Curso de Sistemas de Informação
Universidade Federal de Viçosa – *Campus Rio Paranaíba*
Disciplina: SIN- 131 Introdução a Teoria da Computação

Simulador de Autômatos

Conversão de AFN para AFD, minimização, simulação de equivalência
e Máquina de Turing

Equipe:
6317 - Eliezer Assunção de Paulo
7570 - Camille Custódio de Paiva

1. Introdução

O objetivo deste projeto é implementar um sistema capaz de receber um Autômato Finito Não Determinístico (AFN), convertê-lo em um Autômato Finito Determinístico (AFD) equivalente, simular a aceitação de palavras e demonstrar a equivalência entre o AFN original e o AFD convertido. Além disso, o projeto também envolve a minimização do AFD resultante, se possível.

2. Requisitos do Projeto

O projeto foi dividido em duas categorias, sendo os requisitos funcionais e não funcionais. Ou seja, os requisitos funcionais especificam o que o sistema deve fazer, suas funções e funcionalidades para que a aplicação atenda às necessidades dos usuários. Deve-se descrever ações específicas em que o sistemas será capaz de realizar. Já os requisitos não funcionais, descrevem como o sistema deve funcionar, abordando aspectos como desempenho, segurança, usabilidade e confiabilidade. A Tabela abaixo, demonstra a descrição geral dos requisitos referente ao simulador de autômatos.

2.1 Requisitos Funcionais

O programa deve receber como entrada um AFN.
Implementar o algoritmo de conversão de AFN para AFD.
Simular a aceitação de palavras pelo AFD convertido.
Demonstrar a equivalência entre o AFN original e o AFD convertido.
Implementar o algoritmo de minimização de AFD.

2.2 Requisitos não funcionais

O sistema deve ser eficiente e responder em tempo razoável.
A interface do usuário deve ser intuitiva e fácil de usar.

3. Estrutura do Projeto

3.1 Entrada do AFN

Um Autômato Finito Não-Determinístico (AFN) é um modelo matemático usado para reconhecer linguagens formais. Ele é uma extensão dos autômatos finitos determinísticos (AFD) e permite que, a partir de um estado, o autômato possa transitar para múltiplos estados possíveis, sem que haja uma regra determinística única para essas transições. O AFN aceita uma entrada se houver pelo menos um caminho de transições que leva a um estado final (de aceitação). Os AFNs são importantes porque oferecem uma maneira mais flexível de modelar sistemas que podem ter múltiplos estados ou transições incertas, embora reconheçam exatamente as mesmas linguagens formais que os AFDs, que são as linguagens regulares. Neste projeto, o programa deve ser capaz de receber como entrada um AFN que é definido por:

- Um conjunto de estados Q.
- Um conjunto finito que representa um alfabeto Σ.
- Um conjunto de transições Δ: $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$.
- Um estado inicial $q_0 \in Q$.
- Um conjunto de estados de aceitação F, $F \subseteq Q$.

3.2 Conversão de AFN para AFD

O objetivo é implementar o algoritmo para converter o AFN em um AFD equivalente. Por exemplo, temos um autômato finito não-determinístico, chamado M, que trabalha com um alfabeto composto pelas letras "a" e "b". O objetivo desse autômato é verificar se a entrada contém uma sequência de letras que começa com 'a' e pode ser seguida por qualquer combinação de 'a's e 'b's, garantindo que a entrada atinja um dos estados finais. A Figura 1 representa o diagrama de estados para M:

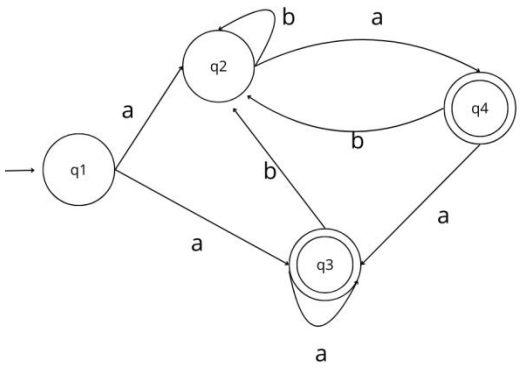


Figura 1: Diagrama de estados para M

Formalmente, o autômato M é definido por um conjunto de estados {q1, q2, q3, q4}, um alfabeto {a, b}, uma função de transição Δ, um estado inicial q1, e dois estados finais {q3,q4}. A função de transição Δ é descrita por uma tabela de transição de estados, que indica como o autômato se move de um estado para outro com base na entrada que está processando. Conforme representado na tabela a seguir:

	a	b
q1	{q2,q3}	-
q2	{q4}	{q2}
q3	{q3}	{q2}
q4	{q3}	{q2}

Dado o AFN definido acima, o programa realiza a conversão para um Autômato Finito Determinístico (AFD) equivalente. A implementação da função que realiza essa conversão é feita da seguinte forma:

```
while (!fila.isEmpty()) {
    Set<Estado> estadosAtuais = fila.poll();
    Estado estadoAfdAtual = conjuntosMap.get(estadosAtuais);

    for (Character simbolo : afn.alfabeto().caracteres()) {
        Set<Estado> estadosDestino = new HashSet<>();

        for (Estado estado : estadosAtuais) {
            for (Transicao transicao : afn.conjuntoTransicoes()) {
                if (transicao.estadoAtual().equals(estado) && transicao.simboloEntrada().equals(simbolo)) {
                    estadosDestino.add(transicao.estadoDestino());
                }
            }
        }

        if (!estadosDestino.isEmpty()) {
            Estado estadoAfdDestino = conjuntosMap.get(estadosDestino);
            if (estadoAfdDestino == null) {
                estadoAfdDestino = new Estado(conjuntoParaNome(estadosDestino));
                conjuntosMap.put(estadosDestino, estadoAfdDestino);
                fila.add(estadosDestino);
            }
            novasTransicoes.add(new Transicao(estadoAfdAtual, simbolo, estadoAfdDestino));
        }
    }
}
```

O código apresentado implementa a conversão de um Autômato Finito Não-determinístico (AFN) para um Autômato Finito Determinístico (AFD). Inicialmente, enquanto a fila de estados a serem processados não está vazia, o conjunto de estados atuais é retirado da fila, e o estado correspondente no AFD é recuperado de um mapa que associa conjuntos de estados do AFN a estados únicos no AFD. Em seguida, para cada símbolo do alfabeto, o código busca determinar o estado de destino do AFD ao considerar todas as possíveis transições no AFN a partir dos estados atuais.

Para cada estado dentro do conjunto de estados atuais, o código verifica as transições possíveis no AFN. Se uma transição correspondente for encontrada, o estado de destino é adicionado a um novo conjunto de estados. Se esse conjunto não estiver vazio, o código verifica se ele já possui um estado correspondente no AFD. Caso contrário, um novo estado é criado no AFD, mapeado e adicionado à fila para processamento posterior. Finalmente, uma nova transição no AFD é criada, conectando o estado atual ao estado de destino, utilizando o símbolo de entrada. Esse processo é repetido até que todos os estados e transições do AFD sejam definidos. Por fim, o código identifica os estados finais do AFD verificando se algum estado nos conjuntos mapeados contém um estado final do AFN. Se encontrar, adiciona esse estado à lista de novos estados finais do AFD, garantindo que a conversão preserve os estados finais corretos.

Logo, após aplicada a conversão, espera-se obter como saída o AFD contido na figura 2 apresentada abaixo:

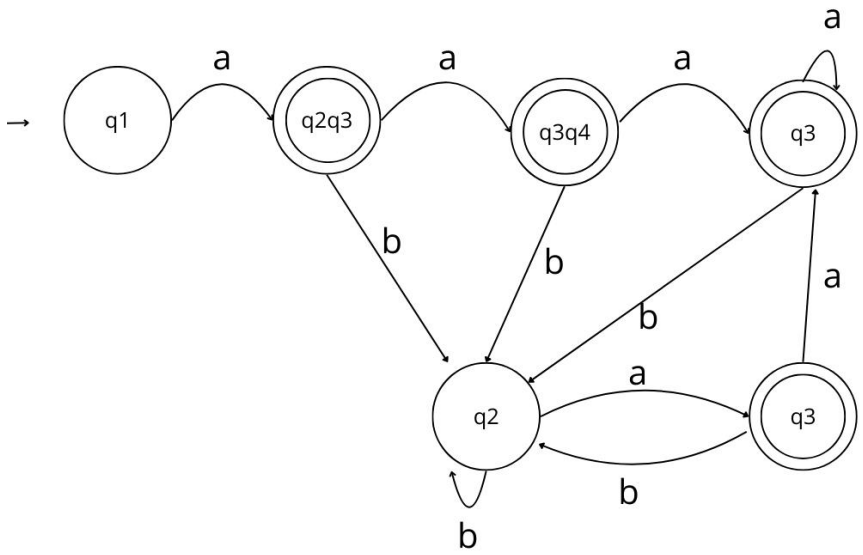


Figura 2: Conversão do AFN em AFD

3.3 Simulação de Aceitação de Palavras

Após a conversão, o programa deve ser capaz de receber uma palavra (sequência de símbolos do alfabeto) e simular a execução do AFD com essa palavra. Em resumo, enquanto o AFN pode explorar múltiplos caminhos simultaneamente e aceita uma palavra se qualquer caminho a leva a um estado de aceitação, o AFD segue um único caminho e aceita a palavra apenas se esse caminho específico terminar em um estado de aceitação. A Figura 2, mostra o código para realizar a simulação de aceitação de palavras para o Automato Finito-Não-Determinístico (AFD):

```java
public static boolean aceitaPalavra(AFD afd, String palavra) {  2 usages  eliezer
    Estado estadoAtual = afd.estadoInicial();

    for (char simbolo : palavra.toCharArray()) {
        estadoAtual = proximoEstado(estadoAtual, simbolo, afd.conjuntoTransicoes());
        if (estadoAtual == null) {
            return false;
        }
    }
    return afd.estadosFinais().contains(estadoAtual);
}
```

O método aceitaPalavra começa no estado inicial e processa a palavra símbolo por símbolo. A cada símbolo, o método proximoEstado é chamado para determinar o próximo estado com base nas transições disponíveis. Se não houver uma transição válida para um símbolo, o método retorna false, indicando que a palavra não é aceita. Se a palavra for completamente processada e o autômato terminar em um estado de aceitação, a palavra é considerada aceita.

Já a implementação da função para os AFN é mais complexa. Sendo realizada de forma recursiva. O método aceitaPalavra inicia a simulação e chama o método verificaEstado, que percorre todas as possíveis transições a partir do estado inicial. Para cada símbolo da palavra, o método verifica todas as transições possíveis para o estado atual e chama recursivamente o método para o próximo símbolo e o estado de destino. Se, ao final da palavra, o autômato estiver em um estado de aceitação, a palavra é aceita. Caso contrário, o método continua explorando outros possíveis caminhos. Conforme representado na Figura 3:

```java
private static boolean verificaEstado(Estado estadoAtual, String palavra, List<Transicao> transicoes, List<Estado> estadosFinais) {
    if (palavra.isEmpty()) {
        return estadosFinais.contains(estadoAtual);
    }

    char simbolo = palavra.charAt(0);
    String restoPalavra = palavra.substring( beginIndex: 1);

    for (Transicao transicao : transicoes) {
        if (transicao.estadoAtual().equals(estadoAtual) && transicao.simboloEntrada().equals(simbolo)) {
            if (verificaEstado(transicao.estadoDestino(), restoPalavra, transicoes, estadosFinais)) {
                return true;
            }
        }
    }

    return false;
}
```

3.4 Demonstração de Equivalência

Após a conversão do AFN para o AFD, o programa deve demonstrar que o AFN original e o AFD convertido são equivalentes, verificando se ambos aceitam a mesma linguagem.

3.5 Minimização de AFDs

Após converter um AFN para um AFD, o próximo passo é implementar o algoritmo de minimização de estados para reduzir o número de estados do AFD resultante, se possível. O exemplo apresentado acima em sua forma simplificada esta apresentado na Figura 3:

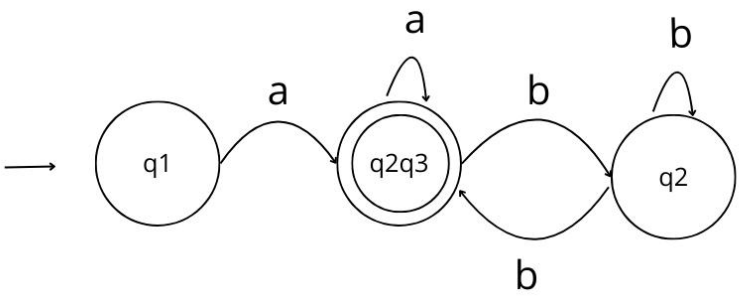


Figura 3: AFD em sua forma minimizada

O algoritmo foi implementado da seguinte maneira: inicialmente, realiza-se a separação entre estados finais e não finais. Em seguida, são realizadas comparações entre os estados finais e entre os estados não finais, resultando em uma lista de estados de destino possíveis.

Após isso, todas as transições do autômato são percorridas, analisando cada uma das comparações realizadas. As transições que levam a destinos que não estão na lista de destinos válidos são descartadas, enquanto aquelas que resultam em destinos válidos são adicionadas a uma nova lista. Esta nova lista será analisada para verificar a possibilidade de minimização do autômato.

A minimização é viável quando estados diferentes alcançam um mesmo destino, permitindo a combinação desses estados em um único estado. Por fim, o resultado é retornado ao *Frontend*, possibilitando que o usuário visualize as informações de forma clara.

4. Máquina de Turing

A Máquina de Turing é um modelo teórico de computação que desempenha um papel fundamental na teoria da computabilidade e na compreensão dos limites do que pode ser computado. Este relatório tem como objetivo apresentar o desenvolvimento e a simulação de duas Máquinas de Turing, desenvolvidas para resolver dois problemas específicos: a verificação de se um número binário é par e o incremento de um número binário.Cada máquina desenvolvida segue uma especificação formal que inclui a definição dos estados, alfabeto da fita, regras de transição, e condições de aceitação e rejeição.

4.1 Máquina de Turing para verificação de número par em binário

Consiste em determinar se um número binário, dado como entrada na fita da Máquina de Turing, é par. Um número binário é par se seu último dígito (bit menos significativo) for '0'. A Máquina deve ler o número binário da esquerda para a direita e, ao identificar o último dígito, decidir se o número é par ou ímpar. Neste exemplo, a máquina foi definida por:

Alfabeto da Fita (Γ): {0, 1, “ ”*}*
Alfabeto de Entrada (Σ): {0, 1}
Estados (E): {estadoInicial, estadoDeAceitacao, estadoDeRejeicao}
Estado Inicial (i): estadoInical
Estados de Aceitação (F): {estadoDeAceitacao}

Função de Transição (δ):
δ(estadoInicial, '0') = (estadoDeAceitacao, '0', N) (Aceita se o último dígito for '0')
δ(estadoInicial, '1') = (estadoDeRejeicao, '1', N) (Rejeita se o último dígito for '1')
δ(estadoInicial, “ ”*) = (*estadoDeRejeicao*, "*, N) (Rejeita se a fita estiver vazia)

A Máquina de Turing inicia no estadoInicial e posiciona a cabeça de leitura na primeira posição da fita. Ela avança até encontrar o delimitador de fim da fita (_). A máquina então verifica o último dígito do número binário. Se o último dígito for ‘0', a máquina muda para o estadoDeACeitacao e aceita a entrada, determinando que o número é par. Caso contrário, se o último dígito for ‘1', a máquina muda para o estadoDeRejeicao, rejeitando a entrada. Por exemplo, ao ler como entrada o número 1010, são seguidos os seguintes passos:

1. Cabeça posicionada no início.
2. Máquina lê o dígito 1 e avança.
3. Máquina lê 0 e avança.
4. Máquina lê 1 e avança.
5. Máquina lê 0 e detecta fim da fita.

6. Máquina aceita, pois o último dígito é '0'.

Resultado: Sim, o número foi aceito pela máquina. Como mostra a Figura a seguir:

4.2 Máquina de Turing para incremento de número binário

Este problema consiste em incrementar um número binário em 1. O incremento de um número binário é realizado alterando o bit menos significativo de ‘0' para ‘1' ou, se for ‘1', mudando para ‘0' e propagando o “vai-um” para os bits à esquerda.Para esta máquina temos a seguinte definição:

Alfabeto da Fita (Γ): {0, 1, _}
Alfabeto de Entrada (Σ): {0, 1}
Estados (E): {estadoInicial, vaiUm, estadoDeAceitacao}
Estado Inicial (i): estadoInicial
Estados de Aceitação (F): {estadoDeAceitacao}

Função de Transição (δ):
δ(vaiUm, '0') = (estadoDeAceitacao, '1', N) (Muda '0' para '1' e aceita)
δ(vaiUm, '1') = (vaiUm, '0', L) (Muda '1' para '0' e propaga vaiUm à esquerda)
δ(vaiUm, '1') = (vaiUm, '0', L) (Continua propagando o vaiUm)
δ(vaiUm, '0') = (estadoDeAceitacao, '1', N) (Resolve vaiUm e aceita)
δ(vaiUm, '_') = (estadoDeAceitacao, '1', N) (Se o vaiUm alcançar o início, incrementa)

Seu funcionamento começa no estadoInicial e move a cabeça de leitura até o final do número binário. Se o bit menos significativo for ‘0', ele é alterado para ‘1' e a

máquina aceita a palavra. Se for ‘1', a máquina muda para o estado vaiUm, transforma o ‘1' em ‘0', e move a cabeça para a esquerda para propagar o vaiUm. Se o vaiUm encontrar um ‘0', ele é alterado para ‘1', e a máquina aceita. Se o vaiUm propagar até o início da fita, um ‘1' é adicionado à frente do número, completando o incremento. Por exemplo, ao receber como entrada o número 101:

1. Máquina lê '1' e muda para '0', propagando carry.
2. Máquina lê '0' e muda para '1'.
3. Máquina aceita, resultando em 110.

Resultado: 110. Conforme apresentado na figura a seguir:

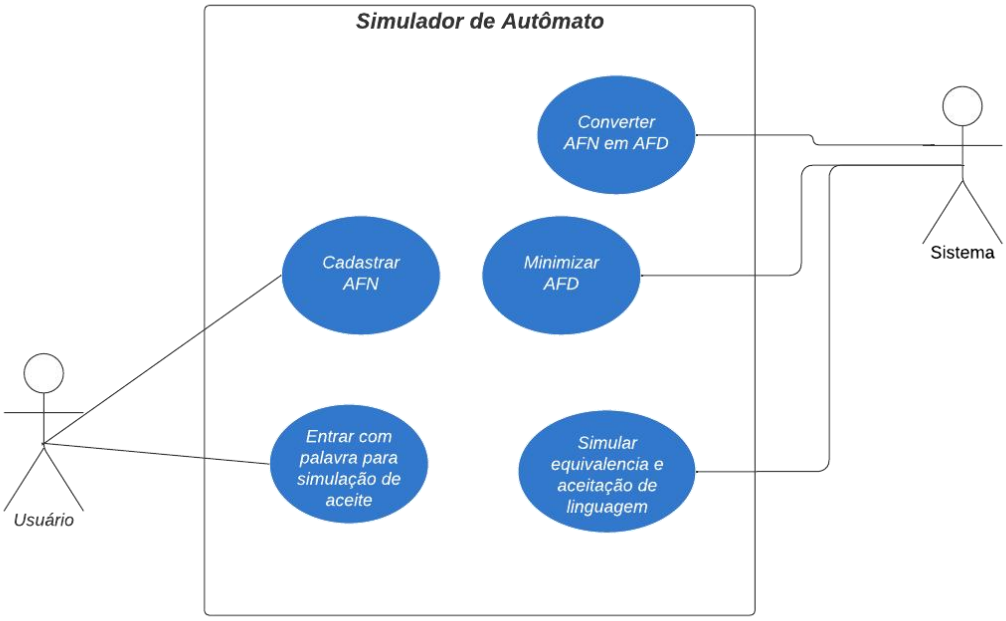
5. Organização do Requisitos

4.1 Caso de Uso

Nome	Atores	Descrição
Cadastrar AFN	Usuário	O Usuário deve preencher os campos referentes ao AFN em que se deseja testar. Constando o conjunto de estados, alfabeto, transição, estados inicial e final.
Converter AFN em AFD	Sistema	Com o AFN criado pelo usuário o sistema deve converter em um AFD equivalente e retornar a resposta ao usuário
Simular aceitação de palavras AFD resultante	Usuário	O usuário deve entrar com uma palavra em que deseja testar para que o sistema retorne se ela é aceita ou rejeitada
Simular equivalência entre AFN e AFD	Sistema	Após o usuário entrar com o AFN e ele ser convertido em um AFD equivalente, dado uma determinada palavra de entrada, os autômatos devem verificar se a linguagem é aceita ou rejeitada e o resultado deve ser o mesmo em ambos para que sejam equivalentes
Minimizar AFD	Sistema	Tendo o AFD resultante da conversão, se for possível reduzir o número de estados o sistema deve retornar ele minimizado

6. Diagrama UML

5.1 Diagrama de caso de uso

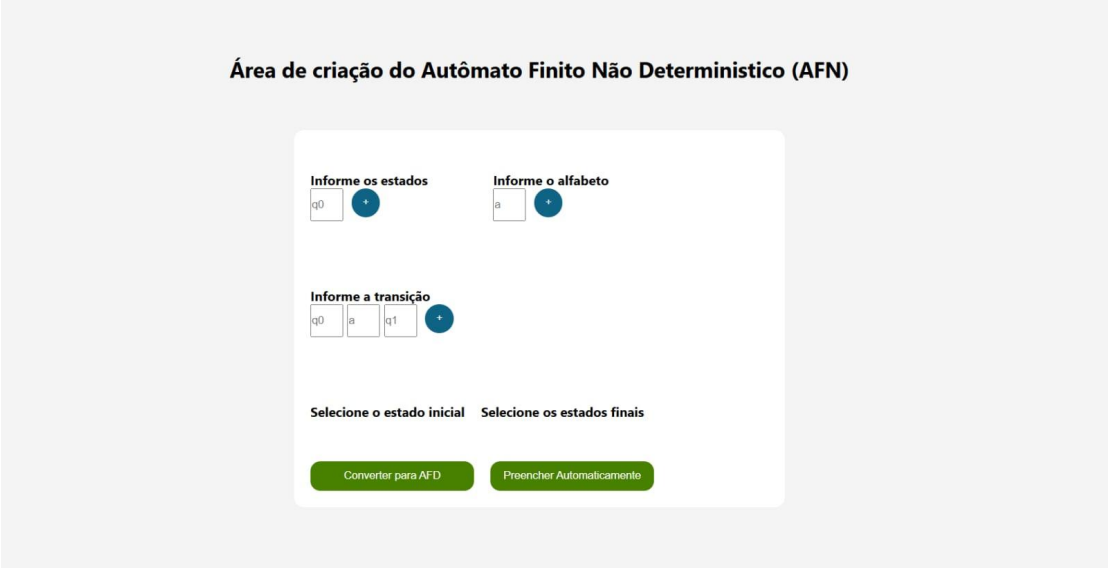


7. Tecnologias Utilizadas

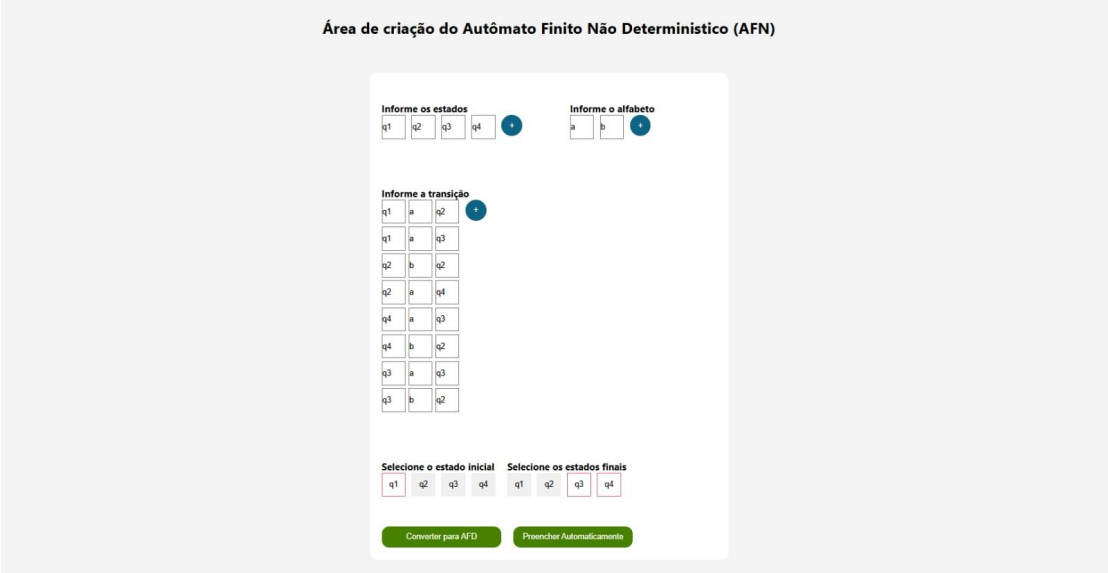
Para este projeto, a linguagem de programação utilizada no backend foi Java, enquanto para a apresentação da interface frontend utilizou-se o framework React com TypeScript. O desenvolvimento foi realizado no ambiente IntelliJ IDEA. A criação dos diagramas foi essencialmente realizada na plataforma Lucidchart, e as imagens dos autômatos foram criadas na plataforma do Canva.

8. Projeto da Interface

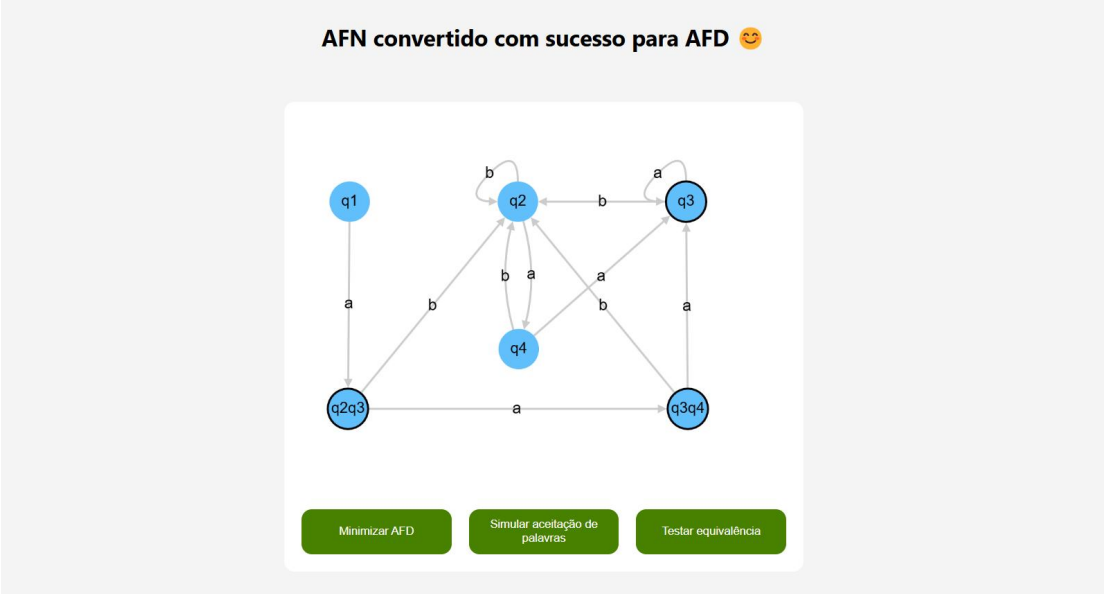
Interface Inicial do Simulador de Autômatos



Área de Criação do AFN (Mesma área para AFD)

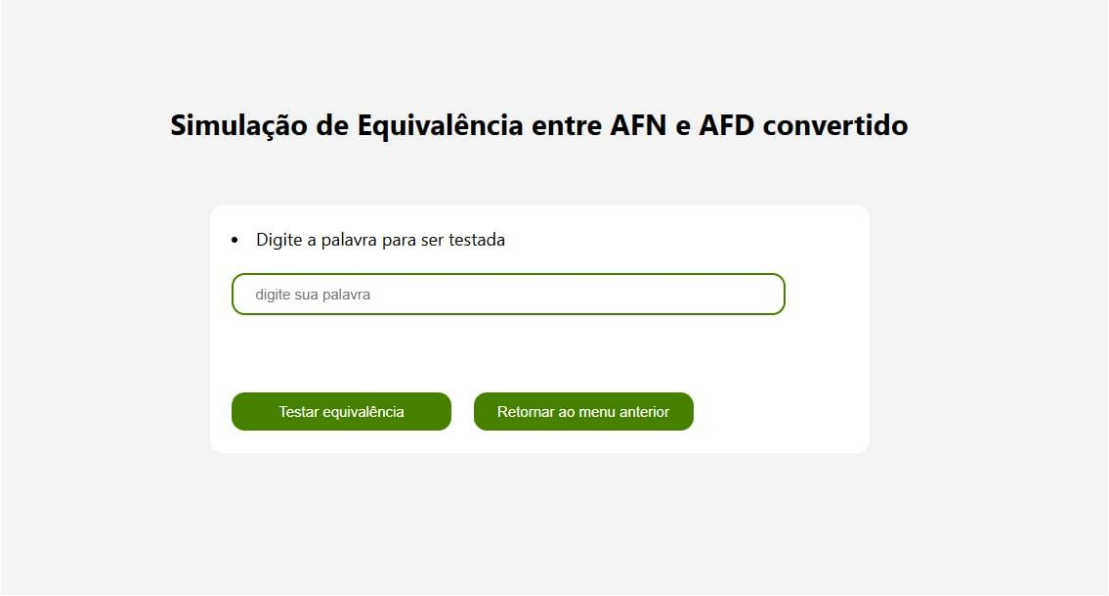


Simulando o preenchimento do autômato



Conversão realizada de AFN para AFD e opções liberadas para o novo AFD

Área de simulação de aceitação de palavras



Área de simulação de equivalência (mesma palavra deve ser aceita por ambos os autômatos)

Área inicial das Máquinas de Turing