# Projeto e Implementação de uma Ferramenta de Compilação para a Linguagem TPP

Alyson Gonçalves Jorge[1]
[1]Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

**Resumo**

Este relatório apresenta o desenvolvimento de um analisador léxico para a linguagem C–, implementado utilizando a biblioteca `automata_python` e uma máquina de Moore. O analisador léxico é responsável por processar o código-fonte da linguagem, reconhecendo e classificando tokens como palavras-chave, identificadores, operadores e delimitadores.

## 1 Introdução

Esse trabalho tem como objetivo criar um analizador léxido para a linguagem c– utilizando-se de máquina de Moore, para isso esse projeto foi desenvolvido em python utilizando a biblioteca automata_lib, além de uma máquina de moore no simulador JFLAP

## 2 Linguagem C--

A linguagem C– é uma linguagem de programação que visa ser uma abstração da linguagem C, sua sintaxe é mais simples e possui menos recursos de linguagem como mostrado na Figura 1. Foi desenvolvida por Simon Peyton Jones e Norman Ramsey no final dos anos 1990 com o objetivo de servir como uma alternativa ao assembly para a implementação de compiladores. Para a execução desse trabalho deveremos utilizar os seus *tokens* de reconhecimento de código, esses *tokens* estão especificados na Figura 2, além dos *tokens* apresentads na imagem, o analizador deve ser capaz de identificar números(NUMBER) e identificadores de variáveis(ID)

**Corresponding author(s):**

```
program ::= declaration-list
declaration-list ::= declaration-list declaration | declaration
declaration ::= var-declaration | fun-declaration
var-declaration ::= type-specifier ID ; | type-specifier ID [ NUM ] ;
type-specifier ::= int | float | void
fun-declaration ::= type-specifier ID ( params ) compound-stmt
params ::= param-list | void
param-list ::= param-list , param | param
param ::= type-specifier ID | type-specifier ID [ ]
compound-stmt ::= { local-declarations statement-list }
local-declarations ::= local-declarations var-declaration | empty
statement-list ::= statement-list statement | empty
statement ::= expression-stmt | compound-stmt | selection-stmt | iteration-stmt |
    return-stmt
expression-stmt ::= expression ; | ;
selection-stmt ::= if ( expression ) statement | if ( expression ) statement else
    statement
iteration-stmt ::= while ( expression ) statement
return-stmt ::= return ; | return expression ;
expression ::= var = expression | simple-expression
var ::= ID | ID [ expression ]
simple-expression ::= additive-expression relop additive-expression |
    additive-expression
relop ::= <= | < | > | >= | == | !=
additive-expression ::= additive-expression addop term | term
addop ::= + | -
term ::= term mulop factor | factor
mulop ::= * | /
factor ::= ( expression ) | var | call | NUMBER
call ::= ID ( args )
args ::= arg-list | empty
arg-list ::= arg-list , expression | expression
```

Figura 1: Regras Sintáticas da linguagem C–

| palavras reservadas, símbolos, lexemas | Token |
|---|---|
| if | IF |
| else | ELSE |
| int | INT |
| float | FLOAT |
| return | RETURN |
| void | VOID |
| while | WHILE |
| + | PLUS |
| - | MINUS |
| * | TIMES |
| / | DIVIDE |
| < | LESS |
| <= | LESS_EQUAL |
| > | GREATER |
| >= | GREATER_EQUAL |
| == | EQUALS |
| != | DIFFERENT |
| ( | LPAREN |
| ) | RPAREN |
| [ | LBRACKETS |
| ] | RBRACKETS |
| { | LBRACES |
| } | RBRACES |
| = | ATTRIBUTION |
| ; | SEMICOLON |
| , | COMMA |

Figura 2: Tokens da linguagem C– para palavras reservadas, símbolos e lexemas

# 3 Análise Léxica

O analisador léxico é a primeira fase da compilação de um programa. Ele recebe o código-fonte como entrada e o divide em unidades menores chamadas tokens, para esse trabalho os tokens não possuem finalidade pois seriam usados para outras etapas de compilação. Um analisador léxico pode ser implementado utilizando máquinas de moore como mostrado.

### 3.0.1 Máquina de Moore

Uma máquina de Moore é um autômato de saída finito, ou seja, todas as letras do alfabeto devem desencadear em um estado, além disso, a saída deste autômato fica no estado, ou seja, a saída só é alterada quando o estado muda. Esta máquina é perfeita para se criar um analisador léxico pois garante que todos os estados possíveis estão verificados. Sua estrutura é composta por:

- **Conjunto de estados** ($Q$) → Representa os diferentes estados do sistema.
- **Alfabeto de entrada** ($\Sigma$) → Conjunto de símbolos que podem ser lidos como entrada.
- **Função de transição** ($\delta : Q \times \Sigma \rightarrow Q$) → Define como os estados mudam com base na entrada.
- **Função de saída** ($\lambda : Q \rightarrow \Gamma$) → Define a saída para cada estado.
- **Estado inicial** ($q_0 \in Q$) → O estado em que a máquina começa.
- **Conjunto de saída** ($\Gamma$) → Possíveis valores que a máquina pode gerar como saída.

. um exemplo de um protótipo de analisador léxico pode ser observado na Figura 3

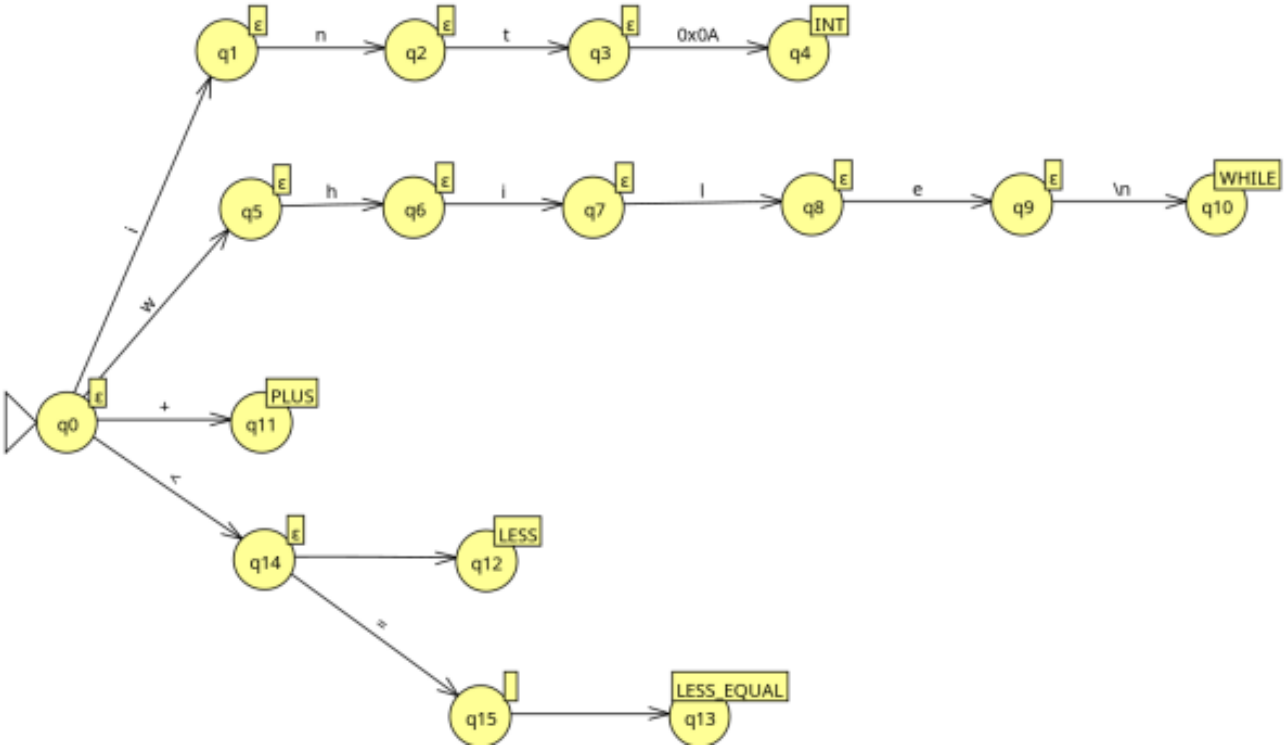


Figura 3: protótipo de analisador léxico utilizando máquina de moore

# 4 Implementação

## 4.1 Primeiros passos

Para começar a implementação foi cedido um material base fornecido pelo professor, nele, temos um código em python utilizando a máquina de Moore implementado pela biblioteca `automata_lib` como visto na Figura 5, um sistemas de testes da biblioteca pytest, a estrutura

do arquivo e o template latex utilizado para realização deste relatório.

## 4.2 configurando a máquina

para melhorar a qualidade do código, a máquina de Moore foi modularizada em vários arquivos, garantindo assim um código mais conciso. Ela está configurada em:

- Estados: Foram feitos 115 estados nesse projeto, por conta de alguns estados precisarem gerar 2 tokens na saída, os estados podem ser encontrados no arquivo *estados.py*
- Alfabeto: o alfabeto de entrada contempla todos os números de 0-9 e todas as letras de a-z, ele conta também com os caracteres: '+', '-', '*', '/', '<', '>', '=', '!', '(', ')', '[', ']', '', '', ';', ',', '.', ' ', '', e pode ser localizado no arquivo *Alfabeto.py*
- Alfabeto de saídas: O alfabeto de saída ou melhor dizendo, os *tokens* que o analizador léxico gera podem ser encontrados na pasta saidas.py, os *tokens* são os já discutidos na seção 2
- transições: As transições estão no arquivos *transicoes.py*, por se tratar de um autômato determinístico as transições ultrapassaram as 1000 linhas.
- estado inicial: O estado inicial pode ser encontrado no arquivo *q0.py* e possui todas as transições iniciais de identificação
- tabelade saídas: as tabelas de saídas podem ser encontradas no arquivo *tabelaSaidas.py*

Após a configuração a máquina de Moore ficou como demonstrado na Figura 4

```
from automata.fa.Moore import Moore
import sys, os
from myerror import MyError
from alfabeto import alfabeto
from estados import estados
from transicoes import transicoes
from saidas import  alfabetoSaidas, tabelaSaidas

error_handler = MyError('LexerErrors')

global check_cm
global check_key

moore = Moore(
    estados,
    alfabeto,
    alfabetoSaidas,
    transicoes,
    'q0',
    tabelaSaidas
)
```

Figura 4: Máquina de Moore após a configuração

```
# from automata.fa.Moore import Moore
import sys, os

from myerror import MyError

error_handler = MyError('LexerErrors')

global check_cm
global check_key

# moore = Moore(['q0', 'q1', 'q2', 'q3', 'q4'],
#               ['i' , 'n' , 't', ' '],
#               ['INT', 'ELSE'],
#               {
#                   'q0' : {
#                       'i' : 'q1',
#                   },
#                   'q1': {
#                       'n': 'q2',
#                   },
#                   'q2': {
#                       't': 'q3',

#                   },
#                   'q3': {
#                       '\n': 'q4',
#                   }
#               },

#               'q0',
#               {
#                   'q0' : '',
#                   'q1' : '',
#                   'q2' : '',
#                   'q3' : '',
#                   'q4' : 'INT'
#               }
#               )
```

Figura 5: Código inicial da Máquina de Moore

# 5 Autômato implementado

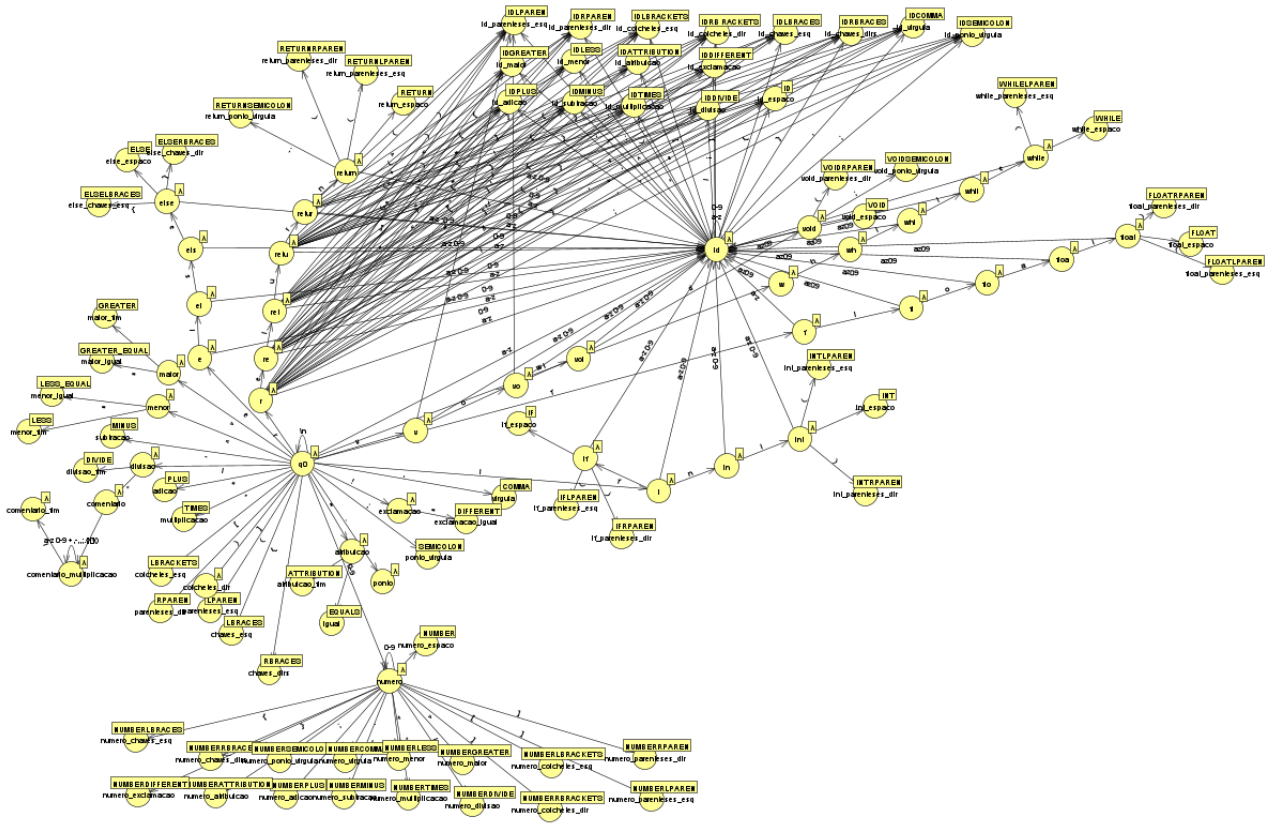

Figura 6: Autômato implementado no JFLAP

# Referências