UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROJETO PRÁTICO – FINAL
RELATÓRIO

LINGUAGENS FORMAIS E AUTÔMATOS
PROF. DR. CELSO OLIVETE JÚNIOR

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE
FEVEREIRO - 2017

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PROJETO PRÁTICO – FINAL
RELATÓRIO

Projeto prático da disciplina de Linguagens Formais e Autômatos, lecionada pelo docente Dr. Celso Olivete Júnior, no curso Bacharelado em Ciência da Computação – Departamento de Matemática e Computação da Faculdade de Ciências e Tecnologia (FCT Unesp – Presidente Prudente).

PRESIDENTE PRUDENTE
FEVEREIRO – 2017

SUMÁRIO

1 INTRODUÇÃO

Neste projeto da disciplina de Linguagens Formais e Autômatos o objetivo da parte 1 é desenvolver uma ferramenta na qual o usuário da mesma consiga trabalhar e simular Expressões Regulares (ER), Autômatos Finitos Determinísticos (AFD) e Não-Determinísticos (AFND) e Gramaticas Regulares (GR).

O objetivo da parte 2 consiste em realizar simulações com Autômatos Finitos com Saída, mais especificamente as chamadas Máquinas de Moore e de Mealy, além disso a ferramenta deve prover conversões entre estruturas equivalentes de definição de linguagens, sendo essas conversões de Gramatica para Autômato Finito, Autômato Finito Para Gramatica, Expressão Regular para Autômato Finito e Autômato Finito para Expressão Regular, como último objetivo da parte 2 deste trabalho foi realizado um mecanismo de leitura e gravação de arquivos XML que representam um Autômato Finito.

A ferramenta que realizará a simulação das ER's, AFD's, AFND's e GR's foi implementada utilizando a linguagem de programação Java, com a utilização do JavaFX para construção da interface gráfica, necessitando assim que o usuário utilize a versão 8 do Java. Como IDE foi utilizado o NetBeans, contando também com um sistema de controle de versão Git para versionamento do projeto.

Este relatório está divido como segue: na seção 2 é apresentado os conceitos de Expressão Regular bem como o funcionamento da ferramenta para este propósito, a seção 3 descreve Autômatos Finitos e apresenta o simulador de AFD e AFND, a seção 4 discorre sobre Gramaticas Regulares mostrando o funcionamento da ferramenta para a simulação de GR's, já a seção 5 exemplifica as funcionalidades de conversões, na seção 6 é retratado os Autômatos Finitos com Saída, por fim a seção 7 relata a representação de um autômato finito em um arquivo XML.

2 EXPRESSÃO REGULAR

Quando citamos Linguagens Regulares temos as chamadas Expressões Regulares (ER) que é uma notação para representar um determinado padrão de strings. As Expressões Regulares denotam Linguagens Regulares e são muito utilizadas em sistemas de processamento de strings, por exemplos em aplicações que utilizam pesquisas em textos.

Na ferramenta desenvolvida neste projeto temos um simulador de Expressão Regular, seu funcionamento é simples basta inserir a regra, ou seja, a Expressão Regular e em seguida informa a string, palavra de entrada, que a ferramenta irá informa se a palavra de entrada é válida de acordo com a Expressão Regular informada anteriormente. Abaixo temos um instantâneo da aplicação mostrando a tela na qual essa verificação, simulação, é utilizada.

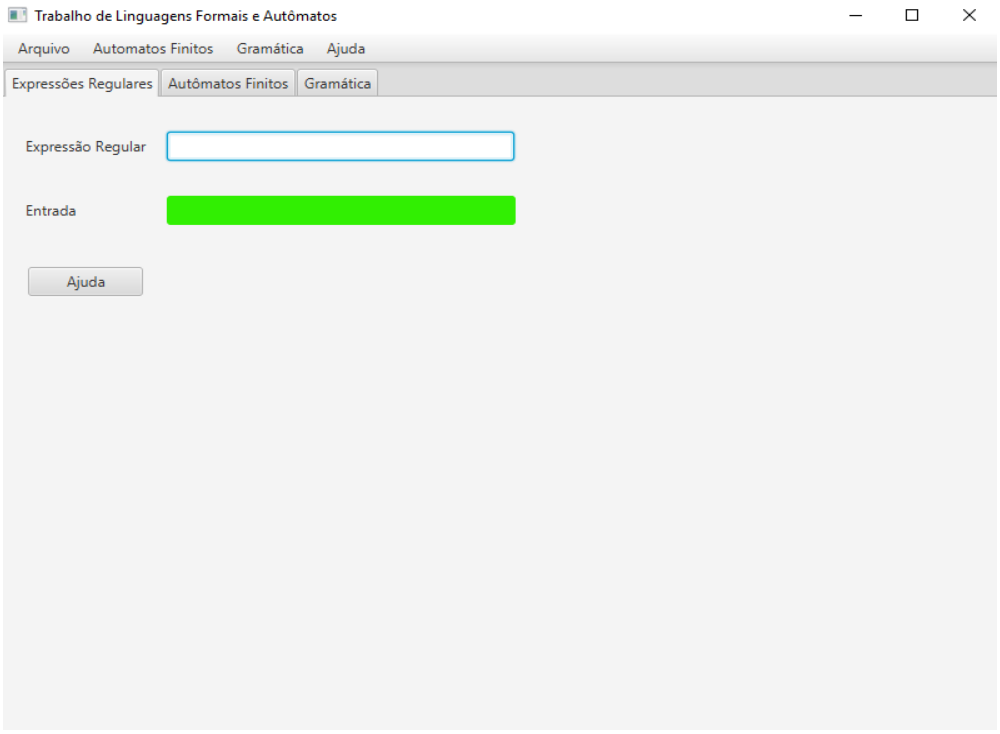


Figura 1 - Instantâneo da ferramenta na tela de Expressões Regulares.

Para exemplificar como a ferramenta atua vamos definir uma Expressão Regular (ER) e em seguida fazer um teste com duas palavras de entrada, uma delas pertence a Expressão Regular (ER) enquanto a outra não é válida pela mesma. Exemplo:

Quadro 1 - Exemplo de possíveis entradas para a expressão.

Expressão Regular (ER)	aa(b|c)*d
Palavra de Entrada 1	aabccbcd
Palavra de Entrada 2	aabcbb

A Expressão Regular (ER) definida como exemplo permite as strings pertencentes a ela comecem obrigatoriamente com a ocorrência de **aa**, seguida de qualquer quantidade da letra **b** ou da letra **c**, inclusive nenhuma, terminando obrigatoriamente com a letra **d**.

Observe no instantâneo de exemplo 1 que a Expressão Regular foi definida e que a palavra de entrada 1 (**aabccbcd**) está de acordo com a ER informada, portando está string é válida perante a ER estabelecida, com isso a ferramenta destaca a palavra de entrada 1 na cor verde.

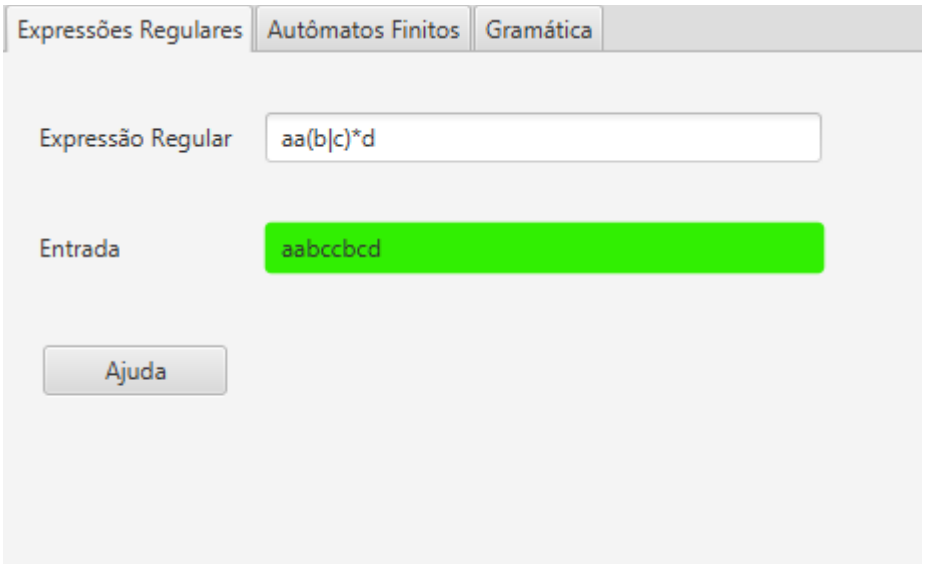


Figura 2 - Exemplo de entrada aceita pela expressão.

Agora observe no instantâneo de exemplo 2 que a Expressão Regular foi definida e que a palavra de entrada 2 está de incorreta com relação a ER informada, pois a ER define como regra que toda string pertencente a ela deve obrigatoriamente terminar com a letra **d** e no caso a palavra de entrada 2 (**aabcbb**) não termina com a letra **d**, portando está string é invalida perante a ER estabelecida, com isso a ferramenta destaca a palavra de entrada 2 na cor vermelha.

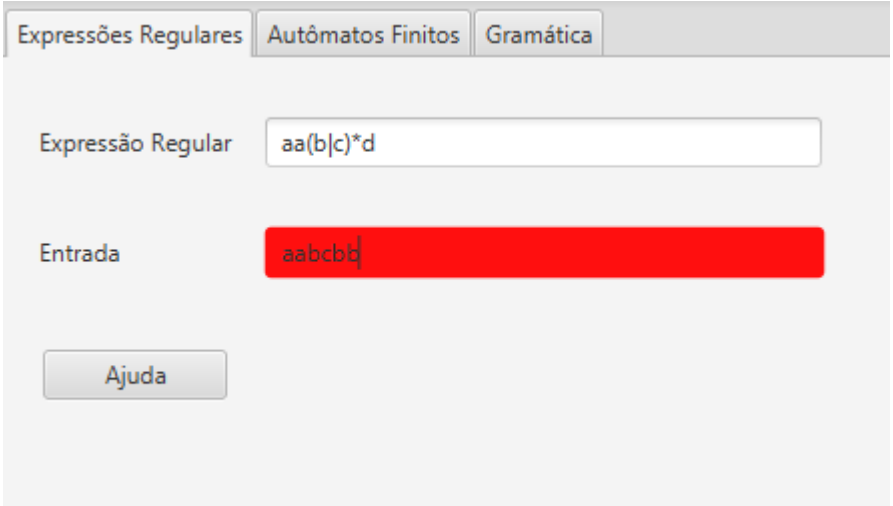


Figura 3 - Exemplo de entrada inválida.

Além disso a aplicação também informa caso a Expressão Regular esteja sintaticamente escrita de forma errada, ou seja, se os caracteres inseridos na ER não são válidos o usuário é informado com uma mensagem e um alerta de cor vermelha no campo de entrada da Expressão Regular, observe o instantâneo abaixo com a exemplificação de um erro de sintaxe na ER.

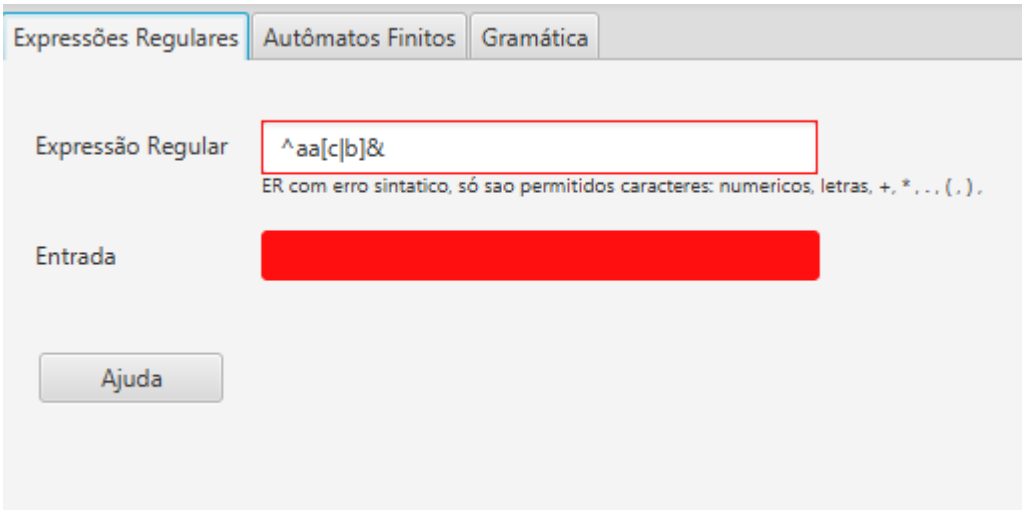


Figura 4 - Exemplo de inserção de símbolos inválidos.

Ainda para facilitar a utilização da ferramenta, existe um botão de ajuda onde é explicado como deve ser utilizada a ferramenta com relação a ER, quais são os caracteres validos, exemplos e outras informações que podem ajudar o usuário, observe o instantâneo abaixo:

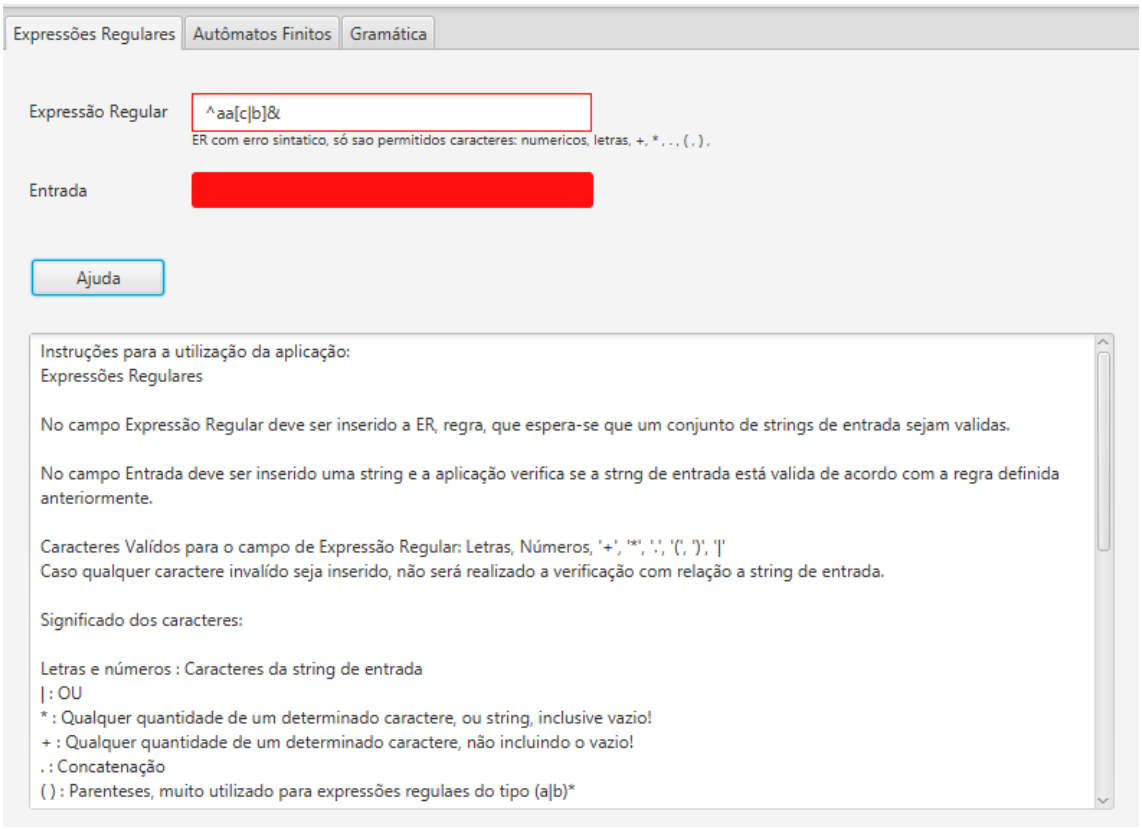


Figura 5 - Exibição do painel de ajuda da expressão regular.

Os caracteres válidos para o campo de Expressão Regular são somente: Letras, Números, '+', '*', '.', '(', ')', '|'. Observe abaixo o significado desses caracteres abaixo:

Quadro 2 - Listagem de caracteres.

Caractere	Significado
Letras	Letras do alfabeto de a até z
Números	De 0 até 9
|	Ou
*	Qualquer quantidade de um determinado caractere, inclusive vazio!
+	Qualquer quantidade de um determinado caractere, não incluindo o vazio!
.	Concatenação
( )	Parênteses, muito utilizado para expressões regulares do tipo (a|b)*

2.1 Implementação do simulador de Expressão Regular

A implementação do simulador de Expressão Regular foi desenvolvida de maneira bem simples utilizando os recursos oferecidos pela linguagem de programa Java.

Observamos abaixo o método que realiza a verificação, basicamente é utilizado o próprio *regex* nativo do Java como sugerido pelo professor, onde no caso são coletados dois dados de entrada, um dado chamado de regra (Expressão Regular) e outro de entrada (palavra de entrada). A regra é compilada por *Pattern.compile()* do Java e posteriormente é verificado se a palavra de entrada respeita a regra definida que faz essa ação é o *Pattern.matches()* do Java retornando um *boolean* com a resposta da verificação, o restante é a parte da interface gráfica, utilizando o JavaFX que realiza.

```
public void patternRegexERcomEntrada(){

    String regra = this.regraTextField.getText();
    String entrada = this.entradaTextField.getText();

    try{
        Pattern.compile(regra);
        boolean verif = Pattern.matches(regra, entrada);
        this.trocarCorTextField(verif, entradaTextField);
    }catch(PatternSyntaxException e){}

}
```

Figura 6 - Trecho da codificação da expressão regular.

3 AUTÔMATOS FINITOS

Outro formalismo importante dentro das Linguagens Regulares temos os Autômatos Finitos, os quais, por meio de estados e suas respectivas transições podem verificar a aceitação ou não de um dado padrão fornecido. Estes podem ser classificados em determinísticos e não-determinísticos, os quais verificam a possibilidade ou não estarem em mais de um estado a partir de uma mesma transição, assim como, a presença de transições vazias.

Para a realização do desenho gráfico foi utilizada a biblioteca gráfica nativa da plataforma JavaFX, que por sua vez é nativa da linguagem Java. Esta biblioteca fornece classes para desenho de arcos, círculos, reta, polígonos, entre outras figuras geométricas. Em seguida, foram desenvolvidas as classes responsáveis pela movimentação dos estados. Por fim, foram implementados os métodos que controlavam o autômato e também para o processamento deste.

Na ferramenta desenvolvida, temos um simulador de autômatos finitos, o qual permite a inserção e remoção de estados assim como de suas respectivas transições, inserção de rótulos para cada estado e para o autômato completo, possibilitando a movimentação dos estados no painel de visualização.

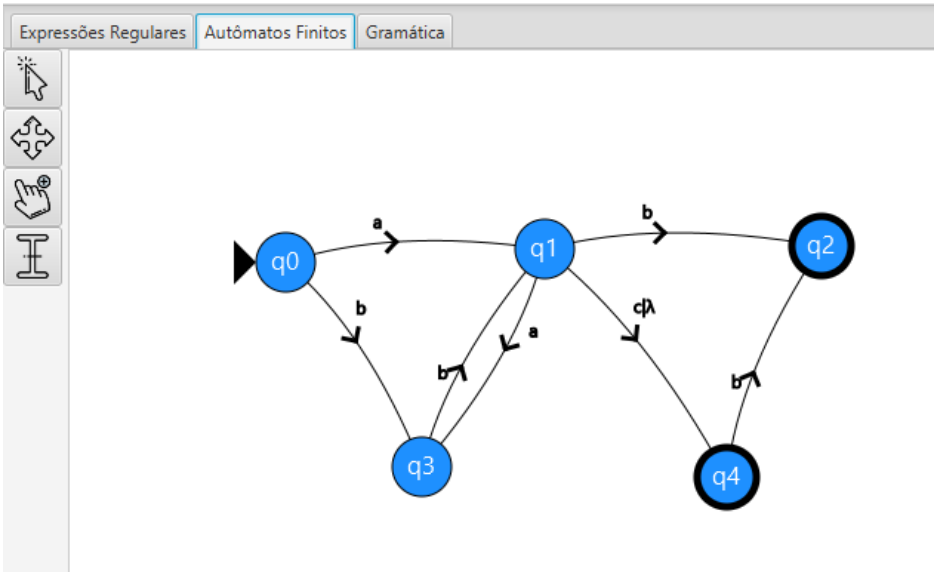


Figura 7 - Exibição de autômato finito.

A interface gráfica é dividida em duas partes: painel de desenho do autômato e o painel lateral.

No painel lateral, estão dispostos quatro botões para a realização de eventos sobre o respectivo autômato, a seguir estão descritas as suas funcionalidades:

- Botão Ponteiro: criação de arestas entre os vértices, além do clique com o botão direito sobre vértices e transições, nas quais é possível remover transições ou indicar que um vértice é inicial e/ou final.
- Botão Mover: responsável pela movimentação de deslocamento de todos os vértices pertencentes ao autômato.
- Botão Inserção de Vértice: responsável pela inserção de novos estados no autômato.
- Botão Texto: responsável pela criação de rótulos de texto no painel de desenho do autômato.

3.1 Atividades no simulador

Abaixo estarão descritos o passo a passo das atividades:

- Inserção de estados: para inserir estados deve-se clicar no botão Inserção de Vértice e em seguida, em qualquer lugar do painel de desenho.
- Remoção de estados: deve-se clicar no botão Ponteiro e em seguida clicar com o botão direito e depois em Remover.
- Marcar como inicial: deve-se clicar no botão Ponteiro e em seguida clicar com o botão direito e depois em Inicial.
- Marcar como final: deve-se clicar no botão Ponteiro e em seguida clicar com o botão direito e depois em Final.
- Inserir legenda: deve-se clicar no botão Ponteiro e em seguida clicar com o botão direito e depois em Inserir Legenda.
- Inserir transição: deve-se clicar no botão Ponteiro e em seguida no vértice inicial e arrastar até o vértice final.
- Remover transição: deve-se clicar no botão Ponteiro e em seguida com o botão direito clicar sobre a transição, o qual listará todas as transições disponíveis e depois basta clicar sobre a opção da transição a ser removida.
- Adicionar rótulo geral: deve clicar no botão Texto e depois em alguma posição do painel de desenho e depois inserir o texto no campo disponível e clicar Enter.

- Movimentação geral: para movimentar qualquer elemento basta clicar no botão e arrastar o elemento desejado para a posição que quiser.

3.2 Implementação do simulador de Autômatos Finitos

A implementação do simulador de autômatos finitos utiliza como base um algoritmo bastante famoso denominado Busca em profundidade, a ideia do algoritmo de foi preservada e adaptada para resolução do processamento de um autômato finito, pois o mesmo tem uma estrutura semelhante a um grafo direcionado.

Tem-se duas funções uma denominada *processamentoAutomato* e outra chamada *verificaRegra*, a primeira função é responsável por realizar a primeira chamada na segunda função, *verificaRegra*, que é uma função de recursão, essa primeira chamada passa o estado inicial do autômato para começar seu processamento afim de encontrar algum caminho para validar a palavra de entrada.

A função *verificaRegra* é recursiva e tem como o objetivo percorrer os estados do autômato de forma com que a palavra de entrada seja validada, encontrando um desses caminhos, ou seja chegando ao estado final, após percorrer a palavra de entrada inteira, a função sai das pilhas recursivas já com o armazenamento desse caminho e com a resposta se a palavra de entrada é válida em um *boolean*. Vale destacar em a cada chamada recursiva parte da palavra de entrada, caractere a caractere é comparado com o do autômato, assim se as comparações não baterem não é gerado caminho de solução e a palavra de entrada é marcada como invalida. A codificação está localizada dentro da pasta *src* na classe *GerenciadorAutomatos*.

4 GRAMÁTICA REGULAR

Por fim, o último formalismo presente nas Linguagens Regulares, temos as Gramáticas Regulares, a qual é definida através da seguinte quádrupla G = (T, NT, P, S), em que T é o conjunto de símbolos terminais, NT é o conjunto de símbolos não-terminais, P consiste no conjunto de predicados que definem as regras as quais restringem a gramática, e por fim, S é definido com o símbolo inicial da gramática.

Tais gramáticas regulares podem ser classificadas nas seguintes categorias:

- GLD: o lado direito da regra da matriz contém n símbolos terminais seguido de um não-terminal.
- GLE: o lado direito da regra contém um símbolo não terminal seguido de n símbolos terminais.
- GLUD: o lado direito da regra contém somente um símbolo terminal seguido de um único símbolo terminal.
- GLUE: o lado direito da regra contém somente um símbolo não-terminal seguido de um único símbolo terminal.

Na ferramenta implementada, foram implementadas as categorias GLUD e GLUE.

A Figura 8, exibe a tabela de entrada a gramática regular, na qual os símbolos não-terminais estão à esquerda da seta, enquanto à direita da regra estão localizados os símbolos terminais, por fim, abaixo da tabela de entrada estão listados de forma categorizada cada um destes símbolos, em terminais e não-terminais.

Figura 8 - Detalhamento de uma gramática regular.

Como observação importante, deve-se salientar que para a entrada de dados na tabela, deve-se realizar o duplo clique na célula da tabela que deseja inserir os dados, após a digitação, para a confirmação deve pressionar a tecla Enter, qualquer outra ação na confirmação perderá o novo valor de entrada. Também foi feita a validação da entrada das regras para gramáticas lineares à esquerda e à direita.

Após a inserção dos dados, é possível testar para uma única entrada ou para múltiplas entradas.

Na Figura 9, podemos visualizar o exemplo de uma única entrada, no qual o resultado é apresentado em uma área de texto indicando se a cadeia de caracteres de entrada foi aprovada ou rejeitada, em caso de aceitação, também é exibido o caminho entre as regras da gramática.

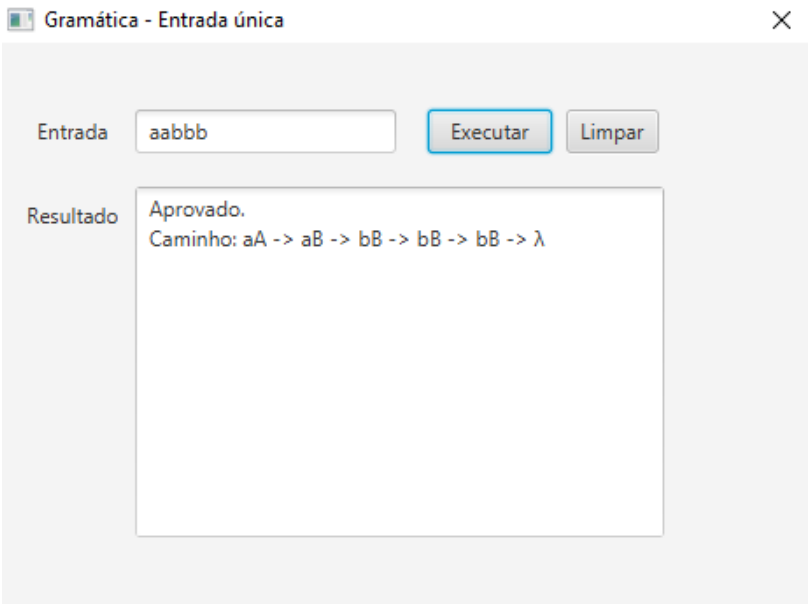


Figura 9 - Resultado da verificação de uma única de entrada para o autômato finito.

Para o caso de múltiplas entradas, o usuário entrará com as várias entradas através de um campo, as quais são adicionadas a tabela, viabilizando uma lista completa das entradas. Em seguida, o usuário deve clicar no botão Executar, o qual processará todas as cadeias de entrada e para cada uma indicará se esta foi aprovada ou rejeitada.

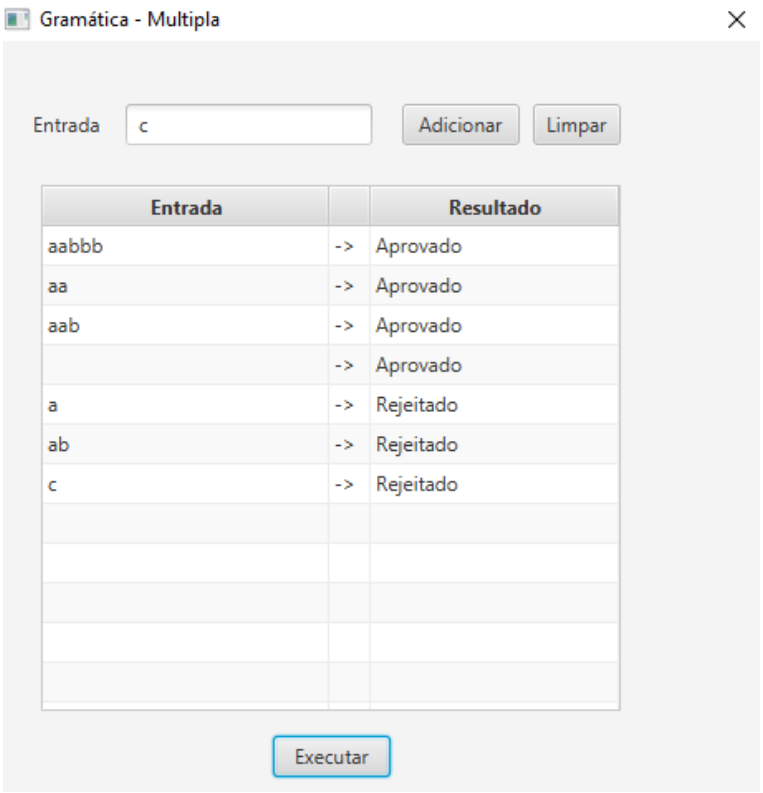


Figura 10 - Resultado da verificação de múltiplas entradas para o autômato finito.

4.1 Implementação do simulador de Gramática Regular

Para a implementação do simulador de Grámatica Regular foram utilizados princípios de recursão, onde tem-se duas funções, sendo elas: *processamentoGramatica* e *verificaRegra.* A função de processamento apenas realiza a primeira chamada na função de verificação de regra, essa primeira chamada passa o símbolo inicial da gramatica como parâmetro.

A função *verificaRegra* temos dois passo importantes um deles realiza uma verificação trabalhando apenas com a gramatica unitária linear a direita e outra apenas com a gramatica unitária linear a esquerda, ao iniciar a função já sabemos qual é o tipo da gramatica que foi inserida, pois na inserção já é identificado o tipo dessa gramatica.

Seja uma GLUD ou uma GLUE o algoritmo tem o mesmo comportamento, existe apenas uma diferença com relação a ordem da leitura caractere a caractere da palavra de entrada, pois a GLUD começa lendo o caractere na posição zero da palavra de entrada e quando realiza uma chamada recursiva incrementa essa posição em mais um, já na GLUE a palavra de entrada é lida da direita para a esquerda, logo pega a última posição, sendo assim na chamada recursiva para caminha por essa String é decrementado uma posição.

Com a situação descrita anteriormente, considerando que a gramatica foi escrita de forma correta pelo usuário, a cada chamada, começando pelo símbolo inicial, vamos percorrendo as regras da gramatica tentando validar a palavra de entrada.

5 CONVERSÕES

É fato que duas estruturas diferentes podem reconhecer a mesma linguagem, assim pode-se gerar uma gramatica que tenha um autômato finito que reconheça a mesma linguagem que ela e vice-versa, nesta seção vamos abordar o conceito de equivalência entre reconhecedores de linguagem focando na conversão entre reconhecedores sendo essas conversões: Gramatica para Autômato Finito, Autômato Finito para Gramatica, Expressão Regular para Autômato Finito e Autômato Finito para Expressão Regular.

Para ter acesso a funcionalidade de conversões na ferramenta deve-se clicar no menu **Conversão** e selecionar a conversão desejada.

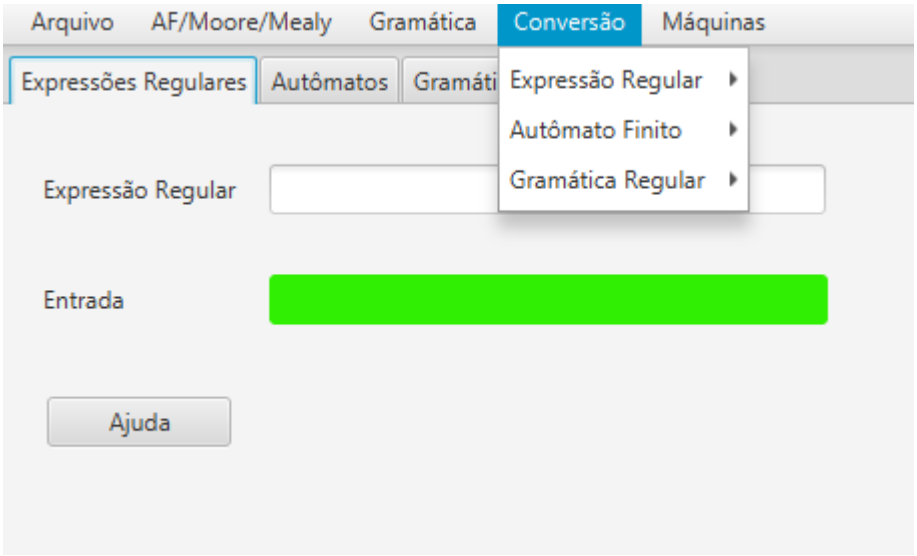


Figura 11 - Menu Conversão

5.1. Gramatica e Autômato

Dada uma gramatica linear à direita é possível construir baseado nesta gramatica um autômato finito que consiga reconhecer a mesma linguagem, assim o processo contrário também ocorre, ou seja, dado um autômato finito é possível construir com base

nele uma gramatica que reconheça a mesma linguagem, dizemos assim que gramaticas e autômatos finitos são equivalentes.

Para realizar a conversão de uma Gramática para Autômato vá para a guia **Gramática** insira uma gramática e depois vá no menu **Conversão -> Gramática Regular -> Autômato Finito**.

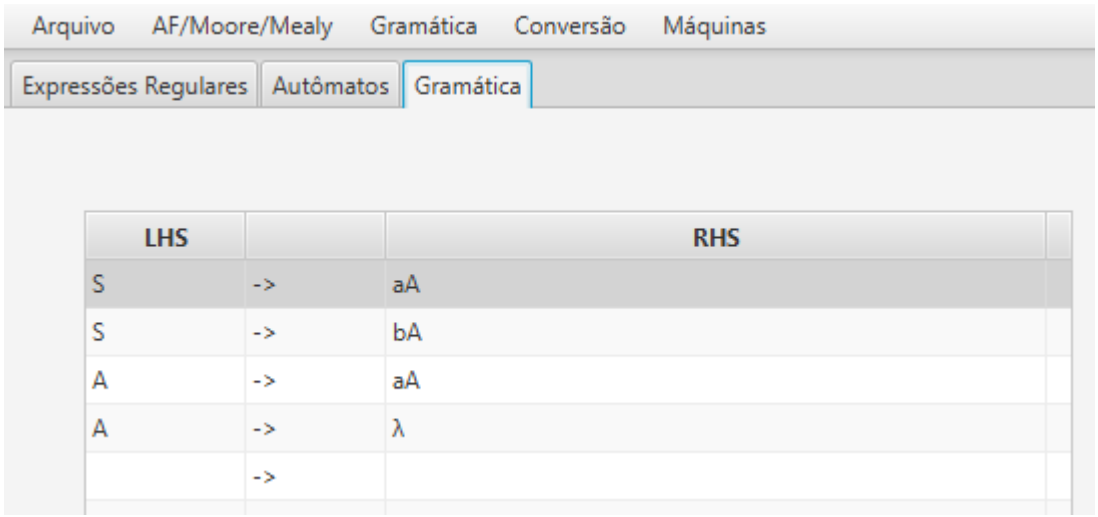


Figura 12 - Inserindo Gramática para Conversão

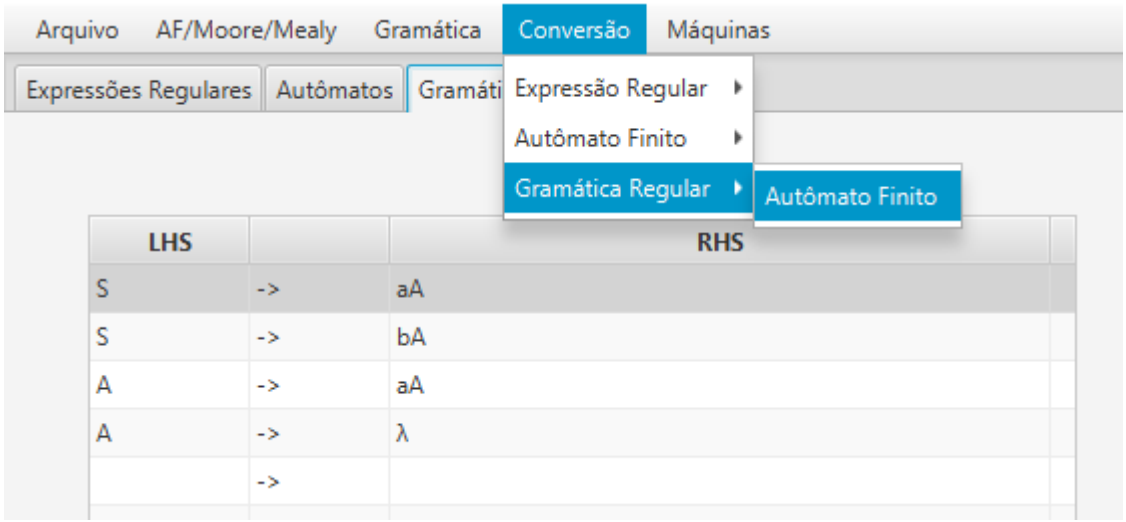


Figura 13 - Aplicando conversão GR para AF

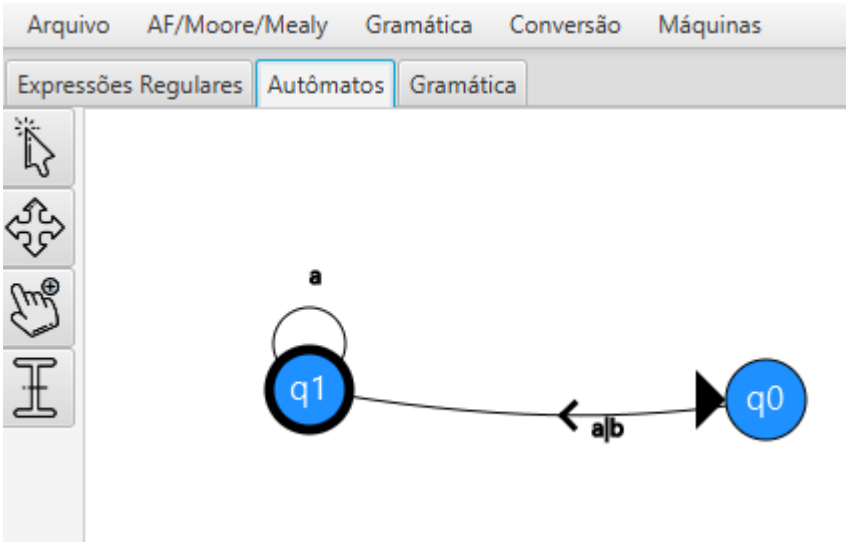


Figura 14 - Resultado da conversão de GR para AF

O processo é o mesmo para realizar a conversão de um Autômato para Gramática vá para a guia **Autômatos** insira um autômato finito e depois vá no menu **Conversão -> Autômato Finito -> Gramática Regular**.

5.1.1 Implementação Gramatica para Autômato

A implementação avalia cada regra da gramática individualmente, em que são extraídos o símbolo de aceitação da transição e o estado alvo. No autômato é adicionado um estado final, em que todas as transições que possuírem somente um símbolo terminal ou forem vazias serão direcionados ao estado adicional.

5.1.2 Implementação Autômato para Gramatica

A implementação após armazenar o respectivo autômato finito em uma estrutura de dados, atribui para cada estado uma respectiva letra, e cada transição de cada estado é transformada em uma regra composta pelo símbolo de aceitação da transição e a letra correspondente ao estado alvo. Todo estado final é acrescido de uma transição vazia.

5.2. Expressão Regular e Autômato

Toda linguagem que é definida por uma Expressão Regular também pode ser definida por um Autômato Finito, a reciproca também é válida, assim também podemos

dizer que Expressões Regulares e Autômatos Finitos são equivalentes e podem ser convertidos um no outro.

Para realizar a conversão de uma Autômato Finito para Expressão Regular vá para a guia **Autômatos** insira um autômato finito e depois vá no menu **Conversão** -> **Autômato Finito** -> **Expressão Regular**. Observe os instantâneos abaixo:

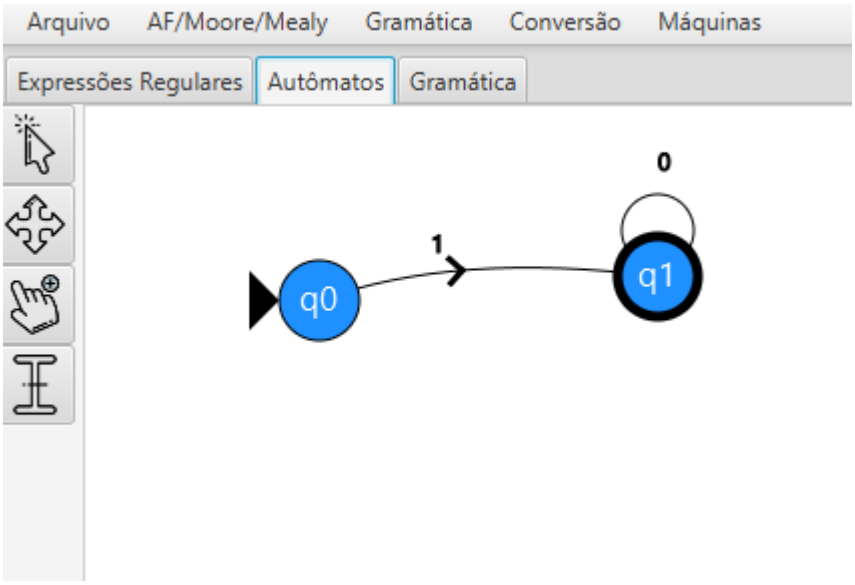


Figura 15 - Inserindo AF para conversão

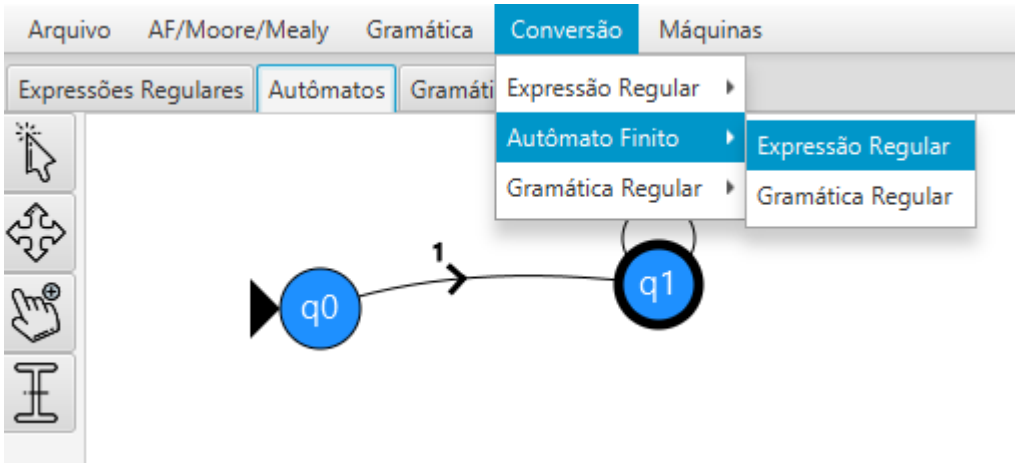


Figura 16 - Convertendo AF para ER

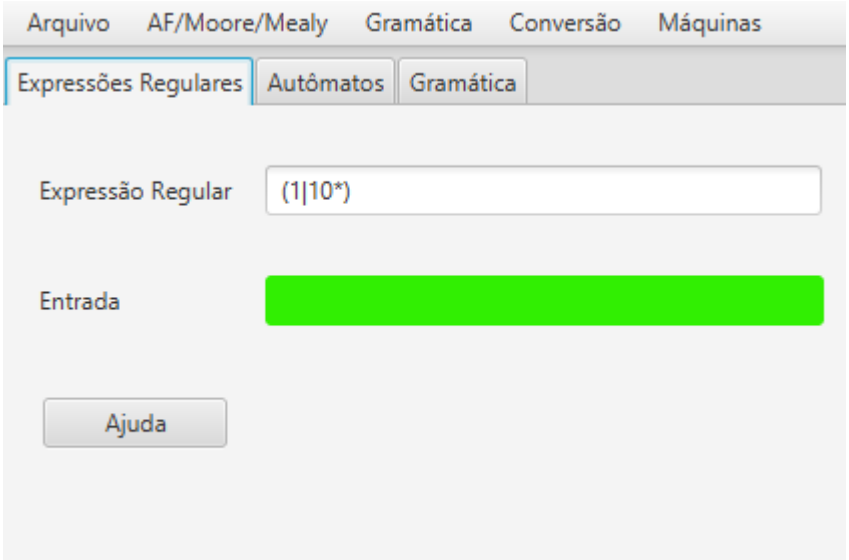


Figura 17 - Resultado da conversão de AF para ER

O processo é o mesmo para realizar a conversão de uma Expressão Regular para Autômato vá para a guia **Expressão Regular** insira uma Expressão Regular e depois vá para menu **Conversão -> Expressão Regular -> Autômato Finito**. Observação, para conversão apenas letras podem estar presentes na Expressão Regular, não podendo ter uma expressão contendo números.

5.2.1. Implementação Expressão Regular para Autômato

A implementação baseia-se no algoritmo de conversão de Thompson e juntamente em uma solução iterativa apoiada pelo uso de estruturas de dados de pilhas em cada abertura e fechamento de parênteses presentes na expressão de forma a compor sub-expressões. Esta implementação suporta a presença de fechamentos abertos e positivos.

5.2.2. Implementação Autômato para Expressão Regular

A implementação inicialmente verifica o número de estados finais do autômato finito, caso este seja maior do que um, são criadas transições vazias de forma a existir somente um estado final, em seguida, para cada estado não inicial Ei, são criadas transições duplas que envolvem todas as transições que chegam ao estado Ei e partem dele, cada par (transição que chega, transição que sai) gera uma nova transição. Quando

todos os pares forem esgotados, somente existirá um que do estado inicial e chega ao único final, representando toda a expressão regular.

6 AUTOMATOS FINITOS COM SAIDA

Autômatos Finitos pode apresentar o conceito de saída, ou seja, ao ler uma string de entrada é gerada uma saída para esta string, os autômatos finitos com saída não aceitam ou rejeitam uma entrada, mas sim gera uma saída para a entrada dada. Como características desses autômatos podemos citar a não existência de um estado final e também que o autômato deve ser determinístico, dois exemplos desses autômatos são as máquinas de Moore e Mealy que são discutidas a seguir.

6.1. Máquina de Moore

A máquina de Moore é um tipo de autômato finito com saída onde em cada estado do autômato produz uma saída, ou seja, toda vez que uma transição levar a um estado este produzira uma saída.

Para expressar uma máquina de Moore na ferramenta primeiramente selecione **Máquinas -> Máquina Moore**, após isso clique na guia **Autômatos**, o instantâneo abaixo exemplifica esta ação:

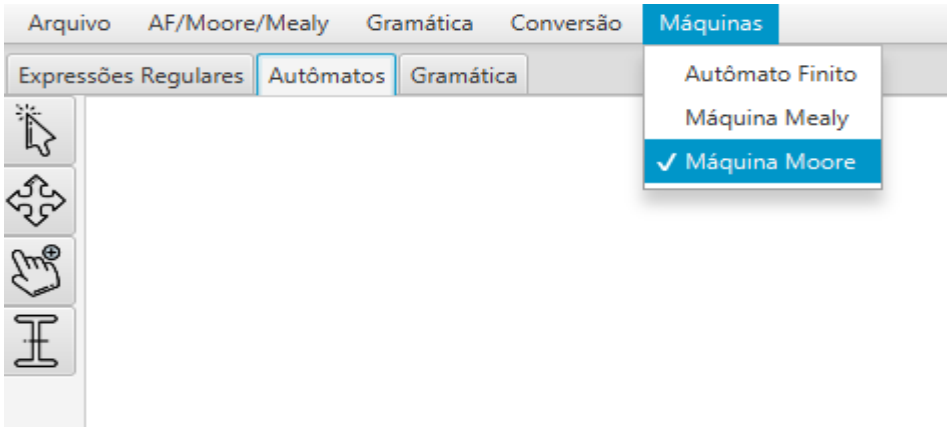


Figura 18 - Inserir Máquina de Moore

Após isso, basta desenhar a máquina de Moore na guia Autômatos e simular obtendo os seus resultados, abaixo temos um exemplo de uma máquina de Moore construída utilizando a ferramenta, este autômato realiza de divisão por 2 em números

binários, ou seja, dado uma entrada binaria uma saída é resultante com os dígitos binários aplicados em uma operação de divisão por 2.

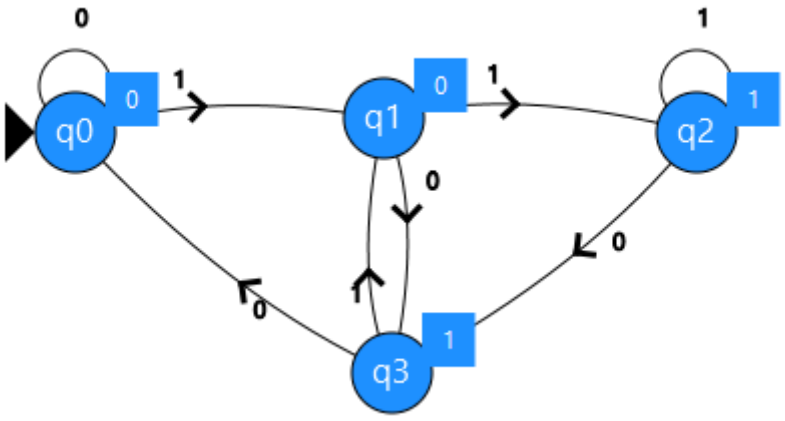


Figura 19 - Máquina de Moore construída utilizando a ferramenta

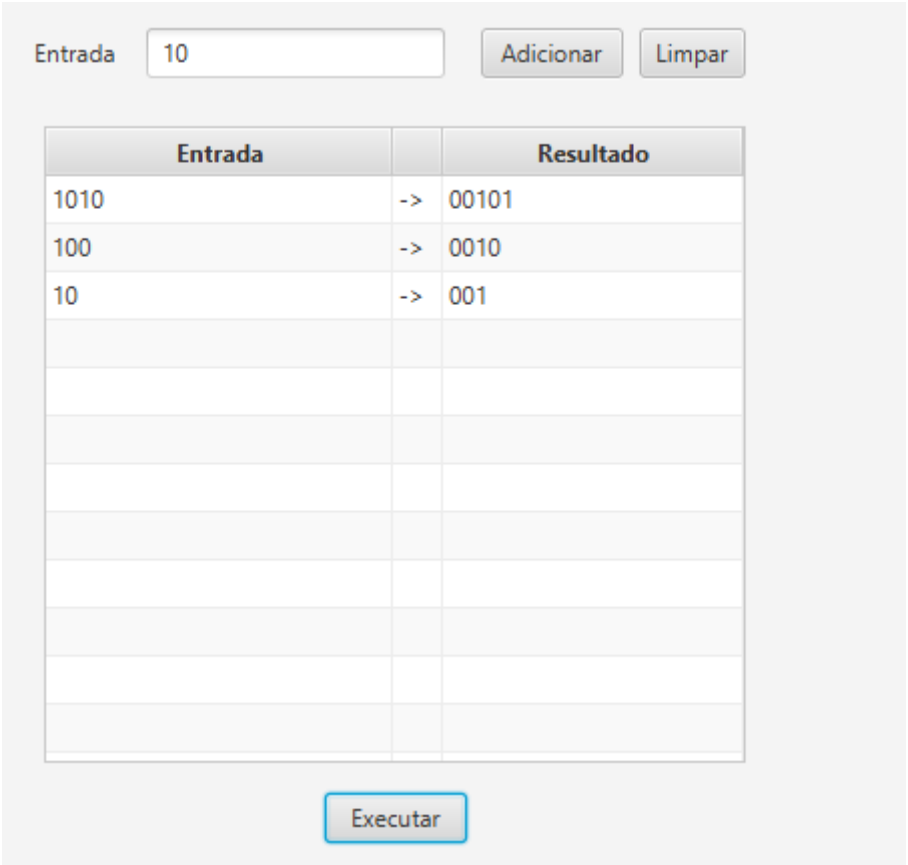


Figura 20 - Simulação da Máquina de Moore

6.2. Máquina de Mealy

A máquina de Mealy é um tipo de autômato finito com saída onde em cada transição do autômato deve-se ter uma saída, ou seja, ao ler um caractere da string de entrada é dado uma saída.

Para expressar uma máquina de Mealy na ferramenta primeiramente selecione **Máquinas -> Máquina Mealy**, após isso clique na guia **Autômatos**, o instantâneo abaixo exemplifica esta ação:

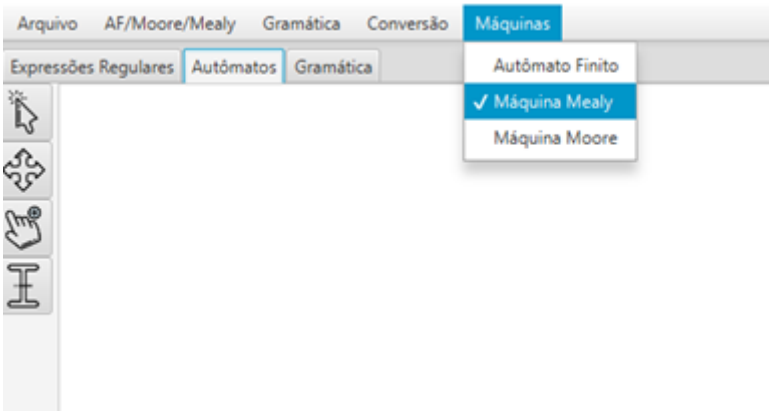


Figura 21 - Inserir Máquina de Mealy

Após isso, basta desenhar a máquina de Mealy na guia Autômatos e simular obtendo os seus resultados, abaixo temos um exemplo de uma máquina de Mealy construída utilizando a ferramenta, este autômato realiza a operação not em números binários, ou seja, dado uma entrada binaria uma saída é resultante com os dígitos binários aplicados em uma operação not.

Figura 22 - Máquina de Mealy construída utilizando a ferramenta

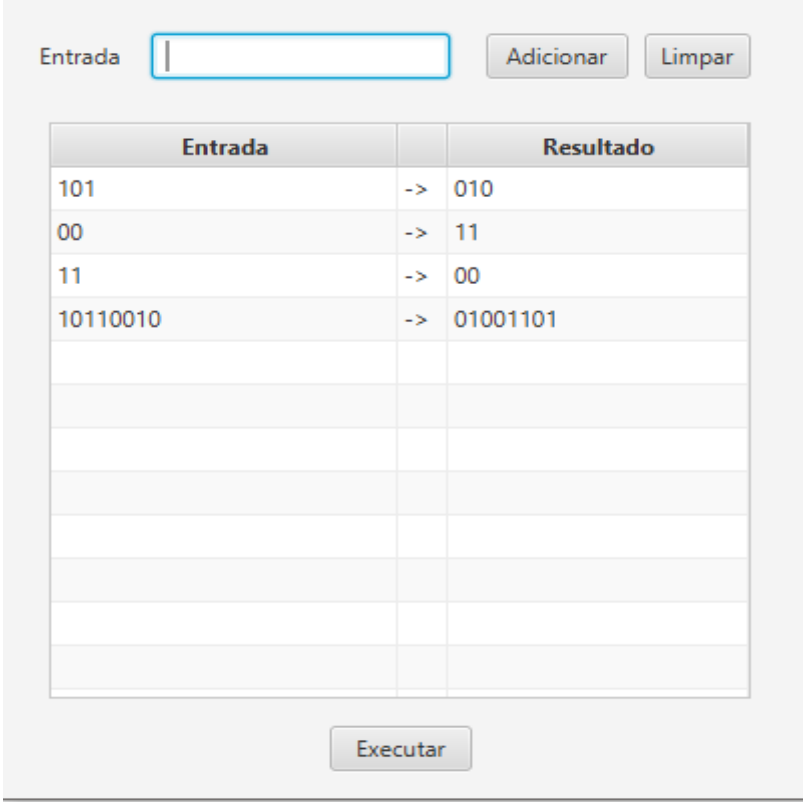


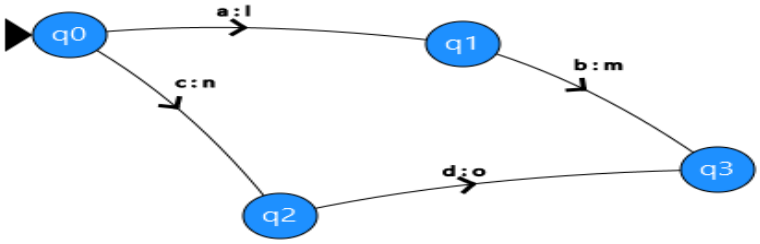
Figura 23 - Simulando a máquina de Mealy

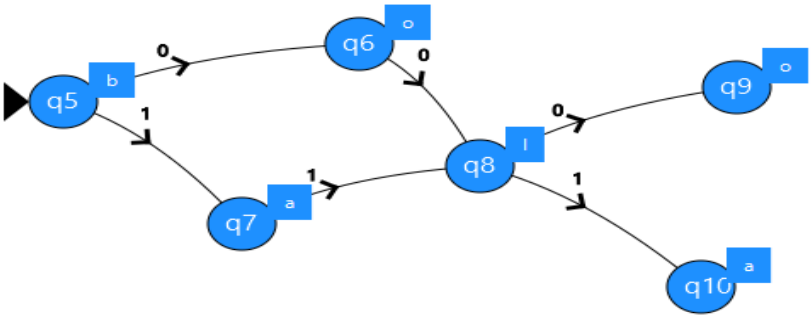
6.3. Implementação das Máquina de Moore e Mealy

Inicialmente, foram realizadas adaptações na versão inicial de representação de autômatos, de forma a representar as devidas singularidades de ambas máquinas adicionais.

Em relação a implementação do algoritmo de aceitação de uma ou múltiplas entradas para cada máquina, partiu-se do princípio de que o autômato está representado corretamente e é determinístico, desta forma, a partir de uma cadeia de entrada, faz-se necessário somente o percurso do autômato com saída enquanto houvesse transições válidas, caso não houvesse transições válidas ou esgota-se as transições o processo é encerrado e a saída exibida, cabe ressaltar que em autômatos com saída não há presença de estados finais.

Por fim, é importante relembrar que para as Máquinas de Moore, o símbolo é adicionado ao entrar no estado, enquanto para as Máquinas de Mealy, o símbolo é adicionado caso a transição seja realizada.





7 LEITURA E GRAVAÇÃO DE AUTÔMATO FINITO EM ARQUIVO XML

A ferramenta permite a leitura de gravação de autômatos finitos em um arquivo XML (*eXtensible Markup Language*), como estruturação do arquivo XML foi utilizado como base a estrutura presente no software JFlap 7, assim a ferramenta além de poder ler e gravar um autômato finito, também pode ler autômatos construídos no JFlap 7.

Para ter acesso à essa funcionalidade basta seguir o caminho: Arquivo -> Abrir Autômato para carregar um arquivo XML de um autômato finito ou Arquivo -> Salvar Autômato para salvar um arquivo XML do autômato finito construído.

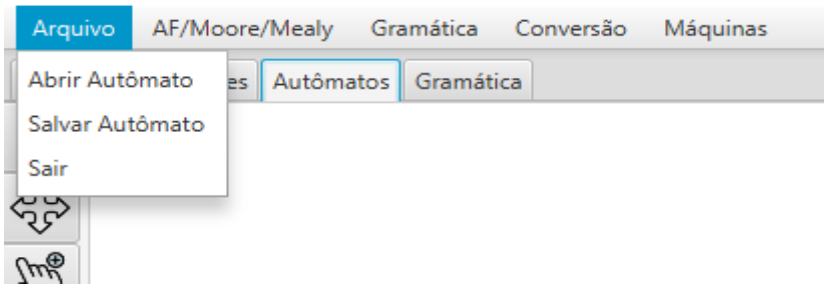


Figura 24 - Menu de leitura e gravação do XML

7.1. Implementação da estrutura do arquivo XML

A leitura e gravação de autômatos finitos em XML (*eXtensible Markup Language*) é compatível com o software JFlap 7, através da criação de arquivos no formato JFF.

Primeiramente, em relação à leitura, a partir da elaboração do autômato são extraídos dados em relação aos estados, tais como identificador, nome, coordenadas e se é um estado inicial e/ou final; e sobre as transições, registrando a origem e o destino assim como o símbolo de aceitação da transição. Adicionalmente são incluídas informações de codificação de caracteres e versão do XML, sendo importante ressaltar que estão versão não suporta a presença de notas de texto nos estados ou no autômato como um todo, as quais se presentes serão ignoradas.

Em relação à gravação, foi utilizada a API nativa do Java para interpretação de XML através do uso de classe de "*parseamento"* DocumentBuilderFactory, a qual viabiliza a extração das *tags*, *sub-tags* e seus respectivos atributos.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <type>fa</type>&#13;
    <automaton>&#13;
        <!--The list of states.-->&#13;
        <state id="0" name="q0">&#13;
            <x>225.0</x>&#13;
            <y>118.0</y>&#13;
            <initial/>&#13;
            <final/>&#13;
        </state>&#13;
        <state id="1" name="q1">&#13;
            <x>420.0</x>&#13;
            <y>132.0</y>&#13;
            <final/>&#13;
        </state>&#13;
        <state id="2" name="q2">&#13;
            <x>375.0</x>&#13;
            <y>269.0</y>&#13;
        </state>&#13;
        <!--The list of transitions.-->&#13;
        <transition>&#13;
            <from>0</from>&#13;
            <to>1</to>&#13;
            <read>a</read>&#13;
        </transition>&#13;
        <transition>&#13;
            <from>2</from>&#13;
            <to>1</to>&#13;
            <read/>&#13;
        </transition>&#13;
        <transition>&#13;
            <from>1</from>&#13;
            <to>1</to>&#13;
```

Figura 25 - Exemplo da estrutura do arquivo XML