Analisador Léxico Python

Gabriel Teixeira Júlio

O objetivo deste código é a criação de um simples analisador léxico, este foi feito em Python e se encontra na pasta ***src***. Nela a três arquivos:

- ***main.py*** - arquivo para executar o analisador
- ***analisador.py*** - arquivo que tem a implementação do AFD, tabela de tokes e tabela de símbolos
- ***codigo.txt*** - arquivo com um código utilizado para testar o analisador

Implementação

analisador.py

Como dito anteriormente neste arquivo está a implementação do AFD, tabela de tonkes e tabela de símbolos. O AFD foi feito com base no AFD do arquivo *"AFDCompleto.pdf*" e os tokens foram criados baseados no arquivo "*Exemplos_Linguagem.pdf*.

AFD

O AFD é implementado em uma classe que tem o seguintes atributos:

- **inicial** - o estado inicial do autômato
- **atual** - o estado atual em que o autômato está
- **final** - todos os estados finais, e para cada estado o tipo de Token que ele representa e retorno que serve para dizer se precisa ou não voltar um caracter na cadeia sendo testada. Exemplo: "*'q2': {'tipo': 'ID', 'retorno': -1}*"
- **transicao** - todas as transições do autômato, para cada estado tem se todas suas transições em que cada transição tem os caracteres válidos da transcição e o estado de destino. Exemplo: "*'q5': [[D, 'q6']],*"

O método ***testaCaracter*** testa se caracter passado para o método manda para um estado válido, se vai para estado final ou se vai para um estado ínvalido. Ele segue a seguinte lógica:

1. Começa pegando todas as transições possíveis do estado atual do AFD
2. Para cada transição obtida é feito:
    1. Verifica se o caracter faz parte dos caracteres válidos da transição, senão for parte pula para próxima transição
    2. Se for parte, o estado atual é atualizado para o estado de destino da transição
    3. Verifica se o estado atual é um estado final
        1. Se for final pega as informações do estado final, ou seja, o token
        2. Resta o estado atual para o inicial
        3. Retorna o Token, tipo e se precisa voltar um caracter da cadeia de teste
    4. Senão for o método retorna vazio
3. Se passar por todas as transições e não for caracter de nenhuma retorna um erro

```python
class AFD:
    def __init__(self):
        # Definição dos estados inicial e atual, e estados finais com tipos e retornos associados
        self.inicial = 'q0'
        self.atual = 'q0'
        self.final = {···
        # Transições de estado para cada estado atual
        self.transicao = {···

    # Método para testar cada caracter e transitar para o próximo estado
    def testaCaracter(self, caracter):
        trns = self.transicao[self.atual]  # Transições possíveis do estado atual

        for i in trns:
            if re.fullmatch(i[0], caracter) != None:  # Verifica se o caractere corresponde à transição
                self.atual = i[1]  # Atualiza o estado atual
                if self.atual in self.final:  # Se o estado atual for final
                    temp = self.final[self.atual]  # Obtém as informações associadas ao estado final
                    self.atual = self.inicial  # Reinicia o estado atual
                    return (temp['tipo'], temp['retorno'])  # Retorna o tipo e o retorno do token
                else:
                    return ('', 0)  # Caso contrário, retorna uma tupla vazia
        else:
            return ('erro', 0)  # Se não houver transição correspondente, retorna um erro
```

Símbolos

A tabela de Símbolos é uma lista que guarda apenas os IDs idetificados pelo analiador.

O método ***inserirID*** serve armazenar na lista de símbolos o valor do ID e retorna para tabela de tokens o token deste ID, enviando como tipo ID e valor o índice do ID inserido na lista de símbolos.

O método ***findID*** é utilizado para verificar se um ID já está na lista de símbolos. Se ele estiver na lista de símbolos ele retorna para tabela de tokens o token deste ID, enviando como tipo ID e valor o índice do ID inserido na lista de símbolos, senão ele utiliza o método *inserirID* para inserir o ID na tabela de símbolos.

```python
# Classe para armazenar os identificadores encontrados
class Simbolos:
    def __init__(self):
        self.ids = []

    def __str__(self):
        return tabulate({'ID': self.ids}, headers=['INDEX', 'ID'], tablefmt="outline", showindex='always')

    # Método para inserir um identificador na lista de identificadores
    def inserirID(self, token):
        # Adiciona o identificador à lista
        self.ids.append(token[1])
        # Retorna o tipo e o índice do identificador na lista
        return ('ID', self.ids.index(token[1]))

    # Método para encontrar um identificador na lista de identificadores
    def findID(self, token):
        # Verifica se o identificador já está na lista
        if token[1] in self.ids:
            # Retorna o tipo e o índice do identificador na lista
            return ('ID', self.ids.index(token[1]))
        else:
            # Caso contrário, insere o identificador na lista e retorna suas informações
            return self.inserirID(token)
```

Tokens

A tabela de Tokens é uma lista em que em cada posição do lista é guardado dois valores:

- **Token** - guardo qual o tipo de token encontrado
- **Value** - guarda o valor do token para os tonkes que necessitam, exemplo para toknes ID armazena o índice do ID na tabela de símbolos

O metódo ***inserirID*** serve para inserir um token(tipo, valor) na lista de tokens.

```
# Classe para armazenar os tokens encontrados
class Tokens:
    def __init__(self):
        self.tokens = []

    def __str__(self):
        return tabulate(self.tokens, headers=['Token', 'Value'], tablefmt="outline")

    # Método para inserir um token na lista de tokens
    def inserirToken(self, token):
        self.tokens.append(token)  # Adiciona o token à lista
```

subtiposId

Para alguns IDs é necessário verificar se ele não é um Token válido, como INT ou FLOAT, pois o AFD não separa estes tokens de ID. Então o método ***subtiposId*** foi criado para verificar se um ID é um Token especial ou apenas um ID, por fim ele retorna o token correto.

```
# Função para retornar o subtipo do identificador
def subtiposId(token, word):
    tipos = ['int', 'float', 'char', 'boolean', 'void', 'if', 'else', 'for', 'while', 'scanf', 'println', 'main', 'return']

    if word in tipos:  # Verifica se o identificador é uma palavra-chave
        return (word.upper(), '')  # Retorna o token
    else:
        return ('ID', word)  # Caso contrário, retorna o tipo como ID e o próprio identificador
```

main.py

O arquivo pode serve visto em duas partes: a primeira o método ***testaLinha*** que testa uma linha inteira do código sendo analisado pois o AFD só testa um caracter por vez, e a segunda para parte de abrir o arquivo do código de teste e printar a tabela de Tokens e Símbolos após testar o código inteiro.

testaLinha

O método usa alguns variáveis de apoios sendo elas:

- **count** - caracter da linha sendo analisado
- **word** - a palavra que está sendo montada ao passar pela linha, é reniciado quando um token é encontrado
- **resposta** - booleana para retornar se tudo ocorreu como devia ou não

O método segue a seguinte lógica:

1. Enquanto *count* for menor que o tamanho da lina
    1. Soma mais 1 em *count* e pega o caracter da linha na posição *count*
    2. Testa o caracter no AFD usando *testaCaracter*
    3. Atualiza o *count* com o valor de retorno do método *testaCaracter*
    4. Verifica se o método *testaCaracter* retornou um erro
    5. Se tiver retornado insere um Token 'ERRO' na tabela de tokens, *resposta* vira *False* e quebra o loop
    6. Senão retornou um erro:
        1. Adiciona o caracter testado em *word*
        2. Verifica o método *testaCaracter* retornou o tipo *SPACE* ou *BREAKLINE* se tiver reseta *word*
        3. Verica se o retorno de *testaCaracter* é diferente de '', *SPACE* e *BREAKLINE*
        4. Se for verifica se no retorno de *testaCaracter* pede para voltar um caracter, se pedir remove o ultimo caracter de *word*
        5. Verifica se o retorno de *testaCaracter* é *ID*
            1. Se for verifica se é algum Token que é identificado como *ID* usnado *subtiposId* e guarda o resultado em *resp*
            2. Verifica se ainda é *ID* se for verifica se o *ID* já está na tabela de Símblos e guarda o retorno em *resp*
        6. Senão for e verifica se o retorno de *testaCaracter* for >, >=, <, <=, *!*= ou == se for *resp* vira o Token de *COMP*
        7. Senão *resp* recebe o tipo de Token do retorno de *testaCaracter*
        8. Verifica se *resp* é um Token *COMMENT*
            1. Se for reseta *word* pula para próxima iteração do loop
        9. Inseri o Token *resp* na tabela de Tokens
        10. Reseta *word*
2. Retorna *resposta*

```python
# Função para testar uma linha do código fonte
def testaLinha(afd, simbolos, tokens, linha, line):
    count = -1  # Inicializa um contador para os caracteres na linha
    word = ''  # Inicializa uma string vazia para armazenar os caracteres
    resposta = True  # Inicializa a variável de resposta como True

    while count < len(line)-1:  # Loop enquanto o contador for menor que o comprimento da linha
        count += 1  # Incrementa o contador
        caracter = line[count]  # Obtém o caracter na posição do contador

        retorno = afd.testaCaracter(caracter)  # Chama o método testaCaracter do autômato finito determinístico (AFD) para testar o caracter
        count += retorno[1]  # Incrementa o contador com o retorno do método testaCaracter

        # Verifica se ocorreu um erro léxico ou um erro
        if retorno[0] == 'erro' or retorno[0] == 'ERRO':
            tokens.inserirToken(('ERRO', 'Tem um erro lexico na linha {}'.format(linha)))  # Insere um token de erro na lista de tokens
            resposta = False  # Define a resposta como False
            break  # Sai do loop

        else:
            word += caracter  # Adiciona o caracter à palavra em construção

            if retorno[0] == 'SPACE' or retorno[0] == 'BREAKLINE':  # Se o retorno for um espaço em branco ou quebra de linha
                word = ''  # Reinicia a palavra

            if retorno[0] != '' and retorno[0] != 'SPACE' and retorno[0] != 'BREAKLINE':  # Se o retorno não for vazio, espaço ou quebra de linha
                if retorno[1] == -1:  # Se o retorno for -1, remove o último caracter da palavra
                    word = word[:-1]

                if retorno[0] == 'ID':  # Se o retorno for um identificador
                    resp = ans.subtiposId(retorno[0], word)  # Obtém o subtipo do identificador

                    if resp[0] == 'ID':  # Se o subtipo for ID
                        resp = simbolos.findID(resp)  # Verifica se o identificador já existe na tabela de símbolos

                elif retorno[0] == '>' or retorno[0] == '>=' or retorno[0] == '<' or retorno[0] == '<=' or retorno[0] == '!=' or retorno[0] == '==':
                    resp = ('COMP', retorno[0])  # Se o retorno for um operador de comparação, define o token como 'COMP'

                else:
                    resp = (retorno[0], word)  # Caso contrário, define o tipo como o próprio retorno e a palavra

                if resp[0] == 'COMENT':  # Se o tipo for um comentário
                    word = ''  # Reinicia a palavra
                    continue  # Pula para a próxima iteração do loop

                tokens.inserirToken(resp)  # Insere o token na lista de tokens
                word = ''  # Reinicia a palavra
    return resposta  # Retorna a resposta
```

Execução

Para executar o analisador basta executar o arquivo ***main.py*** e passar qual o nome do arquivo a ser testado.