Implementação de AFD e APD

Artur Barbosa Pinto
Curso: Sistemas de Informação
Faculdade Cotemig

Introdução

Este trabalho tem por objetivo demonstrar a implementação de um Autômato Finito Determinístico (AFD) e um Autômato de Pilha Determinístico (APD), ambos simulados na linguagem de programação C#, visando a aplicação prática dos fundamentos teóricos da computação.

Descrição da Linguagem

AFD – Múltiplos de 4 em binário

Em um número binário, os dois últimos dígitos representam as potências 2^1 e 2^0. Para que um número seja múltiplo de 4, ou seja, divisível por 2^2, os dois últimos dígitos devem ser 0. Logo, a linguagem reconhecida pelo AFD é o conjunto de todas as sequências binárias que terminam em 00. Exemplificando, as palavras 11011 e 1010 são rejeitadas pelo AFD, já as palavras 10100 e 11100 são aceitas pelo AFD.

APD – Palíndromos pares

A linguagem reconhecida pelo APD é o conjunto de todas as palavras que são lidas da mesma maneira tanto da esquerda para a direita quanto da direita para a esquerda, e que possuem um número par de símbolos. O APD verifica essa condição lendo a primeira metade da palavra e empilhando cada símbolo. Ao chegar na metade da palavra, passa para o estado de desempilhamento, onde para cada símbolo lido da palavra, desempilha seu símbolo respectivo. Para que a palavra seja aceita, a pilha deve estar vazia ao final da leitura, confirmando que todos os símbolos coincidiram e que a palavra tinha um comprimento par. Exemplificando, as palavras aababa e aaabba são rejeitadas, já as palavras aabbaa e abaaba são aceitas pelo APD.

Implementações

Todas as implementações foram feitas utilizando a seguinte classe Main no site OneCompiler:

```
static void Main(string[] args)
{
    Console.OutputEncoding = System.Text.Encoding.UTF8;
    while (true)
    { //menu para ficar repetindo ate o usuario sair (escolha 3)
        Console.WriteLine("------- Implementação de AFD e APD -------\n
           ");
```

```
        Console.WriteLine("Selecione o tipo do Autômato");
        Console.WriteLine("1 - Autômato Finito Determinístico (AFD)");
        Console.WriteLine("2 - Autômato de Pilha Determinístico (APD)")
            ;
        Console.WriteLine("3 - Sair");
        if (!int.TryParse(Console.ReadLine(), out int escolha))
            escolha = 0;

        switch (escolha)
        {
            case 1:
                Console.WriteLine();
                AFD(); //executando o AFD
                break;

            case 2:
                Console.WriteLine();
                APD(); //executando o APD
                break;

            case 3:
                return;

            default:
                Console.WriteLine("\nOpção inválida!\n");
                break;
        }
    }
}
```

Implementação do AFD

A lógica do autômato se baseia na regra de que para ser múltiplo de 4, os dois últimos bits do número binário devem ser 00. O autômato inicia em $q0$ e se mantêm ao ler 0, que permite cadeias de 0 sejam aceitas, já que 0 é um múltiplo de 4. Ao ler 1, o AFD muda de estado para $q1$ e se mantêm caso leia 1. Ao ler 0, o AFD transita para o estado $q2$, que retorna para $q1$ quando lê 1 ou para o estado de aceitação $q0$ ao ler 0, indicando que dois 0 foram lidos em sequência, validando o número como múltiplo de 4.

- Estados: $\{q0, q1, q2\}$
- Alfabeto: $\{0, 1\}$
- Estado inicial: $q0$
- Estado de aceitação: $q0$
- Função de transição:

```
static void AFD()
{
    try
```

Estado atual	Entrada	Próximo estado
$q0$	0	$q0$
$q0$	1	$q1$
$q1$	0	$q2$
$q1$	1	$q1$
$q2$	0	$q0$
$q2$	1	$q1$

```
4       {
5           Console.Clear(); //limpando a tela
6           string[] estados = { "q0", "q1", "q2" }; //array dos estados
7           int estadoInicial = 0; //definindo o estado inicial do automato
                (q0)
8           int estadoFinal = 0; //definindo o estado final do automato (q0
               )
9           int estadoAtual; //estado atual do automato
10
11          Console.WriteLine("Autômato Finito Determinístico (AFD)\n");
12          Console.WriteLine("Estados: {q0, q1, q2}\nAlfabeto: {0, 1}\nFun
               ção de Transição:\r\n(q0, 0) = q0\r\n(q0, 1) = q1\r\n(q1, 0)
                = q2\r\n(q1, 1) = q1\r\n(q2, 0) = q0\r\n(q2, 1) = q1\
               nEstado Inicial: q0\nEstados Finais: {q0}\n");
13          Console.WriteLine("Digite a palavra:");
14          string palavra = Console.ReadLine();
15          if (!Regex.IsMatch(palavra, @"^[01]+$")) //se a palavra nao for
                um numero binario, exibir um erro
16              throw new Exception("A palavra digitada não está em binário
                   !");
17
18          Console.WriteLine($"\nPassos da Função de Transição da palavra
               {palavra}\n");
19
20          estadoAtual = estadoInicial; //comecando pelo estado inicial
21
22          for (int i = 0; i < palavra.Length; i++)
23          {// para cada simbolo da palavra, fazer a funcao de transicao
24              estadoAtual = FuncaoTransicao(estadoAtual, palavra[i],
                   estados); //funcao de transicao
25          }
26
27          Console.WriteLine($"\nPalavra termina em {estados[estadoAtual]}
               ");
28
29          if (estadoAtual == estadoFinal) //se a palavra terminar no
               estado final, palavra aceita
30              Console.WriteLine("Palavra aceita");
31          else
32              Console.WriteLine("Palavra não aceita");
33
34          Console.WriteLine("\nProgramador responsável: Artur Barbosa
               Pinto\n");
35
```

```
36          Console.WriteLine("Aperte qualquer tecla para retornar");
37          Console.ReadKey();
38          Console.Clear(); //limpando a tela
39      }
40      catch (Exception ex)
41      {
42          Console.WriteLine("Erro: " + ex.Message + "\n"); //exibir erro
                na tela
43          Console.WriteLine("Aperte qualquer tecla para retornar");
44          Console.ReadKey();
45          Console.Clear(); //limpando a tela
46      }
47  }
48
49  static int FuncaoTransicao(int estadoAnterior, char simboloLido, string
        [] estados) //AFD
50  {//transicao dos estados
51      /*
52      (q0, 0) = q0
53      (q0, 1) = q1
54      (q1, 0) = q2
55      (q1, 1) = q1
56      (q2, 0) = q0
57      (q2, 1) = q1
58      */
59      int novoEstado = 100;
60      switch (estadoAnterior)
61      {
62          case 0: //(q0)
63              switch (simboloLido)
64              {
65                  case '0':
66                      novoEstado = 0; //(q0, 0) = q0
67                      break;
68                  case '1':
69                      novoEstado = 1; //(q0, 1) = q1
70                      break;
71              }
72              break;
73
74          case 1: //(q1)
75              switch (simboloLido)
76              {
77                  case '0':
78                      novoEstado = 2; //(q1, 0) = q2
79                      break;
80                  case '1':
81                      novoEstado = 1; //(q1, 1) = q1
82                      break;
83              }
84              break;
85
86          case 2: //(q2)
```

```
87              switch (simboloLido)
88              {
89                  case '0':
90                      novoEstado = 0; //(q2, 0) = q0
91                      break;
92                  case '1':
93                      novoEstado = 1; //(q2, 1) = q1
94                      break;
95              }
96              break;
97      }
98
99      Console.WriteLine($"\u03b4({estados[estadoAnterior]}, {simboloLido
            }) = {estados[novoEstado]}"); //imprimindo o passo da funcao de
            transicao
100     return novoEstado; //retornando o novo estado
101 }
```

Implementação do APD

A lógica do APD se baseia no comportamento da pilha que, ao empilhar a metade de uma palavra, desempilha a palavra invertida. O autômato inicia em $q0$, empilhando o primeiro símbolo na pilha e um símbolo de controle, que fica no final da pilha. Após isso, passa para o estado $q1$, onde vai empilhar cada símbolo da palavra até ler λ, significando que chegou na metade da palavra, indo para o estado $q2$, onde vai desempilhar conforme o símbolo lido da palavra. Ao ler λ novamente, significa que chegou ao final da palavra. Caso o topo da pilha seja o símbolo de controle, vai para o estado de aceitação $q3$, caso contrário, a palavra não é um palíndromo de comprimento par.

- Estados: {$q0$, $q1$, $q2$, $q3$}
- Alfabeto: {a, b}
- Alfabeto da pilha: {X, Z, F}
- Estado inicial: $q0$
- Estado de aceitação: $q3$
- Função de transição:

  δ(q0, a, λ) = [q1, XF]
  δ(q0, b, λ) = [q1, ZF]
  δ(q1, a, λ) = [q1, X]
  δ(q1, b, λ) = [q1, Z]
  δ(q1, λ, λ) = [q2, λ]
  δ(q2, a, X) = [q2, λ]
  δ(q2, b, Z) = [q2, λ]
  δ(q2, λ, F) = [q3, λ]

```
static void APD()
{
    try
    {
        Console.Clear(); //limpando a tela
        string[] estados = { "q0", "q1", "q2", "q3" }; //array dos
           estados
        int estadoInicial = 0; //definindo o estado inicial do automato
            (q0)
        int estadoFinal = 3; //definindo o estado final do automato (q3
           )
        int estadoAtual; //estado atual do automato
        int estadoTemp; //estado temporario para execucao do codigo
        string palavraAux; //palavcra temporaria para execucao do
           codigo

        Console.WriteLine("Autômato de Pilha Determinístico (APD)\n");
        Console.WriteLine("Estados: {q0, q1, q2, q3}\nAlfabeto: {a, b}\
           nAlfabeto da pilha: {F, X, Z}\nFunção de Transição:\r\n(q0,
           a, λ) = [q1, XF]\r\n(q0, b, λ) = [q1, ZF]\r\n(q1, a, λ) = [
           q1, X]\r\n(q1, b, λ) = [q1, Z]\r\n(q1, λ, λ) = [q2, λ]\r\n(
           q2, a, X) = [q2, λ]\r\n(q2, b, Z) = [q2, λ]\r\n(q2, λ, F) =
           [q3, λ]\nEstado Inicial: q0\nEstados Finais: {q3}\n");
        Console.WriteLine("Digite a palavra:");
        string palavra = Console.ReadLine();
        if (!Regex.IsMatch(palavra, @"^[ab]+$")) //se a palavra não
           pertencer ao alfabeto, exibir um erro
            throw new Exception("A palavra digitada contém símbolos que
                não pertencem ao alfabeto!");

        palavraAux = palavra.Insert(palavra.Length / 2, "λ"); //
           inserindo lambda no meio e no final da palavra
        palavraAux = palavraAux.Insert(palavraAux.Length, "λ");

        Pilha p = new Pilha(palavraAux.Length); //criando uma nova
           pilha

        Console.WriteLine($"\nPassos da Função de transição da palavra
           {palavra}\n");
        estadoAtual = estadoInicial; //comecando do estado inicial
        Console.WriteLine($"\u22a2 [{estados[estadoAtual]}, {palavra},
           {p.ImprimirPilha()}]"); //imprimindo na tela o comeco da
           funcao de transicao

        for (int i = 0; i < palavraAux.Length; i++)
        {// para cada símbolo da palavra, fazer a funcao de transicao
            if (palavraAux[i] != 'λ') //removendo cada simbolo da
               palavra para exibir a funcao de transicao
                palavra = palavra.Remove(palavra.IndexOf(palavraAux[i])
                    , 1);

            estadoTemp = FuncaoTransicao(estadoAtual, palavraAux[i],
               estados, p, (palavra.Length == 0 ? "λ" : palavra)); //
```

```
                pegando o estado atual do automato
35              if (estadoTemp == 100) //estado 100 -> sem funcao de
                    transicao
36              {
37                  Console.WriteLine($"\nNão há transição para \u03b4({
                        estados[estadoAtual]}, {palavraAux[i]}, {p.Topo()})"
                        ); //mostrando a funcao que nao existe
38                  Console.WriteLine("Palavra não aceita");
39                  Console.WriteLine("\nProgramador responsável: Artur
                        Barbosa Pinto\n");
40                  Console.WriteLine("Aperte qualquer tecla para retornar"
                        );
41                  Console.ReadKey();
42                  Console.Clear(); //limpando a tela
43                  return;
44              }
45              estadoAtual = estadoTemp; //atualizando o estado atual
46          }
47
48          Console.WriteLine($"\nPalavra termina em {estados[estadoAtual]}
                ");
49
50          if (estadoAtual == estadoFinal) //se a palavra terminar no
                estado final, palavra aceita
51              Console.WriteLine("Palavra aceita");
52          else
53              Console.WriteLine("Palavra não aceita");
54
55          Console.WriteLine("\nProgramador responsável: Artur Barbosa
                Pinto\n");
56
57          Console.WriteLine("Aperte qualquer tecla para retornar");
58          Console.ReadKey();
59          Console.Clear(); //limpando a tela
60      }
61      catch (Exception ex)
62      {
63          Console.WriteLine("Erro: " + ex.Message + "\n"); //exibir erro
                na tela
64          Console.WriteLine("Aperte qualquer tecla para retornar");
65          Console.ReadKey();
66          Console.Clear();
67      }
68  }
69
70  static int FuncaoTransicao(int estadoAnterior, char simboloLido, string
        [] estados, Pilha pilha, string palavra) //APD
71  {//transicao dos estados
72      /*
73      (q0, a, λ) = [q1, XF]
74      (q0, b, λ) = [q1, ZF]
75      (q1, a, λ) = [q1, X]
76      (q1, b, λ) = [q1, Z]
```

```
77      (q1, λ, λ) = [q2, λ]
78      (q2, a, X) = [q2, λ]
79      (q2, b, Z) = [q2, λ]
80      (q2, λ, F) = [q3, λ]
81      */
82
83      int novoEstado = 100;
84      switch (estadoAnterior)
85      {
86          case 0: //q0
87              switch (simboloLido)
88              {
89                  case 'a':
90                      pilha.Empilhar('F');
91                      pilha.Empilhar('X');
92                      novoEstado = 1; //(q0, a, λ) = [q1, XF]
93                      break;
94
95                  case 'b':
96                      pilha.Empilhar('F');
97                      pilha.Empilhar('Z');
98                      novoEstado = 1; //(q0, b, λ) = [q1, ZF]
99                      break;
100             }
101             break;
102
103         case 1: //q1
104             switch (simboloLido)
105             {
106                 case 'a':
107                     pilha.Empilhar('X');
108                     novoEstado = 1; //(q1, a, λ) = [q1, X]
109                     break;
110
111                 case 'b':
112                     pilha.Empilhar('Z');
113                     novoEstado = 1; //(q1, b, λ) = [q1, Z]
114                     break;
115
116                 case 'λ':
117                     novoEstado = 2; //(q1, λ, λ) = [q2, λ]
118                     break;
119             }
120             break;
121         case 2: //q2
122             switch (simboloLido)
123             {
124                 case 'a':
125                     if (pilha.Topo() == 'X') //(q2, a, X) = [q2, λ]
126                     {
127                         pilha.Desempilhar();
128                         novoEstado = 2;
129                     }
```

```
                    break;

                case 'b':
                    if (pilha.Topo() == 'Z') //(q2, b, Z) = [q2, λ]
                    {
                        pilha.Desempilhar();
                        novoEstado = 2;
                    }
                    break;

                case 'λ':
                    if (pilha.Topo() == 'F') //(q2, λ, F) = [q3, λ]
                    {
                        pilha.Desempilhar();
                        novoEstado = 3;
                    }
                    break;
            }
            break;
    }

    if (novoEstado == 100)
        return novoEstado; //se nao tiver funcao de transicao, retornar
            o estado 100

    Console.WriteLine($"\u22a2 [{estados[novoEstado]}, {palavra}, {
       pilha.ImprimirPilha()}]"); //imprimindo o passo da funcao de
       transicao
    return novoEstado; //retornadno o estado atual
}
```

Exemplos de Execução

1. Utilizando a palavra 10101 no AFD:

```
Passos da Função de Transição da palavra 10101

δ (q0, 1) = q1
δ (q1, 0) = q2
δ (q2, 1) = q1
δ (q1, 0) = q2
δ (q2, 1) = q1

Palavra termina em q1
Palavra não aceita
```

   Resultado final: Palavra rejeitada.

2. Utilizando a palavra 101100 no AFD:

```
Passos da Função de Transição da palavra 101100

δ (q0, 1) = q1
δ (q1, 0) = q2
```

```
δ(q2, 1) = q1
δ(q1, 1) = q1
δ(q1, 0) = q2
δ(q2, 0) = q0

Palavra termina em q0
Palavra aceita
```

Resultado final: Palavra aceita.

3. Utilizando a palavra abba no APD:

```
Passos da Função de Transição da palavra abba

   [q0, abba, λ]
⊢ [q1, bba, XF]
⊢ [q1, ba, ZXF]
⊢ [q2, ba, ZXF]
⊢ [q2, a, XF]
⊢ [q2, λ, F]
⊢ [q3, λ, λ]

Palavra termina em q3
Palavra aceita
```

Resultado final: Palavra aceita.

4. Utilizando a palavra abbb no APD:

```
Passos da Função de Transição da palavra abbb

   [q0, abbb, λ]
⊢ [q1, bbb, XF]
⊢ [q1, bb, ZXF]
⊢ [q2, bb, ZXF]
⊢ [q2, b, XF]

Não há transição para δ(q2, b, X)
Palavra não aceita
```

Resultado final: Palavra rejeitada.

Referências

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). **Introduction to Automata Theory, Languages, and Computation**.
Sipser, M. (2012). **Introduction to the Theory of Computation**.
Sites de compilação: `https://onecompiler.com`