Análise Léxica para a linguagem T++

Vitor Yudi Shinohara

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal: Campo Mourão - PR - Brasil

vitor.ysh@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 ou também conhecida como *scanner*, tem como objetivo realizar uma espécie de varredura do 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 símbolos léxicos, também chamados de *tokens*. Abordaremos na seção 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 utilizados tal como linguagem de programação utilizada para implementar o mesmo na seção Materiais, além de como foi feita a implementação e finalmente os resultados.

2. Fundamentação

2.1. Linguagem

Antes de abordarmos propriamente a análise léxica, abordaremos a linguagem *T++*, a 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 o fatorial do número de entrada.

Código 1. Algoritmo fatorial

```
1
2 inteiro: n
3
4 inteiro fatorial(inteiro: n)
5
6    inteiro: fat
7    se n > 0 então {não calcula se n > 0}
8       fat := 1
```

```
9       repita
10         fat := fat * n
11         n := n - 1
12      até n = 0
13      retorna(fat) {retorna o valor do fatorial de n}
14   senão
15      retorna(0)
16   fim
17 fim
18
19 inteiro principal()
20   leia(n)
21   escreva(fatorial(n))
22   retorna(0)
23 fim
```

Através do exemplo a cima, é possível indentificar palavras reservadas, laços de repetições e a estrutura do código, onde é notável visualizar que não é necessário o uso de ponto e vírgula após o término de uma frase e as atribuições são feitas através do símbolo ":=".

2.2. Análise Léxica

2.2.1. Tokens

Após conhecermos a linguagem, é necessário realizar o reconhecimento do código-fonte escrito pelo usuário. É efetuado uma varredura capturando os chamados *tokens*, os quais são um conjunto de caracteres (lexema) o qual representa uma unidade léxica, como identifcador, palavras reservadas (inteiro, flutuante, reforna, leia, escreva, etc), números, entre outros. [Marangon 2017]

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

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

2.2.2. Expressões Regulares

Expressões regulares são expressões formadas por um conjunto de caracteres que tem como objetivo fazer o reconhecimento de padrões em um 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. [Lins 2016]

Nas expressões regulares, podemos utilizar símbolos para representar um conjunto de caracteres, obrigatoriedade, e quantidade numérica, como consta abaixo.

Tabela 2. Representações de caracteres em expressões regulares

Símbolo	Descrição	Exemplo
*	Reconhece nenhum ou vários caracteres	[0-9]*
?	Caractere optativo	[+-]?[0-9]
+	Obrigatoriamente um caractere, no máximo vários	[0-9]+[\.][0-9]
.	Reconhece qualquer caractere	[A-Z].*
{n}	Exatamente n ocorrências	[0-9]{5}

Utilizaremos as expressões regulares com o objetivo de reconhecimento dos lexemas que pertencem a determinado token, tornando possível, assim, a construção da análise léxica em um compilador.

2.3. Autômatos

Um autômato, basicamente é uma estrutura a qual, a partir de uma entrada inicial, julga se a entrada é aceita ou rejeitada. Sua aplicação em compiladores é de grande importância, ainda mais quando tratamos de análise léxica.

Um autômato recebe uma entrada, a qual seria um conjunto finitos de símbolos [Wikipedia 2017], inicialmente, o autômato se inicia no estado inicial, e conforme a leitura de caracteres, o estado atual é alterado, até uma possível aceitação, onde 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 abstraida, onde o usuário tem a preocupação de somente analisar e criar a expressão regular de seu interesse com base nos seus objetivos.

3. Materiais

Foi utilizado como linguagem de programação, o *Python*, por conter certas bibliotecas cujo a implementação seria menos complexa tal como exemplo, o *Ply*, o qual foi utilizado para detectar tokens através de expressões regulares, além de realizar a própria varredura do código-fonte.

4. Implementação

4.1. Tokens

A implementação do reconhecimento dos tokens foi baseado na tabela 3, a qual engloba todos presentes na linguagem abordada. Através disso, foi analisado as devidas expressões regulares para realizar o reconhecimento de todos os elementos que compõe o T++,

As palavras reservadas foram tratadas através de um *hashmap*, o qual a chave seria o a própria palavra reservada, e seu valor seria um identificador para a mesma, mostrado no código 2. O principal motivo dessa implementação é facilitar o reconhecimento das memas, o qual será abordado na seção 4.2.

Código 2. Hashmap para palavras reservadas [Beazley 2017]

```
reserved = {

  'se' : 'SE',
  'então' : 'ENTAO',
  'senão' : 'SENAO',
  'fim' : 'FIM',
  'repita' : 'REPITA',
  'flutuante' : 'FLUTUANTE',
  'retorna' : 'RETORNA',
  'leia' : 'LEIA',
  'até' : 'ATE',
  'escreva' : 'ESCREVA',
  'inteiro' : 'INTEIRO',
  'principal' : 'PRINCIPAL'

}
```

Tabela 3. Tokens da linguagem T++

Palavras Reservadas	Símbolos
se	+ soma
então	- subtração
senão	* multiplicação
fim	/ divisão
repita	= igualdade
flutuante	, vírgula
retorna	:= atribuição
até	<menor
leia	>maior
escreva	<= menor igual
inteiro	>= maior igual
	(abre-par
	) fecha-par
	: dois-pontos
	[abre-col
	[fecha-col
	&& e-logico
	! negação

Para os tokens que não palavras reservadas, foi utilizado um simples vetor contendo os mesmos.

4.2. 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.

Tabela 4. Expressões Regulares

Token	Expressão Regular
SOMA	r'\+'
SUBTRCAO	r'\−'
MULTIPLICACAO	r'\∗'
DIVISAO	r'/'
IGUALDADE	r'\='
VIRGULA	r'\,'
ATRIBUICAO	r'\:\='
MENOR	r'\<'
MAIOR	r'\>'
MENORIGUAL	r'\<\ ='
MAIORIGUAL	r'\>\='
ABREPAR	r'\('
FECHAPAR	r'\)'
DOISPONTOS	r'\ :'
ABRECOL	r'\['
FECHACOL	r'\]'
ELOGICO	r'\&\&'
OULOGICO	r'||'
NEGACAO	r'\!'
COMENTARIO	r'\{[}]*[{]*\}'
FLUTUANTE	r'\d + \.\d+'
INTEIRO	r'\d+'
IDENTIFICADOR	r'[a − zA − Z][a − zA − Z_0 − 9à − ú]∗'

Porém, para o reconhecimento das palavras reservadas, foi implementado no método para a verificação dos identificadores, uma forma de percorrer a tabela *hash*, e averiguar se o lexema está presente na mesma, sendo assim, possível constar se a palavra é reservada, ou apenas um identificador.

Como temos palavras reservadas com caracteres especiais, tais como ”até”e ”então”, é necessário realizar o tratamento na expressão regular, para isso tornar possível, adicionamos os caracteres ”à-ú”, os quais abrangem todos os mesmos, além da impossibilidade de variáveis começarem com o caractere ”_”.

Se tratando de comentários diante o código, foi tomado em consideração que, em muitas vezes, existem comentários os quais ultrapassam de uma linha, sendo assim, reconhecida pela expressão regular. Outro tratamento especial sobre esse token, é contar as quebras de linhas dentro do lexema como mostra o código 3, o que é feito dentro de um método referente ao token, sendo útil posteriormente se tratando de análise semântica e sintática.

Código 3. Método do token COMENTARIO

```
76 def t_COMENTARIO(t):
77   r'\{[^}]*[^{]*\}'
```

```
78    for x in xrange(1,len(t.value)):
79      if t.value[x] == "\n":
80        t.lexer.lineno+= 1
81    return t;
```

O código 3 percorre o lexema do comentário buscando ”\n”, contabilizando assim a variável responsável por calcular a quantidade de linhas no total.

4.3. Autômatos

Abordaremos um autômato o qual representaria a expressão regular do identificador, o qual consta na imagem 3, onde o intervalo de caracteres é representado pelo símbolo de subtração (-), além de caracteres especiais, como já abordados anteriormente, ser representados por ”à-ú”.

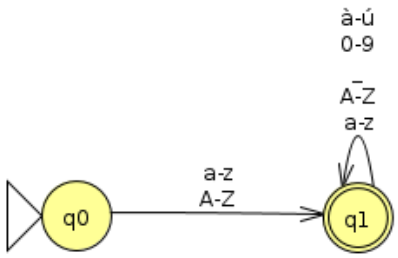

Figura 1. Autômato da expressão regular de identificadores

O estado final do autômato, é o reconhecimento da palavra, sendo alcançado com todas as letras do alfabeto, independente de ser maiúscula ou minúscula. Após isso, pode conter um conjunto de caracteres, sendo números, *underscore* ou caracteres especiais.

Podemos representar outras expressões regulares através de autômatos através das imagens a seguir.

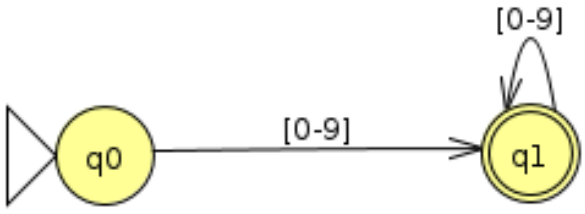

Figura 2. Autômato da expressão regular de números inteiros

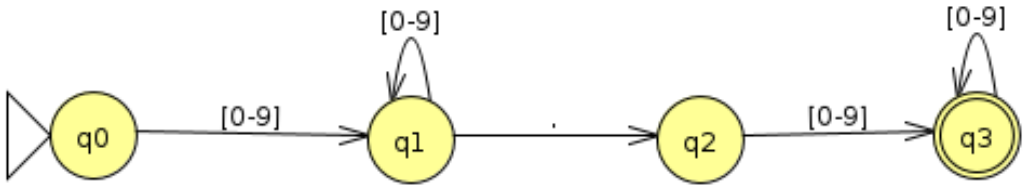

Figura 3. Autômato da expressão regular de números flutuantes

5. Resultados e Conclusão

Foi realizado diversos testes com algoritmos na linguagem T++ para a varredura e captura do código, entre eles, *Bubble Sort*, verificação de números primos, busca linear, multiplicação de vetor, entre outros.

O algoritmo que tomaremos como base é o já abordado anteriormente, algoritmo do cálculo de fatorial (Código 1). Sua saida é representada abaixo.

Código 4. Saída do algoritmo Fatorial

```
<INTEIRO,'inteiro'> : 2
<DOISPONTOS,':'> : 2
<IDENTIFICADOR,'n'> : 2
<INTEIRO,'inteiro'> : 4
<IDENTIFICADOR,'fatorial'> : 4
<ABREPAR,'('> : 4
<INTEIRO,'inteiro'> : 4
<DOISPONTOS,':'> : 4
<IDENTIFICADOR,'n'> : 4
<FECHAPAR,')'> : 4
<INTEIRO,'inteiro'> : 5
<DOISPONTOS,':'> : 5
<IDENTIFICADOR,'fat'> : 5
<SE,'se'> : 6
<IDENTIFICADOR,'n'> : 6
<MAIOR,'>'> : 6
<INTEIRO,'0'> : 6
<ENTAO,'então'> : 6
<COMENTARIO,'{não calcula se n > 0}'> : 6
<IDENTIFICADOR,'fat'> : 7
<ATRIBUICAO,':='> : 7
<INTEIRO,'1'> : 7
<REPITA,'repita'> : 8
<IDENTIFICADOR,'fat'> : 9
<ATRIBUICAO,':='> : 9
<IDENTIFICADOR,'fat'> : 9
<MULTIPLICACAO,'*'> : 9
<IDENTIFICADOR,'n'> : 9
```

```
<IDENTIFICADOR,'n'> : 10
<ATRIBUICAO,':='> : 10
<IDENTIFICADOR,'n'> : 10
<SUBTRACAO,'-'> : 10
<INTEIRO,'1'> : 10
<ATE,'até'> : 11
<IDENTIFICADOR,'n'> : 11
<IGUALDADE,'='> : 11
<INTEIRO,'0'> : 11
<RETORNA,'retorna'> : 12
<ABREPAR,'('> : 12
<IDENTIFICADOR,'fat'> : 12
<FECHAPAR,')'> : 12
<COMENTARIO,'{retorna o valor do fatorial de n}'> : 12
<SENAO,'senão'> : 13
<RETORNA,'retorna'> : 14
<ABREPAR,'('> : 14
<INTEIRO,'0'> : 14
<FECHAPAR,')'> : 14
<FIM,'fim'> : 15
<FIM,'fim'> : 16
<INTEIRO,'inteiro'> : 18
<PRINCIPAL,'principal'> : 18
<ABREPAR,'('> : 18
<FECHAPAR,')'> : 18
<LEIA,'leia'> : 19
<ABREPAR,'('> : 19
<IDENTIFICADOR,'n'> : 19
<FECHAPAR,')'> : 19
<ESCREVA,'escreva'> : 20
<ABREPAR,'('> : 20
<IDENTIFICADOR,'fatorial'> : 20
<ABREPAR,'('> : 20
<IDENTIFICADOR,'n'> : 20
<FECHAPAR,')'> : 20
<FECHAPAR,')'> : 20
<RETORNA,'retorna'> : 21
<ABREPAR,'('> : 21
<INTEIRO,'0'> : 21
<FECHAPAR,')'> : 21
<FIM,'fim'> : 22
```

Como podemos observar, temos como formato de saída <Token, 'lexema'> : linha, sendo assim, uma analise léxica funcional.

Referências

Beazley, D. M. (2017). Ply (python lex-yacc).

Lins, K. (2016). Iniciando expressões regulares.

Marangon, J. D. (2017). Compiladores para humanos.

Wikipedia (2017). Teoria dos autômatos.