

# Construção de um Compilador para MiniLua usando Objective Caml

Henrique Araújo Lima

[henriquebrmg@gmail.com](mailto:henriquebrmg@gmail.com)

Vinicius Lopes da Silva Teixeira

[viniciuslt@outlook.com](mailto:viniciuslt@outlook.com)

Faculdade de Computação  
Universidade Federal de Uberlândia

14 de dezembro de 2016

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	OCaml . . . . .	3
1.1.1	Instalação . . . . .	3
1.2	CRL . . . . .	3
1.3	CIL . . . . .	3
1.3.1	Instruções . . . . .	4
1.3.2	Compilação . . . . .	4
1.3.3	Execução . . . . .	4
1.4	Mono . . . . .	4
1.4.1	Instalação . . . . .	4
<b>2</b>	<b>Programas</b>	<b>5</b>
2.1	Módulo Mínimo . . . . .	5
2.2	Declaração de uma variável . . . . .	5
2.3	Atribuição de um inteiro a uma variável . . . . .	6
2.4	Atribuição de uma soma de inteiros a uma variável . . . . .	6
2.5	Inclusão do comando de impressão . . . . .	7
2.6	Atribuição de uma subtração de inteiros a uma variável . . . . .	7
2.7	Inclusão do comando condicional . . . . .	8
2.8	Inclusão do comando condicional com parte senão . . . . .	9
2.9	Atribuição de duas operações aritméticas sobre inteiro a uma variável . . . . .	9
2.10	Atribuição de duas variáveis inteiras . . . . .	10
2.11	Atribuição de um comando de repetição enquanto . . . . .	11
2.12	Comando condicional aninhado em um comando de repetição . . . . .	12
2.13	Converte graus Celsius para Fahrenheit . . . . .	13
2.14	Decide qual maior . . . . .	16
2.15	Le o número e verifica se está entre 100 e 200 . . . . .	17
2.16	Le números e informa quais estão entre 10 e 150 . . . . .	18
2.17	Le strings e caracteres . . . . .	19
2.18	Escreve um número lido por extenso . . . . .	21
2.19	Decide se os números são positivos, negativos ou zero . . . . .	23
2.20	Decide se um número é maior que 10 . . . . .	24
2.21	Cálculo de preços . . . . .	26
2.22	Calcula o fatorial de um número . . . . .	27
2.23	Decide se um numero é positivo, zero ou negativo com auxilio de uma função . . . . .	29
<b>3</b>	<b>Construindo o Compilador</b>	<b>31</b>
3.1	Analisador Léxico . . . . .	31
3.1.1	Analisador Léxico Manual para Pascal . . . . .	31

3.1.2	Analizador Léxico Automático para Lua . . . . .	36
3.2	Analizador sintático . . . . .	41
3.2.1	Parser preditivo . . . . .	42
3.2.2	Analizador sintático com mensagens de erros . . . . .	46
3.3	Analizador Semântico . . . . .	58
3.3.1	Criando os arquivos semânticos . . . . .	65
3.3.2	Compilando . . . . .	73
3.3.3	Testando . . . . .	74
3.4	Interpretador . . . . .	76
3.4.1	Compilando . . . . .	84
3.4.2	Testando . . . . .	89

# Capítulo 1

## Introdução

Este trabalho tem como objetivo construir um compilador da linguagem de programação Lua. Para esta construção utilizaremos a linguagem funcional OCaml.

### 1.1 OCaml

Objective Caml, também conhecida como OCaml (Objective Categorical Abstract Machine Language), é uma linguagem de programação funcional da família ML, desenvolvida pelo INRIA em 1996. Trata-se da linguagem Caml com a adição de suporte de técnicas de orientação a objetos e algumas alterações e extensões de sintaxe.

#### 1.1.1 Instalação

O comando para instalar o Ocaml no Ubuntu 16.04 é:

```
> sudo apt-get install ocaml
```

### 1.2 CRL

A Common Language Runtime (CLR), um componente da máquina virtual do framework Microsoft .NET que gerencia a execução de programas .NET. Um processo de compilação conhecido por just-in-time que converte o código compilado em instruções de máquina ao qual a CPU do computador executa.

### 1.3 CIL

O Common Intermediate Language (CLI) descreve o código executável e o ambiente de execução que formam o núcleo do Framework .NET da Microsoft, do MONO e do Porta-

ble.NET, sendo as duas últimas implementações gratuitas e de código aberto da mesma. A Common Language Runtime é a implementação da CLI pela Microsoft, em outras palavras, é uma máquina virtual que segue um padrão internacional e a base para a criação e execução de ambientes de desenvolvimento em que as linguagens e as bibliotecas trabalham juntos.

### 1.3.1 Instruções

A lista do conjunto de instruções do CLI podem ser encontradas em: [https://en.wikipedia.org/wiki/List\\_of\\_CIL\\_instructions](https://en.wikipedia.org/wiki/List_of_CIL_instructions)

### 1.3.2 Compilação

O comando para compilar um arquivo é :

```
> ilasm "nome_do_arquivo".il
```

É necessário que o arquivo tenha extensão ".il"

### 1.3.3 Execução

O comando para executar um arquivo é :

```
> mono "nome_do_arquivo".exe
```

É necessário que o arquivo tenha extensão ".exe"

## 1.4 Mono

O Mono é um projeto liderado pela Novell (anteriormente pela Ximian) para criar um conjunto de ferramentas compatíveis com a plataforma .NET, conforme as normas ECMA, incluindo, entre outras, um compilador C Sharp e um CLR. O principal objetivo do Mono não é somente ser capaz de rodar aplicações .NET multi-plataformas, mas também proporcionar melhores ferramentas de desenvolvimento para desenvolvedores Linux.

### 1.4.1 Instalação

O comando para instalar o Mono no Ubuntu 16.04 é:

```
> sudo apt-get install mono-complete
```

# Capítulo 2

## Programas

Todos os programas com extensão ".il" ou seja os assemblys foram gerados "na mão", com exceção do programa descrito na seção 2.13 onde foi utilizado um programa escrito em C# para gerar o correspondente em assembly.

### 2.1 Módulo Mínimo

MiniLua

Programa 2.1: nano01.lua

CIL

Programa 2.2: nano01.il

```
1 .assembly nano01{}
2 .method static void main() cil managed
3 {
4     .entrypoint
5     ret
6 }
```

### 2.2 Declaração de uma variável

MiniLua

Programa 2.3: nano02.il

```
1 local n
```

CIL

Programa 2.4: nano02.il

```

1 .assembly nano02{}
2 .method static void main() cil managed
3 {
4     .locals init (int32 n)
5     .maxstack 1
6     .entrypoint
7     ret
8 }

```

## 2.3 Atribuição de um inteiro a uma variável

MiniLua

Programa 2.5: nano03.lua

```

1 local n
2 n = 1

```

CIL

Programa 2.6: nano03.il

```

1 .assembly nano03{}
2 .method static void main() cil managed
3 {
4     .locals init (int32 n)
5     .maxstack 1
6     .entrypoint
7     ldc.i4.1
8     stloc n
9     ret
10 }

```

## 2.4 Atribuição de uma soma de inteiros a uma variável

MiniLua

Programa 2.7: nano04.lua

```

1 local n
2 n = 1 + 2

```

CIL

Programa 2.8: nano04.il

```

1 .assembly nano04{}
2 .method static void main() cil managed
3 {
4     .locals init (int32 n)
5     .maxstack 2

```

```

6  .entrypoint
7  ldc.i4.1
8  ldc.i4.2
9  add
10 stloc n
11 ret
12 }

```

## 2.5 Inclusão do comando de impressão

MiniLua

Programa 2.9: nano05.lua

```

1 local n
2 n = 2
3 io.write(n)

```

CIL

Programa 2.10: nano05.il

```

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

```

## 2.6 Atribuição de uma subtração de inteiros a uma variável

MiniLua

Programa 2.11: nano06.lua

```

1 local n
2 n = 1 - 2
3 io.write(n)

```

CIL



### Programa 2.12: nano06.il

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

## 2.7 Inclusão do comando condicional

### MiniLua

#### Programa 2.13: nano07.lua

```
1 local n
2 n = 1
3 if n == 1 then
4     io.write(n)
5 end
```

### CIL

#### Programa 2.14: nano07.il

```
1
2 .assembly extern mscorlib {}
3 .assembly nano07{}
4 .method static void main() cil managed
5 {
6     .locals init (int32 n)
7     .maxstack 2
8     .entrypoint
9     ldc.i4.1
10    stloc n
11    ldloc n
12    ldc.i4.1
13    beq IGUAL
14
15    IGUAL:
16        ldloc n
17        call void [mscorlib]System.Console::WriteLine (int32)
18        ret
19 }
```

## 2.8 Inclusão do comando condicional com parte senão

MiniLua

Programa 2.15: nano08.lua

```
1 local n
2 n = 1
3 if n == 1 then
4   io.write(n)
5 else
6   io.write(0)
7 end
```

CIL

Programa 2.16: nano08.il

```
1
2 .assembly extern mscorlib {}
3 .assembly nano08{}
4 .method static void main() cil managed
5 {
6   .locals init (int32 n)
7   .maxstack 2
8   .entrypoint
9   ldc.i4.1
10  stloc n
11  ldloc n
12  ldc.i4.1
13  beq IGUAL
14  ldc.i4.0
15  call void [mscorlib]System.Console::WriteLine (int32)
16  br FIM
17
18  IGUAL:
19    ldloc n
20    call void [mscorlib]System.Console::WriteLine (int32)
21
22  FIM:
23    ret
24 }
```

## 2.9 Atribuição de duas operações aritméticas sobre inteiro a uma variável

MiniLua

Programa 2.17: nano09.lua

```
1 local n
2 n = 1 + 1 / 2
3
4 if n == 1 then
```

```

5  io.write(n)
6  else
7  io.write(0)
8  end

```

CIL

#### Programa 2.18: nano09.il

```

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

```

## 2.10 Atribuição de duas variáveis inteiras

MiniLua

#### Programa 2.19: nano10.lua

```

1 local n, m
2 n = 1
3 m = 2
4
5 if n == m then
6     io.write(n)
7 else
8     io.write(0)
9 end

```

CIL

#### Programa 2.20: nano10.il

```

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

```

## 2.11 Atribuição de um comando de repetição enquanto

MiniLua

Programa 2.21: nano11.lua

```

1 local n, m, x
2 n = 1
3 m = 2
4 x = 5
5
6 while x > n do
7     n = n + m
8     io.write(n, "\n")
9 end

```

CIL

Programa 2.22: nano11.il

```

1 .assembly extern mscorlib {}
2 .assembly nano11{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 3
7     .locals init (int32 n,int32 m,int32 x)
8     ldc.i4.1
9     stloc n
10    ldc.i4.2

```

```

11  stloc m
12  ldc.i4.5
13  stloc x
14
15  LOOP:
16      ldloc x
17      ldloc n
18      ble FIM
19      ldloc n
20      ldloc m
21      add
22      stloc n
23      ldloc n
24      call void [mscorlib]System.Console::WriteLine (int32)
25      br LOOP
26
27  FIM:
28      ret
29  }

```

## 2.12 Comando condicional aninhado em um comando de repetição

MiniLua

Programa 2.23: nano12.lua

```

1 local n, m, x
2 n = 1
3 m = 2
4 x = 5
5
6 while x > n do
7     if n == m then
8         io.write(n)
9     else
10        io.write(0)
11    end
12    x = x - 1
13 end

```

CIL

Programa 2.24: nano12.il

```

1 .assembly extern mscorlib {}
2 .assembly nano12{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 3
7     .locals init (int32 n,int32 m,int32 x)
8     ldc.i4.1
9     stloc n
10    ldc.i4.2

```

```

11  stloc m
12  ldc.i4.5
13  stloc x
14
15  LOOP:
16      ldloc x
17      ldloc n
18      ble FIM
19      ldloc n
20      ldloc m
21      beq COND_SE
22      ldc.i4.0
23      call void [mscorlib]System.Console::WriteLine (int32)
24      br DECREMENTA
25
26  COND_SE:
27      ldloc n
28      call void [mscorlib]System.Console::WriteLine (int32)
29      br DECREMENTA
30
31  DECREMENTA:
32      ldloc x
33      ldc.i4.1
34      sub
35      stloc x
36      br LOOP
37
38  FIM:
39      ret
40 }

```

## 2.13 Converte graus Celsius para Fahrenheit

MiniLua

Programa 2.25: micro01.lua

```

1 local cel, far
2
3 io.write("Tabela de conversão: Celsius -> Fahrenheit\n")
4 io.write("Digite a temperatura em Celsius: ")
5
6 cel = io.read()
7 far = (9*cel+160)/5
8
9 io.write("A nova temperatura é: ",far," F\n")

```

C#

Programa 2.26: micro01.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;

```

```

5
6 namespace CelParaFar
7 {
8     class Programa
9     {
10         static void Main(string[] args)
11         {
12             int cel, far;
13             Console.WriteLine("Tabela de conversão: Celsius -> Fahrenheit"
14                               );
15             Console.WriteLine("Digite a temperatura em Celsius: ");
16             cel = int.Parse(Console.ReadLine());
17             far = (cel * 9) / 5 + 32;
18             Console.WriteLine("A nova temperatura é: " + far + " F");
19         }
20     }
21 }

```

Para gerar o arquivo "micro01.il" a partir do programa "micro01.cs" em C#, foram executados os seguintes comandos:

```

> mcs micro01.cs
> monodis micro01.exe > micro01.il

```

## CIL

### Programa 2.27: micro01.il

```

1 .assembly extern mscorlib
2 {
3     .ver 4:0:0:0
4     .publickeytoken = (B7 7A 5C 56 19 34 E0 89 ) // .z\V.4..
5 }
6 .assembly 'cel'
7 {
8     .custom instance void class [mscorlib]System.Runtime.CompilerServices.
9         RuntimeCompatibilityAttribute::.ctor'() = (
10         01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....T..WrapNonEx
11         63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01      ) // ceptionThrows.
12     .hash algorithm 0x00008004
13     .ver 0:0:0:0
14 }
15 .module cel.exe // GUID = {2CB4B67B-10AD-4DBB-9508-123065B8CB4A}
16
17
18 .namespace CelParaFar
19 {
20     .class private auto ansi beforefieldinit Programa
21     extends [mscorlib]System.Object
22     {
23
24         // method line 1
25         .method public hidebysig specialname rtspecialname
26             instance default void '.ctor' () cil managed
27         {
28             // Method begins at RVA 0x2050
29             // Code size 7 (0x7)

```

```

30 .maxstack 8
31 IL_0000: ldarg.0
32 IL_0001: call instance void object::'.ctor'()
33 IL_0006: ret
34 } // end of method Programa::.ctor
35
36 // method line 2
37 .method private static hidebysig
38     default void Main (string[] args) cil managed
39 {
40     // Method begins at RVA 0x2058
41     .entrypoint
42     // Code size 68 (0x44)
43     .maxstack 3
44     .locals init (
45         int32 V_0,
46         int32 V_1)
47 IL_0000: ldstr bytearray (
48     54 00 61 00 62 00 65 00 6c 00 61 00 20 00 64 00 // T.a.b.e.l.a. .d.
49     65 00 20 00 63 00 6f 00 6e 00 76 00 65 00 72 00 // e. .c.o.n.v.e.r.
50     73 00 e3 00 6f 00 3a 00 20 00 43 00 65 00 6c 00 // s...o... .C.e.l.
51     73 00 69 00 75 00 73 00 20 00 2d 00 3e 00 20 00 // s.i.u.s. .-.>. .
52     46 00 61 00 68 00 72 00 65 00 6e 00 68 00 65 00 // F.a.h.r.e.n.h.e.
53     69 00 74 00 01 ) // i.t..
54
55 IL_0005: call void class [mscorlib]System.Console::WriteLine(string)
56 IL_000a: ldstr "Digite a temperatura em Celsius: "
57 IL_000f: call void class [mscorlib]System.Console::WriteLine(string)
58 IL_0014: call string class [mscorlib]System.Console::ReadLine()
59 IL_0019: call int32 int32::Parse(string)
60 IL_001e: stloc.0
61 IL_001f: ldloc.0
62 IL_0020: ldc.i4.s 0x09
63 IL_0022: mul
64 IL_0023: ldc.i4.5
65 IL_0024: div
66 IL_0025: ldc.i4.s 0x20
67 IL_0027: add
68 IL_0028: stloc.1
69 IL_0029: ldstr bytearray (
70     41 00 20 00 6e 00 6f 00 76 00 61 00 20 00 74 00 // A. .n.o.v.a. .t.
71     65 00 6d 00 70 00 65 00 72 00 61 00 74 00 75 00 // e.m.p.e.r.a.t.u.
72     72 00 61 00 20 00 e9 00 3a 00 20 00 01 ) // r.a. .... .
73
74 IL_002e: ldloc.1
75 IL_002f: box [mscorlib]System.Int32
76 IL_0034: ldstr " F"
77 IL_0039: call string string::Concat(object, object, object)
78 IL_003e: call void class [mscorlib]System.Console::WriteLine(string)
79 IL_0043: ret
80 } // end of method Programa::Main
81
82 } // end of class CelParaFar.Programa
83 }

```

Para compilar e executar o arquivo "micro01.il" foram feitos os seguintes comandos:

```

> ilasm micro01.il
> mono micro01.exe

```



## 2.14 Decide qual maior

MiniLua

Programa 2.28: micro02.lua

```
1 local num1, num2
2
3 io.write("Digite o primeiro numero: ")
4 num1 = io.read()
5 io.write("Digite o segundo numero: ")
6 num2 = io.read()
7
8 if num1 > num2 then
9     io.write("O primeiro número ", num1, " é maior que o segundo ", num2)
10 else
11     io.write("O segundo número ", num2, " é maior que o primeiro ", num1)
12 end
```

CIL

Programa 2.29: micro02.il

```
1 .assembly extern mscorlib {}
2 .assembly micro02{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 2
7     .locals init (int32 num1,int32 num2)
8     ldstr "Escreva o primeiro numero: "
9     call void [mscorlib]System.Console::WriteLine (string)
10    call string [mscorlib]System.Console::ReadLine ()
11    call int32 [mscorlib]System.Int32::Parse(string)
12    stloc num1
13    ldstr "Escreva o segundo numero: "
14    call void [mscorlib]System.Console::WriteLine (string)
15    call string [mscorlib]System.Console::ReadLine ()
16    call int32 [mscorlib]System.Int32::Parse(string)
17    stloc num2
18    ldloc num1
19    ldloc num2
20    bgt PRIMEIRO
21    ldstr "Segundo numero eh maior que o primeiro!"
22    call void [mscorlib]System.Console::WriteLine (string)
23    ldstr "Primeiro numero: "
24    call void [mscorlib]System.Console::WriteLine (string)
25    ldloc num1
26    call void [mscorlib]System.Console::WriteLine (int32)
27    ldstr "Segundo numero: "
28    call void [mscorlib]System.Console::WriteLine (string)
29    ldloc num2
30    call void [mscorlib]System.Console::WriteLine (int32)
31    br FIM
32
33    PRIMEIRO:
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 num1
39     call void [mscorlib]System.Console::WriteLine (int32)
40     ldstr "Segundo numero: "
41     call void [mscorlib]System.Console::WriteLine (string)
42     ldloc num2
43     call void [mscorlib]System.Console::WriteLine (int32)
44
45     FIM:
46     ret
47 }

```

## 2.15 Le o número e verifica se está entre 100 e 200

MiniLua

Programa 2.30: micro03.lua

```

1 local numero
2
3 io.write("Digite um numero: ")
4 numero = tonumber(io.read())
5
6
7 if numero >= 100 then
8
9     if numero <= 200 then
10         io.write("O número está no intervalo entre 100 e 200\n")
11     else
12         io.write("O número não está no intervalo entre 100 e 200\n")
13     end
14
15 else
16     io.write("O número não está no intervalo entre 100 e 200\n")
17 end

```

CIL

Programa 2.31: micro03.il

```

1
2 .assembly extern mscorlib{}
3 .assembly micro03{}
4 .method static void main() cil managed
5 {
6     .entrypoint
7     .maxstack 3
8     .locals init (int32 numero)
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 numero
14    ldloc numero
15    ldc.i4 100

```

```

16  bge MAIORIGUALCEM
17  br FORAINTERVALO
18
19  MAIORIGUALCEM:
20      ldloc numero
21      ldc.i4 200
22      ble DENTROINTERVALO
23      br FORAINTERVALO
24
25  DENTROINTERVALO:
26      ldstr "O numero esta no intervalo entre 100 e 200"
27      call void [mscorlib]System.Console::WriteLine (string)
28      br FIM
29
30  FORAINTERVALO:
31      ldstr "O numero nao esta no intervalo entre 100 e 200"
32      call void [mscorlib]System.Console::WriteLine (string)
33      br FIM
34
35  FIM:
36      ret
37 }

```

## 2.16 Le números e informa quais estão entre 10 e 150

MiniLua

Programa 2.32: micro04.lua

```

1 local x, numero, intervalo
2 intervalo = 0
3
4 for x = 1, 5, 1 do
5     io.write("Digite um numero: ")
6     numero = tonumber(io.read())
7
8     if numero >= 10 then
9         if numero <= 150 then
10             intervalo = intervalo + 1
11         end
12     end
13
14 end
15
16 io.write("Ao total, foram digitados ",intervalo," números no intervalo
    entre 10 e 150\n")

```

CIL

Programa 2.33: micro04.il

```

1 .assembly extern mscorlib{}
2 .assembly micro04{}
3 .method static void main() cil managed
4 {

```

```

5  .entrypoint
6  .maxstack 5
7  .locals init (int32 x,int32 num,int32 intervalo)
8  ldc.i4 x
9  stloc num
10 ldc.i4.1
11 stloc x
12
13 COMECALOOP:
14     ldloc x
15     ldc.i4.5
16     bgt ESCREVEQTD
17     ldstr "Escreva 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 intervalo
22     ldloc x
23     ldc.i4.1
24     add
25     stloc x
26     ldloc intervalo
27     ldc.i4 10
28     bge MAIORQUE10
29     br COMECALOOP
30
31 MAIORQUE10:
32     ldloc intervalo
33     ldc.i4 150
34     ble MENORQUE150
35     br COMECALOOP
36
37 MENORQUE150:
38     ldloc num
39     ldc.i4.1
40     add
41     stloc num
42     br COMECALOOP
43
44 ESCREVEQTD:
45     ldstr "Quantidade de numeros digitados no intervalo entre 10 e 150"
46     call void [mscorlib]System.Console::WriteLine (string)
47     ldloc num
48     call void [mscorlib]System.Console::WriteLine (int32)
49     br FIM
50
51 FIM:
52     ret
53 }

```

## 2.17 Le strings e caracteres

MiniLua

Programa 2.34: micro05.lua

```

1 local nome, sexo, x, h, m
2 h = 0
3 m = 0
4
5 for x = 1, 5, 1 do
6     io.write("Digite o nome: ")
7     nome = io.read()
8     io.write("H - Homem ou M - Mulher: ")
9     sexo = io.read()
10
11     if sexo == 'H' then
12         h = h + 1
13     elseif sexo == 'M' then
14         m = m + 1
15     else
16         io.write("Sexo só pode ser H ou M!\n")
17     end
18
19 end
20
21 io.write("Foram inseridos ",h," Homens\n")
22 io.write("Foram inseridos ",m," Mulheres\n")

```

## CIL

### Programa 2.35: micro05.il

```

1 .assembly extern mscorlib{}
2 .assembly micro05{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 10
7     .locals init (int32 x,int32 h,int32 m,char sexo,string nome)
8     ldc.i4.1
9     stloc x
10    ldc.i4.0
11    stloc h
12    ldc.i4.0
13    stloc m
14    LOOP:
15        ldloc x
16        ldc.i4.5
17        bgt IMPRIME
18        ldstr "Digite o nome: "
19        call void [mscorlib]System.Console::WriteLine (string)
20        call string [mscorlib]System.Console::ReadLine ()
21        stloc nome
22        ldstr "H - Homem ou M - Mulher: "
23        call void [mscorlib]System.Console::WriteLine (string)
24        call string [mscorlib]System.Console::ReadLine ()
25        call char [mscorlib]System.Char::Parse(string)
26        stloc sexo
27        ldloc sexo
28        ldc.i4 72
29        bne.un VERIFMULHER
30        ldloc h
31        ldc.i4.1
32        add

```

```

33     stloc h
34     br INCRX
35 INCRX:
36     ldloc x
37     ldc.i4.1
38     add
39     stloc x
40     br LOOP
41 VERIFMULHER:
42     ldloc sexo
43     ldc.i4 77
44     bne.un OUTROCASO
45     ldloc m
46     ldc.i4.1
47     add
48     stloc m
49     br INCRX
50 OUTROCASO:
51     ldstr "Sexo so pode ser H ou M!"
52     call void [mscorlib]System.Console::WriteLine (string)
53     br LOOP
54 IMPRIME:
55     ldstr "Total homens inseridos: "
56     call void [mscorlib]System.Console::WriteLine (string)
57     ldloc h
58     call void [mscorlib]System.Console::WriteLine (int32)
59     ldstr "Total mulheres inseridas: "
60     call void [mscorlib]System.Console::WriteLine (string)
61     ldloc m
62     call void [mscorlib]System.Console::WriteLine (int32)
63 FIM:
64     ret
65 }

```

## 2.18 Escreve um número lido por extenso

MiniLua

Programa 2.36: micro06.lua

```

1 local numero
2
3 io.write("Digite um numero de 1 a 5: ")
4 numero = tonumber(io.read())
5
6
7 if numero == 1 then
8     io.write("Um\n")
9 elseif numero == 2 then
10    io.write("Dois\n")
11 elseif numero == 3 then
12    io.write("Tres\n")
13 elseif numero == 4 then
14    io.write("Quatro\n")
15 elseif numero == 5 then
16    io.write("Cinco\n")

```

```

17 else
18   io.write("Número Inválido!!!\n")
19 end

```

## CIL

### Programa 2.37: micro06.il

```

1 .assembly extern mscorlib{}
2 .assembly micro06{}
3 .method static void main() cil managed
4 {
5   .entrypoint
6   .maxstack 3
7   .locals init (int32 numero)
8   ldstr "Digite um numero de 1 a 5: "
9   call void [mscorlib]System.Console::WriteLine (string)
10  call string [mscorlib]System.Console::ReadLine ()
11  call int32 [mscorlib]System.Int32::Parse(string)
12  stloc numero
13  ldloc numero
14  ldc.i4.1
15  bne.un DOIS
16  ldstr "Um."
17  call void [mscorlib]System.Console::WriteLine (string)
18  br FIM
19
20  DOIS:
21    ldloc numero
22    ldc.i4.2
23    bne.un TRES
24    ldstr "Dois."
25    call void [mscorlib]System.Console::WriteLine (string)
26    br FIM
27
28  TRES:
29    ldloc numero
30    ldc.i4.3
31    bne.un QUATRO
32    ldstr "Tres."
33    call void [mscorlib]System.Console::WriteLine (string)
34    br FIM
35
36  QUATRO:
37    ldloc numero
38    ldc.i4 4
39    bne.un CINCO
40    ldstr "Quatro."
41    call void [mscorlib]System.Console::WriteLine (string)
42    br FIM
43
44  CINCO:
45    ldloc numero
46    ldc.i4 5
47    bne.un OUTROCASO
48    ldstr "Cinco."
49    call void [mscorlib]System.Console::WriteLine (string)
50    br FIM
51

```

```

52 OUTROCASO:
53     ldstr "Numero invalido!."
54     call void [mscorlib]System.Console::WriteLine (string)
55     br FIM
56 FIM:
57     ret
58 }

```

## 2.19 Decide se os números são positivos, negativos ou zero

MiniLua

Programa 2.38: micro07.lua

```

1 local programa, numero, opc
2 programa = 1
3
4 while programa == 1 do
5     io.write("Digite um numero: ")
6     numero = tonumber(io.read())
7
8     if numero > 0 then
9         io.write("Positivo\n")
10
11     else
12         if numero == 0 then
13             io.write("O numero é igual a 0\n")
14         end
15         if numero < 0 then
16             io.write("Negativo\n")
17         end
18     end
19
20     io.write("Deseja finalizar? (S/N): ")
21     opc = io.read()
22
23     if opc == 'S' then
24         programa = 0
25     end
26 end

```

CIL

Programa 2.39: micro07.il

```

1 .assembly extern mscorlib{}
2 .assembly micro07{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 2
7     .locals init (int32 programa,int32 numero,char opc)
8     ldc.i4.1
9     stloc programa

```



```

10
11 LOOP:
12     ldloc programa
13     ldc.i4.1
14     bne.un 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.i4.0
22     bgt MAIOR0
23     ldloc numero
24     ldc.i4.0
25     beq IGUAL0
26     ldloc numero
27     ldc.i4.0
28     blt MENOR0
29
30 MAIOR0:
31     ldstr "Positivo."
32     call void [mscorlib]System.Console::WriteLine (string)
33     br VERIF
34
35 IGUAL0:
36     ldstr "O numero e igual a 0."
37     call void [mscorlib]System.Console::WriteLine (string)
38     br VERIF
39
40 MENOR0:
41     ldstr "Negativo."
42     call void [mscorlib]System.Console::WriteLine (string)
43     br VERIF
44
45 VERIF:
46     ldstr "Deseja finalizar? (S/N) "
47     call void [mscorlib]System.Console::WriteLine (string)
48     call string [mscorlib]System.Console::ReadLine ()
49     call char [mscorlib]System.Char::Parse(string)
50     stloc opc
51     ldc.i4 83
52     ldloc opc
53     beq FIM
54     br LOOP
55
56 FIM:
57     ret
58 }

```

## 2.20 Decide se um número é maior que 10

MiniLua

Programa 2.40: micro08.lua

```

1 local numero
2 numero = 1
3
4 while numero ~= 0 do
5   io.write("Digite um numero: ")
6   numero = tonumber(io.read())
7
8   if numero > 10 then
9     io.write("O número ",numero," é maior que 10\n")
10
11  else
12    io.write("O número ",numero," é menor que 10\n")
13  end
14 end

```

## CIL

### Programa 2.41: micro08.il

```

1 .assembly extern mscorlib{}
2 .assembly micro08{}
3 .method static void main() cil managed
4 {
5   .entrypoint
6   .maxstack 2
7   .locals init (int32 numero)
8   ldc.i4.1
9   stloc numero
10
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 MAIOR10
23    ldstr "O numero eh menor do que 10."
24    call void [mscorlib]System.Console::WriteLine (string)
25    br LOOP
26
27  MAIOR10:
28    ldstr "O numero eh maior do que 10."
29    call void [mscorlib]System.Console::WriteLine (string)
30    br LOOP
31
32  FIM:
33    ret
34 }

```

## 2.21 Cálculo de preços

MiniLua

Programa 2.42: micro09.lua

```
1 local preco, venda, novo_preco
2
3 io.write("Digite o preço: ")
4 preco = tonumber(io.read())
5 io.write("Digite a venda: ")
6 venda = tonumber(io.read())
7
8 if venda < 500 or preco < 30 then
9     novo_preco = preco + 10/100 * preco
10
11 elseif (venda >= 500 and venda < 1200) or (preco >= 30 and preco < 80)
12     then
13     novo_preco = preco + 15/100 * preco
14
15 elseif venda >= 1200 or preco >= 80 then
16     novo_preco = preco - 20/100 * preco
17 end
18 io.write("O novo preço é ", novo_preco, "\n")
```

CIL

Programa 2.43: micro09.il

```
1 .assembly extern mscorlib{}
2 .assembly micro09{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 3
7     .locals init (float32 preco, float32 venda, float32 novo_preco)
8     ldstr "Digite a preço: "
9     call void [mscorlib]System.Console::WriteLine (string)
10    call string [mscorlib]System.Console::ReadLine ()
11    call float32 [mscorlib]System.Single::Parse(string)
12    stloc preco
13    ldstr "Digite a venda: "
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 venda
18    ldloc venda
19    ldc.r4 500.0
20    blt IF11
21    ldloc preco
22    ldc.r4 30.0
23    blt IF11
24    ldloc venda
25    ldc.r4 500.0
26    bge IF1200
27    IF11:
28        ldc.r4 1.1
29    ldloc preco
```

```

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

```

## 2.22 Calcula o fatorial de um número

MiniLua

Programa 2.44: micro10.lua

```

1 function fatorial(num)
2   if num <= 0 then
3     return 1
4   else
5     return num * fatorial(num-1)
6   end
7 end
8
9 local numero, fat
10
11 io.write("Digite um número: ")

```

```

12 numero = tonumber(io.read())
13
14 fat = fatorial(numero)
15
16 io.write("O fatorial de ")
17 io.write(numero)
18 io.write(" é ")
19 io.write(fat)

```

## CIL

### Programa 2.45: micro10.il

```

1 .assembly extern mscorlib{}
2 .assembly micro10{}
3 .method static void main() cil managed
4 {
5     .entrypoint
6     .maxstack 10
7     .locals init (int32 numero, int32 fat)
8
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
17    ldloc fat
18    call void [mscorlib]System.Console::WriteLine (int32)
19    ret
20 }
21
22 .method public static int32 fatorial(int32 n) cil managed
23 {
24     .maxstack 10
25     ldarg n
26     ldc.i4.0
27     ble RET1
28     ldarg n
29     ldc.i4.1
30     sub
31     call int32 fatorial(int32)
32     ldarg n
33     mul
34     br FIM
35
36 RET1:
37     ldc.i4.1
38     br FIM
39
40 FIM:
41     ret
42 }

```

## 2.23 Decide se um numero é positivo, zero ou negativo com auxilio de uma função

MiniLua

Programa 2.46: micro11.lua

```
1 function verifica(n)
2   local res
3
4   if n > 0 then
5     res = 1
6   elseif n < 0 then
7     res = -1
8   else
9     res = 0
10  end
11
12  return res
13 end
14
15 local numero, x
16
17 io.write("Digite um número: ")
18 numero = tonumber(io.read())
19
20 x = verifica(numero)
21
22 if x == 1 then
23   io.write("Numero positivo\n")
24
25 elseif x == 0 then
26   io.write("Zero\n")
27
28 else
29   io.write("Numero Negativo\n")
30
31 end
```

CIL

Programa 2.47: micro11.il

```
1 .assembly extern mscorlib{}
2 .assembly micro11{}
3 .method static void main() cil managed{
4   .entrypoint
5   .maxstack 3
6   .locals init (int32 numero,int32 x)
7   ldstr "Digite um número: "
8   call void [mscorlib]System.Console::WriteLine (string)
9   call string [mscorlib]System.Console::ReadLine ()
10  call int32 [mscorlib]System.Int32::Parse(string)
11  stloc numero
12  ldloc numero
13  call int32 Verifica(int32)
14  stloc x
15  ldloc x
```

```

16 ldc.i4 1
17 beq POS
18 ldloc x
19 ldc.i4 0
20 beq ZERO
21 br NEG
22 POS:
23     ldstr "Numero Positivo"
24     call void [mscorlib]System.Console::WriteLine (string)
25     br FIM
26 ZERO:
27     ldstr "Zero"
28     call void [mscorlib]System.Console::WriteLine (string)
29     br FIM
30 NEG:
31     ldstr "Numero Negativo"
32     call void [mscorlib]System.Console::WriteLine (string)
33     br FIM
34 FIM:
35     ret
36 }
37
38 .method public static int32 Verifica(int32 n) cil managed{
39     .maxstack 4
40     .locals init (int32 res)
41     ldc.i4 0
42     stloc res
43     ldarg n
44     ldc.i4.0
45     bgt INCR
46     ldarg n
47     ldc.i4.0
48     blt DECR
49     ldc.i4.0
50     stloc res
51     br FIM
52 INCR:
53     ldloc res
54     ldc.i4 1
55     add
56     stloc res
57     br FIM
58 DECR:
59     ldloc res
60     ldc.i4 1
61     sub
62     stloc res
63     br FIM
64 FIM:
65     ldloc res
66     ret
67 }

```

# Capítulo 3

## Construindo o Compilador

### 3.1 Analisador Léxico

Análise léxica é o processo de analisar a entrada de linhas de caracteres e produzir uma seqüência de símbolos chamado "símbolos léxicos", ou somente "símbolos"(tokens). O componente do compilador responsável pela execução desse processo é conhecido como Analisador léxico.

O analisador léxico, faz a varredura do programa fonte caractere por caractere e, traduz em uma seqüência de símbolos léxicos ou tokens. É nessa fase que são reconhecidas as palavras reservadas, constantes, identificadores e outras palavras que pertencem a linguagem de programação. O analisador léxico executa outras tarefas como por exemplo o tratamento de espaços, eliminação de comentários, contagem do número de linhas que o programa possui e etc.

#### 3.1.1 Analisador Léxico Manual para Pascal

Abaixo temos um analisador léxico feito manualmente.

Programa 3.1: 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 | Print
15 | Else
16 | Then
```



```

17 | Atribuicao
18 | Mais
19 | Menos
20 | Maior
21 | Menor
22 | Divisao
23 | Multiplicacao
24 | PAberto
25 | PFechado
26 | PontoVirgula
27 | Igual
28 | DoisPontos
29 | Id of string
30 | Int of string
31 | Branco
32 | EOF
33
34 type estado_lexico = {
35     pos_inicial: posicao; (* posição inicial na string *)
36     pos_final: posicao; (* posicao na string ao encontrar um estado final
37         recente *)
38     ultimo_final: estado; (* último estado final encontrado *)
39     dfa : dfa;
40     rotulo : estado -> entrada -> token
41 }
42 let estado_morto:estado = -1
43
44 let estado_inicial:estado = 0
45
46 let eh_letra (c:simbolo) = ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'
47     ')
48
49 let eh_digito (c:simbolo) = '0' <= c && c <= '9'
50
51 let eh_branco (c:simbolo) = c = ' ' || c = '\t' || c = '\n'
52
53 let eh_estado_final e el =
54     let rotulo = el.rotulo in
55     try
56         let _ = rotulo e "" in true
57     with _ -> false
58
59 let obtem_token_e_estado (str:entrada) el =
60     let inicio = el.pos_inicial
61     and fim = el.pos_final
62     and estado_final = el.ultimo_final
63     and rotulo = el.rotulo in
64     let tamanho = fim - inicio + 1 in
65     let lexema = String.sub str inicio tamanho in
66     let token = rotulo estado_final lexema in
67     let proximo_el = { el with pos_inicial = fim + 1;
68         pos_final = -1;
69         ultimo_final = -1;
70         dfa = { el.dfa with estado = estado_inicial;
71             posicao = fim + 1 }}
72     in
73     (token, proximo_el)

```

```

74
75 let rec analisador (str:entrada) tam el =
76   let posicao_atual = el.dfa.posicao
77   and estado_atual = el.dfa.estado in
78   if posicao_atual >= tam
79   then
80     if el.ultimo_final >= 0
81     then let token, proximo_el = obtem_token_e_estado str el in
82           [token; EOF]
83     else [EOF]
84   else
85     let simbolo = str.[posicao_atual]
86     and transicao = el.dfa.transicao in
87     let proximo_estado = transicao estado_atual simbolo in
88     if proximo_estado = estado_morto
89     then let token, proximo_el = obtem_token_e_estado str el in
90           token :: analisador str tam proximo_el
91     else
92       let proximo_el =
93         if eh_estado_final proximo_estado el
94         then { el with pos_final = posicao_atual;
95                  ultimo_final = proximo_estado;
96                  dfa = { el.dfa with estado = proximo_estado;
97                           posicao = posicao_atual + 1 }}
98         else { el with dfa = { el.dfa with estado = proximo_estado;
99                           posicao = posicao_atual + 1 }}
100       in
101       analisador str tam proximo_el
102
103 let lexico (str:entrada) =
104   let trans (e:estado) (c:simbolo) =
105     match (e,c) with
106     | (0, 'i') -> 7
107     | (0, 'p') -> 1
108     | (0, 't') -> 9
109     | (0, 'e') -> 10
110     | (0, ':') -> 19
111     | (0, '=') -> 21
112     | (0, '+') -> 22
113     | (0, '-') -> 23
114     | (0, '>') -> 24
115     | (0, '<') -> 25
116     | (0, '/') -> 26
117     | (0, '*') -> 27
118     | (0, '(') -> 28
119     | (0, ')') -> 29
120     | (0, ';') -> 30
121     | (0, _) when eh_letra c -> 6
122     | (0, _) when eh_digito c -> 17
123     | (0, _) when eh_branco c -> 18
124     | (0, _) ->
125     failwith ("Erro lexico: caracter desconhecido " ^ Char.
126 escaped c)
127   | (1, 'r') -> 2
128   | (2, 'i') -> 3
129   | (3, 'n') -> 4
130   | (4, 't') -> 5
131   | (7, 'f') -> 8
132   | (9, 'h') -> 11

```

```

133 | (11, 'e') -> 12
134 | (12, 'n') -> 13
135 | (10, 'l') -> 14
136 | (14, 's') -> 15
137 | (15, 'e') -> 16
138 | (19, '=') -> 20
139 | (1, _) when eh_letra c || eh_digito c -> 6
140 | (2, _) when eh_letra c || eh_digito c -> 6
141 | (3, _) when eh_letra c || eh_digito c -> 6
142 | (4, _) when eh_letra c || eh_digito c -> 6
143 | (5, _) when eh_letra c || eh_digito c -> 6
144 | (6, _) when eh_letra c || eh_digito c -> 6
145 | (7, _) when eh_letra c || eh_digito c -> 6
146 | (8, _) when eh_letra c || eh_digito c -> 6
147 | (9, _) when eh_letra c || eh_digito c -> 6
148 | (10, _) when eh_letra c || eh_digito c -> 6
149 | (11, _) when eh_letra c || eh_digito c -> 6
150 | (12, _) when eh_letra c || eh_digito c -> 6
151 | (13, _) when eh_letra c || eh_digito c -> 6
152 | (14, _) when eh_letra c || eh_digito c -> 6
153 | (15, _) when eh_letra c || eh_digito c -> 6
154 | (6, _) when eh_letra c || eh_digito c -> 6
155 | (17, _) when eh_digito c -> 17
156 | (18, _) when eh_branco c -> 18
157 | _ -> estado_morto
158 and rotulo e str =
159 match e with
160 | 8 -> If
161 | 1
162 | 2
163 | 3
164 | 4
165 | 6
166 | 7
167 | 9
168 | 10
169 | 11
170 | 12
171 | 14
172 | 15 -> Id str
173 | 17 -> Int str
174 | 18 -> Branco
175 | 5 -> Print
176 | 13 -> Then
177 | 16 -> Else
178 | 19 -> DoisPontos
179 | 20 -> Atribuicao
180 | 21 -> Igual
181 | 22 -> Mais
182 | 23 -> Menos
183 | 24 -> Maior
184 | 25 -> Menor
185 | 26 -> Divisao
186 | 27 -> Multiplicacao
187 | 28 -> PAberto
188 | 29 -> PFechado
189 | 30 -> PontoVirgula
190 | _ -> failwith ("Erro lexico: sequencia desconhecida " ^ str)
191 in let dfa = { transicao = trans;

```

```

192         estado = estado_inicial;
193         posicao = 0 }
194 in let estado_lexico = {
195     pos_inicial = 0;
196     pos_final = -1;
197     ultimo_final = -1;
198     rotulo = rotulo;
199     dfa = dfa
200 } in
201 analisador str (String.length str) estado_lexico

```

## Como Usar

Para testar o analisador léxico utilizaremos dos seguintes comandos:

```

> rlwrap ocaml
# #use "nome_do_arquivo.ml";;
# lexico "cadeia de strings";;

```

## Estrutura do Analisador Léxico

```

vinicius@vinicius-Vostro1310:~/Documentos/CC$ rlwrap ocaml
OCaml version 4.02.3

# #use "dfalexer.ml";;
type estado = int
type entrada = string
type simbolo = char
type posicao = int
type dfa = {
  transicao : estado -> simbolo -> estado;
  estado : estado;
  posicao : posicao;
}
type token =
  | If
  | Print
  | Else
  | Then
  | Atribuicao
  | Mais
  | Menos
  | Maior
  | Menor
  | Divisao
  | Multiplicacao
  | PAberto
  | PFechado
  | PontoVirgula
  | Igual
  | DoisPontos
  | Id of string
  | Int of string
  | Branco
  | EOF

```

Figura 3.1: Estrutura do Analisador(1)

```

type estado_lexico = {
  pos_inicial : posicao;
  pos_final : posicao;
  ultimo_final : estado;
  dfa : dfa;
  rotulo : estado -> entrada -> token;
}
val estado_morto : estado = -1
val estado_inicial : estado = 0
val eh_letra : simbolo -> bool = <fun>
val eh_digito : simbolo -> bool = <fun>
val eh_branco : simbolo -> bool = <fun>
val eh_estado_final : estado -> estado_lexico -> bool = <fun>
val obtem_token_e_estado : entrada -> estado_lexico -> token * estado_lexico =
  <fun>
val analisador : entrada -> posicao -> estado_lexico -> token list = <fun>
val lexico : entrada -> token list = <fun>

```

Figura 3.2: *Estrutura do Analisador(2)*

## Exemplos

Veremos alguns exemplos para um analisador da linguagem Pascal.

```

# lexico "print (a * b);";
- : token list =
[Print; Branco; PAberto; Id "a"; Branco; Multiplicacao; Branco; Id "b";
PFechado; PontoVirgula; EOF]

```

Figura 3.3: *print (a \* b);*

```

# lexico "if1 := a - 2;";
- : token list =
[Id "if1"; Branco; Atribuicao; Branco; Id "a"; Branco; Menos; Branco;
Int "2"; PontoVirgula; EOF]

```

Figura 3.4: *if1 := a - 2;*

```

# lexico "if if1 > 0;";
- : token list = [If; Branco; Id "if1"; Branco; Maior; Branco; Int "0"; EOF]

```

Figura 3.5: *if if1 > 0*

```

# lexico "then print (if1);";
- : token list =
[Then; Branco; Print; Branco; PAberto; Id "if1"; PFechado; PontoVirgula; EOF]

```

Figura 3.6: *then print (if1);*

```

# lexico "else @print (if2);";
Exception: Failure "Erro lexico: caracter desconhecido @".

```

Figura 3.7: *else @print (if2);*

### 3.1.2 Analisador Léxico Automático para Lua

Lua é uma linguagem de formato livre. Ela ignora espaços (incluindo quebras de linha) e comentários entre elementos léxicos (tokens), exceto como delimitadores entre nomes e palavras-chave.

Nomes (também chamados de identificadores) em Lua podem ser qualquer cadeia de letras, dígitos, e sublinhados, que não iniciam com um dígito.

No programa a seguir serão implementadas as palavras reservadas da linguagem Lua, ou seja, as palavras que não podem ser usadas como nomes, e também as demais regras da linguagem, tais como tratamento de comentários e strings.

#### Programa 3.2: lexico.mll

```
1 {
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   let erro lin col msg =
19     let mensagem = sprintf "%d-%d: %s" lin col msg in
20     failwith mensagem
21
22
23   type tokens = AND
24       | BREAK
25       | DO
26       | ELSE
27       | ELSEIF
28       | END
29       | FALSE
30       | FOR
31       | FUNCTION
32       | IF
33       | IN
34       | LOCAL
35       | NIL
36       | NOT
37       | OR
38       | READ
39       | REPEAT
40       | RETURN
41       | THEN
42       | TONUMBER
43       | TRUE
44       | UNTIL
45       | WHILE
46       | WRITE
47       | ATRIB
48       | MAIS
49       | MENOS
50       | MAIOR
```

```

51         | MENOR
52         | DIV
53         | MULT
54         | MOD
55         | MAIORIGUAL
56         | MENORIGUAL
57         | IGUAL
58         | DIFERENTE
59         | APARENT
60         | FPARENT
61         | ACHAVES
62         | FCHAVES
63         | ACOLCHETE
64         | FCOLCHETE
65         | PONTOVIRG
66         | VIRGULA
67         | PONTO
68         | LFLOAT of float
69         | LITINT of int
70         | LITSTRING of string
71         | ID of string
72         | EOF
73     }
74
75     let digito = ['0' - '9']
76
77     let pontoflutuante = digito+ ('.') (digito)+
78     let inteiro = digito+
79
80     let letra = ['a' - 'z' 'A' - 'Z']
81     let identificador = letra ( letra | digito | '_' ) *
82
83     let brancos = [' ' '\t'] +
84     let novalinha = '\r' | '\n' | "\\n" | "\r\n"
85
86     let comentario = "--" [ ^ '[' [ ^ '\r' '\n' ] * | "--" [ " [ ^ '[' ] [ ^ '\r' '\n' ] *
87
88     rule token = parse
89     brancos      { token lexbuf }
90 | novalinha     { incr_num_linha lexbuf; token lexbuf }
91 | comentario   { token lexbuf }
92 | "--" [ "      { let pos = lexbuf.lex_curr_p in
93                   let lin = pos.pos_lnum
94                   and col = pos.pos_cnum - pos.pos_bol - 1 in
95                   comentario_bloco (lin-1) col 0 lexbuf }
96 | '('          { APARENT }
97 | ')'          { FPARENT }
98 | '{'          { ACHAVES }
99 | '}'          { FCHAVES }
100 | '['          { ACOLCHETE }
101 | ']'          { FCOLCHETE }
102 | '%'          { MOD }
103 | '>'          { MAIOR }
104 | '<'          { MENOR }
105 | ">="         { MAIORIGUAL }
106 | "<="         { MENORIGUAL }
107 | ';'          { PONTOVIRG }
108 | ','          { VIRGULA }

```

```

109 | '.'          { PONTO }
110 | '='          { ATRIB }
111 | '+'          { MAIS }
112 | '-'          { MENOS }
113 | '*'          { MULT }
114 | '/'          { DIV }
115 | "~="         { DIFERENTE }
116 | "=="         { IGUAL }
117 | pontoflutuante as num { let numero = float_of_string num in
118 |                          LFLOAT numero }
119 | inteiro as num { let numero = int_of_string num in
120 |                      LITINT numero }
121 | "and"        { AND }
122 | "break"      { BREAK }
123 | "do"         { DO }
124 | "else"       { ELSE }
125 | "elseif"     { ELSEIF }
126 | "end"        { END }
127 | "false"     { FALSE }
128 | "for"        { FOR }
129 | "function"   { FUNCTION }
130 | "if"         { IF }
131 | "in"         { IN }
132 | "local"      { LOCAL }
133 | "nil"        { NIL }
134 | "not"        { NOT }
135 | "io.read"    { READ }
136 | "io.write"   { WRITE }
137 | "or"         { OR }
138 | "repeat"     { REPEAT }
139 | "return"     { RETURN }
140 | "then"       { THEN }
141 | "true"       { TRUE }
142 | "until"      { UNTIL }
143 | "tonumber"   { TONUMBER }
144 | "while"      { WHILE }
145 | identificador as id { ID id }
146 | '"'         { let pos = lexbuf.lex_curr_p in
147 |                  let lin = pos.pos_lnum
148 |                  and col = pos.pos_cnum - pos.pos_bol - 1 in
149 |                  let buffer = Buffer.create 1 in
150 |                  let str = leia_string lin col buffer lexbuf in
151 |                  LITSTRING str }
152 | _ as c      { failwith (msg_erro lexbuf c) }
153 | eof         { EOF }
154 | and comentario_bloco lin col n = parse
155 | "]"         { if n=0 then token lexbuf
156 |                  else comentario_bloco lin col (n-1) lexbuf }
157 | "--" "["   { comentario_bloco lin col 0 lexbuf }
158 | _          { comentario_bloco lin col n lexbuf }
159 | eof        { erro lin col "Comentario nao fechado" }
160 | and leia_string lin col buffer = parse
161 | '"'        { Buffer.contents buffer }
162 | "\\t"      { Buffer.add_char buffer '\t'; leia_string lin col buffer
163 |                  lexbuf }
163 | "\\n"      { Buffer.add_char buffer '\n'; leia_string lin col buffer lexbuf
164 |                  }
164 | "\\'''"    { Buffer.add_char buffer "'"; leia_string lin col buffer
165 |                  lexbuf }

```



```

165 | '\\''\\' { Buffer.add_char buffer '\\'; leia_string lin col buffer
      lexbuf }
166 | _ as c   { Buffer.add_char buffer c; leia_string lin col buffer lexbuf
      }
167 | eof      { erro lin col "A string nao foi fechada"}

```

## Carregador

Também é necessário o uso de um carregador para fazer load dos arquivos ".cmo" e para pegar a entrada por arquivo ou por uma string.

### Programa 3.3: carregador.ml

```

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

```

## Compilando

Para compilar o analisador léxico de Lua utilizaremos dos seguintes comandos:

```
> ocamllex lexico.mll
```

Após este comando será gerado um arquivo de nome "lexico.ml" que será usado no próximo comando.

```
> ocamlc -c lexico.ml
```

Pode-se verificar que após este comando serão criados dois arquivos, um chamado "lexico.cmi" e outro com nome "lexico.cmo".

## Testando

Para testar o analisador léxico utilizaremos dos seguintes comandos:

```
> rlwrap ocaml
# #use "carregador.ml";;
# lex "arquivo.lua";;
```

## Exemplo

O seguinte programa será utilizado como exemplo:

### Programa 3.4: micro03.lua

```
1 local numero
2
3 io.write("Digite um numero: ")
4 numero = tonumber(io.read())
5
6
7 if numero >= 100 then
8
9     if numero <= 200 then
10         io.write("O número está no intervalo entre 100 e 200\n")
11     else
12         io.write("O número não está no intervalo entre 100 e 200\n")
13     end
14
15 else
16     io.write("O número não está no intervalo entre 100 e 200\n")
17 end
```

e produzirá uma saída com os seguintes tokens:

```
# lex "micro03.lua";;
- : Lexico.tokens list =
[Lexico.LOCAL; Lexico.ID "numero"; Lexico.WRITE; Lexico.APARENT;
Lexico.LITSTRING "Digite um numero: "; Lexico.FPARENT; Lexico.ID "numero";
Lexico.ATRIB; Lexico.TONUMBER; Lexico.APARENT; Lexico.READ; Lexico.APARENT;
Lexico.FPARENT; Lexico.FPARENT; Lexico.IF; Lexico.ID "numero";
Lexico.MAIORIGUAL; Lexico.LITINT 100; Lexico.THEN; Lexico.IF;
Lexico.ID "numero"; Lexico.MENORIGUAL; Lexico.LITINT 200; Lexico.THEN;
Lexico.WRITE; Lexico.APARENT;
Lexico.LITSTRING "O numero esta no intervalo entre 100 e 200\n";
Lexico.FPARENT; Lexico.ELSE; Lexico.WRITE; Lexico.APARENT;
Lexico.LITSTRING "O numero nao esta no intervalo entre 100 e 200\n";
Lexico.FPARENT; Lexico.END; Lexico.ELSE; Lexico.WRITE; Lexico.APARENT;
Lexico.LITSTRING "O numero nao esta no intervalo entre 100 e 200\n";
Lexico.FPARENT; Lexico.END; Lexico.EOF]
```

Figura 3.8: saída do analisador léxico

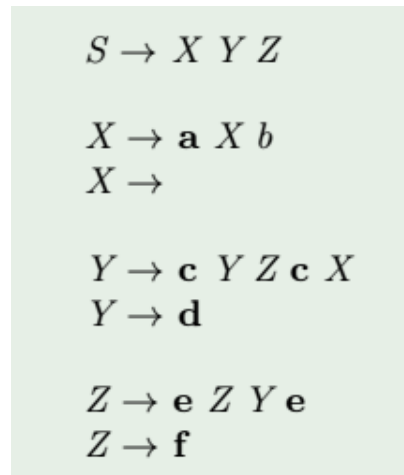
## 3.2 Analisador sintático

Análise sintática é o processo de analisar uma sequência de entrada para determinar sua estrutura gramatical segundo uma determinada gramática formal. A análise sintática

transforma um texto na entrada em uma estrutura de dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise léxica é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura.

### 3.2.1 Parser preditivo

Construindo um parser preditivo para a gramática a seguir:


$$\begin{aligned} S &\rightarrow X Y Z \\ X &\rightarrow \mathbf{a} X b \\ X &\rightarrow \\ Y &\rightarrow \mathbf{c} Y Z \mathbf{c} X \\ Y &\rightarrow \mathbf{d} \\ Z &\rightarrow \mathbf{e} Z Y \mathbf{e} \\ Z &\rightarrow \mathbf{f} \end{aligned}$$

**Figura 3.9:** gramática para gerar o parser

#### Programa 3.5: parserlexico

```
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 let digito = ['0' - '9']
24 let inteiro = digito+
25
26 let letra = ['a' - 'z' 'A' - 'Z']
27 let identificador = letra ( letra | digito | '_' )*
```

```

28
29 let brancos = [' ' '\t']+
30 let novalinha = '\r' | '\n' | "\r\n"
31
32 let comentario = "//" [^ '\r' '\n' ]*
33
34 rule token = parse
35   brancos      { token lexbuf }
36 | novalinha    { incr_num_linha lexbuf; token lexbuf }
37 | comentario   { token lexbuf }
38 | "/*"         { comentario_bloco 0 lexbuf }
39 | 'a'          { A }
40 | 'b'          { B }
41 | 'c'          { C }
42 | 'd'          { D }
43 | 'e'          { E }
44 | 'f'          { F }
45 | _ as c       { failwith (msg_error lexbuf c) }
46 | eof          { EOF }
47 and comentario_bloco n = parse
48   "*/"         { if n=0 then token lexbuf
49                 else comentario_bloco (n-1) lexbuf }
50 | "/*"         { comentario_bloco (n+1) lexbuf }
51 | _            { comentario_bloco n lexbuf }
52 | eof          { failwith "Comentário não fechado" }
53 and leia_string buffer = parse
54   '"'          { Buffer.contents buffer}
55 | "\\t"        { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
56 | "\\n"        { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
57 | '\\\'' '\'' { Buffer.add_char buffer '\''; leia_string buffer lexbuf }
58 | '\\\'' '\\\'' { Buffer.add_char buffer '\\\''; leia_string buffer lexbuf }
59 | _ as c       { Buffer.add_char buffer c; leia_string buffer lexbuf }
60 | eof          { failwith "A string não foi fechada"}

```

### Programa 3.6: parsersintatico

```

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

```

### Programa 3.7: parserarv

```

1 #load "lexico.cmo";;
2 open Sintatico;;
3
4 type variavel = XYZ of variavel * variavel * variavel
5               | AXB of string * variavel * string
6               | YCZCX of string * variavel * variavel * string * variavel
7               | YD of string
8               | ZEZYE of string * variavel * variavel * string
9               | ZF of string
10              | VAZIO
11
12 let tk = ref EOF

```

```

13 let lexbuf = ref (Lexing.from_string "")
14
15 let prox () = tk := Lexico.token !lexbuf
16
17 let to_str tk =
18   match tk with
19     A -> "a"
20   | B -> "b"
21   | C -> "c"
22   | D -> "d"
23   | E -> "e"
24   | F -> "f"
25   | EOF -> "eof"
26
27 let erro esp =
28   let msg = Printf.sprintf "Erro: esperava %s mas encontrei %s"
29     esp (to_str !tk)
30   in
31   failwith msg
32
33 let consome t = if (!tk == t) then prox() else erro (to_str t)
34
35 let rec ntS () =
36   match !tk with
37     A -> let var1 = ntX() in
38         let var2 = ntY() in
39         let var3 = ntZ() in
40         XYZ (var1, var2, var3)
41   | C -> let var1 = ntX() in
42         let var2 = ntY() in
43         let var3 = ntZ() in
44         XYZ (var1, var2, var3)
45   | D -> let var1 = ntX() in
46         let var2 = ntY() in
47         let var3 = ntZ() in
48         XYZ (var1, var2, var3)
49   | _ -> erro "a, c ou d"
50
51 and ntX () =
52   match !tk with
53     A -> let _ = consome A in
54         let var1 = ntX() in
55         let _ = consome B in
56         AXB(to_str A, var1, to_str B)
57   | B -> VAZIO
58   | C -> VAZIO
59   | D -> VAZIO
60   | E -> VAZIO
61   | F -> VAZIO
62   | _ -> erro "a"
63
64 and ntY () =
65   match !tk with
66     C -> let _ = consome C in
67         let var1 = ntY() in
68         let var2 = ntZ() in
69         let _ = consome C in
70         let var3 = ntX() in
71         YCZCX(to_str C, var1, var2, to_str C, var3)

```

```

72 | D      -> let _ = consome D in
73         YD(to_str D)
74 | _      -> erro "C ou D"
75
76 and ntZ () =
77     match !tk with
78     E      -> let _ = consome E in
79                 let var1 = ntZ() in
80                 let var2 = ntY() in
81                 let _ = consome E in
82                 ZEZYE(to_str E , var1, var2, to_str E)
83 | F      -> let f = consome F in
84                 ZF(to_str F)
85 | _      -> erro "e ou f"
86
87 let parser str =
88     lexbuf := Lexing.from_string str;
89     prox ();
90     let arv = ntS () in
91     match !tk with
92     EOF -> let _ = Printf.printf "Ok!\n" in arv
93 | _ -> erro "fim da entrada"
94
95 let teste () =
96     let entrada =
97         "abcdfcf"
98     in
99     parser entrada

```

## Compilar

Para compilar deve-se realizar os seguintes comandos:

```

> ocamllex lexico.mll
> ocaml -c sintatico.mli
> ocaml -c lexico.ml

```

## Testar

Para testar serão utilizados os seguintes comandos:

```

> rlwrap ocaml
# #use "sintaticoArv.ml";;
# teste();;

```

## Exemplo

Como pode ser observado na linha 97 do **Programa 3.7**, será utilizada a palavra "abcdfcf" para verificar se esta pertence a linguagem gerada pela gramática descrita na **figura 3.9**

```
# teste();;
Ok!
- : variavel =
XYZ (AXB ("a", VAZIO, "b"), YCYZCX ("c", YD "d", ZF "f", "c", VAZIO), ZF "f")
```

**Figura 3.10:** *testando o parser*

Além de verificar se a palavra pertence a linguagem, a função "teste()" nos mostra a árvore sintática gerada neste problema.

### 3.2.2 Analisador sintático com mensagens de erros

Nesta subseção será criado um analisador sintático abstrato completo para a linguagem Lua.

Primeiramente os seguintes comandos devem ser executados para a instalação do "opam" e do "menhir":

```
> sudo apt install opam
> opam init
> eval `opam config env`
> sudo apt-get install m4
> opam install menhir
```

### Criando os arquivos iniciais

#### Programa 3.8: Analisador - lexico.mll

```
1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6   exception Erro of string
7
8   let incr_num_linha lexbuf =
9     let pos = lexbuf.lex_curr_p in
10    lexbuf.lex_curr_p <- { pos with
11      pos_lnum = pos.pos_lnum + 1;
12      pos_bol = pos.pos_cnum;
13    }
14
15 }
16
17 let digito = ['0' - '9']
18
19 let inteiro = digito+
20 let pontoflutuante = digito+ ('.') (digito)+
21
22
23 let letra = ['a' - 'z' 'A' - 'Z']
24 let identificador = letra ( letra | digito | '_' ) *
25
26 let brancos = [ ' ' '\t' ] +
```

```

27 let novalinha = '\r' | '\n' | "\r\n"
28
29 let comentario = "--" [^ '['][^ '\r' '\n' ]* | "--[" [^ '[' ][^ '\r' '\n'
    ]*
30
31
32
33 rule token = parse
34   brancos      { token lexbuf }
35 | novalinha    { incr_num_linha lexbuf; token lexbuf }
36 | comentario  { token lexbuf }
37 | "--" "["    { comentario_bloco 0 lexbuf }
38 | '('         { APARENT }
39 | ')'         { FPARENT }
40 | '{'         { ACHAVES }
41 | '}'         { FCHAVES }
42 | '['         { ACOLCHETE }
43 | ']'         { FCOLCHETE }
44 | '%'         { MOD }
45 | '>'         { MAIOR }
46 | '<'         { MENOR }
47 | ">="        { MAIORIGUAL }
48 | "<="        { MENORIGUAL }
49 | ';'         { PONTOVIRG }
50 | ','         { VIRGULA }
51 | '.'         { PONTO }
52 (*| ".."      { PONTOPONTO}*)
53 | ':'         { DOISPONTOS }
54 | '='         { ATRIB }
55 | '+'         { MAIS }
56 | '-'         { MENOS }
57 | '*'         { MULT }
58 | '/'         { DIV }
59 | "~="        { DIFERENTE }
60 | "=="        { IGUAL }
61 | pontoflutuante as num { let numero = float_of_string num in
62                           FLOAT numero }
63 | "float"      { PFLUT }
64 | inteiro as num { let numero = int_of_string num in
65                     INT numero }
66 | "int"        { INTEIRO }
67 | "char"       { CHAR }
68 | "and"        { AND }
69 (*| "break"    { BREAK}*)
70 | "do"         { DO }
71 | "else"       { ELSE }
72 | "elseif"     { ELSEIF }
73 | "end"        { END }
74 | "false"      { FALSE }
75 | "for"        { FOR }
76 | "function"   { FUNCTION }
77 | "if"         { IF }
78 | "in"         { IN }
79 | "local"      { LOCAL }
80 | "nil"        { NIL }
81 | "not"        { NOT }
82 | "io.read"    { READ }
83 | "io.write"   { WRITE }
84 | "or"         { OR }

```



```

85 | "repeat"    { REPEAT }
86 | "return"   { RETURN }
87 | "then"     { THEN }
88 | "true"     { TRUE }
89 | "until"    { UNTIL }
90 | "tonumber" { TONUMBER }
91 | "while"    { WHILE }
92 | identificador as id { ID id }
93 | '"'        { let buffer = Buffer.create 1 in
94 |             let str = leia_string buffer lexbuf in
95 |             STRING str }
96 | _          { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme lexbuf))
97 | eof        { EOF }
98 and comentario_bloco n = parse
99   "]"        { if n=0 then token lexbuf
100 |             else comentario_bloco (n-1) lexbuf }
101 | "--"["["   { comentario_bloco 0 lexbuf }
102 | "\n"       { incr_num_linha lexbuf; comentario_bloco n lexbuf }
103 | _         { comentario_bloco n lexbuf }
104 | eof       { raise (Erro "Comentário não terminado") }
105 and leia_string buffer = parse
106   '"'       { Buffer.contents buffer }
107 | "\\t"     { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
108 | "\\n"     { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
109 | '\\\''"' { Buffer.add_char buffer '"'; leia_string buffer lexbuf }
110 | '\\\''\\' { Buffer.add_char buffer '\\'; leia_string buffer lexbuf }
111 | _ as c   { Buffer.add_char buffer c; leia_string buffer lexbuf }
112 | eof      { raise (Erro "A string não foi terminada") }

```

### Programa 3.9: Analisador - ast.ml

```

1 (* The type of the abstract syntax tree (AST). *)
2
3 type identificador = string
4
5 type programa = Programa of block
6 and block = stat list * retstat option
7
8 and stat = Atribuicao of varlist * explist
9 | Atribuicao2 of varlist * read
10 | Funccall of functioncall
11 | Do of block
12 | While of exp * block
13 | Repeat of block * exp
14 | If of exp * tokens * block * elseif_rule list * else_block_rule
15 | For1 of identificador * exp * exp * comma_exp_rule option *
16 | For2 of name_list * explist * block
17 | Function of funcname * funcbody
18 | Local of namelist
19 | Write of args
20 | Pontovirg
21
22
23 and retstat = Return of explist option
24 and varlist = var * virgula_var_rule list
25

```

```

26 and var = Var_i of identificador
27     |Var_exp of prefixexp * exp
28     |Var_id of prefixexp * identificador
29
30 and prefixexp = Var of var
31     |Funcao of functioncall
32     |Exp of exp
33
34 and functioncall = ChamadaFunc1 of prefixexp * args
35     |ChamadaFunc2 of prefixexp * identificador * args
36
37 and args = ArgParent of explist option
38     |TabelaConst of constTabela
39     |String of string
40
41 and virgula_var_rule = var
42 and explist = exp * virgula_exp_rule list
43 and virgula_exp_rule = exp
44
45 and elseif_rule = Elif of exp * tokens * block
46 and else_block_rule = Else of block
47
48 and comma_exp_rule = exp
49 and namelist = identificador * atribuicao_explist_regra option *
    virgula_id_rule list
50 and virgula_id_rule = identificador * atribuicao_explist_regra option
51
52 and name_list = identificador * virgul_a_id_rule list
53 and virgul_a_id_rule = identificador
54
55 and funcname = NomeFunc of identificador * ponto_id_rule list *
    doispontos_id_rule option
56 and ponto_id_rule = Ponto of identificador
57 and doispontos_id_rule = DoisPontos of identificador
58
59 and funcbody =CorpoFunc of parlist option * block
60 and parlist = ListaParam of listanome
61 and listanome = (identificador * tokens) * virgula_id_regra list
62 and virgula_id_regra = identificador * tokens
63
64 and atribuicao_explist_rule = Recebe of explist
65 and atribuicao_explist_regra = exp
66
67 and exp = Nil
68     |False
69     |True
70     |Int of int
71     |Float of float
72     |String of string
73     |Tabela of constTabela
74     |Funcdef of functiondef
75     |Prefix of prefixexp
76     |Expressao of exp * tokens * exp
77     |ExpressaoUn of tokens * exp
78     |Tonumber of read
79
80 and constTabela = listacampos option
81
82 and listacampos = campo * fieldsep_field_rule list * sepcampos option

```

```

83
84 and campo = AtribExp of exp * exp
85                 | AtribId of identificador * exp
86                 | Exp_campo of exp
87
88 and fieldsep_field_rule = sepcampos * campo
89 and sepcampos = tokens
90
91 and read = Read
92
93 and functiondef = Functiondef
94
95 and tokens = And
96             | Not
97             | Or
98             | Mais
99             | Menos
100            | Maior
101            | Menor
102            | Div
103            | Mult
104            | Mod
105            | MaiorIgual
106            | MenorIgual
107            | Igual
108            | Diferente
109            | PontoVirgula
110            | Entao
111            | Inteiro
112            | PFlut
113            | Char
114            | Virgula

```

### Programa 3.10: Analisador - sintatico.mly

```

1 %{
2   open Ast
3   %}
4
5 %token <int> INT
6 %token <string> ID
7 %token <string> STRING
8 %token <float> FLOAT
9
10 %token INTEIRO
11 %token CHAR
12 %token PFLUT
13
14 %token AND
15 (*%token BREAK*)
16 %token DO
17 %token ELSE
18 %token ELSEIF
19 %token END
20 %token FALSE
21 %token FOR
22 %token FUNCTION
23 %token IF
24 %token IN

```

```

25 %token LOCAL
26 %token NIL
27 %token NOT
28 %token OR
29 %token READ
30 %token REPEAT
31 %token RETURN
32 %token THEN
33 %token TONUMBER
34 %token TRUE
35 %token UNTIL
36 %token WHILE
37 %token WRITE
38 %token ATRIB
39 %token MAIS
40 %token MENOS
41 %token MAIOR
42 %token MENOR
43 %token DIV
44 %token MULT
45 %token MOD
46 %token MAIORIGUAL
47 %token MENORIGUAL
48 %token IGUAL
49 %token DIFERENTE
50 %token APARENT
51 %token FPARENT
52 %token ACHAVES
53 %token FCHAVES
54 %token ACOLCHETE
55 %token FCOLCHETE
56 %token VIRGULA
57 %token PONTOVIRG
58 %token PONTO
59 (*%token PONTOPONTO *)
60 %token DOISPONTOS
61 %token EOF
62
63 %left OR
64 %left AND
65 %left IGUAL DIFERENTE MAIOR MENOR MAIORIGUAL MENORIGUAL
66 %left MAIS MENOS
67 %left MULT DIV MOD
68 %left NOT
69
70 %start <Ast.programa> programa
71
72 %%
73
74 programa:
75   | bl=block EOF { Programa (bl) }
76   ;
77
78 block:
79   | st=stat* rt=retstat? {(st, rt)}
80   ;
81
82 stat:
83   | PONTOVIRG {Pontovirg }

```

```

84 | vrl=varlist ATRIB exl=explist {Atribuicao (vrl, exl)}
85 | vrl=varlist ATRIB r=read { Atribuicao2 (vrl,r)}
86 | fctc=functioncall {Funcall (fctc) }
87 | DO bl=block END {Do (bl) }
88 | WHILE e=exp DO bl=block END {While (e,bl) }
89 | REPEAT bl=block UNTIL e=exp {Repeat (bl,e) }
90 | IF e=exp THEN bl=block elif=elseif_rule* el=else_block_rule? END {If (
    e,Entao,bl,elif,el) }
91 | FOR id=ID ATRIB el=exp VIRGULA e2=exp cer=comma_exp_rule? DO bl=block
    END {For1 (id,el,e2,cer,bl)}
92 | FOR nl=name_list IN expl=explist DO bl=block END {For2 (nl,expl,bl) }
93 | FUNCTION fn=funcname fb=funcbody {Function (fn,fb) }
94 | LOCAL nl=namelist { Local (nl)}
95 | WRITE a=args {Write (a)}
96 ;
97
98 elseif_rule:
99 | ELSEIF e=exp THEN bl=block {Elif (e,Entao,bl) }
100 ;
101
102 else_block_rule:
103 | ELSE bl=block {Else (bl) }
104 ;
105
106 comma_exp_rule:
107 | VIRGULA e=exp {(e) }
108 ;
109 atribuicao_explist_rule:
110 | ATRIB el=explist {Recebe (el) }
111 ;
112
113 atribuicao_explist_regra:
114 | ATRIB e=exp {(e) }
115 ;
116
117 retstat:
118 | RETURN el=explist? PONTOVIRG? {Return (el) }
119 ;
120
121 funcname:
122 | id=ID pir=ponto_id_rule* dpir=doispontos_id_rule? {NomeFunc (id,pir,
    dpir) }
123 ;
124
125 ponto_id_rule:
126 | PONTO id=ID {Ponto (id) }
127 ;
128
129 doispontos_id_rule:
130 | DOISPONTOS id=ID {DoisPontos (id) }
131 ;
132
133 varlist:
134 | v=var vvr=virgula_var_rule* {(v,vvr) }
135 ;
136
137 virgula_var_rule:
138 | VIRGULA v=var { (v) }
139 ;

```

```

140
141 var:
142   | id=ID {Var_i (id)}
143   | p=prefixexp ACOLCHETE e=exp FCOLCHETE {Var_exp (p,e) }
144   | p=prefixexp PONTO id=ID {Var_id (p,id) }
145   ;
146
147 namelist:
148   | id=ID aer=atribuicao_explist_regra? vir=virgula_id_rule* {(id,aer,vir)
149     }
150   ;
151 virgula_id_rule:
152   | VIRGULA id=ID aer=atribuicao_explist_regra? {(id,aer) }
153   ;
154
155 name_list:
156   | id=ID vir=virgul_a_id_rule* {(id,vir)}
157   ;
158
159 virgul_a_id_rule:
160   | VIRGULA id=ID {(id) }
161   ;
162
163 listanome:
164   | id=ID DOISPONTOS INTEIRO vir=virgula_id_regra* {(id,Inteiro),vir)}
165   | id=ID DOISPONTOS PFLUT vir=virgula_id_regra* {(id,PFlut),vir)}
166   | id=ID DOISPONTOS CHAR vir=virgula_id_regra* {(id,Char),vir)}
167   ;
168
169 virgula_id_regra:
170   | VIRGULA id=ID DOISPONTOS INTEIRO {(id, Inteiro) }
171   | VIRGULA id=ID DOISPONTOS PFLUT {(id, PFlut) }
172   | VIRGULA id=ID DOISPONTOS CHAR {(id, Char) }
173   ;
174
175
176 explist:
177   | e=exp ver=virgula_exp_rule* {(e, ver)}
178   ;
179
180 virgula_exp_rule:
181   | VIRGULA e=exp {(e) }
182   ;
183
184 exp:
185   | NIL {Nil }
186   | FALSE {False }
187   | TRUE {True }
188   | i=INT {Int (i) }
189   | f=FLOAT {Float (f)}
190   | s=STRING {String (s)}
191   | ct=constTabela {Tabela (ct)}
192   | f=functiondef {Funcdef (f)}
193   | p=prefixexp {Prefix (p)}
194   | e1=exp MAIS e2=exp {Expressao (e1,Mais,e2)}
195   | e1=exp MENOS e2=exp {Expressao (e1,Menos,e2)}
196   | e1=exp MULT e2=exp {Expressao (e1,Mult,e2)}
197   | e1=exp DIV e2=exp {Expressao (e1,Div,e2)}

```

```

198 | e1=exp MOD e2=exp {Expressao (e1,Mod,e2)}
199 (* | e1=exp PONTOPONTO e2=exp {Expressao (e1,PontoPonto,e2)}
    concatenação de string *)
200 | e1=exp MENOR e2=exp {Expressao (e1,Menor,e2)}
201 | e1=exp MENORIGUAL e2=exp {Expressao (e1,MenorIgual,e2)}
202 | e1=exp MAIOR e2=exp {Expressao (e1,Maior,e2)}
203 | e1=exp MAIORIGUAL e2=exp {Expressao (e1,MaiorIgual,e2)}
204 | e1=exp IGUAL e2=exp {Expressao (e1,Igual,e2)}
205 | e1=exp DIFERENTE e2=exp {Expressao (e1,Diferente,e2)}
206 | e1=exp AND e2=exp {Expressao (e1,And,e2)}
207 | e1=exp OR e2=exp {Expressao (e1,Or,e2)}
208 | MENOS e1=exp {ExpressaoUn (Menos,e1)}
209 | NOT e1=exp {ExpressaoUn (Not,e1)}
210 | TONUMBER APARENT r=read FPARENT {Tonumber(r)}
211 ;
212
213
214
215 read:
216 | READ APARENT FPARENT{ Read }
217 ;
218
219 prefixexp:
220 | v=var {Var(v) }
221 | fc=functioncall {Funcao (fc) }
222 | APARENT e=exp FPARENT {Exp (e) }
223 ;
224
225 functioncall:
226 | p=prefixexp a=args {ChamadaFunc1 (p,a) }
227 | p=prefixexp DOISPONTOS id=ID a=args {ChamadaFunc2 (p,id,a) }
228 ;
229
230 args:
231 | APARENT el=explist? FPARENT {ArgParent (el)}
232 | t=constTabela { TabelaConst(t) }
233 | s=STRING {String (s)}
234 ;
235
236 functiondef:
237 | FUNCTION funcbody { Functiondef}
238 ;
239
240 funcbody:
241 | APARENT p=parlist? FPARENT b=block END {CorpoFunc (p,b) }
242 ;
243
244 parlist:
245 | nl=listanome {ListaParam (nl)}
246 ;
247
248 constTabela:
249 | ACHAVES lc=listacampos? FCHAVES { (lc)}
250
251 listacampos:
252 | c=campo ffr=fieldsep_field_rule* sc=sepcampos? { (c,ffr,sc) }
253 ;
254
255 fieldsep_field_rule:

```

```

256 | sc=sepcampos c=campo {(sc,c)}
257 ;
258
259 campo:
260 | ACOLCHETE e1=exp FCOLCHETE ATRIB e2=exp {AtribExp(e1,e2) }
261 | id=ID ATRIB e=exp {AtribId(id,e) }
262 | e=exp {Exp_campo(e) }
263 ;
264
265 sepcampos:
266 | VIRGULA { Virgula}
267 | PONTOVIRG {PontoVirgula }
268 ;

```

### Programa 3.11: Analisador - sintaticoTest.ml

```

1 open Printf
2 open Lexing
3
4 open Ast
5 open ErroSint (* nome do módulo contendo as mensagens de erro *)
6
7 exception Erro_Sintatico of string
8
9 module S = MenhirLib.General (* Streams *)
10 module I = Sintatico.MenhirInterpreter
11
12 let posicao lexbuf =
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 "linha %d, coluna %d" lin col
17
18 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
19    checkpoint *)
20 let pilha checkpoint =
21   match checkpoint with
22   | I.HandlingError amb -> I.stack amb
23   | _ -> assert false (* Isso não pode acontecer *)
24
25 let estado checkpoint : int =
26   match Lazy.force (pilha checkpoint) with
27   | S.Nil -> (* O parser está no estado inicial *)
28     0
29   | S.Cons (I.Element (s, _, _, _), _) ->
30     I.number s
31
32 let sucesso v = Some v
33
34 let falha lexbuf (checkpoint : Ast.programa I.checkpoint) =
35   let estado_atual = estado checkpoint in
36   let msg = message estado_atual in
37   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
38                                     (Lexing.lexeme_start lexbuf) msg))
39
40 let loop lexbuf resultado =
41   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
42   I.loop_handle sucesso (falha lexbuf) fornecedor resultado

```



```

43
44
45
46 let parse_com_erro lexbuf =
47   try
48     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
49   with
50   | Lexico.Erro msg ->
51     printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52     None
53   | Erro_Sintatico msg ->
54     printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
55     None
56
57 let parse s =
58   let lexbuf = Lexing.from_string s in
59   let ast = parse_com_erro lexbuf in
60   ast
61
62 let parse_arq nome =
63   let ic = open_in nome in
64   let lexbuf = Lexing.from_channel ic in
65   let ast = parse_com_erro lexbuf in
66   let _ = close_in ic in
67   ast

```

## Gerando as mensagens de erro

Após criarmos os arquivos iniciais, as mensagens de erro devem ser geradas. Foi executado no terminal:

```
> menhir -v --list-errors sintatico.mly > sintatico.msg
```

Após a execução um arquivo chamado "sintatico.msg" foi criado. Este arquivo foi modificado da seguinte maneira: onde tinha a string «YOUR SYNTAX ERROR MESSAGE HERE» foi passado para “Esperava um bloco de comandos” por exemplo.

Depois o seguinte comando foi executado:

```
> menhir -v --list-errors sintatico.mly --compile-errors sintatico.msg >
  erroSint.ml
```

## Criando o carregador

O seguinte carregador foi criado com o nome de “ocamlinit”.

### Programa 3.12: Analisador - .ocamlinit

```

1 #use "topfind";;
2 #require "menhirLib";;
3 #directory "_build";;
4 #load "erroSint.cmo";;
5 #load "sintatico.cmo";;
6 #load "lexico.cmo";;

```

```
7 #load "ast.cmo";;  
8 #load "sintaticoTest.cmo";;  
9 open Ast  
10 open SintaticoTest
```

## Compilando

Para compilar, o seguinte comando foi executado:

```
> ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package  
    menhirLib sintaticoTest.byte
```

## Testando

Será testado o seguinte arquivo:

### Programa 3.13: micro11.lua

```
1 function verifica(n)  
2   local res  
3  
4   if n > 0 then  
5     res = 1  
6   elseif n < 0 then  
7     res = -1  
8   else  
9     res = 0  
10  end  
11  
12  return res  
13 end  
14  
15 local numero, x  
16  
17 io.write("Digite um número: ")  
18 numero = tonumber(io.read())  
19  
20 x = verifica(numero)  
21  
22 if x == 1 then  
23   io.write("Numero positivo\n")  
24  
25 elseif x == 0 then  
26   io.write("Zero\n")  
27  
28 else  
29   io.write("Numero Negativo\n")  
30  
31 end
```

Para testar deve-se entrar no ocaml assim:

```
> rlwrap ocaml
```

Depois digitar no terminal:

```
> parse_arq "micro11.lua";;
```

O retorno deste comando será:

```
# parse_arq "micro11.lua";;
- : Ast.programa option option =
Some
  (Some
    (Programa
      ([Function (NomeFunc ("verifica", [], None),
        CorpoFunc (Some (ListaParam (("n", Inteiro), [])),
          ([Local ("res", None, []);
            If (Expressao (Prefix (Var (Var_i "n")), Maior, Int 0), Entao,
              ([Atribuicao ((Var_i "res", []), (Int 1, []))], None),
              [Elif (Expressao (Prefix (Var (Var_i "n")), Menor, Int 0), Entao,
                ([Atribuicao ((Var_i "res", []),
                  (ExpressaoUn (Menos, Int 1), []))],
                  None))]
                ,
                Some (Else ([Atribuicao ((Var_i "res", []), (Int 0, []))], None))),
              Some (Return (Some (Prefix (Var (Var_i "res")), [])))));
            Local ("numero", None, [("x", None)]);
            Write (ArgParent (Some (String "Digite um n\195\186mero: ", [])));
            Atribuicao ((Var_i "numero", []), (Tonumber Read, []));
            Atribuicao ((Var_i "x", []),
              (Prefix
                (Funcao
                  (ChamadaFunc1 (Var (Var_i "verifica"),
                    ArgParent (Some (Prefix (Var (Var_i "numero")), [])))))
                ,
                []));
            If (Expressao (Prefix (Var (Var_i "x")), Igual, Int 1), Entao,
              ([Write (ArgParent (Some (String "Numero positivo\n", []))], None),
              [Elif (Expressao (Prefix (Var (Var_i "x")), Igual, Int 0), Entao,
                ([Write (ArgParent (Some (String "Zero\n", []))], None))]
                ,
                Some
                  (Else
                    ([Write (ArgParent (Some (String "Numero Negativo\n", []))],
                      None))]
                    ,
                    None)))
              ,
              None)))
    )
  )
```

Figura 3.11: Árvore sintática abstrata

### 3.3 Analisador Semântico

Algumas alterações foram feitas nos arquivos “lexico.mll”, “ast.ml” e “sintatico.mly” da análise sintática, como pode ser visto a seguir:

Programa 3.14: Analisador Semântico - lexico.mll

```
1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6   exception Erro of string
7
8   let incr_num_linha lexbuf =
```

```

9     let pos = lexbuf.lex_curr_p in
10     lexbuf.lex_curr_p <-
11         { pos with pos_lnum = pos.pos_lnum + 1;
12           pos_bol = pos.pos_cnum
13         }
14
15     let pos_atual lexbuf = lexbuf.lex_start_p
16
17 }
18
19 let digito = ['0' - '9']
20 let inteiro = '-'? digito+
21 let float = '-'? digito+ '.'? digito+
22
23 let letra = ['a' - 'z' 'A' - 'Z']
24 let identificador = letra ( letra | digito | '_' ) *
25
26 let brancos = [ ' ' '\t' ] +
27 let novalinha = '\r' | '\n' | "\r\n"
28
29 let comentario = "--" [ ^ ' ' ] [ ^ '\r' '\n' ] * | "--[" [ ^ ' ' ] [ ^ '\r' '\n' ] *
30
31 rule token =
32     parse
33     | brancos { token lexbuf }
34     | novalinha { incr_num_linha lexbuf; token lexbuf }
35     | comentario { token lexbuf }
36     | "--[" { comentario_bloco 0 lexbuf }
37     | '+' { MAIS (pos_atual lexbuf) }
38     | '-' { MENOS (pos_atual lexbuf) }
39     | '*' { MULT (pos_atual lexbuf) }
40     | '/' { DIV (pos_atual lexbuf) }
41     | '<' { MENOR (pos_atual lexbuf) }
42     | "<=" { MENORIGUAL (pos_atual lexbuf) }
43     | ">=" { MAIORIGUAL (pos_atual lexbuf) }
44     | "==" { IGUAL (pos_atual lexbuf) }
45     | "!=" { DIFER (pos_atual lexbuf) }
46     | '>' { MAIOR (pos_atual lexbuf) }
47     | " and " { ELOG (pos_atual lexbuf) }
48     | " or " { OULOG (pos_atual lexbuf) }
49     | '^' { CONCAT (pos_atual lexbuf) }
50     | '(' { APAR (pos_atual lexbuf) }
51     | ')' { FPAR (pos_atual lexbuf) }
52     | '[' { ACOL (pos_atual lexbuf) }
53     | ']' { FCOL (pos_atual lexbuf) }
54     | ',' { VIRG (pos_atual lexbuf) }
55     | ".." { PPTO (pos_atual lexbuf) }
56     | '.' { PTO (pos_atual lexbuf) }
57     | ':' { DPTOS (pos_atual lexbuf) }
58     (*| ';' { PTV (pos_atual lexbuf) } *)
59     | "=" { ATRIB (pos_atual lexbuf) }
60     | '"' { let buffer = Buffer.create 1 in
61              let str = leia_string buffer lexbuf in
62              STRING (str, pos_atual lexbuf) }
63     (*| "programa" { PROGRAMA (pos_atual lexbuf) } *)
64     | "function" { FUNCAO (pos_atual lexbuf) }
65     | "return" { RETORNE (pos_atual lexbuf) }
66     | "begin" { INICIO (pos_atual lexbuf) }

```

```

67 | "end"          { FIM (pos_atual lexbuf) }
68 | "inteiro"     { INTEIRO (pos_atual lexbuf) }
69 | "float"       { FLOAT (pos_atual lexbuf) }
70 | "string"      { CADEIA (pos_atual lexbuf) }
71 | "booleano"    { BOOLEANO (pos_atual lexbuf) }
72 | "arranjo"     { ARRANJO (pos_atual lexbuf) }
73 | "de"          { DE (pos_atual lexbuf) }
74 | "registro"    { REGISTRO (pos_atual lexbuf) }
75 | "do"          { DO (pos_atual lexbuf) }
76 | "while"       { WHILE (pos_atual lexbuf) }
77 | "for"         { FOR (pos_atual lexbuf) }
78 | "passo"       { PASSO (pos_atual lexbuf) }
79 | "end"         { FIM (pos_atual lexbuf) }
80 | "if"          { SE (pos_atual lexbuf) }
81 | "then"        { ENTAO (pos_atual lexbuf) }
82 | "else"        { SENAO (pos_atual lexbuf) }
83 | "=io.read()"  { ENTRADA (pos_atual lexbuf) }
84 | "= io.read()" { ENTRADA (pos_atual lexbuf) }
85 | "=tonumber(io.read())" { ENTRADA (pos_atual lexbuf) }
86 | "= tonumber(io.read())" { ENTRADA (pos_atual lexbuf) }
87 | "print"       { SAIDA (pos_atual lexbuf) }
88 | "io.write"     { SAIDA (pos_atual lexbuf) }
89 | "verdadeiro"  { BOOL (true, pos_atual lexbuf) }
90 | "falso"       { BOOL (false, pos_atual lexbuf) }
91 | identificador as x { ID (x, pos_atual lexbuf) }
92 | inteiro as n { INT (int_of_string n, pos_atual lexbuf) }
93 | float as n { PFLOAT (float_of_string n, pos_atual lexbuf) }
94 | _ { raise (Erro ("Caracter desconhecido: " ^ Lexing.lexeme lexbuf)) }
95 | eof { EOF }
96
97 and comentario_bloco n = parse
98   "]" { if n=0 then token lexbuf
99         else comentario_bloco (n-1) lexbuf }
100 | "--[" { comentario_bloco 0 lexbuf }
101 | novalinha { incr_num_linha lexbuf; comentario_bloco n lexbuf }
102 | _ { comentario_bloco n lexbuf }
103 | eof { failwith "Comentario nao fechado" }
104
105 and leia_string buffer = parse
106   '"' { Buffer.contents buffer }
107 | "\\t" { Buffer.add_char buffer '\t'; leia_string buffer lexbuf }
108 | "\\n" { Buffer.add_char buffer '\n'; leia_string buffer lexbuf }
109 | '\\ ' '"' { Buffer.add_char buffer '"'; leia_string buffer lexbuf }
110 | '\\ ' '\\ ' { Buffer.add_char buffer '\\'; leia_string buffer lexbuf }
111 | _ as c { Buffer.add_char buffer c; leia_string buffer lexbuf }
112 | eof { raise (Erro "A string não foi terminada") }

```

### Programa 3.15: Analisador Semântico - ast.ml

```

1 (* The type of the abstract syntax tree (AST). *)
2 open Lexing
3
4 type ident = string
5 type 'a pos = 'a * Lexing.position (* tipo e posição no arquivo fonte *)
6
7 type 'expr programa = Programa of declaracoes * ('expr funcoes) * ('expr
  comandos)
8 and declaracoes = declaracao list
9 and 'expr funcoes = ('expr funcao) list

```

```

10 and 'expr comandos = ('expr comando) list
11
12 and declaracao = DecVar of (ident pos) * tipo
13
14 and 'expr funcao = DecFun of ('expr decfn)
15
16 and 'expr decfn = {
17     fn_nome:      ident pos;
18     fn_tiporet: tipo;
19     fn_formais: (ident pos * tipo) list;
20     fn_locais: declaracoes;
21     fn_corpo:      'expr comandos
22 }
23
24 and tipo = TipoInt
25           | TipoString
26           | TipoFloat
27           | TipoBool
28           | TipoVoid
29           | TipoArranjo of tipo * (int pos) * (int pos)
30           | TipoRegistro of campos
31
32 and campos = campo list
33 and campo = ident pos * tipo
34
35 and 'expr comando =
36   | CmdAtrib of 'expr * 'expr
37   | CmdSe of 'expr * ('expr comandos) * ('expr comandos option)
38   | CmdEntrada of ('expr expressoes)
39   | CmdSaida of ('expr expressoes)
40   | CmdRetorno of 'expr option
41   | CmdChamada of 'expr
42   | While of 'expr * 'expr comandos
43   | For of 'expr * 'expr * 'expr * 'expr * 'expr comandos
44
45 and 'expr variaveis = ('expr variavel) list
46 and 'expr variavel =
47   | VarSimples of ident pos
48   | VarCampo of ('expr variavel) * (ident pos)
49   | VarElemento of ('expr variavel) * 'expr
50 and 'expr expressoes = 'expr list
51
52 and oper =
53   | Mais
54   | Menos
55   | MaiorIgual
56   | MenorIgual
57   | Mult
58   | Div
59   | Menor
60   | Igual
61   | Difer
62   | Maior
63   | E
64   | Ou
65   | Concat

```

```

1
2 %{
3 open Lexing
4 open Ast
5 open Sast
6 %}
7
8 %token <int * Lexing.position> INT
9 %token <float * Lexing.position> PFLOAT
10 %token <string * Lexing.position> ID
11 %token <string * Lexing.position> STRING
12 %token <bool * Lexing.position> BOOL
13 %token <Lexing.position> INICIO
14 %token <Lexing.position> FIM
15 %token <Lexing.position> FUNCAO
16 %token <Lexing.position> VIRG DPTOS PTO PPTO
17 %token <Lexing.position> ACOL FCOL
18 %token <Lexing.position> APAR FPAR
19 %token <Lexing.position> INTEIRO CADEIA BOOLEANO FLOAT
20 %token <Lexing.position> ARRANJO DE
21 %token <Lexing.position> REGISTRO
22 %token <Lexing.position> SE ENTAO SENAO
23 %token <Lexing.position> ENTRADA
24 %token <Lexing.position> SAIDA
25 %token <Lexing.position> ATRIB RETORNE
26 %token <Lexing.position> MAIS
27 %token <Lexing.position> MENOS
28 %token <Lexing.position> MULT
29 %token <Lexing.position> DIV
30 %token <Lexing.position> MENOR
31 %token <Lexing.position> MENORIGUAL
32 %token <Lexing.position> IGUAL
33 %token <Lexing.position> DIFER
34 %token <Lexing.position> MAIOR
35 %token <Lexing.position> MAIORIGUAL
36 %token <Lexing.position> ELOG
37 %token <Lexing.position> OULOG
38 %token <Lexing.position> CONCAT
39 %token <Lexing.position> WHILE
40 %token <Lexing.position> FOR
41 %token <Lexing.position> DO
42 %token <Lexing.position> PASSO
43 %token EOF
44
45 %left OULOG
46 %left ELOG
47 %left IGUAL DIFER
48 %left MAIOR MENOR MAIORIGUAL MENORIGUAL
49 %left CONCAT
50 %left MAIS MENOS
51 %left MULT DIV
52
53
54 %start <Sast.expressao Ast.programa> programa
55
56 %%
57
58 programa: ds = declaracao_de_variavel*
59           fs = declaracao_de_funcao*

```

```

60      INICIO
61      cs = comando*
62      FIM
63      EOF { Programa (List.flatten ds, fs, cs) }
64
65
66 declaracao_de_variavel:
67   ids = separated_nonempty_list(VIRG, ID) DPTOS t = tipo {
68       List.map (fun id -> DecVar (id,t)) ids }
69
70 declaracao_de_funcao:
71   FUNCAO nome = ID APAR formais = separated_list(VIRG, parametro) FPAR
72       DPTOS tret = tipo
73   ds = declaracao_de_variavel*
74   INICIO
75   cs = comando*
76   FIM {
77       DecFun {
78         fn_nome = nome;
79         fn_tiporet = tret ;
80         fn_formais = formais;
81         fn_locais = List.flatten ds;
82         fn_corpo = cs
83       }
84
85 parametro: nome = ID DPTOS t = tipo { (nome, t) }
86
87 tipo: t=tipo_simples { t }
88      | t=tipo_arranjo { t }
89      | t=tipo_registro { t }
90
91
92 tipo_simples: INTEIRO { TipoInt }
93              | FLOAT { TipoFloat }
94              | CADEIA { TipoString }
95              | BOOLEANO { TipoBool }
96
97
98 tipo_arranjo: ARRANJO ACOL lim=limites FCOL DE tp=tipo {
99             let (inicio, fim) = lim in
100             TipoArranjo (tp, inicio, fim)
101           }
102
103 tipo_registro: REGISTRO
104               campos=nonempty_list(id=ID DPTOS tp=tipo { (id,tp) } )
105               FIM REGISTRO { TipoRegistro campos }
106
107
108 limites: inicio=INT PPTO fim=INT { (inicio, fim) }
109
110 comando: c=comando_atribuicao { c }
111          | c=comando_se { c }
112          | c=comando_entrada { c }
113          | c=comando_saida { c }
114          | c=comando_chamada { c }
115          | c=comando_retorno { c }
116          | c=comando_enquanto {c}
117          | c=comando_para {c}

```



```

118
119 comando_atribuicao: esq=expressao ATRIB dir=expressao {
120     CmdAtrib (esq,dir)
121 }
122
123 comando_se: SE APAR teste=expressao FPAR ENTAO
124     entao=comando+
125     senao=option(SENÃO cs=comando+ {cs})
126     FIM {
127         CmdSe (teste, entao, senao)
128     }
129
130 comando_entrada: xs=separated_nonempty_list(VIRG, expressao) ENTRADA {
131     CmdEntrada xs
132 }
133
134 comando_saida: SAIDA APAR xs=separated_nonempty_list (VIRG, expressao)
135     FPAR {
136         CmdSaida xs
137     }
138
139 comando_enquanto:
140     | WHILE APAR e=expressao FPAR DO stm=comando* FIM { While (e,stm) }
141     ;
142
143 comando_para:
144     | FOR lv=expressao VIRG e1=expressao VIRG e2=expressao p=passo DO stm=
145         comando* FIM { For(lv, e1, e2, p, stm) }
146     ;
147
148 passo:
149     | PASSO i=INT {ExpInt i}
150     ;
151
152 comando_chamada: exp=chamada { CmdChamada exp }
153
154 comando_retorno: RETORNE e=expressao? { CmdRetorno e}
155
156 expressao:
157     | v=variavel { ExpVar v }
158     | i=INT { ExpInt i }
159     | f=PFLOAT { ExpFloat f }
160     | s=STRING { ExpString s }
161     | b=BOOL { ExpBool b }
162     | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }
163     | c = chamada { c }
164     | APAR e=expressao FPAR { e }
165
166 chamada : nome=ID APAR args=separated_list(VIRG, expressao) FPAR {
167     ExpChamada (nome, args)}
168
169 %inline oper:
170     | pos = MAIS { (Mais, pos) }
171     | pos = MENOS { (Menos, pos) }
172     | pos = MULT { (Mult, pos) }
173     | pos = DIV { (Div, pos) }
174     | pos = MENOR { (Menor, pos) }
175     | pos = MENORIGUAL { (MenorIgual, pos) }

```

```

175         | pos = IGUAL   { (Igual, pos) }
176         | pos = DIFER   { (Difer, pos) }
177         | pos = MAIOR    { (Maior, pos) }
178         | pos = MAIORIGUAL { (MaiorIgual, pos) }
179         | pos = ELOG      { (E, pos) }
180         | pos = OULOG     { (Ou, pos) }
181         | pos = CONCAT   { (Concat, pos) }
182
183 variavel:
184         | x=ID           { VarSimples x }
185         | v=variavel PTO x=ID { VarCampo (v,x) }
186         | v=variavel ACOL e=expressao FCOL { VarElemento (v,e) }

```

Consequentemente as mensagens de erro também foram atualizadas seguindo os passos indicados em 3.2.2.

### 3.3.1 Criando os arquivos semânticos

O seguinte arquivo foi criado com o nome de “semantico.ml”

Programa 3.17: Analisador Semântico - semantico.ml

```

1 module Amb = Ambiente
2 module A = Ast
3 module S = Sast
4 module T = Tast
5
6 let rec posicao exp = let open S in
7   match exp with
8   | ExpVar v -> (match v with
9     | A.VarSimples (_,pos) -> pos
10    | A.VarCampo (_, (_,pos)) -> pos
11    | A.VarElemento (_,exp2) -> posicao exp2
12   )
13   | ExpInt (_,pos) -> pos
14   | ExpString (_,pos) -> pos
15   | ExpFloat (_,pos) -> pos
16   | ExpBool (_,pos) -> pos
17   | ExpOp ((_,pos),_,_) -> pos
18   | ExpChamada ((_,pos), _) -> pos
19
20 type classe_op = Aritmetico | Relacional | Logico | Cadeia
21
22 let classifica op =
23   let open A in
24   match op with
25   | Ou
26   | E -> Logico
27   | Menor
28   | Maior
29   | MaiorIgual
30   | MenorIgual
31   | Igual
32   | Difer -> Relacional
33   | Mais
34   | Menos

```

```

35 | Mult
36 | Div -> Aritmetico
37 | Concat -> Cadeia
38
39 let msg_erro_pos pos msg =
40   let open Lexing in
41   let lin = pos.pos_lnum
42   and col = pos.pos_cnum - pos.pos_bol - 1 in
43   Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin col msg
44
45 let msg_erro nome msg =
46   let pos = snd nome in
47   msg_erro_pos pos msg
48
49 let nome_tipo t =
50   let open A in
51   match t with
52     TipoInt -> "inteiro"
53   | TipoFloat -> "float"
54   | TipoString -> "string"
55   | TipoBool -> "bool"
56   | TipoVoid -> "void"
57   | TipoArranjo (t,i,f) -> "arranjo"
58   | TipoRegistro cs -> "registro"
59
60 let mesmo_tipo pos msg tinf tdec =
61   if tinf <> tdec
62   then
63     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo tdec) in
64     failwith (msg_erro_pos pos msg)
65
66 let rec infere_exp amb exp =
67   match exp with
68     S.ExpInt n -> (T.ExpInt (fst n, A.TipoInt), A.TipoInt)
69   | S.ExpFloat n -> (T.ExpFloat (fst n, A.TipoFloat), A.TipoFloat)
70   | S.ExpString s -> (T.ExpString (fst s, A.TipoString), A.TipoString)
71   | S.ExpBool b -> (T.ExpBool (fst b, A.TipoBool), A.TipoBool)
72   | S.ExpVar v ->
73     (match v with
74       A.VarSimples nome ->
75         (* Tenta encontrar a definição da variável no escopo local, se não
76            *)
77         (* encontrar tenta novamente no escopo que engloba o atual.
78            Prossegue-se *)
79         (* assim até encontrar a definição em algum escopo englobante ou at
80            é *)
81         (* encontrar o escopo global. Se em algum lugar for encontrado,
82            *)
83         (* devolve-se a definição. Em caso contrário, devolve uma exceção
84            *)
85       let id = fst nome in
86       (try (match (Amb.busca amb id) with
87         | Amb.EntVar tipo -> (T.ExpVar (A.VarSimples nome, tipo),
88           tipo)
89         | Amb.EntFun _ ->
90           let msg = "nome de funcao usado como nome de variavel: "
91             ^ id in
92           failwith (msg_erro nome msg)

```

```

86         )
87         with Not_found ->
88             let msg = "A variavel " ^ id ^ " nao foi declarada" in
89                 failwith (msg_erro nome msg)
90     )
91     | _ -> failwith "infere_exp: não implementado"
92 )
93 | S.ExpOp (op, esq, dir) ->
94     let (esq, tesq) = infere_exp amb esq
95     and (dir, tdir) = infere_exp amb dir in
96
97     let verifica_aritmetico () =
98         (match tesq with
99             A.TipoInt ->
100                 let _ = mesmo_tipo (snd op)
101                     "O operando esquerdo eh do tipo %s mas o direito eh
102                     do tipo %s"
103                     tesq tdir
104                 in tesq (* O tipo da expressão aritmética como um todo *)
105
106             | t -> let msg = "um operador aritmetico nao pode ser usado com o
107                 tipo " ^
108                     (nome_tipo t)
109                 in failwith (msg_erro_pos (snd op) msg)
110         )
111
112     and verifica_relacional () =
113         (match tesq with
114             A.TipoInt
115             | A.TipoString ->
116                 let _ = mesmo_tipo (snd op)
117                     "O operando esquerdo eh do tipo %s mas o direito eh do
118                     tipo %s"
119                     tesq tdir
120                 in A.TipoBool (* O tipo da expressão relacional é sempre booleano
121                     *)
122
123             | t -> let msg = "um operador relacional nao pode ser usado com o
124                 tipo " ^
125                     (nome_tipo t)
126                 in failwith (msg_erro_pos (snd op) msg)
127         )
128
129     and verifica_logico () =
130         (match tesq with
131             A.TipoBool ->
132                 let _ = mesmo_tipo (snd op)
133                     "O operando esquerdo eh do tipo %s mas o direito eh do
134                     tipo %s"
135                     tesq tdir
136                 in A.TipoBool (* O tipo da expressão lógica é sempre booleano *)
137
138             | t -> let msg = "um operador logico nao pode ser usado com o tipo
139                 " ^
140                     (nome_tipo t)
141                 in failwith (msg_erro_pos (snd op) msg)
142         )
143
144     and verifica_cadeia () =
145         (match tesq with

```

```

138     A.TipoString ->
139     let _ = mesmo_tipo (snd op)
140         "O operando esquerdo eh do tipo %s mas o direito eh do
           tipo %s"
141         tesq tdir
142     in A.TipoString (* O tipo da expressão relacional é sempre string
           *)
143
144 | t -> let msg = "um operador relacional nao pode ser usado com o
           tipo " ^
145         (nome_tipo t)
146         in failwith (msg_erro_pos (snd op) msg)
147 )
148
149 in
150 let op = fst op in
151 let tinf = (match (classifica op) with
152     Aritmetico -> verifica_aritmetico ()
153     | Relacional -> verifica_relacional ()
154     | Logico -> verifica_logico ()
155     | Cadeia -> verifica_cadeia ()
156 )
157 in
158 (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
159
160 | S.ExpChamada (nome, args) ->
161     let rec verifica_parametros ags ps fs =
162         match (ags, ps, fs) with
163         (a::ags), (p::ps), (f::fs) ->
164             let _ = mesmo_tipo (posicao a)
165                 "O parametro eh do tipo %s mas deveria ser do tipo %s
                   " p f
166             in verifica_parametros ags ps fs
167         | [], [], [] -> ()
168         | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
169     in
170     let id = fst nome in
171     try
172         begin
173             let open Amb in
174
175             match (Amb.busca amb id) with
176             (* verifica se 'nome' está associada a uma função *)
177             Amb.EntFun {tipo_fn; formais} ->
178                 (* Infere o tipo de cada um dos argumentos *)
179                 let argst = List.map (infere_exp amb) args
180                 (* Obtem o tipo de cada parâmetro formal *)
181                 and tipos_formais = List.map snd formais in
182                 (* Verifica se o tipo de cada argumento confere com o tipo
                   declarado *)
183                 (* do parâmetro formal correspondente.
                   *)
184                 let _ = verifica_parametros args (List.map snd argst)
185                     tipos_formais
186                 in (T.ExpChamada (id, (List.map fst argst), tipo_fn), tipo_fn)
187         | Amb.EntVar _ -> (* Se estiver associada a uma variável, falhe
188                             *)
189             let msg = id ^ " eh uma variavel e nao uma funcao" in
190             failwith (msg_erro nome msg)

```

```

189     end
190     with Not_found ->
191         let msg = "Nao existe a funcao de nome " ^ id in
192             failwith (msg_erro nome msg)
193
194 let rec verifica_cmd amb tiporet cmd =
195     let open A in
196     match cmd with
197     | CmdRetorno exp ->
198         (match exp with
199         (* Se a função não retornar nada, verifica se ela foi declarada como
200          void *)
201         | None ->
202             let _ = mesmo_tipo (Lexing.dummy_pos)
203                           "O tipo retornado eh %s mas foi declarado como %s"
204                           TipoVoid tiporet
205             in CmdRetorno None
206         | Some e ->
207             (* Verifica se o tipo inferido para a expressão de retorno confere
208              com o *)
209             (* tipo declarado para a função. *)
210             let (e1,tinf) = infere_exp amb e in
211             let _ = mesmo_tipo (posicao e)
212                           "O tipo retornado eh %s mas foi declarado
213                           como %s"
214                           tinf tiporet
215             in CmdRetorno (Some e1)
216         )
217     | CmdSe (teste, entao, senao) ->
218         let (testel,tinf) = infere_exp amb teste in
219         (* O tipo inferido para a expressão 'teste' do condicional deve ser
220          booleano *)
221         let _ = mesmo_tipo (posicao teste)
222                           "O teste do if deveria ser do tipo %s e nao %s"
223                           TipoBool tinf in
224         (* Verifica a validade de cada comando do bloco 'então' *)
225         let entao1 = List.map (verifica_cmd amb tiporet) entao in
226         (* Verifica a validade de cada comando do bloco 'senão', se houver *)
227         let senao1 =
228             match senao with
229             | None -> None
230             | Some bloco -> Some (List.map (verifica_cmd amb tiporet) bloco)
231         in
232         CmdSe (testel, entao1, senao1)
233
234     | CmdAtrib (elem, exp) ->
235         (* Infere o tipo da expressão no lado direito da atribuição *)
236         let (exp, tdir) = infere_exp amb exp
237         (* Faz o mesmo para o lado esquerdo *)
238         and (elem1, tesq) = infere_exp amb elem in
239         (* Os dois tipos devem ser iguais *)
240         let _ = mesmo_tipo (posicao elem)
241                           "Atribuicao com tipos diferentes: %s = %s" tesq
242                           tdir
243         in CmdAtrib (elem1, exp)
244
245     | While (teste, corpo) ->
246         let (teste_tipo,tinf) = infere_exp amb teste in

```

```

242     let _ = mesmo_tipo (posicao teste)
243         "O teste do enquanto deveria ser do tipo %s e nao %s
244         "
245         TipoBool tinf in
246     let corpo_tipo = List.map (verifica_cmd amb tiporet) corpo in
247     While (teste_tipo, corpo_tipo)
248
249 | For (var, inicio, fim, avanco, corpo) ->
250     let (var_tipo, tinfv) = infere_exp amb var in
251     let (inicio_tipo, tinfi) = infere_exp amb inicio in
252     let (fim_tipo, tinff) = infere_exp amb fim in
253     let (avanco_tipo, tinfa) = infere_exp amb avanco in
254
255     let _ = mesmo_tipo (posicao var)
256         "A variável deveria ser do tipo %s e nao %s"
257         TipoInt tinfv in
258     let _ = mesmo_tipo (posicao inicio)
259         "O comando DE deveria ser do tipo %s e nao %s"
260         TipoInt tinfi in
261     let _ = mesmo_tipo (posicao fim)
262         "O comando ATE deveria ser do tipo %s e nao %s"
263         TipoInt tinff in
264     let _ = mesmo_tipo (posicao avanco)
265         "O comando PASSO deveria ser do tipo %s e nao %s"
266         TipoInt tinfa in
267
268     let corpo_tipo = List.map (verifica_cmd amb tiporet) corpo in
269     For (var_tipo, inicio_tipo, fim_tipo, avanco_tipo, corpo_tipo)
270
271 | CmdChamada exp ->
272     let (exp, tinf) = infere_exp amb exp in
273     CmdChamada exp
274
275 | CmdEntrada exps ->
276     (* Verifica o tipo de cada argumento da função 'entrada' *)
277     let exps = List.map (infere_exp amb) exps in
278     CmdEntrada (List.map fst exps)
279
280 | CmdSaida exps ->
281     (* Verifica o tipo de cada argumento da função 'saida' *)
282     let exps = List.map (infere_exp amb) exps in
283     CmdSaida (List.map fst exps)
284
285 and verifica_fun amb ast =
286     let open A in
287     match ast with
288     | A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
289         (* Estende o ambiente global, adicionando um ambiente local *)
290         let ambfn = Amb.novo_escopo amb in
291         (* Insere os parâmetros no novo ambiente *)
292         let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
293         let _ = List.iter insere_parametro fn_formais in
294         (* Insere as variáveis locais no novo ambiente *)
295         let insere_local = function
296             (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t in
297         let _ = List.iter insere_local fn_locais in
298         (* Verifica cada comando presente no corpo da função usando o novo
299             ambiente *)

```

```

298     let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporet) fn_corpo
        in
299     A.DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo =
        corpo_tipado}
300
301
302 let rec verifica_dup xs =
303     match xs with
304     [] -> []
305   | (nome,t)::xs ->
306     let id = fst nome in
307     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
308     then (id, t) :: verifica_dup xs
309     else let msg = "Parametro duplicado " ^ id in
310         failwith (msg_erro nome msg)
311
312 let insere_declaracao_var amb dec =
313     let open A in
314     match dec with
315     DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
316
317 let insere_declaracao_fun amb dec =
318     let open A in
319     match dec with
320     DecFun {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
321       (* Verifica se não há parâmetros duplicados *)
322       let formais = verifica_dup fn_formais in
323       let nome = fst fn_nome in
324       Amb.insere_fun amb nome formais fn_tiporet
325
326
327 (* Lista de cabeçalhos das funções pré definidas *)
328 let fn_predefs = let open A in [
329   ("entrada", [( "x", TipoInt); ( "y", TipoInt)], TipoVoid);
330   ("saida",   [( "x", TipoInt); ( "y", TipoInt)], TipoVoid)
331 ]
332
333 (* insere as funções pré definidas no ambiente global *)
334 let declara_predefinidas amb =
335   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
336
337 let semantico ast =
338   (* cria ambiente global inicialmente vazio *)
339   let amb_global = Amb.novo_amb [] in
340   let _ = declara_predefinidas amb_global in
341   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
342   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
343   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
344   (* Verificação de tipos nas funções *)
345   let decs_funs = List.map (verifica_fun amb_global) decs_funs in
346   (* Verificação de tipos na função principal *)
347   let corpo = List.map (verifica_cmd amb_global A.TipoVoid) corpo in
348   (A.Programa (decs_globais, decs_funs, corpo), amb_global)

```

Depois foi criado o “ambiente.ml”:

#### Programa 3.18: Analisador Semântico - ambiente.ml

```

1 module Tab = Tabsimb

```



```

2 module A = Ast
3
4 type entrada_fn = { tipo_fn: A.tipo;
5                     formais: (string * A.tipo) list;
6 }
7
8 type entrada = EntFun of entrada_fn
9               | EntVar of A.tipo
10
11 type t = {
12   ambv : entrada Tab.tabela
13 }
14
15 let novo_amb xs = { ambv = Tab.cria xs }
16
17 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
18
19 let busca amb ch = Tab.busca amb.ambv ch
20
21 let insere_local amb ch t =
22   Tab.insere amb.ambv ch (EntVar t)
23
24 let insere_param amb ch t =
25   Tab.insere amb.ambv ch (EntVar t)
26
27 let insere_fun amb nome params resultado =
28   let ef = EntFun { tipo_fn = resultado;
29                     formais = params }
30   in Tab.insere amb.ambv nome ef

```

Também foram criados o “sast.ml” e o “tast.ml”, mostrados a seguir:

#### Programa 3.19: Analisador Semântico - sast.ml

```

1 open Ast
2
3 type expressao =
4   | ExpVar of (expressao variavel)
5   | ExpInt of int pos
6   | ExpFloat of float pos
7   | ExpString of string pos
8   | ExpBool of bool pos
9   | ExpOp of oper pos * expressao * expressao
10  | ExpChamada of ident pos * (expressao expressoes)

```

#### Programa 3.20: Analisador Semântico - tast.ml

```

1 open Ast
2
3 type expressao =
4   ExpVar of (expressao variavel) * tipo
5   | ExpInt of int * tipo
6   | ExpFloat of float * tipo
7   | ExpString of string * tipo
8   | ExpVoid
9   | ExpBool of bool * tipo
10  | ExpOp of (oper * tipo) * (expressao * tipo) * (expressao * tipo)
11  | ExpChamada of ident * (expressao expressoes) * tipo

```

### 3.3.2 Compilando

Para compilar, primeiramente é necessário criar o arquivo “semanticoTest.ml”:

Programa 3.21: Analisador Semântico - semanticoTest.ml

```
1 open Printf
2 open Lexing
3
4 open Ast
5 open ErroSint
6 exception Erro_Sintatico of string
7
8 module S = MenhirLib.General (* Streams *)
9 module I = Sintatico.MenhirInterpreter
10
11 open Semantico
12
13
14
15 let posicao lexbuf =
16   let pos = lexbuf.lex_curr_p in
17   let lin = pos.pos_lnum
18   and col = pos.pos_cnum - pos.pos_bol - 1 in
19   sprintf "linha %d, coluna %d" lin col
20
21 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
22    checkpoint *)
23 let pilha checkpoint =
24   match checkpoint with
25   | I.HandlingError amb -> I.stack amb
26   | _ -> assert false (* Isso não pode acontecer *)
27
28 let estado checkpoint : int =
29   match Lazy.force (pilha checkpoint) with
30   | S.Nil -> (* O parser está no estado inicial *)
31     0
32   | S.Cons (I.Element (s, _, _, _), _) ->
33     I.number s
34
35 let sucesso v = Some v
36
37 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
38   =
39   let estado_atual = estado checkpoint in
40   let msg = message estado_atual in
41   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
42     (Lexing.lexeme_start lexbuf) msg))
43
44 let loop lexbuf resultado =
45   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
46   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
47
48 let parse_com_erro lexbuf =
49   try
50     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
51   with
```

```

52 | Lexico.Erro msg ->
53   printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
54   None
55 | Erro_Sintatico msg ->
56   printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
57   None
58
59 let parse s =
60   let lexbuf = Lexing.from_string s in
61   let ast = parse_com_erro lexbuf in
62   ast
63
64 let parse_arq nome =
65   let ic = open_in nome in
66   let lexbuf = Lexing.from_channel ic in
67   let ast = parse_com_erro lexbuf in
68   let _ = close_in ic in
69   ast
70
71 let verifica_tipos nome =
72   let ast = parse_arq nome in
73   match ast with
74   | Some (Some ast) -> semantico ast
75   | _ -> failwith "Nada a fazer!\n"

```

Agora deve-se digitar no terminal:

```

> ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
  menhirLib semanticoTest.byte

```

### 3.3.3 Testando

Será testado o seguinte arquivo:

#### Programa 3.22: micro10.lua

```

1 fat, numero : inteiro
2
3 function fatorial(n:inteiro) :inteiro
4   begin
5     if (n == 0) then
6       return 1
7     else
8       return n * fatorial(n - 1)
9     end
10  end
11
12 begin
13 print("Digite um numero:")
14
15 numero = tonumber(io.read())
16
17 fat = fatorial(numero)
18
19 print("O fatorial de", numero, "eh: ", fat)
20 end

```

Para testar deve-se entrar no ocaml assim:

```
> rlwrap ocaml
```

Depois digitar no terminal:

```
> verifica_tipos "micro10.lua";;
```

O retorno deste comando será a seguinte árvore tipada:

#### Programa 3.23: Analisador Semântico - retorno

```
1 (Programa
2   ([DecVar
3     (("fat",
4       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 0}),
5     TipoInt);
6   DecVar
7     (("numero",
8       {Lexing.pos_fname = ""; pos_lnum = 1; pos_bol = 0; pos_cnum = 5}),
9     TipoInt)],
10  [DecFun
11    {fn_nome =
12      ("fatorial",
13        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 23; pos_cnum = 32})
14    ;
15    fn_tiporet = TipoInt;
16    fn_formais =
17      [(("n",
18        {Lexing.pos_fname = ""; pos_lnum = 3; pos_bol = 23; pos_cnum =
19          41}),
20        TipoInt)];
21    fn_locais = [];
22    fn_corpo =
23      [CmdSe
24        (Tast.ExpOp ((Igual, TipoBool),
25          (Tast.ExpVar
26            (VarSimples
27              ("n",
28                {Lexing.pos_fname = ""; pos_lnum = 5; pos_bol = 68;
29                  pos_cnum = 73}),
30              TipoInt),
31              TipoInt),
32              (Tast.ExpInt (0, TipoInt), TipoInt))),
33        [CmdRetorno (Some (Tast.ExpInt (1, TipoInt)))]],
34        Some
35          [CmdRetorno
36            (Some
37              (Tast.ExpOp ((Mult, TipoInt),
38                (Tast.ExpVar
39                  (VarSimples
40                    ("n",
41                      {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol = 103;
42                        pos_cnum = 112}),
43                    TipoInt),
44                    TipoInt),
45                    (Tast.ExpChamada ("fatorial",
46                      [Tast.ExpOp ((Menos, TipoInt),
47                        (Tast.ExpVar
```

```

46         (VarSimples
47         ("n",
48         {Lexing.pos_fname = ""; pos_lnum = 8; pos_bol =
49         103;
50         pos_cnum = 125})),
51         TipoInt),
52         (Tast.ExpInt (1, TipoInt), TipoInt))),
53         TipoInt),
54         TipoInt)))]))]]],
55 [CmdSaida [Tast.ExpString ("Digite um numero:", TipoString)];
56 CmdEntrada
57 [Tast.ExpVar
58 (VarSimples
59 ("numero",
60 {Lexing.pos_fname = ""; pos_lnum = 15; pos_bol = 176;
61 pos_cnum = 176})),
62 TipoInt)];
63 CmdAtrib
64 (Tast.ExpVar
65 (VarSimples
66 ("fat",
67 {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 206;
68 pos_cnum = 206})),
69 TipoInt),
70 Tast.ExpChamada ("fatorial",
71 [Tast.ExpVar
72 (VarSimples
73 ("numero",
74 {Lexing.pos_fname = ""; pos_lnum = 17; pos_bol = 206;
75 pos_cnum = 221})),
76 TipoInt)],
77 TipoInt));
78 CmdSaida
79 [Tast.ExpString ("O fatorial de", TipoString);
80 Tast.ExpVar
81 (VarSimples
82 ("numero",
83 {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 230;
84 pos_cnum = 253})),
85 TipoInt);
86 Tast.ExpString ("eh: ", TipoString);
87 Tast.ExpVar
88 (VarSimples
89 ("fat",
90 {Lexing.pos_fname = ""; pos_lnum = 19; pos_bol = 230;
91 pos_cnum = 269})),
92 TipoInt)]),
93 <abstr>)

```

## 3.4 Interpretador

A ultima etapa foi fazer um interpretador para miniLua, seguindo os seguintes passos:

A tabela de simbolos com nome de “tabsimb.ml” foi criada da seguinte maneira:

### Programa 3.24: Interpretador - 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 encontrada"
21         | Some a -> atualiza a ch v
22
23 let rec busca amb ch =
24     try Hashtbl.find amb.tbl ch
25     with Not_found ->
26         (match amb.pai with
27             None -> raise Not_found
28             | Some a -> busca a ch)
29
30 let rec cria cvs =
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 }

```

Depois foi criado o ambiente do interpretador, que é diferente do ambiente semantico, com o nome de “ambInterp.ml”:

### Programa 3.25: Interpretador - 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.ident * A.tipo) list;
8     locais: A.declaracoes;
9     corpo: T.expressao A.comandos
10 }

```

```

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

```

O ultimo passo foi criar o interpretador, nomeado de “interprete.ml”:

#### Programa 3.26: Interpretador - 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.VarSimples (nome,_) -> (nome,tipo)
13     | _ -> failwith "obtem_nome_tipo_var: nao implementado"
14    )
15  | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
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 (i,_) -> i
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
35     | _ -> failwith "pega_bool: nao eh booleano"
36
37 type classe_op = Aritmetico | Relacional | Logico | Cadeia
38
39 let classifica op =
40     let open A in
41     match op with
42     | Ou
43     | E -> Logico
44     | Menor
45     | Maior
46     | MaiorIgual
47     | MenorIgual
48     | Igual
49     | Difer -> Relacional
50     | Mais
51     | Menos
52     | Mult
53     | Div -> Aritmetico
54     | Concat -> Cadeia
55
56
57 let rec interpreta_exp amb exp =
58     let open A in
59     let open T in
60     match exp with
61     | ExpVoid
62     | ExpInt _
63     | ExpFloat _
64     | ExpString _
65     | ExpBool _ -> exp
66     | ExpVar _ ->
67         let (id,tipo) = obtem_nome_tipo_var exp in
68         (* Tenta encontrar o valor da variável no escopo local, se não *)
69         (* encontrar, tenta novamente no escopo que engloba o atual. Prossegue *)
70         (* assim até encontrar o valor em algum escopo englobante ou até *)
71         (* encontrar o escopo global. Se em algum lugar for encontrado, *)
72         (* devolve-se o valor. Em caso contrário, devolve uma exceção *)
73         (match (Amb.busca amb id) with
74          | Amb.EntVar (tipo, v) ->
75              (match v with
76               | None -> failwith ("variável nao inicializada: " ^ id)
77               | Some valor -> valor
78              )
79          | _ -> failwith "interpreta_exp: expvar"
80         )
81     | ExpOp ((op,top), (esq, tesq), (dir,tdir)) ->
82         let vesq = interpreta_exp amb esq

```



```

83     and vdir = interpreta_exp amb dir in
84
85     let interpreta_aritmetico () =
86         (match tesq with
87         | TipoInt ->
88             (match op with
89             | Mais -> ExpInt (pega_int vesq + pega_int vdir, top)
90             | Menos -> ExpInt (pega_int vesq - pega_int vdir, top)
91             | Mult -> ExpInt (pega_int vesq * pega_int vdir, top)
92             | Div -> ExpInt (pega_int vesq / pega_int vdir, top)
93             | _ -> failwith "interpreta_aritmetico"
94             )
95         | TipoFloat ->
96             (match op with
97             | Mais -> ExpFloat (pega_float vesq +. pega_float vdir, top)
98             | Menos -> ExpFloat (pega_float vesq -. pega_float vdir, top)
99             | Mult -> ExpFloat (pega_float vesq *. pega_float vdir, top)
100            | Div -> ExpFloat (pega_float vesq /. pega_float vdir, top)
101            )
102            | _ -> failwith "interpreta_aritmetico"
103        )
104    )
105
106    and interpreta_relacional () =
107        (match tesq with
108        | TipoInt ->
109            (match op with
110            | Menor -> ExpBool (pega_int vesq < pega_int vdir, top)
111            | Maior -> ExpBool (pega_int vesq > pega_int vdir, top)
112            | MenorIgual -> ExpBool (pega_int vesq <= pega_int vdir, top)
113            | MaiorIgual -> ExpBool (pega_int vesq >= pega_int vdir, top)
114            | Igual -> ExpBool (pega_int vesq == pega_int vdir, top)
115            | Difer -> ExpBool (pega_int vesq != pega_int vdir, top)
116            | _ -> failwith "interpreta_relacional"
117            )
118        | TipoFloat ->
119            (match op with
120            | Menor -> ExpBool (pega_float vesq < pega_float vdir, top)
121            | Maior -> ExpBool (pega_float vesq > pega_float vdir, top)
122            | MenorIgual -> ExpBool (pega_float vesq <= pega_float vdir, top)
123            )
124            | MaiorIgual -> ExpBool (pega_float vesq >= pega_float vdir,
125            top)
126            | Igual -> ExpBool (pega_float vesq == pega_float vdir, top)
127            | Difer -> ExpBool (pega_float vesq != pega_float vdir, top)
128            | _ -> failwith "interpreta_relacional"
129        )
130        | TipoString ->
131            (match op with
132            | Menor -> ExpBool (pega_string vesq < pega_string vdir, top)
133            | Maior -> ExpBool (pega_string vesq > pega_string vdir, top)
134            | MenorIgual -> ExpBool (pega_int vesq <= pega_int vdir, top)
135            | MaiorIgual -> ExpBool (pega_int vesq >= pega_int vdir, top)
136            | Igual -> ExpBool (pega_string vesq = pega_string vdir, top)
137            | Difer -> ExpBool (pega_string vesq != pega_string vdir, top)
138            | _ -> failwith "interpreta_relacional"
139        )
140        | TipoBool ->

```

```

139     (match op with
140     | Menor -> ExpBool (pega_bool vesq < pega_bool vdir, top)
141     | Maior -> ExpBool (pega_bool vesq > pega_bool vdir, top)
142     | MenorIgual -> ExpBool (pega_int vesq <= pega_int vdir, top)
143     | MaiorIgual -> ExpBool (pega_int vesq >= pega_int vdir, top)
144     | Igual -> ExpBool (pega_bool vesq == pega_bool vdir, top)
145     | Difer -> ExpBool (pega_bool vesq != pega_bool vdir, top)
146     | _ -> failwith "interpreta_relacional"
147     )
148 | _ -> failwith "interpreta_relacional"
149 )
150
151 and interpreta_logico () =
152     (match tesq with
153     | TipoBool ->
154         (match op with
155         | Ou -> ExpBool (pega_bool vesq || pega_bool vdir, top)
156         | E -> ExpBool (pega_bool vesq && pega_bool vdir, top)
157         | _ -> failwith "interpreta_logico"
158         )
159     | _ -> failwith "interpreta_logico"
160     )
161 and interpreta_cadeia () =
162     (match tesq with
163     | TipoString ->
164         (match op with
165         | Concat -> ExpString (pega_string vesq ^ pega_string vdir, top)
166         | _ -> failwith "interpreta_cadeia"
167         )
168     | _ -> failwith "interpreta_cadeia"
169     )
170
171 in
172 let valor = (match (classifica op) with
173     Aritmetico -> interpreta_aritmetico ()
174     | Relacional -> interpreta_relacional ()
175     | Logico -> interpreta_logico ()
176     | Cadeia -> interpreta_cadeia ()
177     )
178 in
179     valor
180
181 | ExpChamada (id, args, tipo) ->
182     let open Amb in
183     ( match (Amb.busca amb id) with
184     | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
185         (* Interpreta cada um dos argumentos *)
186         let vargs = List.map (interpreta_exp amb) args in
187         (* Associa os argumentos aos parâmetros formais *)
188         let vformais = List.map2 (fun (n,t) v -> (n, t, Some v))
189             formais vargs
190         in interpreta_fun amb id vformais locais corpo
191     | _ -> failwith "interpreta_exp: expchamada"
192     )
193
194 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo =
195     let open A in
196     (* Estende o ambiente global, adicionando um ambiente local *)
197     let ambfn = Amb.novo_escopo amb in

```

```

197   let insere_local d =
198     match d with
199       (DecVar (v,t)) -> Amb.insere_local ambfn (fst v) t None
200   in
201   (* Associa os argumentos aos parâmetros e insere no novo ambiente *)
202   let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v in
203   let _ = List.iter insere_parametro fn_formais in
204   (* Insere as variáveis locais no novo ambiente *)
205   let _ = List.iter insere_local fn_locais in
206   (* Interpreta cada comando presente no corpo da função usando o novo
207     ambiente *)
208   try
209     let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.ExpVoid
210   with
211     Valor_de_retorno expret -> expret
212
213 and interpreta_cmd amb cmd =
214   let open A in
215   let open T in
216   match cmd with
217     CmdRetorno exp ->
218       (* Levantar uma exceção foi necessária pois, pela semântica do comando
219         de
220         retorno, sempre que ele for encontrado em uma função, a computação
221         deve parar retornando o valor indicado, sem realizar os demais
222         comandos.
223       *)
224       (match exp with
225         (* Se a função não retornar nada, então retorne ExpVoid *)
226         None -> raise (Valor_de_retorno ExpVoid)
227         | Some e ->
228           (* Avalia a expressão e retorne o resultado *)
229           let e1 = interpreta_exp amb e in
230           raise (Valor_de_retorno e1)
231       )
232     | CmdSe (teste, entao, senao) ->
233       let testel = interpreta_exp amb teste in
234       (match testel with
235         ExpBool (true,_) ->
236           (* Interpreta cada comando do bloco 'então' *)
237           List.iter (interpreta_cmd amb) entao
238         | _ ->
239           (* Interpreta cada comando do bloco 'senão', se houver *)
240           (match senao with
241             None -> ()
242             | Some bloco -> List.iter (interpreta_cmd amb) bloco
243           )
244       )
245     | CmdAtrib (elem, exp) ->
246       (* Interpreta o lado direito da atribuição *)
247       let exp = interpreta_exp amb exp in
248       (* Faz o mesmo para o lado esquerdo *)
249       and (elem1, tipo) = obter_nome_tipo_var elem in
250       Amb.atualiza_var amb elem1 tipo (Some exp)
251
252   | While (teste, corpo) ->
253     let rec laco teste corpo =

```

```

254     let condicao = interpreta_exp amb teste in
255     (match condicao with
256     | ExpBool (true,_) ->
257         (* Interpreta cada comando do bloco 'então' *)
258         let _ = List.iter (interpreta_cmd amb) corpo in
259         laco teste corpo
260     | _ -> ())
261     in laco teste corpo
262
263 | For (var, inicio, fim, avanco, corpo) ->
264     let (elem1, tipo) = obtem_nome_tipo_var var in
265     let rec executa_para amb inicio fim avanco corpo elem1 tipo =
266         if (inicio) <= (fim)
267         then begin
268             List.iter (interpreta_cmd amb) corpo;
269
270             Amb.atualiza_var amb elem1 tipo (Some (ExpInt ((inicio +
271                 avanco), TipoInt) ) );
272
273             executa_para amb (inicio + avanco) fim avanco corpo elem1
274                 tipo;
275         end in
276         executa_para amb (pega_int inicio) (pega_int fim) (pega_int avanco)
277             corpo elem1 tipo
278
279
280 | CmdChamada exp -> ignore( interpreta_exp amb exp)
281
282 | CmdEntrada exps ->
283     (* Obtem os nomes e os tipos de cada um dos argumentos *)
284     let nts = List.map (obtem_nome_tipo_var) exps in
285     let leia_var (nome,tipo) =
286         let valor =
287             (match tipo with
288             | A.TipoInt -> T.ExpInt (read_int (), tipo)
289             | A.TipoFloat -> T.ExpFloat (read_float (), tipo)
290             | A.TipoString -> T.ExpString (read_line (), tipo)
291             | _ -> failwith "leia_var: nao implementado"
292             )
293         in Amb.atualiza_var amb nome tipo (Some valor)
294     in
295     (* Lê o valor para cada argumento e atualiza o ambiente *)
296     List.iter leia_var nts
297
298
299 | CmdSaida exps ->
300     (* Interpreta cada argumento da função 'saida' *)
301     let exps = List.map (interpreta_exp amb) exps in
302     let imprima exp =
303         (match exp with
304         | T.ExpInt (n,_) -> let _ = print_int n in print_string " "
305         | T.ExpFloat (n,_) -> let _ = print_float n in print_string "
306             "
307         | T.ExpString (s,_) -> let _ = print_string s in print_string " "
308         | T.ExpBool (b,_) ->
309             let _ = print_string (if b then "true" else "false")
310             in print_string " "
311         | _ -> failwith "imprima: nao implementado"
312         )
313     in

```

```

309     in
310     let _ = List.iter imprima exps in
311     print_newline ()
312
313 let insere_declaracao_var amb dec =
314     match dec with
315     | A.DecVar (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
316     | None
317
318 let insere_declaracao_fun amb dec =
319     let open A in
320     match dec with
321     | DecFun {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
322         let nome = fst fn_nome in
323         let formais = List.map (fun (n,t) -> ((fst n), t)) fn_formais in
324         Amb.insere_fun amb nome formais fn_locais fn_tiporet fn_corpo
325
326 (* Lista de cabeçalhos das funções pré definidas *)
327 let fn_predefs = let open A in [
328     ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
329     ("saida",    [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
330 ]
331
332 (* insere as funções pré definidas no ambiente global *)
333 let declara_predefinidas amb =
334     List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps [] tr c)
335             fn_predefs
336
337 let interprete ast =
338     (* cria ambiente global inicialmente vazio *)
339     let amb_global = Amb.novo_amb [] in
340     let _ = declara_predefinidas amb_global in
341     let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
342     let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
343     let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
344     (* Interpreta a função principal *)
345     let resultado = List.iter (interpreta_cmd amb_global) corpo in
346     resultado

```

### 3.4.1 Compilando

Para compilar, primeiramente é necessário criar o arquivo “interpreteTeste.ml”:

Programa 3.27: Interpretador - interpreteTeste.ml

```

1 open Printf
2 open Lexing
3
4 open Ast
5 exception Erro_Sintatico of string
6
7 module S = MenhirLib.General (* Streams *)
8 module I = Sintatico.MenhirInterpreter
9
10 open Semantico
11

```

```
12 let message =
13     fun s ->
14         match s with
15         | 0 ->
16             "Inicio invalido.\n"
17         | 49 ->
18             "Esperava um \"(\".\n"
19         | 50 ->
20             "Esperava uma expressao valida.\n"
21         | 99 ->
22             "Esperava um \")\" ou um operador.\n"
23         | 100 ->
24             "Esperava a palavra reservada \"do\".\n"
25         | 101 ->
26             "Esperava um comando valido.\n"
27         | 156 ->
28             "Esperava um comando valido.\n"
29         | 128 ->
30             "Esperava um operador ou \"=\".\n"
31         | 97 ->
32             "Esperava uma expressao.\n"
33         | 124 ->
34             "Esperava uma entrada valida.\n"
35         | 63 ->
36             "Esperava uma expressao valida.\n"
37         | 64 ->
38             "Esperava um operador.\n"
39         | 65 ->
40             "Esperava uma expressao valida.\n"
41         | 68 ->
42             "Esperava uma expressao valida.\n"
43         | 69 ->
44             "Esperava um operador.\n"
45         | 72 ->
46             "Esperava uma expressao valida.\n"
47         | 73 ->
48             "Esperava um operador.\n"
49         | 78 ->
50             "Esperava uma expressao valida.\n"
51         | 79 ->
52             "Esperava um operador.\n"
53         | 74 ->
54             "Esperava uma expressao valida.\n"
55         | 75 ->
56             "Esperava um operador.\n"
57         | 80 ->
58             "Esperava uma expressao valida.\n"
59         | 81 ->
60             "Esperava um operador.\n"
61         | 82 ->
62             "Esperava uma expressao valida.\n"
63         | 83 ->
64             "Esperava um operador.\n"
65         | 84 ->
66             "Esperava uma expressao valida.\n"
67         | 85 ->
68             "Esperava um operador.\n"
69         | 86 ->
70             "Esperava uma expressao valida.\n"
```

```
71 | 87 ->
72 |     "Esperava um operador.\n"
73 | 70 ->
74 |     "Esperava uma expressao valida.\n"
75 | 88 ->
76 |     "Esperava uma expressao valida.\n"
77 | 89 ->
78 |     "Esperava um operador.\n"
79 | 76 ->
80 |     "Esperava uma expressao valida.\n"
81 | 77 ->
82 |     "Esperava um operador.\n"
83 | 129 ->
84 |     "Esperava uma expressao valida.\n"
85 | 130 ->
86 |     "Esperava um operador.\n"
87 | 102 ->
88 |     "Esperava um \"(\".\n"
89 | 103 ->
90 |     "Esperava uma expressao valida.\n"
91 | 104 ->
92 |     "Esperava um \")\" ou um operador.\n"
93 | 105 ->
94 |     "Esperava a palavra reservada \"then\".\n"
95 | 106 ->
96 |     "Esperava um comando valido.\n"
97 | 145 ->
98 |     "Esperava a palavra reservada \"end\" ou \"else\".\n"
99 | 143 ->
100 |     "Esperava um comando valido.\n"
101 | 147 ->
102 |     "Esperava a palavra reservada \"end\".\n"
103 | 107 ->
104 |     "Esperava um \"(\".\n"
105 | 108 ->
106 |     "Esperava uma expressao valida.\n"
107 | 109 ->
108 |     "Esperava um \")\".\n"
109 | 111 ->
110 |     "Esperava uma expressao valida.\n"
111 | 113 ->
112 |     "Esperava uma expressao valida ou um operador.\n"
113 | 139 ->
114 |     "Esperava um comando valido.\n"
115 | 54 ->
116 |     "Esperava \"(\" ou \":\" ou \",\".\n"
117 | 59 ->
118 |     "Esperava um identificador.\n"
119 | 58 ->
120 |     "Esperava \"[\" ou um \".\".\n"
121 | 55 ->
122 |     "Esperava uma expressao valida.\n"
123 | 96 ->
124 |     "Esperava um \")\" ou um operador ou \",\".\n"
125 | 94 ->
126 |     "Esperava um \")\".\n"
127 | 141 ->
128 |     "Esperava uma chamada valida.\n"
129 | 61 ->
```

```
130     "Esperava uma expressao valida.\n"
131 | 62 ->
132     "Esperava \"]\" ou um operador.\n"
133 | 114 ->
134     "Esperava uma expressao valida.\n"
135 | 115 ->
136     "Esperava \",\" ou um operador.\n"
137 | 116 ->
138     "Esperava uma expressao valida.\n"
139 | 117 ->
140     "Esperava \",\" ou um operador.\n"
141 | 118 ->
142     "Esperava uma expressao valida.\n"
143 | 119 ->
144     "Esperava um passo ou um operador. \n"
145 | 120 ->
146     "Esperava um inteiro.\n"
147 | 122 ->
148     "Esperava a palavra reservada \"do\".\n"
149 | 123 ->
150     "Esperava um bloco de comando apos o \"do\".\n"
151 | 158 ->
152     "Esperava o fim do arquivo.\n"
153 | 57 ->
154     "Esperava uma expressao valida.\n"
155 | 91 ->
156     "Esperava um operador ou \")\".\n"
157 | 1 ->
158     "Esperava \":\" ou \",\".\n"
159 | 2 ->
160     "Esperava um identificador.\n"
161 | 5 ->
162     "Esperava um tipo.\n"
163 | 6 ->
164     "Esperava um comando valido depois de \"registro\".\n"
165 | 7 ->
166     "Esperava \":\" \n"
167 | 8 ->
168     "Esperava um tipo.\n"
169 | 25 ->
170     "Esperava a palavra reservada \"end\".\n"
171 | 28 ->
172     "Esperava um registro valido.\n"
173 | 153 ->
174     "Esperava um declaracao de variavel valida.\n"
175 | 13 ->
176     "Esperava um \"[\".\n"
177 | 14 ->
178     "Esperava um limite valido.\n"
179 | 15 ->
180     "Esperava \"..\" \n"
181 | 16 ->
182     "Esperava um inteiro.\n"
183 | 18 ->
184     "Esperava um \"]\".\n"
185 | 19 ->
186     "Esperava a palavra reservada \"de\".\n"
187 | 20 ->
188     "Esperava o tipo do arranjo.\n"
```



```

189 | 33 ->
190 |     "Esperava o nome da funcao.\n"
191 | 34 ->
192 |     "Esperava um \"(\".\n"
193 | 35 ->
194 |     "Esperava um parametro valido.\n"
195 | 36 ->
196 |     "Esperava um \":\".\n"
197 | 37 ->
198 |     "Esperava o tipo da variavel.\n"
199 | 40 ->
200 |     "Esperava um \")\" ou \",\".\n"
201 | 41 ->
202 |     "Esperava um parametro valido.\n"
203 | 44 ->
204 |     "Esperava \":\" antes do tipo de retorno.\n"
205 | 45 ->
206 |     "Esperava o tipo de retorno da funcao.\n"
207 | 46 ->
208 |     "Esperava a palavra reservada \"begin\" antes de um comndo.\n"
209 | 48 ->
210 |     "Esperava um comando valido.\n"
211 | 47 ->
212 |     "Esperava a palavra reservada \"begin\".\n"
213 | 160 ->
214 |     "Erro na declaracao de funcao.\n"
215 | _ ->
216 |     raise Not_found
217
218 let posicao lexbuf =
219     let pos = lexbuf.lex_curr_p in
220     let lin = pos.pos_lnum
221     and col = pos.pos_cnum - pos.pos_bol - 1 in
222     sprintf "linha %d, coluna %d" lin col
223
224 (* [pilha checkpoint] extrai a pilha do autômato LR(1) contida em
225    checkpoint *)
226
227 let pilha checkpoint =
228     match checkpoint with
229     | I.HandlingError amb -> I.stack amb
230     | _ -> assert false (* Isso não pode acontecer *)
231
232 let estado checkpoint : int =
233     match Lazy.force (pilha checkpoint) with
234     | S.Nil -> (* O parser está no estado inicial *)
235         0
236     | S.Cons (I.Element (s, _, _, _), _) ->
237         I.number s
238
239 let sucesso v = Some v
240
241 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
242     =
243     let estado_atual = estado checkpoint in
244     let msg = message estado_atual in
245     raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
246         (Lexing.lexeme_start lexbuf) msg))

```

```

246 let loop lexbuf resultado =
247   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.token lexbuf in
248   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
249
250
251 let parse_com_erro lexbuf =
252   try
253     Some (loop lexbuf (Sintatico.Incremental.programa lexbuf.lex_curr_p))
254   with
255   | Lexico.Erro msg ->
256     printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
257     None
258   | Erro_Sintatico msg ->
259     printf "Erro sintático na %s %s\n" (posicao lexbuf) msg;
260     None
261
262 let parse s =
263   let lexbuf = Lexing.from_string s in
264   let ast = parse_com_erro lexbuf in
265   ast
266
267 let parse_arq nome =
268   let ic = open_in nome in
269   let lexbuf = Lexing.from_channel ic in
270   let ast = parse_com_erro lexbuf in
271   let _ = close_in ic in
272   ast
273
274 let verifica_tipos nome =
275   let ast = parse_arq nome in
276   match ast with
277   | Some (Some ast) -> semantico ast
278   | _ -> failwith "Nada a fazer!\n"
279
280
281 let interprete nome =
282   let tast,amb = verifica_tipos nome in
283   Interprete.interprete tast

```

Agora deve-se digitar no terminal:

```

> ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -package
  menhirLib interpreteTeste.byte

```

### 3.4.2 Testando

Será testado o seguinte arquivo:

Programa 3.28: micro10.lua

```

1 fat, numero : inteiro
2
3 function fatorial(n:inteiro) :inteiro
4   begin
5     if (n == 0) then
6       return 1

```

```

7  else
8      return n * fatorial(n - 1)
9  end
10 end
11
12 begin
13 print("Digite um numero:")
14
15 numero = tonumber(io.read())
16
17 fat = fatorial(numero)
18
19 print("O fatorial de", numero, "eh: ", fat)
20 end

```

Para testar deve-se entrar no ocaml assim:

```
> rlwrap ocaml
```

Depois digitar no terminal:

```
> interprete "micro10.lua";;
```

O retorno deste comando será:

```

Digite um numero:
6
O fatorial de 6 eh: 720
- : unit = ()

```

**Figura 3.12:** *Interpretador: Fatorial*