Laboratório 1 - CES41 Compiladores

13 de Março de 2019

Prof. Dr. Fábio Carneiro Mokarzel

Igor Bragaia[1]

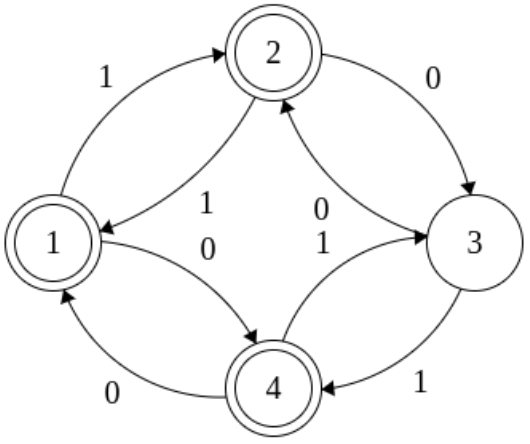
[1]Aluno de Graduação em Engenharia do Instituto Tecnológico de Aeronáutica (ITA).

E-mail: igor.bragaia@gmail.com

---

1) Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, tais que o número de dígitos 0 seja par ou o número de dígitos 1 seja par.

Para a resolução desse exercício, considerou-se inicialmente o autômato reconhecedor de cadeias sobre o alfabeto {0, 1} tal que cada estado refere-se a quantidade par ou ímpar de zeros e uns. Assim, considerando o estado inicial 1, temos



No estado 1, temos uma quantidade par de zeros e uns; em 2, uma quantidade par de zeros e ímpar e uns; em 3, uma quantidade ímpar de zeros e uns; em 4, uma quantidade ímpar de zeros e par de uns. Assim, os estados finais desejados são os estados 1,2 e 4. Finalmente, implementou-se esse autômato no programa Flex. Validou-se seu correto funcionamento por meio do teste abaixo:

Input

10
00
11
000
100001
1000011
0111100
10000110
01111001
10000111
0111101010
1000011010

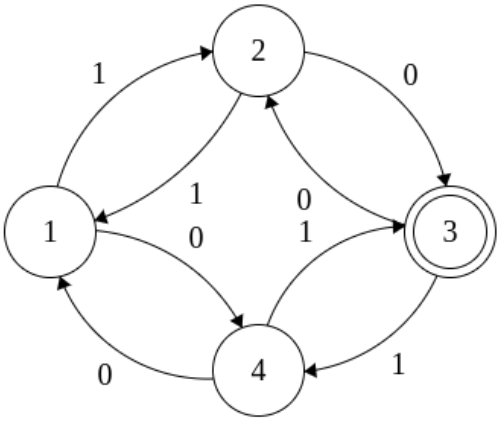
Output

10 : Rejeitada (quantidade ímpar de 1 e ímpar de 0)
00 : Aceita (quantidade par de 1 e par de 0)

11 : Aceita (quantidade par de 1 e par de 0)
000 : Aceita (quantidade par de 1 e ímpar de 0)
100001 : Aceita (quantidade par de 1 e par de 0)
1000011 : Aceita (quantidade ímpar de 1 e par de 0)
0111100 : Aceita (quantidade par de 1 e ímpar de 0)
10000110 : Rejeitada (quantidade ímpar de 1 e ímpar de 0)
01111001 : Rejeitada (quantidade ímpar de 1 e ímpar de 0)
10000111 : Aceita (quantidade par de 1 e par de 0)
0111101010 : Aceita (quantidade par de 1 e par de 0)
1000011010 : Aceita (quantidade par de 1 e par de 0)

2) Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, tais que o número de dígitos 0 seja ímpar e o número de dígitos 1 seja ímpar.

Para a resolução desse exercício, considerou-se inicialmente o autômato reconhecedor de cadeias sobre o alfabeto {0, 1} tal que cada estado refere-se a quantidade par ou ímpar de zeros e uns. Assim, considerando o estado inicial 1, temos



No estado 1, temos uma quantidade par de zeros e uns; em 2, uma quantidade par de zeros e ímpar e uns; em 3, uma quantidade ímpar de zeros e uns; em 4, uma quantidade ímpar de zeros e par de uns. Assim, o estados final desejado é o estado 3. Finalmente, implementou-se esse autômato no programa Flex. Validou-se seu correto funcionamento por meio do teste abaixo:

Input

10
00
11
000
100001
1000011
0111100
10000110
01111001
10000111
0111101010
1000011010
01110100

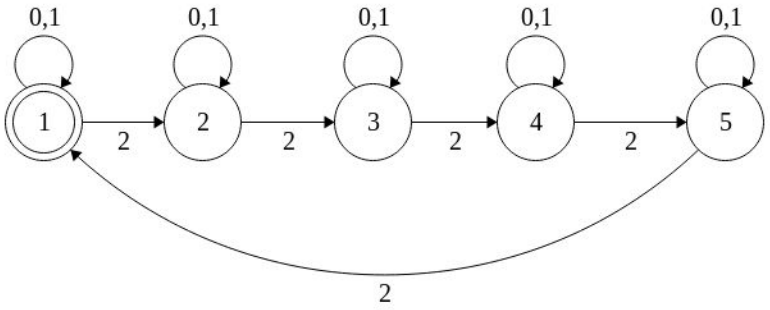
Output

10 : Aceita (quantidade ímpar de 1 e ímpar de 0)
00 : Rejeitada (quantidade par de 1 e par de 0)

11 : Rejeitada (quantidade par de 1 e par de 0)
000 : Rejeitada (quantidade par de 1 e ímpar de 0)
100001 : Rejeitada (quantidade par de 1 e par de 0)
1000011 : Rejeitada (quantidade ímpar de 1 e par de 0)
0111100 : Rejeitada (quantidade par de 1 e ímpar de 0)
10000110 : Aceita (quantidade ímpar de 1 e ímpar de 0)
01111001 : Aceita (quantidade ímpar de 1 e ímpar de 0)
10000111 : Rejeitada (quantidade par de 1 e par de 0)
0111101010 : Rejeitada (quantidade par de 1 e par de 0)
1000011010 : Rejeitada (quantidade par de 1 e par de 0)
01110100 : Rejeitada (quantidade par de 1 e par de 0)

3) Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1, 2}, tais que o número de dígitos 2 seja divisível por 5 (Obs: zero é divisível por 5).

Para a resolução desse exercício, considerou-se inicialmente o autômato reconhecedor de cadeias sobre o alfabeto {0, 1, 2}, tais que o número de dígitos 2 seja divisível por 5 (Obs: zero é divisível por 5). Considerando o estado inicial 1, temos



Nota-se que o estado final desejado é o estado 1, o qual contém todas as cadeias com uma quantidade de dígitos 2 divisível por zero, considerando zero divisível por 5. Extraindo a RegEx a partir desse autômato, obteve-se

RegEx: (0 | 1 | 2 (0 | 1)* 2 (0 | 1)* 2 (0 | 1)* 2 (0 | 1)* 2)*

Implementado o código em *Flex*, validou-se seu correto funcionamento por meio do teste abaixo:

Input

201010
1001100
201111002
220000010101
20000012012
20001110222
2000012201202
200001220120220000012201202
2000012201202200001220120220000012201202
20000122012022000012201202200001220120202

Output

201010 : Rejeitada (1 dígito 2)
1001100 : Aceita (0 dígito 2)
201111002 : Rejeitada (2 dígito 2)
220000010101 : Rejeitada (2 dígito 2)
20000012012 : Rejeitada (3 dígito 2)
20001110222 : Rejeitada (4 dígito 2)

2000012201202 : Aceita (5 dígito 2)
20000122012022000012201202 : Aceita (10 dígito 2)
200001220120220000122012022000012201202 : Aceita (15 dígito 2)
2000012201202200001220120220000122012020 2: Rejeitada (16 dígito 2)

4) Escrever um programa em *Flex* reconhecedor de cadeias sobre o alfabeto {0, 1}, com no mínimo cinco caracteres, tais que qualquer bloco de cinco caracteres consecutivos contenha no mínimo três dígitos 1.

Para a resolução desse exercício, optou-se pela não utilização do autômato pois a quantidade de estados inicial a ser minimizadas seria, no mínimo, 2^5 = 32 estados. A ideia utilizada foi garantir que qualquer bloco de cinco caracteres consecutivos contenha no mínimo três dígitos 1 conforme a entrada foi lida.

Implementado o código em *Flex*, validou-se seu correto funcionamento por meio do teste abaixo:

Input

1 11 111 1111 11111 111101
1110011001
111001101
101101101
101101101101101101

Output

1 : Rejeitada
11 : Rejeitada
111 : Rejeitada
1111 : Rejeitada
11111 : Aceita
111101 : Aceita
1110011001 : Rejeitada
111001101 : Rejeitada
101101101 : Aceita
101101101101101101 : Aceita

5) Escrever um programa em ***Flex*** para fazer análise léxica de uma mini-linguagem que contenha os seguintes átomos: identificadores (ID), constantes inteiras (CTINT), constantes reais (CTREAL), operadores aditivos (OPAD), operadores multiplicativos (OPMULT), abre e fecha-parentesis (ABPAR e FPAR), abre e fecha-chaves (ABCHAV e FCHAV), sinal de atribuição (ATRIB), vírgula e ponto-e-vírgula (VIRG e PVIRG) e ainda as palavras reservadas **program**, **var**, **int** e **real**.

Para a implementação do código em *Flex* para fazer análise léxica da uma mini-linguagem citada, utilizou-se a seguinte sequência de definições:

```
#define OPAD        1
#define OPMLT       2
#define PROGRAM     3
#define VAR         4
#define INT         5
#define REAL        6
#define ABPAR       7
#define FPAR        8
#define ABCHAV      9
#define FCHAV       10
#define ATRIB       11
```

```
#define VIRG            12
#define PVIRG           13
#define ID              14
#define CTINT           15
#define CTREAL          16
#define VEZES           1
#define DIV             2
#define MAIS            3
#define MENOS           4
```

Por fim, validou-se seu correto funcionamento por meio do teste abaixo:

Input

```
var x1 = 123, x2 = 456;
int x3 = 789, x4 = 10;
real x5 = 0.123, x6= 1.2e+3;
program(x3, x4, x5, x6){
  x3 = x3 + 1;
  x4 = x4 - 1;
  x5 = x5 * 2;
  x6 = x6 / 2;
}
```

Output

```
   texto  |   tipo   | atributo
---------------------------------
       var|         4|
        x1|        14|       x1
         =|        11|
       123|        15|      123
         ,|        12|
        x2|        14|       x2
         =|        11|
       456|        15|      456
         ;|        13|
       int|         5|
        x3|        14|       x3
         =|        11|
       789|        15|      789
         ,|        12|
        x4|        14|       x4
         =|        11|
        10|        15|       10
         ;|        13|
      real|         6|
        x5|        14|       x5
         =|        11|
     0.123|        16|0.12300
         ,|        12|
        x6|        14|       x6
         =|        11|
    1.2e+3|        16|1200.00000
         ;|        13|
```

```
program|       3|
      (|       7|
     x3|      14|      x3
      ,|      12|
     x4|      14|      x4
      ,|      12|
     x5|      14|      x5
      ,|      12|
     x6|      14|      x6
      )|       8|
      {|       9|
     x3|      14|      x3
      =|      11|
     x3|      14|      x3
      +|       1|       3
      1|      15|       1
      ;|      13|
     x4|      14|      x4
      =|      11|
     x4|      14|      x4
      -|       1|       4
      1|      15|       1
      ;|      13|
     x5|      14|      x5
      =|      11|
     x5|      14|      x5
      *|       2|       1
      2|      15|       2
      ;|      13|
     x6|      14|      x6
      =|      11|
     x6|      14|      x6
      /|       2|       2
      2|      15|       2
      ;|      13|
      }|      10|
```