

ECM253 – Linguagens Formais, Autômatos e Compiladores

Projeto 01

Interpretador de autômatos determinísticos

Prof. Marco Furlan

15 de maio de 2019

Instruções:

- Este projeto é para ser realizado em **grupos** já formados no início das aulas;
- **Implementar** o projeto em **Python**;
- **O que é para enviar:**
 - (1) Um **arquivo** Python contendo o código **devidamente comentado** que resolve o problema apresentado;
 - (2) **Arquivos-texto** contendo a definição dos **autômatos determinísticos** utilizados nos testes e que são **interpretados** pelo seu programa;
 - (3) Um documento **PDF** contendo os **nomes** e **RAs** dos integrantes da equipe e **descrevendo** os **autômatos** que foram testados por meio de **diagramas** de **máquinas de estado**, bem como a **descrição** das **linguagens** aceitas por cada autômato (em expressões matemáticas) e, para cada autômato/linguagem, fornecer **exemplos** de **entradas** que são **aceitas** e **exemplos** de **entradas** que são **rejeitadas**.

Preparar pelo menos **quatro exemplos** de autômatos determinísticos, à sua escolha (apresentados em aula, consultados em livros, na Internet etc.). Para **cada autômato**, exemplificar **duas cadeias** de teste que são **aceitas** e **duas** que são **rejeitadas**.
 - (4) **Teste obrigatório:** **elaborar** um **autômato** finito **determinístico** que **reconheça** números **reais** tais como 57, -123.89, 6.02E23, 0.33, -9.21e-21. **Incluir** um **diagrama** de **estados** para esse autômato.
- Enviar um único arquivo **ZIP** contendo os itens anteriores devidamente organizados em pastas.

1 Objetivos

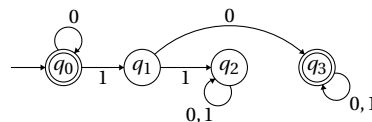
O objetivo deste projeto é **implementar um interpretador de autômatos determinísticos em Python**. O programa deverá **ler a descrição** de um **autômato** finito determinístico (veja a seção 2) **presente** em um **arquivo-texto** e então **aguardar** (até que o usuário termine o programa) a **entrada** de **cadeias** que serão

simuladas pelo autômato. Assim que uma **cadeia** é **entrada** no programa, o **programa** deverá **simular** o **funcionamento** do autômato com **esta cadeia**, **apresentando** na **tela** os **pares** (q, c) a partir de um estado inicial, onde q é um estado e c é um símbolo de entrada. Deverá **apresentar**, no **final**, se esta cadeia foi **aceita** ou **rejeitada** e, neste caso, imprimir o **motivo**: estado de não aceitação com a cadeia vazia ou a cadeia não está vazia mas não conseguiu aplicar uma transição.

2 Descrição de um autômato determinístico

Pode-se utilizar um arquivo-texto contendo a estrutura da máquina descrita em Python. **Python permite interpretar no “ar” códigos Python com o auxílio da função eval()**.

Por exemplo, considerar a máquina M representada a seguir:



Ele aceita a linguagem $L(M) = \{0^n, 0^n 10x\}$, onde x é qualquer cadeia contendo 0s e 1s. Esse **autômato** pode ser assim **codificado**, em um **arquivo-texto**, por exemplo, m.dfa, que contém estruturas válidas em **Python**:

```
{
  'states' : set([0,1,2,3]),
  'initial_state' : 0,
  'sigma' : set(['0','1']),
  'delta' : { (0,'0'):0, (0,'1'):1,
              (1,'0'):3, (1,'1'):2,
              (2,'0'):2, (2,'1'):2,
              (3,'0'):3, (3,'1'):3,
            },
  'final_states' : set([0, 3])
}
```

Notar que esta codificação está bem próxima dos elementos de um autômato finito determinístico: $M = (Q, \Sigma, \delta, q_0, F)$. Neste arquivo:

- A **descrição do autômato** é um **dicionário Python** (dict) cujas **chaves** são os **elementos** dele;
- O **conjunto de estados** e o **conjunto de estados finais** e o **alfabeto** são **representados** por um **conjunto** (set) de **Python**. Um set é uma **lista** onde a **ordem não importa** e onde **não se tem duplicatas**. Seguem algumas **operações** que se pode utilizar quando se tem um set **Python** (onde x , é um elemento do conjunto e R e S são sets quaisquer):
 - $x \text{ in } S$: o mesmo que $x \in S$;
 - $x \text{ not in } S$: o mesmo que $x \notin S$;
 - $R < S$: o mesmo que $R \subset S$;
 - $R \leq S$: o mesmo que $R \subseteq S$.
- A **função de transição** é **representada** por **outro dicionário**: neste dicionário, a **chave** é um **par** (q, c) onde q é o **estado** e c é um **símbolo** da **entrada** e que **mapeia** para um outro **estado**;
- Notar que foram usados **números inteiros** para **estados** e **caracteres** (cadeias) para **símbolos** da **entrada**.

Como interpretar este arquivo? Primeiramente é necessário **ler** este **arquivo**. Felizmente, esta tarefa é muito **simples** em **Python**. A **função** `open()` **abre** um **arquivo** e atribui a ele um **objeto de arquivo** com a construção a seguir:

```
with open('m.dfa') as dfa_file:
    dfa_data = dfa_file.read()
```

A função `read()` lê todo arquivo e armazena seu conteúdo como uma cadeia de caracteres na variável `data` do exemplo. Por fim, como transformar uma cadeia de caracteres contendo código Python em elementos de um programa que podem ser manipulados? Isso pode ser feito com a função `eval()`, assim:

```
dfa = eval(dfa_data)
# Para conferir o conteúdo
print(dfa)
```

Daqui para frente, com esses elementos, é possível realizar a interpretação do autômato a partir de uma entrada qualquer.

3 Interpretação do autômato finito determinístico

O algoritmo é bem simples e está descrito a seguir em pseudocódigo:

```
function simular_dfa(dfa, entrada)
begin
    estado = obter o estado inicial do autômato
    aceitar = falso
    while comprimento(entrada) > 0
        begin
            c = remover o símbolo mais à esquerda da entrada
            if c não está no alfabeto then
                begin
                    erro('0 símbolo', c, 'não pertence ao alfabeto do autômato!')
                    recolocar c no início da entrada
                    break
                end
            if estado não está no conjunto de estados then
                begin
                    erro('0 estado', estado, 'não pertence ao conjunto de estados do autômato!')
                    break
                end
            estado = obter o próximo estado a partir de estado e c
            if não for possível realizar a transição then
                begin
                    erro('Não foi possível realizar a transição do estado', estado, 'com entrada', c)
                    break
                end
            end
        end
        if estado estiver no conjunto de estados finais e a entrada estiver vazia then
            begin
                aceitar = verdadeiro
            end
            if aceitar for verdadeiro then
                begin
                    exibir('A cadeia', entrada, 'foi aceita pelo autômato!')
                end
            else
                begin
                    exibir('A cadeia', entrada, 'foi rejeitada pelo autômato!')
                end
            end
        end
    end
end
```

4 Requisitos funcionais do programa

Lista de **funcionalidades** que o **programa** deverá apresentar:

- (1) **Ler a especificação** de um autômato finito determinístico **de** um **arquivo-texto**;
- (2) **Apresentar** os **estados transitados** durante uma simulação;
- (3) **Sinalizar**:
 - a. **Quando** uma **cadeia** é **reconhecida** com sucesso;
 - b. **Erro quando** um **estado** selecionado **não está** no **conjunto** de **estados**;
 - c. **Erro** quando um **símbolo** lido da **entrada não está** no **alfabeto**;
 - d. **Erro** quando **não** for **possível** aplicar uma **transição**..
- (4) O **programa** deve **interagir** com o **usuário** – ele poderá testar quantas cadeias quiser até interromper;
- (5) A **forma** como se **entrará** o **nome** do **arquivo** do **autômato** no programa é **livre** (pode usar `input()` ou argumentos de linha de comando).

5 Exemplo de uso

5.1 Testes positivos

```
marco@JUPITER:~$ python3 /home/marco/Projects/Python/DFASimulator/dfasim.py m3.dfa
Digite a cadeia:
A cadeia foi aceita pelo autômato!
Digite a cadeia: 0000
(0, '0') -> 0
(0, '0') -> 0
(0, '0') -> 0
(0, '0') -> 0
A cadeia 0000 foi aceita pelo autômato!
Digite a cadeia: 00010101010
(0, '0') -> 0
(0, '0') -> 0
(0, '0') -> 0
(0, '1') -> 1
(1, '0') -> 3
(3, '1') -> 3
(3, '0') -> 3
(3, '1') -> 3
(3, '0') -> 3
(3, '1') -> 3
(3, '0') -> 3
A cadeia 00010101010 foi aceita pelo autômato!
Digite a cadeia: 10
(0, '1') -> 1
(1, '0') -> 3
A cadeia 10 foi aceita pelo autômato!
Digite a cadeia: ^D
Programa finalizado pelo usuário!
```

5.2 Testes negativos

```
marco@JUPITER:~$ python3 /home/marco/Projects/Python/DFASimulator/dfasim.py m3.dfa
Digite a cadeia: 1110
(0, '1') -> 1
(1, '1') -> 2
(2, '1') -> 2
(2, '0') -> 2
A cadeia 1110 foi rejeitada pelo autômato!
Digite a cadeia: 10a1
(0, '1') -> 1
(1, '0') -> 3
O símbolo a não pertence ao alfabeto do autômato!
A cadeia 10a1 foi rejeitada pelo autômato!
Digite a cadeia: ^D
Programa finalizado pelo usuário!
```

6 Dicas

Usar se desejar:

- Para **ler a entrada** que está em uma **cadeia de caracteres**, **transforme** a cadeia entrada **em** uma **lista** de caracteres assim: `s = list(s)`. Desse modo, será possível **utilizar a função** `pop()` para **extrair** (e reduzir a cadeia) **assim**: `c = s.pop(0)`;
- Se **quiser ler** o nome do arquivo pela **linha de comando**: **importar** de `sys` a **variável** `args`, que contém a **lista** de **argumentos** do script (na posição 0 é o nome do script, na posição 1 tem o primeiro argumento etc.);
- Se quiser **interromper** “graciosamente” o **programa** tecando CTRL+C ou CTRL+D (fim de arquivo em Linux) ou CTRL+Z (fim de arquivo em Windows): adicionar um bloco `try/except` na linha que lê a entrada. Tratar respectivamente as exceções `KeyboardInterrupt` e `EEOFError`;
- Se quiser **tratar exceção** de **chave não encontrada** no dicionário: proteger a instrução que usa a chave no dicionário e tratar a exceção `KeyError`.