

UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho Prático III Fundamentos da Teoria da Computação

Artur Souza Papa - 3886

Luciano Belo de Alcântara Júnior - 3897

Larissa Isabelle - 3871

Trabalho Prático III apresentado à disciplina
de Fundamentos da Teoria da Computação
do curso de Ciência da Computação da
Universidade Federal de Viçosa.

Florestal

03 de Agosto de 2022

CCF 131 - Fundamentos da Teoria da Computação

Trabalho Prático III

Artur Souza Papa - 3886

Luciano Belo de Alcântara Júnior - 3897

Larissa Isabelle Rodrigues e Silva - 3871

03 de Agosto de 2022

Contents

1	Introdução	2
2	Tarefa Obrigatória	3
3	Implementação	3
4	Tarefas extras	3
4.1	Alfabeto de entrada	4
4.2	Não determinismo	4
4.3	Múltiplos tipos	6
4.4	Interface Gráfica	6
4.5	Observações	6
5	Testes	7
5.1	AFD	7
5.2	AFN	8
6	Conclusão	8

1 Introdução

A princípio, vale dizer que as máquinas de estado-finito são máquinas abstratas que capturam as partes essenciais de algumas máquinas concretas. Estas últimas vão desde máquinas de vender jornais e de vender refrigerantes, passando por relógios digitais e elevadores, até programas de computador digital, se considerarmos que sua memória é limitada, pode ser modelado por meio de uma máquina de estado-finito[1].

Para esse trabalho, foi implementado um programa que fosse capaz de, a partir da entrada, configurar um autômato finito determinístico (AFD) e utilizá-lo para rejeitar ou aceitar palavras.

Um AFD é uma estrutura matemática constituída de 3 tipos de entidades: um conjunto de estados, um alfabeto e um conjunto de transições. Dos estados, destaca-se um como o estado inicial, e destaca-se um subconjunto de estados como sendo composto dos estados finais[1]. Um autômato finito determinístico é composto por 5 tuplas Q, Σ, q, F, δ .

Q : conjunto de todos os estados.

Σ : conjunto de símbolos de entrada ou alfabeto. (Símbolos que a máquina recebe como entrada).

q : Estado inicial.

F : conjunto de estado final.

δ : Função de Transição, definida como $\delta : Q \times \Sigma \rightarrow Q$.

```
class DFA:
    def __init__(self, sigma={0: '0', 1: '1'}):
        self.states = dict()           # Q : conjunto de todos os estados.
        self.sigma = sigma             # Σ : conjunto de símbolos de entrada ou alfabeto
        self.initial_state = None      # q : estado inicial.
        self.final_state = dict()      # F : conjunto de estado final.
        self.transitions = dict()      # δ : função de Transição
```

Figura 1: Estrutura AFD

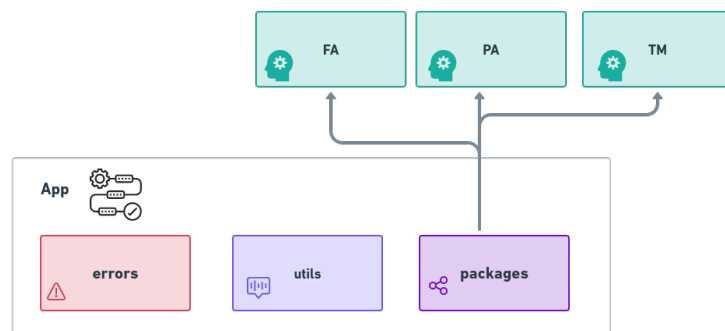


Figura 2: Arquitetura do projeto

2 Tarefa Obrigatória

Seguindo a especificação, foi implementado um AFD que fosse capaz de rejeitar ou aceitar uma palavra, conforme um padrão de entrada definido.

O alfabeto do autômato é composto por apenas 0 e 1. Na primeira linha da entrada, temos os nomes de cada estado do autômato, separados por espaços e podem possuir até no máximo 7 caracteres.

Já na segunda linha temos os estados iniciais e na linha seguinte temos os estados finais, sendo que seus nomes devem estar listados na primeira linha.

Nas linhas seguintes a função de transição é definida. Em cada linha uma ou mais transições são definidas. Cada transição é composta pelo nome de dois estados separados por espaços e por uma seta à direita (\rightarrow), depois uma barra vertical ($|$), e em seguida uma lista de caracteres, separados por espaço, que são os caracteres de entrada sobre os quais aquela transição ocorre.

O fim da definição da função de transição ocorre numa linha contendo apenas —.

As linhas seguintes definem, cada uma, um caso de teste, compostas apenas pelos símbolos do alfabeto de entrada, 0 e 1. Palavra vazia também pode ser uma entrada. A entrada acaba com o fim do arquivo.

3 Implementação

Para a implementação, escolhemos a linguagem Python, já que possui mais funcionalidades bibliotecas que ajudaram na implementação e possui a possibilidade de se utilizar dicionários, que facilita a implementação.

Para o autômato finito determinístico foi feita uma classe, que tem como atributos os estados, Σ , o estado inicial, um dicionário para estado final e um dicionário para transição. Além disso, temos todos os métodos para conseguir atribuir valores a eles.

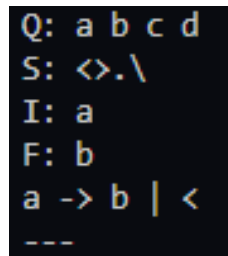
O valor de Σ , caso não seja informado, vai ter o alfabeto assim como passado na especificação, 0 e 1. Para conseguir computar uma palavra que queremos verificar, os estados são percorridos a partir do estado inicial e se o último estado a ser acessado a partir das computações for o estado final, é considerada como palavra aceita.

4 Tarefas extras

Por conseguinte, vale ressaltar que foi implementado três funcionalidades extras, além de ter sido feita uma interface gráfica.

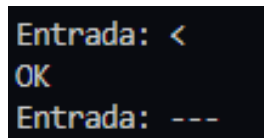
4.1 Alfabeto de entrada

A primeira funcionalidade extra implementada foi o **alfabeto de entrada**, nesta o autômato poderá utilizar de um alfabeto arbitrário, que será especificado também na entrada junto com seus outros parâmetros. Neste caso esses caracteres não serão separados, ou seja, todos os caracteres que aparecerem nesta linha serão considerados símbolos de entrada.



```
Q: a b c d
S: <>.\
I: a
F: b
a -> b | <
---
```

Figura 3: Exemplo de alfabeto de entrada



```
Entrada: <
OK
Entrada: ---
```

Figura 4: Resultado da computação

4.2 Não determinismo

A segunda funcionalidade extra implementada foi o **não determinismo**. Para esta, vale dizer que nosso autômato teria que ser capaz de fazer transições para estados diferentes utilizando o mesmo símbolo, assim, foi feito uso de uma *fila* para que fosse feita todas as computações possíveis de uma palavra e, caso alguma computação finalize no estado final, a palavra é aceita, caso todas as possibilidades sejam rejeitadas, a palavra também será.

```

def check(self, string):
    q = deque() # queue -> states from i to last character in S | (index, state)
    q.append([0, self.initial_states[0]]) # Starts from 0
    ans = False # Flag

    while q and not ans:
        front = q.popleft()
        index = front[0]
        state = front[1]

        if index == len(string):
            if state in self.final_states.values():
                ans = True
            elif string[index] not in self.sigma.values():
                raise exceptions.InvalidSymbolError(
                    string[index], 'Is not declared in sigma')
            elif state in self.transitions:
                # Search through states
                for transition in self.transitions[state].items():
                    d = transition[0]
                    states = transition[1]

                    if d == "\":
                        # Is epsilon
                        for state in states:
                            # Do not consume character
                            q.append([index, state])
                    elif string[index] == d:
                        for state in states:
                            # Consume character
                            q.append([index + 1, state])

```

Figura 5: Função para computar a palavra na AFN

4.3 Múltiplos tipos

Por fim, a última implementação extra feita pelo grupo foi a de **múltiplos tipos**, em que é permitido ao usuário escolher a máquina a ser usada como um parâmetro na entrada.

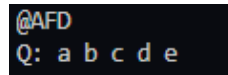


Figura 6: Selecionar AFD

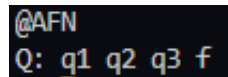


Figura 7: Selecionar AFN

4.4 Interface Gráfica

Por conseguinte, vale ressaltar que além das funcionalidades extras, foi feita uma interface gráfica usando a biblioteca *tkinter* para fazer as computações das palavras seja utilizando **AFD** ou **AFN**.

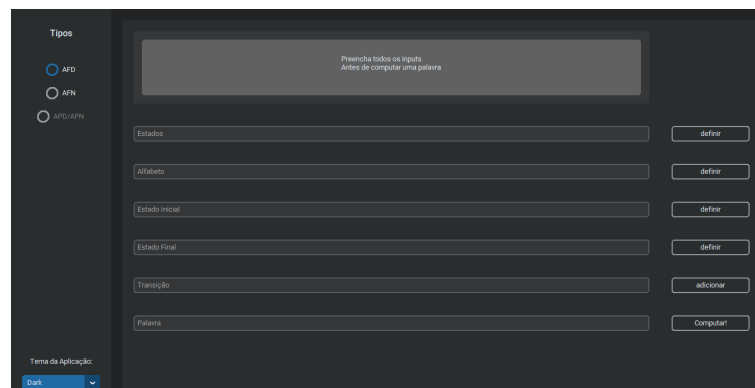


Figura 8: Interface gráfica

Primeiro selecionamos a máquina que queremos com a opção à direita, depois inserimos os estados, alfabeto, estado inicial, estado final, transição e a palavra, vale ressaltar que após a inserção de cada item é necessário clicar na opção **”definir”** para que seja computado e, por fim, após ser inserida a palavra, o usuário precisa escolher a opção **Computar!** e um *pop-up* aparecerá na tela informando o resultado da computação.

4.5 Observações

Outrossim, apesar de não ter sido citado acima, o grupo implementou as estruturas para os autômatos de pilha, máquina de Turing e autômato linearmente limitado, haja vista que existem pastas com as estruturas das três máquinas no nosso projeto, além de suas respectivas classes e funções.

5 Testes

5.1 AFD

```
@AFD
Q: a b c d e
S:
I: a
F: a d
a -> b | 1
a -> d | 0
b -> c | 0
b -> e | 1
c -> c | 0
c -> d | 1
d -> d | 0 1
e -> e | 0 1
---
```

Figura 9: Teste base

```
Entrada: 001
OK
Entrada: 1000
X
Entrada:
OK
Entrada: 10000011
OK
Entrada: 111
X
Entrada: 1
X
Entrada: ---
```

Figura 10: Resultados teste

5.2 AFN

```
@AFN
Q: q1 q2 q3 f
S: a
I: q1
F: f
q1 -> q2 | a
q1 -> q3 | a
q2 -> f | a
---

-----NFA-----

Q: {0: 'q1', 1: 'q2', 2: 'q3', 3: 'f'}
Σ: {0: 'a'}
q: {0: 'q1'}
F: {0: 'f'}
δ: {'q1': {'a': {'q3', 'q2'}}, 'q2': {'a': {'f'}}}

-----NFA-----

Entrada: a
X
Entrada: aa
OK
Entrada: aaa
X
Entrada: ---
```

Figura 11: Teste e resultados do teste

6 Conclusão

Posto isso, conclui-se que o trabalho em questão foi desenvolvido conforme o esperado, atingindo as especificações requeridas na descrição do mesmo em implementar um autômato finito determinístico (AFD) para aceitar ou rejeitar palavras de acordo com a linguagem configurada, bem como algumas funcionalidades extras.

Por conseguinte podemos ressaltar que o material apresentado na disciplina foi de suma importância para o desenvolvimento do projeto, haja vista que as explicações dadas pelo professor nos ajudaram bastante a entender as etapas a serem executadas, assim como na construção dos códigos, além disso vale ressaltar que o livro-texto foi de grande ajuda para a criação dos autômatos.

Em adição, é válido dizer que apesar das dificuldades na implementação do código, o trio foi capaz de superar e corrigir quaisquer erros no desenvolvimento do projeto. Por fim, verificou-se a assertiva para o objetivo do projeto em implementar um autômato finito determinístico (AFD) para aceitar ou rejeitar palavras de acordo com a linguagem configurada, bem como algumas funcionalidades extras.

References

- [1] Newton José Vieira. *Linguagens e Máquinas: Uma Introdução aos Fundamentos da Computação*. UFMG, 2004.