André Felipe Mireski

Prática Aula 02
Autômatos Finitos e Determinísticos

Relatório técnico de atividade prática solicitado pelo professor Rogério Aparecido Gonçalves na disciplina de Teoria da Computação do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Universidade Tecnológica Federal do Paraná – UTFPR

Departamento Acadêmico de Computação – DACOM

Bacharelado em Ciência da Computação – BCC

Campo Mourão

Abril / 2025

Resumo

Atividade prática com o intuito de fixar o conteúdo da aula sobre Autômatos Finitos e Determinísticos. Para tanto, foram realizados 7 exercícios, onde em cada um foi criado um AFD para reconhecer uma dada linguagem, primeiro, montando-o no JFLAP e depois criando um código python para testá-los usando a lib `automata-lib`.

Palavras-chave: Teoria da Computação. Autômatos Finitos e Determinístcos

Sumário

1 Exercícios

Nessa seção se encontram as imagens dos autômatos feitas no `JFLAP` e os testes feitos em python usando a lib `automata-lib`.

O código, com os resultados, se encontra disponível no repositório do github[1].

1.1 AFD M_4 que reconhece $L_4 = \{ba^n ba \mid n \geq 0\}$

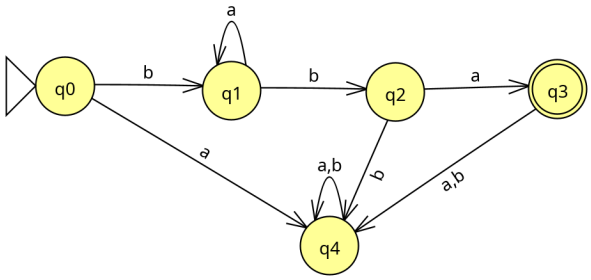

Figura 1 – AFD - Exercício 01

Listing 1 – AFD M_4 em Python com a biblioteca `automata-lib`

```python
from automata.fa.dfa import DFA

dfa01 = DFA(
    states={"q0", "q1", "q2", "q3", "q4"},
    input_symbols={"a", "b"},
    transitions={
        "q0": {"a": "q4", "b": "q1"},
        "q1": {"a": "q1", "b": "q2"},
        "q2": {"a": "q3", "b": "q4"},
        "q3": {"a": "q4", "b": "q4"},
        "q4": {"a": "q4", "b": "q4"},
    },
    initial_state="q0",
```

[1] https://github.com/afmireski/BCC5003-atividades-praticas/tree/main/aula02_automatos_finitos_e_deterministicos

```
    final_states={"q3"},
)

test_inputs = [
    "ba",
    "baba",
    "babab",
    "bababab",
    "babababa",
    "abba",
    "bbab",
    "bab",
    "baaa",
    "bbbaaa",
    "bba",
    "baaaba",
    "baaaaaaaaaba"
]

for input_str in test_inputs:
    if dfa01.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.2 AFD M_5 que reconhece $L_5 = \{x \in \{a,b\}^* \mid |x| \bmod 3 = 0\}$

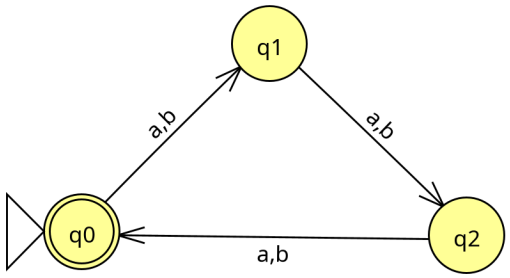

Figura 2 – AFD - Exercício 02

Listing 2 – AFD M_5 em Python com a biblioteca `automata-lib`

```
from automata.fa.dfa import DFA

dfa02 = DFA(
    states={"q0", "q1", "q2"},
    input_symbols={"a", "b"},
    transitions={
        "q0": {"a": "q1", "b": "q1"},
        "q1": {"a": "q2", "b": "q2"},
        "q2": {"a": "q0", "b": "q0"},
    },
    initial_state="q0",
    final_states={"q0"},
)

test_inputs = [
    "aaa",
    "bbb",
    "ababab",
    "bababababb",
    "aabaab",
    "a",
    "b",
```

```
    "ab",
    "ba",
    "aabb",
    "bbaa",
    "bbaab",
]

for input_str in test_inputs:
    if dfa02.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.3 AFD M_6 que reconhece $L_6 = \{w \in \{a,b\}^* \mid (|w|_a + |w|_b) \bmod 2 = 0\}$

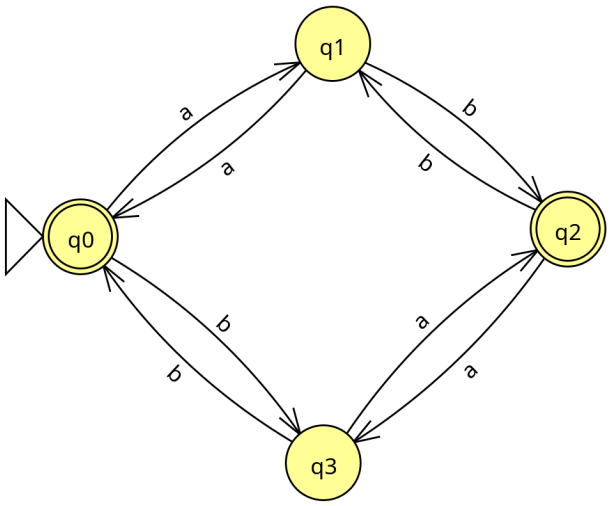

Figura 3 – AFD - Exercício 03

Listing 3 – AFD M_6 em Python com a biblioteca `automata-lib`

```
from automata.fa.dfa import DFA

dfa03 = DFA(
    states={"q0", "q1", "q2", "q3"},
    input_symbols={"a", "b"},
    transitions={
        "q0": {"a": "q1", "b": "q3"},
        "q1": {"a": "q0", "b": "q2"},
```

```
        "q2": {"a": "q3", "b": "q1"},
        "q3": {"a": "q2", "b": "q0"},
    },
    initial_state="q0",
    final_states={"q0", "q2"},
)

test_inputs = [
    "aaa", # R
    "bbb",  # R
    "abababab", # A
    "bababababbba", # A
    "aabaab", # A
    "a", # R
    "b", # R
    "ab", # A
    "ba", # A
    "aabb", # A
    "bbaa", # A
    "bbaab", # R
]

for input_str in test_inputs:
    if dfa03.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.4 AFD M_7 que reconhece $L_7 = \{a^m b^n \mid m, n \geq 0 \wedge (m+n) \bmod 2 = 0\}$

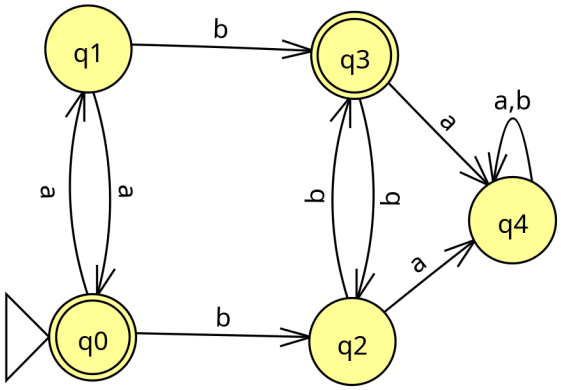

Figura 4 – AFD - Exercício 04

Listing 4 – AFD M_7 em Python com a biblioteca `automata-lib`

```python
from automata.fa.dfa import DFA

dfa04 = DFA(
    states={"q0", "q1", "q2", "q3", "q4"},
    input_symbols={"a", "b"},
    transitions={
        "q0": {"a": "q1", "b": "q2"},
        "q1": {"a": "q0", "b": "q3"},
        "q2": {"a": "q4", "b": "q3"},
        "q3": {"a": "q4", "b": "q2"},
        "q4": {"a": "q4", "b": "q4"},
    },
    initial_state="q0",
    final_states={"q0", "q3"},
)

test_inputs = [
    "aa", # A
    "bb",  # A
    "aaab", # A
    "bbba", # R
    "ababab", # R
```

```
    "a", # R
    "b", # R
    "ba", # R
    "bbbb", # A
    "aaaa", # A
    "abab", # R
    "aaaaabbb", # A
]

for input_str in test_inputs:
    if dfa04.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.5 AFD M_8 que reconhece $L_8 = \{x \in \Sigma^* \mid x \bmod 2 = 0\}$

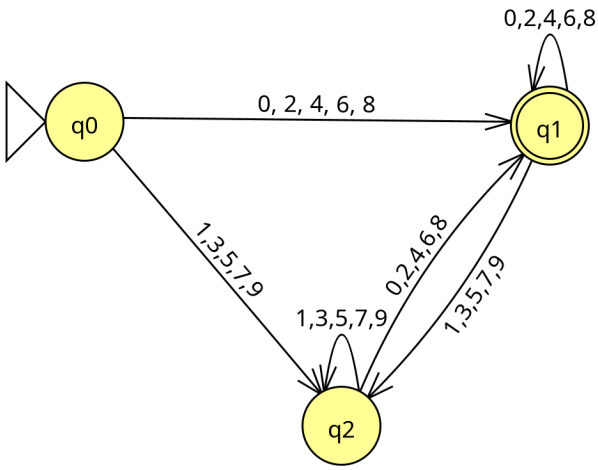

Figura 5 – AFD - Exercício 05

Listing 5 – AFD M_8 em Python com a biblioteca `automata-lib`

```
from automata.fa.dfa import DFA

dfa05 = DFA(
    states={"q0", "q1", "q2"},
    input_symbols={"0", "1", "2", "3", "4", "5", "6", "7", "8", "
       9"},
```

```
    transitions={
        "q0": {"0": "q1", "2": "q1", "4": "q1", "6": "q1", "8": "
           q1", "1": "q2", "3": "q2", "5": "q2", "7": "q2", "9": "
           q2"},
        "q1": {"0": "q1", "2": "q1", "4": "q1", "6": "q1", "8": "
           q1", "1": "q2", "3": "q2", "5": "q2", "7": "q2", "9": "
           q2"},
        "q2": {"0": "q1", "2": "q1", "4": "q1", "6": "q1", "8": "
           q1", "1": "q2", "3": "q2", "5": "q2", "7": "q2", "9": "
           q2"},
    },
    initial_state="q0",
    final_states={"q1"},
)


test_inputs = [
    "0", # A
    "024",  # A
    "02134", # A
    "12345678", # A
    "975310", # A
    "012", # A
    "1", # R
    "1311", # R
    "021345", # R
    "1234567", # R
    "864201", # R
    "123", # R
]

for input_str in test_inputs:
    if dfa05.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.6 AFD M_9 que reconhece $L_9 = \{x \in \Sigma^* \mid x \bmod 5 = 0\}$

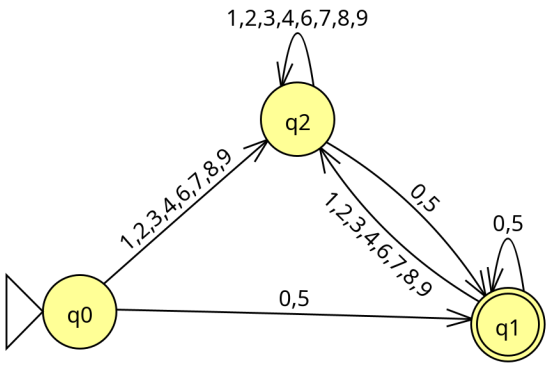

Figura 6 – AFD - Exercício 06

Listing 6 – AFD M_9 em Python com a biblioteca `automata-lib`

```
from automata.fa.dfa import DFA

dfa06 = DFA(
    states={"q0", "q1", "q2"},
    input_symbols={"0", "1", "2", "3", "4", "5", "6", "7", "8", "
       9"},
    transitions={
        "q0": {"0": "q1", "5": "q1", "1": "q2", "2": "q2", "3": "
           q2", "4": "q2", "6": "q2", "7": "q2", "8": "q2", "9": "
           q2"},
        "q1": {"0": "q1", "5": "q1", "1": "q2", "2": "q2", "3": "
           q2", "4": "q2", "6": "q2", "7": "q2", "8": "q2", "9": "
           q2"},
        "q2": {"0": "q1", "5": "q1", "1": "q2", "2": "q2", "3": "
           q2", "4": "q2", "6": "q2", "7": "q2", "8": "q2", "9": "
           q2"},
    },
    initial_state="q0",
    final_states={"q1"},
)


```

```
test_inputs = [
    "0", # A
    "05",  # A
    "5500055", # A
    "0123467895", # A
    "9876543210", # A
    "000", # A
    "1", # R
    "0123456789", # R
    "123456789", # R
    "050501", # R
    "551556", # R
    "106", # R
]

for input_str in test_inputs:
    if dfa06.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

1.7 AFD M_{10} que reconhece $L = \{w \in \{a,b\}^* \mid |w|_a \bmod 2 = 1 \wedge |w|_b \bmod 3 = 0\}$

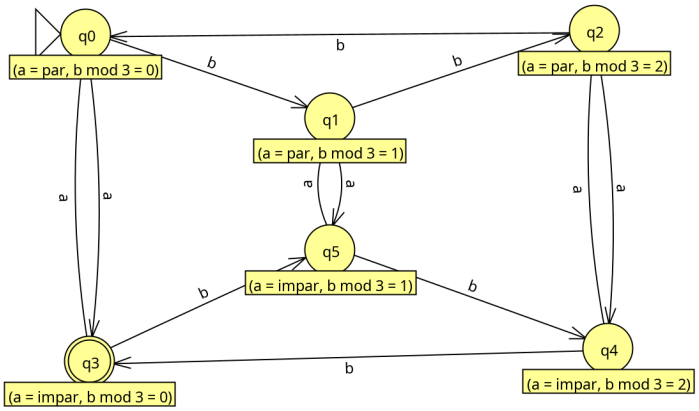

Figura 7 – AFD - Exercício 07

Listing 7 – AFD M_10 em Python com a biblioteca `automata-lib`

```python
from automata.fa.dfa import DFA

dfa07 = DFA(
    states={'q0', 'q1', 'q2', 'q3', 'q4', 'q5'},
    input_symbols={'a', 'b'},
    transitions={
        'q0': {'a': 'q3', 'b': 'q1'},
        'q1': {'a': 'q5', 'b': 'q2'},
        'q2': {'a': 'q4', 'b': 'q0'},
        'q3': {'a': 'q0', 'b': 'q5'},
        'q4': {'a': 'q2', 'b': 'q3'},
        'q5': {'a': 'q1', 'b': 'q4'},
    },
    initial_state='q0',
    final_states={'q3'}
)

test_inputs = [
    "a", # A
    "abbb",  # A
    "abababb", # A
    "aaabbb", # A
    "bbba", # A
    "bababa", # A
    "b", # R
    "bbb", # R
    "abbba", # R
    "aaababb", # R
    "bbbaa", # R
    "ababa", # R
]

for input_str in test_inputs:
    if dfa07.accepts_input(input_str):
        print(f"'{input_str}' - ACEITO.")
    else:
        print(f"'{input_str}' - REJEITADO.")
```

2 Referências