**1º Projeto de CTC 34 – Automata e linguagens formais**

Alunos: Gustavo Ceci Guimarães e Luiz Felipeh Aguiar de Lima Alves

1. **Introdução**

Na disciplina CTC – 34, o estudo de linguagens formais e Automata se iniciou em autômatos finitos determinísticos. A diferença destes para os autômatos finitos bidirecionais está no fato da “fita de leitura” dos 2AFDs poder se movimentar para ambos os lados(direita e esquerda). A seguir, uma definição formal dos 2AFDs.

Um autômato finito bidirecional pode ser definido como uma quíntupla M = (Q, Σ, δ, q0, F), na qual:

• Q é um conjunto finito de estados

• Σ é um alfabeto

• δ é uma função de transição de estados, δ : Q × Σ → Q × {L, R}, onde L significa Left (fita de move para a esquerda) e R significa Right (fita se move para a direita)

• q0 é o estado inicial

• F ⊆ Q é o conjunto de estados finais

Uma configuração instantânea de um autômato finito bidirecional M é uma cadeia wqx ∈ Σ∗QΣ∗ tal que

• wx é a cadeia de entrada;

• q é o estado do autômato;

• a cabeça lê o 1◦ símbolo da subcadeia x.

Uma linguagem L(M) aceita por um autômato finito bidirecional é formado por cadeias w tais que

O projeto foi feito na linguagem de programação *Python*, e teve como objetivo criar um leitor de 2AFDs e validar diversas cadeias para este dado autômato. O formato da entrada, assim como o algoritmo e ideias serão posteriormente discutidas neste relatório, na seção *Implementação*. O código se encontra no fim do relatório, na seção *Código*.

**2. Implementação**

O projeto foi concebido na linguagem *Python*, no arquivo *2afd.py*, e sua entrada foi escolhida como um arquivo de texto. Sua padronização de *input* foi feita em conjunto com outro grupo, e suas instruções de uso se encontram detalhadamente explicadas no arquivo de texto que se encontrar no GitHub do projeto. Basicamente, as informações passadas neste arquivo são: alfabeto, nome dos estados, estados finais, transições e cadeia a ser lida. Foi feita a assunção de que, como o projeto foi desenvolvido para autômatos determinísticos, a entrada sempre conterá todas as transições necessárias e todas as transições são válidas.

A lógica de armazenamento de transições foi implementada utilizando o conceito de matriz e seguindo a seguinte lógica: as linhas se referem ao estado atual, as colunas se referem ao atual símbolo a ser lido, os módulos dos valores se referem ao próximo estado e os sinais dos valores se referem a direção da leitura (positivo: direita; negativo: esquerda). Além disso, na matriz, a linha 0 não poderia ser usada, pois a informação “sinal” seria perdida. A seguir, um exemplo:

Alfabeto: [a, b] Estados: [q0, q1, q2] Estado inicial: q0 Estado final: q1

Transições: q0 --a--> q1, R q1 --a--> q1, L q2 --a--> q2, L

q0 --b--> q2, R q1 --b--> q0, R q2 --b--> q0, L

Matriz:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 0 | 1 |
|  |  | a | b |
| 0 |  |  |  |
| 1 | q0 | +2 | +3 |
| 2 | q1 | -2 | +1 |
| 3 | q2 | -3 | -1 |

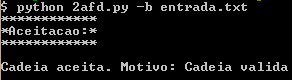
Utilizando a lógica supracitada, a principal parte da verificação de cadeias consistiu em andar com a fita pelo vetor “cadeia”, guardando todos os pares “estado atual” e “posição da fita” para identificar loops infinitos. Se a fita chegasse a uma posição maior do que o tamanho da cadeia ou menor que 0, isso significava que a leitura havia acabado. Caso o estado final de leitura estivesse presente na lista de estados finais do autômato, a cadeia é aceita. Caso não, é rejeitada. Além disso, caso houvesse um símbolo não presente no alfabeto, a cadeia é rejeitada.

Os principais métodos do projeto são:

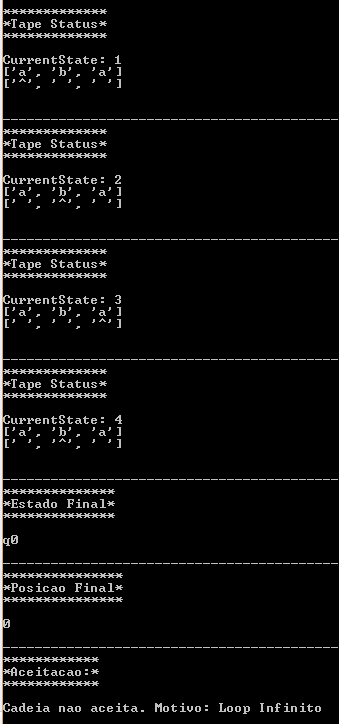
* **GetStates**: Após a leitura do arquivo, esse método monta a matriz da maneira indicada acima.
* **IsValid**: Verifica se a cadeia é válida para o dado autômato.
* **DebugLog**: Método que organiza a impressão de informação na tela, garantindo um debug mais efeitvo

A execução do programa se dá de duas formas:

* python 2afd.py –b \_\_\_\_\_\_\_\_.txt
  + Modo normal de execução. Seu output é feito na própria linha de comando, como exemplificado na figura abaixo:



* python 2afd.py –d \_\_\_\_\_\_\_\_.txt
  + Modo debug de execução. Além do output do modo normal, essa execução executa o debug para visualização da execução passo a passo, como exemplificado na imagem abaixo:



**3. Resultados**

Diversas cadeias foram testadas no programa, todas com resultados idênticos aos teóricos, ou seja, resultados satisfatórios. A seguir, será definida uma configuração de teste e serão mostrados os resultados obtidos com o programa.

Alfabeto: [a, b] Estados: [q0, q1, q2] Estado inicial: q0 Estado final: q1

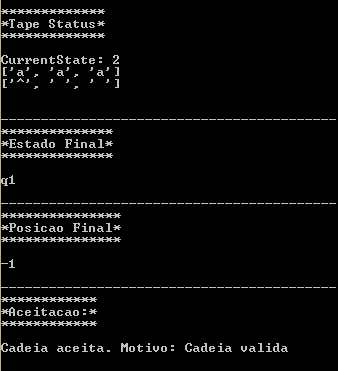
Transições: q0 --a--> q1, R q1 --a--> q1, L q2 --a--> q2, L

q0 --b--> q2, R q1 --b--> q0, R q2 --b--> q0, L

Cadeias:

* aaa: Teórico => Cadeia aceita

Prático:



* abababbbabaa: Teórico => Cadeia não aceita por loop infinito

Prático:



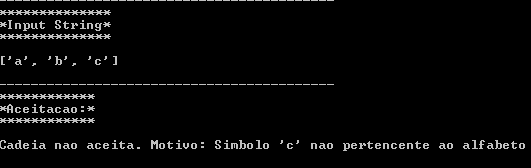
* baaaab: Teórico => Cadeia não aceita por terminar em um estado não-final

Prático:



* abc: Teórico => Cadeia não aceita por conter símbolo não presente no alfabeto

Prático:



**4. Conclusão**

Baseado no resultado obtido, o objetivo do projeto foi obtido. Foi feito um programa em Python relativamente simples, bem comentado e de fácil entendimento, seguindo também as diretrizes de Engenharia de Software aprendidas no curso. A entrada foi relativamente trabalhosa de ler, mas foi escolhida por ser mais intuitiva pro usuário. A saída, por linha de código, foi escolhida por ser um programa de aceitação ou não. Todos os resultados foram comparados com sucesso a seus teóricos. Por esses motivos, considera-se que o objetivo foi cumprido.

**5. Código**

O código também se encontra no repositório <https://github.com/Ghust1995/2afd>. A seguir, segue-se o código para leitura:



