Análise Léxica utilizando PLY em Python

Guilherme Vasco da Silva

Departamento de Ciências da Computação
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão, PR – Brazil

guilhermevasco@alunos.utfpr.edu.br

Resumo. *Este artigo é um trabalho com o intuito de documentar um programa para realização da análise léxica de códigos na linguagem de programação TPP, realizando a varredura de códigos e retornando os valores dos tokens.*

1. Linguagem TPP

A linguagem a ser lida pelo analisador léxico será a TPP, que suporta dados inteiros e flutuantes e necessidade da especificação de tipos das variáveis locais e globais, porém as funções podem ter tipo omitido. A linguagem possui também operações de adição, multiplicação, subtração, divisão e também operadores lógicos E, OU e NÃO.

2. O Programa

O programa utilizado foi feito em Python, utilizando a biblioteca de auxílio para análise léxica, PLY. Ele trabalha realizando uma varredura do código fornecido e buscando traduzir cada um dos valores coletados, comparando eles aos valores em uma lista fornecida de palavras reservadas e símbolos da linguagem TPP para tradução.

2.1. Vetores de auxílio

Para o auxílio dessa varredura, a tradução de todas as palavras reservadas e símbolos (tokens) é armazenada em dois vetores.

```
tokens = [
    "ID",  # identificador
    # numerais
    "NUM_NOTACAO_CIENTIFICA",  # ponto flutuante em notaçao científica
    "NUM_PONTO_FLUTUANTE",  # ponto flutuate
    "NUM_INTEIRO",  # inteiro
    # operadores binarios
    "MAIS",  # +
    "MENOS",  # -
    "MULTIPLICACAO",  # *
    "DIVISAO",  # /
    "E_LOGICO",  # &&
    "OU_LOGICO",  # ||
    "DIFERENCA",  # <>
    "MENOR_IGUAL",  # <=
    "MAIOR_IGUAL",  # >=
    "MENOR",  # <
    "MAIOR",  # >
    "IGUAL",  # =
    # operadores unarios
    "NEGACAO",  # !
    # simbolos
    "ABRE_PARENTESE",  # (
    "FECHA_PARENTESE",  # )
    "ABRE_COLCHETE",  # [
    "FECHA_COLCHETE",  # ]
    "VIRGULA",  # ,
    "DOIS_PONTOS",  # :
    "ATRIBUICAO",  # :=
    # 'COMENTARIO', # {***}
]
```

Figura 1. Vetor de auxílio de símbolos (tokens).

```
reserved_words = {
    "se": "SE",
    "então": "ENTAO",
    "senão": "SENAO",
    "fim": "FIM",
    "repita": "REPITA",
    "flutuante": "FLUTUANTE",
    "retorna": "RETORNA",
    "até": "ATE",
    "leia": "LEIA",
    "escreva": "ESCREVA",
    "inteiro": "INTEIRO",
}
```

Figura 2. Vetor de auxílio de palavras reservadas.

Por conta desses vetores, não é necessário aplicar regras para cada palavra reservada, pois se o token lido não for uma palavra reservada ele é, automaticamente, um ID.

2.2. Tabela de símbolos

Com a tabela de símbolos da linguagem TPP tendo sido fornecida, é possível criar variáveis no código para armazenar esses símbolos para serem comparados com as traduções.

```
# Expressões Regulaes para tokens simples.
# Símbolos.
t_MAIS = r'\+'
t_MENOS = r'-'
t_MULTIPLICACAO = r'\*'
t_DIVISAO = r'/'
t_ABRE_PARENTESE = r'\('
t_FECHA_PARENTESE = r'\)'
t_ABRE_COLCHETE = r'\['
t_FECHA_COLCHETE = r'\]'
t_VIRGULA = r','
t_ATRIBUICAO = r':='
t_DOIS_PONTOS = r':'

# Operadores Lógicos.
t_E_LOGICO = r'&&'
t_OU_LOGICO = r'\|\|'
t_NEGACAO = r'!'

# Operadores Relacionais.
t_DIFERENCA = r'<>'
t_MENOR_IGUAL = r'<='
t_MAIOR_IGUAL = r'>='
t_MENOR = r'<'
t_MAIOR = r'>'
t_IGUAL = r'='
```

Figura 3. Variáveis no código carregando os símbolos da linguagem TPP.

2.2. Varredura

A varredura é realizada na função *main* do código, ela se baseia em ler o arquivo de entrada valor por valor enquanto houver valor no arquivo. Essa leitura utiliza a função de análise léxica do PLY, *lexer*. O *lexer* possibilita a conversão do valor lido para token e a obtenção da sua tradução (tipo).

```python
def main():
    aux = argv[1].split('.')
    if aux[-1] != 'tpp':
      raise IOError("Not a .tpp file!")
    data = open(argv[1])

    source_file = data.read()
    lexer.input(source_file)

    # Tokenize
    while True:
      tok = lexer.token()
      if not tok:
        break      # No more input
      #print(tok)
      print(tok.type)
      #print(tok.value)
```

Figura 4. Função *main* com varredura do arquivo de entrada.

3. Execução

Para realizar a execução do programa é necessário, primeiramente, possuir o Python e a biblioteca PLY instalada, possível através do comando *pip install ply* em ambientes Linux. Os arquivos disponibilizados para execução são *lextab.py*, um código padrão gerado pela biblioteca PLY e o *tpplex.py*, que é a parte a ser executada. Está disponibilizada também uma pasta com códigos a serem usados de exemplo (lexica-testes).

A execução do programa é, então, realizada executando o arquivo *tpplex.py* com o python, em ambientes Linux isso é possível através do comando *python tpplex.py* seguido do nome do arquivo que será lido.

3.1. Exemplo de execução

Para exemplo de execução, será utilizado o arquivo *fat.tpp*, disponibilizado na pasta lexica-testes. Com os dois arquivos na mesma pasta, o comando para execução será: *python tpplex.py fat.tpp*

O resultado esperado após a execução é a análise léxica, com os valores impressos linha por linha. No exemplo citado, o resultado esperado é:

INTEIRO

DOIS_PONTOS

ID

FLUTUANTE

DOIS_PONTOS

ID
ABRE_COLCHETE
NUM_INTEIRO
FECHA_COLCHETE
INTEIRO
ID
ABRE_PARENTESE
INTEIRO
DOIS_PONTOS
ID
FECHA_PARENTESE
INTEIRO
DOIS_PONTOS
ID
SE
ID
MAIOR
NUM_INTEIRO
ENTAO
ID
ATRIBUICAO
NUM_INTEIRO
REPITA
ID
ATRIBUICAO
ID
MULTIPLICACAO
ID
ID
ATRIBUICAO
ID
MENOS
NUM_INTEIRO

ATE
ID
IGUAL
NUM_INTEIRO
RETORNA
ABRE_PARENTESE
ID
FECHA_PARENTESE
SENAO
RETORNA
ABRE_PARENTESE
NUM_INTEIRO
FECHA_PARENTESE
FIM
FIM
INTEIRO
ID
ABRE_PARENTESE
FECHA_PARENTESE
LEIA
ABRE_PARENTESE
ID
FECHA_PARENTESE
ESCREVA
ABRE_PARENTESE
ID
ABRE_PARENTESE
ID
FECHA_PARENTESE
FECHA_PARENTESE
RETORNA
ABRE_PARENTESE
NUM_INTEIRO

FECHA_PARENTESE

FIM

Se a execução do programa estiver correta, esse resultado deve ser idêntico ao arquivo, também disponibilizado na mesma pasta, de nome *fat.tpp.out*.

4. Testes

Para verificar o funcionamento do programa, foi realizada uma bateria de testes com 32 algoritmos na linguagem TPP fornecidos e comparados com os arquivos de saídas esperadas, também fornecidos (na pasta lexica-testes). O teste geral foi realizado através do sh do Linux, com o comando *run-tests.sh*, tal comando foi executado através de sua versão disponibilizada no Google Colab, com os arquivos de teste e saída esperada sendo colocados na mesma pasta onde está o analisador.

Os resultados foram satisfatórios, com o analisador passando no teste em todos os 32 algoritmos fornecidos, conforme o esperado.

```
------------------------------------------------
Relatório dos Testes:
------------------------------------------------
00. Teste bubble_sort-2020-2.tpp.diff        [OK]
00. Teste bubble_sort_2.tpp.diff             [OK]
00. Teste bubble_sort.tpp.diff               [OK]
00. Teste Busca_Linear_1061992.tpp.diff      [OK]
00. Teste buscaLinear-2020-2.tpp.diff        [OK]
00. Teste comp.tpp.diff                      [OK]
00. Teste fatorial-2020-2.tpp.diff           [OK]
00. Teste fatorial.tpp.diff                  [OK]
00. Teste fat.tpp.diff                       [OK]
00. Teste fibonacci-2020-2.tpp.diff          [OK]
00. Teste fibonacci.tpp.diff                 [OK]
00. Teste hanoi-2020-2.tpp.diff              [OK]
00. Teste insertionSort-2020-2.tpp.diff      [OK]
00. Teste insertSort-2020-2.tpp.diff         [OK]
00. Teste maiorDoVetor.tpp.diff              [OK]
00. Teste multiplicavetor.tpp.diff           [OK]
00. Teste operacao_vetor-2020-2.tpp.diff     [OK]
00. Teste paraBinario-2020-2.tpp.diff        [OK]
00. Teste primo.tpp.diff                     [OK]
00. Teste produtoEscalar.tpp.diff            [OK]
00. Teste prog_test.tpp.diff                 [OK]
00. Teste sample.tpp.diff                    [OK]
00. Teste selectionSort-2020-2.tpp.diff      [OK]
00. Teste selectionsort.tpp.diff             [OK]
00. Teste soma_maior_que_3.tpp.diff          [OK]
00. Teste somavet.tpp.diff                   [OK]
00. Teste subtraiVetores.tpp.diff            [OK]
00. Teste teste-001.tpp.diff                 [OK]
00. Teste teste-002.tpp.diff                 [OK]
00. Teste teste-003.tpp.diff                 [OK]
00. Teste verifica_valor_10.tpp.diff         [OK]
00. Teste verif_num_negativo.tpp.diff        [OK]
------------------------------------------------
OK
------------------------------------------------
```

Figura 5. Resultado dos testes do analisador com o *run-tests.sh*.

Na Figura 5 é possível ver pelo nome do arquivo qual a função de cada um dos algoritmos em TPP, na frente é mostrado um “[OK]”, que confirma que a execução foi um sucesso e o resultado compatível com a saída esperada.

5. Resultados

O algoritmo se provou eficiente, sendo testado nos arquivos de teste fornecidos não houve erros de execução e a saída foi de acordo com o esperado (fornecido também nos arquivos de exemplo). A biblioteca PLY tornou possível a criação do programa por possuir funções específicas para análise léxica, o maior trabalho restante é o de obter e organizar os símbolos e palavras reservadas da linguagem.

Referências

Gonçalves, R. A. (2021) “Análise Léxica”, https://colab.research.google.com/github/rogerioag/tutorial-de-compiladores/blob/master/tppcompiler/01-compiladores-analise-lexica-tpplex.ipynb, Outubro.

Gonçalves, R. A. (2021) “Execução dos Testes da Análise Léxica”, https://colab.research.google.com/drive/1GDRMrJZVDuYzPaaO923zuMLPgxv_OeDI?usp=sharing#scrollTo=BzQXw87XsjOp, Outubro.

Beazley, D. M. (2021) “PLY (Python Lex-Yacc)”, https://www.dabeaz.com/ply/ply.html, Outubro.