



Universidade de São Paulo  
Instituto de Ciências Matemáticas e de Computação (ICMC)  
SCC0605 - Teoria da Computação e Compiladores

## Trabalho 1: Análise Léxica

Lucas Furco Granela - 11299978  
Matheus Yasuo Ribeiro Utino - 11233689  
Pedro Ribas Serras - 11234328  
Vinícius Silva Montanari - 11233709

Docente: Dr. Thiago A. S. Pardo

São Carlos  
05 de junho de 2022

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Construção dos autômatos</b>	<b>3</b>
2.1	Autômato para reconhecimento de Identificadores . . . . .	3
2.2	Autômato para reconhecimento de números inteiros e reais . . . . .	4
2.3	Autômato para reconhecimento de comentários . . . . .	5
2.4	Autômato para reconhecimento de operadores . . . . .	6
2.5	Autômato final . . . . .	7
<b>3</b>	<b>Decisões de projeto</b>	<b>8</b>
<b>4</b>	<b>Compilação do código</b>	<b>9</b>
<b>5</b>	<b>Exemplos de uso</b>	<b>10</b>
5.1	Exemplo 1 - Sem erros . . . . .	10
5.2	Exemplo 2 - Com erros . . . . .	12

## 1 Introdução

Esse projeto se destina a construção de um analisador léxico, a primeira etapa de um compilador, responsável por ler o arquivo como código-fonte realizado na linguagem P- e identificar os tokens/símbolos correspondentes, além de poder tratar alguns erros de forma básica.

Este documento exhibe as decisões de projeto e os procedimentos adotados pelo grupo para o desenvolvimento do analisador, onde sua construção pode ser dividida em três etapas: decisões de projeto, construção dos autômatos e codificação.

## 2 Construção dos autômatos

Após ter decisões de projeto bem definidas, é necessário construir os autômatos que vão reconhecer as cadeias, para que assim seja possível codificá-las.

### 2.1 Autômato para reconhecimento de Identificadores

Para um correto reconhecimento dos identificadores pelo autômato é necessário que o primeiro caractere sempre seja uma letra. Caso contrário, o autômato entrará em um estado de erro, acusando que o identificador não começa com uma letra.

Em seguida, o autômato aceita tanto letras quando dígitos, transicionando para o estado de aceitação quando algum outro um símbolo conhecido pela linguagem (exceto letras e números) é inserido, ou caso contrário, para um estado de erro que acusa que o identificador é mal formado.

Caso nenhum erro ocorrer, é feita uma busca na tabela de palavras reservadas, para conferir se a palavra reconhecida não é reservada pelo compilador. Caso for encontrada a palavra na tabela, ela será considerada como uma palavra reservada, do contrário, um identificador.

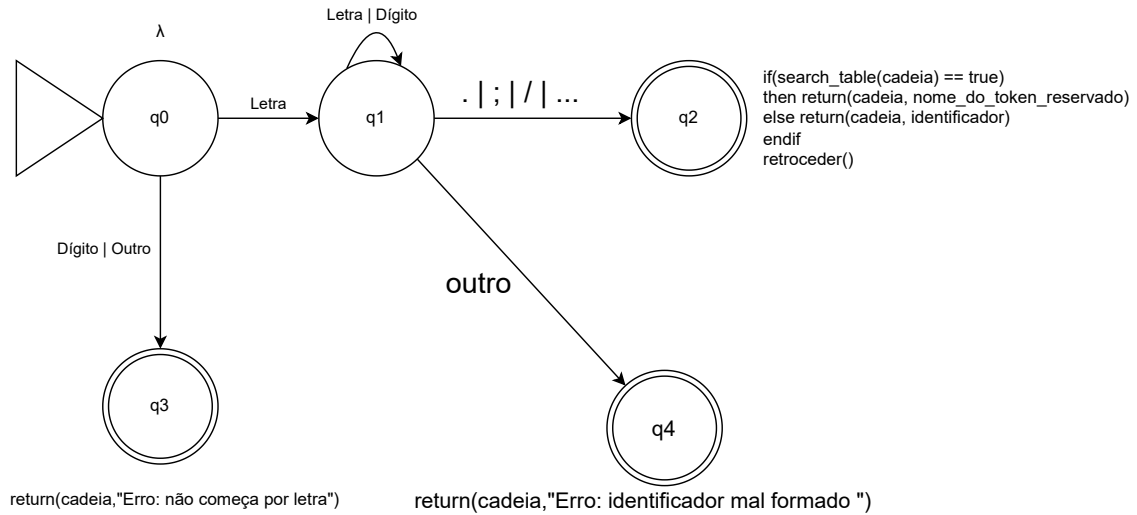


Figura 1: Autômato para reconhecimento de identificadores.

## 2.2 Autômato para reconhecimento de números inteiros e reais

Esse autômato define regras de formatação e classificação dos números inseridos, onde duas classificações são possíveis: números inteiros e números reais. Sendo que ambos os números podem aceitar símbolos negativos e positivos (+ e -).

O autômato inicialmente em  $q_0$  transiciona para o próximo estado caso um dígito for consumido, ou para erro no caso de um outro símbolo consumido. Em seguida, em  $q_1$  o autômato passa a consumir qualquer dígito, transicionando para um estado de aceitação  $q_8$ , que reconhece a cadeia como um número inteiro quando um símbolo reconhecido pela linguagem (exceto letras), para  $q_2$  quando um ponto for consumido, passando a reconhecer a cadeia apenas como real, ou para um estado de erro  $q_5$  quando um outro símbolo é inserido. Em  $q_2$ , após o ponto, o autômato passa a esperar a inserção de um dígito para passar ao estado  $q_6$ , transicionando para erro no caso de não existir um número após o ponto. Por fim, em  $q_6$ , o autômato consome qualquer dígito e transiciona para  $q_7$  quando um símbolo reconhecido pela linguagem for reconhecido (exceto letras novamente), ou para um estado de erro quando algum outro símbolo for inserido.

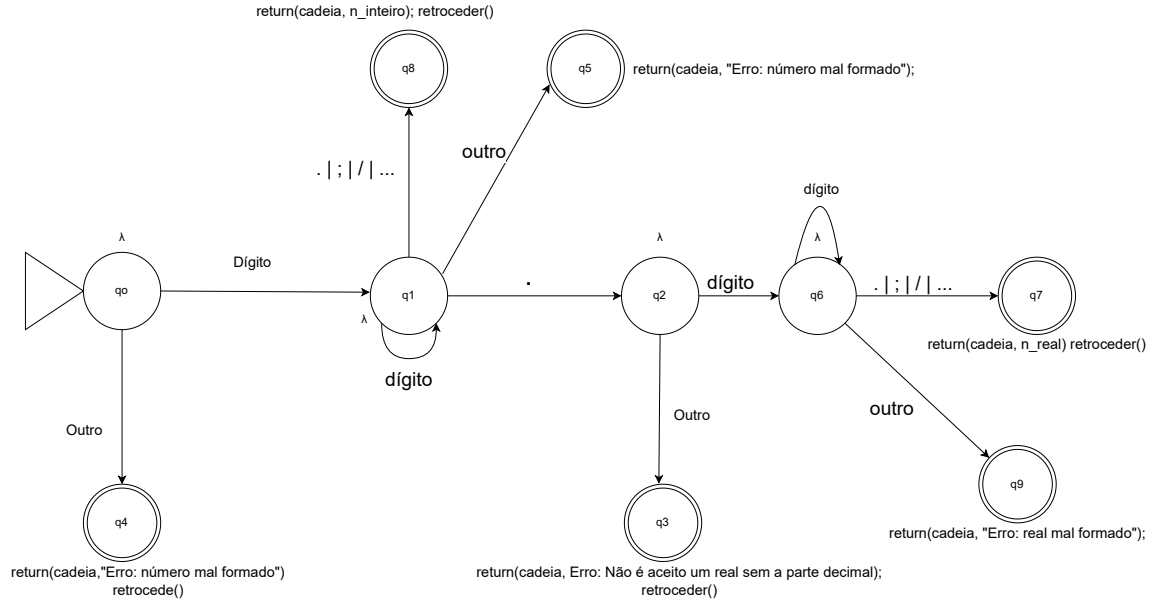


Figura 2: Autômato para reconhecimento de números reais e inteiros.

## 2.3 Autômato para reconhecimento de comentários

Esse autômato reconhece e estabelece a formatação para o uso do comentário. Inicialmente o autômato aguarda a abertura do comentário, dada pelo símbolo de chave a direita '}', caso o contrário é emitida uma mensagem de erro indicando que o comentário não foi aberto. Em seguida, o autômato começa a consumir todos os símbolos diferentes de '{' e 'n'. Após isso, caso seja encontrado um 'n' indica que o comentário não foi corretamente fechado e então é exibido uma mensagem de erro. Por fim, caso seja identificado o símbolo '}' significa que o comentário foi fechado e portanto o comentário reconhecido sem erros.

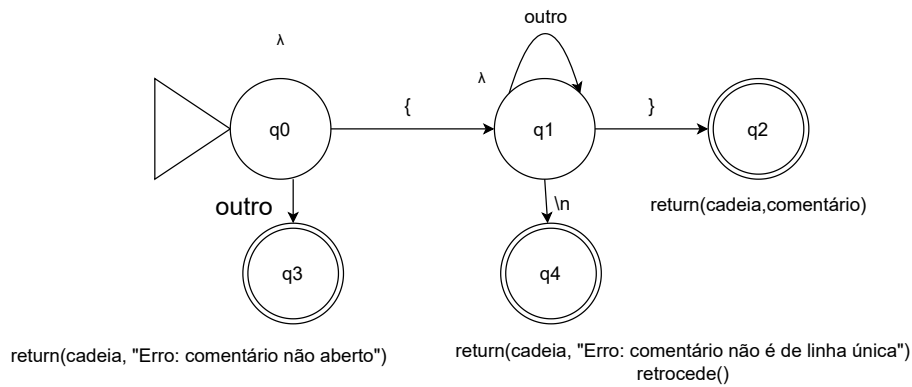


Figura 3: Autômato para reconhecimento de comentários.

## 2.4 Autômato para reconhecimento de operadores

O autômato para reconhecimento de operadores tem um funcionamento simples e uma grande quantidade de estados.

Inicialmente, caso um identificador único seja reconhecido, ou seja, qualquer identificador que não tenha composição com outro símbolo (como '+', '-', ',', entre outros), a cadeia já é aceita. Enquanto em casos que um símbolo que possua uma combinação (como '>', '<', ':', entre outros) o autômato transiciona para um estado em que, ou é consumido um símbolo que exista combinação com o consumido anteriormente (no caso do ' ' ou '=' por exemplo), ou se consome qualquer outro símbolo, indicando que o identificador é único.



O autômato final se trata de uma junção de todos os autômatos descritos anteriormente em um único autômato, com algumas adaptações.

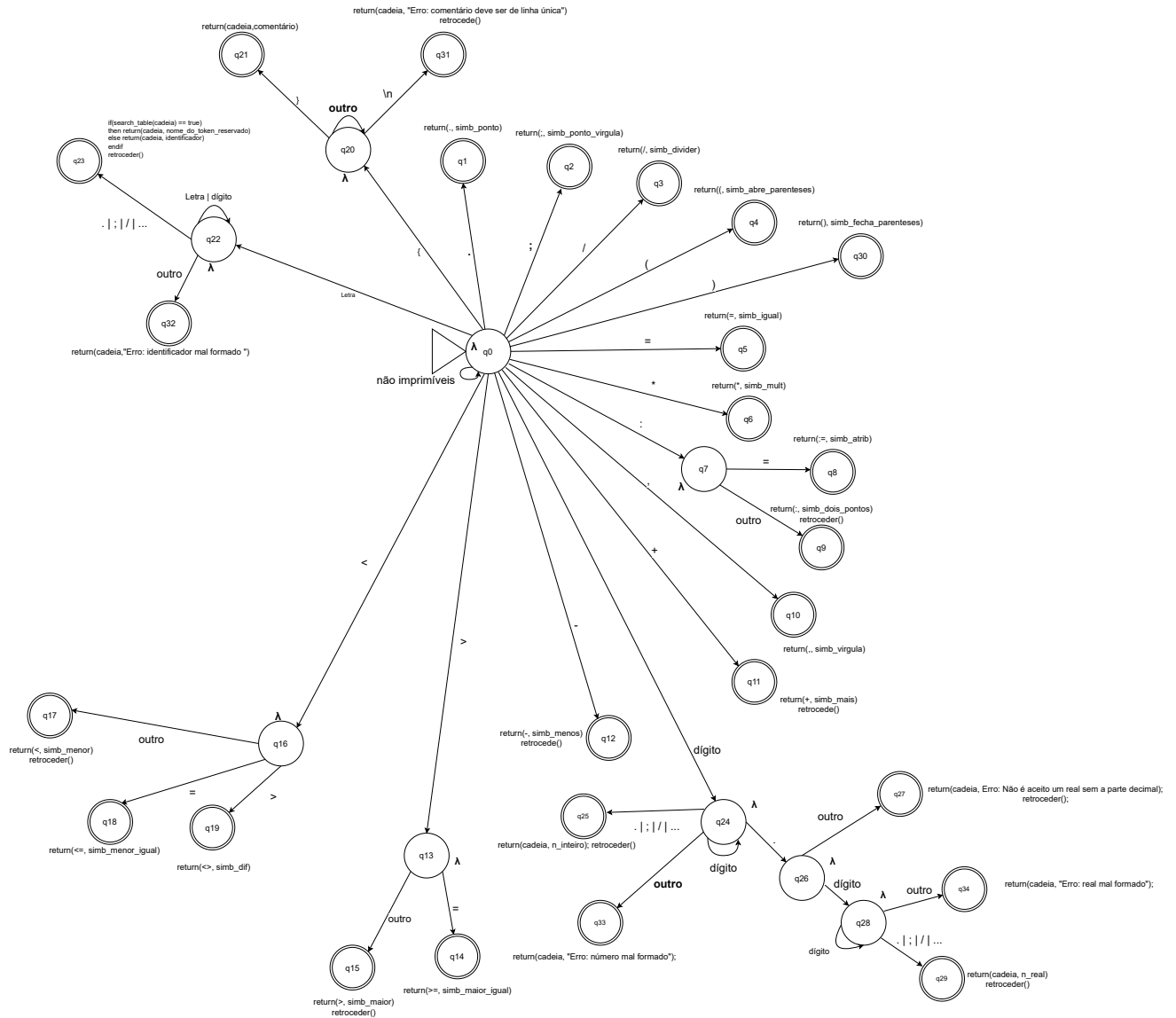


Figura 5: Autômato final

### 3 Decisões de projeto

Algumas decisões foram tomadas para que fosse possível fazer o analisador sintático. Inicialmente, juntamos todos os autômatos em um grande autômato para facilitar o desenvolvimento. A partir desse grande automato foi feita a tabela de transições. Essa tabela abrange todas as transições e apenas as transições. Ela foi feita usando uma matriz de inteiros sendo que o índice da linha refere-se



ao estado e o índice da coluna a todos os símbolos da tabela ASCII. Para saber se os estados eram finais, se tinha que retroceder e a mensagem que eles retornavam, foi feito um *array* de estados, onde o índice é relativo ao estado, sendo que foi criado uma *structure* para guardar as informações de um estado. Também foi feito um *array* de strings para armazenar as palavras reservadas da linguagem, já que são apenas 17 para o p- (contando com o for).

Para tratar os erros com mais precisão e menos generalidade, foi criado um array com os caracteres que são reconhecidos pelo estado inicial do automato. Tendo esses caracteres, em algumas transições de "outros", foi trocado para por duas transições, sendo considerado um estado final sem erro quando o elemento de parada era um dos elementos reconhecidos pelo automato principal e reconhecido como erro quando não reconhecido pelo estado inicial. Por exemplo, para reconhecer o identificador, é necessário uma transição de parada "outro", então foram criados dois estados finais, uma de erro e outro não. O que não é de erro é reconhecido quando o símbolo de parada também é reconhecido pelo estado inicial do automato, como "+", "-", ",", ".", etc. E o de erro é acontece com o novo "outro", desconsiderando os elementos reconhecidos pelo automato inicial.

Os caracteres não reconhecidos pelo automato principal não geram um estado no automato, mas no código eles são considerados como estado -1, sendo que eles retornam "Erro: carácter não reconhecido". Por facilidade, os símbolos não imprimíveis ('\\t', '\\n' e '\\r'), também são considerados estados -1, mas eles em vez retornar esse erro, eles são simplesmente ignorados (isso no estado inicial 0).

## 4 Compilação do código

É necessário alguns requisitos para a compilação do código:

- gcc
- make

Com os requisitos satisfeitos, para compilar o código fonte a partir do Windows, Linux e Mac, basta caminhar até a pasta raiz do projeto pelo terminal e executar o seguinte comando:

Em seguida para executar o código, basta usar o seguinte comando:

```
1 make
```

Código 1: Compilação do código

```
1 make run
```

Código 2: Execução do código

## 5 Exemplos de uso

Com o código compilado, podemos realizar alguns testes para testar o comportamento do analisador léxico construído. Abaixo são exibidos dois exemplos. O exemplo 1 consiste de uma entrada que não possui erros, enquanto a segunda contém erros léxicos.

### 5.1 Exemplo 1 - Sem erros

```
1 program leimprime;
2 {exemplo 1}
3 var a: real;
4 var b: integer;
5 procedure nomep(x: real);
6 var a, c: integer;
7 begin
8 read(c, a);
9 if a<x+c then
10 begin
11 a:= c+x;
12 write(a);
13 end
14 else c:= a+x;
15 end;
16 begin {programa principal}
17 read(b);
18 nomep(b);
19 end.
```

Código 3: Entrada - Exemplo 1

```

1  program, program
2  leimprime, identificador
3  ;; simb_ponto_virgula
4  {exemplo 1}, comentario
5  var, var
6  a, identificador
7  :, simb_dois_pontos
8  real, real
9  ;; simb_ponto_virgula
10 var, var
11 b, identificador
12 :, simb_dois_pontos
13 integer, integer
14 ;; simb_ponto_virgula
15 procedure, procedure
16 nomep, identificador
17 (, simb_abre_parenteses
18 x, identificador
19 :, simb_dois_pontos
20 real, real
21 ), simb_fecha_parenteses
22 ;; simb_ponto_virgula
23 var, var
24 a, identificador
25 ,, simb_virgula
26 c, identificador
27 :, simb_dois_pontos
28 integer, integer
29 ;; simb_ponto_virgula
30 begin, begin
31 read, read
32 (, simb_abre_parenteses
33 c, identificador
34 ,, simb_virgula
35 a, identificador
36 ), simb_fecha_parenteses
37 ;; simb_ponto_virgula
38 if, if
39 a, identificador
40 <, simb_menor
41 x, identificador
42 +, simb_mais
43 c, identificador
44 then, then
45 begin, begin
46 a, identificador
47 :=, simb_atrib

```

```

48  c, identificador
49  +, simb_mais
50  x, identificador
51  ;, simb_ponto_virgula
52  write, write
53  (, simb_abre_parenteses
54  a, identificador
55  ), simb_fecha_parenteses
56  ;, simb_ponto_virgula
57  end, end
58  else, else
59  c, identificador
60  :=, simb_atrib
61  a, identificador
62  +, simb_mais
63  x, identificador
64  ;, simb_ponto_virgula
65  end, end
66  ;, simb_ponto_virgula
67  begin, begin
68  {programa principal}, comentario
69  read, read
70  (, simb_abre_parenteses
71  b, identificador
72  ), simb_fecha_parenteses
73  ;, simb_ponto_virgula
74  nomep, identificador
75  (, simb_abre_parenteses
76  b, identificador
77  ), simb_fecha_parenteses
78  ;, simb_ponto_virgula
79  end, end
80  ., simb_ponto

```

Código 4: Saída - Exemplo 1

## 5.2 Exemplo 2 - Com erros

```

1  prog@ram leimprime;
2  {comentario em uma linha}
3  {comentario com quebra(invalido) :)
4  }
5  @var a: real;
6  var b: integer;
7  procedure nomep(x: real);

```

```

8  vasdfar a, c: integer;
9  begin
10 read(c);
11 a:=23;
12 a:=23#;
13 a:=23.;
14 a:=23.32;
15 a:=23.32%;
16 a:=23.3%2
17 if a<x+c then
18 begin
19 a:= c + x;
20 write(a);
21 end
22 else c:= a+x;
23 end;
24 begin {programa principal}
25 read(b);
26 nomep(b);
27 end.

```

Código 5: Entrada - Exemplo 2

```

1  prog@, Erro: identificador mal formado
2  ram, identificador
3  leimprime, identificador
4  ;; simb_ponto_virgula
5  {comentario em uma linha}, comentario
6  {comentario com quebra(invalido) :), Erro: comentario deve ser de linha unica
7  }, Caractere invalido
8  @, Caractere invalido
9  var, var
10 a, identificador
11 :, simb_dois_pontos
12 real, real
13 ;; simb_ponto_virgula
14 var, var
15 b, identificador
16 :, simb_dois_pontos
17 integer, integer
18 ;; simb_ponto_virgula
19 procedure, procedure
20 nomep, identificador
21 (, simb_abre_parenteses
22 x, identificador
23 :, simb_dois_pontos

```

```

24 real, real
25 ), simb_fecha_parenteses
26 ;, simb_ponto_virgula
27 vasdfar, identificador
28 a, identificador
29 ,, simb_virgula
30 c, identificador
31 :, simb_dois_pontos
32 integer, integer
33 ;, simb_ponto_virgula
34 begin, begin
35 read, read
36 (, simb_abre_parenteses
37 c, identificador
38 ), simb_fecha_parenteses
39 ;, simb_ponto_virgula
40 a, identificador
41 :=, simb_atrib
42 23, n_inteiro
43 ;, simb_ponto_virgula
44 a, identificador
45 :=, simb_atrib
46 23#, Erro: numero mal formado
47 ;, simb_ponto_virgula
48 a, identificador
49 :=, simb_atrib
50 23., Erro: Nao e aceito um real sem a parte decimal
51 ;, simb_ponto_virgula
52 a, identificador
53 :=, simb_atrib
54 23.32, n_real
55 ;, simb_ponto_virgula
56 a, identificador
57 :=, simb_atrib
58 23.32%, Erro: real mal formado
59 ;, simb_ponto_virgula
60 a, identificador
61 :=, simb_atrib
62 23.3%, Erro: real mal formado
63 2, n_inteiro
64 if, if
65 a, identificador
66 <, simb_menor
67 x, identificador
68 +, simb_mais
69 c, identificador
70 then, then

```

```

71  begin, begin
72  a, identificador
73  :=, simb_atrib
74  c, identificador
75  +, simb_mais
76  x, identificador
77  ;, simb_ponto_virgula
78  write, write
79  (, simb_abre_parenteses
80  a, identificador
81  ), simb_fecha_parenteses
82  ;, simb_ponto_virgula
83  end, end
84  else, else
85  c, identificador
86  :=, simb_atrib
87  a, identificador
88  +, simb_mais
89  x, identificador
90  ;, simb_ponto_virgula
91  end, end
92  ;, simb_ponto_virgula
93  begin, begin
94  {programa principal}, comentario
95  read, read
96  (, simb_abre_parenteses
97  b, identificador
98  ), simb_fecha_parenteses
99  ;, simb_ponto_virgula
100 nomep, identificador
101 (, simb_abre_parenteses
102 b, identificador
103 ), simb_fecha_parenteses
104 ;, simb_ponto_virgula
105 end, end
106 ., simb_ponto

```

Código 6: Saída - Exemplo 2