Análise Léxica - Compiladores

Otávio Silva Goes[1]

[1]Departamento Acadêmido de Computação - Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 87301 – Campo Mourão – PR – Brazil

ogoes@alunos.utfpr.edu.br

Abstract. *This paper aims to present the process and definition of the lexical analysis step for the creation of a compiler. This work is the first of 4 parts and serves as a basis for the subsequent ones..*

Resumo. *Este relatório tem o objetivo de apresentar o processo e a definição da etapa de análisa léxica para a criação de um compilador. Este trabalho é a primeira de 4 partes e serve de base para as estas subsequentes.*

1. Introdução

A análise léxica é a primeira etapa do processo de compilação de um código fonte, nessa fase do processo o código -e examinado e a partir disso são gerados ***tokens*** que representam cada lexema presente no ”texto”. Os *tokens* são definidos pela linguagem e fazem parte do modo de escrever empregado na mesma.

A linguagem que serviu de base para esse trabalho é a ***T++***, uma linguagem que foi aperefiçoada com base em uma linguagem definida por Kenneth C. Louden em seu livro [Louden 1997], chamada de Tiny. Além de ”aumentar”a linguagem, adicionando estruturas de decisão e repetição e funções, também foi feito um processo de tradução de suas palavras reservadas para que se adequasse ao português brasileiro, visto que a matéria é feitas por pessoas que têm essa lingua como nativa do seu país.

Este relatório está particionado em 4 seções que demonstram o funcionamento e a implementação do programa responsável por identificar os ***tokens*** a partir dos lexemas de entrada que identificam um código escrito na linguagem ***T++***. Na seção 2 será apresentada a especificação da linguagem, bem como os itens que definem toda a sua estrutura léxica, a seção 3 mostra a representação de cada lexema reconhecido pela linguagem por um dispositivo formal (DFA). A implementação bem como a biblioteca usada para esse projeto serão abordados na seção 4, além de exemplos de entrada e seus respectivos resultados na seção 5, a conclusão é feita na seção 6 e abordada os resultados e os próximos passos para o desenvolvimento de um compilador.

2. Especificação da Linguagem

A linguagem descrita neste relatório é a ***T++***, linguagem uma linguagem que foi modificada a partir da linguagem empregada no livro [Louden 1997] de Louden. As modificações aplicadas nesta linguagem vai desde a tradução de suas palavras reservadas até a inclusão de mais estruturas que permitem que esta linguagem se assemelhe mais com a linguagens reais usadas, ser perder o propósito educacional que está ligado a linguagem.

A linguagem é capaz de reconhecer cerca de 35 tipos diferentes de lexemas, incluindo palavras reservadas. A linguagem pode ”reconhecer”desde nomes de variáveis e funções, chamados de *identificadores*, até números em diferentes notações e símbolos comuns em linguagens de programação, os tipos de *tokens* reconhecidos pela linguagem podem ser vistos na figura 1.

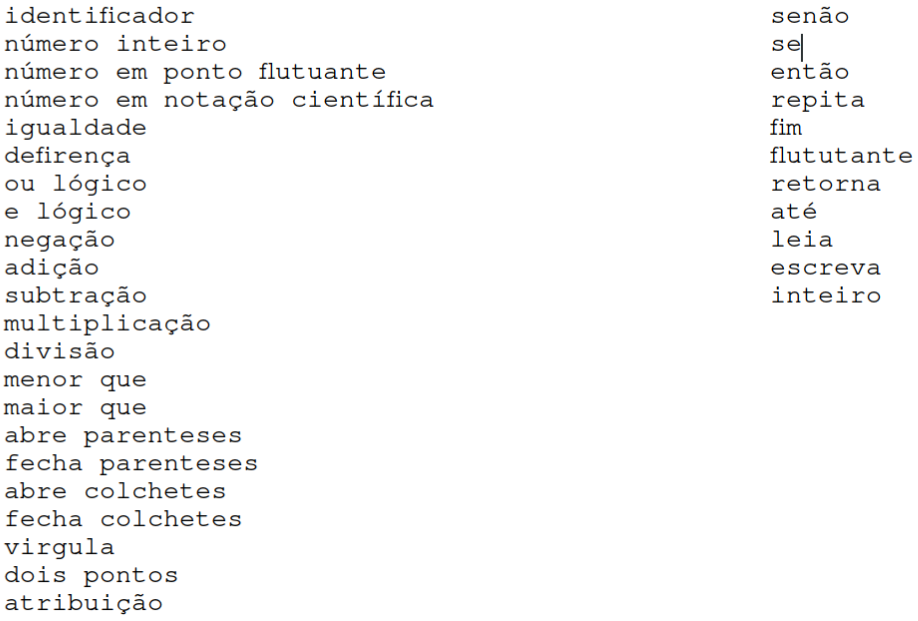

Figura 1. Figura contendo todos os possíveis tipos de *tokens* reconhecidos pela linguagem *T++*

Na figura 1 pode-se obeservar que as palavras reservadas da linguagem estão a direita da figura e representam exatamente o lexema usado no código, diferentemente dos *tokens* dispostos a esquerda, que são descritos dessa forma textualmente mas no código são representados por símbolos diferentes.

3. DFA: Autômatos Finitos Determinísticos

Assim como a todas as linguagens livre de contexto, os lexemas usados pela linguagem podem ser descritos de uma maneira formal por meio de distpositivos formais tais como os autômatos finitos.

Um autômato finito é composto por um conjunto de estados, um conjunto de símbolos reconhecidos pela linguagem,um conjunto de transições responsáveis por ligar os estados uns aos outros, o estado inicial do autômato e um conjunto de possíveis estados finais.

O estado inicial do autômato e marcado com um triângulo na lateral esquerda, os possíveis estados finais são denotados com um círculo na sua borda, além de seu rótolo que identifica o estado. As transições são identificadas pelos símbolos ou classe de

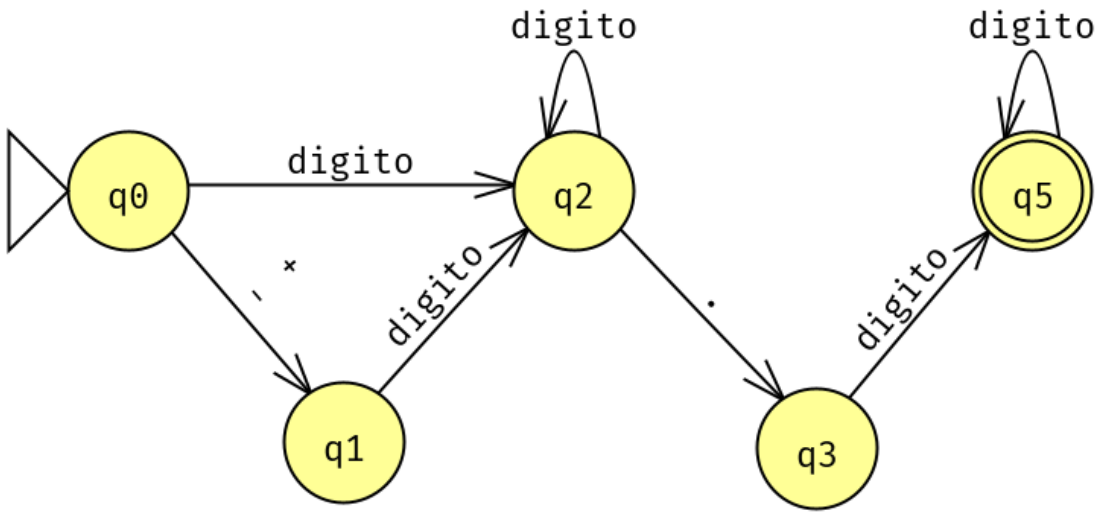

Figura 2. Exemplo de autômato que representa formalmente a identificação da linguagem responsável por ”reconhecer”um número de ponto flutuante

símbolos (ex: letra) que possibilitam a troca de estados no autômato, como pode ser visto na figura 2.

Os autômatos têm uma forte ligação com as expressões regulares empregadas a cada tipo de lexema encontrado no código. As expressões regulares serão explicadas e exemplificadas na seção 4, onde serão abosrdados também assuntos com relação a aplicação dos autômatos e expressões regulares no reconhecimento desses lexemas.

O autômatos servem, nesse caso, somente para formalizar as ”linguagens”livre de contexto que são reconhecidas no código, cada estado final de um autômato ”reconhece”ou identifica os lexemas individualmente no texto, portanto não seria a abordagem adequada para este caso, já que a entrada para o *lexer* é um texto contendo vários lexemas.

4. Implementação

A implementação feita para esta etapa consiste na defição das expressões regulares usadas para identificar os lexemas no texto de entrada, reconhecimento de possíveis erros (que se limitam nessa etapa a símbolos que não são reconhecidos pela linguagem de progamação que o compilador está sendo desenvolvido) e o retorno de uma lista de *tokens* e seus respectivos lexemas reconhecidos do texto.

Todo o projeto foi desnvolvido na linguagem de programação Python com o auxílio da biblioteca PLY [Beazley], uma biblioteca que auxília no uso de ferramentas de *parser* de textos.

O texto é análisado pela biblioteca *token* por *token* até que a entrada de texto termine, assim quando passado um arquivo fonte com o código na linguagem, todo ele é examinado para que se obtenha os *tokens* retirados do mesmo. Um novo *token* é buscado a cada um já encontrado, desde que o texto ainda possua símbolos válidos.

4.1. Expressões Regulares

As expressões regulares têm o propósito de identificar padrões textuais a partir de uma entrada, sendo assim possíveis identificar textos e símbolos que seriam muito difíceis de serem identificados com a leitura humana do texto ou que demandaria muito tempo para que isso ocorresse.

```
identificador = r'[a-zA-Z_]+[a-zA-Z_0-9]*'
```

Figura 3. Exemplo de expressão regular que identifica a um identificador no texto de entrada

4.2. PLY

A biblioteca PLY [Beazley] possue um procedimento específioa quando se trata de reconhecer *tokens* de um texto, é necessário definir o *token* em uma lista e definir uma variável ou função com o nome do *token* e um prefixo ***t_*** que definem a expressão regular desse *token*, como na figura 4.

```
t_ADICAO = r'\+'

def t_NUM_INTEIRO (token):
        r'(([-\+]?)([0-9]+))'
        return token
```

Figura 4. Sintaxe de decaração usada na biblioteca PLY

5. Exemplos

Na figura 5 e 6 são mostrados alguns exemplos de entradas de texto na linguagem ***T++*** e seus respectivos resultados após análise léxica aplicada na entrada.

6. Conclusão

A partir do resultado obtido com a análise léxica dos códigos fintos a etapa seguinte define as regras sintaticas que a linguagem estipula e que devem ser seguidas pelo código de entrada. a partir disso a árvore sintatica pode ser criada e poderá auxiliar na visualização da estruta sintatica do código.

Referências

Beazley, D. M. Ply (python lex-yacc), howpublished = `https://www.dabeaz.com/ply/`, note = Accessed: 2019-09-04.

Louden, K. C. (1997). *Compiler Construction*.

```
inteiro principal()
   inteiro: a
   a := 25.1
   escreva(a)
fim
```

```
<INTEIRO, 'inteiro'>
<ID, 'principal'>
<ABRE_PAR, '('>
<FECHA_PAR, ')'>
<INTEIRO, 'inteiro'>
<DOIS_PONTOS, ':'>
<ID, 'a'>
<ID, 'a'>
<ATRIBUICAO, ':='>
<NUM_PONTO_FLUTUANTE, '25.1'>
<ESCREVA, 'escreva'>
<ABRE_PAR, '('>
<ID, 'a'>
<FECHA_PAR, ')'>
<FIM, 'fim'>
```

Figura 5. Exemplo simples de entrada e saída (I)

```
inteiro principal()
   inteiro: a
   leia(a)
   escreva(a)
fim
```

```
<INTEIRO, 'inteiro'>
<ID, 'principal'>
<ABRE_PAR, '('>
<FECHA_PAR, ')'>
<INTEIRO, 'inteiro'>
<DOIS_PONTOS, ':'>
<ID, 'a'>
<LEIA, 'leia'>
<ABRE_PAR, '('>
<ID, 'a'>
<FECHA_PAR, ')'>
<ESCREVA, 'escreva'>
<ABRE_PAR, '('>
<ID, 'a'>
<FECHA_PAR, ')'>
<FIM, 'fim'>
```

Figura 6. Exemplo simples de entrada e saída (II)