FACULDADE DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE MATEMÁTICA E COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

JUAN CARDOSO DA SILVA

LUIZ FILIPE MIGUEL MONGE

**Trabalho de LFA – Parte 1**

Relatório.



PRESIDENTE PRUDENTE

2019

**Sumário**

[Introdução 3](#_Toc19914756)

[Implementação do RegEx 4](#_Toc19914757)

[Implementação da Gramática 5](#_Toc19914758)

[Implementação do Autômato 7](#_Toc19914759)

# Introdução

Neste semestre foi requisitado o desenvolvimento de um simulador com base no conteúdo dado em aula de Linguagens Formais e Autômatos com os requisitos do professor, este simulador foi desenvolvido em HTML + CSS (com o framework bootstrap 4 para a interface), JavaScript e JQuery (para a lógica e funcionamento da interface).

O desenvolvimento foi por sua maioria conturbado ao desenvolver a gramática e o autômato, devido a determinados algoritmos a serem implementados durante o tempo dado para desenvolver o projeto, porém o RegeEx foi facilmente implementado. O código (parte em JavaScript e JQuery) foi dividido em três partes, interface, lógica, construção de variáveis, apesar de não poder deixa-las completamente separadas (devido a necessidade de que algumas funções sejam executadas antes das outras independente, de qual das três partes elas pertençam(principalmente as funções e, JQuery).

Estrutura e Interface

Divido em quatro arquivos principais, index.html(menu principal), autômatos.html(Autômatos Finitos), regex.html(parte do RegEx) e gramar.html(parte das gramáticas).

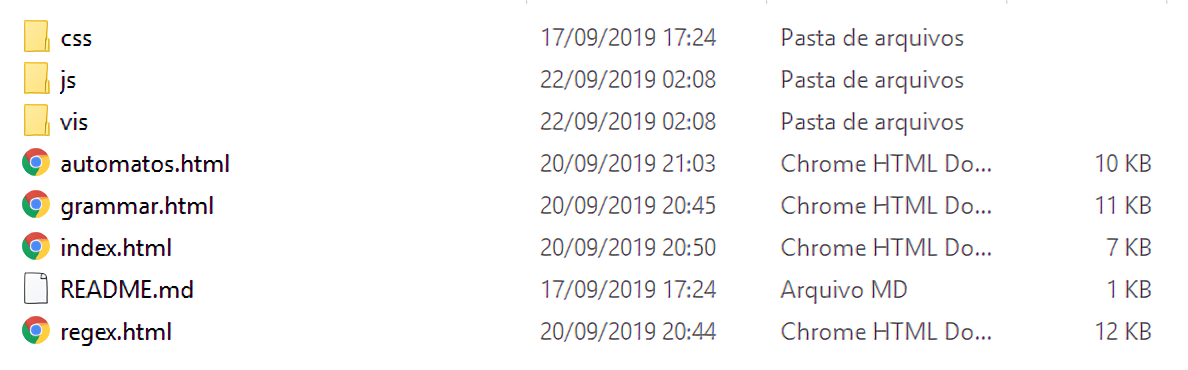


Figura Estruturamento do projeto

A pasta *js* contém todas as lógicas e algoritmos para funcionamento do projeto e da interface e a pasta *css* é para a estilização da interface.

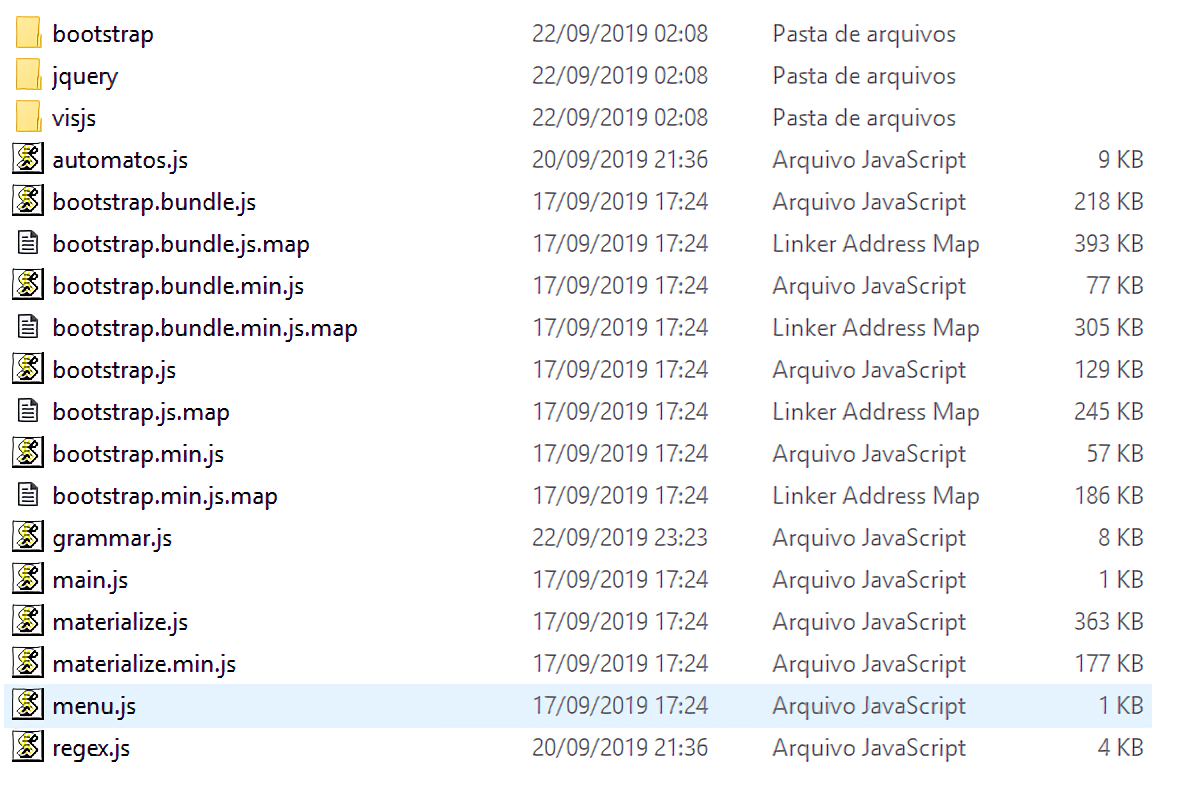


Figura Estruturação dos códigos em JavaScript

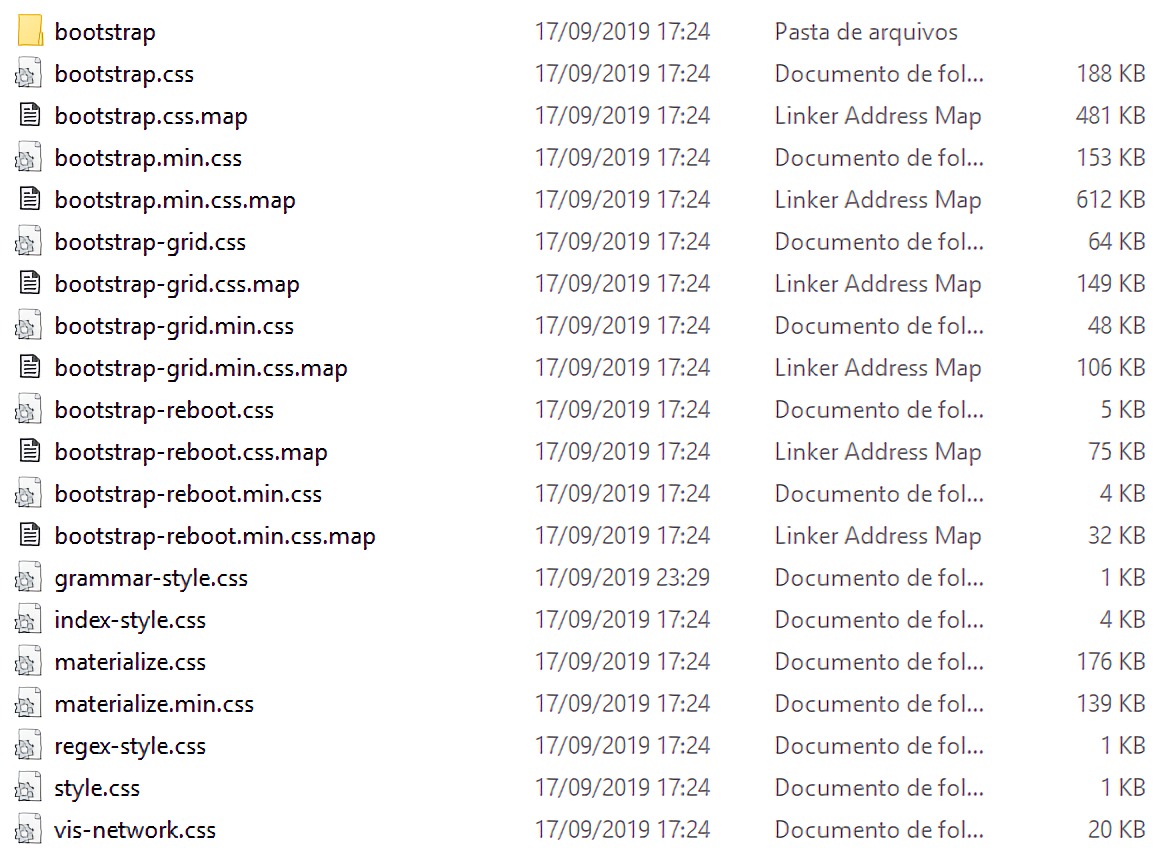


Figura Estruturação do CSS.

**Interface**

Ao abrir o projeto em um navegador com suporte as ferramentas utilizadas e abrir o index.html, a seguinte página irá abrir:

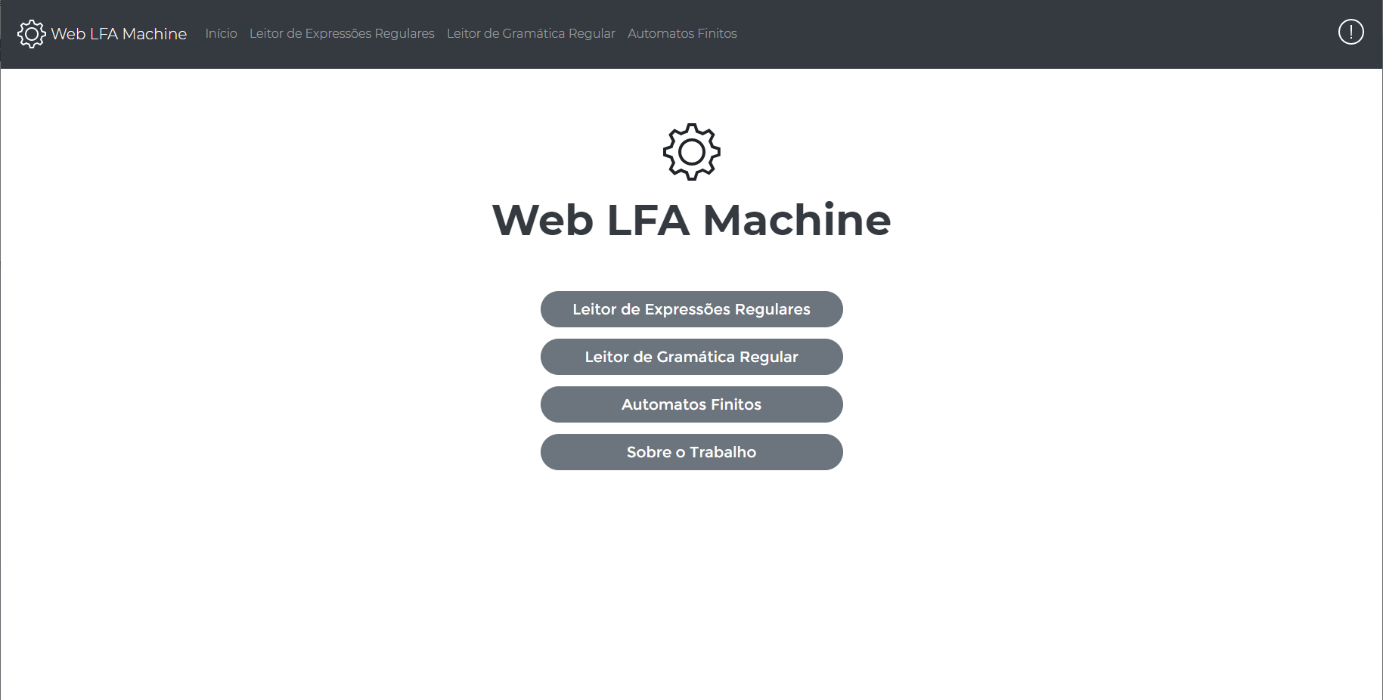
****

Figura página da web principal contendo o menu para acessar os simuladores.

Essa página é o menu do projeto, nela e possível acessar cada simulador que foi projetado, primeiramente vejamos o simulador do RegEx.

**Simulador de RegEx**

O simulador de RegEx, segue de interface bem simples e fácil manuseamento, é colocado no campo de *Expressão* a sua expressão regular e no campo de *Entrada*, para uma *string* que deseja verificar. Para múltiplas entradas, deve-se clicar no botão de mesmo nome para abrir um modal que funciona da mesma maneira que a página principal.

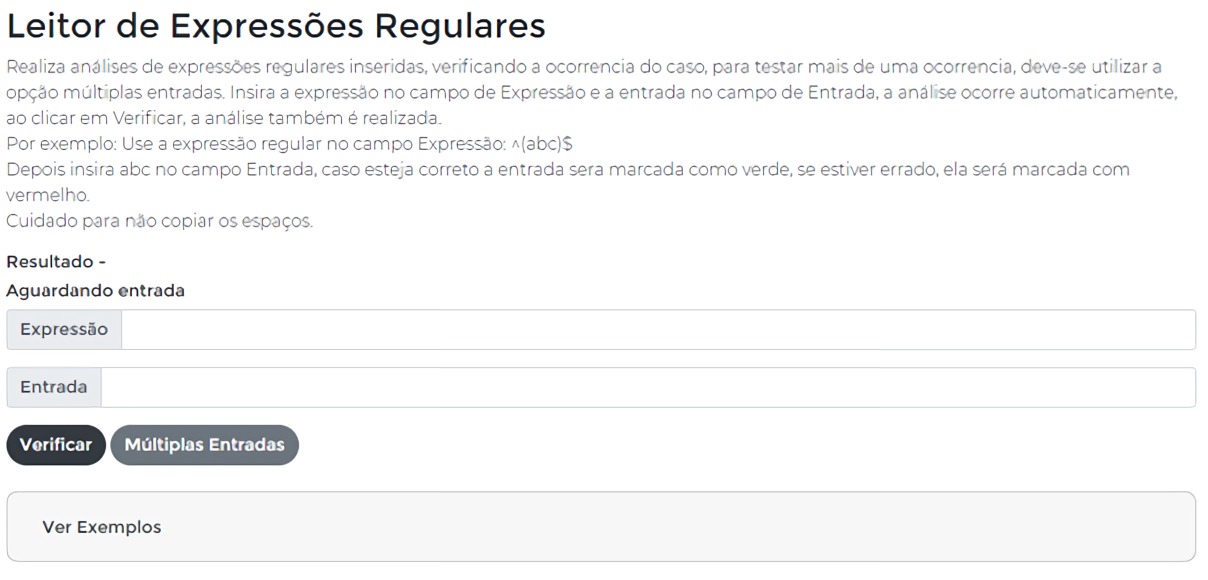


Figura página contendo a interface para manipular o simulador de RegEx..

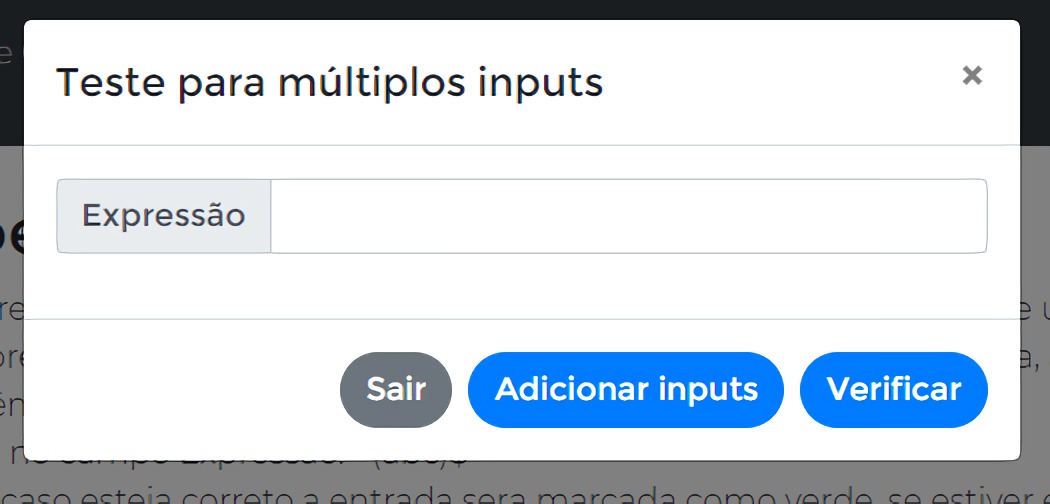


Figura modal para manipular multiplas entradas no simulador.

Clicando em *Adicionar Inputs*, uma nova *Entrada* é adicionado para ser usada, utilize quantas vezes quiser o botão para adicionar quantas entradas achar melhor.

A verificação ocorre quando clicar em verificar para ambos casos de *Entrada* única ou *Múltiplas* *Entradas* (observação*: a entrada única faz verificação automática, mas também permite verificar só de clicar no botão*).

**Exemplos abaixo:**

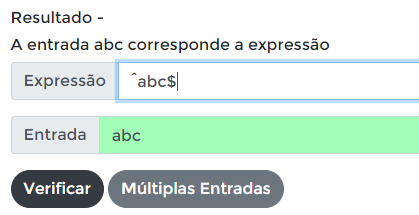
****

Figura exemplo para a única entrada no simualdor de RegEx.



Figura caso para múltiplas entradas.

Caso deseje testar RegEx prontas, existe uma seção com exemplos para ser utilizada no fim da página contendo o simulador.

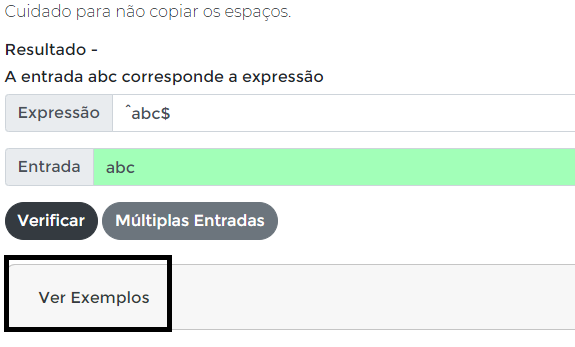
****

Figura visualizador de exemplos.

**Gramática Regular**

A página contendo a gramática também ficou com uma interface simples manuseio, foi utilizado uma tabela fornecida pelo framework do bootstrap 4 para criação de uma tabela, onde cada linha contem três colunas, a primeira sendo as variáveis de derivação, a segunda a seta indicando em que será derivado e a terceira, as derivações em si ( a gramática utilizado no simulador pedido, foi a GLUD, *Gramática Linear Unitária a Direita,*).

****

Figura interface principal para manipular o simulador de gramática.

A opção *Adicionar*, cria uma nova linha na tabela, por padrão, cada linha é preenchida como:

S -> λ

**Observação:** tabela é editável por padrão.

*Testar entrada* e *Múltiplas Entradas* seguem a mesma ideia do RegEx, sendo o primeiro par única entrada e o segundo para várias, apenas a opção de Ajuda se difere.

Ao adicionar uma linha na tabela, sua tabela deve ficar da seguinte maneria:



Nota-se o botão vermelho remover, ele cumpre a função de remover cada linha, individualmente.

Abaixo um exemplo de Gramática Regular ( GLUD ) no simulador:

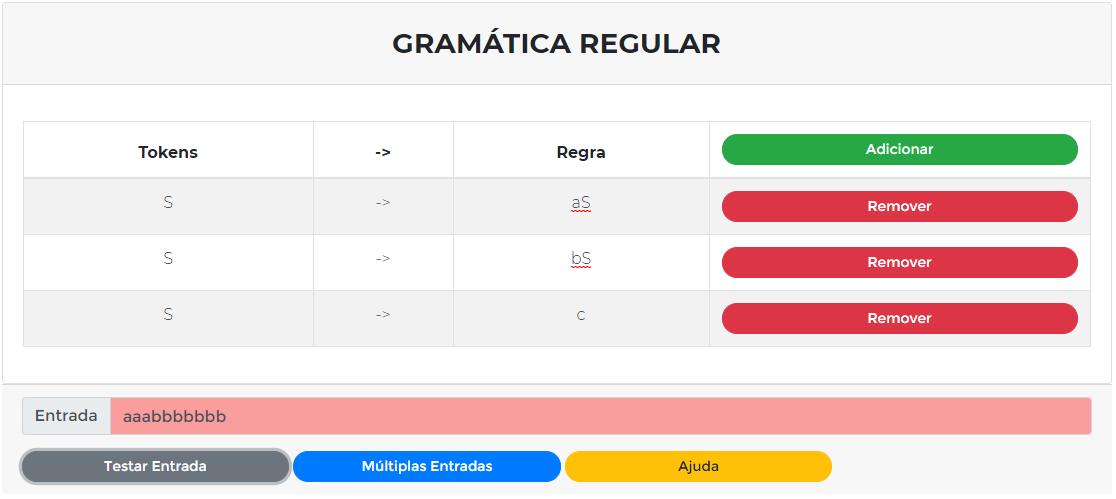
**G = ( {S}, {a,b,c}, P, S ) , termina obrigatoriamente com c.**

Entrada: aaabbbbbbbc.

****

Agora com caso errado:

Entrada: aaabbbbbbb.

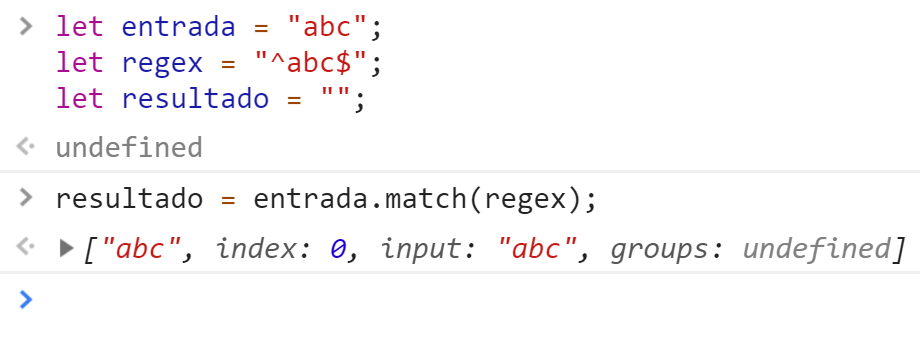
****

**Autômatos Finitos**

A

# Implementação do RegEx

A implementação do RegEx foi utilizando JavaScript(JS), mais precisamente pela documentação mais padronizada MDN Web Docs. O método utilizado foi o “.match()” da própria biblioteca do JS, que realiza uma busca por correspondências entre uma determinada entrada digitada e um RegEx, como no exemplo abaixo:



Logo em seguida é retornado um objeto RegEx com as informações, caso tenha um match entre *entrada* e *regex*, do contrário é retornado null. De resto foi apenas adaptar esse funcionamento da biblioteca fornecida pela própria linguagem para os requisitos do trabalho, utilizando duas funções, uma para verificar uma entrada e a outra para múltiplas entrada, se o padrão existir e o objeto regex for criado, o CSS da página é alterado, verde indicando se está certo e vermelho para errado.



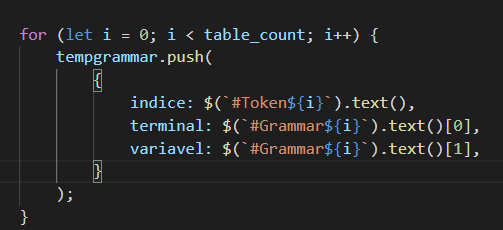
Figura código contendo a lógica da análise do RegEx.

A versão de *Múltiplas Entradas* realiza o mesmo teste, mas iterativamente para cada Entrada adicionada.

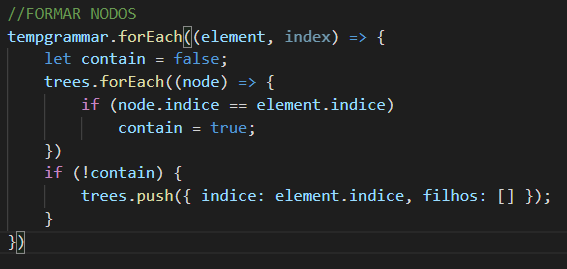
# Implementação da Gramática

A gramática escolhida pela dupla foi GLUD (Gramática Linear Unitária a Direita) e o algoritmo utilizado para realizar a análise foi uma árvore de florestas e uma função com recursividade para percorrer essa árvore,

Primeiramente percorremos todas as linhas das tabelas e pegamos as informações e separamos elas em três variáveis, índice, terminal e variável, cada uma delas recebendo a sua respectiva informação.



Depois criamos os nodos, cada nodo é dividido em 2 variáveis, um índice pra receber a variável/token e uma variável para objeto do vetor de objetos, chamada filhos, para receber o objeto do vetor de objetos.



Logo em seguida é realizada a associação de nodes e sub-nodes conforme o índice (a variável a esquerda da tabela) e a variável( na direita da tabela), os terminais também são adicionados respectivamente.



A função recursiva simplesmente percorre a arvore recursivamente e dentro da função, percorre os filhos iterativamente, se existir um match entre a entrada do usuário, o terminal e a variável conter um *null*, retornamos *true*, do contrário verificamos as mesmas condições, porém, a variável é verificada se ela é diferente de null, assim entramos na função e incrementamos para ir para próxima parte da árvore e da entrada do usuário, o retorno *false* ocorre se não existir nenhum match entre terminais e entrada.

# Implementação do Autômato

Para desenvolver a interface do autômato, foi-se utilizado o Cytoscape, uma biblioteca Open-Source para visualização de redes de interação molecular, a dupla pegou essa base da biblioteca e modificou para facilitar o desenvolvimento visual dos autômatos junto da interface e a criação da lógica.

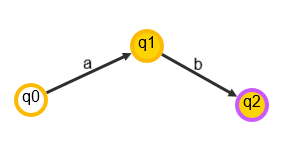


Figura 12 Exemplo de um Autômato feito na biblioteca modificada

Representamos os estados da seguinte maneira, laranja e amarelo para um estado, laranja e branco para o estado no início, roxo e amarelo para o estado no fim.

A lógica em volta da análise do autômato é realizada recursivamente, entra-se na função e verifica a iteração por iteração dos *“links”* de um node com o outro. Nessa iteração verifica-se se o valor armazenado no link é igual ao valor da entrada, caso seja, adicionamos *true* a uma variável *result* e realizamos um break, verificamos se estamos no node final, se não tem mais nodes para percorrer, caso contrário entramos na função incrementando para próxima iteração na recursão, essa próxima iteração vai retornar um valor para o *result*, se nenhuma das verificações for verdade, retornaremos apenas *false*.