

Construção de um compilador de Lua para Parrot Virtual Machine usando Objective Caml

Guilherme Pacheco de Oliveira

`guilherme.061@gmail.com`

Faculdade de Computação
Universidade Federal de Uberlândia

13 de setembro de 2016

Lista de Figuras

2.1	Instalando e testando LUA	8
2.2	Instalando e testando OCaml	9
2.3	Instalando e testando Parrot	10

Lista de Tabelas

Lista de Listagens

2.1	Output Simples em Parrot Assembly Language	10
2.2	Output Simples em Parrot Intermediate Representation	10
3.1	Programa nano 01 em Ruby	12
3.2	Programa nano 01 em PIR	13
4.1	Programa nano 01 em Lua	14
4.2	Programa nano 01 em PASM	14
4.3	Programa nano 02 em Lua	14
4.4	Programa nano 02 em PASM	14
4.5	Programa nano 03 em Lua	15
4.6	Programa nano 03 em PASM	15
4.7	Programa nano 04 em Lua	15
4.8	Programa nano 04 em PASM	15
4.9	Programa nano 05 em Lua	15
4.10	Programa nano 05 em PASM	15
4.11	Programa nano 06 em Lua	15
4.12	Programa nano 06 em PASM	16
4.13	Programa nano 07 em Lua	16
4.14	Programa nano 07 em PASM	16
4.15	Programa nano 08 em Lua	16
4.16	Programa nano 08 em PASM	17
4.17	Programa nano 09 em Lua	17
4.18	Programa nano 09 em PASM	17
4.19	Programa nano 10 em Lua	18
4.20	Programa nano 10 em PASM	18
4.21	Programa nano 11 em Lua	18
4.22	Programa nano 11 em PASM	19
4.23	Programa nano 12 em Lua	19
4.24	Programa nano 12 em PASM	19
4.25	Programa micro 01 em Lua	20
4.26	Programa Micro 01 em PASM	20
4.27	Programa micro 02 em Lua	21
4.28	Programa Micro 02 em PASM	21
4.29	Programa micro 03 em Lua	22
4.30	Programa Micro 03 em PASM	22
4.31	Programa micro 04 em Lua	23
4.32	Programa Micro 04 em PASM	23
4.33	Programa micro 05 em Lua	24
4.34	Programa Micro 05 em PASM	25
4.35	Programa micro 06 em Lua	26
4.36	Programa Micro 06 em PASM	26

4.37 Programa micro 07 em Lua	27
4.38 Programa Micro 07 em PASM	28
4.39 Programa micro 08 em Lua	28
4.40 Programa Micro 08 em PASM	29
4.41 Programa micro 09 em Lua	29
4.42 Programa Micro 09 em PASM	30
4.43 Programa micro 10 em Lua	31
4.44 Programa Micro 10 em PASM	31
4.45 Programa micro 11 em Lua	32
4.46 Programa Micro 11 em PASM	32
5.1 Automato reconhecedor da linguagem descrita	38

Sumário

Lista de Figuras	2
Lista de Tabelas	3
1 Introdução	7
2 Instalação dos componentes	8
2.1 Homebrew	8
2.2 Lua	8
2.2.1 Instalação e Teste	8
2.2.2 Informações sobre a linguagem Lua	9
2.3 Ocaml	9
2.3.1 Instalação e Teste	9
2.3.2 Informações sobre a linguagem OCaml	9
2.4 Parrot Virtual Machine	9
2.4.1 Instalação e Teste	9
2.4.2 Informações sobre a Parrot Virtual Machine	10
2.4.3 Parrot Assembly Language (PASM)	11
2.4.4 Parrot Intermediate Representation (PIR)	11
3 Compilação de Código Ruby para Parrot Intermediate Representation (PIR)	12
4 Códigos LUA e Parrot Assembly (PASM)	14
4.0.1 Nano Programas	14
4.0.2 Micro Programas	20
5 Analisador Léxico	34
5.1 Análise Léxica	34
5.2 Analisador Manual	34
5.2.1 Linguagem a ser interpretada	34
5.2.2 Autômato reconhecedor da linguagem	35
5.2.3 Implementação	37
5.3 Analisador com Ocamllexer	41
5.3.1 Convenções Léxicas da Linguagem Lua	41
5.3.2 Implementação	42
5.3.3 Testes	42
6 Referências	43

Capítulo 1

Introdução

Este documento foi escrito para documentar o processo de instalação de todas as ferramentas necessárias para a construção de um compilador da Linguagem Lua para a máquina virtual Parrot, utilizando a linguagem Ocaml para fazer a implementação.

Um segundo objetivo é mostrar uma série de programas simples na linguagem Lua e sua versão na linguagem PASM, que é a linguagem assembly utilizada pela Parrot, afim de estabelecer um guia sobre a saída dos programas que passarão pelo compilador.

Outro objetivo é adquirir conhecimento sobre a linguagem Lua, ter um contato inicial com OCaml e conhecer como funciona a máquina virtual Parrot, suas linguagens de Assembly e bytecode e de compiladores já existentes

O Sistema Operacional utilizado é OS X El Capitan 10.11.6

Capítulo 2

Instalação dos componentes

2.1 Homebrew

Homebrew é um gerenciador de pacotes para Mac OS X, escrito em Ruby, e é responsável por instalar pacotes nos diretórios adequados e fazer adequadamente a configuração desses pacotes, instalá-lo facilita todo o processo de instalação dos componentes necessários.

Para instalar o homebrew basta digitar no terminal:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2.2 Lua

2.2.1 Instalação e Teste

Para instalar Lua através do homebrew, basta digitar no terminal:

```
$ brew install lua
```

Resultado:

Figura 2.1: *Instalando e testando LUA*

```
oliveira@lua oliveira$ brew install lua
=> Downloading https://homebrew.bintray.com/bottles/lua-5.2.4.3.el_capitan.bott
##### 100.0%
=> Pouring Lua-5.2.4.3.el_capitan.bottle.tar.gz
=> Caveats
Please be aware due to the way Luarocks is designed any binaries installed
via Luarocks-5.2 AND 5.1 will overwrite each other in /usr/local/bin.

This is, for now, unavoidable. If this is troublesome for you, you can build
rocks with the --tree= command to a special, non-conflicting location and
then add that to your $PATH.
=> Summary
[0] /usr/local/Cellar/lua/5.2.4.3: 143 files, 697.3K
oliveira@lua oliveira$ lua
Lua 5.2.4 Copyright (C) 1994-2015 Lua.org, PUC-Rio
> print("Hello World")
Hello World
> ^D
oliveira@lua oliveira$
```


2.2.2 Informações sobre a linguagem Lua

A principal referência para Lua é a documentação em seu site oficial [1]. Lua é uma linguagem de programação de extensão, projetada para dar suporte à outras linguagens de programação procedimental e planejada para ser usada como uma linguagem de script leve e facilmente embarcável, é implementada em C.

2.3 Ocaml

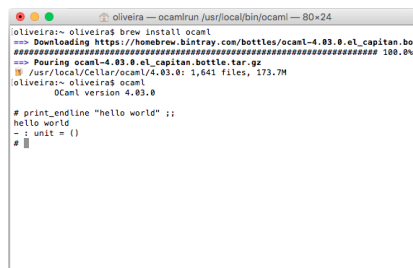
2.3.1 Instalação e Teste

Novamente através do homebrew, basta digitar:

```
$ brew install ocaml
```

Resultado:

Figura 2.2: *Instalando e testando OCaml*



```

oliveira~ oliveira$ brew install ocaml
=> Downloading https://homebrew.bintray.com/bottles/ocaml-4.03.0.el_capitan.bot
===== 100.0%
=> Pouring ocaml-4.03.0.el_capitan.bottle.tar.gz
  /usr/local/Cellar/ocaml/4.03.0: 1,641 files, 173.7M
oliveira~ oliveira$ ocaml
OCaml version 4.03.0

# print_endline "hello world" ;;
hello world
- : unit = ()
#

```

2.3.2 Informações sobre a linguagem OCaml

A documentação oficial do OCaml [2] possui manuais, licenças, documentos e algumas dicas sobre como programar adequadamente na linguagem. OCaml é uma linguagem de programação funcional, imperativa e orientada à objetos.

2.4 Parrot Virtual Machine

2.4.1 Instalação e Teste

Digitar no Terminal:

```
$ brew install parrot
```

Resultado:

Figura 2.3: Instalando e testando Parrot



```

Last login: Tue Aug 9 19:01:04 on ttys000
oliveira:~ oliveira$ brew install parrot
=> Downloading https://homebrew.bintray.com/bottles/parrot-8.1.0.el_capitan.bot
##### 100.0%
=> Pouring parrot-8.1.0.el_capitan.bottle.tar.gz
   /usr/local/Cellar/parrot/8.1.0: 706 files, 28.4M
oliveira:~ oliveira$ parrot
parrot - [ac6hrtWuy.] - [Ddt][HEXFLAGS] [-O[level] [-[LIX] path] [-R runcore] [-o
FILE] <file> -args
oliveira:~ oliveira$

```

2.4.2 Informações sobre a Parrot Virtual Machine

A máquina virtual Parrot é utilizada principalmente para linguagens dinâmicas como Perl, Python, Ruby e PHP, seu design foi originalmente feito para trabalhar com a versão 6 de Perl, mas seu uso foi expandido como uma máquina virtual dinâmica e de propósito geral, apta a lidar com qualquer linguagem de programação de alto nível. [3]

Parrot pode ser programada em diversas linguagens, os dois mais utilizados são: Parrot Assembly Language (PASM): É a linguagem de mais baixo nível utilizada pela Parrot, muito similar a um assembly tradicional. Parrot Intermediate Representation(PIR): De mais alto nível que PASM, também um pouco mais facil de se utilizar e mais utilizada.

Fazendo alguns testes com PASM e PIR:

Listagem 2.1: Output Simples em Parrot Assembly Language

```

1 say "Here are the news about Parrots."
2 end

```

Para executar o código:

```
$ parrot news.pasm
```

Listagem 2.2: Output Simples em Parrot Intermediate Representation

```

1 .sub main :main
2   print "No parrots were involved in an accident on the M1 today...\n"
3 .end

```

Para executar o código:

```
$ parrot hello.pir
```

Os arquivos PASM e PIR são convertidos para Parrot Bytecode (PBC) e somente então são executados pela máquina virtual, é possível obter o arquivo .pbc através comando:

```
$ parrot -o output.pbc input.pasm
```

De acordo com a documentação oficial, o Compilador Intermediário de Parrot é capaz de traduzir códigos PIR para PASM através do comando:

```
$ parrot -o output.pasm input.pir
```

Mas essa execução resultou em um código bytecode invés do assembly.

Apesar da documentação oficial enfatizar que PIR é mais utilizado e mais recomendado para o desenvolvimento de compiladores para Parrot, o alvo será a linguagem Assembly PASM.

2.4.3 Parrot Assembly Language (PASM)

A linguagem PASM é muito similar a um assembly tradicional, com exceção do fato de que algumas instruções permitem o acesso a algumas funções dinâmicas de alto nível do sistema Parrot.

Parrot é uma máquina virtual baseada em registradores, há um número ilimitado de registradores que não precisam ser instanciados antes de serem utilizados, a máquina virtual se certifica de criar os registradores de acordo com a sua necessidade, tal como fazer a reutilização e se livrar de registradores que não estão mais sendo utilizados, todos os registradores começam com o símbolo "\$" e existem 4 tipos de dados, cada um com suas regras:

Strings: Registradores de strings começam com um S, por exemplo: "\$S10"

Inteiros: Registradores de inteiros começam com um I, por exemplo: "\$I10"

Número: Registradores de números de ponto flutuante, começam com a letra N, por exemplo: "\$N10"

PMC: São tipos de dados utilizados em orientação a objetos, podem ser utilizados para guardar vários tipos de dados, começam com a letra P, por exemplo: "\$P10"

Para mais referências sobre PASM, consultar [4], [7] para os opcodes e [8] para exemplos.

2.4.4 Parrot Intermediate Representation (PIR)

A maior dos compiladores possuem como alvo o PIR, inclusive o que será utilizado para estudar qual o comportamento um compilador deve ter ao gerar o assembly. A própria máquina virtual Parrot possui um módulo intermediário capaz de interpretar a linguagem PIR e gerar o bytecode ou o próprio assembly (PASM), além disso, existem compiladores capazes de realizar a mesma tarefa.

PIR é de nível mais alto que assembly mas ainda muito próximo do nível de máquina, o principal benefício é a facilidade em programar em PIR em comparação com a programação em PASM, além disso, ela foi feita para compiladores de linguagens de alto nível gerarem código PIR para trabalhar com a máquina Parrot. Mais informações sobre PIR e sua sintaxe podem ser encontradas em [5].

Capítulo 3

Compilação de Código Ruby para Parrot Intermediate Representation (PIR)

O compilador que será utilizado será o Cardinal [6], é um compilador da linguagem Ruby para a máquina virtual Parrot capaz de gerar código o código intermediário (PIR) como saída.

A documentação do compilador é simples e clara, para baixar o compilador basta digitar no terminal:

```
$ git clone git://github.com/parrot/cardinal.git
```

Entre as várias opções de instalação, é possível fazê-la utilizando do próprio parrot, para isso basta entrar na pasta onde foi baixado o Cardinal e digitar:

```
$ winxed setup.winxed build
```

Para compilar é necessário estar na pasta de instalação e o comando é:

```
$ parrot cardinal.pbc [arquivo].rb
```

Sendo o arquivo o diretório do arquivo Ruby que se deseja executar, para gerar o PIR o comando é:

```
$ parrot cardinal.pbc -o [output].pir --target=pir [arquivo].rb
```

Sendo output o diretório onde será salvo o arquivo PIR.

Exemplo

Listagem 3.1: Programa nano 01 em Ruby

```
1 # modulo minimo
```

Compilação do Código Ruby:

```
$ parrot /Users/oliveira/cardinal/cardinal/cardinal.pbc -o
../parrot/nano01.pir --target=pir nano01.rb
```

Listagem 3.2: Programa nano 01 em PIR

```
1
2 .HLL "cardinal"
3
4 .namespace []
5 .sub "_block1000" :load :main :anon :subid("10_1471301651.1019")
6     .param pmc param_1002 :optional :named("!BLOCK")
7     .param int has_param_1002 :opt_flag
8 .annotate 'file', "nano01.rb"
9 .annotate 'line', 0
10    .const 'Sub' $P1004 = "11_1471301651.1019"
11    capture_lex $P1004
12 .annotate 'line', 1
13    if has_param_1002, optparam_13
14    new $P100, "Undef"
15    set param_1002, $P100
16    optparam_13:
17    .lex "!BLOCK", param_1002
18    .return ()
19 .end
20
21
22 .HLL "cardinal"
23
24 .namespace []
25 .sub "" :load :init :subid("post12") :outer("10_1471301651.1019")
26 .annotate 'file', "nano01.rb"
27 .annotate 'line', 0
28    .const 'Sub' $P1001 = "10_1471301651.1019"
29    .local pmc block
30    set block, $P1001
31 .end
32
33
34 .HLL "parrot"
35
36 .namespace []
37 .sub "_block1003" :init :load :anon :subid("11_1471301651.1019") :outer("
    10_1471301651.1019")
38 .annotate 'file', "nano01.rb"
39 .annotate 'line', 0
40 $P0 = compreg "cardinal"
41 unless null $P0 goto have_cardinal
42 load_bytecode "cardinal.pbc"
43 have_cardinal:
44    .return ()
45 .end
```

A compilação dos programas Ruby não foi bem sucedida para todos os programas, além disso, programas que utilizavam a linha de código a seguir, que é utilizada para pegar dados do usuário, compilavam mas não funcionavam na máquina virtual.

```
input = gets.chomp
```

Capítulo 4

Códigos LUA e Parrot Assembly (PASM)

Os códigos PASM dessa seção foram feitos manualmente, não foram utilizados compiladores para esse fim.

4.0.1 Nano Programas

Nano 01

Listagem 4.1: Programa nano 01 em Lua

```
1 -- Listagem 1: Modulo minimo que caracteriza um programa
```

Listagem 4.2: Programa nano 01 em PASM

```
1 # Modulo Minimo
2 end
```

Nano 02

Listagem 4.3: Programa nano 02 em Lua

```
1 -- Listagem 2: Declaracao de uma variavel
2
3 -- Em Lua, declaracao de variaveis limitam apenas seu escopo
4 -- As variaveis podem ser local ou global
5 -- local: local x = 10 - precisam ser inicializadas
6 -- global: x = 10      - nao precisam ser inicializadas
7 -- local x            e um programa aceito em lua (declaracao de uma
    variavel local)
8 -- x                  nao e um programa aceito em lua
```

Listagem 4.4: Programa nano 02 em PASM

```
1 # Declarando uma variavel
2
3 end
```

Nano 03

Listagem 4.5: Programa nano 03 em Lua

```
1 -- Atribuicao de um inteiro a uma variavel
2 n = 1
```

Listagem 4.6: Programa nano 03 em PASM

```
1 # Atribuição de um inteiro a uma variavel
2
3 set I1, 1
4 end
```

Nano 04

Listagem 4.7: Programa nano 04 em Lua

```
1 -- Atribuicao de uma soma de inteiros a uma variavel
2 n = 1 + 2
```

Listagem 4.8: Programa nano 04 em PASM

```
1 # Atribuição de uma soma de inteiros a uma variavel
2 set I1, 1
3 set I2, 2
4 add I3, I1, I2
5 end
```

Nano 05

Listagem 4.9: Programa nano 05 em Lua

```
1 -- Inclusao do comando de impressao
2 n = 2
3 print(n)
```

Listagem 4.10: Programa nano 05 em PASM

```
1 # Inclusão do comando de impressão
2 set I1, 2
3 print I1
4 print "\n"
5
6 end
```

Saída:

2

Nano 06

Listagem 4.11: Programa nano 06 em Lua

```
1 -- Listagem 6: Atribuicao de uma subtracao de inteiros a uma variavel
```

```

2
3 n = 1 - 2
4 print(n)

```

Listagem 4.12: Programa nano 06 em PASM

```

1 # Atribuição de uma subtração de inteiros a uma variável
2 set I1, 1
3 set I2, 2
4 sub I3, I1, I2
5
6 print I3
7 print "\n"
8
9 end

```

Saída:

```
-1
```

Nano 07

Listagem 4.13: Programa nano 07 em Lua

```

1 -- Listagem 7: Inclusao do comando condicional
2 n = 1
3 if (n == 1)
4 then
5   print(n)
6 end

```

Listagem 4.14: Programa nano 07 em PASM

```

1 # Inclusão do comando condicional
2
3 set      I1, 1 # atribuição
4 eq      I1, 1, VERDADEIRO
5 branch  FIM
6
7 VERDADEIRO:
8 print   I1
9 print   "\n"
10
11 FIM:
12 end

```

Saída:

```
1
```

Nano 08

Listagem 4.15: Programa nano 08 em Lua

```

1 -- Listagem 8: Inclusao do comando condicional com parte senao
2

```


4.0

```
3 n = 1
4 if (n == 1)
5 then
6   print(n)
7 else
8   print("0")
9 end
```

Listagem 4.16: Programa nano 08 em PASM

```
1 # Inclusão do comando condicional senão
2
3 set      I1, 1
4 eq       I1, 1, VERDADEIRO
5 print    "0\n"
6 branch   FIM
7
8 VERDADEIRO:
9 print    I1
10 print   "\n"
11
12 FIM:
13 end
```

Saída:

1

Nano 09

Listagem 4.17: Programa nano 09 em Lua

```
1 -- Listagem 9: Atribuicao de duas operacoes aritmeticas sobre inteiros a
   uma variavel
2
3 n = 1 + 1 / 2
4 if (n == 1)
5 then
6   print(n)
7 else
8   print("0")
9 end
```

Listagem 4.18: Programa nano 09 em PASM

```
1 # Atribuição de duas operações aritmeticas sobre inteiros a uma variável
2
3 set      I1, 1
4 set      I2, 2
5 div      I3, I1, I2
6 add      I4, I1, I3
7
8 eq       I4, 1, VERDADEIRO
9 print    "0\n"
10 branch   FIM
11
12 VERDADEIRO:
```

```

13 print    I4
14 print    "\n"
15
16 FIM:
17 end

```

Saída:

1

Nano 10

Listagem 4.19: Programa nano 10 em Lua

```

1 -- Listagem 10: Atribuicao de duas variaveis inteiras
2 n = 1
3 m = 2
4
5 if(n == m)
6 then
7   print(n)
8 else
9   print("0")
10 end

```

Listagem 4.20: Programa nano 10 em PASM

```

1 # Atribuição de duas variáveis inteiras
2
3 set    I1, 1
4 set    I2, 2
5
6 eq I1, I2, VERDADEIRO
7 print  "0\n"
8 branch FIM
9
10 VERDADEIRO:
11 print  I1
12 print  "\n"
13
14 FIM:
15 end

```

Saída:

0

Nano 11

Listagem 4.21: Programa nano 11 em Lua

```

1 -- Listagem 11: Introducao do comando de repeticao enquanto
2 n = 1
3 m = 2
4 x = 5
5

```

4.0

```
6 while (x > n)
7 do
8   n = n + m
9   print(n)
10 end
```

Listagem 4.22: Programa nano 11 em PASM

```
1 # Introdução do comando de repetição enquanto
2
3 set      I1, 1 # n
4 set      I2, 2 # m
5 set      I3, 5 # x
6
7 TESTE:
8 gt       I3, I1, LOOP # gt = greater then
9 branch   FIM
10
11 LOOP:
12 add      I1, I1, I2
13 print    I1
14 print    "\n"
15 branch   TESTE
16
17 FIM:
18 end
```

Saída:

```
3
5
```

Nano 12

Listagem 4.23: Programa nano 12 em Lua

```
1 -- Listagem 12: Comando condicional aninhado em um comando de repeticao
2 n = 1
3 m = 2
4 x = 5
5
6 while (x > n)
7 do
8   if (n == m)
9   then
10    print(n)
11  else
12    print("0")
13  end
14  x = x - 1
15 end
```

Listagem 4.24: Programa nano 12 em PASM

```
1 # Comando condicional aninhado com um de repeticao
2
3 set      I1, 1
```

```

4 set      I2, 2
5 set      I3, 5
6
7 TESTE_ENQUANTO:
8 gt       I3, I1, LOOP
9 branch   FIM
10
11 LOOP:
12 eq       I1, I2, VERDADEIRO
13 print    "0\n"
14 branch   POS_CONDICIONAL
15
16 VERDADEIRO:
17 print    I1
18 print    "\n"
19
20 POS_CONDICIONAL:
21 dec      I3                # decrementa I3 (x)
22 branch   TESTE_ENQUANTO
23
24 FIM:
25 end

```

Saída:

```

0
0
0
0

```

4.0.2 Micro Programas

Micro 01

Listagem 4.25: Programa micro 01 em Lua

```

1 -- Listagem 13: Converte graus Celsius para Fahrenheit
2
3 -- [[ Funcao: Ler uma temperatura em graus Celsius e apresenta-la
   convertida em graus Fahrenheit. A formula de conversao e : F=(9*C+160)
   / 5, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius.
   --]]
4
5 print("Tabela de Conversão: Celsius -> Fahrenheit")
6 print("Digite a Temperatura em Celsius: ")
7 cel = io.read("*number")
8 far = (9*cel+160)/5
9 print("A nova temperatura é:", far)

```

Listagem 4.26: Programa Micro 01 em PASM

```

1 # Converte graus Celsius para Fahrenheit
2 .loadlib 'io_ops'          # Para fazer IO
3
4 set      S1, "Tabela de Conversao: Celsius -> Fahrenheit\n"
5 set      S2, "Digite a Temperatura em Celsius: "
6 set      S3, "A nova temperatura e: "

```

4.0

```
7 set      S4, " graus F."
8
9 print    S1
10 print   S2
11 read    S10, 5
12 set     I1, S10
13
14 mul     I1, I1, 9
15 add     I1, I1, 160
16 div     I1, I1, 5
17
18 print    S3
19 print    I1
20 print    S4
21 print    "\n"
22
23 end
```

Tabela de Conversao: Celsius -> Fahrenheit
Digite a Temperatura em Celsius: 20
A nova temperatura e: 68 graus F.

Micro 02

Listagem 4.27: Programa micro 02 em Lua

```
1 -- Listagem 14: Ler dois inteiros e decide qual e maior
2
3 --[[ Funcao : Escrever um algoritmo que leia dois valores inteiro
   distintos e informe qual e o maior --]]
4
5 print("Escreva o primeiro número:")
6 num1 = io.read("*number")
7 print("Escreva o segundo número:")
8 num2 = io.read("*number")
9
10 if(num1 > num2)
11 then
12   print("O primeiro número", num1, "é maior que o segundo", num2)
13 else
14   print("O segundo número", num2, "é maior que o primeiro", num1)
15 end
```

Listagem 4.28: Programa Micro 02 em PASM

```
1 # Ler dois inteiros e decidir qual e maior
2 .loadlib 'io_ops'
3
4 set      S1, "Digite o primeiro numero: "
5 set      S2, "Digite o segundo numero: "
6 set      S3, "o primeiro numero"
7 set      S4, "o segundo numero"
8 set      S5, " e maior que "
9
10 print    S1
11 read     S10, 3
12 set     I1, S10
13 print    S2
```

```

14 read      S11, 3
15 set      I2, S11
16
17 gt       I1, I2, VERDADEIRO
18 print    S4
19 print    S5
20 print    S3
21 print    "\n"
22 branch   FIM
23
24 VERDADEIRO:
25 print    S3
26 print    S5
27 print    S4
28 print    "\n"
29
30 FIM:
31 end

```

```

Digite o primeiro numero: 10
Digite o segundo numero: 20
o segundo numero e maior que o primeiro numero
Digite o primeiro numero: 20
Digite o segundo numero: 10
o primeiro numero e maior que o segundo numero

```

Micro 03

Listagem 4.29: Programa micro 03 em Lua

```

1 -- Le um numero e verifica se ele esta entre 100 e 200
2 --[[ Funcao: Faca um algoritmo que receba um numero e diga se este numero
   esta no intervalo entre 100 e 200 --]]
3
4 print("Digite um número:")
5 numero = io.read("*number")
6
7 if(numero >= 100)
8 then
9     if(numero <= 200)
10 then
11     print("O número está no intervalo entre 100 e 200")
12 else
13     print("O número não está no intervalo entre 100 e 200")
14 end
15 else
16     print("O número não está no intervalo entre 100 e 200")
17 end

```

Listagem 4.30: Programa Micro 03 em PASM

```

1 # Le um numero e verifica se ele esta entre 100 e 200
2 .loadlib 'io_ops'
3
4 set      S1, "Digite um numero: "
5 set      S2, "O numero esta no intervalo entre 100 e 200\n"
6 set      S3, "O numero nao esta no intervalo entre 100 e 200\n"
7

```

4.0

```
8 print      S1
9 read      S10, 3
10 set      I1, S10
11
12 ge        I1, 100, MAIOR_QUE_100
13 branch    NAO_ESTA_NO_INTERVALO
14
15 MAIOR_QUE_100:
16 le        I1, 200, MENOR_QUE_200
17
18 NAO_ESTA_NO_INTERVALO:
19 print      S3
20 branch    FIM
21
22 MENOR_QUE_200:
23 print      S2
24
25 FIM:
26 end
```

```
Digite um numero: 5
O numero nao esta no intervalo entre 100 e 200

Digite um numero: 150
O numero esta no intervalo entre 100 e 200

Digite um numero: 201
O numero nao esta no intervalo entre 100 e 200
```

Micro 04

Listagem 4.31: Programa micro 04 em Lua

```
1 -- Listagem 16: Le numeros e informa quais estao entre 10 e 150
2
3 --[[ Função: Ler 5 numeros e ao final informar quantos numeros estao no
   intervalo entre 10 (inclusive) e 150(inclusive) --]]
4
5 intervalo = 0
6
7 for x=1,5,1
8 do
9   print("Digite um número")
10  num = io.read("*number")
11  if(num >= 10)
12  then
13    if(num <= 150)
14    then
15      intervalo = intervalo + 1
16    end
17  end
18 end
19
20 print("Ao total, foram digitados",intervalo,"números no intervalo entre 10
   e 150")
```

Listagem 4.32: Programa Micro 04 em PASM

```

1 # Le numeros e informa quais estao entre 10 e 150
2 .loadlib 'io_ops'
3
4 set      S1, "Digite um numero: "
5 set      S2, "Ao total foram digitados "
6 set      S3, " numeros no intervalo entre 10 e 150."
7
8 set      I1, 1
9 set      I2, 0
10
11 LOOP_TESTE:
12 le      I1, 5, INICIO_LOOP
13 branch  FIM
14
15 INICIO_LOOP:
16 print    S1
17 read     S10, 3
18 set      I10, S10
19
20 ge      I10, 10, MAIOR_QUE_10
21 branch  FIM_LOOP
22
23 MAIOR_QUE_10:
24 le      I10, 150, MENOR_QUE_150
25 branch  FIM_LOOP
26
27 MENOR_QUE_150:
28 inc     I2
29
30 FIM_LOOP:
31 inc     I1
32 branch  LOOP_TESTE
33
34
35 FIM:
36 print    S2
37 print    I2
38 print    S3
39 print    "\n"
40 end

```

```

Digite um numero: 50
Digite um numero: 50
Digite um numero: 50
Digite um numero: 50
Digite um numero: 50
Ao total foram digitados 5 numeros no intervalo entre 10 e 150.

Digite um numero: 02
Digite um numero: 03
Digite um numero: 25
Digite um numero: 60
Digite um numero: 160
Ao total foram digitados 2 numeros no intervalo entre 10 e 150.

```

Micro 05

4.0

```
1 -- Listagem 17: Le strings e caracteres
2 --[[ Funcao: Escrever um algoritmo que leia o nome e o sexo de 56 pessoas
   e informe o nome e se ela e homem ou mulher. No final informe o total
   de homens e mulheres --]]
3
4 h = 0
5 m = 0
6 for x=1,5,1
7 do
8   print("Digite o nome: ")
9   nome = io.read()
10  print("H - Homem ou M - Mulher")
11  sexo = io.read()
12  if(sexo == 'H') then h = h + 1
13  elseif (sexo == 'M') then m = m + 1
14  else print("Sexo só pode ser H ou M!")
15  end
16 end
17
18 print("Foram inseridos",h,"homens")
19 print("Foram inseridas",m,"mulheres")
```

Listagem 4.34: Programa Micro 05 em PASM

```
1 # Le strings e caracteres
2 .loadlib 'io_ops'
3
4 set      S2, "H - Homem ou M - Mulher: "
5 set      S3, "Sexo so pode ser H ou M!\n"
6 set      S4, "Foram inseridos "
7 set      S5, "Foram inseridas "
8 set      S6, " homens"
9 set      S7, " mulheres"
10
11 set      I1, 1           # x
12 set      I2, 0           # homens
13 set      I3, 0           # mulheres
14
15 LOOP_TESTE:
16 le       I1, 5, INICIO_LOOP
17 branch   FIM
18
19 INICIO_LOOP:
20 print    S2
21 read     S11, 2
22
23 eq       S11, "H\n", HOMEM
24 eq       S11, "M\n", MULHER
25
26 print    S3
27 branch   FIM_LOOP
28
29 HOMEM:
30 inc      I2
31 branch   FIM_LOOP
32
33 MULHER:
34 inc      I3
35
```

```

36 FIM_LOOP:
37 inc      I1
38 branch   LOOP_TESTE
39
40 FIM:
41 print     S4
42 print     I2
43 print     S6
44 print     "\n"
45
46 print     S5
47 print     I3
48 print     S7
49 print     "\n"
50 end

```

```

H - Homem ou M - Mulher: H
H - Homem ou M - Mulher: M
H - Homem ou M - Mulher: H
H - Homem ou M - Mulher: M
H - Homem ou M - Mulher: M
Foram inseridos 2 homens
Foram inseridas 3 mulheres

```

Micro 06

Listagem 4.35: Programa micro 06 em Lua

```

1 -- Escreve um numero lido por extenso
2
3 --[[ Funcao: Faça um algoritmo que leia um número de 1 a 5 e o escreva por
   extenso. Caso o usuario digite um numero que nao esteja nesse
   intervalo, exibir mensagem: numero invalido --]]
4
5 print("Digite um número de 1 a 5")
6 numero = io.read("*number")
7 if(numero == 1) then print("Um")
8 elseif (numero == 2) then print("Dois")
9 elseif (numero == 3) then print("Três")
10 elseif (numero == 4) then print("Quatro")
11 elseif (numero == 5) then print("Cinco")
12 else print("Número Invalido!!!")
13 end

```

Listagem 4.36: Programa Micro 06 em PASM

```

1 # Escrever um numero por extenso
2 .loadlib 'io_ops'
3
4 print      "Digite um numero de 1 a 5: "
5 read      S1, 2
6 set       I1, S1
7
8 eq        I1, 1, UM
9 eq        I1, 2, DOIS
10 eq       I1, 3, TRES
11 eq       I1, 4, QUATRO
12 eq       I1, 5, CINCO

```

4.0

```
13
14 print      "Numero invalido!!!"
15 branch     FIM
16
17 CINCO:
18 print      "Cinco"
19 branch     FIM
20
21 QUATRO:
22 print      "Quatro"
23 branch     FIM
24
25 TRES:
26 print      "Tres"
27 branch     FIM
28
29 DOIS:
30 print      "Dois"
31 branch     FIM
32
33 UM:
34 print      "Um"
35
36 FIM:
37 print      "\n"
38 end
```

```
Digite um numero de 1 a 5: 3
Tres
```

Micro 07

Listagem 4.37: Programa micro 07 em Lua

```
1 -- Listagem 19: Decide se os numeros sao positivos, zeros ou negativos
2
3 --[[ Funcao: Faca um algoritmo que receba N numeros e mostre positivo,
   negativo ou zero para cada número --]]
4
5 programa = 1
6 while (programa == 1)
7 do
8   print("Digite um numero: ")
9   numero = io.read()
10  numero = tonumber(numero)
11
12  if(numero > 0)
13  then print("Positivo")
14  elseif(numero == 0)
15  then print("O número é igual a 0")
16  elseif(numero < 0)
17  then print("Negativo")
18  end
19
20
21 print("Deseja Finalizar? (S/N) ")
22 opc = io.read("*line")
23
```

```

24  if(opc == "S")
25  then programa = 0
26  end
27  end

```

Listagem 4.38: Programa Micro 07 em PASM

```

1  # Decide se os numeros sao positivos, zeros ou negativos
2  .loadlib 'io_ops'
3
4  LOOP:
5  print      "Digite um numero: "
6  read      S1, 3
7  set       I1, S1
8
9  # Testar se e maior que 0
10 gt        I1, 0, POSITIVO
11 eq        I1, 0, ZERO
12 lt        I1, 0, NEGATIVO
13
14 POSITIVO:
15 print      "Positivo!\n"
16 branch    FINALIZAR
17
18 ZERO:
19 print      "Zero!\n"
20 branch    FINALIZAR
21
22 NEGATIVO:
23 print      "Negativo!\n"
24
25 # Parte de DESEJA FINALIZAR?
26 FINALIZAR:
27 print      "Deseja finalizar? (S/N): "
28 read      S10, 2
29 eq        S10, "S\n", FIM
30 branch    LOOP
31
32 FIM:
33 end

```

```

Digite um numero: 5
Positivo!
Deseja finalizar? (S/N): N
Digite um numero: -5
Negativo!
Deseja finalizar? (S/N): N
Digite um numero: 0
Zero!
Deseja finalizar? (S/N): S

```

Micro 08

Listagem 4.39: Programa micro 08 em Lua

```

1  -- Listagem 20: Decide se um numero e maior ou menor que 10
2
3  numero = 1

```

```

4 while(numero ~= 0)
5 do
6   print("Escreva um numero: ")
7   numero = tonumber(io.read())
8
9   if(numero > 10)
10    then print("O numero",numero,"e maior que 10")
11    else print("O numero",numero,"e menor que 10")
12    end
13 end

```

Listagem 4.40: Programa Micro 08 em PASM

```

1 # Decide se um número é maior ou menor que 10
2 .loadlib 'io_ops'
3
4 set      I1, 1                # variavel numero
5
6 TESTE_LOOP:
7 ne      I1, 0, LOOP
8 branch   FIM
9
10 LOOP:
11 print    "Digite um numero: "
12 read     S10, 3
13 set      I1, S10
14
15 gt      I1, 10, MAIOR
16 print    "O numero "
17 print    I1
18 print    " e menor que 10.\n"
19 branch   TESTE_LOOP
20
21 MAIOR:
22 print    "O numero "
23 print    I1
24 print    " e maior que 10.\n"
25 branch   TESTE_LOOP
26
27 FIM:
28 end

```

```

Digite um numero: 50
O numero 50 e maior que 10.
Digite um numero: 5
O numero 5 e menor que 10.
Digite um numero: 0
O numero 0 e menor que 10.

```

Micro 09

Listagem 4.41: Programa micro 09 em Lua

```

1 -- Listagem 21: Calculo de Precos
2
3 print("Digite o preco: ")
4 preco = tonumber(io.read())
5 print("Digite a venda: ")

```

```

6 venda = tonumber(io.read())
7
8 if ((venda < 500) or (preco < 30))
9 then novo_preco = preco + (10/100 * preco)
10 elseif ((venda >= 500 and venda < 1200) or (preco >= 30 and preco < 80))
11 then novo_preco = preco + (15/100 * preco)
12 elseif (venda >= 1200 or preco >= 80)
13 then novo_preco = preco - (20/100 * preco)
14 end
15
16 print("O novo preco e: ", novo_preco)

```

Listagem 4.42: Programa Micro 09 em PASM

```

1 # Calculo de precos
2 .loadlib 'io_ops'
3
4
5 print          "Digite o preco (max. 2 digitos): "
6 read          S1, 3
7 set           N1, S1
8 print          "Digite a venda (max. 4 digitos): "
9 read          S1, 5
10 set           N2, S1
11
12 lt            N2, 500, AUMENTAR_10_PORCENTO
13 ge            N1, 30, FALSO1
14
15 AUMENTAR_10_PORCENTO:
16 mul           N3, 10, N1
17 div           N3, N3, 100
18 add           N3, N3, N1
19 branch        FIM
20
21 FALSO1:
22 lt            N2, 500, SEGUNDO_TESTE
23 lt            N2, 1200, AUMENTAR_15_PORCENTO
24 SEGUNDO_TESTE:
25 lt            N1, 30, FALSO2
26 ge            N1, 80, FALSO2
27
28 AUMENTAR_15_PORCENTO:
29 mul           N3, N1, 15
30 div           N3, N3, 100
31 add           N3, N3, N1
32 branch        FIM
33
34 FALSO2:
35 ge            N2, 1200, DIMINUIR_20_PORCENTO
36 lt            N1, 80, FIM
37
38 DIMINUIR_20_PORCENTO:
39 mul           N3, 20, N1
40 div           N3, N3, 100
41 sub           N3, N1, N3
42
43 FIM:
44 print          "O novo preco e: "
45 print          N3

```

4.0

```
46 print          "\n"
47 end
```

```
Digite o preco: 10
Digite a venda: 10
O novo preco e: 11

Digite o preco: 40
Digite a venda: 600
O novo preco e: 46

Digite o preco: 90
Digite a venda: 1500
O novo preco e: 72
```

Micro 10

Listagem 4.43: Programa micro 10 em Lua

```
1 --Listagem 22: Calcula o fatorial de um numero
2
3 --[[ Funcao: recebe um numero e calcula recursivamente o fatorial desse nú
    mero --]]
4
5 function fatorial(n)
6     if(n <= 0)
7     then return 1
8     else return (n* fatorial(n-1))
9     end
10 end
11
12 print("Digite um numero: ")
13 numero = tonumber(io.read())
14 fat = fatorial(numero)
15
16 print("O fatorial de", numero, "e: ", fat)
```

Listagem 4.44: Programa Micro 10 em PASM

```
1 # Calcula o fatorial de um numero
2 .loadlib 'io_ops'
3
4 print          "Digite um numero: "
5 read          S1, 2
6 set           I1, S1
7 set           I10, S1
8
9 branch        FATORIAL
10 RETURN:
11 print         "O fatorial de "
12 print         I1
13 print         " e: "
14 print         I10
15 print         "\n"
16
17 end
18
19
```

```

20 FATORIAL:
21 set      I11, I10
22 dec      I11
23
24 TESTE:
25 eq        I11, 0, RETURN
26 mul       I10, I10, I11
27 dec       I11
28 branch    TESTE

```

```

Digite um numero: 5
O fatorial de 5 e: 120

```

Micro 11

Listagem 4.45: Programa micro 11 em Lua

```

1 -- Listagem 23: Decide se um numero e positivo, zero ou negativo com o
  auxilio de uma funcao.
2
3 --[[ Funcao: recebe um numero e verifica se o numero e positivo, nulo ou
  negativo com o auxilio de uma funcao --]]
4
5 function verifica(n)
6     if(n > 0)
7     then res = 1
8     elseif (n < 0)
9     then res = -1
10    else res = 0
11    end
12
13    return res
14 end
15
16 print("Escreva um numero: ")
17 numero = tonumber(io.read())
18 x = verifica(numero)
19
20 if(x==1)
21 then print("Numero positivo")
22 elseif(x==0)
23 then print("Zero")
24 else print("Numero negativo")
25 end

```

Listagem 4.46: Programa Micro 11 em PASM

```

1 # Decide se um numero e positivo, zero ou negativo com auxilio de uma
  subrotina
2 .loadlib 'io_ops'
3
4 print      "Digite um numero: "
5 read       S1, 3
6 set        I1, S1
7
8 set        I2, 0                # variavel que tera o resultado
9 branch     VERIFICA
10 RETORNO:

```


4.0

```
11
12 eq      I2, 1,  POSITIVO
13 eq      I2, 0,  ZERO
14 print   "Negativo\n"
15 branch  FIM
16
17 ZERO:
18 print   "Zero\n"
19 branch  FIM
20
21 POSITIVO:
22 print   "Positivo\n"
23
24 FIM:
25 end
26
27 VERIFICA:
28 gt      I1, 0,  MAIOR
29 lt      I1, 0,  MENOR
30 branch  FIM_SUB
31
32 MENOR:
33 set     I2, -1
34 branch  FIM_SUB
35
36 MAIOR:
37 set     I2, 1
38
39 FIM_SUB:
40 branch  RETORNO
```

Digite um numero: 5
Positivo

Digite um numero: -5
Negativo

Digite um numero: 0
Zero

Capítulo 5

Analizador Léxico

5.1 Análise Léxica

A Análise Léxica tem como principal objetivo facilitar o entedimento do programa para as análises subsequentes do compilador. Essa análise poderia ser feita juntamente com a análise sintática, mas é feita separada por motivos de eficiência, modularização (facilita manutenção e alterações futuras) e por tradição, as linguagens geralmente são criadas com módulos separados para análise léxica e sintática.

As expressões para a análise geralmente são escritas em expressões regulares, assim, os analisadores léxicos são criados como um automato finito determinístico.

Um Analizador Léxico tem como input uma string correspondente a todo o código digitado, essa string é separada em uma lista de caracteres que subsequentemente é separada em tokens. Após a separação em tokens, é trabalho do analisador verificar a corretude léxica do código bem como rotular corretamente cada token reconhecido.

5.2 Analizador Manual

5.2.1 Linguagem a ser interpretada

A linguagem a ser interpretada é uma linguagem simples que reconhece algumas palavras reservadas básicas de todas as linguagens de programação, como: print, if, then, else e comandos como atribuição, soma, subtração e multiplicação, além de reconhecer identificadores e números inteiros, todas as linhas devem ser terminadas com um ponto e vírgula, um exemplo de um programa nessa linguagem:

```
b := 2;
a := 1 + b;
print (a * b);
if1 := a - 2;
if2 := b + 3;
if if1 > 0
then print (if1);
```

```
else print (if2);
```

5.2.2 Autômato reconhecedor da linguagem

O autômato capaz de interpretar a linguagem é um autômato do tipo $M = (ALF, Q, d, q_0, F)$, onde F é o alfabeto de símbolos de entrada, Q são os estados possíveis do autômato, d é a função de transição, q_0 é o estado inicial e F é o conjunto de todos os estados finais do autômato.

ALF = o alfabeto de entrada é qualquer conjunto de palavras sobre o conjunto ASCII 2

Q = inicio, p, pr, pri, prin, print, i, if, t, th, the, then, e, el, els, else, identificador, inteiro, abre-parentese, fecha-parentese, comparador, operador, dois-pontos, atribuicao, ponto-virgula, estado-morto

q_0 = inicio

F = print, if, then, else, identificador, inteiro, abre-parentese, fecha-parentese, comparador, operador, atribuicao, ponto-virgula

d = A função de transição será descrita na forma: (estado atual, simbolo lido, proximo estado), todos as transições que não forem listadas dessa forma levam ao estado morto que invalida a palavra lida.

(inicio, 'p', p)

(inicio, 'i', i)

(inicio, 't', t)

(inicio, 'e', e)

(inicio, '(', abre-parentese)

(inicio, ')', fecha-parentese)

(inicio, '>', comparador)

(inicio, '+', '-', '*', operador)

(inicio, ':', dois-pontos)

(inicio, ';', ponto-virgula)

(inicio, '0'-'9', inteiro)

(inicio, 'a'-'z', 'A'-'Z' ou '_', identificador)

(p, 'r', pr)

(p, 'a'-'z' exceto 'r' ou 'A'-'Z' ou '0'-'9', identificador)

(pr, 'i', pri)
 (pr, 'a'-'z' exceto 'i' ou 'A'-'Z' ou '0'-'9', identificador)
 (pri, 'n', prin)
 (pri, 'a'-'z' exceto 'n' ou 'A'-'Z' ou '0'-'9', identificador)
 (prin, 't', print)
 (print, 'a'-'z' exceto 't' ou 'A'-'Z' ou '0'-'9', identificador)
 (print, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (i, 'f', if)
 (i, 'a'-'z' exceto 'f', 'A'-'Z' ou '0'-'9', identificador)
 (if, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (e, 'l', el)
 (e, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (el, 's', els)
 (el, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (els, 'e', else)
 (els, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (else, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (t, 'h', th)
 (t, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (th, 'e', the)
 (th, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (the, 'n', then)
 (the, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (then, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (identificador, 'a'-'z' ou 'A'-'Z' ou '0'-'9', identificador)
 (inteiro, '0'-'9', inteiro)
 (dois-pontos, '=', atribuicao)
 (atribuicao, -, estado-morto)

(abre-parentese, -, estado-morto)

(fecha-parentese, -, estado-morto)

(comparador, -, estado-morto)

(operador, -, estado-morto)

(ponto-virgula, -, estado-morto)

5.2.3 Implementação

Convenções

Para facilitar a escrita e o entendimento do código, os estados descritos acima foram codificados como números inteiros da seguinte forma:

Nome do Estado	Inteiro Correspondente
inicio	0
p	1
pr	2
pri	3
prin	4
print	5
i	6
if	7
t	8
th	9
the	10
then	11
e	12
el	13
els	14
else	15
identificador	16
inteiro	17
abre-parentese	18
fecha-parentese	19
comparador	20
operador	21
dois-pontos	22
ponto-virgula	23
branco	24
atribuicao	25
estado-morto	-1

Código do Autômato

Abaixo se encontram as modificações feitas no código já fornecido do analisador léxico para que ele pudesse reconhecer a linguagem descrita mais acima:

Listagem 5.1: Automato reconhecedor da linguagem descrita

```

1 type token =
2   | If
3   | Then
4   | Else
5   | AbreParentese
6   | FechaParentese
7   | Comparador of string
8   | Operador of string
9   | Atribuicao
10  | PontoVirgula
11  | Id of string
12  | Int of string
13  | Print
14  | Branco
15  | EOF
16
17 let lexico (str:entrada) =
18   let trans (e:estado) (c:simbolo) =
19     match (e,c) with
20       | (0, 'p') -> 1
21       | (0, 'i') -> 6
22       | (0, 't') -> 8
23       | (0, 'e') -> 12
24       | (0, '(') -> 18
25       | (0, ')') -> 19
26       | (0, '>') -> 20
27       | (0, '+') -> 21
28       | (0, '-') -> 21
29       | (0, '*') -> 21
30       | (0, ':') -> 22
31       | (0, ';') -> 23
32       | (0, _) when eh_letra c -> 16
33       | (0, _) when eh_digito c -> 17
34       | (0, _) when eh_branco c -> 24
35       | (0, _) ->
36         failwith ("Erro lexico: caracter desconhecido " ^ Char.escaped c)
37
38       | (1, 'r') -> 2
39       | (1, _) when eh_letra c || eh_digito c -> 16
40
41       | (2, 'i') -> 3
42       | (2, _) when eh_letra c || eh_digito c -> 16
43
44       | (3, 'n') -> 4
45       | (3, _) when eh_letra c || eh_digito c -> 16
46
47       | (4, 't') -> 5
48       | (4, _) when eh_letra c || eh_digito c -> 16
49
50       | (5, _) when eh_letra c || eh_digito c -> 16
51
52       | (6, 'f') -> 7

```

5.2

```
53
54 | (7, _) when eh_letra c || eh_digito c -> 16
55
56 | (8, 'h') -> 9
57 | (8, _) when eh_letra c || eh_digito c -> 16
58
59 | (9, 'e') -> 10
60 | (9, _) when eh_letra c || eh_digito c -> 16
61
62 | (10, 'n') -> 11
63 | (10, _) when eh_letra c || eh_digito c -> 16
64
65 | (11, _) when eh_letra c || eh_digito c -> 16
66
67 | (12, 'l') -> 13
68 | (12, _) when eh_letra c || eh_digito c -> 16
69
70 | (13, 's') -> 14
71 | (13, _) when eh_letra c || eh_digito c -> 16
72
73 | (14, 'e') -> 15
74 | (14, _) when eh_letra c || eh_digito c -> 16
75
76 | (15, _) when eh_letra c || eh_digito c -> 16
77
78 | (16, _) when eh_letra c || eh_digito c -> 16
79
80 | (17, _) when eh_digito c -> 17
81
82 | (22, '=') -> 25
83
84 | (24, _) when eh_branco c -> 24
85 | _ -> estado_morto
86 and rotulo e str =
87 match e with
88 | 7 -> If
89 | 5 -> Print
90 | 11 -> Then
91 | 15 -> Else
92 | 18 -> AbreParentese
93 | 19 -> FechaParentese
94 | 20 -> Comparador str
95 | 21 -> Operador str
96 | 25 -> Atribuicao
97 | 23 -> PontoVirgula
98 | 1
99 | 2
100 | 3
101 | 4
102 | 6
103 | 8
104 | 9
105 | 10
106 | 12
107 | 13
108 | 14
109 | 16 -> Id str
110 | 17 -> Int str
111 | 24 -> Branco
```

```
112 | _ -> failwith ("Erro lexico: sequencia desconhecida " ^ str)
```

Reconhecimento do Código

Para demonstrar o funcionamento do autômato, será feita a entrada de um exemplo de programa, que foi especificado anteriormente e está novamente listado abaixo:

```
b := 2;
a := 1 + b;
print (a * b);
if1 := a - 2;
if2 := b + 3;
if if1 > 0
then print(if1);
else print(if2);
```

Para iniciar o Ocaml, ler o arquivo e interpretar um código, os seguintes comandos devem ser especificados no terminal:

```
rlwrap ocaml
#use "dfalexer.ml";;
lexico "comando";;

exit 0;;
```

Para testar o exemplo, será feita a entrada do programa como uma única string:

```
#use "dfalexer.ml";;
lexico "b := 2;\na := 1 + b;\nprint (a * b);\nif1 := a - 2;\nif2 := b +
3;\nif if1 > 0\nthen print(if1);\nelse print(if2);";;

- : token list =
[Id "b"; Branco; Atribuicao; Branco; Int "2"; PontoVirgula; Branco; Id "a"
;
Branco; Atribuicao; Branco; Int "1"; Branco; Operador "+"; Branco; Id "b"
;
PontoVirgula; Branco; Print; Branco; AbreParentese; Id "a"; Branco;
Operador "*"; Branco; Id "b"; FechaParentese; PontoVirgula; Branco;
Id "if1"; Branco; Atribuicao; Branco; Id "a"; Branco; Operador "-";
Branco;
Int "2"; PontoVirgula; Branco; Id "if2"; Branco; Atribuicao; Branco;
Id "b"; Branco; Operador "+"; Branco; Int "3"; PontoVirgula; Branco; If;
Branco; Id "if1"; Branco; Comparador ">"; Branco; Int "0"; Branco; Then;
Branco; Print; AbreParentese; Id "if1"; FechaParentese; PontoVirgula;
Branco; Else; Branco; Print; AbreParentese; Id "if2"; FechaParentese;
PontoVirgula; EOF]
```

Para verificar a corretude do automato, testarei também com um exemplo negativo, por exemplo, colocando um @ no meio da string de teste:

```
#use "dfalexer.ml";;
lexico "b := 2;\na := 1 @ + b;\nprint (a * b);\nif1 := a - 2;\nif2 := b +
3;\nif if1 > 0\nthen print(if1);\nelse print(if2);";;

Exception: Failure "Erro lexico: caracter desconhecido @".
```


5.3 Analisador com Ocamllexer

5.3.1 Convenções Léxicas da Linguagem Lua

Na documentação oficial da Linguagem Lua, as convenções léxicas podem ser encontradas na Seção 3.1.

Lua ignora espaços brancos, novas linhas e comentários entre elementos léxicos (tokens).

Nomes (identificadores) em Lua podem ser strings de letras, dígitos e underscore, não podendo começar com um dígito. Identificadores são utilizados para rotular variáveis, tabelas e rótulos.

A Linguagem possui as seguintes palavras reservadas:

and	break	do	else	elseif	end
false	for	function	goto	if	in
local	nil	not	or	repeat	return
then	true	until	while		

Lua é caso sensível, and é uma palavra reservada, porém AND, And, aNd, etc... podem ser utilizadas como identificadores. Por convenção, identificadores que começam com underscore e são seguidos por letras maiúsculas (como `_VERSION`) são variáveis utilizadas pela linguagem

São outros tokens reconhecidos pela linguagem:

+	-	*	/	%	x	#
==	=	<=	>=	<	>	=
()	{	}	[]	:
;	:	,	

Strings podem ser limitadas por aspas simples ou duplas e pode conter qualquer caractere de escape contido no C:

```
'\a', '\b', '\f', '\n', '\r', '\t', '\v', '\\', '\"' e '\'
```

Strings em Lua também podem ser criadas utilizando uma notação de colchetes, em n níveis, as seguintes strings são aceitas em lua:

```
[[teste]] [= [teste]=] [== [teste]==] ...
```

Uma constante numérica pode ser escrita com uma parte fracionária opcional, marcada por 'e' ou 'E'. Lua também aceita constantes hexadecimais, que começam com '0x' ou '0X', e aceitam um complemento binário que vem após um 'p' ou 'P', as seguintes constantes numéricas são aceitas:

```
3      3.0      3.1416      314.16e-2      0.31416E1
0xff   0x0.1E   0xA23p-4    0X1.921FB54442D18P+1
```

Um comentário começa com `--` fora de uma string, se após os dois hífen não vier um colchete, o comentário é de uma linha, caso contrário, o comentário segue até encontrar um colchete fechando o comentário.

5.3.2 Implementação

5.3.3 Testes

Capítulo 6

Referências

- [1] Documentação Lua - <https://www.lua.org/docs.html>
- [2] Documentação OCaml - <https://ocaml.org/docs/>
- [3] Wikibooks, Parrot Virtual Machine - https://en.wikibooks.org/wiki/Parrot_Virtual_Machine
- [4] Wikibooks, PASM Reference - https://en.wikibooks.org/wiki/Parrot_Virtual_Machine/PASM_Reference
- [5] Wikibooks, Parrot Intermediate Representation (PIR) - https://en.wikibooks.org/wiki/Parrot_Virtual_Machine/Parrot_Intermediate_Representation
- [6] Cardinal, Github - <https://github.com/parrot/cardinal/>
- [7] Opcodes de PASM - <http://docs.parrot.org/parrot/latest/html/ops.html>
- [8] Parrot Documentation, Exemplos de PASM - <http://parrot.org/dev/examples/pasm>
- [9] Basics of Copiler Design - Torben Ægidius Mogensen