Análise Léxica

Wesley Franco Ferreira

[1]Universidade Tecnológica Federal do Paraná (UTFPR)

wesley2ff@gmail.com

Abstract. *One of the bases for a compiler work is do a code scan typed by the user, also called lexical analysis. In this meta-paper will be demonstrated how the implementation of a scan system for the language T++, followed by examples of code and further explanations.*

Resumo. *Uma das bases para o funcionamento de um compilador é realizar uma varredura do código fonte digitado pelo usuário, também chamado de análise léxica. Neste relatório será demonstrado como foi realizada a implementação de um sistema de varreduras para a linguagem T++, seguidos de exemplos de códigos e explicações posteriores.*

1. Introdução

A análise léxica possui o objetivo de realizar varreduras no código-fonte digitado pelo usuário, neste caso, sobre a linguagem T++, tendo o propósito de agrupar caracteres e assim formar lexemas e produzir uma sequência de sımbolos léxicos, também chamados de tokens. Será abordado a seguir conceitos para o entendimento da implementação da principal base de e do primeiro passo para o desenvolvimento de um compilador. Será apresentado os materiais utilizado sendo:a linguagem de programação utilizada e softwares auxiliares.

2. Fundamentação teórica

2.1. Linguagem

Para o entendimento da análise léxica, será abordado a linguagem T++, na qual tem base no idioma português, tendo algum grau de semelhança com a linguagem C. Segue um exemplo de um algoritmo cujo objetivo é calcular se o número é primo ou não.

```
1  inteiro principal()
2          inteiro: digitado
3          inteiro: i
4          i := 1
5          repita
6                  flutuante: f
7                  inteiro: int
8                  flutuante: resultado
9                  f := i/2.
10                 int := i/2
11                 resultado := f - int
12
13                 se  resultado > 0
14                         escreva (i)
```

```
15                      fim
16                      i := i+1
17              ate i <= digitado
18  fim
```

3. Análise léxica

3.1. Tokens

Após detalhada a linguagem, é necessário realizar o reconhecimento do código-fonte escrito pelo usuário. Uma varredura foi feita, capturando os chamados tokens, que são um conjunto de caracteres (lexema), no qual representa uma unidade léxica, como identificadores, palavras reservadas (inteiro, flutuante, retorna, leia, escreva, etc), números, entre outros.

Token	Padrão	Lexema	Descrição
<inteiro, >	Sequência de 'i', 'n', 't', 'e', 'i', 'r', 'o'	inteiro	Palavra Reservada
<fim, >	Sequência de 'f', 'i', 'm'	fim	Palavra Reservada
<=, >	=	=	Atribuição
<inteiro, 18>	Digitos numéricos	18, 1, 6, etc	Número

Figure 1. Exemplos de tokens para a linguagem T++

3.2. Expressões Regulares

Expressões regulares são expressões formadas por conjuntos de caracteres que possuem o objetivo fazer o reconhecimento de padrões em texto. É muito utilizado para obtenção de trechos, endereço ou link de imagens em uma página HTML, modificar formato de texto ou remover caracteres inválidos. Nas expressões regulares, podemos utilizar simbolos para representar os conjuntos de caracteres, obrigatoriedade, e quantidade numérica, como demonstrado com trechos de código na Figura 2 e Figura 3.

```
def t_ID(t):
    r'[A-Za-z_][\w_]*'
    t.type = reserved.get(t.value, 'ID')
    return t
```

Figure 2. Expressão para detecção de identificadores.

```
def t_FLUTUANTE(t):
    r'\d+\.\d+'
    t.value = float(t.value)
    return t
```

Figure 3. Expressão para detecção de números flutuantes.

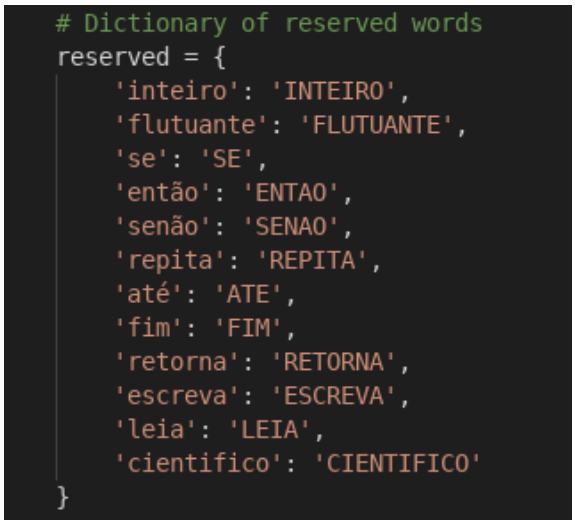
4. Materiais

Python foi a linguagem determinada para a implementação por conter certas bibliotecas que ajudariam a implementação ser mais simples, como por exemplo, a biblioteca Ply, no qual foi utilizado para a detecção de tokens através de expressões regulares, além de realizar a própria varredura do código-fonte.

5. Implementação

5.1. Palavras reservadas e expressões regulares.

Para o reconhecimento de tokens foi utilizada a biblioteca Ply, juntamente com a especificação de palavras reservadas e símbolos. As palavras reservadas foram tratadas utilizando hashmap, no qual a chave seria a própria palavra reservada, e seu valor seria um identificador para a mesma como demonstrado na Figura 4.



```
# Dictionary of reserved words
reserved = {
    'inteiro': 'INTEIRO',
    'flutuante': 'FLUTUANTE',
    'se': 'SE',
    'então': 'ENTAO',
    'senão': 'SENAO',
    'repita': 'REPITA',
    'até': 'ATE',
    'fim': 'FIM',
    'retorna': 'RETORNA',
    'escreva': 'ESCREVA',
    'leia': 'LEIA',
    'cientifico': 'CIENTIFICO'
}
```

Figure 4. Hashmap de palavras reservadas.

5.2. Expressões regulares

As expressões regulares para a formação dos lexemas foram implementados conforme a tabela 4, abrangindo todos os tokens da linguagem T++ especificados na documentação sendo demonstrados na Figura 5.

6. Autômatos

Um autômato recebe uma entrada, a qual seria um conjunto finitos de sımbolos, inicialmente, o autômato se inicia no estado inicial, e conforme a leitura de caracteres, o estado atual é alterado, até a possıvel aceitação, onde o estado atual encontraria o estagio final e a palavra seria reconhecida.

A biblioteca utilizada em Python utiliza autômatos para realizar o reconhecimento de caracteres, juntamente com as expressões regulares, porém de forma abstraída, onde o usuário tem a preocupação de somente analisar e criar a expressão regular para as detecções com base nos seus objetivos. A Figura 6 trata de um autômato o qual representaria a expressão regular para detecção de números flutuantes,onde o intervalo de caracteres é representado pelo sımbolo de subtração (-). Já na Figura 7 é detalhado o automato de detecção para números inteiros.

```
# Regular expressions rulers
t_ADICAO = r'\+'
t_SUBTRACAO = r'-'
t_MULTIPLICACAO = r'\*'
t_DIVISAO = r'/'
t_ATRIBUICAO = r':='
t_IGUAL = r'='
t_MAIOR = r'>'
t_MENOR = r'<'
t_MENOR_IGUAL = r'<='
t_MAIOR_IGUAL = r'>='
t_VIRGULA = r','
t_DOIS_PONTOS = r':'
t_E_LOGICO = r'&&'
t_OU_LOGICO = r'\|\|'
t_NEGACAO = r'\!'
t_ABRE_PARENTESES = r'\('
t_FECHA_PARENTESES = r'\)'
t_ABRE_COLCHETES = r'\['
t_FECHA_COLCHETES = r'\]'
```

Figure 5. Expressões regulares.

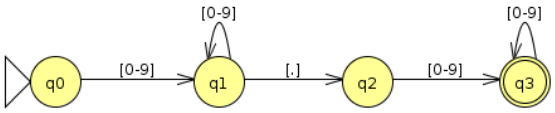


Figure 6. Autômato finito para a detecção de números flutuantes.

7. Resultados e Conclusão

Foram realizados diversos testes com algoritmos na linguagem T++ para a varredura e captura do código, entre eles, Bubble Sort, verificação de números primos, busca linear, ultiplicação de vetor, entre outros. Todos disponibilizados via Moodle da disciplina. O trecho abaixo demonstra a saída que é gerada no terminal e posteriormente gravada em um arquivo chamado "saida$_lex.txt$".

```
1  Linha: 1 | Tipo: INTEIRO | Valor: inteiro
2  Linha: 1 | Tipo: DOIS_PONTOS | Valor: :
3  Linha: 1 | Tipo: ID | Valor: vet
4  Linha: 1 | Tipo: ABRE_COLCHETES | Valor: [
5  Linha: 1 | Tipo: INTEIRO | Valor: 10
6  Linha: 1 | Tipo: FECHA_COLCHETES | Valor: ]
7  Linha: 2 | Tipo: INTEIRO | Valor: inteiro
8  Linha: 2 | Tipo: DOIS_PONTOS | Valor: :
9  Linha: 2 | Tipo: ID | Valor: tam
10 Linha: 4 | Tipo: ID | Valor: tam
11 Linha: 4 | Tipo: ATRIBUICAO | Valor: :=
12 Linha: 4 | Tipo: INTEIRO | Valor: 10
```

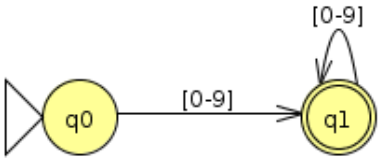


Figure 7. Autômato para a detecção de números inteiros.

```
13 Linha: 7 | Tipo: ID | Valor: preencheVetor
14 Linha: 7 | Tipo: ABRE_PARENTESES | Valor: (
15 Linha: 7 | Tipo: FECHA_PARENTESES | Valor: )
16 Linha: 8 | Tipo: INTEIRO | Valor: inteiro
17 Linha: 8 | Tipo: DOIS_PONTOS | Valor: :
18 Linha: 8 | Tipo: ID | Valor: i
19 Linha: 9 | Tipo: INTEIRO | Valor: inteiro
20 Linha: 9 | Tipo: DOIS_PONTOS | Valor: :
21 Linha: 9 | Tipo: ID | Valor: j
```