Implementação de AFD e APD

Nicolas Samuel
Curso: Ciência da Computação
Faculdade Cotemig

6 de junho de 2025

Introdução

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

Descrição da Linguagem

AFD – Múltiplos de 4 em binário

A linguagem reconhecida por este AFD consiste em cadeias formadas pelos símbolos `0` e `1`, que representam números inteiros em binário divisíveis por 4. Um número binário é múltiplo de 4 quando termina com dois zeros, ou seja, seu sufixo é `00`. O autômato simula a leitura desses números bit a bit e determina se a palavra atende à regra.

Exemplos de palavras aceitas:

- `100`
- `1000`
- `1100`
- `10000`

Exemplos de palavras rejeitadas:

- `1`
- `10`
- `11`
- `101`
- `111`

APD – Palíndromos pares

A linguagem reconhecida por este APD é o conjunto de todas as palavras palíndromas de comprimento par formadas pelos símbolos `a` e `b`. Uma palavra palíndroma é aquela que pode ser lida da esquerda para a direita ou da direita para a esquerda com o mesmo resultado. No entanto, o APD implementado rejeita automaticamente palavras de tamanho ímpar, mesmo que sejam palíndromos, pois o símbolo central não encontra correspondência.

Durante a execução, o autômato empilha e desempilha após comparar os símbolos opostos da palavra — por exemplo, o primeiro símbolo com o último, o segundo com o penúltimo e assim por diante. Se, ao final da execução, não restar nada na pilha e a palavra tiver sido totalmente lida, significa que ela foi aceita pelo autômato.

Exemplos de palavras aceitas:

- `aa`
- `abba`
- `baab`
- `aabbaa`

Exemplos de palavras rejeitadas:

- `a` (ímpar)
- `aba` (ímpar, apesar de ser palíndromo)
- `abb` (não é palíndromo par)
- `abab` (não é palíndromo)

Implementação do AFD

```python
is_accepted = False # Varivel para identificar se a palavra foi aceita.

def main():
    try:
        palavra = input("Digite a palavra (0, 1):\n") # Entrada para o autmato ler a palavra.
        if len(palavra) > 0:
            afd(palavra)
        else: # Caso o usurio  no  digite nada.
            print("Digite algo!")
            quit()
    except:
        print("Digite apenas 0 ou 1") # Mensagem de erro para caso o usurio digite algo diferente de 0 ou 1.
        quit()

def afd(palavra):
    vetor = [int(c) for c in palavra] # Vetor com cada caractere da palavra.

    nao_aceita = []
    for i in vetor:
        if i not in {0, 1}: # Caso a palavra tenha algum elemento diferente de 0 ou 1.
            nao_aceita.append(i)
    if len(nao_aceita) != 0:
```

```
        print(f"A palavra no pode conter {nao_aceita}!!!")
        quit()

    q1(vetor) # Chamada do estado inicial.

    if is_accepted: # Mensagem para mostrar se a palavra foi aceita ou no.
        print(f"A palavra {palavra} foi aceita")
        print("Programador: Nicolas Samuel 72300515")
    else:
        print(f"A palavra {palavra} foi rejeitada")
        print("Programador: Nicolas Samuel 72300515")

# Estado inicial
def q1(vetor):
    func_transicao() # Imprime na tela a funo de  transio .
    if vetor: # Verifica se a palavra j foi totalmente lida.
        if vetor[0] == 0: # Identifica se o caractere  0; se sim, volta para o estado
        ↪inicial.
            print(get_palavra(vetor))
            print(f" (q1, {vetor[0]}) = q1")
            print(f"=======================")
            vetor.pop(0)
            q1(vetor)

        elif vetor[0] == 1: # Identifica se o caractere  1; se sim, passa para o prximo
        ↪estado.
            print(get_palavra(vetor))
            print(f" (q1, {vetor[0]}) = q2")
            print(f"=======================")
            vetor.pop(0)
            q2(vetor)
    else: # Se a palavra terminar aqui, ela  rejeitada, pois q1 no  um estado de  aceitao .
        is_accepted = False

def q2(vetor):
    global is_accepted

    if vetor:
        if vetor[0] == 1: # Identifica se o caractere  1; se sim, volta para o estado atual.
            print(get_palavra(vetor))
            print(f" (q2, {vetor[0]}) = q2")
            print(f"=======================")
            vetor.pop(0)
            q2(vetor)

        elif vetor[0] == 0: # Se o caractere  0, avana para o prximo estado.
            print(get_palavra(vetor))
            print(f" (q2, {vetor[0]}) = q3")
            print(f"=======================")
            vetor.pop(0)
            q3(vetor)

    else: # Se a palavra terminar aqui, ela  rejeitada, pois q2 no  um estado de  aceitao .
        is_accepted = False

def q3(vetor):
    global is_accepted

    if vetor:
        if vetor[0] == 1: # Identifica se o caractere  1; se sim, volta para o estado q2.
```

```
81            print(get_palavra(vetor))
82            print(f" (q3, {vetor[0]}) = q2")
83            print(f"=======================")
84            vetor.pop(0)
85            q2(vetor)
86
87        elif vetor[0] == 0: # Identifica se o caractere  0; se sim, avana para o prximo
          ↪estado.
88            print(get_palavra(vetor))
89            print(f" (q3, {vetor[0]}) = q4")
90            print(f"=======================")
91            vetor.pop(0)
92            q4(vetor)
93
94    else: # Se a palavra terminar aqui, ela  rejeitada, pois q3 no  um estado de  aceitao .
95        is_accepted = False
96
97# Estado de  aceitao
98def q4(vetor):
99    global is_accepted
100
101    if vetor:
102        if vetor[0] == 0: # Identifica se o caractere  0; se sim, volta para o estado atual.
103            print(get_palavra(vetor))
104            print(f" (q4, {vetor[0]}) = q4")
105            print(f"=======================")
106            vetor.pop(0)
107            q4(vetor)
108
109        elif vetor[0] == 1: # Identifica se o caractere  1; se sim, volta para o estado q2.
110            print(get_palavra(vetor))
111            print(f" (q4, {vetor[0]}) = q2")
112            print(f"=======================")
113            vetor.pop(0)
114            q2(vetor)
115    else: # Se a palavra terminar aqui (no estado q4), ela  aceita, pois q4  um estado de
    ↪aceitao.
116        is_accepted = True
117
118def get_palavra(palavra): # Funo  para converter a lista de caracteres de volta para uma
  ↪string.
119    palavra_inteira = ""
120    for i in palavra:
121        palavra_inteira += str(i)
122    if palavra_inteira == "":
123        return " "
124    else:
125        return palavra_inteira
126
127def func_transicao(): # Funo  para exibir a  funo  de  transio  do AFD.
128    print(f"+=======================+")
129    print(f" Funo  de  transio :")
130    print(" (q1, 0) = q1;")
131    print(" (q1, 1) = q2;")
132    print(" (q2, 1) = q2;")
133    print(" (q2, 0) = q3;")
134    print(" (q3, 1) = q2;")
135    print(" (q3, 0) = q4;")
136    print(" (q4, 1) = q2;")
137    print(" (q4, 0) = q4;")
```

```
    print(f"+======================+")
main()
```

Listing 1: Código do AFD - Múltiplos de 4 em binário

Implementação do APD

```
class Pilha: # Classe da pilha.
    def __init__(self): # Construtor.
        self.items = []

    def push(self, item): # Empilha um item.
        self.items.append(item)

    def pop(self): # Desempilha um item.
        if not self.esta_vazia():
            return self.items.pop()
        return None

    def topo(self): # Retorna o elemento do topo da pilha.
        if not self.esta_vazia():
            return self.items[-1]
        return None

    def esta_vazia(self): # Retorna True se a pilha estiver vazia, False caso contrrio.
        return len(self.items) == 0

def main():
    palavra = input("Digite uma palavra com tamanho par (a, b):\n") # Entrada para o autmato
    ↪ ler a palavra.
    array = list(palavra) # Array para separar cada caractere.
    if len(palavra) > 0:
        if not all(char in ('a', 'b') for char in array): # Verifica se a palavra contm
        ↪apenas 'a' ou 'b'.
            print("A palavra s pode conter 'a' ou 'b'!\n")
            quit()
        else:
            is_accepted = apd(array) # Chama a execuo do APD e armazena o resultado (True ou
            ↪ False).

        if is_accepted: # Mensagem para mostrar se a palavra foi aceita ou no.
            print(f'A palavra "{palavra}" foi aceita!')
            print("Programador: Nicolas Samuel 72300515")
        else:
            print(f'A palavra "{palavra}" foi rejeitada!')
            print("Programador: Nicolas Samuel 72300515")
    else: # Caso o usurio no digite nada.
        print("Digite algo!")
        quit()

def apd(palavra):
    pilha = Pilha() # Instancia a pilha.
    palavra_temp = palavra.copy() # Cpia da palavra para fins de exibio.

    tamanho = len(palavra)
    # Variveis para manipular a posio dos caracteres.
    posicao_atual = 0
    posicao_reversa = tamanho - 1
```

```
49
50      print(f"Estado inicial (pilha vazia) [{get_palavra(palavra)},{pilha.items}]")
51      print("================================================================")
52
53      while posicao_atual < posicao_reversa: # Processa a palavra at que os ponteiros de posio
        ↪ se encontrem.
54
55          if palavra[posicao_atual] == "a": # Empilha 'A' se o caractere atual for 'a'.
56              pilha.push("A")
57              print("Empilha A")
58              print(f"Palavra: {get_palavra(palavra_temp)}")
59              palavra_temp[posicao_atual] = ""
60              print(f"Lendo 'a' na posio {posicao_atual + 1} [{get_palavra(palavra_temp)},{
                ↪pilha.items}]")
61              print("================================================================")
62          else:
63              pilha.push("B") # Empilha 'B' se o caractere atual for 'b'.
64              print("Empilha B")
65              print(f"Palavra: {get_palavra(palavra_temp)}")
66              palavra_temp[posicao_atual] = ""
67              print(f"Lendo 'b' na posio {posicao_atual + 1} [{get_palavra(palavra_temp)},{
                ↪pilha.items}]")
68              print("================================================================")
69
70
71          if palavra[posicao_reversa] == "a" and pilha.topo() == "A": # Desempilha se o
            ↪caractere reverso corresponder.
72              pilha.pop()
73              print("Desempilha A")
74              print(f"Palavra: {get_palavra(palavra_temp)}")
75              palavra_temp[posicao_reversa] = ""
76              print(f"Lendo 'a' na posio {posicao_reversa + 1} [{get_palavra(palavra_temp)},{
                ↪pilha.items}]")
77              print("================================================================")
78
79          elif palavra[posicao_reversa] == "b" and pilha.topo() == "B": # Desempilha se o
            ↪caractere reverso corresponder.
80              pilha.pop()
81              print("Desempilha B")
82              print(f"Palavra: {get_palavra(palavra_temp)}")
83              palavra_temp[posicao_reversa] = ""
84              print(f"Lendo 'b' na posio {posicao_reversa + 1} [{get_palavra(palavra_temp)},{
                ↪pilha.items}]")
85              print("================================================================")
86          else:
87              return False # Se no corresponder, a palavra rejeitada.
88
89          # Atualiza os ponteiros de posio.
90          posicao_atual += 1
91          posicao_reversa -= 1
92
93      # Se os ponteiros se encontram (posicao_atual == posicao_reversa),
94      # significa que a palavra tem comprimento mpar.
95      if posicao_atual == posicao_reversa:
96          return False # Rejeita, pois a linguagem s aceita palavras de comprimento par.
97
98      # Verifica se o caractere reverso corresponde.
99      if palavra[posicao_atual] != palavra[posicao_reversa]:
100         return False # Retorna o resultado da aceitao.
101
```

```
    # Verifica se a pilha est vazia no final.
    if not pilha.esta_vazia():
        return False # Retorna o resultado da aceitao.

    # Se a  posio  atual ultrapassou a reversa, a palavra foi lida por completo e  par.
    if posicao_atual > posicao_reversa:
        return True # Retorna o resultado da aceitao.

def get_palavra(palavra): # Funo para converter a lista de caracteres de volta para uma string.
    palavra_inteira = ""
    for i in palavra:
        palavra_inteira += i
    if palavra_inteira == "":
        return " "
    else:
        return palavra_inteira


main()
```

Listing 2: Código do APD - Palíndromos pares com a e b

Exemplos de Execução

AFD – Múltiplos de 4 em binário

Palavra: 100

- Estado atual: **q1**, lê `1` → vai para **q2**
- Estado atual: **q2**, lê `0` → vai para **q3**
- Estado atual: **q3**, lê `0` → vai para **q4**

Estado final: **q4** (estado de aceitação)
Resultado: Aceita

Palavra: 101

- Estado atual: **q1**, lê `1` → vai para **q2**
- Estado atual: **q2**, lê `0` → vai para **q3**
- Estado atual: **q3**, lê `1` → vai para **q2**

Estado final: **q2** (não é estado de aceitação)
Resultado: Rejeitada

APD – Palíndromos pares

Palavra: abba

Passo a passo:

- Lê 'a' (posição 1) → empilha 'A' → Pilha: [A]
- Lê 'a' (posição 4, reversa) → corresponde → desempilha → Pilha: [λ]
- Lê 'b' (posição 2) → empilha 'B' → Pilha: [B]
- Lê 'b' (posição 3, reversa) → corresponde → desempilha → Pilha: [λ]

Resultado final:
Palavra: $\lambda\lambda\lambda\lambda$
Pilha: [λ]
Palavra é par e a pilha ficou vazia.
Resultado: Aceita

Palavra: aba
Passo a passo:

- Lê 'a' (posição 1) → empilha 'A' → Pilha: [A]
- Lê 'a' (posição 3, reversa) → corresponde → desempilha → Pilha: [A]

Resultado final:
Palavra: λbλ
Pilha: [λ]
Palavra é ímpar e sobrou um símbolo, logo ela será rejeitada.
Resultado: Rejeitada

Uso de IA

Prompt:

Quero usar a seguinte palavra como exemplo: abba. O APD vai ler o primeiro símbolo, que é "a", e vai empilhar "A"na pilha. Se o reverso — ou seja, o outro "a", que está na posição 3 da palavra (abb"a") — for igual ao primeiro símbolo, ele vai desempilhar. Caso contrário, ele vai recusar a palavra. Se a palavra for ímpar, por exemplo "aba", ele vai ler e manipular a pilha normalmente com o primeiro elemento e seu reverso, mas vai sobrar um elemento no meio. Isso significa que a palavra é ímpar, logo deverá ser recusada, mesmo sendo um palíndromo. Eu quero que ele leia qualquer palavra formada pelos símbolos a e b. O meio será identificado automaticamente durante o processamento da palavra, quando sobrar um único caractere no centro. Exemplo: abb"a"bba — esse "a"do meio sobrou, então a palavra deve ser recusada.

Finalidade da Resposta:
A finalidade da resposta é implementar um Autômato de Pilha Determinístico (APD) que aceita apenas palavras palíndromas de tamanho par compostas pelos símbolos a e b. Ele empilha os primeiros caracteres e, simultaneamente, verifica os caracteres correspondentes do final. Se sobrar um caractere no meio (palavra ímpar) ou a pilha não esvaziar corretamente, a palavra é 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`