

Linguagens Formais, Autômatos e Computabilidade

Breno Farias da Silva

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão – PR – Brazil

²Departamento Acadêmico de Computação
(DACOM)

{brenofarias@alunos.utfpr.edu.br}

Resumo. *O relatório está dividido em duas partes, a primeira reservada para compreender o raciocínio usado na criação do autômato no simulador JFLAP e em explicar, de forma breve, o raciocínio desenvolvido no algoritmo criado na linguagem de programação C. É relevante mencionar que será explicado a teoria por trás da máquina de moore no decorrer deste artigo. Tem-se como objetivo projetar e implementar um autômato com saída, do tipo Máquina de Moore, que funcione como um Analisador Léxico para a linguagem C- (CM).*

1. Introdução

Inicialmente, antes de falar sobre as implementações desenvolvidas, é necessário entender a teoria que acompanha a máquina de moore. Existem dois tipos de autômatos com saída, a máquina de mealy e a máquina de moore. A máquina de mealy trata de gerar uma saída na transição entre estados, já a máquina de moore, trata de gerar uma saída ao atingir um estado. Uma máquina de moore é dada pela tupla $M = (Q, \Sigma, \delta, Q_0, F, \Delta, \delta_s)$. O símbolo Q refere-se a todos os estados do autômato, o símbolo Σ refere-se ao alfabeto, o que consiste nos símbolos de entrada do autômato, δ é a função de transição, Δ refere-se ao alfabeto de saída, Q_0 refere-se ao símbolo inicial e δ_s as saídas do autômato.

2. Autômato

A seguir tem-se o autômato construído para representar a máquina de moore. Seguindo a tupla explicada anteriormente, tem-se que o estado inicial Q_0 é dado pelo estado q_0 . O alfabeto Σ do autômato consiste em todas as letras, números e símbolos do alfabeto, visto que, quando tem-se como regra que os nomes de variáveis podem usar letras maiúsculas ou minúsculas (de A a Z, sem acentos), algarismos arábicos (0-9) e o caractere sublinhado ($_$), mas o primeiro caractere deve ser obrigatoriamente uma letra ou o sublinhado. O conjunto Q de estado consistem nos estados que variam de Q_0 até Q_{42} . Os estados finais, F , é sempre que uma palavra é reconhecida, ou seja, seriam os estados $Q_2, Q_4, Q_{10}, Q_{14}, Q_{18}, Q_{23}, Q_{28}$ e Q_{51} e do estado Q_{29} até o estado Q_{48} . A função programa ou de transição $\delta: Q \times \Sigma \rightarrow Q$ consiste em, para cada símbolo do alfabeto, ter uma transição de um estado para outro. Por exemplo, ao situar-se no estado Q_3 , com o símbolo "n" de entrada, há a transição para o estado Q_4 . Esse algoritmo repete-se para todos os símbolos do alfabeto e para todos os estados. Por fim, a função de saída $\delta_s: Q \rightarrow \Delta$ a qual é uma função total. O alfabeto de saída Δ consiste em todos os símbolos que formam as palavras: IF, ELSE, INT, RETURN, VOID, WHILE, PLUS, MINUS, TIMES, DIVIDE, LESS,


```

#define VOID "void"
#define WHILE "while"

// Defining the keywords output
#define IF_OUTPUT "IF"
#define ELSE_OUTPUT "ELSE"
#define INT_OUTPUT "INT"
#define FLOAT_OUTPUT "FLOAT"
#define FOR_OUTPUT "FOR"
#define RETURN_OUTPUT "RETURN"
#define VOID_OUTPUT "VOID"
#define WHILE_OUTPUT "WHILE"

// Defining the symbols
#define PLUS '+'
#define MINUS '-'
#define MULTIPLY '*'
#define DIVIDE '/'
#define LESS_THAN '<'
#define LESS_THAN_EQUAL '<='
#define GREATER_THAN '>'
#define GREATER_THAN_EQUAL '>='
#define EQUAL '='
#define SEMICOLON ';'
#define COMMA ','
#define LEFT_PARENTHESIS '('
#define RIGHT_PARENTHESIS ')'
#define LEFT_BRACKET '['
#define RIGHT_BRACKET ']'
#define LEFT_BRACE '{'
#define RIGHT_BRACE '}'

// Defining the symbols output
#define PLUS_OUTPUT "PLUS"
#define MINUS_OUTPUT "MINUS"
#define MULTIPLY_OUTPUT "MULTIPLY"
#define DIVIDE_OUTPUT "DIVIDE"
#define LESS_THAN_OUTPUT "LESS_THAN"
#define LESS_THAN_EQUAL_OUTPUT "LESS_THAN_EQUAL"
#define GREATER_THAN_OUTPUT "GREATER_THAN"
#define GREATER_THAN_EQUAL_OUTPUT "GREATER_THAN_EQUAL"
#define EQUAL_OUTPUT "EQUAL TO"
#define ATTRIBUTION_OUTPUT "ATTRIBUTION"
#define SEMICOLON_OUTPUT "SEMICOLON"
#define COMMA_OUTPUT "COMMA"
#define LEFT_PARENTHESIS_OUTPUT "LPAREN"

```

```

#define RIGHT_PARENTHESIS_OUTPUT "RPAREN"
#define LEFT_BRACKET_OUTPUT "LBRACKETS"
#define RIGHT_BRACKET_OUTPUT "RBRACKETS"
#define LEFT_BRACE_OUTPUT "LBRACES"
#define RIGHT_BRACE_OUTPUT "RBRACES"
#define NUM_OUTPUT "NUMBER"

int main(int argc, char *argv[]) {
    // Verifica se o n mero de argumentos est correto (igual a 2)
    if (argc != 2) {
        printf("Use: ./main <input_file_name>\n");
        return 1;
    }

    // Abrindo o arquivo
    FILE* fptr = fopen(argv[1], "r");

    if (fptr == NULL) {
        printf("Error: File not found\n");
        return 1;
    }

    int state = 0;

    char c = 0;

    while ((c = fgetc(fptr)) != EOF)
    {
        // se for espa o , tab , ou newline , ignora
        if (c == ' ' || c == '\n' || c == '\t') {
            state = 0;
        }

        // verifica se um palavra reservada
        else if (c == 'i' && fgetc(fptr) == 'n' && fgetc(fptr) == 't')
            state = 4;
            printf("%s\n", INT_OUTPUT);
        }
        else if (c == 'f' && fgetc(fptr) == 'o' && fgetc(fptr) == 'r')
            state = 51;
            printf("%s\n", FOR_OUTPUT);
        }
        else if (c == 'f' && fgetc(fptr) == 'l' && fgetc(fptr) == 'o' &
            state = 28;
            printf("%s\n", FLOAT_OUTPUT);
        }
    }

```

```

else if (c == 'v' && fgetc(fp) == 'o' && fgetc(fp) == 'i' &&
state = 18;
printf("%s\n", VOID_OUTPUT);
}
else if (c == 'i' && fgetc(fp) == 'f') {
state = 2;
printf("%s\n", IF_OUTPUT);
}
else if (c == 'e' && fgetc(fp) == 'l' && fgetc(fp) == 's' &&
state = 14;
printf("%s\n", ELSE_OUTPUT);
}
else if (c == 'r' && fgetc(fp) == 'e' && fgetc(fp) == 't' &&
state = 10;
printf("%s\n", RETURN_OUTPUT);
}
else if (c == 'w' && fgetc(fp) == 'h' && fgetc(fp) == 'i' &&
state = 23;
printf("%s\n", WHILE_OUTPUT);
}

// Verifica se um n mero , seja um n mero inteiro ou um n mero
else if (isdigit(c))
{
while (isdigit(c) || c == '.') // se for um n mero (int ou float)
c = fgetc(fp);
printf("%s\n", NUM_OUTPUT);
fseek(fp, -1, SEEK_CUR);
state = 48;
}

// Verifica se um s mbolo
else if (c == EQUAL && fgetc(fp) == EQUAL) {
state = 37;
printf("%s\n", EQUAL_OUTPUT);
}
else if (c == EQUAL) {
state = 37;
printf("%s\n", CONTRIBUTION_OUTPUT);
}
else if (c == PLUS) {
state = 30;
printf("%s\n", PLUS_OUTPUT);
}
else if (c == MINUS) {
state = 29;
}

```

```

        printf("%s\n", MINUS_OUTPUT);
    }
    else if (c == MULTIPLY) {
        state = 31;
        printf("%s\n", MULTIPLY_OUTPUT);
    }
    else if (c == LESS_THAN && fgetc(fp) == EQUAL) {
        state = 34;
        printf("%s\n", LESS_THAN_EQUAL_OUTPUT);
    }
    else if (c == GREATER_THAN && fgetc(fp) == EQUAL) {
        state = 36;
        printf("%s\n", GREATER_THAN_EQUAL_OUTPUT);
    }
    else if (c == DIVIDE) {
        state = 32;
        printf("%s\n", DIVIDE_OUTPUT);
    }
    else if (c == LESS_THAN) {
        state = 33;
        printf("%s\n", LESS_THAN_OUTPUT);
    }
    else if (c == GREATER_THAN) {
        state = 35;
        printf("%s\n", GREATER_THAN_OUTPUT);
    }
    else if (c == SEMICOLON) {
        state = 39;
        printf("%s\n", SEMICOLON_OUTPUT);
    }
    else if (c == COMMA) {
        state = 38;
        printf("%s\n", COMMA_OUTPUT);
    }
    else if (c == LEFT_PARENTHESIS) {
        state = 46;
        printf("%s\n", LEFT_PARENTHESIS_OUTPUT);
    }
    else if (c == RIGHT_PARENTHESIS) {
        state = 45;
        printf("%s\n", RIGHT_PARENTHESIS_OUTPUT);
    }
    else if (c == LEFT_BRACKET) {
        state = 44;
        printf("%s\n", LEFT_BRACKET_OUTPUT);
    }

```

```

else if (c == RIGHT_BRACKET) {
    state = 43;
    printf("%s\n", RIGHT_BRACKET_OUTPUT);
}
else if (c == LEFT_BRACE) {
    state = 42;
    printf("%s\n", LEFT_BRACE_OUTPUT);
}
else if (c == RIGHT_BRACE) {
    state = 41;
    printf("%s\n", RIGHT_BRACE_OUTPUT);
}

// Else      um ID
else
{ // Enquanto o pr ximo caracter      um n mero ou uma letra . S
    while (isdigit(c) || isalpha(c))
        c = fgetc(fptr);

    fseek(fptr, -1, SEEK_CUR);
    printf("ID\n");
    state = 0;
}
}

// fechar arquivo argv[1];
fclose(fptr);

return 0;
}

}

else if (c == 'r' && fgetc(fptr) == 'e' && fgetc(fptr) == 't' &
    state = 10;
    printf("%s\n", RETURN_OUTPUT);
}
else if (c == 'w' && fgetc(fptr) == 'h' && fgetc(fptr) == 'i' &
    state = 23;
    printf("%s\n", WHILE_OUTPUT);
}

// Verifica se      um n mero , seja um n mero inteiro ou um n m
else if (isdigit(c))
{
    while (isdigit(c) || c == '.') // se for um n mero (int ou
        c = fgetc(fptr);
    printf("%s\n", NUM_OUTPUT);
}

```

```

        fseek(fp, -1, SEEK_CUR);
        state = 48;
    }

    // Verifica se um símbolo
    else if (c == EQUAL) {
        state = 37;
        printf("%s\n", EQUAL_OUTPUT);
    }
    else if (c == PLUS) {
        state = 30;
        printf("%s\n", PLUS_OUTPUT);
    }
    else if (c == MINUS) {
        state = 29;
        printf("%s\n", MINUS_OUTPUT);
    }
    else if (c == MULTIPLY) {
        state = 31;
        printf("%s\n", MULTIPLY_OUTPUT);
    }
    else if (c == DIVIDE) {
        state = 32;
        printf("%s\n", DIVIDE_OUTPUT);
    }
    else if (c == LESS_THAN) {
        state = 33;
        printf("%s\n", LESS_THAN_OUTPUT);
    }
    else if (c == LESS_THAN_EQUAL) {
        state = 34;
        printf("%s\n", LESS_THAN_EQUAL_OUTPUT);
    }
    else if (c == GREATER_THAN) {
        state = 35;
        printf("%s\n", GREATER_THAN_OUTPUT);
    }
    else if (c == GREATER_THAN_EQUAL) {
        state = 36;
        printf("%s\n", GREATER_THAN_EQUAL_OUTPUT);
    }
    else if (c == SEMICOLON) {
        state = 39;
        printf("%s\n", SEMICOLON_OUTPUT);
    }
    else if (c == COMMA) {

```



```

        state = 38;
        printf("%s\n", COMMA_OUTPUT);
    }
    else if (c == LEFT_PARENTHESIS) {
        state = 46;
        printf("%s\n", LEFT_PARENTHESIS_OUTPUT);
    }
    else if (c == RIGHT_PARENTHESIS) {
        state = 45;
        printf("%s\n", RIGHT_PARENTHESIS_OUTPUT);
    }
    else if (c == LEFT_BRACKET) {
        state = 44;
        printf("%s\n", LEFT_BRACKET_OUTPUT);
    }
    else if (c == RIGHT_BRACKET) {
        state = 43;
        printf("%s\n", RIGHT_BRACKET_OUTPUT);
    }
    else if (c == LEFT_BRACE) {
        state = 42;
        printf("%s\n", LEFT_BRACE_OUTPUT);
    }
    else if (c == RIGHT_BRACE) {
        state = 41;
        printf("%s\n", RIGHT_BRACE_OUTPUT);
    }
}

// Else      um ID
else
{ // Enquanto o pr ximo caracter      um n mero ou uma letra . S
    while (isdigit(c) || isalpha(c))
        c = fgetc(fptr);

    fseek(fptr, -1, SEEK_CUR);
    printf("ID\n");
    state = 0;
}
}

// fechar arquivo argv[1];
fclose(fptr);

return 0;
}

```

Foram usados vários *defines*, para palavras-chave e símbolos. A razão pela qual

elas foram criadas seria para ter os outputs em definidas já no formato esperado pela especificação do trabalho. Adiante, é feito um while que percorre todo o arquivo, de caracter em caracter. Dentro do while é usada técnicas para verificar se uma condição é verdadeira, todavia, ao usar um if que contém várias condições, como o fgetc(fp), o que acontece é que tem-se uma pesquisa em maior profundidade sobre o arquivo. Dado o exemplo, `c == 'i' fgetc(fp) == 'n' fgetc(fp) == 't'`, o que acontece é, verifica-se o caracter atual do arquivo (c) é um 'i' porém, verifica também, se o caracter seguinte é um 'n' e, após o n, se tem-se um 't'. Isso seria apenas uma forma mais sofisticada e curta de simplesmente aninhar vários ifs, cada um verificando um caracter específico fazendo o deslocamento do ponteiro sempre que necessário. Dessa forma, essa técnica é usada para palavras reservadas e, para símbolos, basta verificar se o caracter atual refere-se a um dado símbolo usando um if com uma condição simples. Por fim, só falta cobrir dois casos, o de ter-se um número ou um id. O id é a exceção, então é tratado por último. Um número é basicamente uma repetição de números, o qual pode ou não conter um ponto, o qual tornaria-o um valor em ponto flutuante. Para isso, bastou verifica com um if se, no caracter atual tem-se um dígito (por meio da função `isdigit()`). Se sim, fica preso em um while que consome todos os próximos números e, para o caso de ser um ponto flutuante, ele também verifica se há um ponto ('.'). Dessa forma, sobrou apenas o ID, todavia o ID é qualquer coisa que não foi tratada, logo basta ter um else para reconhecê-lo. Todavia, e se houverem espaços em branco, quebra de linhas ou tabulações? Por questões óbvias, não deseja-se dizer que essas coisas são IDs, logo um if é colocado no começo do while para apenas consumir esses casos, impedindo que eles recebam uma flag de id. Além disso, dentro de cada if também foi necessário atualizar o valor da variável *state*, a qual cuida da transição entre estados.

4. Testes

Será apresentado um teste para exemplificar o funcionamento do algoritmo. Para executar o algoritmo é necessário executar os seguintes passos:

1º - Compilar:

```
gcc main.c -o main
```

2º - Executar:

```
./main <fileName.format>
```

Teste 01: O arquivo que será lido é:

```
int main(void){  
    return(0);  
}
```

Tem-se então os seguintes tokens: INT = int ID = main LPAREN = (VOID = void RPAREN =) LBRACES = { RETURN = return RBRACES = } NUMBER = 0 RBRACES = } SEMICOLON = ; RBRACES = }

E o output gerado pelo código é:

```
PS D:\Backup\Mega Sync\Code\C Programing\University\LFA> ./main test.c
INT
ID
LPAREN
VOID
RPAREN
LBRACES
RETURN
LPAREN
NUMBER
RPAREN
SEMICOLON
RBRACES
PS D:\Backup\Mega Sync\Code\C Programing\University\LFA> █
```

Figura 2. Máquina de Moore - Simulador JFLAP