SCC0505
Introdução à Teoria da Computação
Trabalho 1

Andrey Lucas Esteves Garcia | 10734290
Arthur Nobre Kuwahara | 12563672
Humberto Guenzo Yoshimoto Tello | 10310888
Orlando Pasqual Filho | 5881844

Prazo de entrega: 6 de maio de 2022

1 Sobre a solução

A solução funciona para a simulação tanto de um Autômato Finito Determinístico(AFD) quanto para um Autômato Finito Não Determinístico(AFN). Suportando os dados na especificação que nos foi dada pelo arquivo de especificação do trabalho, consistindo de número de estados, símbolos terminais, transições e cadeias a serem analisadas pelo simulador.

2 Sobre o Código

2.1 Linguagem e estrutura

O código foi implementado na linguagem Python que pode ser dividido em 3 partes, a parte responsável pela leitura de dados, as funções de análise de cadeia, e a respectiva chamada destas funções após a leitura.

O Python foi a linguagem escolhida devido a facilidade que a mesma proporciona para trabalhar com dados textuais(strings) o que foi bastante útil para o desenvolvimento do programa.

2.2 Leitura de dados

A leitura de dados é feita na função principal (main) do programa, onde dado um arquivo de entrada especificado pelo usuário durante a execução do programa, os dados serão salvos em variáveis relativas ao que eles representam no formato do arquivo fornecido na especificação deste trabalho. As variáveis responsáveis por este armazenamento são:

- **newFile**: variável onde o arquivo está armazenado
- **newFileLines**: uma lista que separa e armazena o arquivo em linhas.
- **numberOfStates**: variável que armazena o número de estados do autômato.
- **quantityOfTerminalSymbols**: variável que armazena a quantidade de símbolos terminais aceitos pelo autômato.
- **terminalSymbolsList**: lista que armazena todos os possíveis símbolos terminais do arquivo de entrada do autômato.
- **quantityOfAcceptanceStates**: variável que armazena o número de estados de aceitação do autômato.
- **acceptanceStatesList**: lista que armazena os estados de aceitação do autômato.
- **numberOfTransitions**: variável que armazena o número de transições possíveis de serem feitas entre os estados do autômato.
- **numberOfChains**: variável que armazena o número de cadeias que serão testadas pelo autômato.
- **chainsList**: lista que armazena as cadeias a serem interpretadas pelo autômato.

2.3 Funções de análise de cadeias

No código existem 2 funções responsáveis por analisar as cadeias e informar se o autômato aceitará ou rejeitará as cadeias dadas pelo arquivo de entrada. Essas funções são:

- **testChain**: Esta função é responsável por definir o estado inicial do autômato que, devido às especificações deste projeto, será sempre 0 e por chamar a função recursiva **testSubchain**, passando como parâmetro para ela, a cadeia e o estado inicial.
- **testSubchain**: Esta é uma função recursiva responsável por retornar se uma subcadeia gerada a partir da cadeia principal que está sendo testada é aceita ou rejeitada pelo autômato.

Para isso, ela tem como caso base da recursão a cadeia ser uma string vazia (“”) ou a string “-”, caso isso ocorra e o estado atual do autômato encontre-se dentro dos estados válidos indicados pelo arquivo de entrada ela irá retornar o valor **True**, caso contrário ela irá retornar o valor **False**.

Caso não esteja em um dos casos base, a função irá percorrer a lista que contém as transições aceitas pelo autômato e testar cada uma das transições que são possíveis de se realizar dado o estado atual e o caractere que está sendo lido pelo autômato(ao testar recursivamente todas as possíveis transições da iteração, é garantido que o programa funcionará para AFNs). Caso alguma das subcadeias testadas nessa parte do código contemple o caso base da recursão, é retornado o valor **True**, que significa que a cadeia é Aceita pelo autômato, no entanto caso nenhuma delas o contempla, é retornado o valor **False**, que significa que a cadeia é rejeitada pelo autômato.

2.4 Execução das funções de análise de cadeias

A parte final do código é responsável por percorrer a lista de cadeias a serem testadas e para cada uma delas chamar a função test Chain para cada uma delas, caso esta função retorna o valor **True**, uma linha contendo a string “aceita” é escrita no arquivo de saída do programa e impressa na tela, caso o valor seja **False**, a string “rejeita” é escrita e impressa na tela.

2.5 Execução do Programa

As instruções para a execução do programa constam no arquivo: **manualT1.txt**, mas em resumo é necessário em um terminal utilizar o comando “py main.py” ou “python main.py” a depender do sistema operacional ou instalação da linguagem.

Quando o programa está sendo executado, será necessário digitar o nome do arquivo de entrada que será utilizado para execução do programa para que todo o código seja executado, aqui é importante ressaltar que o usuário deve digitar também a extensão do arquivo que será utilizado e que, caso não seja utilizado um caminho de onde o arquivo está, o arquivo deverá estar contido no mesmo diretório que o arquivo main.py . Como dito na seção 2.4, o programa gera um arquivo de saída chamado Saida.txt e também imprime o resultado na tela.

3 Análise de desempenho

Os pontos principais a serem analisados neste código são relacionados a sua eficiência em relação ao tempo e ao espaço.

Quanto ao espaço, o maior uso do espaço ocorre nas listas usadas para armazenar os dados de entrada, onde quanto maior elas forem, mais espaço será necessário para armazená-las. Seria possível trabalhar diretamente com as linhas de entrada ao invés de armazenar os dados em variáveis, porém isso dificultaria muito a implementação do programa, então a melhor solução encontrada foi utilizar variáveis e listas para armazenar os dados de entrada.

Quanto ao tempo necessário para a execução do código ele será dependente também da entrada utilizada em cada execução, o que leva em consideração: o número de transições realizadas pelo autômato, a quantidade de cadeias a serem testadas e o número de estados de aceitação. Sendo que aqui o número de transições realizadas pelo autômato é de grande importância para o desempenho do programa em relação a esta métrica. Isso é especialmente relevante no caso dos **AFNs** pois nesse caso, durante a execução da função **testSubchain**, quanto maior o número de transições possíveis de se realizar do mesmo estado para outro com o mesmo caractere, mais vezes essa mesma função recursiva será chamada para ser testada até que uma delas retorne **True** ou até que todas as transações possíveis sejam testadas com nenhuma contemplando o caso base e o valor **False** seja retornado, sendo assim é seguro afirmar que este dado é o mais impactante no tempo de execução do projeto.