Projeto e Implementação de uma Ferramenta de Compilação para a Linguagem C-

Yagho Junior Petini[1]
[1]Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

Abstract

The present work presents the development of a lexical analyzer for the C- language. A Moore Machine was used for the recognition and generation of *tokens*. The main objective of the project is to demonstrate the development of the initial phase of a compiler. For this purpose, the *Python* programming language and its *automata-lib* library were used, thus facilitating the implementation of a Moore Machine capable of identifying the lexical patterns of the language and efficiently generating the *tokens*. The implementation followed a structured process, covering the definition of states, transitions, initial state, input and output alphabet, and output table, as well as the results obtained with this tool.

Resumo

O presente trabalho apresenta o desenvolvimento de um analisador léxico para a linguagem C-. Foi feito o uso de uma Máquina de Moore para o reconhecimento e geração dos *tokens*. O projeto tem como principal objetivo demonstrar como é o desenvolvimento da fase inicial de um compilador, para isso, foi utilizado a linguagem *Python* e sua biblioteca automata-lib, dessa forma facilitando a implementação de uma Máquina de Moore que seja capaz de identificar os padrões léxicos da linguagem e gerar os *tokens* de forma eficiente. A implementação seguiu um processo estruturado, foi mostrado desde a definição dos estados, transições, estado inicial, alfabeto de entrada e saída e tabela de saída até os resultado obtidos com o tal ferramenta.

1 Introdução

Antes de adentrar no trabalho que será apresentado ao leitor, tem-se que o universo da computação frequentemente se inicia com uma pergunta, *"O que é um computador?"*. Ainda que os computadores possam ser descritos matematicamente, os dispositivos atuais são altamente complexos para uma análise de seu funcionamento. Por conta disso, iremos utilizar modelos computacionais mais simples e idealizados, estes que servem para resolver algum problema especifico, mesmo que deixando brechas para outros problemas.

Dessa forma, o modelo optado para seguirmos com a explicação será o automato finito, esse modelo é muito utilizado sistemas que possuem tarefas baseadas em estados ou transições desses estados. Os autômatos finitos podem ser determinísticos (AFD), com uma única transição possível para cada símbolo de entrada, ou não determinísticos (AFND), onde múltiplas transições são permitidas para um determinado símbolo.

No presente projeto, será desenvolvido um "Analisador Léxico"a partir de uma Máquina de Moore, ela funciona como um automato finito que associa uma saída aos estados. Dessa forma

Corresponding author(s): Yagho Petini, Universidade Tecnológica Federal do Paraná, yaghopetini@alunos.utfpr.edu.br

a Máquina de Moore se torna extremamente previsível e útil para aplicações como analisadores léxicos e também para reconhecimento de padrões. No uso dessa ferramenta para a aplicação em analisadores léxicos sua previsibilidade se torna util para reconhecer padrões de uma linguagem e a partir disso, gerar e classificar *token* de uma sequência de caracteres de entrada, assim, tornando o processo de compiladores mais eficientes e estruturado por facilitar a compreensão de um código fonte.

2 Token

No contexto de um analisador léxico, o *token* é a unidade léxica mínima que pode ser reconhecida pelo compilador, representando um elemento válido da linguagem de programação que será compilada. O *token* pode variar, podendo ser classificado como palavra-chave, identificador, operadores, números, entre outros. Na linguagem C-, que será abordada neste projeto, dividiremos seus elementos em um dos tipos apresentados.

Para exemplificar como será a classificação dos elementos de um código fonte para os *tokens*, o código 1 trás um trecho de um algoritmo implementado na linguagem *C-*. Já na tabela 1 Tabela 1 está demonstrando a classificação dos elementos desse algoritmo em *tokens*.

Código 1: Exemplo de Código em C-

```
1 int a = 1 + 2;
```

int	Palavra-chave
a	Identificador
=	Operador de Atribuição
1	Número
+	Operador de Soma
2	Número
;	Símbolo Especial

Tabela 1: Tokens do Código 1 de Exemplo

3 Máquina de Moore

A Máquina de Moore é um *AFD* com saídas baseadas diretamente aos estados. Normalmente é representada por uma septupla, sendo composta dos seguintes componentes: conjunto de estados, alfabeto de símbolo de entrada, função de transição, estado inicial, conjunto de estados finais, alfabeto de símbolos de saída e função de saída. A sua saída depende dos estados em que máquina ”passará”a partir de uma cadeia de entrada.

As aplicações da Máquina de Moore estão sempre relacionados a componentes e implementação onde a saída precisa ser instável ou padronizada. Dessa forma, pode-se citar o uso dessa ferramenta para o desenvolvimento de circuitos digitais, compiladores e reconhecimento de padrões.

Algumas das vantagens da utilização de uma Máquina de Moore está na sua simplificação de ser implementada, veja o exemplo de sua implementação na Figura 1, a estabilidade de suas saídas, reduzindo a chance de erros e também a sua previsibilidade de resultado, que em muitos casos se torna útil em aplicações críticas. Dessa forma, como pode-ser visto na Figura 1 que cada estado possui uma saída, podendo ser vazia ou não. Nota-se que se a cadeia de entrada for a *string WHILE*, a máquina de moore irá carácter por carácter reconhecendo o padrão dessa cadeia, e se chegar ao final e os estados presentes para formar o *token WHILE*, ela irá resultar na saída *WHILE*.

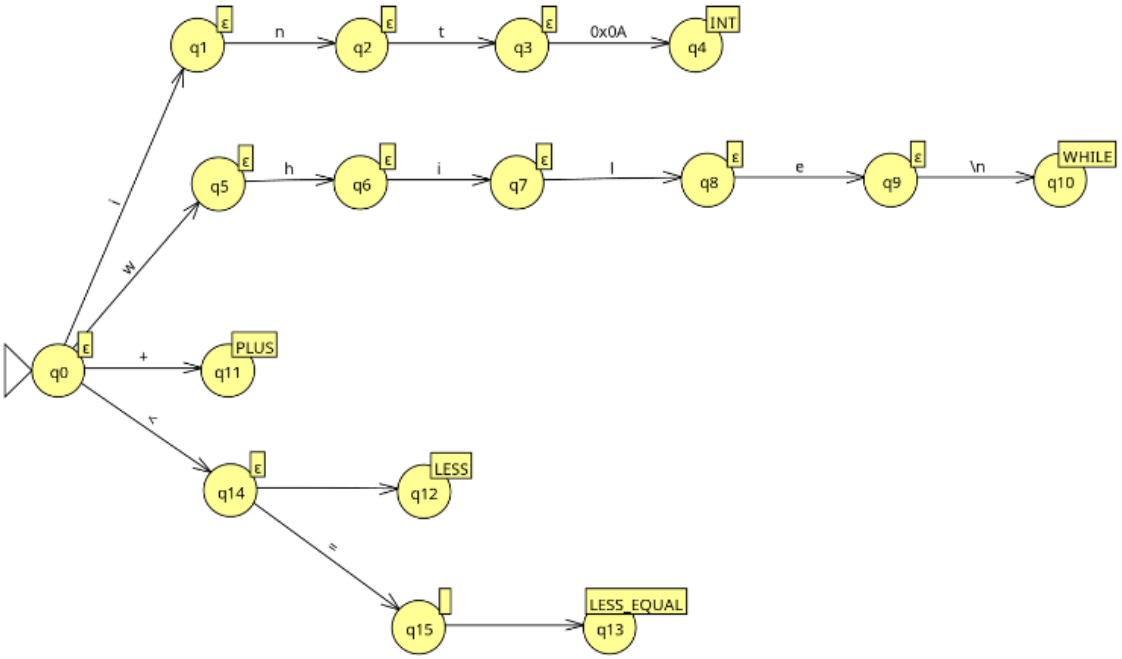

Figura 1: Máquina de Moore

4 Analisador Léxico

O analisador léxico, ou *scanner* é a primeira etapa do processo de compilação de um código fonte, trata-se da implementação de um automato finito que tem como função reconhecer símbolos válidos de uma linguagem. Dessa forma, a partir de uma sequência de símbolos de entrada o analisador léxico irá percorrer cada carácter associá-lo ao seu respectivo *token*.

A partir disso, a saída desse componente será uma lista de *tokens* que posteriormente será utilizada por outra etapa do processo de compilação chamada "Analisador Sintático". É função do *scanner* detectar os erros mais simples, como o uso de um caracter inválido pela linguagem, assim o analisador deve retornar um erro para o usuário, dessa forma impedindo que o programa siga no processo de compilação. Abaixo, no código 2 ilustra a saída do analisador léxico, demonstrando que todo e qualquer carácter deve ser analisado e classificado em um determinado *token*.

Código 2: Lista de Tokens

```
1  Entrada:
2  int main(void){
3
4  }
5
6
7  Lista de Tokens:
8  INT
9  ID
10 LPAREN
11 VOID
12 RPAREN
13 LBRACES
14 RBRACES
```

5 Implementação

A implementação do analisador léxico será realizada utilizado a linguagem de programação *Python* e algumas de suas bibliotecas, sendo a principal *automata-lib* que será responsável por criar e manipular uma Máquina de Moore. O objetivo desse analisador léxico será reconhecer e gerar os *tokens* da linguagem de programação C-, uma versão reduzida da linguagem mundialmente conhecida C.

O desenvolvimento foi dividido em várias etapas. Inicialmente foi elaborado um esboço do automato responsável pela geração dos *tokens*, este que foi desenvolvido utilizando o software *JFLAP*. Em sequência, foi iniciado o desenvolvimento do algoritmo do analisador, foi utilizado o editor de texto *VS Code*. A partir do desenvolvimento, foi criado os elementos principais que necessitam ser fornecidos para que a Máquina de Moore da *automata-lib* possa analisar a cadeia de entrada e fornecer uma saída. Cada uma das partes mencionadas estarão detalhadas abaixo.

5.1 Estados

A linguagem C- possui um conjunto de estados bem definidos, projetados para reconhecer todos os elementos léxicos da linguagem, como palavras-chave, identificadores, números, operadores e símbolos especiais. Dessa forma, a linguagem poderia passar por um analisador léxico independente da entrada, assim, facilitando todo o processo de compilação. Abaixo, estão listados os principais estados da linguagem C- em conjunto com os estados auxiliriares que foram criados para o desenvolvimento do presente projeto.

Código 3: Lista de Estados

```
1     q0, id, number,
2     i, in, int, if,
3     if_esq_parenteses, if_final,
4     int_esq_parenteses, int_dir_parenteses, int_final,
5     e, el, els, else,
6     else_esq_chaves, else_final,
7     r, re, ret, retu, retur, return
8     return_esq_parenteses, return_ponto_virgula, return_final,
9     v, vo, voi, void,
10    void_dir_parenteses, void_ponto_virgula, void_final,
11    w, wh, whi, whil, while,
12    while_esq_parenteses, while_final,
13    f, fl, flo, floa, float,
14    float_esq_parenteses, float_dir_parenteses, float_final,
15    adicao, subtracao, multiplicacao, divisao,
16    divisao_finalizado, divisao_equal, divisao_id, divisao_number,
17    maior, maior_igual, menor, menor_igual,
18    maior_finalizado, menor_finalizado,
19    atribuicao, igual,
20    atribuicao_finalizado,
21    exclamacao, exclamacao_igual,
22    virgula, ponto_virgula, ponto,
23    esq_parenteses, dir_parenteses, esq_colchetes, dir_colchetes,
24    esq_chaves, dir_chaves,id_esq_parenteses, id_dir_parenteses,
25    id_esq_colchetes, id_dir_colchetes,id_esq_chaves, id_dir_chaves,
26    id_ponto_virgula, id_virgula, id_final,id_adicao, id_subtracao,
27    id_multiplicacao, id_divisao, id_maior, id_menor, id_atribuicao,
          id_exclamacao,
28    number_esq_parenteses, number_dir_parenteses, number_esq_colchetes,
29    number_dir_colchetes, number_esq_chaves, number_dir_chaves,
30    number_ponto_virgula, number_virgula, number_final,
```

```
31      number_adicao, number_subtracao, number_multiplicacao, number_divisao,
            number_maior, number_menor, number_atribuicao, number_exclamacao,
32      divisao_multiplicacao, comentario, comentario_multiplicacao,
            comentario_finalizado
```

5.2 Alfabeto de Entrada

O Alfabeto de entrada será os símbolos que a Máquina de Moore poderá processar, é essencial para definir as transições que poderão ser feitas. Sem um alfabeto de entrada bem definido, a máquina não será capaz de processar as cadeias de carácteres de entrada corretamente. Abaixo está o alfabeto de entrada escolhido para a realização desse projeto.

Código 4: Alfabeto de Entrada

```
1  alfabeto_entrada = [
2      a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
           ,
3      +, -, *, /, <, >, =, !, (, ), [, ], {, }, ;, ,, .,
4      0, 1, 2, 3, 4, 5, 6, 7, 8, 9,  , \n
5  ]
```

5.3 Símbolo Inicial

O símbolo inicial é o ponto de partida do autômato, no caso desse projeto, da Máquina de Moore. Ele representa o estado que a máquina começa antes de iniciar o processamento de uma entrada. No analisador léxico o símbolo inicial adotado no projeto foi o *"q0"*.

5.4 Transições

Assim como os estados, na linguagem C- as transições são bem definidas, projetadas para cobrir todos os elementos léxicos da linguagem, dessa forma, dando garantia que o analisador léxico funcione de maneira eficiente. Cada transição é definida a partir das regras gramáticais da linguagem, permitindo que o analisador consiga classificar tanto os símbolos mais simples, quanto os mais complexos. Abaixo no código 5 está um exemplo de como irá funcionar as transições implementadas nesse projeto. Estas, que foram feitas utilizando dicionários da linguagem *Python*.

Código 5: Exemplo de Transição da Linguagem C-

```
1      'id':{
2          **{chr(caracter): 'id' for caracter in range(ord('a'), ord('z') + 1)},
3          **{chr(caracter): 'id' for caracter in range(ord('0'), ord('9') + 1)},
4          '+' : 'id_adicao',
5          '-' : 'id_subtracao',
6          '*' : 'id_multiplicacao',
7          '/' : 'id_divisao',
8          '>' : 'id_maior',
9          '<' : 'id_menor',
10         '=' : 'id_atribuicao',
11         '!' : 'id_exclamacao',
12         '.' : 'id',
13         ',' : 'id_virgula',
14         ';' : 'id_ponto_virgula',
15         '(' : 'id_esq_parenteses',
16         ')' : 'id_dir_parenteses',
17         '[' : 'id_esq_colchetes',
```

```
18          ']' : 'id_dir_colchetes',
19          '{' : 'id_esq_chaves',
20          '}' : 'id_dir_chaves',
21          ' ' : 'id_final',
22          '\n' : 'id_final',
23      },
```

5.5 Tabela de Saída

A tabela de saída é uma estrutura que associa o estado final a um respectivo *token* que ele representará. Dessa forma, quando a Máquina de Moore atingir um determinado *token*, ela irá consultar essa tabela de saída afim de verificar se existe um *token* para emitir. Esse processo é essencial para o analisador léxico produzir as saídas corretar independente da entrada. Abaixo está um exemplo da tabela de saída implementada no presente projeto.

Código 6: Tabela de Saída

```
1  tabela_saida = {
2
3      q0: ,
4      #Saida dos IDs
5      id: ,
6      id_esq_parenteses :ID\nLPAREN\n,
7      id_dir_parenteses:ID\nRPAREN\n,
8      id_esq_colchetes:ID\nLBRACKETS\n,
9      id_dir_colchetes:ID\nRBRACKETS\n,
10     id_esq_chaves:ID\nLBRACES,
11     id_dir_chaves:ID\nRBRACES,
12     id_ponto_virgula : ID\nSEMICOLON\n,
13     id_virgula : ID\nCOMMA\n,
14     id_adicao : ID\nPLUS\n,
15     id_subtracao : ID\nMINUS\n,
16     id_multiplicacao : ID\nTIMES\n,
17     id_divisao : ID\nDIVIDE\n,
18     id_maior : ID\nGREATER\n,
19     id_menor : ID\nLESS\n,
20     id_atribuicao : ID\nATTRIBUTION\n,
21     id_exclamacao : ID\nDIFFERENT\n,
22     id_final:ID\n,
23
24     #Saída do INT
25     i:,
26     in:,
27     int:,
28     int_final: INT\n,  # Estado final do int
29     int_esq_parenteses : INT\nLPAREN\n,
30     int_dir_parenteses : INT\nRPAREN\n,
31 }
```

5.6 Automato de Reconhecimento

O autômato de reconhecimento é a parte central do analisador léxico, pois é responsável por identificar e classificar os *tokens* da linguagem C-. No presente projeto, ele foi implementado como uma Máquina de Moore, onde cada estado irá representar um passo do reconhecimento de um *tokens*, as transições entre esses estados são determinados pela cadeia de caracteres de entrada. Dessa forma, o *token* será gerado a partir de um estado que tenha uma saída, assim como

explicado na seção sobre a Máquina de Moore, geralmente sendo os estados finais na geração de um *token*.

A Máquina de Moore utilizada neste projeto está sendo ilustrado na Figura 2. Seu funcionamento é bastante intuitivo, seguindo um processo claro de reconhecimento de padrões. Por exemplo, ao receber uma cadeia de caracteres ”INT ”como entrada, a máquina começar sua análise pelo estado inicial, denominado de *”q0”*. A partir do estado inicial, ela processará em sequência cada um dos caracteres da entrada, dessa forma:

1. O primeiro carácter lido será o *”i”*, a máquina verifica se há alguma transição válida a partir do estado atual, neste momento é o *”q0”*, utilizando o esse simbolo lido como entrada.
2. Após encontrar uma transição para o carácter *”i”*, a máquina vai para o estado *”i”*, e processa o próximo carácter da entrada e novamente verifica se existe uma transição para esse símbolo.
3. O terceiro carácter lido será o *”t”*, dessa forma completando a sequência da palavra-chave *”INT ”*.
4. O ultimo passo, é a máquina encontrar o carácter de saída, nesse caso o , pois a partir dele, a máquina irá transitar para o estado final desse *token*, reconhecendo totalmente essa entrada e classificando-a como o *token INT*.

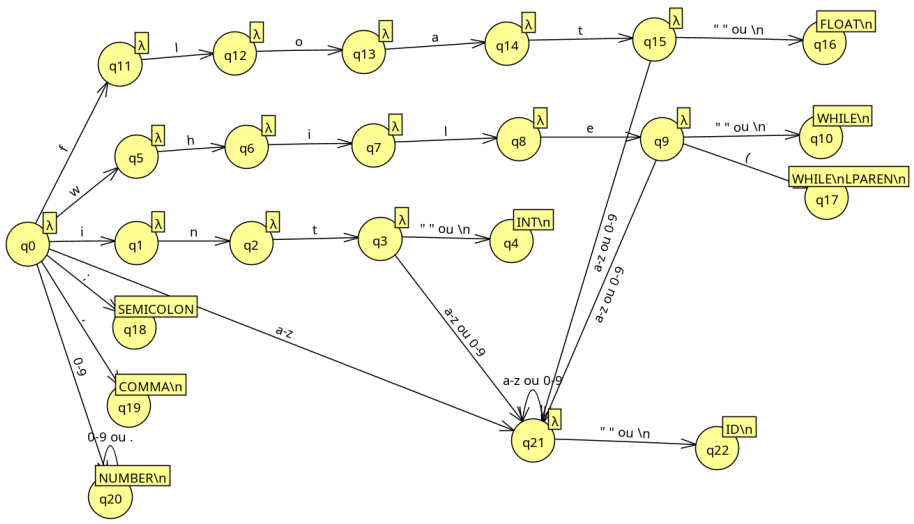

Figura 2: Máquina de Moore

A partir disso, tem-se que ao atingir um estado final, a máquina irá emitir um *token* como saída. Esse processo irá ser o mesmo para a classificação de qualquer entrada válida. Os possíveis *tokens* gerados estão na Lista de *tokens* 7.

Código 7: Tokens da Linguagem C-

```
1     ID,
2     NUMBER,
3     INT,
4     IF,
5     ELSE,
6     RETURN,
7     VOID,
8     WHILE,
9     FLOAT,
10    PLUS,
11    MINUS,
12    TIMES,
13    DIVIDE,
```

```
14        GREATER,
15        GREATER_EQUAL,
16        LESS,
17        LESS_EQUAL,
18        ATTRIBUTION,
19        EQUALS,
20        DIFFERENT,
21        COMMA,
22        SEMICOLON,
23        LPAREN,
24        RPAREN,
25        LBRACKETS,
26        RBRACKETS,
27        LBRACES,
28        RBRACES,
```

5.7 Bibliotecas Utilizadas

Como já foi dito, para a implementação do analisador léxico foi utilizado a linguagem de programação *Python* e algumas de suas bibliotecas. As principais foram a *automata-lib* para utilizar a Máquina de Moore que já está implementada nela, dessa forma, foi apenas fornecidos os elementos necessários para que a máquina funcione, sendo eles: Estados, Alfabeto de Entrada, Alfabeto de Saída, Transições, Estado Inicial e Tabela de Saída, *sys* e *os* foram as bibliotecas utilizadas para pegar o comando de execução do terminal e verificar se estava sendo passado os argumentos corretos para executar o programa de forma correta.

6 Resultado

O resultado obtido a partir da implementação do analisador léxico mostrado anteriormente foi bastante satisfatório, o uso da Máquina de Moore para reconhecer os padrões da linguagem C- se mostrou muito eficaz. Independentemente da entrada, o analisador léxico pode analisar e gerar uma lista precisa dos *tokens* corretamente classificados. Abaixo no 8 está uma demonstração do resultado obtido no projeto.

Código 8: Exemplo do Resultado do analisador léxico

```
1  Entrada:
2  int a;
3  float b;
4
5  int main(void){
6    int x;
7    float y;
8
9    return(0);
10 }
11
12
13 Lista de Tokens:
14 INT
15 ID
16 SEMICOLON
17 FLOAT
18 ID
19 SEMICOLON
20 INT
```

```
21 ID
22 LPAREN
23 VOID
24 RPAREN
25 LBRACES
26 INT
27 ID
28 SEMICOLON
29 FLOAT
30 ID
31 SEMICOLON
32 RETURN
33 LPAREN
34 NUMBER
35 RPAREN
36 SEMICOLON
37 RBRACES
```

7 Conclusão

Conclui-se que o desenvolvimento do analisador léxico para a linguagem C- é bastante complexo e trabalhoso, mas utilizando uma Máquina de Moore para auxiliar no reconhecimento dos padrões e geração dos *tokens* o trabalho fica relativamente mais simples. Contudo, é válido lembrar da ferramenta de desenvolvimento *Python* que também foi de extrema importância por sua abrangência e facilidade com o uso de sua biblioteca *automata-lib*.

Portanto, após os resultados obtidos com o analisador léxico é importante ressaltar como os modelos teóricos, como os autômatos finitos e o seu uso em ferramentas, como a Máquina de Moore, podem ser aplicados para resolver um problema reais de forma eficiente.

Referências

Referências

[1] SIPSER, Michael. **Introdução à Teoria da Computação: Trad. 2ª ed. norte-americana**. Porto Alegre: +A Educação - Cengage Learning Brasil, 2007. E-book. p. 82. ISBN 9788522108862. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788522108862/. Acesso em: 29 jan. 2025.