        outrocaso  
18            escreval("Sexo so pode ser H ou M!")  
19        fim_escolha  
20    fim_para  
21    escreval("Foram inseridos",h," Homens")  
22    escreval("Foram inseridos",m," Mulheres")  
23 fim_algoritmo
```

Listing 36: Lê strings e caracteres(CIL)

```
1 .assembly extern mscorlib{  
2 .assembly micro05{  
3  
4 .method static void Main() cil managed  
5 {  
6     .entrypoint
```

3 CÓDIGOS

```
7  .maxstack 10
8  .locals init (int32 x,int32 h,int32 m,char sexo,string nome
   )
9  ldc.i4.1
10 stloc.0 //x
11 ldc.i4.0
12 stloc.1 //h
13 ldc.i4.0
14 stloc.2 //m
15
16 LOOP:
17     ldloc.0
18     ldc.i4.5
19     bgt IMPRIME
20     ldstr "Digite o nome: "
21     call void [mscorlib]System.Console::WriteLine (string)
22     call string [mscorlib]System.Console::ReadLine ()
23     stloc nome //nome
24     ldstr "H - Homem ou M - Mulher: "
25     call void [mscorlib]System.Console::WriteLine (string)
26     call string [mscorlib]System.Console::ReadLine ()
27     call char [mscorlib]System.Char::Parse(string)
28     stloc sexo
29     ldloc sexo
30     ldc.i4 72
31     bne.un VERIFMULHER
32     ldloc.1 //h
33     ldc.i4.1
34     add
35     stloc.1 //h
36     br INCRX
37
38
39 INCRX:
40     ldloc.0 //x
41     ldc.i4.1
42     add
43     stloc.0
44     br LOOP
45
46 VERIFMULHER:
47     ldloc sexo
48     ldc.i4 77
49     bne.un OUTROCASO
50     ldloc.2 //m
```

3 CÓDIGOS

```
51     ldc.i4.1
52     add
53     stloc.2 //m
54     br INCRX
55
56 OUTROCASO:
57     ldstr "Sexo so pode ser H ou M!"
58     call void [mscorlib]System.Console::WriteLine (string)
59     br LOOP
60
61 IMPRIME:
62     ldstr "Foram inseridos "
63     call void [mscorlib]System.Console::Write(string)
64     ldloc.1 //h
65     call void [mscorlib]System.Console::Write(int32)
66     ldstr " homens"
67     call void [mscorlib]System.Console::WriteLine (string)
68     ldstr "Foram inseridas "
69     call void [mscorlib]System.Console::Write(string)
70     ldloc.2 //m
71     call void [mscorlib]System.Console::Write(int32)
72     ldstr " mulheres"
73     call void [mscorlib]System.Console::WriteLine(string)
74
75 FIM:
76     ret
77 }
```

3.19 Micro 06

Listing 37: Escreve um número lido por extenso(Portugol)

```
1 algoritmo "micro06"
2
3 var
4     numero: inteiro
5 inicio
6     escreva("Digite um numero de 1 a 5: ")
7     leia(numero)
8     escolha numero
9     caso 1
10        escreval("Um")
11    caso 2
12        escreval("Dois")
13    caso 3
```

3 CÓDIGOS

```
14     escreval("Tres")
15 caso 4
16     escreval("Quatro")
17 caso 5
18     escreval("Cinco")
19 outrocaso
20     escreval("Numero Invalido!!!")
21 fim_escolha
22 fim_algoritmo
```

Listing 38: Escreve um número lido por extenso(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro06 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32 numero)
9     ldstr "Digite um numero de 1 a 5: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13    stloc numero
14    ldloc numero
15    ldc.i4.1
16    bne.un DOIS //Comparo se 1 e igual ao numero digitado, se
                //nao for ramifica pq "bne"
17    ldstr "Um" //Se for entao printa 1
18    call void [mscorlib]System.Console::WriteLine (string)
19    br FIM
20
21    DOIS:
22        ldloc numero
23        ldc.i4.2
24        bne.un TRES //Comparo se 2 e igual ao numero digitado, se
                    //nao for ramifica pq "bne"
25        ldstr "Dois" //Se for entao printa 2 e assim por diante...
26        call void [mscorlib]System.Console::WriteLine (string)
27        br FIM
28
29    TRES:
30        ldloc numero
31        ldc.i4.3
```

3 CÓDIGOS

```
32     bne.un QUATRO
33     ldstr "Tres"
34     call void [mscorlib]System.Console::WriteLine (string)
35     br FIM
36
37 QUATRO:
38     ldloc numero
39     ldc.i4 4
40     bne.un CINCO
41     ldstr "Quatro"
42     call void [mscorlib]System.Console::WriteLine (string)
43     br FIM
44
45 CINCO:
46     ldloc numero
47     ldc.i4 5
48     bne.un OUTROCASO
49     ldstr "Cinco"
50     call void [mscorlib]System.Console::WriteLine (string)
51     br FIM
52
53 OUTROCASO:
54     ldstr "Numero invalido!!!"
55     call void [mscorlib]System.Console::WriteLine (string)
56     br FIM
57
58 FIM:
59     ret
60
61 }
```

A instrução **bne.un** (branch if unequal or unordered) ramifica se dois operandos são diferentes ou desordenados.

3.20 Micro 07

Listing 39: Decide se os números são positivos zeros ou negativos(Portugol)

```
1 algoritmo "micro07"
2
3 var
4     programa, numero: inteiro
5     opc: caractere
6 inicio
7     programa <- 1
8     enquanto programa = 1 faca
```


3 CÓDIGOS

```
9     escreva("Digite um numero: ")
10    leia(numero)
11    se numero > 0 entao
12        escreval("Positivo")
13    senao
14        se numero = 0 entao
15            escreval("0 numero e igual a 0")
16        fim_se
17        se numero < 0 entao
18            escreval("Negativo")
19        fim_se
20    fim_se
21    escreva("Deseja finalizar? (S/N) ")
22    leia(opc)
23    se opc = "S" entao
24        programa <- 0
25    fim_se
26    fim_enquanto
27 fim_algoritmo
```

Listing 40: Decide se os números são positivos zeros ou negativos(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro10 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 10
8     .locals init (int32 numero, int32 fat)
9     ldstr "Digite um numero: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13    stloc numero
14    ldloc numero
15    call int32 fatorial(int32)
16    stloc fat //guardo em fat o retorno da funcao
17    ldloc fat
18    call void [mscorlib]System.Console::WriteLine (int32)
19
20    ret
21 }
22
23 .method public static int32 fatorial(int32 n) cil managed
```

3 CÓDIGOS

```
24 {
25     .maxstack 10
26     ldarg n //Carrega o argumento da funcao na pilha
27     ldc.i4.0
28     ble RET1 //branch if less than or equal to
29     ldarg n
30     ldc.i4.1
31     sub
32     call int32 fatorial(int32)
33     ldarg n
34     mul
35     br FIM
36
37     RET1:
38     ldc.i4.1
39     br FIM
40
41     FIM:
42     ret
43 }
```

3.21 Micro 08

Listing 41: Decide se um número é maior ou menor que 10(Portugol)

```
1 algoritmo "micro08"
2
3 var
4     numero: inteiro
5 inicio
6     numero <- 1
7     enquanto numero <> 0 faca
8         escreva("Digite um numero: ")
9         leia(numero)
10        se (numero > 10) entao
11            escreval("O numero ",numero," e maior que 10")
12        senao
13            escreval("O numero ",numero," e menor que 10")
14        fim_se
15    fim_enquanto
16 fim_algoritmo
```

Listing 42: Decide se um número é maior ou menor que 10(CIL)

```
1 .assembly extern mscorlib{}
```

3 CÓDIGOS

```
2 .assembly micro08 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 2
8     .locals init (int32 numero)
9     ldc.i4.1
10    stloc numero
11    LOOP:
12        ldloc numero
13        ldc.i4.0
14        beq FIM
15        ldstr "Digite um numero: "
16        call void [mscorlib]System.Console::WriteLine (string)
17        call string [mscorlib]System.Console::ReadLine ()
18        call int32 [mscorlib]System.Int32::Parse(string)
19        stloc numero
20        ldloc numero
21        ldc.i4 10
22        bgt MAIORQUE10 //verifica se e maior que 10
23        ldstr "O numero eh menor do que 10."
24        call void [mscorlib]System.Console::WriteLine (string)
25        br LOOP
26
27    MAIORQUE10:
28        ldstr "O numero eh maior do que 10."
29        call void [mscorlib]System.Console::WriteLine (string)
30        br LOOP
31    FIM:
32        ret
33 }
```

3.22 Micro 09

Listing 43: Cálculo de preços(Portugol)

```
1 algoritmo "micro09"
2 var
3     preco, venda, novo_preco: real
4 inicio
5     escreva("Digite o preco: ")
6     leia(preco)
7     escreva("Digite a venda: ")
8     leia(venda)
```

3 CÓDIGOS

```
9  se (venda < 500) OU (preco < 30)
10     entao novo_preco <- preco + 10/100 * preco
11     senao se (vendas >= 500 E venda < 1200) OU
12     (preco >= 30 E preco < 80)
13         entao novo_preco <- preco + 15/100 * preco
14         senao se venda >= 1200 OU preco >= 80
15             entao novo_preco <- preco - 20/100 * preco
16         fim_se
17     fim_se
18 fim_se
19 escreval("O novo preco e ", novo_preco)
20 fim_algoritmo
```

Listing 44: Cálculo de preços(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro09 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (float32 preco, float32 venda, float32
        novo_preco)
9     ldstr "Digite a preco: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call float32 [mscorlib]System.Single::Parse(string)
13    stloc preco
14    ldstr "Digite a venda: "
15    call void [mscorlib]System.Console::WriteLine (string)
16    call string [mscorlib]System.Console::ReadLine ()
17    call float32 [mscorlib]System.Single::Parse(string)
18    stloc venda
19    ldloc venda
20    ldc.r4 500.0
21    blt IF11
22    ldloc preco
23    ldc.r4 30.0
24    blt IF11
25    ldloc venda
26    ldc.r4 500.0
27    bge IF1200
28
29    IF11:
```

3 CÓDIGOS

```
30 ldc.r4 1.1
31 ldloc preco
32 mul
33 stloc novo_preco
34 ldloc novo_preco
35 call void [mscorlib]System.Console::WriteLine (float32)
36 br FIM
37
38 IF1200:
39 ldloc venda
40 ldc.r4 1200.0
41 blt IF115
42 br IF80
43
44 IF80:
45 ldloc preco
46 ldc.r4 80.0
47 blt IF115
48 br IF08
49
50 IF115:
51 ldc.r4 1.15
52 ldloc preco
53 mul
54 stloc novo_preco
55 ldloc novo_preco
56 call void [mscorlib]System.Console::WriteLine (float32)
57 br FIM
58
59 IF08:
60 ldc.r4 0.8
61 ldloc preco
62 mul
63 stloc novo_preco
64 ldloc novo_preco
65 call void [mscorlib]System.Console::WriteLine (float32)
66 br FIM
67
68 FIM:
69 ret
70
71 }
```

3.23 Micro 10

3 CÓDIGOS

Listing 45: Calcula o fatorial de um número(Portugol)

```
1 algoritmo "micro10"
2
3 var
4     numero: inteiro
5     fat: inteiro
6 inicio
7     escreva("Digite um numero: ")
8     leia(numero)
9     fat <- fatorial(numero)
10    escreva("O fatorial de ")
11    escreva(numero)
12    escreva(" e ")
13    escreval(fat)
14 fim_algoritmo
15
16 funcao fatorial(n: inteiro) : inteiro
17     se n <= 0 entao
18         retorne 1
19     senao
20         retorne n * fatorial(n-1)
21 fim_funcao fatorial
```

Listing 46: Calcula o fatorial de um número(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro10 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 10
8     .locals init (int32 numero, int32 fat)
9     ldstr "Digite um numero: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13    stloc numero
14    ldloc numero
15    call int32 fatorial(int32)
16    stloc fat //guardo em fat o retorno da funcao
17    ldloc fat
18    call void [mscorlib]System.Console::WriteLine (int32)
19
20    ret
```

3 CÓDIGOS

```
21 }
22
23 .method public static int32 fatorial(int32 n) cil managed
24 {
25     .maxstack 10
26     ldarg n //Carrega o argumento da funcao na pilha
27     ldc.i4.0
28     ble RET1 //branch if less than or equal to
29     ldarg n
30     ldc.i4.1
31     sub
32     call int32 fatorial(int32)
33     ldarg n
34     mul
35     br FIM
36
37 RET1:
38     ldc.i4.1
39     br FIM
40
41 FIM:
42     ret
43 }
```

3.24 Micro 11

Listing 47: Decide se um número é positivo zero ou negativo com auxílio de uma função(Portugol)

```
1 algoritmo "micro11"
2
3 var
4     numero: inteiro
5     x: inteiro
6 inicio
7     escreva("Digite um numero: ")
8     leia(numero)
9     x <- verifica(numero)
10    se x = 1
11        entao escreval("Numero positivo")
12    senao se x = 0
13        entao escreval("Zero")
14    senao escreval("Numero negativo")
15    fim_se
16 fim_se
```

3 CÓDIGOS

```
17 fim_algoritmo
18
19 funcao verifica(n: inteiro) : inteiro
20
21 var
22     res: inteiro
23     se n > 0 entao
24         res <- 1
25     senao se n < 0
26         entao res <- -1
27     senao res <- 0
28     fim_se
29     fim_se
30     retorne res
31
32 fim_funcao verifica
```

Listing 48: Decide se um número é positivo zero ou negativo com auxílio de uma função(CIL)

```
1 .assembly extern mscorlib{}
2 .assembly micro11 {}
3
4 .method static void Main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32 numero,int32 x)
9     ldstr "Digite um numero: "
10    call void [mscorlib]System.Console::WriteLine (string)
11    call string [mscorlib]System.Console::ReadLine ()
12    call int32 [mscorlib]System.Int32::Parse(string)
13
14    stloc numero
15    ldloc numero
16    call int32 Verifica(int32)
17    stloc x
18    ldloc x
19    ldc.i4 1
20    beq POSITIVO
21    ldloc x
22    ldc.i4 0
23    beq ZERO
24    br NEGATIVO
25
```


3 CÓDIGOS

```
26 POSITIVO:
27     ldstr "Numero positivo"
28     call void [mscorlib]System.Console::WriteLine (string)
29     br FIM
30
31 ZERO:
32     ldstr "Zero"
33     call void [mscorlib]System.Console::WriteLine (string)
34     br FIM
35
36 NEGATIVO:
37     ldstr "Numero negativo"
38     call void [mscorlib]System.Console::WriteLine (string)
39     br FIM
40
41 FIM:
42     ret
43
44 }
45
46 .method public static int32 Verifica(int32 n) cil managed
47 {
48     .maxstack 4
49     .locals init (int32 res)
50     ldc.i4 0
51     stloc res
52     ldarg n
53     ldc.i4.0
54     bgt INCREMENTA
55     ldarg n
56     ldc.i4.0
57     blt DECREMENTA
58     ldc.i4.0
59     stloc res
60     br FIM
61
62 INCREMENTA:
63     ldloc res
64     ldc.i4 1
65     add
66     stloc res
67     br FIM
68
69 DECREMENTA:
70     ldloc res
```

3 CÓDIGOS

```
71  ldc.i4 1
72  sub
73  stloc res
74  br FIM
75
76  FIM:
77  ldloc res
78  ret
79
80 }
```

4 Analisador Léxico

4.1 Autômato Finito Determinístico

O analisador léxico será representado por um autômato finito determinístico, que será implementado na linguagem Ocaml. Inicialmente esse analisador reconhece a linguagem especificada pelo professor Alexandro em sala de aula, que contém as seguintes palavras reservadas: *if*, *then*, *else* e *print*.

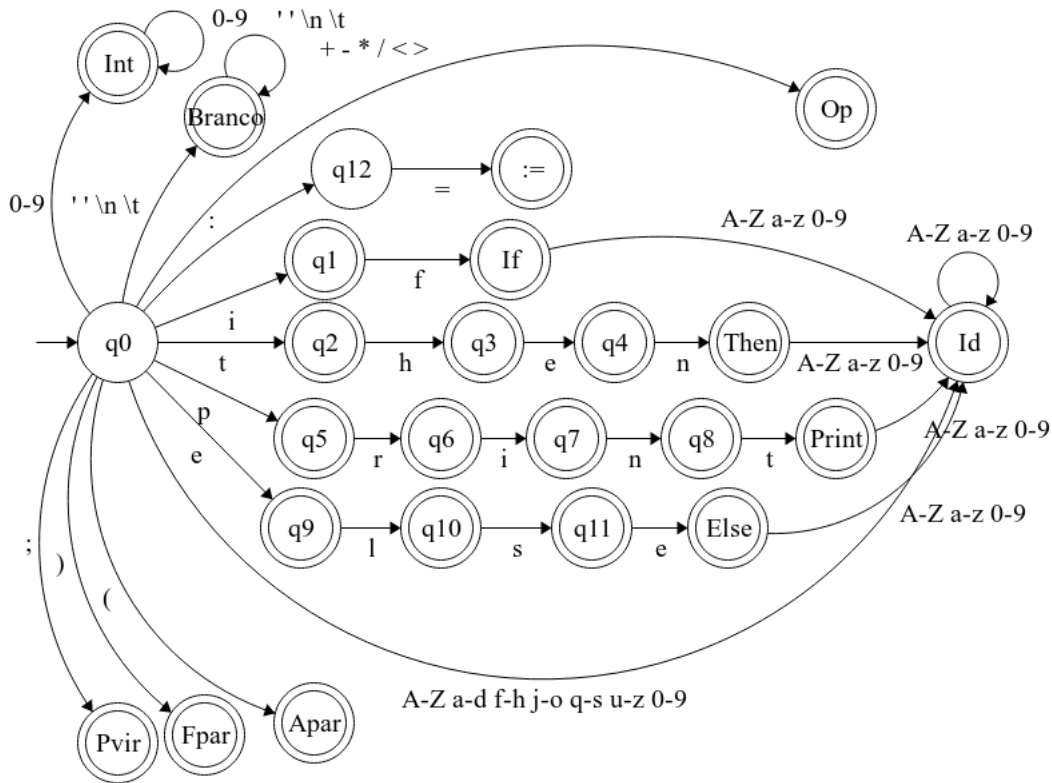


Figura 9: Autômato Finito Determinístico que reconhece a linguagem definida pelo professor em sala de aula.

Legenda:

- Int: Estado final que reconhece números inteiros.
- Branco: Estado final que reconhece caracteres em branco.
- Op: Estado final que reconhece operadores.
- := : Estado final que reconhece atribuição.
- If: Estado final que reconhece o comando if.
- Then: Estado final que reconhece o comando then.
- Else: Estado final que reconhece o comando else.
- Id: Estado final que reconhece identificadores.
- Apar: Estado final que reconhece abertura de parenteses.
- Fpar: Estado final que reconhece fechamento de parenteses.
- Pvir: Estado final que reconhece ponto e vírgula.

Além do que foi representado no autômato, existem transições dos estados ***q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11*** para o estado **Id** que é o estado de identificador, possibilitando identificadores que possuam letras das palavras reservadas. Tal escolha foi tomada devido a dificuldade para ler o autômato caso todas essas transições fossem adicionadas.

4.2 Implementação em Ocaml

Listing 49: dfalexer.ml

```
1 type estado = int
2 type entrada = string
3 type simbolo = char
4 type posicao = int
5
6 type dfa = {
7   transicao : estado -> simbolo -> estado;
8   estado: estado;
9   posicao: posicao
10 }
11
12 type token =
13   | If
14   | Then
15   | Else
16   | Print
17   | Id of string
18   | Int of string
19   | APar
20   | FPar
21   | Op of string
22   | Atribuicao
23   | PVir
24   | Branco
25   | EOF
26
27 type estado_lexico = {
28   pos_inicial: posicao; (* posição inicial na string *)
29   pos_final: posicao; (* posicao na string ao encontrar um
30     estado final recente *)
31   ultimo_final: estado; (* último estado final encontrado *)
32   dfa : dfa;
33   rotulo : estado -> entrada -> token
34 }
35 let estado_morto:estado = -1
36
37 let estado_inicial:estado = 0
38
39 let eh_letra (c:simbolo) = ('a' <= c && c <= 'z') || ('A' <=
40   c && c <= 'Z')
```

4 ANALISADOR LÉXICO

```
40
41 let eh_digito (c:simbolo) = '0' <= c && c <= '9'
42
43 let eh_branco (c:simbolo) = c = ' ' || c = '\t' || c = '\n'
44
45 let eh_operador (c:simbolo) = c = '+' || c = '-' || c = '*'
    || c = '>' || c = '<' || c = '/'
46
47 let eh_estado_final e el =
48   let rotulo = el.rotulo in
49   try
50     let _ = rotulo e "" in true
51   with _ -> false
52
53 let obtem_token_e_estado (str:entrada) el =
54   let inicio = el.pos_inicial
55   and fim = el.pos_final
56   and estado_final = el.ultimo_final
57   and rotulo = el.rotulo in
58   let tamanho = fim - inicio + 1 in
59   let lexema = String.sub str inicio tamanho in
60   let token = rotulo estado_final lexema in
61   let proximo_el = { el with pos_inicial = fim + 1;
62                        pos_final = -1;
63                        ultimo_final = -1;
64                        dfa = { el.dfa with estado =
65                                estado_inicial;
66                                posicao =
67                                    fim + 1
68                                }}
69
70   in
71   (token, proximo_el)
72
73 let rec analisador (str:entrada) tam el =
74   let posicao_atual = el.dfa.posicao
75   and estado_atual = el.dfa.estado in
76   if posicao_atual >= tam
77   then
78     if el.ultimo_final >= 0
79     then let token, proximo_el = obtem_token_e_estado str el
80         in
81         [token; EOF]
82     else [EOF]
83   else
```

4 ANALISADOR LÉXICO

```
80     let simbolo = str.[posicao_atual]
81     and transicao = el.dfa.transicao in
82     let proximo_estado = transicao estado_atual simbolo in
83     if proximo_estado = estado_morto
84     then let token, proximo_el = obtem_token_e_estado str el
85           in
86           token :: analisador str tam proximo_el
87     else
88       let proximo_el =
89         if eh_estado_final proximo_estado el
90         then { el with pos_final = posicao_atual;
91                ultimo_final = proximo_estado;
92                dfa = { el.dfa with estado =
93                      proximo_estado;
94                      posicao =
95                        posicao_atual +
96                          1 }}
97         else { el with dfa = { el.dfa with estado =
98                               proximo_estado;
99                               posicao =
100                                posicao_atual +
101                                  1 }}
102       in
103       analisador str tam proximo_el
104
105 let lexico (str:entrada) =
106   let trans (e:estado) (c:simbolo) =
107     match (e,c) with
108     | (0, 'i') -> 1
109     | (0, 't') -> 2
110     | (0, 'p') -> 5
111     | (0, 'e') -> 9
112     | (0, ':') -> 12
113     | (0, '(') -> 22
114     | (0, ')') -> 23
115     | (0, ';') -> 24
116     | (0, _) when eh_letra c -> 21
117     | (0, _) when eh_digito c -> 13
118     | (0, _) when eh_branco c -> 14
119     | (0, _) when eh_operador c -> 15
120     | (0, _) -> failwith ("Erro lexico: caracter desconhecido
121                            " ^ Char.escaped c)
122     | (1, 'f') -> 17
123     | (1, _) when eh_letra c || eh_digito c -> 21
124     | (2, 'h') -> 3
```

4 ANALISADOR LÉXICO

```
117 | (2, _) when eh_letra c || eh_digito c -> 21
118 | (3, 'e') -> 4
119 | (3, _) when eh_letra c || eh_digito c -> 21
120 | (4, 'n') -> 18
121 | (4, _) when eh_letra c || eh_digito c -> 21
122 | (5, 'r') -> 6
123 | (5, _) when eh_letra c || eh_digito c -> 21
124 | (6, 'i') -> 7
125 | (6, _) when eh_letra c || eh_digito c -> 21
126 | (7, 'n') -> 8
127 | (7, _) when eh_letra c || eh_digito c -> 21
128 | (8, 't') -> 19
129 | (8, _) when eh_letra c || eh_digito c -> 21
130 | (9, 'l') -> 10
131 | (9, _) when eh_letra c || eh_digito c -> 21
132 | (10, 's') -> 11
133 | (10, _) when eh_letra c || eh_digito c -> 21
134 | (11, 'e') -> 20
135 | (11, _) when eh_letra c || eh_digito c -> 21
136 | (12, '=') -> 16
137 | (13, _) when eh_digito c -> 13
138 | (14, _) when eh_branco c -> 14
139 | (17, _) when eh_letra c || eh_digito c -> 21
140 | (18, _) when eh_letra c || eh_digito c -> 21
141 | (19, _) when eh_letra c || eh_digito c -> 21
142 | (20, _) when eh_letra c || eh_digito c -> 21
143 | (21, _) when eh_letra c || eh_digito c -> 21
144 | _ -> estado_morto
145 and rotulo e str =
146   match e with
147   | 17 -> If
148   | 1
149   | 2
150   | 3
151   | 4
152   | 5
153   | 6
154   | 7
155   | 8
156   | 9
157   | 10
158   | 11
159   | 21 -> Id str
160   | 13 -> Int str
161   | 14 -> Branco
```


4 ANALISADOR LÉXICO

```
162 | 15 -> Op str
163 | 16 -> Atribuicao
164 | 18 -> Then
165 | 19 -> Print
166 | 20 -> Else
167 | 22 -> APar
168 | 23 -> FPar
169 | 24 -> PVir
170 | _ -> failwith ("Erro lexico: sequencia desconhecida " ^
    str)
171 in let dfa = { transicao = trans;
172                estado = estado_inicial;
173                posicao = 0 }
174 in let estado_lexico = {
175     pos_inicial = 0;
176     pos_final = -1;
177     ultimo_final = -1;
178     rotulo = rotulo;
179     dfa = dfa
180 } in
181 analisador str (String.length str) estado_lexico
```

Os estados do autômato que possuíam rótulos como: Int, Branco, Op, Atribuicao If, Then, Else, Id, Apar, Fpar, Pvir. Foram numerados da seguinte maneira no código em Ocaml:

- Int = 13
- Branco = 14
- Op = 15
- Atribuicao = 16
- If = 17
- Then = 18
- Print = 19
- Else = 20
- Id = 21
- Apar = 22

4 ANALISADOR LÉXICO

- Fpar = 23
- Pvir = 24

4.3 Execução

Para executar o código implementado em Ocaml, primeiramente utilizamos o comando **rlwrap ocaml** em um terminal Linux. Depois utilizamos o comando **#use "nomedoarquivo.ml"**, para informar ao interpretador que utilizaremos as funções definidas neste arquivo.

```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorLexico$  
rlwrap ocaml  
OCaml version 4.02.3  
  
# #use "dfalexer.ml";;
```

Figura 10: Preparando ambiente para execução do analisador léxico

```
# lexico "i := 1 + b;  
print (a*b);  
if1 := a-2;  
if if1 > 0;  
then print(if1);  
else print(if2);";  
- : token list =  
[Id "i"; Branco; Atribuicao; Branco; Int "1"; Branco; Op "+"; Branco;  
Id "b"; PVir; Branco; Print; Branco; APar; Id "a"; Op "*"; Id "b"; FPa  
r;  
PVir; Branco; Id "if1"; Branco; Atribuicao; Branco; Id "a"; Op "-";  
Int "2"; PVir; Branco; If; Branco; Id "if1"; Branco; Op ">"; Branco;  
Int "0"; PVir; Branco; Then; Branco; Print; APar; Id "if1"; FPar; PVir  
;  
Branco; Else; Branco; Print; APar; Id "if2"; FPar; PVir; EOF]  
#
```

Figura 11: Execução do analisador léxico no programa apresentado em sala de aula

4.4 Analisador Léxico para linguagem Portugol

Segue abaixo o código do arquivo **lexico.mll** em Ocaml para o analisador léxico da linguagem Portugol. Foi utilizado o auxiliador **ocamllex**.

Listing 50: lexico.mll

```
1 {
```

4 ANALISADOR LÉXICO

```
2  open Lexing
3  open Printf
4
5  let incr_num_linha lexbuf =
6      let pos = lexbuf.lex_curr_p in
7      lexbuf.lex_curr_p <- { pos with
8          pos_lnum = pos.pos_lnum + 1;
9          pos_bol = pos.pos_cnum;
10     }
11
12  let msg_erro lexbuf c =
13      let pos = lexbuf.lex_curr_p in
14      let lin = pos.pos_lnum
15      and col = pos.pos_cnum - pos.pos_bol - 1 in
16      sprintf "%d-%d: caracter desconhecido %c" lin col c
17
18  type tokens = APAR
19                | FPAR
20                | ACOL
21                | FCOL
22                | ATRIB
23                | SOMA
24                | SUB
25                | MULT
26                | DIVISAO
27                | POTENCIA
28                | MOD
29                | IGUAL
30                | MENOR
31                | MENORIGUAL
32                | MAIOR
33                | MAIORIGUAL
34                | DIFERENTE
35                | NAO
36                | OU
37                | E
38                | XOU
39                | ALEATORIO
40                | ALGORITMO
41                | ARQUIVO
42                | ASC
43                | ATE
44                | CARAC
45                | CARACPNUM
46                | CHARACTER
```

4 ANALISADOR LÉXICO

47		CASO
48		COMPR
49		COPIA
50		CRONOMETRO
51		DE
52		DEBUG
53		DECLARA
54		ECO
55		ENQUANTO
56		ENTAO
57		ESCOLHA
58		ESCREVA
59		ESCREVAL
60		FACA
61		FALSO
62		FIMALGORITMO
63		FIMENQUANTO
64		FIMESCOLHA
65		FIMFUNCAO
66		FIMPARA
67		FIMPROCEDIMENTO
68		FIMREPITA
69		FIMSE
70		FUNCAO
71		INICIO
72		INT
73		INTEIRO
74		INTERROMPA
75		LEIA
76		LIMPATELA
77		LOGICO
78		MAIUSC
79		MINUSC
80		NUMPCARAC
81		OUTROCASO
82		PARA
83		PASSO
84		PAUSA
85		POS
86		REAL
87		PROCEDIMENTO
88		REPITA
89		RETORNE
90		SE
91		SENAO

4 ANALISADOR LÉXICO

```
92         | TIMER
93         | VAR
94         | VETOR
95         | VERDADEIRO
96         | VIRGULA
97         | LITINT of int
98         | LITSTRING of string
99         | LITCHAR of string
100        | ID of string
101        | EOF
102    }
103
104    let digito = ['0' - '9']
105    let inteiro = digito+
106
107    let letra = ['a' - 'z' 'A' - 'Z']
108    let identificador = letra ( letra | digito | '_' ) *
109
110    let brancos = [' ' '\t'] +
111    let novalinha = '\r' | '\n' | "\r\n"
112
113    let comentario = "//" [ ^ '\r' '\n' ] *
114
115
116    rule token = parse
117      brancos      { token lexbuf }
118    | novalinha    { incr_num_linha lexbuf; token lexbuf }
119    | comentario   { token lexbuf }
120    | "{ "        { comentario_bloco 0 lexbuf }
121    | ','          { VIRGULA }
122    | '('          { APAR }
123    | ')'          { FPAR }
124    | '['          { ACOL }
125    | ']'          { FCOL }
126    | "<- "        { ATRIB }
127    | ':'          { DECLARA }
128    | '+'          { SOMA }
129    | '-'          { SUB }
130    | '*'          { MULT }
131    | '/'          { DIVISAO }
132    | '^'          { POTENCIA }
133    | '%'          { MOD }
134    | "MOD "       { MOD }
135    | '='          { IGUAL }
136    | '<'          { MENOR }
```

4 ANALISADOR LÉXICO

```
137 | '>'          { MAIOR }
138 | "<="         { MENORIGUAL }
139 | ">="         { MAIORIGUAL }
140 | "<>"         { DIFERENTE }
141 | "nao"        { NAO }
142 | "ou"         { OU }
143 | "e"          { E }
144 | "xou"        { XOU }
145 | inteiro as num { let numero = int_of_string num in
146 |                 LITINT numero }
147 | "aleatorio"  { ALEATORIO }
148 | "algoritmo"  { ALGORITMO }
149 | "arquivo"    { ARQUIVO }
150 | "asc"        { ASC }
151 | "ate"        { ATE }
152 | "carac"      { CARAC }
153 | "caracpnum"  { CARACPNUM }
154 | "caractere"  { CHARACTER }
155 | "caso"       { CASO }
156 | "compr"      { COMPR }
157 | "copia"      { COPIA }
158 | "cronometro" { CRONOMETRO }
159 | "de"         { DE }
160 | "debug"      { DEBUG }
161 | "eco"        { ECO }
162 | "enquanto"   { ENQUANTO }
163 | "entao"      { ENTAO }
164 | "escolha"    { ESCOLHA }
165 | "escreva"    { ESCRIVA }
166 | "escreval"   { ESCRIVAL }
167 | "faca"       { FACA }
168 | "falso"      { FALSO }
169 | "fimalgoritmo" { FIMALGORITMO }
170 | "fimenquanto" { FIMENQUANTO }
171 | "fimescolha" { FIMESCOLHA }
172 | "fimfuncao"  { FIMFUNCAO }
173 | "fimpara"    { FIMPARA }
174 | "fimprocedimento" { FIMPROCEDIMENTO }
175 | "fimrepita"  { FIMREPITA }
176 | "fimse"      { FIMSE }
177 | "funcao"     { FUNCAO }
178 | "inicio"     { INICIO }
179 | "int"        { INT }
180 | "inteiro"    { INTEIRO }
181 | "interrompa" { INTERROMPA }
```

4 ANALISADOR LÉXICO

```
182 | "leia"      { LEIA }
183 | "limpatela" { LIMPATELA }
184 | "logico"    { LOGICO }
185 | "maiusc"    { MAIUSC }
186 | "minusc"    { MINUSC }
187 | "numpcarac" { NUMPCARAC }
188 | "outrocaso" { OUTROCASO }
189 | "para"      { PARA }
190 | "passo"     { PASSO }
191 | "pausa"     { PAUSA }
192 | "pos"       { POS }
193 | "real"      { REAL }
194 | "procedimento" { PROCEDIMENTO }
195 | "repita"    { REPITA }
196 | "retorne"   { RETORNE }
197 | "se"        { SE }
198 | "senao"     { SENAO }
199 | "timer"     { TIMER }
200 | "var"       { VAR }
201 | "vetor"     { VETOR }
202 | "verdadeiro" { VERDADEIRO }
203 | identificador as id { ID id }
204 | '''         { let buffer = Buffer.create 1 in
205 |             let c = leia_string buffer lexbuf in
206 |             LITCHAR c }
207 | '''         { let buffer = Buffer.create 1 in
208 |             let str = leia_string buffer lexbuf in
209 |             LITSTRING str }
210 | _ as c { failwith (msg_erro lexbuf c) }
211 | eof    { EOF }
212 and comentario_bloco n = parse
213   "}"    { if n=0 then token lexbuf
214 |         else comentario_bloco (n-1) lexbuf }
215 | "{"    { comentario_bloco (n+1) lexbuf }
216 | _      { comentario_bloco n lexbuf }
217 | eof    { failwith "Comentário não fechado" }
218 and leia_string buffer = parse
219   '''    { Buffer.contents buffer}
220 | '''    { Buffer.contents buffer}
221 | "\\t"   { Buffer.add_char buffer '\t'; leia_string buffer
222 |         lexbuf }
222 | "\\n"   { Buffer.add_char buffer '\n'; leia_string buffer
223 |         lexbuf }
223 | '\\\''  { Buffer.add_char buffer '\''; leia_string buffer
224 |         lexbuf }
```

4 ANALISADOR LÉXICO

```
224 | '\\\\' '\\\\' { Buffer.add_char buffer '\\\\'; leia_string
      buffer lexbuf }
225 | _ as c      { Buffer.add_char buffer c; leia_string buffer
      lexbuf }
226 | eof        { failwith "A string não foi fechada"}
```

Listing 51: carregador.ml

```
1
2  #load "lexico.cmo";;
3
4  let rec tokens lexbuf =
5      let tok = Lexico.token lexbuf in
6      match tok with
7      | Lexico.EOF -> [Lexico.EOF]
8      | _ -> tok :: tokens lexbuf
9  ;;
10
11 let lexico str =
12     let lexbuf = Lexing.from_string str in
13     tokens lexbuf
14 ;;
15
16 let lex arq =
17     let ic = open_in arq in
18     let lexbuf = Lexing.from_channel ic in
19     let toks = tokens lexbuf in
20     let _ = close_in ic in
21     toks
```

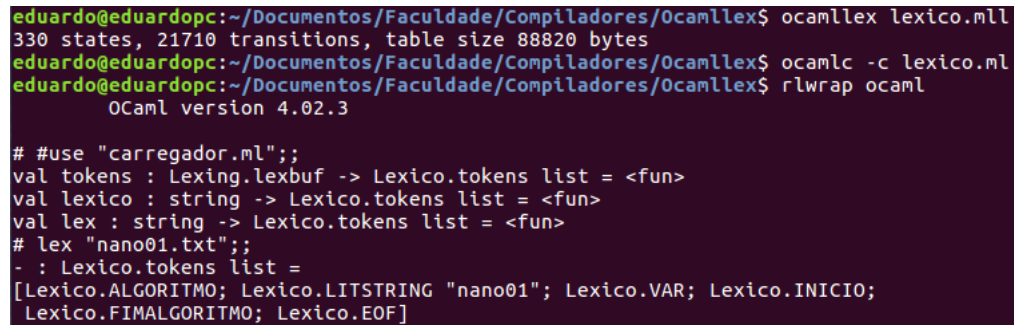
4.4.1 Execução

Para executar o código deve-se seguir a seguinte ordem:

1. `ocamllex nomedoarquivo.mll`
2. `ocamlc -c nomedoarquivo.ml`
3. `rlwrap ocaml`
4. `# load "nomedoarquivo.cmo"` (Esse passo pode ser inserido no arquivo carregador)
5. `# use "nomedoarquivocarregador.ml";;`

4 ANALISADOR LÉXICO

6. `lex "nano01.txt";;`



```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/Ocamllex$ ocamllex lexico.mll
330 states, 21710 transitions, table size 88820 bytes
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/Ocamllex$ ocamlc -c lexico.ml
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/Ocamllex$ rlwrap ocaml
OCaml version 4.02.3

# #use "carregador.ml";;
val tokens : Lexing.lexbuf -> Lexico.tokens list = <fun>
val lexico : string -> Lexico.tokens list = <fun>
val lex : string -> Lexico.tokens list = <fun>
# lex "nano01.txt";;
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano01"; Lexico.VAR; Lexico.INICIO;
 Lexico.FINALGORITMO; Lexico.EOF]
```

Figura 12: Procedimento para compilar e executar analisador léxico.

4 ANALISADOR LÉXICO

```
# lex "nano01.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano01"; Lexico.VAR; Lexico.INICIO;
Lexico.FINALGORITMO; Lexico.EOF]
# lex "nano02.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano02"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.FINALGORITMO;
Lexico.EOF]
# lex "nano03.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano03"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATTRIB;
Lexico.LITINT 1; Lexico.FINALGORITMO; Lexico.EOF]
# lex "nano04.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano04"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATTRIB;
Lexico.LITINT 1; Lexico.SOMA; Lexico.LITINT 2; Lexico.FINALGORITMO;
Lexico.EOF]
# lex "nano05.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano05"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATTRIB;
Lexico.LITINT 2; Lexico.ESCREVA; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
Lexico.FINALGORITMO; Lexico.EOF]
# lex "nano06.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano06"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATTRIB;
Lexico.LITINT 1; Lexico.SUB; Lexico.LITINT 2; Lexico.ESCREVA; Lexico.APAR;
Lexico.ID "n"; Lexico.FPAR; Lexico.FINALGORITMO; Lexico.EOF]
# lex "nano07.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano07"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATTRIB;
Lexico.LITINT 1; Lexico.SE; Lexico.ID "n"; Lexico.IGUAL; Lexico.LITINT 1;
Lexico.ENTAO; Lexico.ESCREVA; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
Lexico.FIMSE; Lexico.FINALGORITMO; Lexico.EOF]
```

Figura 13: Tokens dos arquivos nano01 até nano07.

4 ANALISADOR LÉXICO

```
# lex "nano08.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano08"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATRIB;
Lexico.LITINT 1; Lexico.SE; Lexico.ID "n"; Lexico.IGUAL; Lexico.LITINT 1;
Lexico.ENTAO; Lexico.ESCREVA; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
Lexico.SENAO; Lexico.ESCREVA; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR;
Lexico.FIMSE; Lexico.FIMALGORITMO; Lexico.EOF]

# lex "nano09.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano09"; Lexico.VAR; Lexico.ID "n";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATRIB;
Lexico.LITINT 1; Lexico.SOMA; Lexico.LITINT 1; Lexico.DIVISAO;
Lexico.LITINT 2; Lexico.SE; Lexico.ID "n"; Lexico.IGUAL; Lexico.LITINT 1;
Lexico.ENTAO; Lexico.ESCREVA; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR;
Lexico.SENAO; Lexico.ESCREVA; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR;
Lexico.FIMSE; Lexico.FIMALGORITMO; Lexico.EOF]

# lex "nano10.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano10"; Lexico.VAR; Lexico.ID "n";
Lexico.VIRGULA; Lexico.ID "m"; Lexico.DECLARA; Lexico.INTEIRO;
Lexico.INICIO; Lexico.ID "n"; Lexico.ATRIB; Lexico.LITINT 1; Lexico.ID "m";
Lexico.ATRIB; Lexico.LITINT 2; Lexico.SE; Lexico.ID "n"; Lexico.IGUAL;
Lexico.ID "m"; Lexico.ENTAO; Lexico.ESCREVA; Lexico.APAR; Lexico.ID "n";
Lexico.FPAR; Lexico.SENAO; Lexico.ESCREVA; Lexico.APAR; Lexico.LITINT 0;
Lexico.FPAR; Lexico.FIMSE; Lexico.FIMALGORITMO; Lexico.EOF]

# lex "nano11.txt";
- : Lexico.tokens list =
[Lexico.ALGORITMO; Lexico.LITSTRING "nano11"; Lexico.VAR; Lexico.ID "n";
Lexico.VIRGULA; Lexico.ID "m"; Lexico.VIRGULA; Lexico.ID "x";
Lexico.DECLARA; Lexico.INTEIRO; Lexico.INICIO; Lexico.ID "n"; Lexico.ATRIB;
Lexico.LITINT 1; Lexico.ID "m"; Lexico.ATRIB; Lexico.LITINT 2;
Lexico.ID "x"; Lexico.ATRIB; Lexico.LITINT 5; Lexico.ENQUANTO;
Lexico.ID "x"; Lexico.MAIOR; Lexico.ID "n"; Lexico.FACA; Lexico.ID "n";
Lexico.ATRIB; Lexico.ID "n"; Lexico.SOMA; Lexico.ID "m"; Lexico.ESCREVA;
Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.FIMENQUANTO;
Lexico.FIMALGORITMO; Lexico.EOF]
```

Figura 14: Tokens dos arquivos nano08 até nano11.

5 Analisador sintático preditivo

Foi solicitado a entrega de um analisador sintático preditivo para a seguinte gramática:

Exercício 1

Construa a tabela LL(1) para a gramática

$$S \rightarrow X Y Z$$

$$X \rightarrow \mathbf{a} X b$$

$$X \rightarrow$$

$$Y \rightarrow \mathbf{c} Y Z \mathbf{c} X$$

$$Y \rightarrow \mathbf{d}$$

$$Z \rightarrow \mathbf{e} Z Y \mathbf{e}$$

$$Z \rightarrow \mathbf{f}$$

Depois, use-a para verificar se *abcdfcf* pertence à linguagem gerada pela gramática.

Figura 15: Gramática

Solução

	anulável	FIRST	FOLLOW		a	b	c	d	e	f
S	Não	a c d		S	XYZ		XYZ	XYZ		
X	Sim	a	b c d e f	X	aXb	ε	ε	ε	ε	ε
Y	Não	c d	e f	Y			cYZcX	d		
Z	Não	e f	c d	Z					eZYe	f

Figura 16: Tabela LL(1) para a gramática da figura acima

Segue abaixo os códigos em Ocaml para representar o analisador sintático para a gramática acima. Vale ressaltar que é necessário fazer o analisador léxico para a linguagem, portanto o arquivo *lexico.mll* deve ser alterado.

5.1 Códigos

Listing 52: lexico.mll

```
1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6
7   let incr_num_linha lexbuf =
8     let pos = lexbuf.lex_curr_p in
9     lexbuf.lex_curr_p <- { pos with
10      pos_lnum = pos.pos_lnum + 1;
11      pos_bol = pos.pos_cnum;
12    }
13
14   let msg_erro lexbuf c =
15     let pos = lexbuf.lex_curr_p in
16     let lin = pos.pos_lnum
17     and col = pos.pos_cnum - pos.pos_bol - 1 in
18     sprintf "%d-%d: caracter desconhecido %c" lin col c
19
20
21 }
22
23 rule token = parse
24 | 'a'          {A}
25 | 'b'          {B}
26 | 'c'          {C}
27 | 'd'          {D}
28 | 'e'          {E}
29 | 'f'          {F}
30 | _ as c      { failwith (msg_erro lexbuf c) }
31 | eof         { EOF }
```

Listing 53: sintatico.mli

```
1 type tokens = A
2             | B
3             | C
4             | D
5             | E
6             | F
7             | EOF
```

5 ANALISADOR SINTÁTICO PREDITIVO

Listing 54: sintaticoArv.ml

```
1 (* Parser preditivo *)
2 #load "lexico.cmo";;
3 open Sintatico;;
4
5 type regra = S of regra * regra * regra
6             | X of tokens * regra * tokens
7             | Y of tokens * regra * regra * tokens * regra
8             | Z of tokens * regra * regra * tokens
9             | X_vazio
10            | Y_d of tokens
11            | Z_f of tokens
12
13 let tk = ref EOF (* variável global para o token atual *)
14 let lexbuf = ref (Lexing.from_string "")
15
16 (* lê o próximo token *)
17 let prox () = tk := Lexico.token !lexbuf
18
19 let to_str tk =
20   match tk with
21   | A -> "a"
22   | B -> "b"
23   | C -> "c"
24   | D -> "d"
25   | E -> "e"
26   | F -> "f"
27   | EOF -> "eof"
28
29 let erro esp =
30   let msg = Printf.sprintf "Erro: esperava %s mas encontrei %s"
31                               esp (to_str !tk)
32   in
33   failwith msg
34
35 let consome t = if (!tk == t) then prox() else erro (to_str t)
36
37 let rec ntS () =
38   match !tk with
39   | A
40   | C
41   | D ->
42     let cmd1 = ntX() in
```

5 ANALISADOR SINTÁTICO PREDITIVO

```
43         let cmd2 = ntY() in
44         let cmd3 = ntZ() in
45         S (cmd1, cmd2, cmd3)
46     | _ -> erro "a, c ou d"
47 and ntX () =
48     match !tk with
49     | B
50     | C
51     | D
52     | E
53     | F -> X_vazio
54     | A -> let _ = consome A in
55           let cmd = ntX() in
56           let _ = consome B in
57           X (A, cmd, B)
58     | _ -> erro "a"
59 and ntY () =
60     match !tk with
61     | C -> let _ = consome C in
62           let cmd = ntY() in
63           let cmd2 = ntZ() in
64           let _ = consome C in
65           let cmd3 = ntX() in
66           Y (C,cmd,cmd2, C, cmd3)
67     | D -> let _ = consome D in
68           Y_d (D)
69     | _ -> erro "c ou d"
70 and ntZ () =
71     match !tk with
72     | E -> let _ = consome E in
73           let cmd = ntZ() in
74           let cmd2 = ntY() in
75           let _ = consome E in
76           Z (E, cmd, cmd2, E)
77     | F -> let _ = consome F in
78           Z_f (F)
79     | _ -> erro "e ou f"
80
81 let parser str =
82     lexbuf := Lexing.from_string str;
83     prox (); (* inicializa o token *)
84     let arv = ntS () in
85     match !tk with
86     | EOF -> let _ = Printf.printf "Ok!\n" in arv
87     | _ -> erro "fim da entrada"
```

5 ANALISADOR SINTÁTICO PREDITIVO

```
88
89 let teste () =
90     let entrada =
91         "abcdfcf"
92     in
93     parser entrada
```

Para a execução devemos seguir os seguintes passos:

```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnaliseSintaticaTarefa4$ ocamlc -c sintatico.mli
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnaliseSintaticaTarefa4$ ocamllex lexico.mli
9 states, 257 transitions, table size 1082 bytes
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnaliseSintaticaTarefa4$ ocamlc -c lexico.ml
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnaliseSintaticaTarefa4$ rlrwrap ocaml
OCaml version 4.02.3

# #use "sintaticoArv.ml";;
type regra =
  S of regra * regra * regra
| X of Sintatico.tokens * regra * Sintatico.tokens
| Y of Sintatico.tokens * regra * regra * Sintatico.tokens * regra
| Z of Sintatico.tokens * regra * regra * Sintatico.tokens
| X_vazio
| Y_d of Sintatico.tokens
| Z_f of Sintatico.tokens
val tk : Sintatico.tokens ref = {contents = EOF}
val lexbuf : Lexing.lexbuf ref =
  {contents =
    {Lexing.refill_buff = <fun>; lex_buffer = ""; lex_buffer_len = 0;
      lex_abs_pos = 0; lex_start_pos = 0; lex_curr_pos = 0; lex_last_pos = 0;
      lex_last_action = 0; lex_eof_reached = true; lex_mem = [[]];
      lex_start_p =
        {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 0};
      lex_curr_p =
        {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 0}}}
val prox : unit -> unit = <fun>
val to_str : Sintatico.tokens -> string = <fun>
val erro : string -> 'a = <fun>
val consome : Sintatico.tokens -> unit = <fun>
val ntS : unit -> regra = <fun>
val ntX : unit -> regra = <fun>
val ntY : unit -> regra = <fun>
val ntZ : unit -> regra = <fun>
val parser : string -> regra = <fun>
```

Figura 17: Passos para a execução do analisador sintático

Segue a execução para a palavra "abcdfcf"

```
# teste();;
Ok!
- : regra = S (X (A, X_vazio, B), Y (C, Y_d D, Z_f F, C, X_vazio), Z_f F)
#
```

Figura 18: Execução para a palavra abcdfcf

6 Analisador sintático para a linguagem Portugol

O analisador sintático para a linguagem portugol foi feito utilizando o *Menhir* que é um Parser LR(1) para a linguagem de programação Ocaml.

Para intalar o *Menhir* basta seguir os passos abaixo. As mensagens durante a instação serão diferentes, pois o Menhir já estava instalado na máquina onde os testes foram realizados.

6.1 Instalação Menhir

```
eduardo@eduardopc:~$ sudo apt-get install opam
[sudo] senha para eduardo:
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
opam is already the newest version (1.2.2-4).
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  linux-headers-4.4.0-38 linux-headers-4.4.0-38-generic
  linux-image-4.4.0-38-generic linux-image-extra-4.4.0-38-generic
  linux-signed-image-4.4.0-38-generic
Utilize 'sudo apt autoremove' para os remover.
0 pacotes atualizados, 0 pacotes novos instalados, 0 a serem removidos e 28 não atualizados.
eduardo@eduardopc:~$ opam init
OPAM has already been initialized.
In normal operation, OPAM only alters files within ~/.opam.

During this initialisation, you can allow OPAM to add information to two
other files for best results. You can also make these additions manually
if you wish.

If you agree, OPAM will modify:

- ~/.profile (or a file you specify) to set the right environment
  variables and to load the auto-completion scripts for your shell (bash)
  on startup. Specifically, it checks for and appends the following line:

  . /home/eduardo/.opam/opam-init/init.sh > /dev/null 2> /dev/null || true

- ~/.ocamlinit to ensure that non-system installations of 'ocamlfind'
  (i.e. those installed by OPAM) will work correctly when running the
  OCaml toplevel. It does this by adding $OCAML_TOPLEVEL_PATH to the list
  of include directories.

If you choose to not configure your system now, you can either configure
OPAM manually (instructions will be displayed) or launch the automatic setup
later by running:

  opam config setup -a

Do you want OPAM to modify ~/.profile and ~/.ocamlinit?
(default is 'no', use 'f' to name a file other than ~/.profile)
```

Figura 19: Instalação Menhir - Parte 1

```
eduardo@eduardopc:~$ eval `opam config env`
eduardo@eduardopc:~$ opam install menhir
[NOTE] Package menhir is already installed (current version is 20160825).
eduardo@eduardopc:~$ █
```

Figura 20: Instalação Menhir - Parte 2

6.2 Códigos do Parser e da Árvore Sintática

Listing 55: parser.mly

```
1
2 %{
3   open Ast
4 %}
5
6 %token <int> INT
7 %token <float> FLOAT
8 %token <string> ID
9 %token <string> LITSTRING
10 %token <string> LITCHAR
11 %token ALGORITMO
12 %token SOMA SUB MULT DIVISAO MOD
13 %token POTENCIA
14 %token APAR
15 %token FPAR
16 %token ACOL
17 %token FCOL
18 %token IGUAL
19 %token DIFERENTE
20 %token MAIOR
21 %token MAIORIGUAL
22 %token MENOR
23 %token MENORIGUAL
24 %token E OU NAO XOU
25 %token EOF
26 %token ATRIB
27 %token DECLARA
28 %token PTV
29 %token VAR
30 %token INTEIRO
31 %token LOGICO
32 %token REAL
33 %token ATE
34 %token CARACTER
35 %token CASO
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
36 %token DE
37 %token VIRGULA
38 %token INICIO
39 %token FUNCAO
40 %token FIMFUNCAO
41 %token FIMPARA
42 %token FIMSE
43 %token FIMENQUANTO
44 %token FIMESCOLHA
45 %token SE
46 %token ENTAO
47 %token SENA0
48 %token ENQUANTO
49 %token ESCOLHA
50 %token ESCREVA
51 %token ESCREVAL
52 %token LEIA
53 %token FACA
54 %token OUTROCASO
55 %token PARA
56 %token PASSO
57 %token RETORNE
58 %token VERDADEIRO
59 %token FALSO
60 %token FIMALGORITMO
61
62
63 %left OU XO0
64 %left E
65 %left IGUAL DIFERENTE
66 %left MAIOR MAIORIGUAL MENOR MENORIGUAL
67 %left SOMA SUB
68 %left MULT DIVISAO MOD
69 %right POTENCIA
70
71
72
73
74 %start <Ast.prog> prog
75
76 %%
77
78 prog:
79     | da=declaracao_algoritmo vdb=var_decl_block? fd=
        func_decl* stmb=stm_block EOF { Prog (da,vdb,fd,stmb)
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
    }
80    ;
81
82 declaracao_algoritmo:
83     | ALGORITMO LITSTRING { DeclAlg }
84     ;
85
86 var_decl_block:
87     | VAR v=var_decl* { VarDeclBlock (v) }
88     ;
89
90 var_decl:
91     | ids = separated_nonempty_list(VIRGULA, ID) DECLARA t=
          tp_primitivo PTV { List.map (fun id -> VarDecl (id,t)
          ) ids }
92     ;
93
94
95 tp_primitivo:
96     | INTEIRO { Inteiro }
97     | REAL { Real }
98     | CARACTER { Caractere }
99     | LOGICO { Booleano }
100    ;
101
102 stm_block:
103     | INICIO stms=stm_list* FIMALGORITMO { StmBlock(stms)}
104     ;
105 stm_list:
106     | stm=stm_attr {stm}
107     | stm=fcall {stm}
108     | stm=stm_ret {stm}
109     | stm=stm_se {stm}
110     | stm=stm_enquanto {stm}
111     | stm=stm_para {stm}
112     | stm=stm_leia {stm}
113     | stm=stm_escreva {stm}
114     | stm=stm_escreval {stm}
115     | stm=stm_escolha {stm}
116     ;
117
118 stm_ret:
119     | RETORNE expr=expr? PTV { StmRetorne(expr)}
120     ;
121
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
122 lvalue:
123     | id=ID { Var(id) }
124     | lv=lvalue ACOL e=expr FCOL {VarElement(lv,e)}
125     ;
126
127
128 stm_attr:
129     | v=lvalue ATRIB e=expr PTV { StmAttrib(v,e) }
130     ;
131
132 stm_se:
133     | SE e=expr ENTÃO stms=stm_list* senao=stm_senao? FIMSE {
134         StmSe(e,stms,senao)}
135     ;
136
137 stm_senao:
138     | SENÃO stm=stm_list* { StmSeNao(stm) }
139     ;
140
141 stm_escolha:
142     | ESCOLHA id=ID c=case+ OUTROCASO stms=stm_list*
143         FIMESCOLHA {StmEscolha(id,c,stms) }
144     ;
145
146 case:
147     | CASO LITCHAR stms=stm_list* {StmCase(stms)}
148     | CASO INT stms=stm_list* {StmCase(stms) }
149     ;
150
151 stm_enquanto:
152     | ENQUANTO expr FACA stm=stm_list* FIMENQUANTO {
153         StmEnquanto stm }
154     ;
155
156 stm_para:
157     | PARA lvalue DE expr ATE expr passo? FACA stm=stm_list*
158         FIMPARA {StmPara stm }
159     ;
160
161 stm_leia:
162     | LEIA APAR id=ID FPAR PTV {StmLeia id}
163     ;
164
165 stm_escrava:
166     | ESCREVA APAR stm=separated_nonempty_list(VIRGULA, expr)
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```

    FPAR PTV {StmEscreva stm}
163     ;
164
165 stm_escreval:
166     | ESCREVAL APAR stm=separated_nonempty_list(VIRGULA, expr
    ) FPAR PTV {StmEscreval stm }
167     ;
168
169 passo:
170     | PASSO SOMA INT { }
171     | PASSO SUB INT { }
172     ;
173
174 expr:
175     | e1=expr o=op e2=expr { ExpOp(e1,o,e2) }
176     | t=termo {ExpTerm t}
177     | NAO t=termo { ExpTermNeg t}
178     | APAR e=expr FPAR { e }
179     ;
180
181 %inline op:
182     | SOMA { Soma }
183     | SUB { Subtracao }
184     | MULT { Multiplicacao }
185     | DIVISAO { Divisao }
186     | POTENCIA { Potencia }
187     | MOD { Modulo }
188     | IGUAL { Igual }
189     | DIFERENTE { Diferente }
190     | MENOR { Menor }
191     | MENORIGUAL { MenorIgual }
192     | MAIOR { Maior }
193     | MAIORIGUAL { MaiorIgual }
194     | E { ELogico }
195     | OU { OuLogico }
196     | XOUC { XouLogico }
197     ;
198
199 termo:
200     | f=fcall { TermoFcall}
201     | l=literal {TermoLiteral l}
202     | lv=lvalue {TermoVar lv}
203     ;
204
205 literal:
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
206 | LITSTRING { LitString}
207 | i=INT { Int i}
208 | f=FLOAT { Float f}
209 | LITCHAR { LitChar}
210 | l=logico_value { Bool l}
211 ;
212
213 logico_value:
214 | VERDADEIRO { Verdadeiro }
215 | FALSO { Falso }
216 ;
217
218 fcall:
219 | id=ID APAR args=fargs? FPAR { Fcallargs(id,args) }
220 ;
221
222 fargs:
223 | exprs=separated_nonempty_list(VIRGULA, expr) { List.
224   map (fun expr -> Fargs(expr)) exprs}
225 ;
226
227 func_decl:
228 | FUNCAO ID APAR fp=fparams? FPAR fy=func_type? fv=
229   fvar_decl fb=func_bloc { FuncDecl (fp,fy,fv,fb) }
230 ;
231
232 func_type:
233 | DECLARA t=tp_primitivo { FuncTipo(t) }
234 ;
235
236 func_bloc:
237 | INICIO stm=stm_list* FIMFUNCAO {FuncBloc(stm)}
238 ;
239
240 fvar_decl:
241 | v=var_decl_block? { FVarDecl(v) }
242 ;
243
244 fparams:
245 | fparam=separated_nonempty_list(VIRGULA,fparam){FParams(
246   fparam)}
247 ;
248
249 fparam:
```

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
248 | id=ID DECLARA t=tp_primitivo {FParam(id,t)}
249 ;
```

Listing 56: ast.ml

```
1
2 type identificador = string
3
4 type prog = Prog of declaracao_algoritmo * var_decl_block
  option * func_decl_list * statements
5 and declaracao_algoritmo = DeclAlg
6 and var_decl_block = VarDeclBlock of var_decl list
7 and var_decl = vars list
8 and vars = VarDecl of identificador * tipo
9 and func_decl_list = func_decl list
10 and func_decl = FuncDecl of fparams option * functype option
  * fvardecl * funcbloc
11 and fparams = FParams of fparam list
12 and fparam = FParam of identificador * tipo
13 and functype = FuncTipo of tipo
14 and fvardecl = FVarDecl of var_decl_block option
15 and funcbloc = FuncBloc of stm_list list
16 and statements = StmBlock of stm_list list
17
18 and tipo = Inteiro
19           | Real
20           | Booleano
21           | Caractere
22
23 and stm_list = StmAttrib of lvalue * expr
24               | Fcallargs of identificador * fargs option
25               | StmRetorne of expr option
26               | StmSe of expr * stm_list list * senao option
27               | StmEscolha of identificador * case list *
28                 stm_list list
29               | StmPara of stm_list list
30               | StmLeia of identificador
31               | StmEscreva of expr list
32               | StmEscreval of expr list
33               | StmEnquanto of stm_list list
34
35 and senao = StmSeNao of stm_list list
36
37 and case = StmCase of stm_list list
```


6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

```
38 and lvalue = Var of identificador
39     | VarElement of lvalue * expr
40
41 and expr = ExpOp of expr * op * expr
42     | ExpTerm of termo
43     | ExpTermNeg of termo
44
45 and op = Soma
46     | Subtracao
47     | Multiplicacao
48     | Divisao
49     | Potencia
50     | Modulo
51     | Igual
52     | Diferente
53     | Menor
54     | MenorIgual
55     | Maior
56     | MaiorIgual
57     | ELogico
58     | OuLogico
59     | XouLogico
60
61 and termo = TermoFcall
62     | TermoLiteral of literal
63     | TermoVar of lvalue
64
65 and fargs = args list
66
67 and args = Fargs of expr
68
69
70 and literal = LitString
71     | Int of int
72     | Float of float
73     | LitChar
74     | Bool of logico_value
75
76 and logico_value = Verdadeiro
77     | Falso
```

6.3 Compilando e Executando

```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ menhir --infer parser.mly
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ ocamllex lexer.mll
214 states, 12895 transitions, table size 52864 bytes
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ ocamlc -c parser.mli
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ ocamlc -c parser.ml
File "parser.ml", line 2509, characters 17-18:
Warning 26: unused variable f.
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ ocamlc -c lexer.ml
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoMenhir/
portugol$ rlwrap ocaml
OCaml version 4.02.3

# #use "mainAst.ml";;
val parse : string -> Ast.prog = <fun>
val parse_arq : string -> Ast.prog = <fun>
# █
```

Figura 21: Compilando arquivos necessários para o Parser

6 ANALISADOR SINTÁTICO PARA A LINGUAGEM PORTUGOL

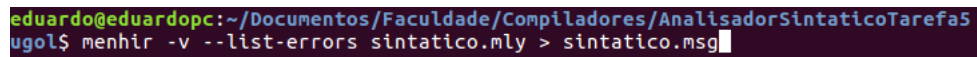
```
# parse_arq "codigosportugol/micro11.ptg";;
- : Ast.prog =
Ast.Prog (Ast.DeclAlg,
  Some
    (Ast.VarDeclBlock
      [[Ast.VarDecl ("numero", Ast.Inteiro)]; [Ast.VarDecl ("x", Ast.Inteiro)]]),
  [Ast.FuncDecl (Some (Ast.FParams [Ast.FParam ("n", Ast.Inteiro)]),
    Some (Ast.FuncTipo Ast.Inteiro),
    Ast.FVarDecl
      (Some (Ast.VarDeclBlock [[Ast.VarDecl ("res", Ast.Inteiro)]])),
    Ast.FuncBloc
      [Ast.StmSe
        (Ast.ExpOp (Ast.ExpTerm (Ast.TermoVar (Ast.Var "n")), Ast.Maior,
          Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 0))),
        [Ast.StmAttrib (Ast.Var "res",
          Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 1)))]),
        Some
          (Ast.StmSeNao
            [Ast.StmSe
              (Ast.ExpOp (Ast.ExpTerm (Ast.TermoVar (Ast.Var "n")), Ast.Menor,
                Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 0))),
              [Ast.StmAttrib (Ast.Var "res",
                Ast.ExpTerm (Ast.TermoLiteral (Ast.Int (-1)))]),
              Some
                (Ast.StmSeNao
                  [Ast.StmAttrib (Ast.Var "res",
                    Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 0)))])))]),
            Ast.StmRetorne (Some (Ast.ExpTerm (Ast.TermoVar (Ast.Var "res")))]))],
        Ast.StmBlock
          [Ast.StmEscreva [Ast.ExpTerm (Ast.TermoLiteral Ast.LitString)];
            Ast.StmLeia "numero";
            Ast.StmAttrib (Ast.Var "x", Ast.ExpTerm Ast.TermoFcall);
            Ast.StmSe
              (Ast.ExpOp (Ast.ExpTerm (Ast.TermoVar (Ast.Var "x")), Ast.Igual,
                Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 1))),
              [Ast.StmEscreval [Ast.ExpTerm (Ast.TermoLiteral Ast.LitString)]]),
              Some
                (Ast.StmSeNao
                  [Ast.StmSe
                    (Ast.ExpOp (Ast.ExpTerm (Ast.TermoVar (Ast.Var "x")), Ast.Igual,
                      Ast.ExpTerm (Ast.TermoLiteral (Ast.Int 0))),
                    [Ast.StmEscreval [Ast.ExpTerm (Ast.TermoLiteral Ast.LitString)]]),
                    Some
```

Figura 22: Árvore sintática para código micro11.ptg

7 Tratamento de Erros

7.1 Gerando arquivo de mensagens de erro

Para gerar os arquivos de mensagens de erro, basta digitar o seguinte comando:



```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoTarefa5
ugol$ menhir -v --list-errors sintatico.mly > sintatico.msg
```

Figura 23: Comando para gerar arquivo de mensagens de erro

7.2 Arquivo completo com as mensagens de erro

```
1 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR DECLARA REAL
    XOU
2 ##
3 ## Ends in an error in state: 33.
4 ##
5 ## func_decl -> FUNCAO ID APAR option(fparams) FPAR option(
    func_type) . fvar_decl func_bloc [ INICIO FUNCAO ]
6 ##
7 ## The known suffix of the stack is as follows:
8 ## FUNCAO ID APAR option(fparams) FPAR option(func_type)
9 ##
10
11 <Erro: após tipo de retorno de uma função>
12
13 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR DECLARA XOU
14 ##
15 ## Ends in an error in state: 31.
16 ##
17 ## func_type -> DECLARA . tp_primitivo [ VAR INICIO ]
18 ##
19 ## The known suffix of the stack is as follows:
20 ## DECLARA
21 ##
22
23 <Erro: após ":" da declaracao de uma função>
24
25 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR INICIO
    FIMFUNCAO XOU
26 ##
```

7 TRATAMENTO DE ERROS

```
27 ## Ends in an error in state: 188.
28 ##
29 ## list(func_decl) -> func_decl . list(func_decl) [ INICIO ]
30 ##
31 ## The known suffix of the stack is as follows:
32 ## func_decl
33 ##
34
35 <Erro: após fimfuncao>
36
37 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR INICIO RETORNE
      PTV FIMESCOLHA
38 ##
39 ## Ends in an error in state: 174.
40 ##
41 ## func_bloc -> INICIO list(stm_list) . FIMFUNCAO [ INICIO
      FUNCAO ]
42 ##
43 ## The known suffix of the stack is as follows:
44 ## INICIO list(stm_list)
45 ##
46 ## WARNING: This example involves spurious reductions.
47 ## This implies that, although the LR(1) items shown above
      provide an
48 ## accurate view of the past (what has been recognized so far
      ), they
49 ## may provide an INCOMPLETE view of the future (what was
      expected next).
50 ## In state 142, spurious reduction of production list(
      stm_list) ->
51 ## In state 153, spurious reduction of production list(
      stm_list) -> stm_list list(stm_list)
52 ##
53
54 <Erro: após retorno de uma função>
55
56 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR INICIO XOU
57 ##
58 ## Ends in an error in state: 36.
59 ##
60 ## func_bloc -> INICIO . list(stm_list) FIMFUNCAO [ INICIO
      FUNCAO ]
61 ##
62 ## The known suffix of the stack is as follows:
63 ## INICIO
```

7 TRATAMENTO DE ERROS

```
64 ##
65
66 <Erro: após inicio de uma função>
67
68 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR VAR FUNCAO
69 ##
70 ## Ends in an error in state: 35.
71 ##
72 ## func_decl -> FUNCAO ID APAR option(fparams) FPAR option(
    func_type) fvar_decl . func_bloc [ INICIO FUNCAO ]
73 ##
74 ## The known suffix of the stack is as follows:
75 ## FUNCAO ID APAR option(fparams) FPAR option(func_type)
    fvar_decl
76 ##
77 ## WARNING: This example involves spurious reductions.
78 ## This implies that, although the LR(1) items shown above
    provide an
79 ## accurate view of the past (what has been recognized so far
    ), they
80 ## may provide an INCOMPLETE view of the future (what was
    expected next).
81 ## In state 5, spurious reduction of production list(var_decl
    ) ->
82 ## In state 19, spurious reduction of production
    var_decl_block -> VAR list(var_decl)
83 ## In state 20, spurious reduction of production option(
    var_decl_block) -> var_decl_block
84 ## In state 34, spurious reduction of production fvar_decl ->
    option(var_decl_block)
85 ##
86
87 <Erro: após declaração de variáveis de uma função>
88
89 prog: ALGORITMO LITSTRING FUNCAO ID APAR FPAR XOU
90 ##
91 ## Ends in an error in state: 30.
92 ##
93 ## func_decl -> FUNCAO ID APAR option(fparams) FPAR . option(
    func_type) fvar_decl func_bloc [ INICIO FUNCAO ]
94 ##
95 ## The known suffix of the stack is as follows:
96 ## FUNCAO ID APAR option(fparams) FPAR
97 ##
98
```

7 TRATAMENTO DE ERROS

```
99 <Erro: após fechamento do paranteses de uma função >
100
101 prog: ALGORITMO LITSTRING FUNCAO ID APAR ID DECLARA CARACTER
      VIRGULA XOU
102 ##
103 ## Ends in an error in state: 180.
104 ##
105 ## separated_nonempty_list(VIRGULA,fparam) -> fparam VIRGULA
      . separated_nonempty_list(VIRGULA,fparam) [ FPAR ]
106 ##
107 ## The known suffix of the stack is as follows:
108 ## fparam VIRGULA
109 ##
110
111 <Erro: entre parâmetros de uma função>
112
113 prog: ALGORITMO LITSTRING FUNCAO ID APAR ID DECLARA CARACTER
      XOU
114 ##
115 ## Ends in an error in state: 179.
116 ##
117 ## separated_nonempty_list(VIRGULA,fparam) -> fparam . [ FPAR
      ]
118 ## separated_nonempty_list(VIRGULA,fparam) -> fparam .
      VIRGULA separated_nonempty_list(VIRGULA,fparam) [ FPAR ]
119 ##
120 ## The known suffix of the stack is as follows:
121 ## fparam
122 ##
123
124 <Erro: após tipo de parâmetro de uma função>
125
126 prog: ALGORITMO LITSTRING FUNCAO ID APAR ID DECLARA XOU
127 ##
128 ## Ends in an error in state: 26.
129 ##
130 ## fparam -> ID DECLARA . tp_primitivo [ VIRGULA FPAR ]
131 ##
132 ## The known suffix of the stack is as follows:
133 ## ID DECLARA
134 ##
135
136 <Erro: após ":" dentro dos parâmetros de uma função>
137
138 prog: ALGORITMO LITSTRING FUNCAO ID APAR ID XOU
```

7 TRATAMENTO DE ERROS

```
139 ##
140 ## Ends in an error in state: 25.
141 ##
142 ## fparam -> ID . DECLARA tp_primitivo [ VIRGULA FPAR ]
143 ##
144 ## The known suffix of the stack is as follows:
145 ## ID
146 ##
147
148 <Erro: após identificador de parâmetro de uma função>
149
150 prog: ALGORITMO LITSTRING FUNCAO ID APAR XOU
151 ##
152 ## Ends in an error in state: 24.
153 ##
154 ## func_decl -> FUNCAO ID APAR . option(fparams) FPAR option(
    func_type) fvar_decl func_bloc [ INICIO FUNCAO ]
155 ##
156 ## The known suffix of the stack is as follows:
157 ## FUNCAO ID APAR
158 ##
159
160 <Erro: após abertura de paranteses dos parâmetros de uma função>
161
162 prog: ALGORITMO LITSTRING FUNCAO ID XOU
163 ##
164 ## Ends in an error in state: 23.
165 ##
166 ## func_decl -> FUNCAO ID . APAR option(fparams) FPAR option(
    func_type) fvar_decl func_bloc [ INICIO FUNCAO ]
167 ##
168 ## The known suffix of the stack is as follows:
169 ## FUNCAO ID
170 ##
171
172 <Erro: após nome de uma função>
173
174 prog: ALGORITMO LITSTRING FUNCAO XOU
175 ##
176 ## Ends in an error in state: 22.
177 ##
178 ## func_decl -> FUNCAO . ID APAR option(fparams) FPAR option(
    func_type) fvar_decl func_bloc [ INICIO FUNCAO ]
179 ##
```


7 TRATAMENTO DE ERROS

```
180 ## The known suffix of the stack is as follows:
181 ## FUNCAO
182 ##
183
184 <Erro: após palavra-chave funcao>
185
186 prog: ALGORITMO LITSTRING INICIO ENQUANTO VERDADEIRO FACA
      RETORNE PTV FIMALGORITMO
187 ##
188 ## Ends in an error in state: 155.
189 ##
190 ## stm_enquanto -> ENQUANTO expr FACA list(stm_list) .
      FIMENQUANTO [ SENAO SE RETORNE PARA OUTROCASO LEIA ID
      FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
191 ##
192 ## The known suffix of the stack is as follows:
193 ## ENQUANTO expr FACA list(stm_list)
194 ##
195 ## WARNING: This example involves spurious reductions.
196 ## This implies that, although the LR(1) items shown above
      provide an
197 ## accurate view of the past (what has been recognized so far
      ), they
198 ## may provide an INCOMPLETE view of the future (what was
      expected next).
199 ## In state 142, spurious reduction of production list(
      stm_list) ->
200 ## In state 153, spurious reduction of production list(
      stm_list) -> stm_list list(stm_list)
201 ##
202
203 <Erro: Comando enquanto sem fimenquanto >
204
205 prog: ALGORITMO LITSTRING INICIO ENQUANTO VERDADEIRO FACA XOU
206 ##
207 ## Ends in an error in state: 138.
208 ##
209 ## stm_enquanto -> ENQUANTO expr FACA . list(stm_list)
      FIMENQUANTO [ SENAO SE RETORNE PARA OUTROCASO LEIA ID
      FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
210 ##
211 ## The known suffix of the stack is as follows:
212 ## ENQUANTO expr FACA
```

7 TRATAMENTO DE ERROS

```
213 ##
214
215 <Erro: após comando faca>
216
217 prog: ALGORITMO LITSTRING INICIO ENQUANTO VERDADEIRO VIRGULA
218 ##
219 ## Ends in an error in state: 137.
220 ##
221 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
222 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
223 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
224 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
225 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
226 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
227 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
228 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
229 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
230 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
231 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
232 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
```

7 TRATAMENTO DE ERROS

```
233 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
234 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
235 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E DIVISAO
    DIFERENTE ]
236 ## stm_enquanto -> ENQUANTO expr . FACA list(stm_list)
    FIMENQUANTO [ SENAO SE RETORNE PARA OUTROCASO LEIA ID
    FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
    FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
237 ##
238 ## The known suffix of the stack is as follows:
239 ## ENQUANTO expr
240 ##
241
242 <Erro: após enquanto>
243
244 prog: ALGORITMO LITSTRING INICIO ENQUANTO XOU
245 ##
246 ## Ends in an error in state: 136.
247 ##
248 ## stm_enquanto -> ENQUANTO . expr FACA list(stm_list)
    FIMENQUANTO [ SENAO SE RETORNE PARA OUTROCASO LEIA ID
    FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
    FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
249 ##
250 ## The known suffix of the stack is as follows:
251 ## ENQUANTO
252 ##
253
254 <Erro: após palavra-chave enquanto>
255
256 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO INT
    OUTROCASO RETORNE PTV FIMENQUANTO
257 ##
258 ## Ends in an error in state: 162.
259 ##
260 ## stm_escolha -> ESCOLHA ID nonempty_list(case) OUTROCASO
    list(stm_list) . FIMESCOLHA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
```

7 TRATAMENTO DE ERROS

```
261 ##
262 ## The known suffix of the stack is as follows:
263 ## ESCOLHA ID nonempty_list(case) OUTROCASO list(stm_list)
264 ##
265 ## WARNING: This example involves spurious reductions.
266 ## This implies that, although the LR(1) items shown above
    provide an
267 ## accurate view of the past (what has been recognized so far
    ), they
268 ## may provide an INCOMPLETE view of the future (what was
    expected next).
269 ## In state 142, spurious reduction of production list(
    stm_list) ->
270 ## In state 153, spurious reduction of production list(
    stm_list) -> stm_list list(stm_list)
271 ##
272
273 <Erro: comando escolha>
274
275 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO INT
    OUTROCASO XOU
276 ##
277 ## Ends in an error in state: 161.
278 ##
279 ## stm_escolha -> ESCOLHA ID nonempty_list(case) OUTROCASO .
    list(stm_list) FIMESCOLHA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
280 ##
281 ## The known suffix of the stack is as follows:
282 ## ESCOLHA ID nonempty_list(case) OUTROCASO
283 ##
284
285 <Erro: comando escolha>
286
287 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO INT XOU
288 ##
289 ## Ends in an error in state: 158.
290 ##
291 ## case -> CASO INT . list(stm_list) [ OUTROCASO CASO ]
292 ##
293 ## The known suffix of the stack is as follows:
294 ## CASO INT
295 ##
```

7 TRATAMENTO DE ERROS

```
296
297 <Erro: comando escolha>
298
299 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO LITCHAR
      RETORNE PTV SENAO
300 ##
301 ## Ends in an error in state: 164.
302 ##
303 ## nonempty_list(case) -> case . [ OUTROCASO ]
304 ## nonempty_list(case) -> case . nonempty_list(case) [
      OUTROCASO ]
305 ##
306 ## The known suffix of the stack is as follows:
307 ## case
308 ##
309 ## WARNING: This example involves spurious reductions.
310 ## This implies that, although the LR(1) items shown above
      provide an
311 ## accurate view of the past (what has been recognized so far
      ), they
312 ## may provide an INCOMPLETE view of the future (what was
      expected next).
313 ## In state 142, spurious reduction of production list(
      stm_list) ->
314 ## In state 153, spurious reduction of production list(
      stm_list) -> stm_list list(stm_list)
315 ## In state 157, spurious reduction of production case ->
      CASO LITCHAR list(stm_list)
316 ##
317
318 <Erro: comando escolha>
319
320 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO LITCHAR XOU
321 ##
322 ## Ends in an error in state: 135.
323 ##
324 ## case -> CASO LITCHAR . list(stm_list) [ OUTROCASO CASO ]
325 ##
326 ## The known suffix of the stack is as follows:
327 ## CASO LITCHAR
328 ##
329
330 <Erro: comando escolha>
331
332 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID CASO XOU
```

7 TRATAMENTO DE ERROS

```
333 ##
334 ## Ends in an error in state: 134.
335 ##
336 ## case -> CASO . LITCHAR list(stm_list) [ OUTROCASO CASO ]
337 ## case -> CASO . INT list(stm_list) [ OUTROCASO CASO ]
338 ##
339 ## The known suffix of the stack is as follows:
340 ## CASO
341 ##
342
343 <Erro: comando escolha>
344
345 prog: ALGORITMO LITSTRING INICIO ESCOLHA ID XOU
346 ##
347 ## Ends in an error in state: 133.
348 ##
349 ## stm_escolha -> ESCOLHA ID . nonempty_list(case) OUTROCASO
    list(stm_list) FIMESCOLHA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
350 ##
351 ## The known suffix of the stack is as follows:
352 ## ESCOLHA ID
353 ##
354
355 <Erro: comando escolha>
356
357 prog: ALGORITMO LITSTRING INICIO ESCOLHA XOU
358 ##
359 ## Ends in an error in state: 132.
360 ##
361 ## stm_escolha -> ESCOLHA . ID nonempty_list(case) OUTROCASO
    list(stm_list) FIMESCOLHA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
362 ##
363 ## The known suffix of the stack is as follows:
364 ## ESCOLHA
365 ##
366
367 <Erro: comando escolha>
368
```

7 TRATAMENTO DE ERROS

```
369 prog: ALGORITMO LITSTRING INICIO ESCRIVA APAR VERDADEIRO FPAR
      XOU
370 ##
371 ## Ends in an error in state: 130.
372 ##
373 ## stm_escrava -> ESCRIVA APAR separated_nonempty_list(
      VIRGULA,expr) FPAR . PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCRIVAL ESCRIVA ESCOLHA ENQUANTO CASO ]
374 ##
375 ## The known suffix of the stack is as follows:
376 ## ESCRIVA APAR separated_nonempty_list(VIRGULA,expr) FPAR
377 ##
378
379 <Erro: comando escreva>
380
381 prog: ALGORITMO LITSTRING INICIO ESCRIVA APAR XOU
382 ##
383 ## Ends in an error in state: 128.
384 ##
385 ## stm_escrava -> ESCRIVA APAR . separated_nonempty_list(
      VIRGULA,expr) FPAR PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCRIVAL ESCRIVA ESCOLHA ENQUANTO CASO ]
386 ##
387 ## The known suffix of the stack is as follows:
388 ## ESCRIVA APAR
389 ##
390
391 <Erro: comando escreva>
392
393 prog: ALGORITMO LITSTRING INICIO ESCRIVA XOU
394 ##
395 ## Ends in an error in state: 127.
396 ##
397 ## stm_escrava -> ESCRIVA . APAR separated_nonempty_list(
      VIRGULA,expr) FPAR PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCRIVAL ESCRIVA ESCOLHA ENQUANTO CASO ]
398 ##
399 ## The known suffix of the stack is as follows:
400 ## ESCRIVA
401 ##
402
403 <Erro: comando escreva>
```

7 TRATAMENTO DE ERROS

```
404
405 prog: ALGORITMO LITSTRING INICIO ESCREVAL APAR VERDADEIRO
      FPAR XOY
406 ##
407 ## Ends in an error in state: 125.
408 ##
409 ## stm_escraval -> ESCREVAL APAR separated_nonempty_list(
      VIRGULA,expr) FPAR . PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
410 ##
411 ## The known suffix of the stack is as follows:
412 ## ESCREVAL APAR separated_nonempty_list(VIRGULA,expr) FPAR
413 ##
414
415 <Erro: comando escraval>
416
417 prog: ALGORITMO LITSTRING INICIO ESCREVAL APAR XOY
418 ##
419 ## Ends in an error in state: 123.
420 ##
421 ## stm_escraval -> ESCREVAL APAR . separated_nonempty_list(
      VIRGULA,expr) FPAR PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
422 ##
423 ## The known suffix of the stack is as follows:
424 ## ESCREVAL APAR
425 ##
426
427 <Erro: comando escraval>
428
429 prog: ALGORITMO LITSTRING INICIO ESCREVAL XOY
430 ##
431 ## Ends in an error in state: 122.
432 ##
433 ## stm_escraval -> ESCREVAL . APAR separated_nonempty_list(
      VIRGULA,expr) FPAR PTV [ SENAO SE RETORNE PARA OUTROCASO
      LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
      FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
434 ##
435 ## The known suffix of the stack is as follows:
436 ## ESCREVAL
437 ##
438
```


7 TRATAMENTO DE ERROS

```
439 <Erro: comando escreval>
440
441 prog: ALGORITMO LITSTRING INICIO FIMALGORITMO XOU
442 ##
443 ## Ends in an error in state: 186.
444 ##
445 ## prog -> declaracao_algoritmo option(var_decl_block) list(
      func_decl) stm_block . EOF [ # ]
446 ##
447 ## The known suffix of the stack is as follows:
448 ## declaracao_algoritmo option(var_decl_block) list(func_decl
      ) stm_block
449 ##
450
451 <Erro: após fimalgoritmo>
452
453 prog: ALGORITMO LITSTRING INICIO ID ACOL VERDADEIRO VIRGULA
454 ##
455 ## Ends in an error in state: 54.
456 ##
457 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
      DIFERENTE ]
458 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
      DIFERENTE ]
459 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
      DIFERENTE ]
460 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E
      DIVISAO DIFERENTE ]
461 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E
      DIVISAO DIFERENTE ]
462 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
      DIFERENTE ]
463 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
      DIFERENTE ]
464 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E
      DIVISAO DIFERENTE ]
```

7 TRATAMENTO DE ERROS

```
465 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
    DIFERENTE ]
466 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E
    DIVISAO DIFERENTE ]
467 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
    DIFERENTE ]
468 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E
    DIVISAO DIFERENTE ]
469 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
    DIFERENTE ]
470 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
    DIFERENTE ]
471 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FCOL E DIVISAO
    DIFERENTE ]
472 ## lvalue -> lvalue ACOL expr . FCOL [ XOU VIRGULA SUB SOMA
    PTV POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE DE
    ATRIB ATE ACOL ]
473 ##
474 ## The known suffix of the stack is as follows:
475 ## lvalue ACOL expr
476 ##
477
478 <Erro: após abrir colchetes>
479
480 prog: ALGORITMO LITSTRING INICIO ID ACOL XOU
481 ##
482 ## Ends in an error in state: 50.
483 ##
484 ## lvalue -> lvalue ACOL . expr FCOL [ XOU VIRGULA SUB SOMA
    PTV POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE DE
    ATRIB ATE ACOL ]
485 ##
486 ## The known suffix of the stack is as follows:
487 ## lvalue ACOL
488 ##
489
```

7 TRATAMENTO DE ERROS

```
490 <Erro: após abrir colchetes>
491
492 prog: ALGORITMO LITSTRING INICIO ID APAR VERDADEIRO
      VERDADEIRO
493 ##
494 ## Ends in an error in state: 92.
495 ##
496 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA POTENCIA
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
497 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
498 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA POTENCIA
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
499 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA
      POTENCIA OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR
      IGUAL FPAR E DIVISAO DIFERENTE ]
500 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA
      POTENCIA OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR
      IGUAL FPAR E DIVISAO DIFERENTE ]
501 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
502 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA POTENCIA
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
503 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA
      POTENCIA OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR
      IGUAL FPAR E DIVISAO DIFERENTE ]
504 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA POTENCIA
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
505 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA
      POTENCIA OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR
      IGUAL FPAR E DIVISAO DIFERENTE ]
506 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA POTENCIA
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
507 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA
      POTENCIA OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR
      IGUAL FPAR E DIVISAO DIFERENTE ]
508 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
```

7 TRATAMENTO DE ERROS

```

    DIVISAO DIFERENTE ]
509 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
    DIVISAO DIFERENTE ]
510 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
    DIVISAO DIFERENTE ]
511 ## separated_nonempty_list(VIRGULA,expr) -> expr . [ FPAR ]
512 ## separated_nonempty_list(VIRGULA,expr) -> expr . VIRGULA
    separated_nonempty_list(VIRGULA,expr) [ FPAR ]
513 ##
514 ## The known suffix of the stack is as follows:
515 ## expr
516 ##
517
518 <Erro: expressão incorreta>
519
520 prog: ALGORITMO LITSTRING INICIO ID APAR VERDADEIRO VIRGULA
    XOU
521 ##
522 ## Ends in an error in state: 93.
523 ##
524 ## separated_nonempty_list(VIRGULA,expr) -> expr VIRGULA .
    separated_nonempty_list(VIRGULA,expr) [ FPAR ]
525 ##
526 ## The known suffix of the stack is as follows:
527 ## expr VIRGULA
528 ##
529
530 <Erro: expressão incorreta>
531
532 prog: ALGORITMO LITSTRING INICIO ID APAR XOU
533 ##
534 ## Ends in an error in state: 44.
535 ##
536 ## fcall -> ID APAR . option(fargs) FPAR [ XOU VIRGULA SUB
    SOMA SENAO SE RETORNE PTV POTENCIA PASSO PARA OUTROCASO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR LEIA IGUAL ID
    FPAR FIMSE FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO
    FIMALGORITMO FCOL FACA ESCREVAL ESCREVA ESCOLHA ENTAO
    ENQUANTO E DIVISAO DIFERENTE CASO ATE ]
537 ##
538 ## The known suffix of the stack is as follows:
539 ## ID APAR
540 ##
```

7 TRATAMENTO DE ERROS

```
541
542 <Erro: após "inicio" bloco de comandos deve estar incorreto>
543
544 prog: ALGORITMO LITSTRING INICIO ID ATRIB VERDADEIRO VIRGULA
545 ##
546 ## Ends in an error in state: 151.
547 ##
548 ## expr -> expr . SOMA expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
549 ## expr -> expr . SUB expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
550 ## expr -> expr . MULT expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
551 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
552 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
553 ## expr -> expr . MOD expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
554 ## expr -> expr . IGUAL expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
555 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
556 ## expr -> expr . MENOR expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
557 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
558 ## expr -> expr . MAIOR expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
559 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
560 ## expr -> expr . E expr [ XOU SUB SOMA PTV POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
```

7 TRATAMENTO DE ERROS

```

    DIFERENTE ]
561 ## expr -> expr . OU expr [ XOU SUB SOMA PTV POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
562 ## expr -> expr . XOU expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
563 ## stm_attr -> lvalue ATRIB expr . PTV [ SENAO SE RETORNE
    PARA OUTROCASO LEIA ID FIMSE FIMPARGA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
564 ##
565 ## The known suffix of the stack is as follows:
566 ## lvalue ATRIB expr
567 ##
568
569 <Erro: atribuição incorreta>
570
571 prog: ALGORITMO LITSTRING INICIO ID ATRIB XOU
572 ##
573 ## Ends in an error in state: 150.
574 ##
575 ## stm_attr -> lvalue ATRIB . expr PTV [ SENAO SE RETORNE
    PARA OUTROCASO LEIA ID FIMSE FIMPARGA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
576 ##
577 ## The known suffix of the stack is as follows:
578 ## lvalue ATRIB
579 ##
580
581 <Erro: atribuição incorreta>
582
583 prog: ALGORITMO LITSTRING INICIO ID VERDADEIRO
584 ##
585 ## Ends in an error in state: 43.
586 ##
587 ## fcall -> ID . APAR option(fargs) FPAR [ XOU VIRGULA SUB
    SOMA SENAO SE RETORNE PTV POTENCIA PASSO PARA OUTROCASO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR LEIA IGUAL ID
    FPAR FIMSE FIMPARGA FIMFUNCAO FIMESCOLHA FIMENQUANTO
    FIMALGORITMO FCOL FACA ESCREVAL ESCREVA ESCOLHA ENTAO
    ENQUANTO E DIVISAO DIFERENTE CASO ATE ]
588 ## lvalue -> ID . [ XOU VIRGULA SUB SOMA PTV POTENCIA PASSO
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR
```

7 TRATAMENTO DE ERROS

```
        FCOL FACA ENTAO E DIVISAO DIFERENTE ATRIB ATE ACOL ]
589 ##
590 ## The known suffix of the stack is as follows:
591 ## ID
592 ##
593
594 <Erro: após "inicio" bloco de comandos deve estar incorreto>
595
596 prog: ALGORITMO LITSTRING INICIO ID XOU
597 ##
598 ## Ends in an error in state: 149.
599 ##
600 ## lvalue -> lvalue . ACOL expr FCOL [ ATRIB ACOL ]
601 ## stm_attr -> lvalue . ATRIB expr PTV [ SENAO SE RETORNE
        PARA OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
602 ##
603 ## The known suffix of the stack is as follows:
604 ## lvalue
605 ##
606 ## WARNING: This example involves spurious reductions.
607 ## This implies that, although the LR(1) items shown above
        provide an
608 ## accurate view of the past (what has been recognized so far
        ), they
609 ## may provide an INCOMPLETE view of the future (what was
        expected next).
610 ## In state 43, spurious reduction of production lvalue -> ID
611 ##
612
613 <Erro: após identificador>
614
615 prog: ALGORITMO LITSTRING INICIO LEIA APAR ID FPAR XOU
616 ##
617 ## Ends in an error in state: 120.
618 ##
619 ## stm_leia -> LEIA APAR ID FPAR . PTV [ SENAO SE RETORNE
        PARA OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
620 ##
621 ## The known suffix of the stack is as follows:
622 ## LEIA APAR ID FPAR
623 ##
```

7 TRATAMENTO DE ERROS

```
624
625 <Erro: comando leia>
626
627 prog: ALGORITMO LITSTRING INICIO LEIA APAR ID XOU
628 ##
629 ## Ends in an error in state: 119.
630 ##
631 ## stm_leia -> LEIA APAR ID . FPAR PTV [ SENAO SE RETORNE
        PARA OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
632 ##
633 ## The known suffix of the stack is as follows:
634 ## LEIA APAR ID
635 ##
636
637 <Erro: comando leia>
638
639 prog: ALGORITMO LITSTRING INICIO LEIA APAR XOU
640 ##
641 ## Ends in an error in state: 118.
642 ##
643 ## stm_leia -> LEIA APAR . ID FPAR PTV [ SENAO SE RETORNE
        PARA OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
644 ##
645 ## The known suffix of the stack is as follows:
646 ## LEIA APAR
647 ##
648
649 <Erro: comando leia>
650
651 prog: ALGORITMO LITSTRING INICIO LEIA XOU
652 ##
653 ## Ends in an error in state: 117.
654 ##
655 ## stm_leia -> LEIA . APAR ID FPAR PTV [ SENAO SE RETORNE
        PARA OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
656 ##
657 ## The known suffix of the stack is as follows:
658 ## LEIA
659 ##
```


7 TRATAMENTO DE ERROS

```
660
661 <Erro: comando leia>
662
663 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
        VERDADEIRO FACA RETORNE PTV FIMFUNCAO
664 ##
665 ## Ends in an error in state: 166.
666 ##
667 ## stm_para -> PARA lvalue DE expr ATE expr option(passo)
        FACA list(stm_list) . FIMPARA [ SENAO SE RETORNE PARA
        OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
668 ##
669 ## The known suffix of the stack is as follows:
670 ## PARA lvalue DE expr ATE expr option(passo) FACA list(
        stm_list)
671 ##
672 ## WARNING: This example involves spurious reductions.
673 ## This implies that, although the LR(1) items shown above
        provide an
674 ## accurate view of the past (what has been recognized so far
        ), they
675 ## may provide an INCOMPLETE view of the future (what was
        expected next).
676 ## In state 142, spurious reduction of production list(
        stm_list) ->
677 ## In state 153, spurious reduction of production list(
        stm_list) -> stm_list list(stm_list)
678 ##
679
680 <Erro: comando para>
681
682 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
        VERDADEIRO FACA XOU
683 ##
684 ## Ends in an error in state: 116.
685 ##
686 ## stm_para -> PARA lvalue DE expr ATE expr option(passo)
        FACA . list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
        OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
        FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
        CASO ]
687 ##
688 ## The known suffix of the stack is as follows:
```

7 TRATAMENTO DE ERROS

```
689 ## PARA lvalue DE expr ATE expr option(passo) FACA
690 ##
691
692 <Erro: comando para>
693
694 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
      VERDADEIRO PASSO SOMA XOU
695 ##
696 ## Ends in an error in state: 112.
697 ##
698 ## passo -> PASSO SOMA . INT [ FACA ]
699 ##
700 ## The known suffix of the stack is as follows:
701 ## PASSO SOMA
702 ##
703
704 <Erro: comando para>
705
706 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
      VERDADEIRO PASSO SUB INT ESCREVAL
707 ##
708 ## Ends in an error in state: 115.
709 ##
710 ## stm_para -> PARA lvalue DE expr ATE expr option(passo) .
      FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
      OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
      FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
      CASO ]
711 ##
712 ## The known suffix of the stack is as follows:
713 ## PARA lvalue DE expr ATE expr option(passo)
714 ##
715
716 <Erro: comando para>
717
718 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
      VERDADEIRO PASSO SUB XOU
719 ##
720 ## Ends in an error in state: 110.
721 ##
722 ## passo -> PASSO SUB . INT [ FACA ]
723 ##
724 ## The known suffix of the stack is as follows:
725 ## PASSO SUB
726 ##
```

7 TRATAMENTO DE ERROS

```
727
728 <Erro: comando para>
729
730 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
      VERDADEIRO PASSO XOU
731 ##
732 ## Ends in an error in state: 109.
733 ##
734 ## passo -> PASSO . SOMA INT [ FACA ]
735 ## passo -> PASSO . SUB INT [ FACA ]
736 ##
737 ## The known suffix of the stack is as follows:
738 ## PASSO
739 ##
740
741 <Erro: comando para>
742
743 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
      VERDADEIRO VIRGULA
744 ##
745 ## Ends in an error in state: 108.
746 ##
747 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA PASSO OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
748 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA PASSO OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
749 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA PASSO OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
750 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA PASSO
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
751 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA PASSO
      OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA
      E DIVISAO DIFERENTE ]
752 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA PASSO OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
753 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA PASSO OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
      DIVISAO DIFERENTE ]
754 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA
      PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
```

7 TRATAMENTO DE ERROS

```

    FACA E DIVISAO DIFERENTE ]
755 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA PASSO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
756 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FACA E DIVISAO DIFERENTE ]
757 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA PASSO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
758 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FACA E DIVISAO DIFERENTE ]
759 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA PASSO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
760 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA PASSO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
761 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA PASSO OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FACA E
    DIVISAO DIFERENTE ]
762 ## stm_para -> PARA lvalue DE expr ATE expr . option(passo)
    FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
763 ##
764 ## The known suffix of the stack is as follows:
765 ## PARA lvalue DE expr ATE expr
766 ##
767
768 <Erro: comando para>
769
770 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO ATE
    XOU
771 ##
772 ## Ends in an error in state: 107.
773 ##
774 ## stm_para -> PARA lvalue DE expr ATE . expr option(passo)
    FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
775 ##
```

7 TRATAMENTO DE ERROS

```
776 ## The known suffix of the stack is as follows:
777 ## PARA lvalue DE expr ATE
778 ##
779
780 <Erro: comando para>
781
782 prog: ALGORITMO LITSTRING INICIO PARA ID DE VERDADEIRO
      VIRGULA
783 ##
784 ## Ends in an error in state: 106.
785 ##
786 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
787 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
788 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
789 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
790 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
791 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
792 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
793 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
794 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
795 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
796 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
      DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
797 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ATE ]
798 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ATE ]
799 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ATE ]
800 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ATE ]
801 ## stm_para -> PARA lvalue DE expr . ATE expr option(passo)
    FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
802 ##
803 ## The known suffix of the stack is as follows:
804 ## PARA lvalue DE expr
805 ##
806
807 <Erro: comando para>
808
809 prog: ALGORITMO LITSTRING INICIO PARA ID DE XOU
810 ##
811 ## Ends in an error in state: 105.
812 ##
813 ## stm_para -> PARA lvalue DE . expr ATE expr option(passo)
    FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
    OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
814 ##
815 ## The known suffix of the stack is as follows:
816 ## PARA lvalue DE
817 ##
818
819 <Erro: comando para>
820
821 prog: ALGORITMO LITSTRING INICIO PARA ID XOU
822 ##
823 ## Ends in an error in state: 104.
824 ##
825 ## lvalue -> lvalue . ACOL expr FCOL [ DE ACOL ]
```

7 TRATAMENTO DE ERROS

```
826 ## stm_para -> PARA lvalue . DE expr ATE expr option(passo)
      FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
      OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
      FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
      CASO ]
827 ##
828 ## The known suffix of the stack is as follows:
829 ## PARA lvalue
830 ##
831
832 <Erro: comando para>
833
834 prog: ALGORITMO LITSTRING INICIO PARA XOU
835 ##
836 ## Ends in an error in state: 102.
837 ##
838 ## stm_para -> PARA . lvalue DE expr ATE expr option(passo)
      FACA list(stm_list) FIMPARA [ SENAO SE RETORNE PARA
      OUTROCASO LEIA ID FIMSE FIMPARA FIMFUNCAO FIMESCOLHA
      FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
      CASO ]
839 ##
840 ## The known suffix of the stack is as follows:
841 ## PARA
842 ##
843
844 <Erro: comando para>
845
846 prog: ALGORITMO LITSTRING INICIO RETORNE PTV CASO
847 ##
848 ## Ends in an error in state: 184.
849 ##
850 ## stm_block -> INICIO list(stm_list) . FIMALGORITMO [ EOF ]
851 ##
852 ## The known suffix of the stack is as follows:
853 ## INICIO list(stm_list)
854 ##
855 ## WARNING: This example involves spurious reductions.
856 ## This implies that, although the LR(1) items shown above
      provide an
857 ## accurate view of the past (what has been recognized so far
      ), they
858 ## may provide an INCOMPLETE view of the future (what was
      expected next).
```

7 TRATAMENTO DE ERROS

```
859 ## In state 142, spurious reduction of production list(  
    stm_list) ->  
860 ## In state 153, spurious reduction of production list(  
    stm_list) -> stm_list list(stm_list)  
861 ##  
862  
863 <Erro: após "inicio" bloco de comandos deve estar incorreto>  
864  
865  
866 prog: ALGORITMO LITSTRING INICIO RETORNE PTV XOU  
867 ##  
868 ## Ends in an error in state: 142.  
869 ##  
870 ## list(stm_list) -> stm_list . list(stm_list) [ SENA  
    OUTROCASO FIMSE FIMPARGA FIMFUNCAO FIMESCOLHA FIMENQUANTO  
    FIMALGORITMO CASO ]  
871 ##  
872 ## The known suffix of the stack is as follows:  
873 ## stm_list  
874 ##  
875  
876 <Erro: após "inicio" bloco de comandos deve estar incorreto>  
877  
878  
879 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO DIFERENTE  
    VERDADEIRO VERDADEIRO  
880 ##  
881 ## Ends in an error in state: 82.  
882 ##  
883 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV  
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL  
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]  
884 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV  
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL  
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]  
885 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV  
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL  
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]  
886 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV  
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL  
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]  
887 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV  
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL  
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```


7 TRATAMENTO DE ERROS

```
888 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
889 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
890 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
891 ## expr -> expr DIFERENTE expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
892 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
893 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
894 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
895 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
896 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
897 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
898 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
899 ##
900 ## The known suffix of the stack is as follows:
901 ## expr DIFERENTE expr
902 ##
903
904 <Erro: expressão incorreta>
905
906 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO DIFERENTE
    XOU
907 ##
908 ## Ends in an error in state: 81.
909 ##
```

7 TRATAMENTO DE ERROS

```
910 ## expr -> expr DIFERENTE . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
911 ##
912 ## The known suffix of the stack is as follows:
913 ## expr DIFERENTE
914 ##
915
916 <Erro: expressão incorreta>
917
918 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO DIVISAO
    VERDADEIRO VERDADEIRO
919 ##
920 ## Ends in an error in state: 66.
921 ##
922 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
923 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
924 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
925 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
926 ## expr -> expr DIVISAO expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
927 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
928 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
929 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
930 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
931 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
932 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
933 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
934 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
935 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
936 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
937 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
938 ##
939 ## The known suffix of the stack is as follows:
940 ## expr DIVISAO expr
941 ##
942
943 <Erro: expressão incorreta>
944
945 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO DIVISAO
    XOU
946 ##
947 ## Ends in an error in state: 65.
948 ##
949 ## expr -> expr DIVISAO . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
950 ##
951 ## The known suffix of the stack is as follows:
952 ## expr DIVISAO
953 ##
954
955 <Erro: expressão incorreta>
956
957 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO E
    VERDADEIRO VERDADEIRO
958 ##
959 ## Ends in an error in state: 80.
960 ##
```

7 TRATAMENTO DE ERROS

```
961 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
962 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
963 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
964 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
965 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
966 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
967 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
968 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
969 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
970 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
971 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
972 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
973 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
974 ## expr -> expr E expr . [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
975 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
976 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
977 ##
978 ## The known suffix of the stack is as follows:
979 ## expr E expr
980 ##
981
982 <Erro: expressão incorreta>
983
984 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO E XOU
985 ##
986 ## Ends in an error in state: 79.
987 ##
988 ## expr -> expr E . expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
989 ##
990 ## The known suffix of the stack is as follows:
991 ## expr E
992 ##
993
994 <Erro: expressão incorreta>
995
996 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO IGUAL
    VERDADEIRO VERDADEIRO
997 ##
998 ## Ends in an error in state: 78.
999 ##
1000 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1001 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1002 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1003 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1004 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1005 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1006 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1007 ## expr -> expr IGUAL expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1008 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1009 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1010 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1011 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1012 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1013 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1014 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1015 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1016 ##
1017 ## The known suffix of the stack is as follows:
1018 ## expr IGUAL expr
1019 ##
1020
1021 <Erro: expressão incorreta>
1022
1023 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO IGUAL XOU
1024 ##
1025 ## Ends in an error in state: 77.
1026 ##
```

7 TRATAMENTO DE ERROS

```
1027 ## expr -> expr IGUAL . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1028 ##
1029 ## The known suffix of the stack is as follows:
1030 ## expr IGUAL
1031 ##
1032
1033 <Erro: expressão incorreta>
1034
1035 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MAIOR
    VERDADEIRO VERDADEIRO
1036 ##
1037 ## Ends in an error in state: 76.
1038 ##
1039 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1040 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1041 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1042 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1043 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1044 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1045 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1046 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1047 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1048 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1049 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1050 ## expr -> expr MAIOR expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1051 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1052 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1053 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1054 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1055 ##
1056 ## The known suffix of the stack is as follows:
1057 ## expr MAIOR expr
1058 ##
1059
1060 <Erro: expressão incorreta>
1061
1062 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MAIOR XOU
1063 ##
1064 ## Ends in an error in state: 75.
1065 ##
1066 ## expr -> expr MAIOR . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1067 ##
1068 ## The known suffix of the stack is as follows:
1069 ## expr MAIOR
1070 ##
1071
1072 <Erro: expressão incorreta>
1073
1074 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO
    MAIORIGUAL VERDADEIRO VERDADEIRO
1075 ##
1076 ## Ends in an error in state: 74.
1077 ##
```


7 TRATAMENTO DE ERROS

```
1078 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1079 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1080 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1081 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1082 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1083 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1084 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1085 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1086 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1087 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1088 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1089 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1090 ## expr -> expr MAIORIGUAL expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1091 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1092 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1093 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1094 ##
1095 ## The known suffix of the stack is as follows:
1096 ## expr MAIORIGUAL expr
1097 ##
1098
1099 <Erro: expressão incorreta>
1100
1101 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO
      MAIORIGUAL XOU
1102 ##
1103 ## Ends in an error in state: 73.
1104 ##
1105 ## expr -> expr MAIORIGUAL . expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1106 ##
1107 ## The known suffix of the stack is as follows:
1108 ## expr MAIORIGUAL
1109 ##
1110
1111 <Erro: expressão incorreta>
1112
1113 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MENOR
      VERDADEIRO VERDADEIRO
1114 ##
1115 ## Ends in an error in state: 72.
1116 ##
1117 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1118 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1119 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1120 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1121 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1122 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1123 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1124 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1125 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1126 ## expr -> expr MENOR expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1127 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1128 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1129 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1130 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1131 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1132 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1133 ##
1134 ## The known suffix of the stack is as follows:
1135 ## expr MENOR expr
1136 ##
1137
1138 <Erro: expressão incorreta>
1139
1140 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MENOR XOU
1141 ##
1142 ## Ends in an error in state: 71.
1143 ##
```

7 TRATAMENTO DE ERROS

```
1144 ## expr -> expr MENOR . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1145 ##
1146 ## The known suffix of the stack is as follows:
1147 ## expr MENOR
1148 ##
1149
1150 <Erro: expressão incorreta>
1151
1152 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO
    MENORIGUAL VERDADEIRO VERDADEIRO
1153 ##
1154 ## Ends in an error in state: 70.
1155 ##
1156 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1157 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1158 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1159 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1160 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1161 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1162 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1163 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1164 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1165 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1166 ## expr -> expr MENORIGUAL expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1167 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1168 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1169 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1170 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1171 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1172 ##
1173 ## The known suffix of the stack is as follows:
1174 ## expr MENORIGUAL expr
1175 ##
1176
1177 <Erro: expressão incorreta>
1178
1179 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO
    MENORIGUAL XOU
1180 ##
1181 ## Ends in an error in state: 69.
1182 ##
1183 ## expr -> expr MENORIGUAL . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1184 ##
1185 ## The known suffix of the stack is as follows:
1186 ## expr MENORIGUAL
1187 ##
1188
1189 <Erro: expressão incorreta>
1190
1191 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MOD
    VERDADEIRO VERDADEIRO
1192 ##
1193 ## Ends in an error in state: 64.
1194 ##
```

7 TRATAMENTO DE ERROS

```
1195 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1196 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1197 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1198 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1199 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1200 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1201 ## expr -> expr MOD expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1202 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1203 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1204 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1205 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1206 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1207 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1208 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1209 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1210 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1211 ##
1212 ## The known suffix of the stack is as follows:
1213 ## expr MOD expr
1214 ##
1215
1216 <Erro: expressão incorreta>
1217
1218 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MOD XOU
1219 ##
1220 ## Ends in an error in state: 63.
1221 ##
1222 ## expr -> expr MOD . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1223 ##
1224 ## The known suffix of the stack is as follows:
1225 ## expr MOD
1226 ##
1227
1228 <Erro: expressão incorreta>
1229
1230 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MULT
    VERDADEIRO VERDADEIRO
1231 ##
1232 ## Ends in an error in state: 62.
1233 ##
1234 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1235 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1236 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1237 ## expr -> expr MULT expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1238 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1239 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1240 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1241 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1242 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1243 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1244 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1245 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1246 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1247 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1248 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1249 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1250 ##
1251 ## The known suffix of the stack is as follows:
1252 ## expr MULT expr
1253 ##
1254
1255 <Erro: expressão incorreta>
1256
1257 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO MULT XOU
1258 ##
1259 ## Ends in an error in state: 61.
1260 ##
```


7 TRATAMENTO DE ERROS

```
1261 ## expr -> expr MULT . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1262 ##
1263 ## The known suffix of the stack is as follows:
1264 ## expr MULT
1265 ##
1266
1267 <Erro: expressão incorreta>
1268
1269 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO OU
    VERDADEIRO VERDADEIRO
1270 ##
1271 ## Ends in an error in state: 84.
1272 ##
1273 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1274 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1275 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1276 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1277 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1278 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1279 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1280 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1281 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1282 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1283 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1284 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1285 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1286 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1287 ## expr -> expr OU expr . [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1288 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1289 ##
1290 ## The known suffix of the stack is as follows:
1291 ## expr OU expr
1292 ##
1293
1294 <Erro: expressão incorreta>
1295
1296 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO OU XOU
1297 ##
1298 ## Ends in an error in state: 83.
1299 ##
1300 ## expr -> expr OU . expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1301 ##
1302 ## The known suffix of the stack is as follows:
1303 ## expr OU
1304 ##
1305
1306 <Erro: expressão incorreta>
1307
1308 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO POTENCIA
    VERDADEIRO VERDADEIRO
1309 ##
1310 ## Ends in an error in state: 60.
1311 ##
```

7 TRATAMENTO DE ERROS

```
1312 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1313 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1314 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1315 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1316 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1317 ## expr -> expr POTENCIA expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1318 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1319 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1320 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1321 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1322 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1323 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1324 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1325 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1326 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1327 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1328 ##
1329 ## The known suffix of the stack is as follows:
1330 ## expr POTENCIA expr
1331 ##
1332
1333 <Erro: expressão incorreta>
1334
1335 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO POTENCIA
      XOU
1336 ##
1337 ## Ends in an error in state: 59.
1338 ##
1339 ## expr -> expr POTENCIA . expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1340 ##
1341 ## The known suffix of the stack is as follows:
1342 ## expr POTENCIA
1343 ##
1344
1345 <Erro: expressão incorreta>
1346
1347 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO SOMA
      VERDADEIRO VERDADEIRO
1348 ##
1349 ## Ends in an error in state: 68.
1350 ##
1351 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1352 ## expr -> expr SOMA expr . [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1353 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1354 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1355 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1356 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1357 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1358 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1359 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1360 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1361 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1362 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1363 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1364 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1365 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1366 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1367 ##
1368 ## The known suffix of the stack is as follows:
1369 ## expr SOMA expr
1370 ##
1371
1372 <Erro: expressão incorreta>
1373
1374 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO SOMA XOU
1375 ##
1376 ## Ends in an error in state: 67.
1377 ##
```

7 TRATAMENTO DE ERROS

```
1378 ## expr -> expr SOMA . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1379 ##
1380 ## The known suffix of the stack is as follows:
1381 ## expr SOMA
1382 ##
1383
1384 <Erro: expressão incorreta>
1385
1386 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO SUB
    VERDADEIRO VERDADEIRO
1387 ##
1388 ## Ends in an error in state: 58.
1389 ##
1390 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1391 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1392 ## expr -> expr SUB expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1393 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1394 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1395 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1396 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1397 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1398 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1399 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1400 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1401 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1402 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1403 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1404 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1405 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1406 ##
1407 ## The known suffix of the stack is as follows:
1408 ## expr SUB expr
1409 ##
1410
1411 <Erro: expressão incorreta>
1412
1413 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO SUB XOU
1414 ##
1415 ## Ends in an error in state: 57.
1416 ##
1417 ## expr -> expr SUB . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1418 ##
1419 ## The known suffix of the stack is as follows:
1420 ## expr SUB
1421 ##
1422
1423 <Erro: expressão incorreta>
1424
1425 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO VIRGULA
1426 ##
1427 ## Ends in an error in state: 101.
1428 ##
1429 ## expr -> expr . SOMA expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
```

7 TRATAMENTO DE ERROS

```

    DIFERENTE ]
1430 ## expr -> expr . SUB expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1431 ## expr -> expr . MULT expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1432 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
1433 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
1434 ## expr -> expr . MOD expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1435 ## expr -> expr . IGUAL expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1436 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
1437 ## expr -> expr . MENOR expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1438 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
1439 ## expr -> expr . MAIOR expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1440 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA PTV POTENCIA
    OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E
    DIVISAO DIFERENTE ]
1441 ## expr -> expr . E expr [ XOU SUB SOMA PTV POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1442 ## expr -> expr . OU expr [ XOU SUB SOMA PTV POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1443 ## expr -> expr . XOU expr [ XOU SUB SOMA PTV POTENCIA OU
    MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL E DIVISAO
    DIFERENTE ]
1444 ## option(expr) -> expr . [ PTV ]
1445 ##
```


7 TRATAMENTO DE ERROS

```
1446 ## The known suffix of the stack is as follows:
1447 ## expr
1448 ##
1449
1450 <Erro: expressão incorreta>
1451
1452 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO XOU
      VERDADEIRO VERDADEIRO
1453 ##
1454 ## Ends in an error in state: 56.
1455 ##
1456 ## expr -> expr . SOMA expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1457 ## expr -> expr . SUB expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1458 ## expr -> expr . MULT expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1459 ## expr -> expr . DIVISAO expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1460 ## expr -> expr . POTENCIA expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1461 ## expr -> expr . MOD expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1462 ## expr -> expr . IGUAL expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1463 ## expr -> expr . DIFERENTE expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1464 ## expr -> expr . MENOR expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1465 ## expr -> expr . MENORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1466 ## expr -> expr . MAIOR expr [ XOU VIRGULA SUB SOMA PTV
      POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
      MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1467 ## expr -> expr . MAIORIGUAL expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1468 ## expr -> expr . E expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1469 ## expr -> expr . OU expr [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1470 ## expr -> expr . XOU expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1471 ## expr -> expr XOU expr . [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1472 ##
1473 ## The known suffix of the stack is as follows:
1474 ## expr XOU expr
1475 ##
1476
1477 <Erro: expressão incorreta>
1478
1479 prog: ALGORITMO LITSTRING INICIO RETORNE VERDADEIRO XOU XOU
1480 ##
1481 ## Ends in an error in state: 55.
1482 ##
1483 ## expr -> expr XOU . expr [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1484 ##
1485 ## The known suffix of the stack is as follows:
1486 ## expr XOU
1487 ##
1488
1489 <Erro: expressão incorreta>
1490
1491 prog: ALGORITMO LITSTRING INICIO RETORNE XOU
1492 ##
1493 ## Ends in an error in state: 98.
1494 ##
1495 ## stm_ret -> RETORNE . option(expr) PTV [ SENAO SE RETORNE
    PARA OUTROCASO LEIA ID FIMSE FIMPARG FIMFUNCAO FIMESCOLHA
    FIMENQUANTO FIMALGORITMO ESCREVAL ESCREVA ESCOLHA ENQUANTO
    CASO ]
1496 ##
```

7 TRATAMENTO DE ERROS

```
1497 ## The known suffix of the stack is as follows:
1498 ## RETORNE
1499 ##
1500
1501 <Erro: após retorne>
1502
1503 prog: ALGORITMO LITSTRING INICIO SE APAR VERDADEIRO VIRGULA
1504 ##
1505 ## Ends in an error in state: 86.
1506 ##
1507 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1508 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1509 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1510 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
1511 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
1512 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1513 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1514 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
1515 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1516 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
      DIVISAO DIFERENTE ]
1517 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
      DIFERENTE ]
1518 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E
```

7 TRATAMENTO DE ERROS

```

    DIVISAO DIFERENTE ]
1519 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
    DIFERENTE ]
1520 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA OU MULT MOD
    MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
    DIFERENTE ]
1521 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA OU MULT
    MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL FPAR E DIVISAO
    DIFERENTE ]
1522 ## expr -> APAR expr . FPAR [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1523 ##
1524 ## The known suffix of the stack is as follows:
1525 ## APAR expr
1526 ##
1527
1528 <Erro: expressão incorreta>
1529
1530 prog: ALGORITMO LITSTRING INICIO SE APAR XOU
1531 ##
1532 ## Ends in an error in state: 47.
1533 ##
1534 ## expr -> APAR . expr FPAR [ XOU VIRGULA SUB SOMA PTV
    POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1535 ##
1536 ## The known suffix of the stack is as follows:
1537 ## APAR
1538 ##
1539
1540 <Erro: comando se>
1541
1542 prog: ALGORITMO LITSTRING INICIO SE ID ATRIB
1543 ##
1544 ## Ends in an error in state: 49.
1545 ##
1546 ## lvalue -> lvalue . ACOL expr FCOL [ XOU VIRGULA SUB SOMA
    PTV POTENCIA PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL
    MAIOR IGUAL FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE
    ACOL ]
1547 ## termo -> lvalue . [ XOU VIRGULA SUB SOMA PTV POTENCIA
    PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
    FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
```

7 TRATAMENTO DE ERROS

```
1548 ##
1549 ## The known suffix of the stack is as follows:
1550 ## lvalue
1551 ##
1552 ## WARNING: This example involves spurious reductions.
1553 ## This implies that, although the LR(1) items shown above
1554 ## provide an
1555 ## accurate view of the past (what has been recognized so far
1556 ## ), they
1557 ## may provide an INCOMPLETE view of the future (what was
1558 ## expected next).
1559 ## In state 43, spurious reduction of production lvalue -> ID
1560 ##
1561 <Erro: comando se>
1562
1563 prog: ALGORITMO LITSTRING INICIO SE NAO XOU
1564 ##
1565 ## Ends in an error in state: 39.
1566 ##
1567 ## expr -> NAO . termo [ XOU VIRGULA SUB SOMA PTV POTENCIA
1568 ## PASSO OU MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL
1569 ## FPAR FCOL FACA ENTAO E DIVISAO DIFERENTE ATE ]
1570 ##
1571 ## The known suffix of the stack is as follows:
1572 ## NAO
1573 ##
1574 <Erro: comando se>
1575
1576 prog: ALGORITMO LITSTRING INICIO SE VERDADEIRO ENTAO RETORNE
1577 PTV FIMPARA
1578 ##
1579 ## Ends in an error in state: 168.
1580 ##
1581 ## stm_se -> SE expr ENTAO list(stm_list) . option(stm_senao)
1582 ## FIMSE [ SENAO SE RETORNE PARA OUTROCASO LEIA ID FIMSE
1583 ## FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO FIMALGORITMO
1584 ## ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
1585 ##
1586 ## The known suffix of the stack is as follows:
1587 ## SE expr ENTAO list(stm_list)
1588 ##
1589 ## WARNING: This example involves spurious reductions.
```

7 TRATAMENTO DE ERROS

```
1583 ## This implies that, although the LR(1) items shown above
      provide an
1584 ## accurate view of the past (what has been recognized so far
      ), they
1585 ## may provide an INCOMPLETE view of the future (what was
      expected next).
1586 ## In state 142, spurious reduction of production list(
      stm_list) ->
1587 ## In state 153, spurious reduction of production list(
      stm_list) -> stm_list list(stm_list)
1588 ##
1589
1590 <Erro: comando se>
1591
1592 prog: ALGORITMO LITSTRING INICIO SE VERDADEIRO ENTAO SENAO
      RETORNE PTV SENAO
1593 ##
1594 ## Ends in an error in state: 172.
1595 ##
1596 ## stm_se -> SE expr ENTAO list(stm_list) option(stm_senao) .
      FIMSE [ SENAO SE RETORNE PARA OUTROCASO LEIA ID FIMSE
      FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO FIMALGORITMO
      ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
1597 ##
1598 ## The known suffix of the stack is as follows:
1599 ## SE expr ENTAO list(stm_list) option(stm_senao)
1600 ##
1601 ## WARNING: This example involves spurious reductions.
1602 ## This implies that, although the LR(1) items shown above
      provide an
1603 ## accurate view of the past (what has been recognized so far
      ), they
1604 ## may provide an INCOMPLETE view of the future (what was
      expected next).
1605 ## In state 142, spurious reduction of production list(
      stm_list) ->
1606 ## In state 153, spurious reduction of production list(
      stm_list) -> stm_list list(stm_list)
1607 ## In state 170, spurious reduction of production stm_senao
      -> SENAO list(stm_list)
1608 ## In state 171, spurious reduction of production option(
      stm_senao) -> stm_senao
1609 ##
1610
1611 <Erro: comando se>
```

7 TRATAMENTO DE ERROS

```
1612
1613 prog: ALGORITMO LITSTRING INICIO SE VERDADEIRO ENTAO SENAO
      XOU
1614 ##
1615 ## Ends in an error in state: 169.
1616 ##
1617 ## stm_senao -> SENAO . list(stm_list) [ FIMSE ]
1618 ##
1619 ## The known suffix of the stack is as follows:
1620 ## SENAO
1621 ##
1622
1623 <Erro: comando senao>
1624
1625 prog: ALGORITMO LITSTRING INICIO SE VERDADEIRO ENTAO XOU
1626 ##
1627 ## Ends in an error in state: 97.
1628 ##
1629 ## stm_se -> SE expr ENTAO . list(stm_list) option(stm_senao)
      FIMSE [ SENAO SE RETORNE PARA OUTROCASO LEIA ID FIMSE
      FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO FIMALGORITMO
      ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
1630 ##
1631 ## The known suffix of the stack is as follows:
1632 ## SE expr ENTAO
1633 ##
1634
1635 <Erro: comando se>
1636
1637 prog: ALGORITMO LITSTRING INICIO SE VERDADEIRO VIRGULA
1638 ##
1639 ## Ends in an error in state: 96.
1640 ##
1641 ## expr -> expr . SOMA expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1642 ## expr -> expr . SUB expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1643 ## expr -> expr . MULT expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1644 ## expr -> expr . DIVISAO expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
```

7 TRATAMENTO DE ERROS

```
1645 ## expr -> expr . POTENCIA expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1646 ## expr -> expr . MOD expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1647 ## expr -> expr . IGUAL expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1648 ## expr -> expr . DIFERENTE expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1649 ## expr -> expr . MENOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1650 ## expr -> expr . MENORIGUAL expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1651 ## expr -> expr . MAIOR expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1652 ## expr -> expr . MAIORIGUAL expr [ XOU SUB SOMA POTENCIA OU
      MULT MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1653 ## expr -> expr . E expr [ XOU SUB SOMA POTENCIA OU MULT MOD
      MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E DIVISAO
      DIFERENTE ]
1654 ## expr -> expr . OU expr [ XOU SUB SOMA POTENCIA OU MULT MOD
      MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E DIVISAO
      DIFERENTE ]
1655 ## expr -> expr . XOU expr [ XOU SUB SOMA POTENCIA OU MULT
      MOD MENORIGUAL MENOR MAIORIGUAL MAIOR IGUAL ENTAO E
      DIVISAO DIFERENTE ]
1656 ## stm_se -> SE expr . ENTAO list(stm_list) option(stm_senao)
      FIMSE [ SENAO SE RETORNE PARA OUTROCASO LEIA ID FIMSE
      FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO FIMALGORITMO
      ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
1657 ##
1658 ## The known suffix of the stack is as follows:
1659 ## SE expr
1660 ##
1661
1662 <Erro: comando se>
1663
1664 prog: ALGORITMO LITSTRING INICIO SE XOU
```


7 TRATAMENTO DE ERROS

```
1665 ##
1666 ## Ends in an error in state: 37.
1667 ##
1668 ## stm_se -> SE . expr ENTAO list(stm_list) option(stm_senao)
      FIMSE [ SENAO SE RETORNE PARA OUTROCASO LEIA ID FIMSE
      FIMPARA FIMFUNCAO FIMESCOLHA FIMENQUANTO FIMALGORITMO
      ESCREVAL ESCREVA ESCOLHA ENQUANTO CASO ]
1669 ##
1670 ## The known suffix of the stack is as follows:
1671 ## SE
1672 ##
1673
1674 <Erro: comando se>
1675
1676 prog: ALGORITMO LITSTRING INICIO XOU
1677 ##
1678 ## Ends in an error in state: 183.
1679 ##
1680 ## stm_block -> INICIO . list(stm_list) FIMALGORITMO [ EOF ]
1681 ##
1682 ## The known suffix of the stack is as follows:
1683 ## INICIO
1684 ##
1685
1686 <Erro: após palavra reservada início>
1687
1688 prog: ALGORITMO LITSTRING VAR ID DECLARA CARACTER PTV XOU
1689 ##
1690 ## Ends in an error in state: 9.
1691 ##
1692 ## list(var_decl) -> var_decl . list(var_decl) [ INICIO
      FUNCAO ]
1693 ##
1694 ## The known suffix of the stack is as follows:
1695 ## var_decl
1696 ##
1697
1698 <Erro: declaração de variáveis incorreta>
1699
1700 prog: ALGORITMO LITSTRING VAR ID DECLARA REAL XOU
1701 ##
1702 ## Ends in an error in state: 16.
1703 ##
1704 ## var_decl -> separated_nonempty_list(VIRGULA, ID) DECLARA
      tp_primitivo . PTV [ INICIO ID FUNCAO ]
```

7 TRATAMENTO DE ERROS

```
1705 ##
1706 ## The known suffix of the stack is as follows:
1707 ## separated_nonempty_list(VIRGULA,ID) DECLARA tp_primitivo
1708 ##
1709
1710 <Erro: declaração de variáveis incorreta>
1711
1712 prog: ALGORITMO LITSTRING VAR ID DECLARA XOU
1713 ##
1714 ## Ends in an error in state: 11.
1715 ##
1716 ## var_decl -> separated_nonempty_list(VIRGULA,ID) DECLARA .
1717 ##          tp_primitivo PTV [ INICIO ID FUNCAO ]
1718 ##
1719 ## The known suffix of the stack is as follows:
1720 ## separated_nonempty_list(VIRGULA,ID) DECLARA
1721 ##
1722 <Erro: declaração de variáveis incorreta>
1723
1724 prog: ALGORITMO LITSTRING VAR ID VIRGULA XOU
1725 ##
1726 ## Ends in an error in state: 7.
1727 ##
1728 ## separated_nonempty_list(VIRGULA,ID) -> ID VIRGULA .
1729 ##          separated_nonempty_list(VIRGULA,ID) [ DECLARA ]
1730 ##
1731 ## The known suffix of the stack is as follows:
1732 ## ID VIRGULA
1733 ##
1734 <Erro: declaração de variáveis incorreta>
1735
1736 prog: ALGORITMO LITSTRING VAR ID XOU
1737 ##
1738 ## Ends in an error in state: 6.
1739 ##
1740 ## separated_nonempty_list(VIRGULA,ID) -> ID . [ DECLARA ]
1741 ## separated_nonempty_list(VIRGULA,ID) -> ID . VIRGULA
1742 ##          separated_nonempty_list(VIRGULA,ID) [ DECLARA ]
1743 ##
1744 ## The known suffix of the stack is as follows:
1745 ## ID
1746 ##
```

7 TRATAMENTO DE ERROS

```
1747 <Erro: declaração de variáveis incorreta>
1748
1749 prog: ALGORITMO LITSTRING VAR XOU
1750 ##
1751 ## Ends in an error in state: 5.
1752 ##
1753 ## var_decl_block -> VAR . list(var_decl) [ INICIO FUNCAO ]
1754 ##
1755 ## The known suffix of the stack is as follows:
1756 ## VAR
1757 ##
1758
1759 <Erro: declaração de variáveis incorreta>
1760
1761 prog: ALGORITMO LITSTRING XOU
1762 ##
1763 ## Ends in an error in state: 4.
1764 ##
1765 ## prog -> declaracao_algoritmo . option(var_decl_block) list
1766 ##      (func_decl) stm_block EOF [ # ]
1767 ##
1768 ## The known suffix of the stack is as follows:
1769 ## declaracao_algoritmo
1770 ##
1771 <Erro: após palavra reservada algoritmo>
1772
1773 prog: ALGORITMO XOU
1774 ##
1775 ## Ends in an error in state: 1.
1776 ##
1777 ## declaracao_algoritmo -> ALGORITMO . LITSTRING [ VAR INICIO
1778 ##      FUNCAO ]
1779 ##
1780 ## The known suffix of the stack is as follows:
1781 ## ALGORITMO
1782 ##
1783 <Erro: após palavra reservada algoritmo>
1784
1785 prog: XOU
1786 ##
1787 ## Ends in an error in state: 0.
1788 ##
1789 ## prog' -> . prog [ # ]
```

7 TRATAMENTO DE ERROS

```
1790 ##
1791 ## The known suffix of the stack is as follows:
1792 ##
1793 ##
1794
1795 <Erro: início do programa incorreto>
```

7.3 Gerando arquivo .ml com as mensagens de erro

```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoTarefa5/portugol
$ menhir -v --list-errors sintatico.mly --compile-errors sintatico.msg > erroSint.ml
```

Figura 24: Arquivo .ml de mensagens de erro

7.4 Construindo analisador sintatico com mensagens de erro

```
eduardo@eduardopc:~/Documentos/Faculdade/Compiladores/AnalizadorSintaticoTarefa5/portugol
$ ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package menhirLib sintaticoTest.byte
```

Figura 25: Construindo projeto com mensagens de erro

7.5 Exemplo de mensagem de erro

```
$ rlwrap ocaml
OCaml version 4.02.3

Findlib has been successfully loaded. Additional directives:
#require "package";;      to load a package
#list;;                  to list the available packages
#camlp4o;;               to load camlp4 (standard syntax)
#camlp4r;;               to load camlp4 (revised syntax)
#predicates "p,q,...";;  to set these predicates
Topfind.reset();;        to force that packages will be reloaded
#thread;;                to enable threads

/home/eduardo/.opam/system/lib/menhirLib: added to search path
/home/eduardo/.opam/system/lib/menhirLib/menhirLib.cmo: loaded
# parse_arq "codigosportugol/micro10.ptg";;
Erro lexico na linha 23, coluna 1:
  Caracter desconhecido: @
- : Ast.prog option option = None
# parse_arq "codigosportugol/micro10.ptg";;
Erro sintático na linha 24, coluna 28 338 - <Erro: comando escreva>
.
- : Ast.prog option option = None
#
```

Figura 26: Erro léxico e sintático

8 Analisador Semântico

Nesta seção será descrito o analisador semântico para a linguagem **Portugol**. A tarefa do analisador semântico é receber a árvore sintática, construída pelo analisador sintático, e verificar se os tipos de seus elementos estão corretos.

Segue os códigos responsáveis por tal tarefa.

8.1 Árvore sintática

Listing 57: ast.ml

```
1
2 open Lexing
3
4 type identificador = string
5 type 'a pos = 'a * Lexing.position (* tipo e posição no
   arquivo fonte *)
```

```
6
7 type 'expr prog = Prog of var_decl * ('expr func_decl_list) *
  ('expr statements)
8 and declaracao_algoritmo = DeclAlg
9 and var_decl = vars list
10 and vars = DecVar of (identificador pos) * tipo
11 and 'expr func_decl_list = ('expr func_decl) list
12 and 'expr func_decl = FuncDecl of ('expr decfun)
13 and fparams = fparam list
14 and fparam = identificador * tipo
15 and functype = tipo
16 and 'expr funcbloc = ('expr stm_list) list
17 and 'expr statements = ('expr stm_list) list
18
19 and 'expr decfun = {
20   fn_id: identificador pos;
21   fn_params: (identificador pos * tipo) list;
22   fn_tiporet: functype;
23   fn_locais: var_decl;
24   fn_corpo: 'expr funcbloc;
25 }
26 }
27
28 and tipo = TipoInteiro
29         | TipoReal
30         | TipoBooleano
31         | TipoVoid
32         | TipoCaractere
33
34 and 'expr stm_list = Attrib of 'expr * 'expr
35                   | Chamada of 'expr
36                   | Retorne of 'expr option
37                   | Se of 'expr * 'expr stm_list list * ('expr senao)
38                     option
39                   | Escolha of identificador pos * 'expr case list *
40                     'expr stm_list list
41                   | Para of ('expr) * 'expr * 'expr * ('expr ) * '
42                     expr statements
43                   | Leia of 'expr list
44                   | Escreva of 'expr list
45                   | Escreval of 'expr list
46                   | Enquanto of 'expr * 'expr stm_list list
47
48 and 'expr senao = 'expr stm_list list
```

8 ANALISADOR SEMÂNTICO

```
47
48 and 'expr case = CaseInt of int pos * 'expr stm_list list
49     | CaseChar of char pos * 'expr stm_list list
50
51 and 'expr lvalue = Var of identificador pos
52     | VarElement of ('expr lvalue) * 'expr
53
54 and op = Soma
55     | Subtracao
56     | Multiplicacao
57     | Divisao
58     | Potencia
59     | Modulo
60     | Igual
61     | Diferente
62     | Menor
63     | MenorIgual
64     | Maior
65     | MaiorIgual
66     | ELogico
67     | OuLogico
68     | XouLogico
69
70 (*let arguments:
71     match args with
72     None -> []
73     Some xs -> xs*)
74 and 'expr fargs = 'expr list
75
76 and logico_value = Verdadeiro of bool
77     | Falso of bool
```

8.2 Definição de uma expressão para árvore sintática

Listing 58: sast.ml

```
1 open Ast
2
3 type expressao =
4     | ExpOp of op pos * expressao * expressao
5     | ExpFunCall of identificador pos * expressao fargs
6     | ExpString of string pos
7     | ExpInt of int pos
8     | ExpFloat of float pos
9     | ExpChar of char pos
```

```
10      |ExpBool of bool pos
11      |ExpVar of (expressao lvalue)
```

8.3 Definição de uma expressão para árvore semântica

Listing 59: tast.ml

```
1 open Ast
2
3 type expressao = ExpVar of (expressao lvalue) * tipo
4               |ExpOp of (op * tipo) * (expressao * tipo) * (
5                 expressao * tipo)
6               |ExpFunCall of identificador * expressao fargs * tipo
7               |ExpString of string * tipo
8               |ExpInt of int * tipo
9               |ExpFloat of float * tipo
10              |ExpChar of char * tipo
11              |ExpBool of bool * tipo
```

8.4 Definição do Ambiente

Listing 60: ambiente.ml

```
1
2 module Tab = Tabsimb
3 module A = Ast
4
5 type entrada_fn = { tipo_fn: A.tipo;
6                     formais: (string * A.tipo) list;
7 }
8
9 type entrada = EntFun of entrada_fn
10              | EntVar of A.tipo
11
12 type t = {
13   ambv : entrada Tab.tabela
14 }
15
16 let novo_amb xs = { ambv = Tab.cria xs }
17
18 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
19
20 let busca amb ch = Tab.busca amb.ambv ch
21
22 let insere_local amb ch t =
```



```
23 Tab.inserere amb.ambv ch (EntVar t)
24
25 let insere_param amb ch t =
26   Tab.inserere amb.ambv ch (EntVar t)
27
28 let insere_fun amb nome params resultado =
29   let ef = EntFun { tipo_fn = resultado;
30                     formais = params }
31   in Tab.inserere amb.ambv nome ef
```

8.5 Definição da tabela de símbolos

Listing 61: tabsimb.ml

```
1
2 type 'a tabela = {
3   tbl: (string, 'a) Hashtbl.t;
4   pai: 'a tabela option;
5 }
6
7 exception Entrada_existente of string;;
8
9 let insere amb ch v =
10   if Hashtbl.mem amb.tbl ch
11   then raise (Entrada_existente ch)
12   else Hashtbl.add amb.tbl ch v
13
14 let substitui amb ch v = Hashtbl.replace amb.tbl ch v
15
16 let rec atualiza amb ch v =
17   if Hashtbl.mem amb.tbl ch
18   then Hashtbl.replace amb.tbl ch v
19   else match amb.pai with
20     None -> failwith "tabsim atualiza: chave nao
21                       encontrada"
22     | Some a -> atualiza a ch v
23
24 let rec busca amb ch =
25   try Hashtbl.find amb.tbl ch
26   with Not_found ->
27     (match amb.pai with
28       None -> raise Not_found
29       | Some a -> busca a ch)
30
31 let rec cria cvs =
```

8 ANALISADOR SEMÂNTICO

```
31 let amb = {
32     tbl = Hashtbl.create 5;
33     pai = None
34 } in
35 let _ = List.iter (fun (c,v) -> insere amb c v) cvs
36 in amb
37
38 let novo_escopo amb_pai = {
39     tbl = Hashtbl.create 5;
40     pai = Some amb_pai
41 }
```

8.6 Definição do analisador semântico

Listing 62: semantico.ml

```
1 module Amb = Ambiente
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6
7 let rec posicao exp = let open S in
8     match exp with
9     | ExpVar v -> (match v with
10         | A.Var (_,pos) -> pos
11         | A.VarElement (_,exp2) -> posicao exp2
12     )
13     (*| ExpNot (_, pos) -> pos *)
14     | ExpInt (_,pos) -> pos
15     | ExpFloat (_,pos) -> pos
16     | ExpChar (_,pos) -> pos
17     | ExpString (_,pos) -> pos
18     | ExpBool (_,pos) -> pos
19     | ExpOp ((_,pos),_,_) -> pos
20     | ExpFunCall ((_,pos), _) -> pos
21
22
23 type classe_op = Aritmetico | Relacional | Logico
24
25 let classifica op =
26     let open A in
27     match op with
28     | OuLogico
29     | ELogico
```

8 ANALISADOR SEMÂNTICO

```
30 | XouLogico -> Logico
31 | Menor
32 | MenorIgual
33 | Maior
34 | MaiorIgual
35 | Igual
36 | Diferente -> Relacional
37 | Soma
38 | Subtracao
39 | Multiplicacao
40 | Divisao
41 | Potencia
42 | Modulo -> Aritmetico
43
44 let msg_erro_pos pos msg =
45   let open Lexing in
46   let lin = pos.pos_lnum
47   and col = pos.pos_cnum - pos.pos_bol - 1 in
48   Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin
     col msg
49
50 let msg_erro nome msg =
51   let pos = snd nome in
52   msg_erro_pos pos msg
53
54 let nome_tipo t =
55   let open A in
56   match t with
57     TipoInteiro -> "inteiro"
58   | TipoCaractere -> "caractere"
59   | TipoBooleano -> "logico"
60   | TipoReal -> "real"
61   | TipoVoid -> "vazio"
62
63
64 let mesmo_tipo pos msg tinf tdec =
65   if tinf <> tdec
66   then
67     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo
68       tdec) in
69     failwith (msg_erro_pos pos msg)
70
71 let rec infere_exp amb exp =
72   match exp with
73     S.ExpInt n -> (T.ExpInt (fst n, A.TipoInteiro),
```

```

    A.TipoInteiro)
73 | S.ExpString s -> (T.ExpString (fst s, A.TipoCaractere), A
    .TipoCaractere)
74 | S.ExpBool b   -> (T.ExpBool (fst b, A.TipoBooleano),
    A.TipoBooleano)
75 | S.ExpFloat f  -> (T.ExpFloat (fst f, A.TipoReal), A.
    TipoReal)
76 | S.ExpChar c   -> (T.ExpChar (fst c, A.TipoCaractere), A.
    TipoCaractere)
77 | S.ExpVar v ->
78   (match v with
79     A.Var nome ->
80       (* Tenta encontrar a definição da variável no escopo
81         local, se não *)
81       (* encontrar tenta novamente no escopo que engloba o
82         atual. Prossegue-se *)
82       (* assim até encontrar a definição em algum escopo
83         englobante ou até *)
83       (* encontrar o escopo global. Se em algum lugar for
84         encontrado, *)
84       (* devolve-se a definição. Em caso contrário, devolve
85         uma exceção *)
85       let id = fst nome in
86       (try (match (Amb.busca amb id) with
87         | Amb.EntVar tipo -> (T.ExpVar (A.Var nome,
88           tipo), tipo)
88         | Amb.EntFun _ ->
89           let msg = "nome de funcao usado como nome de
90             variavel: " ^ id in
91             failwith (msg_erro nome msg)
92         )
93       with Not_found ->
94         let msg = "A variavel " ^ id ^ " nao foi
95           declarada" in
96         failwith (msg_erro nome msg)
97       )
98   | _ -> failwith "infere_exp: não implementado"
99 )
100 | S.ExpOp (op, esq, dir) ->
101   let (esq, tesq) = infere_exp amb esq
102   and (dir, tdir) = infere_exp amb dir in
103   let verifica_aritmetico () =
104     (match tesq with
```

8 ANALISADOR SEMÂNTICO

```
105         A.TipoInteiro
106     | A.TipoReal ->
107         let _ = mesmo_tipo (snd op)
108             "O operando esquerdo eh do tipo %s mas
              o direito eh do tipo %s"
109             tesq tdir
110     in tesq (* O tipo da expressão aritmética como um
              todo *)
111
112     | t -> let msg = "um operador aritmetico nao pode ser
              usado com o tipo " ^
113             (nome_tipo t)
114     in failwith (msg_erro_pos (snd op) msg)
115 )
116
117 and verifica_relacional () =
118     (match tesq with
119     | A.TipoInteiro
120     | A.TipoReal
121     | A.TipoCaractere ->
122         let _ = mesmo_tipo (snd op)
123             "O operando esquerdo eh do tipo %s mas o
              direito eh do tipo %s"
124             tesq tdir
125     in A.TipoBooleano (* O tipo da expressão relacional
              é sempre booleano *)
126
127     | t -> let msg = "um operador relacional nao pode ser
              usado com o tipo " ^
128             (nome_tipo t)
129     in failwith (msg_erro_pos (snd op) msg)
130 )
131
132 and verifica_logico () =
133     (match tesq with
134     | A.TipoBooleano ->
135         let _ = mesmo_tipo (snd op)
136             "O operando esquerdo eh do tipo %s mas o
              direito eh do tipo %s"
137             tesq tdir
138     in A.TipoBooleano (* O tipo da expressão lógica é
              sempre booleano *)
139
140     | t -> let msg = "um operador logico nao pode ser
              usado com o tipo " ^
```

```
141         (nome_tipo t)
142         in failwith (msg_erro_pos (snd op) msg)
143     )
144     (* and verifica_cadeia () =
145       (match tesq with
146       A.TipoCaractere ->
147         let _ = mesmo_tipo (snd op)
148           "0 operando esquerdo eh do tipo %s mas o
149             direito eh do tipo %s"
150           tesq tdir
151       in A.TipoCaractere (* 0 tipo da expressão relacional
152         é sempre string *)
153
154       | t -> let msg = "um operador relacional nao pode ser
155         usado com o tipo " ^
156           (nome_tipo t)
157         in failwith (msg_erro_pos (snd op) msg)
158       )
159     *)
160     in
161     let op = fst op in
162     let tinf = (match (classifica op) with
163       Aritmetico -> verifica_aritmetico ()
164       | Relacional -> verifica_relacional ()
165       | Logico -> verifica_logico ()
166     )
167     in
168     (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
169
170 | S.ExpFunCall (nome, args) ->
171   let rec verifica_parametros ags ps fs =
172     match (ags, ps, fs) with
173     (a::ags), (p::ps), (f::fs) ->
174       let _ = mesmo_tipo (posicao a)
175         "0 parametro eh do tipo %s mas deveria
176           ser do tipo %s" p f
177       in verifica_parametros ags ps fs
178     | [], [], [] -> ()
179     | _ -> failwith (msg_erro nome "Numero incorreto de
180       parametros")
181   in
182   let id = fst nome in
183   try
184     begin
185       let open Amb in
```

```

181
182     match (Amb.busca amb id) with
183     (* verifica se 'nome' está associada a uma função *)
184     Amb.EntFun {tipo_fn; formais} ->
185     (* Infere o tipo de cada um dos argumentos *)
186     let argst = List.map (infere_exp amb) args
187     (* Obtem o tipo de cada parâmetro formal *)
188     and tipos_formais = List.map snd formais in
189     (* Verifica se o tipo de cada argumento confere
190       com o tipo declarado *)
191     (* do parâmetro formal correspondente.
192       *)
193     let _ = verifica_parametros args (List.map snd
194       argst) tipos_formais
195     in (T.ExpFunCall (id, (List.map fst argst),
196       tipo_fn), tipo_fn)
197 | Amb.EntVar _ -> (* Se estiver associada a uma variável,
198   falhe *)
199   let msg = id ^ " eh uma variavel e nao uma funcao"
200   in
201   failwith (msg_erro nome msg)
202 end
203 with Not_found ->
204   let msg = "Nao existe a funcao de nome " ^ id in
205   failwith (msg_erro nome msg)
206
207 let rec verifica_cmd amb tiporet cmd =
208   let open A in
209   match cmd with
210   Retorne exp ->
211   (match exp with
212   (* Se a função não retornar nada, verifica se ela foi
213     declarada como void *)
214   None ->
215   let _ = mesmo_tipo (Lexing.dummy_pos)
216     "0 tipo retornado eh %s mas foi declarado
217     como %s"
218     TipoVoid tiporet
219   in Retorne None
220 | Some e ->
221   (* Verifica se o tipo inferido para a expressão de
222     retorno confere com o *)
223   (* tipo declarado para a função.
224     *)
225   let (e1,tinf) = infere_exp amb e in

```

8 ANALISADOR SEMÂNTICO

```
216         let _ = mesmo_tipo (posicao e)
217             "O tipo retornado eh %s mas foi
                declarado como %s"
218             tinf tiporet
219         in Retorne (Some e1)
220     )
221 | Se (teste, entao, senao) ->
222     let (teste1,tinf) = infere_exp amb teste in
223     (* O tipo inferido para a expressão 'teste' do
        condicional deve ser booleano *)
224     let _ = mesmo_tipo (posicao teste)
225         "O teste do if deveria ser do tipo %s e nao %s"
226         TipoBooleano tinf in
227     (* Verifica a validade de cada comando do bloco 'então'
        *)
228     let entao1 = List.map (verifica_cmd amb tiporet) entao in
229     (* Verifica a validade de cada comando do bloco 'senão',
        se houver *)
230     let senao1 =
231         match senao with
232         None -> None
233         | Some bloco -> Some (List.map (verifica_cmd amb
                tiporet) bloco)
234     in
235     Se (teste1, entao1, senao1)
236
237 | Attrib (elem, exp) ->
238     (* Infere o tipo da expressão no lado direito da atribuiç
        ão *)
239     let (exp, tdir) = infere_exp amb exp
240     (* Faz o mesmo para o lado esquerdo *)
241     and (elem1, tesq) = infere_exp amb elem in
242     (* Os dois tipos devem ser iguais *)
243     let _ = mesmo_tipo (posicao elem)
244         "Atribuicao com tipos diferentes: %s =
                %s" tesq tdir
245     in Attrib (elem1, exp)
246
247 | Enquanto (teste, corpo) ->
248     let (teste_tipo,tinf) = infere_exp amb teste in
249     let _ = mesmo_tipo (posicao teste)
250         "O teste do enquanto deveria ser do
                tipo %s e nao %s"
251         TipoBooleano tinf in
252     let corpo_tipo = List.map (verifica_cmd amb tiporet)
```



```

    corpo in
253   Enquanto (teste_tipo, corpo_tipo)
254
255 | Para (var, inicio, fim, avanco, corpo) ->
256   let (var_tipo, tinfv) = infere_exp amb var in
257   let (inicio_tipo, tinfi) = infere_exp amb inicio in
258   let (fim_tipo, tinff) = infere_exp amb fim in
259   let (avanco_tipo, tinfa) = infere_exp amb avanco in
260
261   let _ = mesmo_tipo (posicao var)
262     "A variável deveria ser do tipo %s e nao %s"
263     TipoInteiro tinfv in
264   let _ = mesmo_tipo (posicao inicio)
265     "O comando DE deveria ser do tipo %s e nao %s"
266     TipoInteiro tinfi in
267   let _ = mesmo_tipo (posicao fim)
268     "O comando ATE deveria ser do tipo %s e nao %s"
269     TipoInteiro tinff in
270   let _ = mesmo_tipo (posicao avanco)
271     "O comando PASSO deveria ser do tipo %s e nao %s"
272     TipoInteiro tinfa in
273
274   let corpo_tipo = List.map (verifica_cmd amb tiporet)
    corpo in
275   Para (var_tipo, inicio_tipo, fim_tipo, avanco_tipo,
    corpo_tipo)
276
277 | Escolha (id, caso, corpo) ->
278   let (teste_id, tinf) = infere_exp amb id in
279   let caso_tipo = List.map (fun s ->
280     match s with
281       (tipob, exp) ->
282         let (tipob1, tinfb) = infere_exp amb tipob in
283         let exp1 = List.map (verifica_cmd amb tiporet)
            exp in
284         (tipob1, exp1)
285     ) caso in
286   let corpo_tipo = List.map (verifica_cmd amb tiporet)
    corpo in
287   Escolha (teste_id, caso_tipo, corpo_tipo)
288
289
290 | Chamada exp ->
291   (* Verifica o tipo de cada argumento da função 'entrada'
    *)
```

8 ANALISADOR SEMÂNTICO

```
292   let (exp,tinf) = infere_exp amb exp in
293   Chamada exp
294
295 | Leia exps ->
296   (* Verifica o tipo de cada argumento da função 'entrada'
297   *)
297   let exps = List.map (infere_exp amb) exps in
298   Leia (List.map fst exps)
299
300 | Escreva exps ->
301   (* Verifica o tipo de cada argumento da função 'saida' *)
302   let exps = List.map (infere_exp amb) exps in
303   Escreva (List.map fst exps)
304
305 | Escreval exps ->
306   (* Verifica o tipo de cada argumento da função 'saida' *)
307   let exps = List.map (infere_exp amb) exps in
308   Escreval (List.map fst exps)
309
310 and verifica_fun amb ast =
311   let open A in
312   match ast with
313   | A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
314     fn_corpo} ->
315     (* Estende o ambiente global, adicionando um ambiente
316     local *)
317     let ambfn = Amb.novo_escopo amb in
318     (* Insere os parâmetros no novo ambiente *)
319     let insere_parametro (v,t) = Amb.insere_param ambfn (fst
320       v) t in
321     let _ = List.iter insere_parametro fn_params in
322     (* Insere as variáveis locais no novo ambiente *)
323     let insere_local = function
324       (DecVar(v,t)) -> Amb.insere_local ambfn (fst v) t in
325     let _ = List.iter insere_local fn_locais in
326     (* Verifica cada comando presente no corpo da função
327     usando o novo ambiente *)
328     let corpo_tipado = List.map (verifica_cmd ambfn
329       fn_tiporet) fn_corpo in
330     A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
331       fn_corpo = corpo_tipado}
332
333 let rec verifica_dup xs =
334   match xs with
```

```
330 [] -> []
331 | (nome,t)::xs ->
332   let id = fst nome in
333   if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
334   then (id, t) :: verifica_dup xs
335   else let msg = "Parametro duplicado " ^ id in
336         failwith (msg_erro nome msg)
337
338 let insere_declaracao_var amb dec =
339   let open A in
340     match dec with
341     | DecVar(nome, tipo) -> Amb.insere_local amb (fst nome)
342       tipo
343
344 let insere_declaracao_fun amb dec =
345   let open A in
346     match dec with
347     | FuncDecl {fn_id; fn_params; fn_tiporet; fn_corpo} ->
348       (* Verifica se não há parâmetros duplicados *)
349       let formais = verifica_dup fn_params in
350       let nome = fst fn_id in
351       Amb.insere_fun amb nome formais fn_tiporet
352
353 (*let analisa_dec amb dec =
354   let open A in
355     match dec with
356     | DecVar(nome, tipo) -> Amb.insere_local amb (fst nome)
357       tipo
358     | FuncDecl {fn_id; fn_params; fn_tiporet; fn_corpo} ->
359       (* Verifica se não há parâmetros duplicados *)
360       let _ = verifica_dup fn_params in
361       Amb.insere_fun amb fn_id fn_params fn_tiporet
362
363 let verifica_dec amb dec =
364   let open A in
365     match amb with
366     | A.DecVar {nome; tipo} ->
367       A.DecVar {nome; tipo}
368     | A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
369       fn_corpo} ->
370       (* Estende o ambiente global, adicionando um ambiente
371         local *)
372       let ambfn = Amb.novo_escopo amb in
373       (* Insere os parâmetros no novo ambiente *)
374       let insere_parametro (v,t) = Amb.insere_param ambfn (fst
```

```

    v) t in
371   let _ = List.iter insere_parametro fn_params in
372   (* Insere as variáveis locais no novo ambiente *)
373   let insere_local = function
374     (DecVar(v,t)) -> Amb.insere_local ambfn (fst v) t in
375   let _ = List.iter insere_local fn_locais in
376   (* Verifica cada comando presente no corpo da função
      usando o novo ambiente *)
377   let corpo_tipado = List.map (verifica_cmd ambfn
      fn_tiporet) fn_corpo in
378   A.FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
      fn_corpo = corpo_tipado}
379 *)
380
381 (* Lista de cabeçalhos das funções pré definidas *)
382 let fn_predefs = let open A in [
383   ("escreva", [("x", TipoCaractere); ("y", TipoInteiro)],
      TipoVoid);
384   ("leia",    [("x", TipoReal); ("y", TipoInteiro)], TipoVoid
      )
385 ]
386
387 (* insere as funções pré definidas no ambiente global *)
388 let declara_predefinidas amb =
389   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr)
      fn_predefs
390
391 let semantico ast =
392   (* cria ambiente global inicialmente vazio *)
393   let amb_global = Amb.novo_amb [] in
394   let _ = declara_predefinidas amb_global in
395   let (A.Prog (decs_globais, decs_funs, corpo)) = ast in
396   let _ = List.iter (insere_declaracao_var amb_global)
      decs_globais in
397   let _ = List.iter (insere_declaracao_fun amb_global)
      decs_funs in
398   (* Verificação de tipos nas funções *)
399   let decs_funs = List.map (verifica_fun amb_global)
      decs_funs in
400   (* Verificação de tipos na função principal *)
401   let corpo = List.map (verifica_cmd amb_global A.TipoVoid)
      corpo in
402   (A.Prog (decs_globais, decs_funs, corpo), amb_global)
```

9 Interpretador

O Interpretador, implementado neste trabalho, é responsável por interpretar o código fonte da linguagem Portugol e executar os comandos definidos nesta.

9.1 Código do interpretador

Listing 63: interprete.ml

```
1 module Amb = AmbInterp
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6 exception Valor_de_retorno of T.expressao
7
8 let obtem_nome_tipo_var exp = let open T in
9   match exp with
10  | ExpVar (v,tipo) ->
11    (match v with
12     | A.Var (nome,_) -> (nome,tipo)
13     | _ -> failwith "obtem_nome_tipo_var: nao implementado")
14  | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
15
16
17 let pega_int exp =
18   match exp with
19   | T.ExpInt (i,_) -> i
20   | _ -> failwith "pega_int: nao eh inteiro"
21
22 let pega_float exp =
23   match exp with
24   | T.ExpFloat (f, _) -> f
25   | _ -> failwith "pega_float: nao eh float"
26
27 let pega_string exp =
28   match exp with
29   | T.ExpString (s,_) -> s
30   | _ -> failwith "pega_string: nao eh string"
31
32 let pega_bool exp =
33   match exp with
34   | T.ExpBool (b,_) -> b
```

9 INTERPRETADOR

```
35 | _ -> failwith "pega_bool: nao eh booleano"
36
37 type classe_op = Aritmetico | Relacional | Logico
38
39 let classifica op =
40   let open A in
41   match op with
42   | OuLogico
43   | ELogico
44   | XouLogico -> Logico
45   | Menor
46   | MenorIgual
47   | Maior
48   | MaiorIgual
49   | Igual
50   | Diferente -> Relacional
51   | Soma
52   | Subtracao
53   | Multiplicacao
54   | Divisao
55   | Potencia
56   | Modulo -> Aritmetico
57
58
59 let rec interpreta_exp amb exp =
60   let open A in
61   let open T in
62   match exp with
63   | ExpVoid
64   | ExpInt _
65   | ExpString _
66   | ExpFloat _
67   | ExpChar _
68   | ExpBool _ -> exp
69   | ExpVar _ ->
70     let (id,tipo) = obtem_nome_tipo_var exp in
71     (* Tenta encontrar o valor da variável no escopo local,
72        se não *)
72     (* encontrar, tenta novamente no escopo que engloba o
73        atual. Prossegue-se *)
73     (* assim até encontrar o valor em algum escopo englobante
74        ou até *)
74     (* encontrar o escopo global. Se em algum lugar for
75        encontrado, *)
```

9 INTERPRETADOR

```
75     (* devolve-se o valor. Em caso contrário, devolve uma
76        exceção *)
76     (match (Amb.busca amb id) with
77     | Amb.EntVar (tipo, v) ->
78         (match v with
79         | None -> failwith ("variável nao inicializada: " ^
80                             id)
81         | Some valor -> valor
82         )
83     | _ -> failwith "interpreta_exp: expvar"
84 )
84 | ExpOp ((op,top), (esq, tesq), (dir,tdir)) ->
85     let vesq = interpreta_exp amb esq
86     and vdir = interpreta_exp amb dir in
87
88     let interpreta_aritmetico () =
89         (match tesq with
90         | TipoInteiro ->
91             (match op with
92             | Soma -> ExpInt (pega_int vesq + pega_int vdir
93                               , top)
94             | Subtracao -> ExpInt (pega_int vesq - pega_int
95                                   vdir, top)
96             | Multiplicacao -> ExpInt (pega_int vesq *
97                                       pega_int vdir, top)
98             | Divisao -> ExpInt (pega_int vesq / pega_int
99                                   vdir, top)
100             | _ -> failwith "interpreta_aritmetico"
101             )
102         | TipoReal ->
103             (match op with
104             | Soma -> ExpFloat (pega_float vesq +.
105                                 pega_float vdir, top)
106             | Subtracao -> ExpFloat (pega_float vesq -.
107                                       pega_float vdir, top)
108             | Multiplicacao -> ExpFloat (pega_float vesq *.
109                                           pega_float vdir, top)
110             | Divisao -> ExpFloat (pega_float vesq /.
111                                       pega_float vdir, top)
112             | _ -> failwith "interpreta_aritmetico"
113             )
114         | _ -> failwith "interpreta_aritmetico"
115         )
116     )
117
118     and interpreta_relacional () =
```

9 INTERPRETADOR

```
110     (match tesq with
111     | TipoInteiro ->
112       (match op with
113       | Menor -> ExpBool (pega_int vesq < pega_int vdir,
114                           top)
114       | MenorIgual -> ExpBool (pega_int vesq <= pega_int
115                               vdir, top)
115       | Maior -> ExpBool (pega_int vesq > pega_int vdir,
116                           top)
116       | MaiorIgual -> ExpBool (pega_int vesq >= pega_int
117                               vdir, top)
117       | Igual -> ExpBool (pega_int vesq == pega_int
118                           vdir, top)
118       | Diferente -> ExpBool (pega_int vesq != pega_int
119                               vdir, top)
119       | _ -> failwith "interpreta_relacional"
120     )
121   | TipoReal ->
122     (match op with
123     | Menor -> ExpBool (pega_float vesq < pega_float
124                         vdir, top)
124     | MenorIgual -> ExpBool (pega_float vesq <=
125                             pega_float vdir, top)
125     | Maior -> ExpBool (pega_float vesq > pega_float
126                         vdir, top)
126     | MaiorIgual -> ExpBool (pega_float vesq >=
127                             pega_float vdir, top)
127     | Igual -> ExpBool (pega_float vesq == pega_float
128                         vdir, top)
128     | Diferente -> ExpBool (pega_float vesq !=
129                             pega_float vdir, top)
129     | _ -> failwith "interpreta_relacional"
130   )
131   | TipoCaractere ->
132     (match op with
133     | Menor -> ExpBool (pega_string vesq < pega_string
134                         vdir, top)
134     | MenorIgual -> ExpBool (pega_string vesq <=
135                             pega_string vdir, top)
135     | Maior -> ExpBool (pega_string vesq > pega_string
136                         vdir, top)
136     | MaiorIgual -> ExpBool (pega_string vesq >=
137                             pega_string vdir, top)
137     | Igual -> ExpBool (pega_string vesq =
138                         pega_string vdir, top)
```


9 INTERPRETADOR

```
138         | Diferente    -> ExpBool (pega_string vesq <>
139             pega_string vdir, top)
140     )
141     | TipoBooleano ->
142     (match op with
143     | Menor -> ExpBool (pega_bool vesq < pega_bool vdir
144         , top)
145     | MenorIgual -> ExpBool (pega_bool vesq <=
146         pega_bool vdir, top)
147     | Maior -> ExpBool (pega_bool vesq > pega_bool
148         vdir, top)
149     | MaiorIgual -> ExpBool (pega_bool vesq >=
150         pega_bool vdir, top)
151     | Igual -> ExpBool (pega_bool vesq == pega_bool
152         vdir, top)
153     | Diferente -> ExpBool (pega_bool vesq !=
154         pega_bool vdir, top)
155     | _ -> failwith "interpreta_relacional"
156     )
157     | _ -> failwith "interpreta_relacional"
158 )
159
160 and interpreta_logico () =
161     (match tesq with
162     | TipoBooleano ->
163     (match op with
164     | OuLogico -> ExpBool (pega_bool vesq || pega_bool
165         vdir, top)
166     | ELogico -> ExpBool (pega_bool vesq && pega_bool
167         vdir, top)
168     | _ -> failwith "interpreta_logico"
169     )
170     | _ -> failwith "interpreta_logico"
171     )
172
173 in
174 let valor = (match (classifica op) with
175     Aritmetico -> interpreta_aritmetico ()
176     | Relacional -> interpreta_relacional ()
177     | Logico -> interpreta_logico ()
178 )
179 in
180     valor
```

9 INTERPRETADOR

```
174 | ExpFunCall (id, args, tipo) ->
175   let open Amb in
176   ( match (Amb.busca amb id) with
177     | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
178       (* Interpreta cada um dos argumentos *)
179       let vargs = List.map (interpreta_exp amb) args in
180       (* Associa os argumentos aos parâmetros formais *)
181       let vformais = List.map2 (fun (n,t) v -> (n, t,
182         Some v)) formais vargs
183       in interpreta_fun amb id vformais locais corpo
184     | _ -> failwith "interpreta_exp: expchamada"
185   )
186 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo
187   =
188   let open A in
189   (* Estende o ambiente global, adicionando um ambiente local
190    *)
191   let ambfn = Amb.novo_escopo amb in
192   let insere_local d =
193     match d with
194     (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t
195     None
196   in
197   (* Associa os argumentos aos parâmetros e insere no novo
198    ambiente *)
199   let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v
200   in
201   let _ = List.iter insere_parametro fn_formais in
202   (* Insere as variáveis locais no novo ambiente *)
203   let _ = List.iter insere_local fn_locais in
204   (* Interpreta cada comando presente no corpo da função
205    usando o novo
206    ambiente *)
207   try
208     let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.
209     ExpVoid
210   with
211     Valor_de_retorno expret -> expret
212 and interpreta_cmd amb cmd =
213   let open A in
214   let open T in
215   match cmd with
216   Retorne exp ->
```

9 INTERPRETADOR

```
211 (* Levantar uma exceção foi necessária pois, pela semâ
    ntica do comando de
212     retorno, sempre que ele for encontrado em uma função,
        a computação
213     deve parar retornando o valor indicado, sem realizar
        os demais comandos.
214 *)
215 (match exp with
216 (* Se a função não retornar nada, então retorne ExpVoid
    *)
217     None -> raise (Valor_de_retorno ExpVoid)
218   | Some e ->
219     (* Avalia a expressão e retorne o resultado *)
220     let e1 = interpreta_exp amb e in
221     raise (Valor_de_retorno e1)
222 )
223
224 | Se (teste, entao, senao) ->
225     let teste1 = interpreta_exp amb teste in
226     (match teste1 with
227     ExpBool (true, _) ->
228         (* Interpreta cada comando do bloco 'então' *)
229         List.iter (interpreta_cmd amb) entao
230     | _ ->
231         (* Interpreta cada comando do bloco 'senão', se houver
            *)
232         (match senao with
233         None -> ()
234         | Some bloco -> List.iter (interpreta_cmd amb) bloco
235         )
236     )
237
238 | Attrib (elem, exp) ->
239     (* Interpreta o lado direito da atribuição *)
240     let exp = interpreta_exp amb exp
241     (* Faz o mesmo para o lado esquerdo *)
242     and (elem1, tipo) = obtem_nome_tipo_var elem in
243     Amb.atualiza_var amb elem1 tipo (Some exp)
244
245 | Chamada exp -> ignore( interpreta_exp amb exp)
246
247 | Leia exps ->
248     (* Obtem os nomes e os tipos de cada um dos argumentos *)
249     let nts = List.map (obtem_nome_tipo_var) exps in
250     let leia (nome, tipo) =
```

9 INTERPRETADOR

```
251     let valor =
252       (match tipo with
253         | A.TipoInteiro    -> T.ExpInt    (read_int (),
254           tipo)
255         | A.TipoCaractere -> T.ExpString (read_line (), tipo)
256         | A.TipoReal      -> T.ExpFloat (read_float(), tipo)
257         | _ -> failwith "leia: nao implementado"
258       )
259     in  Amb.atualiza_var amb nome tipo (Some valor)
260   in
261   (* Lê o valor para cada argumento e atualiza o ambiente *)
262   List.iter leia nts
263 | Escreva exps ->
264   (* Interpreta cada argumento da função 'saida' *)
265   let exps = List.map (interpreta_exp amb) exps in
266   let imprima exp =
267     (match exp with
268       | T.ExpInt (n,_) ->      let _ = print_int n in
269         print_string " "
270       | T.ExpFloat (n,_) ->    let _ = print_float n in
271         print_string " "
272       | T.ExpString (s,_) ->   let _ = print_string s in
273         print_string " "
274       | T.ExpBool (b,_) ->
275         let _ = print_string (if b then "true" else "false")
276         in print_string " "
277       | _ -> failwith "escreva: nao implementado"
278     )
279   in
280   let _ = List.iter imprima exps in
281   print_newline ()
282 |Escreval exps ->
283   (* Interpreta cada argumento da função 'saida' *)
284   let exps = List.map (interpreta_exp amb) exps in
285   let imprima exp =
286     (match exp with
287       | T.ExpInt (n,_) ->      let _ = print_int n in
288         print_string " "
289       | T.ExpString (s,_) ->   let _ = print_string s in
290         print_string " "
```

9 INTERPRETADOR

```
287 | T.ExpFloat (n, _) -> let _ = print_float n in
    print_string " "
288 | T.ExpBool (b, _) ->
289   let _ = print_string (if b then "true" else "false")
290   in print_string " "
291 | _ -> failwith "escreval: nao implementado"
292 )
293 in
294 let _ = List.iter imprima exps in
295 print_newline ()
296
297 | Enquanto (teste, corpo) ->
298   let rec laco teste corpo =
299     let condicao = interpreta_exp amb teste in
300     (match condicao with
301      | ExpBool (true, _) ->
302        (* Interpreta cada comando do bloco 'então' *)
303        let _ = List.iter (interpreta_cmd amb)
304                          corpo in
305        laco teste corpo
306      | _ -> ())
307   in laco teste corpo
308
309 | Para (var, inicio, fim, avanco, corpo) ->
310   let (elem1, tipo) = obtem_nome_tipo_var var in
311   let rec executa_para amb inicio fim avanco corpo elem1
312     tipo =
313     if (inicio) <= (fim)
314     then begin
315       List.iter (interpreta_cmd amb) corpo;
316
317       Amb.atualiza_var amb elem1 tipo (Some (ExpInt
318         ((inicio + avanco), TipoInteiro) ) );
319
320       executa_para amb (inicio + avanco) fim avanco
321         corpo elem1 tipo;
322     end in
323   executa_para amb (pega_int inicio) (pega_int fim) (
324     pega_int avanco) corpo elem1 tipo
325
326 (* Tentativa de switch com ambos tipos (char e int) *)
327 (*| Escolha (teste, casos, outrocaso) ->
328   (match teste with
```

9 INTERPRETADOR

```
325 | T.ExpInt (n,_) ->
326   let esc1 = interpreta_exp amb teste in
327   let escolha = pega_int esc1 in
328   let _ = List.map (fun s ->
329     match s with
330     (tipob, exp) ->
331       if escolha == (pega_int tipob)
332       then List.iter (interpreta_cmd amb) exp
333     ) casos in ()
334 | T.ExpString (s, _) ->
335   let esc2 = interpreta_exp amb teste in
336   let escolha = pega_string esc2 in
337   let _ = List.map (fun s ->
338     match s with
339     (tipob, exp) ->
340       if escolha == (pega_string tipob)
341       then List.iter (interpreta_cmd amb) exp
342     ) casos in ()
343
344   )*)
345
346 | Escolha (teste, casos, outrocaso) ->
347   let teste1 = interpreta_exp amb teste in
348   let escolha = pega_int teste1 in
349   let _ = List.map (fun s ->
350     match s with
351     (tipob, exp) ->
352       if escolha == (pega_int tipob)
353       then List.iter (interpreta_cmd amb) exp
354   ) casos in ()
355
356
357
358 let insere_declaracao_var amb dec =
359   match dec with
360   A.DecVar (nome, tipo) -> Amb.insere_local amb (fst
361     nome) tipo None
362
363 let insere_declaracao_fun amb dec =
364   let open A in
365   match dec with
366   FuncDecl {fn_id; fn_params; fn_tiporet; fn_locais;
367     fn_corpo} ->
368     let nome = fst fn_id in
```

9 INTERPRETADOR

```
367         let formais = List.map (fun (n,t) -> ((fst n), t))
           fn_params in
368     Amb.insere_fun amb nome formais fn_locais fn_tiporet
           fn_corpo
369
370
371 (* Lista de cabeçalhos das funções pré definidas *)
372 let fn_predefs = let open A in [
373     ("entrada", [("x", TipoInteiro); ("y", TipoInteiro)],
           TipoVoid, []);
374     ("saida",      [("x", TipoInteiro); ("y", TipoInteiro)],
           TipoVoid, []);
375 ]
376
377 (* insere as funções pré definidas no ambiente global *)
378 let declara_predefinidas amb =
379     List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps [] tr
           c) fn_predefs
380
381 let interprete ast =
382     (* cria ambiente global inicialmente vazio *)
383     let amb_global = Amb.novo_amb [] in
384     let _ = declara_predefinidas amb_global in
385     let (A.Prog (decs_globais, decs_funs, corpo)) = ast in
386     let _ = List.iter (insere_declaracao_var amb_global)
           decs_globais in
387     let _ = List.iter (insere_declaracao_fun amb_global)
           decs_funs in
388     (* Interpreta a função principal *)
389     let resultado = List.iter (interpreta_cmd amb_global) corpo
           in
390     resultado
```

9.2 Ambiente

Listing 64: ambInterp.ml

```
1 module Tab = Tabsimb
2 module A = Ast
3 module T = Tast
4
5 type entrada_fn = {
6     tipo_fn:   A.tipo;
7     formais: (A.identificador * A.tipo) list;
8     locais:   A.var_decl;
```

9 INTERPRETADOR

```
9  corpo: T.expressao A.statements
10 }
11
12 type entrada = EntFun of entrada_fn
13                | EntVar of A.tipo * (T.expressao
14                               option)
15
16 type t = {
17   ambv : entrada Tab.tabela
18 }
19
20 let novo_amb xs = { ambv = Tab.cria xs }
21
22 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
23
24 let busca amb ch = Tab.busca amb.ambv ch
25
26 let atualiza_var amb ch t v =
27   Tab.atualiza amb.ambv ch (EntVar (t,v))
28
29 let insere_local amb nome t v =
30   Tab.insere amb.ambv nome (EntVar (t,v))
31
32 let insere_param amb nome t v =
33   Tab.insere amb.ambv nome (EntVar (t,v))
34
35 let insere_fun amb nome params locais resultado corpo =
36   let ef = EntFun { tipo_fn = resultado;
37                     formais = params;
38                     locais = locais;
39                     corpo = corpo }
40   in Tab.insere amb.ambv nome ef
```


10 Bibliografia

1. *Processo de Compilação .NET*
2. *Entendendo a Common Intermediate Language (CIL)*
3. *Implementação de um compilador utilizando .NET*
4. *Básico do projeto Mono*
5. *Lista de instruções CIL*
6. *Trabalho de Construção de Compiladores - MiniC para CIL (Gabriel Henrique Montezelo, João Pedro Ferreira)*
7. *Trabalho de Construção de Compiladores - MiniJava para CIL (Igor Benaventana Alves, Márcio Scofoni Manfré)*