UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

PROJETO PRÁTICO – PARTE 1
RELATÓRIO

LINGUAGENS FORMAIS E AUTÔMATOS
PROF. DR. CELSO OLIVETE JÚNIOR

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PRESIDENTE PRUDENTE
JANEIRO - 2017

BRUNO SANTOS DE LIMA
LEANDRO UNGARI CAYRES

PROJETO PRÁTICO – PARTE 1
RELATÓRIO

Projeto prático parte 1 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
JANEIRO – 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).

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, por fim a seção 4 discorre sobre Gramaticas Regulares mostrando o funcionamento da ferramenta para a simulação de GR's.

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.

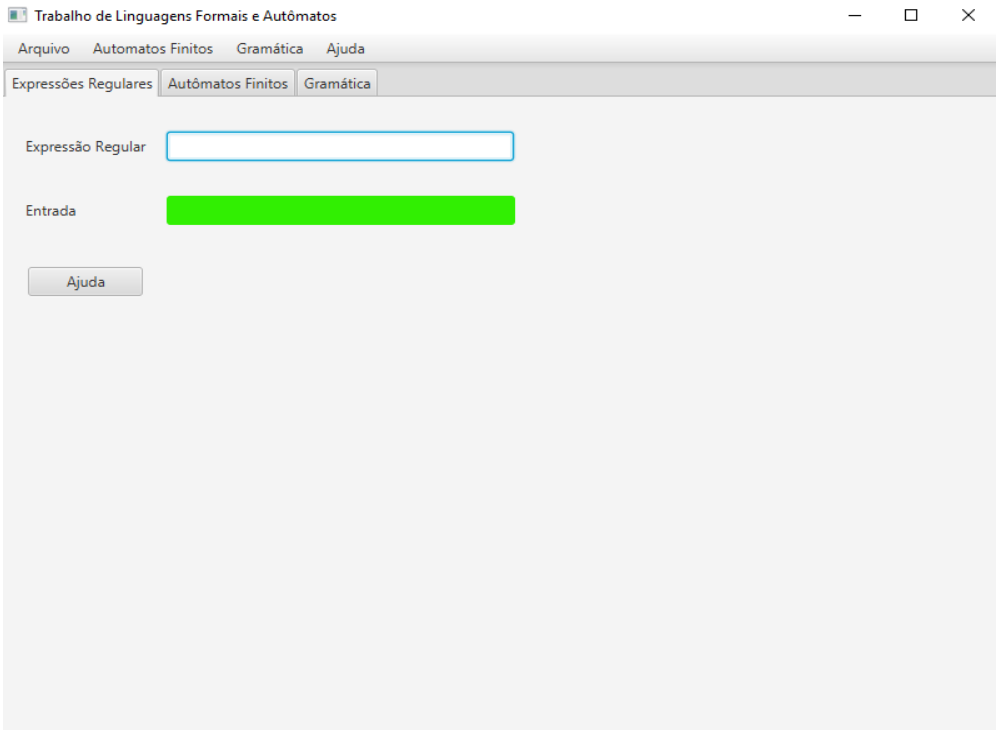

Figura 1 - Instantâneo da ferramenta na tela de Expressões Regulares.

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.

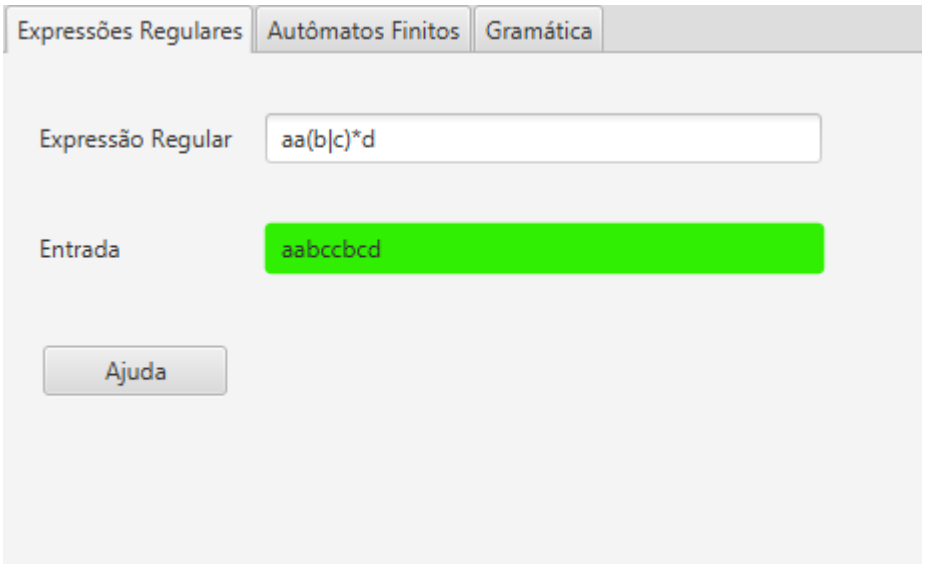

Figura 2 - Exemplo de entrada aceita pela expressão.

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.

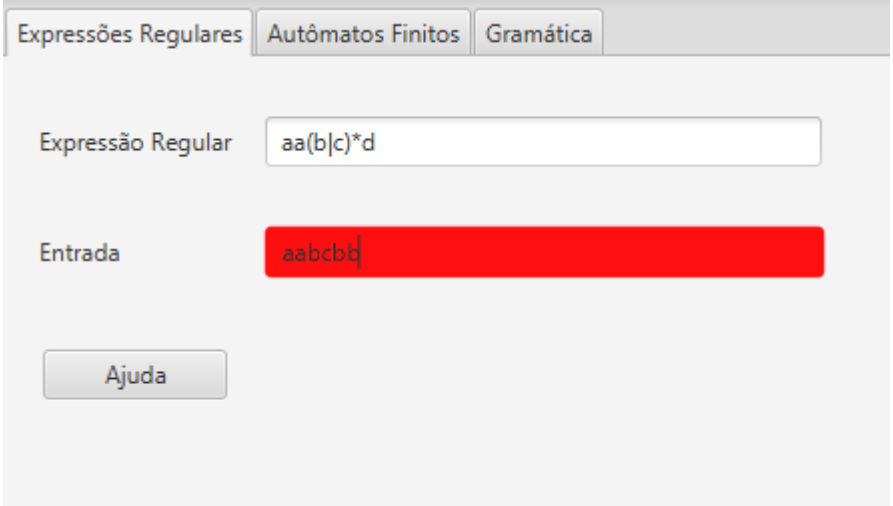

Figura 3 - Exemplo de entrada inválida.

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.

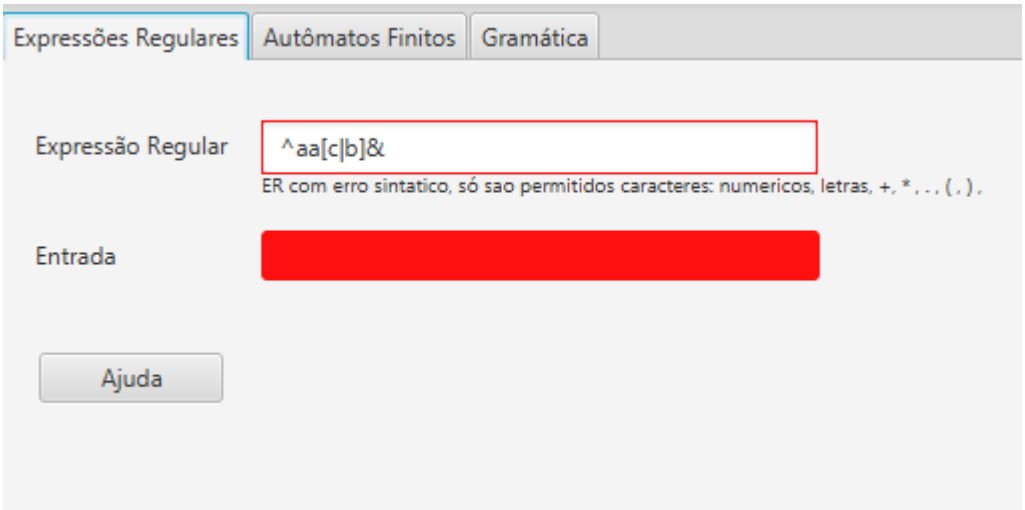

Figura 4 - Exemplo de inserção de símbolos inválidos.

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:

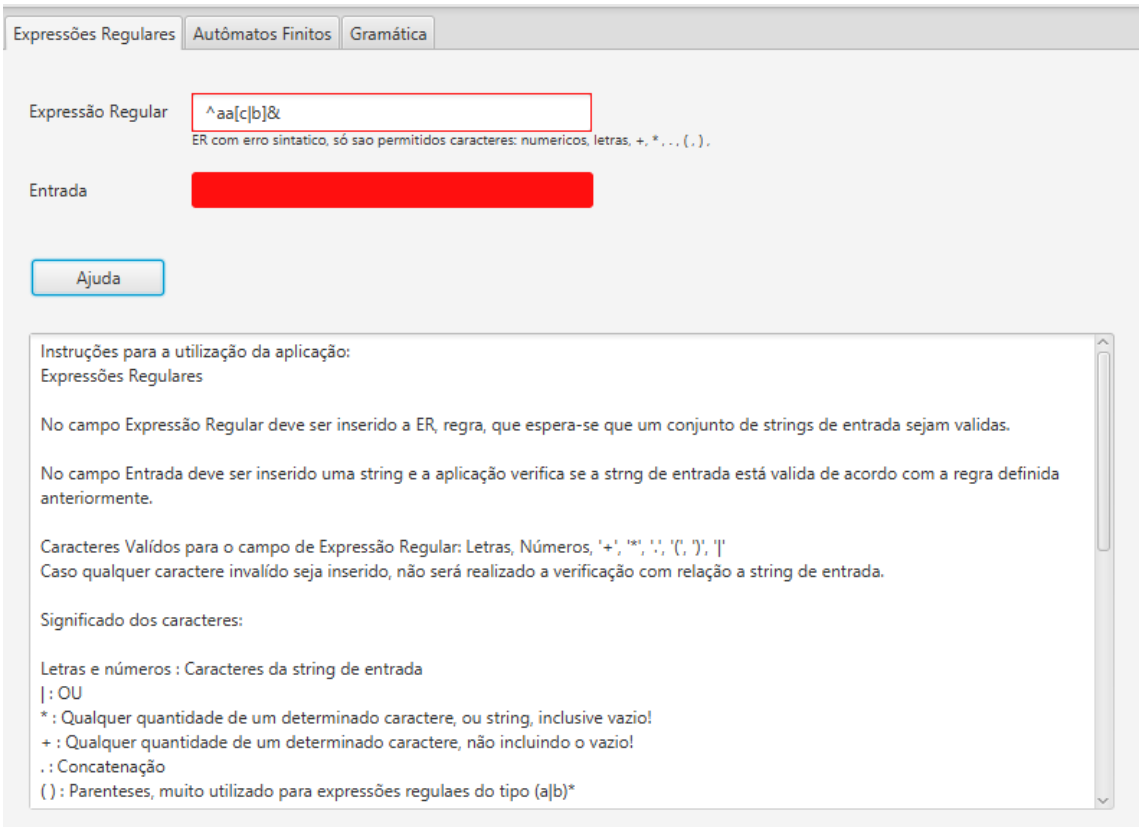

Figura 5 - Exibição do painel de ajuda da expressão regular.

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.

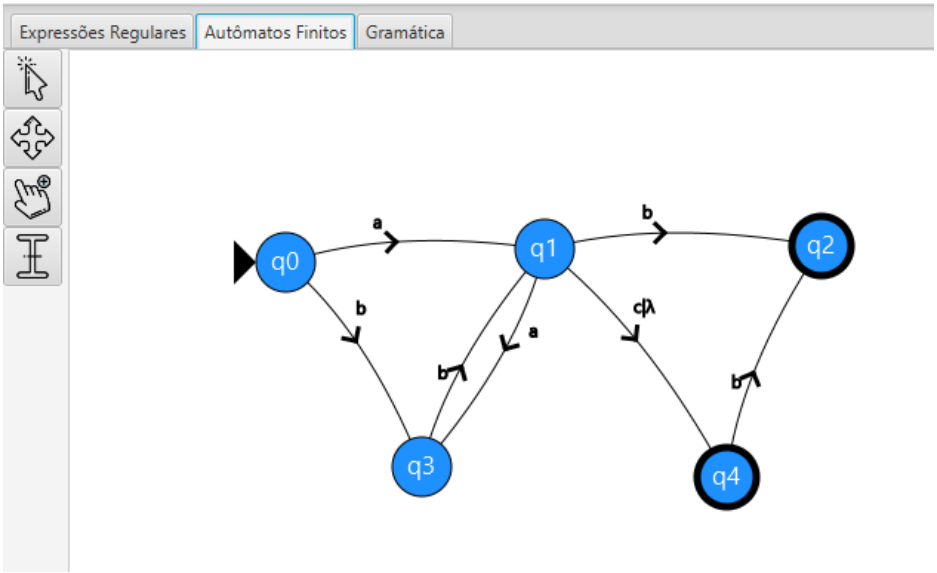

Figura 7 - Exibição de autômato finito.

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.

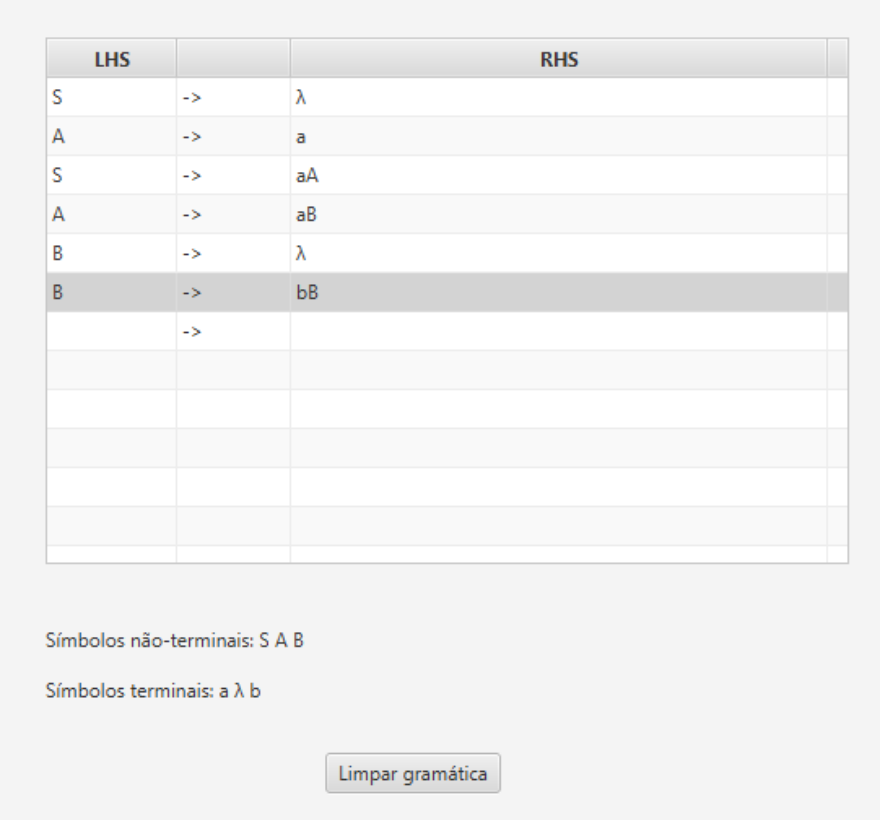

Figura 8 - Detalhamento de uma gramática regular.

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.

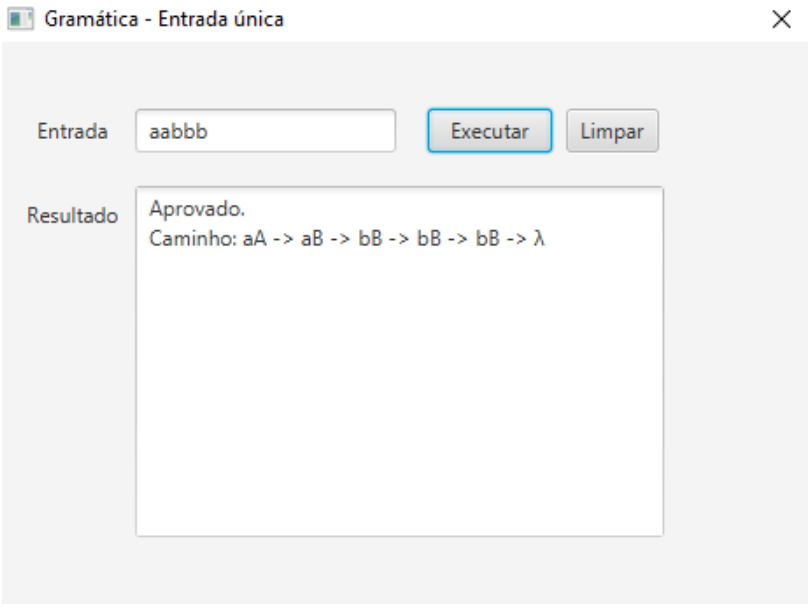

Figura 9 - Resultado da verificação de uma única de entrada para o autômato finito.

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.

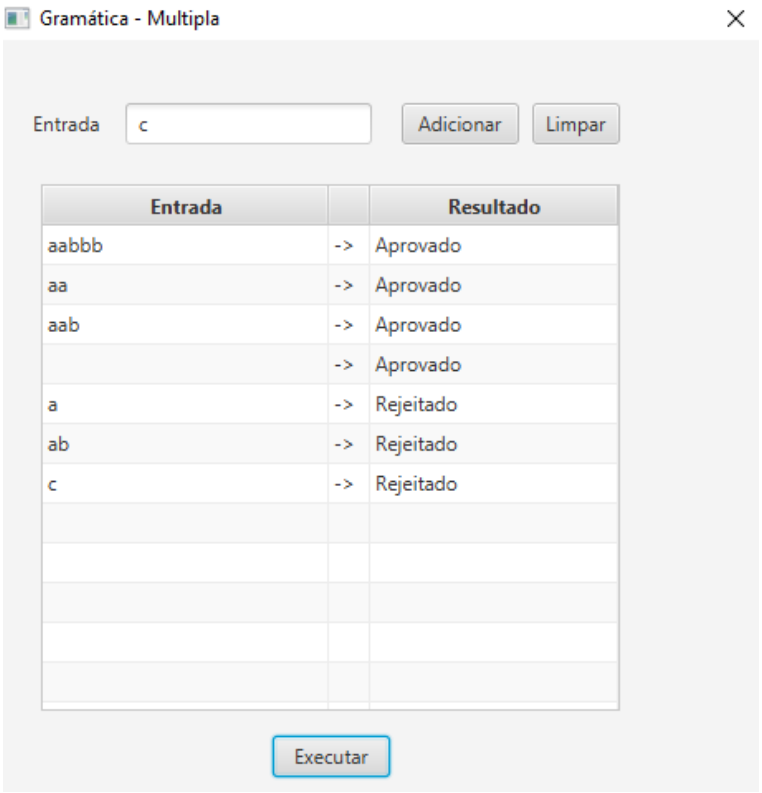

Figura 10 - Resultado da verificação de múltiplas entradas para o autômato finito.

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.