UNIVERSIDADE FEDERAL DE SANTA CATARINA

GRADUAÇÃO SISTEMAS DE INFORMAÇÃO

INTRODUÇÃO A COMPILADORES - INE5622

Relatório Parte 0:

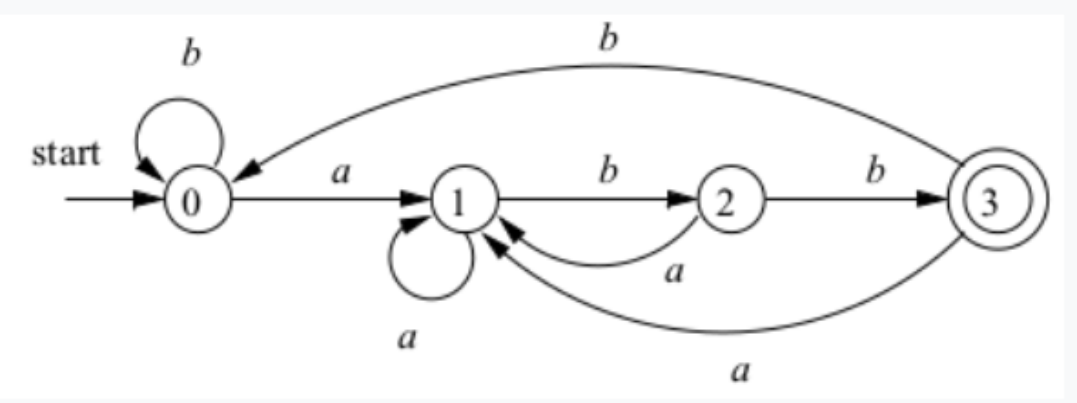
Implementação de AFD Simples

Lucas Broering dos Santos[1]

[1] Graduando em Sistemas de Informação; Contato: lucas.broering@grad.ufsc.br

Introdução

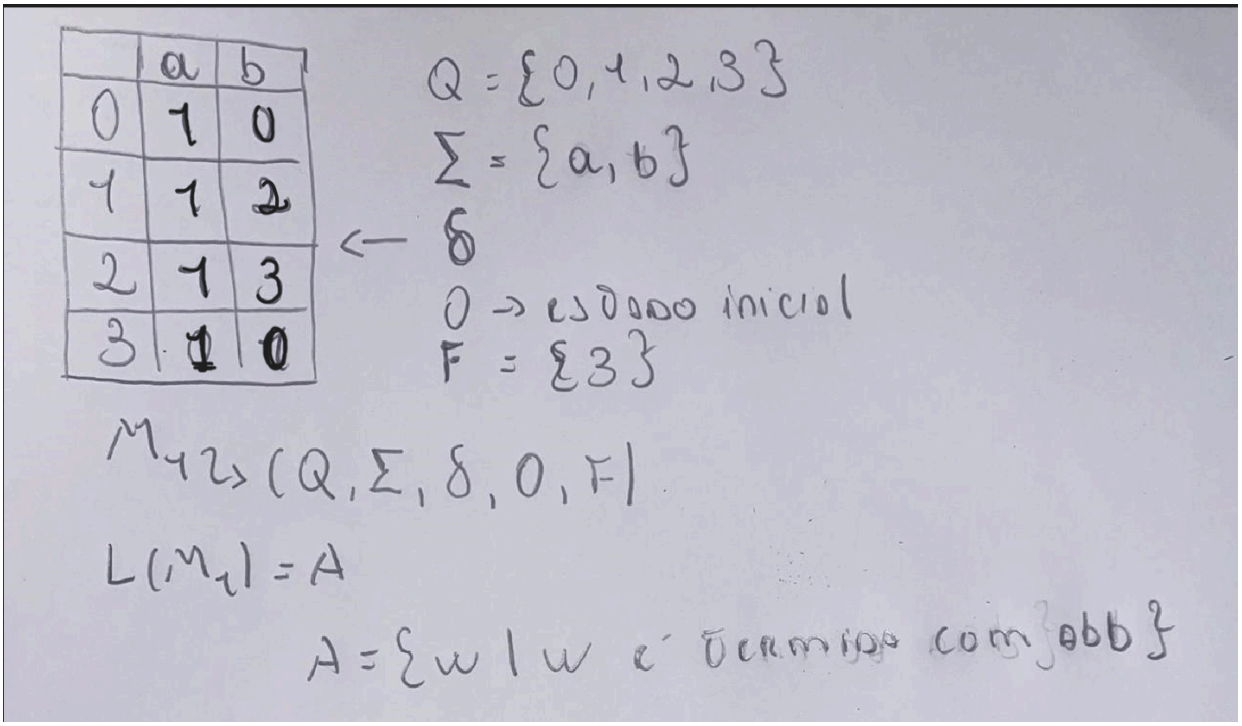
Esse relatório visa exemplificar as escolhas tomadas na implementação do AFD abaixo:



Optei por fazer essa implementação utilizando a linguagem C++, por questão de ser fortemente tipada e ter suporte para classes. Também por conta de eu querer me desafiar, aprendendo mais sobre essa linguagem, já que não a utilizo diariamente no trabalho.

Breve explicação sobre o AFD

O AFD possui a seguinte representação formal e reconhece a seguinte linguagem:



Ele reconhece a linguagem onda é terminada em abb.

Implementação

Como comentei anteriormente, fiz a implementação em C++. Optei por fazer uma classe chamada **Dfa**, onde a mesma possui os 5 atributos de um AFD.

```
class Dfa
{
private:
    // Definição dos atributos do autômato finito determinístico
    std::vector<std::string> states;
    std::vector<std::string> alphabet;
    std::unordered_map<std::string, std::unordered_map<std::string, std::string>> transaction_function;
    std::string start_state;
    std::vector<std::string> accept_states;
```

Os atributos são:

- states é um vector de strings.
- alphabet é um vector de strings.
- transaction_function é um unordered_map, como se fosse um dicionário do python. Eu utilizo um unordered_map “dentro” de outro. O externo tem uma key string e um value map, já o interno possui uma key string e um value string. Essa foi a forma que pensei para “simular” a tabela dos valores da função de transição do AFD.
- start_state é uma string.
- accept_states é um vector de strings.

Obs: Utilizei a estrutura vector, por ser uma espécie de lista onde não precisa definir o tamanho previamente.

Essa classe possui alguns métodos, privados e públicos. Primeiro vou explicar os privados.

```
    // Método para verificar se o input é válido
    bool checkInput(const std::string &input)
    {
        for (int i = 0; i < input.length(); i++)
        {
            std::string symbol(1, input[i]);
            bool valid_symbol = false;

            for (int i = 0; i < alphabet.size(); i++)
            {
                if (symbol == alphabet[i])
                {
                    valid_symbol = true;
                    break;
                }
            }

            if (!valid_symbol)
            {
                return false;
            }
        }

        return true;
    }
```

O método checkInput faz uma checagem no input que o usuário fez, checa se cada símbolo pertence ao alfabeto.

```
// Método para exbir mensagem de erro caso o input contenha símbolos inválidos
void displayAlphabetErrorMessage()
{
    printf("\nInvalid input!!\n");
    printf("Input symbols should be from the following alphabet:\n");
    printf("- ");
    for (int i = 0; i < alphabet.size(); i++)
    {
        printf("%s ", alphabet[i].c_str());
    }
    printf("\n\n");
}
```

O displayAlphabetErrorMessage, faz um print da mensagem de erro no terminal, juntamente com o alfabeto do autômato.

```
public:
    // Construtor da classe
    Dfa(std::vector<std::string> states,
        std::vector<std::string> alphabet,
        std::unordered_map<std::string,
                           std::unordered_map<std::string, std::string>>
            transaction_function,
        std::string start_state, std::vector<std::string> accept_states)
    {
        this->states = states;
        this->alphabet = alphabet;
        this->transaction_function = transaction_function;
        this->start_state = start_state;
        this->accept_states = accept_states;
    }
```

Construtor da classe Dfa.

```
void isAccept(std::string input)
    std::string current_state = start_state;

    // Verifica se a string de input é válida
    bool valid_input = checkInput(input);

    // Caso a string de input seja válida, o autômato é percorrido, caso contrário, uma mensagem de erro é exibida
    if (valid_input)
    {
        // Esse for pega cada símbolo do input e vai "percorrendo" os estados do autômato
        for (int i = 0; i < input.length(); i++)
        {
            std::string symbol(1, input[i]);

            // Atualiza o estado atual do autômato
            current_state = transaction_function[current_state][symbol];

            printf("\n%s -> %s", symbol.c_str(), current_state.c_str());
        }

        // Verifica se o estado atual é um estado de aceitação
        for (int i = 0; i < accept_states.size(); i++)
        {
            if (current_state == accept_states[i])
            {
                printf("\n\nAccepted\n");
            }
            else
            {
                printf("\n\nRejected\n");
            }
        }
    }
    else
    {
        displayAlphabetErrorMessage();
    }
}
```

O método público isAccept, ele que vai “percorrer” o AFD com base no input. Primeiro a variável current_state é declarada e recebe o start_state, depois é feita a validação do input, chamando o checkInput e guardando na variável valid_input, caso o input seja válido o método continua na lógica de passar pelos estados, caso não, uma mensagem é exibida no terminal. Caso válido, o for vai pegar cada símbolo do input e atualizar o estado atual, passando o estado atual e o símbolo (seria a key do map externo) e retornando o próximo estado. O for seguinte pega todos os estados de aceitação e checa se o current_state está em algum deles, se sim o autômato aceita o input, se não rejeita. Eu optei por imprimir no terminal, poderia dar algum tipo de return e tratar depois, mas achei assim mais direto.

```
void displayDfa()
{
    printf("----- States -----\n");
    printf("-");
    for (int i = 0; i < states.size(); i++)
    {
        printf(" %s ", states[i].c_str());
    }
    printf("\n");

    printf("\n----- Alphabet -----\n");
    printf("-");
    for (int i = 0; i < alphabet.size(); i++)
    {
        printf(" %s ", alphabet[i].c_str());
    }
    printf("\n");

    printf("\n----- Transition Function -----\n");
    for (int i = 0; i < states.size(); i++)
    {
        for (int j = 0; j < alphabet.size(); j++)
        {
            printf(" %s -> %s -> %s\n", states[i].c_str(), alphabet[j].c_str(), transaction_function[states[i]][alphabet[j]].c_str());
        }
        printf("\n");
    }

    printf("----- Start State -----\n");
    printf("- %s\n\n", start_state.c_str());

    printf("----- Accept States -----\n");
    printf("-");
    for (int i = 0; i < accept_states.size(); i++)
    {
        printf(" %s", accept_states[i].c_str());
    }
    printf("\n");
```

Esse último método serve apenas para imprimir o autômato.

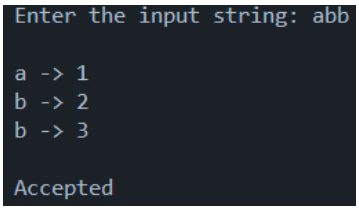
Essa explicação foi do arquivo dfa.h, o arquivo seguinte main.cpp, é onde faço uma instância de um autômato e passo os parâmetros do AFD específico, juntamente com a leitura do input. Para compilar utilize g++ main.cpp -o main no terminal. Para rodar ./main.

```
int main()

{
    // Instanciando um autômato finito determinístico
    std::vector<std::string> states = {"0", "1", "2", "3"};
    std::vector<std::string> alphabet = {"a", "b"};
    std::unordered_map<std::string, std::unordered_map<std::string, std::string>> transaction_function;
    transaction_function["0"]["a"] = "1";
    transaction_function["0"]["b"] = "0";
    transaction_function["1"]["a"] = "1";
    transaction_function["1"]["b"] = "2";
    transaction_function["2"]["a"] = "1";
    transaction_function["2"]["b"] = "3";
    transaction_function["3"]["a"] = "1";
    transaction_function["3"]["b"] = "0";
    std::string start_state = "0";
    std::vector<std::string> accept_states = {"3"};

    Dfa dfa(states, alphabet, transaction_function, start_state, accept_states);

    std::string input;
    std::cout << "Enter the input string: ";
    std::cin >> input;

    // Se descomentar essa linha o autômato será exibido
    // dfa.displayDfa();

    dfa.isAccept(input);

    return 0;
}
```

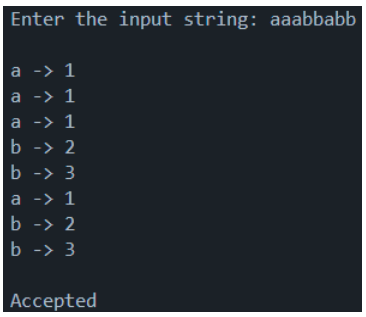
Exemplos de inputs

Strings que o AFD aceita:

- **abb**

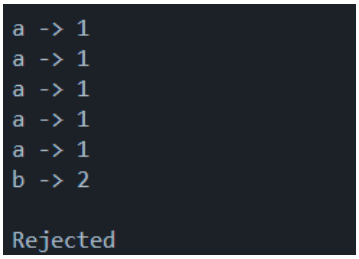


- **aababababbababababababababababababababbbbbbaaabb**
  (não tirei print por ser grande no terminal)
- **aaabbabb**

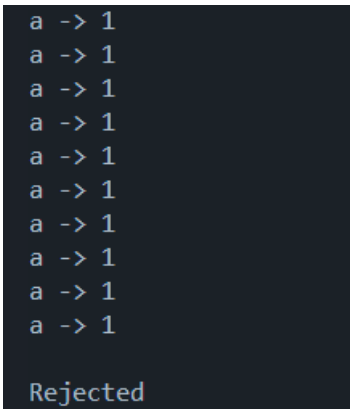


Strings que o AFD rejeita:

- **aaaaab**



- **aaaaaaaaaa**



- **aaaaaaaaaaaaaaaaaaaaabbbbbbbbbababababbbbbaaabbbbbb**
  (não tirei print por ser grande no terminal)