GERADOR DE AUTÔMATOS FINITOS DETERMINÍSTICOS

Alexsandro Meurer Schneider
Ismael Bortoluzzi

Curso de Ciências da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó-SC, Brasil

RESUMO: *Este artigo tem por objetivo explicar o passo a passo da geração de um autômato finito não determinístico, determinação deste autômato, assim como sua minimização, isto é, eliminação de estados mortos e inalcançáveis, e também a criação do estado de erro. O projeto foi desenvolvido utilizando a linguagem Python e tem por objetivo lançar as bases para realização do projeto prático da disciplina subsequente, Compiladores.*

Palavras-chave: autômato finito; minimização; determinação;

1. INTRODUÇÃO

Linguagens formais são modelos que possibilitam a especificação e o reconhecimento de linguagens, suas classificações, estruturas e propriedades. O autômato finito determinístico (AFD) é um reconhecedor simples para linguagens regulares. Sua principal aplicação na ciências da computação é como analisador léxico de linguagens de programação, também sendo útil para geradores de *dummy text, como o famoso Lorem Ipsum.* Será apresentado neste artigo como construir um gerador de AFDs a partir de um arquivo contendo tokens e gramáticas regulares

2. REFERENCIAL TEÓRICO

Neste tópico serão explicados quatro conceitos fundamentais para compreensão do projeto: o que são gramáticas, autômatos finitos, determinação e minimização de um autômato finito.

Uma gramática é uma ferramenta que gera cadeias de um determinado alfabeto. Gramáticas contém regras para a definição destas cadeias. Elas são úteis para definir características sintáticas de uma linguagem.

Autômato Finito é o tipo mais básico de reconhecedor de linguagens. É usado como analisador léxico e para reconhecer padrões em textos.

Determinação de um autômato finito consiste em eliminar transições que podem ir para dois estados distintos. Faz-se isso através da junção dos dois estados em um só. Isso pode gerar mais dessas transições ambíguas, logo o processo deve ser feito recursivamente até eliminar todas.

Minimização consiste em eliminar estados inalcançáveis e mortos do AFD. Inalcançáveis seriam estados que por nenhum caminho a partir da regra inicial podem ser atingidos. Mortos seriam estados que a partir deles, não existe condição de parada.

3. DESENVOLVIMENTO

Este trabalho foi desenvolvido utilizando a linguagem Python em sua versão 3, sem o auxílio de bibliotecas, salvo para exibir o autômato no console bem formatado. O código consiste em uma classe chamada Automata q contém toda a lógica de criar, determinar e minimizar o automato e algumas funções auxiliares para ler o arquivo e extrair os *tokens*, palavras reservadas, símbolos e a gramática regular. Abaixo se encontra a explicação de cada etapa:

Num primeiro momento a classe Automata inicializa alguns atributos, cria o estado inicial na matriz de estados e extrai os tokens com o auxílio de uma função externa:

```
class Automata:
    def __init__(self, words, grammar):
        self.initial_symbol = 'S'
        self.sigma = '\u03B4'
        self.epsilon = '\u03B5'
        self.alphabet = [
            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
            'N', 'O', 'P', 'Q', 'R', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
        ]
        self.words = words
        self.grammar = grammar
        self.all_tokens = get_all_tokens(self.words, self.grammar)
        self.states = [[self.initial_symbol] + [''] * len(self.all_tokens)]
        self.last_state = 0
        self.checked = set()
```

Depois o método *compile()* deve ser chamado para iniciar a construção do AFD. Este método é responsável por chamar de forma ordenada os quatro passos principais da construção e imprimir no console cada passo:

```
    def compile(self):
        self.build_afnd()
        print_table(self.states, [self.sigma] + self.all_tokens, 'autômato não determinizado')
        self.build_afd()
        print_table(self.states, [self.sigma] + self.all_tokens, 'autômato determinizado')
        self.minimize()
        print_table(self.states, [self.sigma] + self.all_tokens, 'autômato minimizado')
        self.create_error_state()
        print_table(self.states, [self.sigma] + self.all_tokens, 'autômato com estado de erro')
```

O método *build_afnd()* tem por objetivo pegar as palavras reservadas e criar um autômato finito não determinístico (AFND), trocar o nome das regras da gramática para não ter conflito com as regras já criadas através da chamada do método *change_states_signature()* e depois incluir a gramática no AFND

```
def build_afnd(self):
    for num_word, word in enumerate(self.words):
        len_word = len(word) - 1
        for num_token, token in enumerate(word):
            if num_token == 0:
                if num_word == 0:
                    self.states[0][self.all_tokens.index(token) + 1] += self.alphabet[self.last_state]
                    self.create_state()
                else:
                    self.states[0][self.all_tokens.index(token) + 1] += self.alphabet[self.last_state - 1]
            else:
                if num_token == len_word:
                    self.states[self.last_state][self.all_tokens.index(token) + 1] = self.alphabet[self.last_state]
                    self.create_state(state_final='*')
                    try:
                        self.words[num_word + 1]
                    except IndexError:
                        continue
                    self.create_state()
                else:
                    self.states[self.last_state][self.all_tokens.index(token) + 1] = self.alphabet[self.last_state]
                    self.create_state()

    self.change_states_signature()

    for sentence in self.grammar:
        state = sentence.split('::=')[0].strip()[1]
        for x in range(len(self.states)):
            if self.states[x][0] in (state, f'*{state}'):
                tokens_with_next_state = sentence.split('::=')[1].replace(' ', '').replace('>', '').split('|')
                self.remove_epsilon(tokens_with_next_state)
                for token_state in tokens_with_next_state:
                    token, state = token_state.split('<')
                    self.states[x][self.all_tokens.index(token) + 1] += state
                break
```

```
def change_states_signature(self):
    for x in range(1, len(self.grammar)):
        state = self.grammar[x].split('::=')[0].strip()[1]
        self.create_state()
        if self.epsilon in self.grammar[x]:
            self.states[self.last_state][0] = '*' + self.states[self.last_state][0]

        self.replace_signature(state, self.alphabet[self.last_state - 1])

def replace_signature(self, state, new_state):
    for x in range(len(self.grammar)):
        self.grammar[x] = self.grammar[x].replace(state, new_state)

def remove_epsilon(self, tokens_with_next_state):
    if self.epsilon in tokens_with_next_state:
        tokens_with_next_state.remove(self.epsilon)
```

Após o AFND estar feito é hora de transformá-lo em um AFD. Para isso partimos para a segunda etapa do processo invocando o método *build_afd*()

```
def build_afd(self):
    for state in self.states:
        for signature in state:
            if not any(signature in state[0] for state in self.states) and signature:
                state_final = False
                for token_new_state in signature:
                    if any('*' + token_new_state in state[0] for state in self.states) and \
                            not any('*' + signature in state[0] for state in self.states):
                        state_final = True
                if state_final:
                    self.create_state(state=signature, state_final='*')
                else:
                    self.create_state(state=signature)
                indexes = self.get_index(signature)
                self.states[indexes[0][0]][indexes[0][1]] = self.states[indexes[0][0]][indexes[0][1]]
                for new_token in signature:
                    indexes_tokens = self.get_index(new_token)
                    for key, valor in enumerate(self.states[indexes_tokens[-1][0]]):
                        if key == 0 or not valor:
                            continue
                        self.states[-1][key] += valor
    for state in self.states:
        for signature in state:
            if not any(signature in state[0] for state in self.states) and signature:
                self.build_afd()
```

Com o AFD pronto, precisa agora minimizá-lo. Este processo é divido entre eliminar estados mortos e inalcançáveis. Para isso, chamamos o método *minimize*() para orquestrar tudo:

```
def minimize(self):
    self.remove_unreachable()
    self.remove_dead()
```

Começamos removendo os estados inalcançáveis com *remove_unreachable*():

```
def remove_unreachable(self):
    self.checked.update(self.states[0])
    len_checked = len(self.checked)
    newly_added_states = set()

    new_len_checked = self.annotate_states(newly_added_states)

    while len_checked != new_len_checked:
        len_checked = new_len_checked
        new_len_checked = self.annotate_states(newly_added_states)

    for state in self.states:
        if '*' in state[0]:
            state_signature = state[0][1:]
        else:
            state_signature = state[0]
        if state_signature not in self.checked:
            state[0] = 'UNREACHABLE'

    self.states = list(filter(lambda x: x[0] != 'UNREACHABLE', self.states))
```

```
def annotate_states(self, newly_added_states):
    for state_signature in self.checked:
        if state_signature == '':
            continue
        state = self.get_state_by_signature(state_signature)
        for signature in state:
            if '*' in signature:
                newly_added_states.add(signature[1:])
            else:
                newly_added_states.add(signature)
    self.checked = self.checked.union(newly_added_states)
    return len(self.checked)

def get_state_by_signature(self, signature):
    for x in self.states:
        if x[0] in (signature, f'*{signature}'):
            return x
```

Depois removemos os estados mortos com *remove_dead*():

```
def remove_dead(self):
    for index, state in enumerate(self.states):
        self.checked = set(self.states[index])
        len_checked = len(self.checked)
        newly_added_states = set()
        is_dead = True

        new_len_checked, is_dead = self.check_if_is_dead(newly_added_states, is_dead)
        while len_checked != new_len_checked:
            if is_dead is False:
                break
            len_checked = new_len_checked
            new_len_checked, is_dead = self.check_if_is_dead(newly_added_states, is_dead)

        if is_dead is True:
            self.states[index][0] = 'DEAD'

    self.states = list(filter(lambda x: x[0] != 'DEAD', self.states))

    self.remove_transitions_to_dead()
```

```
def check_if_is_dead(self, newly_added_states, is_dead):
    for state in self.checked:
        if state == '':
            continue
        s = self.get_state_by_signature(state)
        for signature in s:
            if '*' in signature:
                newly_added_states.add(signature[1:])
                is_dead = False
                break
            else:
                newly_added_states.add(signature)
        if is_dead is False:
            break

    self.checked = self.checked.union(newly_added_states)
    return len(self.checked), is_dead
```

Para acabar a minimização removeremos transições para estados mortos. O próprio método *remove_dead*() chama o método *remove_transitions_to_dead*() que faz isso:

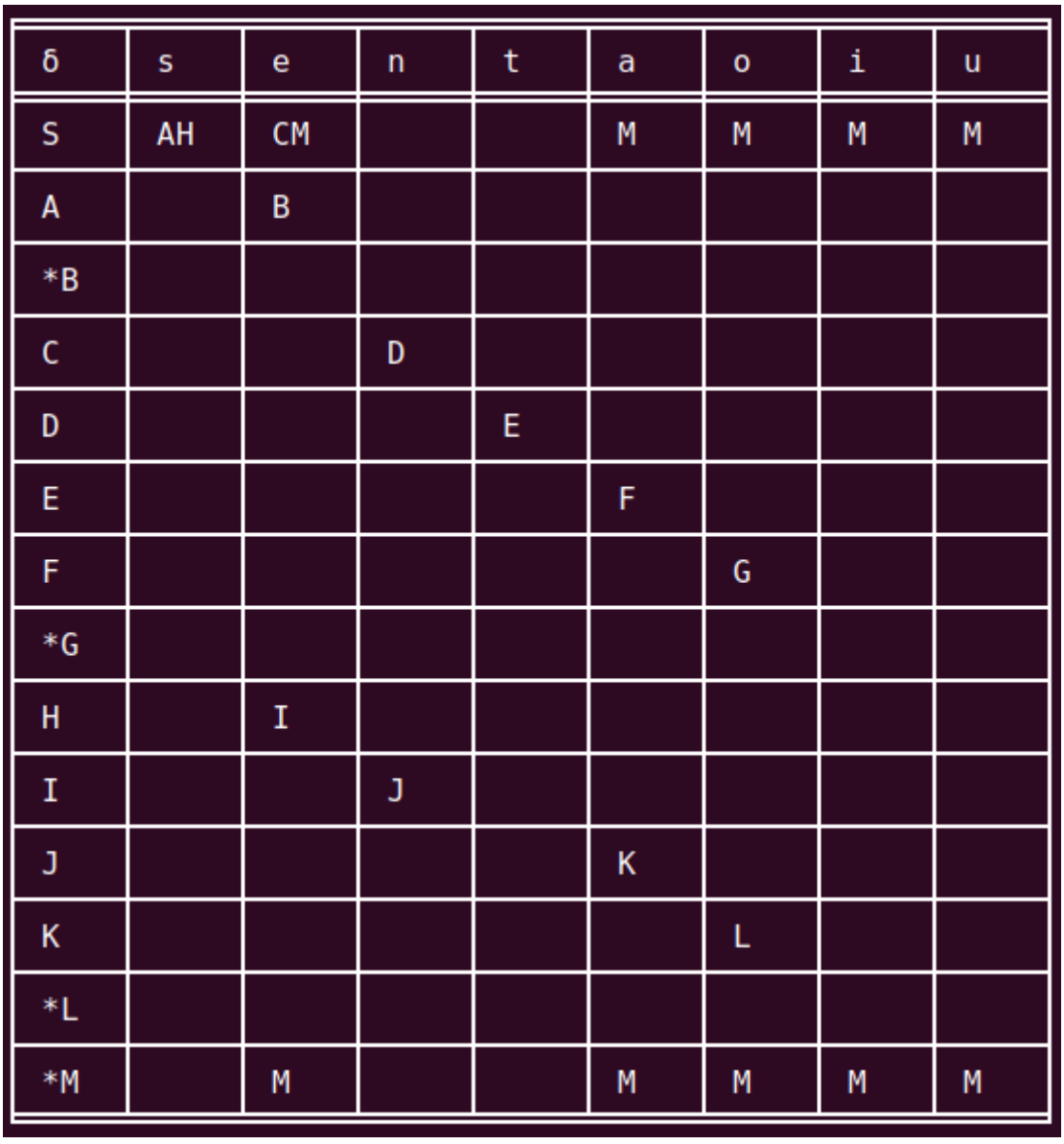
```
def remove_transitions_to_dead(self):
    all_states = [x[0] if '*' not in x[0] else x[0][1:] for x in self.states]
    for index_one, states in enumerate(self.states):
        for index_two, state in enumerate(states):
            if state not in all_states and index_two != 0:
                self.states[index_one][index_two] = ''
```

Por fim, o quarto passo é a criação do estado de erro no AFD com *create_error_state()*:

```
def create_error_state(self):
    self.create_state(state='<ERROR>', state_final='*')
    for key_state, state in enumerate(self.states):
        for key_token, token in enumerate(state):
            if not token:
                self.states[key_state][key_token] = '<ERROR>'
```

A seguir será apresentado a impressão no console após cada um dos quatro processos:

build_afnd():



δ	s	e	n	t	a	o	i	u
S	AH	CM			M	M	M	M
A		B						
*B								
C			D					
D				E				
E					F			
F						G		
*G								
H		I						
I			J					
J					K			
K						L		
*L								
*M		M			M	M	M	M

build_afd():

δ	s	e	n	t	a	o	i	u
S	AH	CM			M	M	M	M
A		B						
*B								
C			D					
D				E				
E					F			
F						G		
*G								
H		I						
I			J					
J					K			
K						L		
*L								
*M		M			M	M	M	M
AH		BI						
*CM		M	D		M	M	M	M
*BI		B	J					

minimize():

δ	s	e	n	t	a	o	i	u
S	AH	CM			M	M	M	M
*B								
D				E				
E					F			
F						G		
*G								
J					K			
K						L		
*L								
*M		M			M	M	M	M
AH		BI						
*CM		M	D		M	M	M	M
*BI		B	J					

create_error_state():

δ	s	e	n	t	a	o	i	u
S	AH	CM	<ERROR>	<ERROR>	M	M	M	M
*B	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>
D	<ERROR>	<ERROR>	<ERROR>	E	<ERROR>	<ERROR>	<ERROR>	<ERROR>
E	<ERROR>	<ERROR>	<ERROR>	<ERROR>	F	<ERROR>	<ERROR>	<ERROR>
F	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	G	<ERROR>	<ERROR>
*G	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>
J	<ERROR>	<ERROR>	<ERROR>	<ERROR>	K	<ERROR>	<ERROR>	<ERROR>
K	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	L	<ERROR>	<ERROR>
*L	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>
*M	<ERROR>	M	<ERROR>	<ERROR>	M	M	M	M
AH	<ERROR>	BI	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>
*CM	<ERROR>	M	D	<ERROR>	M	M	M	M
*BI	<ERROR>	B	J	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>
*<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>	<ERROR>

Para gerar este AFD foi utilizado um arquivo de texto contendo:

```
se
entao
senao
<S> ::= a<A> | b<A>
<A> ::= a<B> | b<A>
<B> ::= a<A> | b<B> | ε
```

4. CONCLUSÃO

Com este projeto finalizado pode-se concluir que os conhecimentos adquiridos em aula sobre linguagens formais, gramáticas e autômatos finitos foram de suma importância para realização do mesmo, ao mesmo tempo em que aplicamos os conhecimentos teóricos adquiridos em sala tirando o máximo proveito de tudo.

REFERÊNCIAS

SCHEFFEL. R. M. **Apostila Linguagens Formais e Autômatos**: Universidade do Sul de Santa Catarina.