Ellison William Medrado Guimarães, discente UESC, matrícula 201620215.

Implementação de autômato para reconhecimento de expressões matemáticas

O projeto consiste em implementar um autômato que reconheça expressões aritméticas simples de uma linguagem de programação, onde, essa expressão aritmética é composta por **identificadores** e **operadores aritméticos simples**, no qual devemos desenvolver autômatos separados para cada finalidade. Nos tópicos posteriores será analisado cada autômato por parte, e finalmente, a implementação em *Python*.

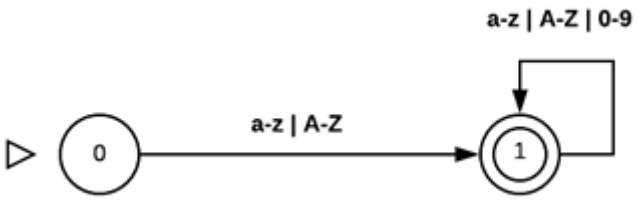
Foi utilizado nessa implementação a linguagem de programação *Python*. A linguagem foi utilizada devido as estruturas de dados básicas enriquecidas que facilitou o desenvolvimento do mesmo.

1. Autômato dos Identificadores

Os nomes dos identificadores tem por padrão as seguintes regras de escrita:

- Iniciam com uma letra do alfabeto, seja maiúscula ou minúscula;
- Todos os outros caracteres podem conter letras do alfabeto e números.

Com isso, tem expressão regular do tipo: `[a-zA-Z] . [a-zA-Z0-9]*`. E autômato finito determinístico, sendo `|Q| = 2` e o estado final sendo somente o estado **1**:

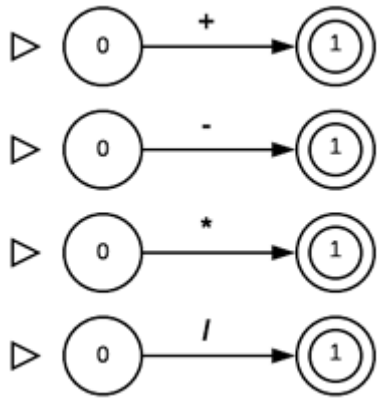


Com funções de transição:

- ς(0, **a-zA-Z**) = 1
- ς(1, **a-zA-Z0-9**) = 1

2. Autômato dos operadores aritméticos simples

Como os operadores aritméticos são compostos por somente um único símbolo, os autômatos finitos são compostos por um estado inicial (**0**) e um estado final (**1**), com função de transição sendo ς(0, **+ | - | * | /**) = 1:



3. Implementação

A implementação é composta por três arquivos. O arquivo `input.txt`, no qual será inserido a expressão matemática que será analisada pelo autômato, o arquivo `output.txt` que terá os *tokens* da expressão de entrada após a execução do algoritmo. Já o algoritmo encontra-se no arquivo de nome `automato.py`.

Para exemplificar, inserindo a seguinte expressão no arquivo de `input.txt`:

```
a + b - c
a * c - a
a - b * c
```

O arquivo de saída, ou seja, `output.txt` será:

```
<identificador, a>
<soma,>
<identificador, b>
<sub,>
<identificador, c>
<identificador, a>
<mult,>
<identificador, c>
<sub,>
<identificador, a>
<identificador, a>
<sub,>
<identificador, b>
<mult,>
<identificador, c>
```

3.1. Funcionamento do autômato

O algoritmo contém uma classe de nome **Autômato**, no qual representa o autômato. O seu construtor recebe:

1. Quantidade de estados (`qtd_estados`);
2. Lista de estados finais (`estados_finais`);
3. Funções de transições em forma de tuplas (estado_atual, lista_alfabeto, próximo_estado).

Além disso, a classe tem o método `retorna_cadeia` que recebe uma cadeia, analisa os primeiros caracteres que são aceitos pelo autômato até encontrar um símbolo que não é aceito, retornando o índice final da palavra reconhecida.

Com isso, foi instanciado o autômato de **identificador**:

```
AF_identificador = Automato(2, [1], (0, alfabeto, 1), (1, alfabeto + numeros, 1))
```

Como visto anteriormente, o **identificador** é composto por dois estados, sendo somente o estado **1** final, e as funções de transição: ς(0, **a-zA-Z**) = 1 (`(0, alfabeto, 1)`) e ς(1, **a-zA-Z0-9**) = 1 (`(1, alfabeto + numeros, 1)`).

E o autômato das **operações básicas**:

```
AF_soma = Automato(2, [1], (0, ['+'], 1)) #ς(0, +) = 1
AF_subt = Automato(2, [1], (0, ['-'], 1)) #ς(0, -) = 1
AF_divi = Automato(2, [1], (0, ['/'], 1)) #ς(0, /) = 1
AF_mult = Automato(2, [1], (0, ['*'], 1)) #ς(0, *) = 1
```

Após instanciado, basta testar uma cadeia através do método `retorna_cadeia`:

```
AF_identificador.retorna_cadeia('contador+i')   # 8
AF_soma.retorna_cadeia('+i')                    # 1
AF_identificador.retorna_cadeia('i')            # 1
```

Ou seja, é retornado o índice no qual, do início até o indice retornado é pertencente ao autômato. Podendo assim, recortar a string através de recursão. Exemplo:

```
string = 'contador+num'
string = string[AF_identificador.retorna_cadeia(string):]   # '+num'
string = string[AF_soma.retorna_cadeia(string):]            # 'num'
string = string[AF_identificador.retorna_cadeia(string):]   # ''
```

4. Dificuldades

A dificuldade encontrada nesse projeto, era representar a expressão com espaços entre os identificadores e os símbolos de operações matemáticas. Foi resolvida após alteração no método `retorna_cadeia`, onde retorna a cadeira até onde é reconhecida, após isso, são analisadas por outros autômatos da mesma forma de forma recursiva até o final do arquivo.