



## Pró-reitoria de Pesquisa

---

Santo André, 31 de agosto de 2016.

À Ilustríssima Pró-Reitora de Pesquisa,

Profª. Dra. Marcela Sorelli Carneiro Ramos

Encaminho o relatório do aluno **Rafael Cardoso da Silva** referente ao projeto de pesquisa junto ao programa de Iniciação Científica na modalidade **PIC** no edital **01/2015**.

O aluno Rafael Cardoso da Silva desenvolveu com sucesso o projeto "Um sistema para auxiliar na aprendizagem da disciplina Linguagens Formais e Autômatos". Seu desempenho foi excelente: ele implementou os algoritmos propostos e desenvolveu o sistema segundo o cronograma que consta no projeto. O sistema já foi utilizado para aplicação e correção automatizada de exercícios para duas turmas da disciplina no segundo quadrimestre de 2016 e não apresentou falhas. A única dificuldade que o Rafael teve foi na hora de escrever o relatório. Ele tem dificuldade de redigir textos usando a língua portuguesa formal e ainda está trabalhando para sanar essa deficiência.

Nome do Orientador: **Daniel M. Martin**

Rafael Cardoso da Silva

**Um sistema para auxiliar na  
aprendizagem da disciplina  
Linguagens Formais e Autômatos**

Santo André - SP, Brasil

2016

Rafael Cardoso da Silva

**Um sistema para auxiliar na  
aprendizagem da disciplina  
Linguagens Formais e Autômatos**

Relatório final apresentado ao Programa de  
Iniciação Científica da Universidade Federal  
do ABC.

Universidade Federal do ABC  
Bacharelado em Ciências da Computação

Orientador: Prof. Daniel M. Martin

Santo André - SP, Brasil

2016

# Resumo

Resolver exercícios é fundamental para um aluno fixar os conceitos apresentados em aula. Por outro lado, ter seus exercícios corrigidos também é muito importante, para que ele possa avaliar o seu aprendizado. Na UFABC, a disciplina de Linguagens Formais e Autômatos contempla vários exercícios que admitem infinitas respostas, o que torna a correção deles praticamente impossível, principalmente quando as turmas são grandes. O objetivo deste projeto foi a criação e implementação de um sistema para aplicação e correção automática de exercícios envolvendo autômatos finitos determinísticos. Através do estudo de métodos e algoritmos presentes na literatura, foi possível implementar o teste de equivalência entre o autômato-resposta do aluno e o autômato-gabarito previamente armazenado no banco de dados. Ao final do projeto, o sistema foi usado em caráter experimental numa turma da UFABC da disciplina de Linguagens Formais e Autômatos, afim de testar a sua qualidade. E ao final da disciplina, a nota que os alunos obtiverem ao revolver os exercícios do sistema ajudarão a compor o conceito final de cada um na disciplina.

**Palavras-chave:** autômato finito, equivalência de autômatos, minimização de autômato, programação para web

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>5</b>
<b>2</b>	<b>O PRINCIPAL PROBLEMA</b>	<b>6</b>
<b>3</b>	<b>MÉTODOS PARA RESOLVER O PROBLEMA DA EQUIVALÊNCIA</b>	<b>8</b>
<b>3.1</b>	<b>Minimização de Moore</b>	<b>8</b>
3.1.1	Recuperação de Palavra	12
<b>3.2</b>	<b>Equivalência em Tempo Linear</b>	<b>13</b>
3.2.1	A Estrutura Union-Find	13
3.2.2	Algoritmo de Hopcroft e Karp	15
3.2.3	Detalhes de Implementação do Algoritmo de Teste de Equivalência	16
3.2.4	Recuperação de Palavra	17
<b>3.3</b>	<b>Escolha do Melhor Método</b>	<b>18</b>
<b>4</b>	<b>O SISTEMA</b>	<b>19</b>
<b>4.1</b>	<b>Interface de entrada da resposta</b>	<b>19</b>
<b>5</b>	<b>DESENVOLVENDO O SISTEMA</b>	<b>22</b>
<b>5.1</b>	<b>Modelo de Processo</b>	<b>22</b>
<b>5.2</b>	<b>Requisitos e Modelagem do Sistema</b>	<b>23</b>
5.2.1	Requisitos Funcionais	23
5.2.2	Requisitos Não-Funcionais	24
5.2.3	Modelos Funcionais	25
5.2.4	Modelos de Dados	26
<b>5.3</b>	<b>Análise</b>	<b>26</b>
<b>5.4</b>	<b>Projeto</b>	<b>28</b>
5.4.1	Estilo Arquitetural	28
5.4.2	Interface do Sistema	30
5.4.3	O DFAdesigner e o Juiz	33
<b>5.5</b>	<b>Implementação</b>	<b>37</b>
<b>6</b>	<b>O SISTEMA DFAJUDGE</b>	<b>40</b>
<b>6.1</b>	<b>Designer do DFAjudge</b>	<b>52</b>
<b>7</b>	<b>TESTE DO SISTEMA EM UMA TURMA</b>	<b>54</b>
<b>8</b>	<b>PRÓXIMOS PASSOS</b>	<b>56</b>

<b>9</b>	<b>CONCLUSÃO . . . . .</b>	<b>57</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>58</b>

# 1 Introdução

Resolução de exercícios é uma atividade fundamental para que um aluno possa praticar o que foi apresentado em aula. A correção dos exercícios também é importante para que o aluno possa avaliar seu aprendizado. Para turmas grandes, torna-se inviável corrigir rapidamente todos os exercícios propostos e devolvê-los em tempo hábil aos alunos. Na disciplina MC3106 – Linguagens Formais e Autômatos – vários exercícios da parte inicial da disciplina pedem como resposta um autômato finito determinístico. Tais exercícios são particularmente trabalhosos de se corrigir porque cada um deles admite infinitas respostas corretas e, por esse motivo, sua correção se torna exaustiva para o monitor ou docente responsável.

Com o intuito de agilizar essa tarefa, este projeto propõe o desenvolvimento de um sistema web que permita ao aluno inserir seu autômato-resposta por meio de uma interface intuitiva e fácil de ser utilizada e obter um *feedback* quase que imediatamente, um tempo drasticamente reduzido se comparado à correção manual. Acreditamos que um sistema como esse será de grande valia para o aprendizado dos alunos. Por outro lado, para o docente responsável, facilitará o processo de composição do conceito final na disciplina de Linguagens Formais e Autômatos.

Além do estudo de autômatos e linguagens regulares, a disciplina de Linguagens Formais e Autômatos trata de linguagens não-regulares, em especial das que podem ser descritas por gramáticas livres de contexto (GLC). Nesta segunda parte da disciplina, é também comum a ocorrência de exercícios que pedem, como resposta, uma gramática livre de contexto para descrever uma linguagem livre de contexto. Automatizar a correção deste tipo de exercício é impossível pois o problema geral de se determinar se duas gramáticas livres de contexto descrevem a mesma linguagem é um problema indecidível (SIPSER, 2012). Por outro lado, diversos tópicos em linguagens livres de contexto podem ser examinados de maneira automática, incluindo a equivalência de certas subclasses de gramáticas livres de contexto (NIJHOLT, 1981), a transformação de uma gramática livre de contexto qualquer para a Forma Normal de Chomsky (FNC), a busca da árvore de derivação para um texto segundo uma GLC na FNC, problemas de *parsing* em geral.

## 2 O Principal Problema

Um *autômato finito determinístico* (também chamado de AFD ou máquina de estados) consiste de um conjunto de estados não vazio  $Q$ , um alfabeto  $\Sigma$ , um estado inicial  $s \in Q$ , um conjunto de estados de aceitação  $F \subseteq Q$  e uma função de transição  $\delta: Q \times \Sigma \rightarrow Q$ . É comum estender a função de transição para uma função  $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$  definida indutivamente por

- (i)  $\hat{\delta}(q, \varepsilon) = q$  para todo estado  $q \in Q$ ;
- (ii)  $\hat{\delta}(q, \sigma) = \delta(q, \sigma)$  para todo estado  $q \in Q$  e símbolo  $\sigma \in \Sigma$ ;
- (iii)  $\hat{\delta}(q, \sigma_1 \cdots \sigma_k) = \delta(\hat{\delta}(q, \sigma_1 \cdots \sigma_{k-1}), \sigma_k)$  para todo estado  $q \in Q$  e símbolos  $\sigma_1, \dots, \sigma_k \in \Sigma$ .

Todo autômato pode ser representado por um diagrama de estados. Considere o diagrama da Figura 1 abaixo.

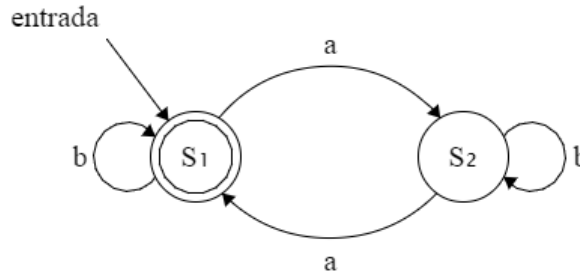


Figura 1 – Um autômato  $M_1$ .

No autômato da Figura 1, temos  $Q = \{S_1, S_2\}$ ,  $\Sigma = \{a, b\}$ ,  $s = S_1$ ,  $F = \{S_1\}$  e a função de transição é dada pela tabela a seguir.

$\delta$	<b>a</b>	<b>b</b>
$S_1$	$S_2$	$S_1$
$S_2$	$S_1$	$S_2$

Tabela 1 – Tabela de transições do  $M_1$ .

Nesse exemplo, se o AFD  $M_1$  é alimentado com a palavra **aabab**, ele irá começar no estado  $S_1$  e irá percorrer os estados  $S_2, S_1, S_1, S_2, S_2$  nesta ordem, e irá rejeitar a palavra dada (pois o último estado  $S_2 \notin F$ ). Se, para uma outra palavra  $w$ , a simulação tivesse terminado em  $S_1$  (que pertence a  $F$ ) diríamos que o autômato aceita a palavra  $w$ . Mais formalmente, se  $M_1 = (Q, \Sigma, \delta, s, F)$  é um autômato finito determinístico, dizemos que  $M_1$  *aceita*  $w$  se  $\hat{\delta}(s, w) \in F$  e que  $M_1$  *rejeita*  $w$  caso contrário.

A *linguagem reconhecida* por um autômato  $M_1 = (Q, \Sigma, \delta, s, F)$  é o conjunto de palavras que são aceitas por esse autômato, ou seja, é definida pelo conjunto

$$L(M_1) = \{w \in \Sigma^* : \hat{\delta}(s, w) \in F\}.$$



No exemplo da Figura 1 acima, a linguagem reconhecida por  $M_1$  é  $L(M_1) = \{w \in \Sigma^* : w \text{ tem número par de símbolos } a\}$ .

Vários tipos de exercícios pedem por uma resposta que é um autômato. Por exemplo:

- (i) construir um autômato que reconheça a linguagem regular dada por um certo conjunto de palavras descrito matematicamente;
- (ii) transformar um autômato finito não determinístico num determinístico equivalente;
- (iii) criar um autômato que reconheça a mesma linguagem descrita por uma expressão regular dada;
- (iv) construir o menor autômato finito determinístico que reconheça uma certa linguagem regular.

Cada um destes tipos de exercícios possui centenas de casos particulares que os alunos podem fazer para praticar os conceitos de autômatos abordados na disciplina. Tais exercícios podem, por exemplo, ser encontrados abundantemente no livro *Introdução à Teoria da Computação* de M. Sipser (SIPSER, 2012). Outras referências importantes da disciplina que possuem exercícios semelhantes são (HOPCROFT; MOTWANI; ULLMAN, 2006) e (LINZ, 2011). Em todos os casos acima, contudo, há uma infinidade de respostas corretas para cada exercício. Por exemplo, considere o diagrama abaixo

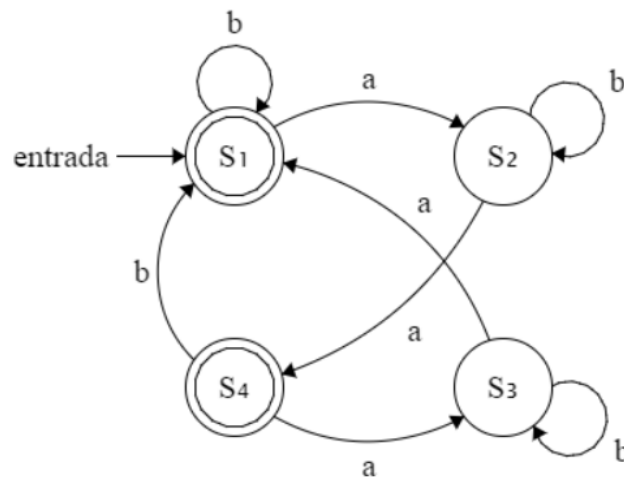


Figura 2 – Um autômato  $M_2$ .

Apesar de visivelmente maior e mais complexo que  $M_1$ , este autômato reconhece a mesma linguagem que o autômato  $M_1$  da Figura 1. É possível demonstrar que, para cada linguagem regular, existem infinitos autômatos que a reconhecem. Como decidir então se o autômato-resposta dado por um aluno reconhece a linguagem pedida? No caso do sistema que iremos desenvolver isso se reduz ao seguinte problema:

**Problema:** Decidir se dois autômatos finitos determinísticos reconhecem a mesma linguagem.

## 3 Métodos para resolver o problema da equivalência

Determinar se dois autômatos finitos e determinísticos reconhecem linguagens diferentes é um problema bem resolvido. A ideia fundamental é tentar encontrar uma palavra  $w$  que é aceita por um autômato e rejeitada pelo outro. Se tal palavra puder ser encontrada, então as linguagens reconhecidas por esses autômatos são distintas. Caso tal palavra não exista, os autômatos reconhecem a mesma linguagem.

Mais formalmente, dizemos que dois estados  $q_1$  e  $q_2$  (no mesmo autômato ou não) são *equivalentes* se, para toda palavra  $w \in \Sigma^*$  vale que  $\hat{\delta}(q_1, w) \in F$  se e somente se  $\hat{\delta}(q_2, w) \in F$ , caso contrário  $q_1$  e  $q_2$  são chamados distinguíveis. Dois autômatos serão, portanto, equivalentes se e somente se seus respectivos estados iniciais forem equivalentes no sentido que acabamos de definir.

Esse problema está intimamente relacionado com o problema de minimização de autômatos finitos determinísticos. Os primeiros algoritmos desenvolvidos para testar a equivalência de autômatos resolviam também o problema da minimização. Eles apareceram em (HUFFMAN, 1954) e (MOORE, 1956) e podem ser implementados sem muito esforço em tempo  $O(n^4)$ , onde  $n$  é o número total de estados. O algoritmo mais eficiente conhecido para minimização de autômatos (HOPCROFT, 1971) executa em tempo  $O(n \log n)$ . Posteriormente Hopcroft e Karp (HOPCROFT; KARP, 1971b) desenvolveram um algoritmo linear para testar a equivalência de autômatos.

### 3.1 Minimização de Moore

Nessa seção vamos investigar um algoritmo de minimização de autômato finito determinístico, mais especificamente o algoritmo de Minimização de Moore (MOORE, 1956) e discutido no (HOPCROFT; MOTWANI; ULLMAN, 2006).

Para facilitar a compreensão utilizaremos como exemplo o autômato  $M_2$  da Figura 2. Queremos determinar se  $M_2$  é equivalente ao autômato  $M_1$  da Figura 1.

O autômato  $M_2$  é representado pela quintupla  $M_2 = (Q, \Sigma, \delta, s, F)$ , onde temos  $Q = \{S_1, S_2, S_3, S_4\}$ ,  $\Sigma = \{a, b\}$ ,  $s = S_1$ ,  $F = \{S_1, S_4\}$  e a função de transição é dada pela tabela a seguir.

$\delta$	<b>a</b>	<b>b</b>
$S_1$	$S_2$	$S_1$
$S_2$	$S_4$	$S_2$
$S_3$	$S_1$	$S_3$
$S_4$	$S_3$	$S_1$

Tabela 2 – Tabela de transição do  $M_2$ .

Note que a definição de estados equivalentes na seção anterior não se traduz imediatamente em um algoritmo, pois não é possível testar se  $\hat{\delta}(p, w) \in F$  para todas as palavras  $w \in \Sigma^*$

No entanto, podemos desenvolver um algoritmo para determinar a equivalência entre estados  $p$  e  $q$  baseando-se no seguinte lema:

**Lema 1.** *Se  $r = \delta(q, \sigma)$  e  $s = \delta(p, \sigma)$  e se  $r$  e  $s$  são distinguíveis, então  $p$  e  $q$  são distinguíveis.*

**Prova.** Por hipótese deve haver uma palavra  $w$  que distingue  $r$  de  $s$ , ou seja, exatamente um dos estados  $\hat{\delta}(r, w)$  e  $\hat{\delta}(s, w)$  é de aceitação. Então a palavra  $\sigma w$  distingue  $p$  de  $q$ , já que  $\{\hat{\delta}(p, \sigma w), \hat{\delta}(q, \sigma w)\}$  é o mesmo par de estados que  $\{\hat{\delta}(r, w), \hat{\delta}(s, w)\}$ .  $\square$

A partir do Lema acima podemos elaborar um processo para investigar quais são os pares de estados que são distinguíveis. Para qualquer par de estados, eles são distinguíveis se constatar que um simbolo do alfabeto aplicado na função  $\delta$ , em cada estado deste par, resulta em um par de estados que já estão distinguidos. E também ter como premissa que se num par de estados, um destes estados é de aceitação e o outro não, então estes estados são distinguidos com a palavra vazia, denotada por  $\varepsilon$ .

Entendido o processo de como verificar se um par de estados são distinguíveis, podemos ver o pseudo código, que realiza as verificações para todos os possíveis pares, logo a baixo.

**Algoritmo 1:** Acha Pares Distinguíveis pela Minimização de Moore**Entrada:**  $Q, \Sigma, \delta, F$ **Saída:** Tabela de distinguibilidade

```

1 início
2   para cada  $p \in Q$  faça
3     para cada  $q \in Q$  faça
4        $D\{p, q\} \leftarrow \emptyset$ 
5   para cada  $p \in F$  faça
6     para cada  $q \in Q \setminus F$  faça
7        $D\{p, q\} \leftarrow x$ 
8   distinguiu_algo  $\leftarrow$  verdadeiro
9   enquanto distinguiu_algo faça
10    distinguiu_algo  $\leftarrow$  falso
11    para cada  $p \in Q$  faça
12      para cada  $q \in Q$  faça
13        se  $D\{p, q\} = \emptyset$  então
14          para cada  $\sigma \in \Sigma$  faça
15            se  $D\{\delta(p, \sigma), \delta(q, \sigma)\} \neq \emptyset$  então
16               $D\{p, q\} \leftarrow x$ 
17              distinguiu_algo  $\leftarrow$  verdadeiro
18  retorna  $D$ 

```

Ao analisarmos este pseudo código de algoritmo de minimização vemos que este recebe como entrada o conjunto de estados  $Q$ , o alfabeto  $\Sigma$ , a tabela de transição  $\delta$  e o conjunto de estados finais  $F$  de um autômato.

E contamos com o apoio de uma matriz quadrada simétrica  $D$  de dimensão  $|Q| \times |Q|$ , cujas células abaixo da diagonal principal serão utilizadas para representar todos os pares de estados possíveis. Inicialmente,  $D$  será inicializada com valores vazios. Ao longo da execução do algoritmo, se um par for distinguido, então atribuiremos o símbolo “x” à célula correspondente a este par. Logo em seguida, preenchemos as células dos pares trivialmente distinguíveis, ou seja, pares que consistem de um estado final e um não final.

Então iniciam-se as rodadas de verificação, onde percorre-se  $D$  e para cada par  $\{p, q\}$  com valor vazio, o algoritmo testa para cada carácter  $\sigma$  do conjunto  $\Sigma$ , se o par  $\{p, q\}$  se torna distinguível pelo caractere  $\sigma$ , considerando  $\hat{\delta}(p, w)$  e  $\hat{\delta}(q, w)$  em vista do Lema 1.

Além disto, com o apoio da variável booleana *distinguiu\_algo*, utilizada para

marcar se houve uma alteração em  $D$ , verificamos que as rodadas somente cessarão se, na mesma rodada, nenhum “x” é adicionado na tabela de distinguibilidade, concluindo assim que não existem mais pares à ser distinguidos. Retornando, ao fim, a tabela de distinguibilidade.

Concluimos que não pode haver mais de  $O(n^2)$  verificações em uma rodada, assim certamente o limite superior para o número de verificações é  $O(n^4)$ , visto que o pior caso é atingido quando somente um único par de estados é distinguido por rodada, causando  $O(n^2)$  rodadas com  $O(n^2)$  verificações em cada, onde  $n$  é o tamanho do conjunto  $Q$ .

Então, para solucionarmos o nosso problema de verificar se dois autômatos são equivalentes, podemos considerar um terceiro autômato que é a união disjunta dos dois autômatos dados e executamos o algoritmo acima sobre a união. E, ao final, apenas verificamos se a posição na tabela que representa o par de estados iniciais encontra-se vazia.

Para o nosso exemplo, ao unirmos o  $M_1$  com o  $M_2$ , obtemos a seguinte tabela de transição. Mas com a nomenclatura dos estados de  $M_2$  mapeados de  $\{S_1, S_2, S_3, S_4\}$  para  $\{S_3, S_4, S_5, S_6\}$ .

$\delta$	<b>a</b>	<b>b</b>
$S_1$	$S_2$	$S_1$
$S_2$	$S_1$	$S_2$
$S_3$	$S_4$	$S_3$
$S_4$	$S_6$	$S_4$
$S_5$	$S_3$	$S_5$
$S_6$	$S_5$	$S_3$

Tabela 3 – Tabela de Transição dos autômatos  $M_1$  e  $M_2$  unidos.

E ao executar o algoritmos de equivalência, obteremos a seguinte tabela de distinguibilidade.

$S_2$	x				
$S_3$		x			
$S_4$	x		x		
$S_5$	x		x		
$S_6$		x		x	x
	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$

Tabela 4 – Tabela de distinguibilidade dos autômatos  $M_1$  e  $M_2$  unidos.

Basta agora verificar se a posição da matriz  $D$  correspondente ao par de estados iniciais está com o seu valor vazio, o que indica que os estados iniciais são equivalentes. Neste caso, os autômatos  $M_1$  e  $M_2$  são equivalentes.

### 3.1.1 Recuperação de Palavra

Este sistema de apoio a aprendizagem tem como objetivo o rápido feedback sobre a resposta submetida pelo usuário. Com a ajuda do Algoritmo 1 conseguimos verificar se o autômato-resposta é equivalente ao autômato-gabarito perfeitamente. Mas, caso o Algoritmo 1 retornar que não são equivalentes, então se faz necessário de uma dica ao usuário para que ele tome conhecimento de seu erro. Esta dica será uma palavra  $w \in \Sigma^*$ , que distingue o estado inicial do autômato-resposta submetido do estado inicial do autômato-gabarito, e ao ser percorrida pelo usuário, permite encontrar o erro cometido em sua resposta.

Então é necessária a implementação de um algoritmo que descubra qual é essa palavra que distingue os dois autômatos. Para que seja possível encontrar a palavra desejada é necessário modificar o Algoritmo 1, para que ele não somente descubra quais pares de estados são distinguíveis, mas também guarde os símbolos iniciais das palavras que as distinguem. Isso pode ser feito alterando-se apenas as linhas 7 e 16 do Algoritmo 1 como indicado a seguir:

7.  $D\{p, q\} \leftarrow \varepsilon$   
 16.  $D\{p, q\} \leftarrow \sigma$

Então supondo que  $D\{p, q\} \neq \emptyset$ , temos o seguinte pseudo código.

---

**Algoritmo 2:** Acha a palavra que distingue os dois autômatos

---

**Entrada:**  $D, \delta, \{p_0, q_0\}$

**Saída:**  $w$

```

1 início
2    $w \leftarrow \varepsilon$ 
3    $\{p, q\} \leftarrow \{p_0, q_0\}$ 
4    $\sigma \leftarrow D(\{p, q\})$ 
5   enquanto  $\sigma \neq \varepsilon$  faça
6      $w \leftarrow w \cdot \sigma$ 
7      $\{p, q\} \leftarrow \{\delta(p, \sigma), \delta(q, \sigma)\}$ 
8      $\sigma \leftarrow D(\{p, q\})$ 
9   retorna  $w$ 
```

---

O algoritmo recebe como entrada  $D, \delta$  e  $\{p_0, q_0\}$ , que é a tabela de distinguibilidade gerado pelo Algoritmo 1 com as modificações sugeridas acima, a tabela de transição dos autômatos unidos e o par de estados iniciais dos autômatos, respectivamente.

O Algoritmo 2 irá coletando os símbolos contidos em células específicas de  $D$  e unindo-os ao conjunto  $w$ . Iniciando pelo par  $\{p_0, q_0\}$ , descobrindo o símbolo que os distinguem e atribuindo a  $\sigma$ . E enquanto não encontrar o símbolo  $\sigma$  com o valor  $\varepsilon$ , unirá

ao conjunto  $w$  o símbolo  $\sigma$ , em seguida descobre o próximo par de estados que o símbolo  $\sigma$  levará utilizando a tabela de transição, ou seja, o novo par será  $\{\delta(p, \sigma), \delta(q, \sigma)\}$ , e por ultimo descobre qual é o símbolo que distinguiu este próximo par de estados.

Então será retornado o a palavra  $w$ , que é a palavra que distinguiu os dois autômatos que estávamos procurando.

## 3.2 Equivalência em Tempo Linear

A maioria dos algoritmos para teste de equivalência de autômatos finitos determinísticos na literatura tem crescimento assintótico proporcional ao quadrado do número de estados (HOPCROFT; MOTWANI; ULLMAN, 2006). Um algoritmo desenvolvido por Hopcroft e Karp (HOPCROFT; KARP, 1971a), para este mesmo fim, tem complexidade de  $O(n \log^* n)$ , onde  $n$  é o número de estados. Nesta seção estudaremos este algoritmo.

Mas antes temos que entender a estrutura de dados utilizada para armazenar conjuntos disjuntos, conhecida popularmente como *Union-Find*, que será fundamental para o método de equivalência de Hopcroft e Karp.

### 3.2.1 A Estrutura Union-Find

Para o método que veremos na seção seguinte, envolve o agrupamento de elementos em uma coleção de conjuntos disjuntos<sup>1</sup>, e há uma estrutura de dados que atende este requisito (CORMEN et al., 2001) e admite duas operações que será vital para a implementação do algoritmo de equivalência em tempo linear.

Na estrutura de dados de conjuntos disjuntos guarda-se um coleção de elementos disjuntos. Onde cada conjunto é identificado por um representante, que será um de seus elementos, não importando na maioria dos casos qual seja este elemento, somente que será sempre ele que atenderá quando for procurado o representante daquele conjunto.

Para se criar um conjunto utiliza-se a função **MAKE-SET**( $x$ ), que aloca um conjunto contendo um elemento  $x$  na memória, este elemento será o representante deste conjunto, e claro  $x$  não deverá estar presente em nenhum outro conjunto.

A operação **FIND**( $x$ ) basicamente tem a função de dado um elemento  $x$ , encontrar a qual conjunto este elemento pertence, retornando o seu elemento representante.

A outra operação é o **MERGE**( $x, y$ ), que tem a função de fundir os dois conjuntos que os elementos  $x$  e  $y$  pertencem em somente um na coleção.

Há mais de um modo de implementar esta estrutura, usando listas ligadas ou até lista de árvores, por exemplo.

<sup>1</sup> São quando os conjuntos não tem elementos em comum, ou seja, suas interseções é o conjunto vazio.

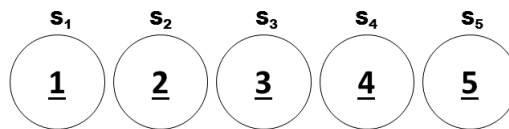
Com a lista ligada, temos o primeiro elemento de cada lista sendo o representante, e cada objeto desta lista tem um ponteiro para o próximo objeto e um ponteiro direto para o representante. Deste modo facilita a operação **FIND** por ter um ponteiro direto para o representante. Mas para o **MERGE**, não basta somente o ultimo objeto do primeiro conjunto apontar para o inicio do segundo conjunto, há a necessidade de atualizar o ponteiro para o novo representante de todos os elementos do segundo conjunto. Causando um aumento na complexidade do algoritmo.

Já se utilizarmos uma estrutura de árvore, em que cada elemento será um objeto que representa um nó na árvore. Então este nó somente basta ter um ponteiro para seu nó pai, e o representante será aquele que apontar para ele mesmo. A operação **MERGE** se torna mais simples, mas o ponteiro da raiz do segundo apontar para a raiz do primeiro conjunto. Já o **FIND** terá que percorrer a partir do pais deste elemento, todos os pais de seus ancestrais, até encontrar a raiz, que é o representante deste conjunto. Isso pode se tornar custoso caso a altura da árvore seja muito grande, por isso uma otimização da estrutura se faz necessária, utilizando um método de compactação do caminho até a raiz.

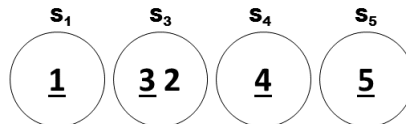
Cada implementação tem a sua particularidade e a escolha depende de seu uso, para o Algoritmo de Hopcroft e Karp, será implementada como modo de árvores, que discutiremos e analisaremos melhor nas seções seguintes.

Para ilustrar melhor esta estrutura, temos o seguinte exemplo:

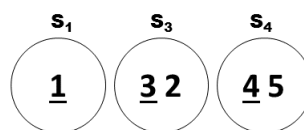
- Criamos uma coleção de conjuntos disjuntos  $\Psi = \{S_1, S_2, S_3, S_4, S_5\}$ , onde para cada conjunto  $S_i$  temos o elemento  $i$  contido nele e este será o representante do conjunto disjunto. Criando  $\{\{\underline{1}\}, \{\underline{2}\}, \{\underline{3}\}, \{\underline{4}\}, \{\underline{5}\}\}$ . Os elementos sublinhados caracteriza o representantes daquele conjunto.



- **MERGE**(2, 3) resulta em  $\{\{\underline{1}\}, \{2, \underline{3}\}, \{\underline{4}\}, \{\underline{5}\}\}$ .



- **MERGE**(5, 4) resultado em  $\{\{\underline{1}\}, \{2, \underline{3}\}, \{\underline{4}, 5\}\}$ .



- **FIND**(2) será retornado o elemento 3.
- **FIND**(3) será retornado o elemento 3.



### 3.2.2 Algoritmo de Hopcroft e Karp

Nesta seção estudaremos o algoritmo de Hopcroft e Karp para o teste de equivalência em tempo linear (HOPCROFT; KARP, 1971a). O algoritmo utiliza-se da estrutura de dados pilha e uma estrutura de dados popularizada como Union-Find para armazenar conjuntos disjuntos (CORMEN et al., 2001). Em especial esta última estrutura de dados tem fundamental importância no algoritmo para testar a equivalência dos autômatos, pois armazenará no mesmo conjunto os estados que são equivalentes.

Vejamos a seguir o algoritmo.

---

**Algoritmo 3:** Teste de Equivalência de Hopcroft e Karp

---

**Entrada:**  $Q, \delta, \{p_0, q_0\}$

**Saída:** Conjuntos disjuntos

```

1 início
2   conjuntos.INIT(|Q|)
3   conjuntos.MERGE( $p_0, q_0$ )
4   pilha.empilha( $\{p_0, q_0\}$ )
5   enquanto pilha  $\neq \emptyset$  faça
6      $\{p, q\} \leftarrow$  pilha.desempilha()
7     para cada  $\sigma \in \Sigma$  faça
8        $r \leftarrow$  conjuntos.FIND( $\delta(p, \sigma)$ )
9        $s \leftarrow$  conjuntos.FIND( $\delta(q, \sigma)$ )
10      se  $r \neq s$  então
11        conjuntos.MERGE( $r, s$ )
12        pilha.empilha( $\{r, s\}$ )
13  retorna conjuntos

```

---

A estrutura de dados Union-Find armazena conjuntos disjuntos na memória, e deseja-se executar essencialmente duas operações:  $\text{MERGE}(i, j)$ , une o conjunto  $i$  ao conjunto  $j$ , e  $\text{FIND}(i)$  busca o conjunto ao qual pertence o elemento  $i$ .

O algoritmo inicia-se criando a estrutura de dados Union-Find, que atribuímos o nome de *conjuntos*, e terá o tamanho igual ao tamanho do conjunto  $Q$ , que é novamente a união dos dois autômatos que queremos testar sua equivalência. Assim a estrutura conterá  $|Q|$  conjuntos e em cada um contendo um único elemento que corresponde a um estado de  $Q$ , e cada conjunto é denominado pelo nome do elemento que é o representante daquele conjunto. Ou seja, cada estado de  $Q$  está em seu próprio conjunto, sendo ele mesmo o representante e nome deste conjunto.

Realizaremos a primeira fusão, entre os conjunto  $p_0$  e  $q_0$ , pois supomos que se os autômatos são equivalentes, então seus estados iniciais também deverão ser equivalentes.

EM seguida utilizaremos uma estrutura de dados de Pilha para armazenar o par destes dois estado iniciais.

Assim inicia-se as verificações até que esta pilha esteja vazia. Primeiro desempilhamos um par da pilha e atribuímos a variáveis  $\{p, q\}$ . Então para cada simbolo  $\sigma$  do alfabeto  $\Sigma$  será feito algumas etapas. Buscamos qual é o conjunto que contém o estado que foi levado pela função de transição  $\delta$  ao entrar com o simbolo  $\sigma$  e denominaremos estes de  $r$  e  $s$ . E verificaremos se  $r \neq s$ , ou seja, se o conjunto  $r$  é diferente do conjunto  $s$ . Se sim, então uniremos estes dois conjuntos e empilhamos o par  $\{r, s\}$  na pilha.

Assim o algoritmo somente terminará ao analisar todos os estados para todas as letras do alfabeto, retornando ao final o *conjuntos*.

Então para constatar a equivalência, basta examinar em cada conjunto  $q$  da lista de conjuntos disjuntos se  $q \subset F$  ou  $q \subset Q \setminus F$ . Ou seja, os dois autômatos são equivalentes se, e somente se em nenhum conjunto da lista há estados de aceitação e de não aceitação no mesmo conjunto.

Os passos das linhas 2, 3, 4 e a verificação final desta lista, são executadas em uma quantidade limitada por uma constante de tempo  $n$ , onde  $n$  é o número de estados de  $Q$ .

Já o tempo de execução da interações da linha 5 até a 12, tem o tempo limitado pelo número de vezes que os pares são removidos da pilha. Que por sua vez, o número de pares retirados da pilha é limitado por  $n$ , pois cada vez que um par é inserido na pilha, dois conjuntos são fundidos, assim o número total de conjuntos é diminuído em um. Desde a inicialização, são definidos  $n$  conjuntos e no máximo  $n - 1$  pares são colocados na pilha.

Então o tempo de execução deste algoritmo para testar a equivalência de dois autômatos finitos é delimitados por uma constante vezes o produto do tamanho do alfabeto pelo o número de estados, ou seja,  $O(c \times |\Sigma| \times |Q|)$ .

Mas como utilizamos a estrutura de dados Union-Find temos o consumo de tempo limitados superiormente por  $O(n \log^* n)$ .

### 3.2.3 Detalhes de Implementação do Algoritmo de Teste de Equivalência

O algoritmo utiliza-se de duas estruturas de dados, uma pilha para armazenar o próximo par a ser analisado e o Union-Find para armazenar conjuntos disjuntos.

A pilha utilizada na implementação é a oferecida pela biblioteca padrão da linguagem. Já o conjunto disjunto pode ser simplificada, pois sua utilização neste algoritmo, a estrutura de dados não necessita ter alocação de memória dinâmica. Logo, para sua implementação basta a alocação de um único vetor sequencial de tamanho igual ao passado por parâmetro no método INIT. Para isto, os atributos padrão desta estrutura de dados serão simplificados, provendo assim uma economia de memória.

Cada índice deste vetor, representará um elemento a ser agrupado em conjuntos, que por sua vez representa um estado da união dos dois autômatos. O valor daquele elemento no vetor representará o seu pai. E quando este valor apontar para si mesmo, significa que este elemento é o elemento raiz, também chamados de representante do conjunto que ele está contido. O atributo *rank* não se fará necessário para esta implementação simplificada. Esta estrutura se assemelhará a uma árvore, em que cada elemento há somente o ponteiro para seu pai.

Assim com esta estrutura, podemos implementar somente os métodos necessários para manipular os conjuntos disjuntos que o algoritmo necessita.

Os passos para a implementação da função MERGE serão as seguintes: recebido dois índices por parâmetro, buscamos os conjuntos que os contêm com auxílio da função FIND e por fim fazemos o primeiro conjunto ser subconjunto do segundo, ou seja, a raiz do primeiro terá seu valor apontando para a raiz do segundo conjunto.

Já o método FIND, recebe um elemento por parâmetro, e retorna o índice do elemento raiz (o representante) do conjunto que contém o elemento passado como parâmetro.

Para testar a equivalência de dois autômatos, inicialmente supomos que eles são equivalentes, logo os seus estados iniciais são equivalentes. Por conta disto acontece a primeira chamada do método MERGE para o par  $\{p_0, q_0\}$  correspondente ao par de estados iniciais dos dois autômatos. E assim inicia-se as verificações até que não haja mais conjuntos a serem unidos.

Há ainda a possibilidade de otimizar este código, pois ao final da execução do algoritmo, temos que verificar todos os conjuntos resultantes da estrutura de dados Union-Find. Se durante a execução do Algoritmo 3 for haver uma união de conjuntos, basta verificarmos cada um dos representantes se um é estado de aceitação e o outro não. Então comprovando que aqueles dois autômatos não são equivalentes, não necessitando terminar todas as interações para verificar todos os conjuntos somente ao final. Então se encontrado uma união heterogênea, o algoritmo deve cessar. E acontecerá esta verificação anteriormente as linhas 3 e 11 do Algoritmo 3 nos conjuntos que pretendem ser unidos.

### 3.2.4 Recuperação de Palavra

Para poder encontrar uma palavra que demonstre a distinção dos dois autômatos, a partir da otimização proposta na seção anterior, facilmente percebemos que, se o algoritmo for terminado por tentar unir dois conjuntos heterogêneos, então existe uma palavra que os distingue. Basta agora recuperar esta palavra, e para isto temos que recuperar todas as uniões que foram efetuadas até o momento do término.

Uma possível solução, que foi implementada, é que a cada união de conjuntos devemos guardar em uma nova pilha as informações a respeito desta união. Então armaze-

namos a tupla  $\{p, q, \sigma, r, s\}$ , que basicamente são as variáveis daquela interação, onde  $p$  e  $q$  é o par que estava sendo verificado pelo simbolo  $\sigma$ , que causou a união do par  $r$  e  $s$ .

Então no Algoritmo 3, basta acrescentar a instrução de empilhar esta tupla na pilha, que chamaremos de  $pHist$ , após as linhas: 3 e 11.

No pseudo código a seguir veremos a implementação da recuperação desta palavra.

---

**Algoritmo 4:** Recuperação da palavra que os distinguem

---

**Entrada:**  $pHist$

**Saída:**  $w$

```

1 início
2    $w \leftarrow \varepsilon$ 
3   enquanto  $pHist \neq \emptyset$  faça
4      $\{p, q, \sigma, r, s\} \leftarrow pHist.desempilha()$ 
5      $w \leftarrow \sigma \cdot w$ 
6     enquanto  $(pHist.topo().r \neq p) \mid (pHist.topo().s \neq q)$  faça
7        $pHist.desempilha()$ 
8   retorna  $w$ 

```

---

Assim teremos uma pilha com o histórico das uniões. Então se o Algoritmo 3 terminou por ter encontrado uma união não desejada, basta agora ir retirando os elementos desta pilha histórico até esvaziar, adicionando à variável  $w$  o simbolo que causou a união. E por fim encontrar o par de estados que uniu este par sucessor, dispensando os elementos que não estamos procurando.

Poderíamos abstrair melhor este algoritmo como se a cada verificação a partir da raiz ( $\{p_0, q_0\}$ ) no Algoritmo 3 fosse uma busca em profundidade, que se interrompida em algum ponto, então o caminho de volta até a raiz a partir deste ponto, recolhendo os símbolos associada a cada união, forma a palavra que distinguiu os dois autômatos.

### 3.3 Escolha do Melhor Método

Após implementar e analisar cada algoritmo que discutimos até aqui, devemos escolher qual deles deve ser incluído no sistema. Ambos cumprem a tarefa de verificar a equivalência entre dois autômatos, e ambos são capazes de recuperar a palavra que diferencia um autômato do outro, no caso de eles não serem equivalentes.

Claramente o algoritmo de teste de equivalência de Hopcroft e Karp se destaca comparado ao método de minimização de Moore, pelo fato dele ter complexidade linear, o que possibilita atingir um maior desempenho, que é uma característica desejável para o sistema.

## 4 O sistema

A plataforma que desenvolvemos possibilita uma experiência de uso semelhante ao de um aplicativo para *desktop*, mas não necessita de instaladores e não depende de um sistema operacional específico como os programas tradicionais.

Uma aplicação web é um sistema que possibilita a intercomunicação entre um cliente e um servidor via internet. O cliente é um navegador, que exibe uma interface gráfica e proporciona uma interação com o usuário. Já o processamento fica ao encargo de um servidor HTTP<sup>1</sup>. Este servidor tem a função de receber requisições e devolver respostas ao cliente. E o cliente exibe as informações ao usuário e solicita recursos a este servidor conforme o usuário interage com o navegador.

Assim este sistema se torna muito prático e acessível para o aluno poder praticar o conceito aprendido em aula. Então, o objetivo deste projeto é que, por meio de um navegador de internet, o aluno responda a questões e obtenha um retorno praticamente imediato do servidor que analisará sua resposta. Por este motivo foi utilizada a plataforma web para desenvolver este sistema.

No próximo capítulo será documentado todo o processo de desenvolvimento deste sistema. Discutiremos os requisitos e analisaremos as melhores estratégias para projetar e implementar este sistema de apoio na aprendizagem da disciplina de Linguagens Formais e Autômatos. Também discutiremos o conceito do juiz, que tem um papel fundamental neste sistema: o de julgar a equivalência entre autômato-resposta do aluno e o autômato-gabarito, utilizando o algoritmo anteriormente escolhido.

Mas, primeiramente, é necessário viabilizar um requisito de extrema importância para este projeto, de como será coletado o autômato-resposta do aluno.

### 4.1 Interface de entrada da resposta

Inicialmente, os exercícios que o sistema vai atender têm como resposta um autômato finito determinístico. Então, há a necessidade do uso de algum tipo de formulário, ou método, que receba esta resposta. Este autômato-resposta poderia ser descrito matematicamente, como definimos anteriormente, mas não seria dinâmico nem intuitivo para o aluno.

Então, a adoção de um método intuitivo e com interação gráfica se torna um requisito essencial deste sistema, e acarreta no aumento da complexidade deste projeto.

---

<sup>1</sup> Abreviação para *Hypertext Transfer Protocol*, em português Protocolo de Transferência de Hipertexto, é um protocolo base para a comunicação de dados da *World Wide Web*.

Um Autômato Finito Determinístico, que também pode ser chamado de Máquina de Estado Finito Determinístico, é uma máquina de estados que aceita ou rejeita cadeias de símbolos. E cada cadeia, ao ser processada, faz com que o autômato percorra uma sequência de passos determinadas unicamente pela palavra lida.

Esta máquina de estados pode ser representada por um diagrama de estados, constituída por círculos com borda duplas ou simples que correspondem, respectivamente, aos estados de aceitação ou de não aceitação. Um indicador para o estado inicial. E as regras de transições são retratadas como ligações direcionais entre dois estados quaisquer, ou seja, setas com um simbolo associado ligando dois estados.

Após uma pequena pesquisa na internet, encontramos a *Finite State Machine Designer*, uma ferramenta para construção de máquinas de estados finitos desenvolvida por Evan Wallace (WALLACE, 2010), como vemos abaixo. Ela tem como objetivo facilitar a confecção de diagramas de máquinas de estados e a possibilidade de exportar o diagrama criado para diversos formatos, proporcionando uma grande praticidade.

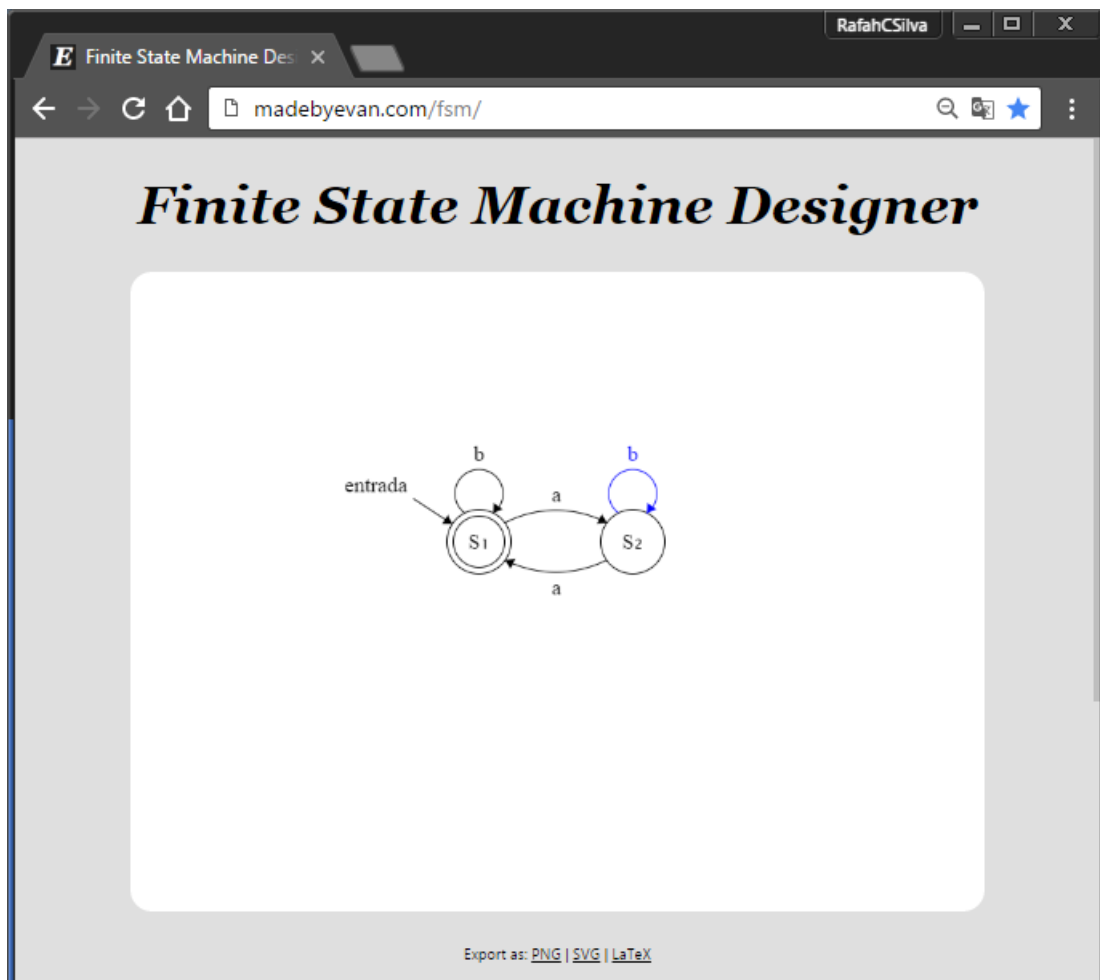


Figura 3 – Construtor de máquinas de estados.

Esta ferramenta utiliza o elemento *CANVAS*<sup>2</sup> do *HTML5*<sup>3</sup>, onde objetos são desenhados por meio de um *JavaScript*<sup>4</sup>. Então, como esta ferramenta requer somente um navegador que suporte o elemento *CANVAS*, ela é a ferramenta perfeita para ser utilizada na hora que os usuários do sistema (principalmente os alunos) precisarem fornecer o diagrama de um autômato. Como esta ferramenta tem licença MIT (MIT, 1988), qualquer pessoa pode usar, copiar, modificar, mesclar, publicar, distribuir, sublicenciar e vender como bem entender.

Esta ferramenta tem estas características e pode ser usada para:

- Criar estados com duplo clique em uma área em branco;
- Criar transições com a tecla *shift* e o clicar e arrastar do *mouse*;
- Arrastar um objeto com o clicar e arrastar do *mouse*;
- Deletar um objeto selecionado clicando a tecla *delete*;
- Tornar um estado em aceitação, ou não, com um duplo clique sobre ele;
- Subscrever números com “\_”, do seguinte modo “ $S_0$ ” para obter “ $S_0$ ”;
- Escrever letras gregas do seguinte modo “\beta” para obter “ $\beta$ ”;
- Exportar para PNG, SVG e  $\text{\LaTeX}$  a máquina de estados;
- Manipular os objetos desta máquina de estados em memória;
- Desenhar os objetos no elemento *CANVAS* do *HTML5*;
- Salvar e recuperar automaticamente o diagrama no armazenamento local do navegador;

O código fonte está disponível no repositório pessoal do Evan Wallace<sup>5</sup>. Para este projeto precisaremos modificar o código para atender alguns requisitos do sistema. Dentre as funções destacadas acima, somente as funções de exportar, de salvamento e recuperação automática, de inserção de letras gregas e subscrito não nos interessa inicialmente. E, para este sistema, foi necessário a adição de funções extras, tais como: a verificação da correta confecção de um autômato finito determinístico e a extração de sua descrição matemática que para ser enviada ao juiz.

Desenvolvemos assim uma especialização da ferramenta *Finite State Machine Designer*, que possibilita a construção e verificação de autômatos finitos determinísticos, a qual demos o nome de **DFAdesigner**. Abreviação de *Deterministic Finite Automaton Designer*, ou em português, Construtor de Autômato Finito Determinístico.

<sup>2</sup> Um elemento do HTML para exibir gráficos em uma página web.

<sup>3</sup> HTML5 (*Hypertext Markup Language*, versão 5) é uma linguagem para estruturação e apresentação de conteúdo para a web.

<sup>4</sup> Linguagem de programação para páginas HTML e web, executada pelo navegador do cliente.

<sup>5</sup> Disponível em: <<https://github.com/evanw/fsm>>.

## 5 Desenvolvendo o sistema

Neste capítulo são documentados as etapas do desenvolvimento do sistema web. Definimos os objetivos e requisitos a partir do contexto, para assim analisar e projetar este sistema, para assim, finalmente, implementa-lo e testa-lo.

O sistema pode ser acessado por dois tipos de usuários: Aluno e Professor, que desempenham papéis diferentes no sistema. O aluno é o discente matriculado na disciplina e o professor é o docente que ministra aquela turma.

O objetivo principal deste sistema é a aplicação de listas de exercícios aos alunos de uma turma. E desta maneira, o sistema deve possibilitar também todo o gerenciamento destas aplicações.

Um exercício nada mais é do que uma questão, constituída de um título, um enunciado e um autômato-gabarito que apresenta uma resposta correta. Questões ordenadas compõem uma lista, cada questão pode estar presente em diversas listas.

Uma turma é formada por alunos e está subordinada a um professor, e nela são aplicadas listas de questões. Na aplicação de uma lista, é definida uma data e hora limite para o envio da solução do aluno. Um aluno pode fazer parte de mais de uma turma (por exemplo, no caso dele vir a refazer a disciplina).

Então o professor deve gerenciar todas as turmas, listas e questões criadas por ele. E também o sistema pode suportar mais de um professor, no caso da disciplina ser ofertada por professores diferentes no mesmo período ou em períodos distintos.

Já a administração de todos os usuários, como alunos ou outros professores, fica ao encargo de todos os professores.

Além de alunos e professores, há também um terceiro componente deste sistema, que denominaremos de Juiz, que tem a função de corrigir instantaneamente uma submissão. Ao longo deste capítulo, será melhor detalhado cada componente e seus relacionamentos.

### 5.1 Modelo de Processo

Para o desenvolvimento deste projeto, foi utilizado o processo de desenvolvimento RAD<sup>1</sup>, que é um modelo de processo de software incremental, que enfatiza um ciclo de desenvolvimento curto. Esse modelo permitiu o desenvolvimento rápido do projeto, que demorou 90 dias. Isso só foi possível porque houve muita reutilização de componentes no projeto, o que garantiu uma padronização tanto na aparência final do sistema, quanto da

---

<sup>1</sup> Abreviação de *Rapid Application Development*, em português, Desenvolvimento de Aplicação Rápida.



implementação de seus módulos, visto que o gerenciamento das Questões, Listas, Turmas e Usuários é praticamente igual.

RAD também possibilita uma rápida prototipagem do sistema, proporcionando uma maior flexibilidade para projetar e um envolvimento maior do usuário. Assim, após definir os requisitos e restringir o escopo do sistema, pudemos fracionar o projeto em subprojetos. E, para a confecção de cada subprojeto, foram reutilizados componentes e padrões predefinidos foram seguidos, e cada etapa só foi considerada encerrada após as validações do usuário.

Esse modelo de processo possibilita que a construção e implementação do projeto ocorra paralelamente entre diversas equipes, proporcionando uma rápida conclusão, que é uma das grandes vantagens deste modelo. Porém este sistema tem seus principais módulos (Questão, Lista, Turmas e Usuários) dependentes entre si, dificultando este paralelismo. Mas mesmo contando somente com um desenvolvedor, este projeto se beneficiou de todas as outras vantagens que o RAD proporciona.

## 5.2 Requisitos e Modelagem do Sistema

Nesta seção, listaremos os requisitos do sistema e sua modelagem.

### 5.2.1 Requisitos Funcionais

Os Requisitos Funcionais são os itens que o sistema deverá oferecer ou restringir:

- RF01. Acessar o sistema via web e funcionar em navegadores modernos;
- RF02. Garantir acesso somente a usuários previamente cadastrados pelo professor;
- RF03. Ser acessado utilizando um e-mail único e uma senha pessoal;
- RF04. Poder redefinir a senha, caso esquecida, e enviar no e-mail do usuário as informações de como ele deve proceder;
- RF05. Ter dois tipos de usuários: Professor e Aluno;
- RF06. Cadastrar um usuário com RA, nome completo e um e-mail;
- RF07. Enviar e-mail ao usuário explicando como ter acesso ao sistema e definir sua senha pessoal;
- RF08. Uma questão deve ter título e enunciado, que suportem comando  $\text{\LaTeX}$ , e um autômato-gabarito;
- RF09. Uma lista deve ter título e conter questões em uma ordem definida;
- RF10. Uma turma deve ser formada por alunos e nela deve ser aplicado listas com datas limites de submissão de respostas;

- RF11. Adicionar Alunos em lote em uma turma;
- RF12. O professor será o administrador, possibilitando:
  - RF12.1. Cadastrar, editar e remover usuários;
  - RF12.2. Cadastrar, editar e remover questões;
  - RF12.3. Cadastrar, editar e remover listas;
  - RF12.4. Cadastrar, editar e remover turmas;
  - RF12.5. Gerar relatório de desempenho de uma turma, e exportar em compatibilidade com editores de planilhas eletrônicas;
- RF13. O aluno somente deve poder:
  - RF13.1. Visualizar as turmas em que ele está incluso;
  - RF13.2. Visualizar as listas aplicadas à sua turma;
  - RF13.3. Visualizar as questões contidas na lista;
  - RF13.4. Submeter um autômato-resposta de uma questão;
  - RF13.5. Salvar um autômato-resposta como rascunho;
  - RF13.6. Visualizar os veredictos do juiz;
  - RF13.7. Responder à lista de questões se esta não tiver o prazo de entrega expirado;

### 5.2.2 Requisitos Não-Funcionais

Os Requisitos Não-Funcionais são critérios que qualificam os Requisitos Funcionais:

- RNF01. Garantir a autenticação do usuário para todas suas ações;
- RNF02. Armazenar as senhas com criptografia;
- RNF03. Possibilitar a utilização do sistema com rápido tempo de treinamento;
- RNF04. Corrigir com rapidez e confiabilidade;
- RNF05. Garantir a integridade dos dados;
- RNF06. Garantir a disponibilidade;
- RNF07. Exigir o mínimo de recursos como espaço e CPU, a fim de baratear o custo do servidor;
- RNF08. Seguir um padrão estético em todas as interfaces;
- RNF09. Sanitizar todas as entradas de dados pelos usuários;
- RNF10. Usar um conexão segura com o banco de dados;

### 5.2.3 Modelos Funcionais

Um modelo funcional serve para representar graficamente as funções relacionadas às regras de negócio, dentro de um contexto definido. Assim é possível modelar e descrever os processos realizados por cada componente.

Para descrever as funções do sistema, foram utilizados os seguintes modelos funcionais: Diagrama de Caso de Uso e Diagrama de Fluxo de Dados.

O diagrama de caso de uso abaixo mostra os atores do sistema e as funcionalidades associadas à eles.

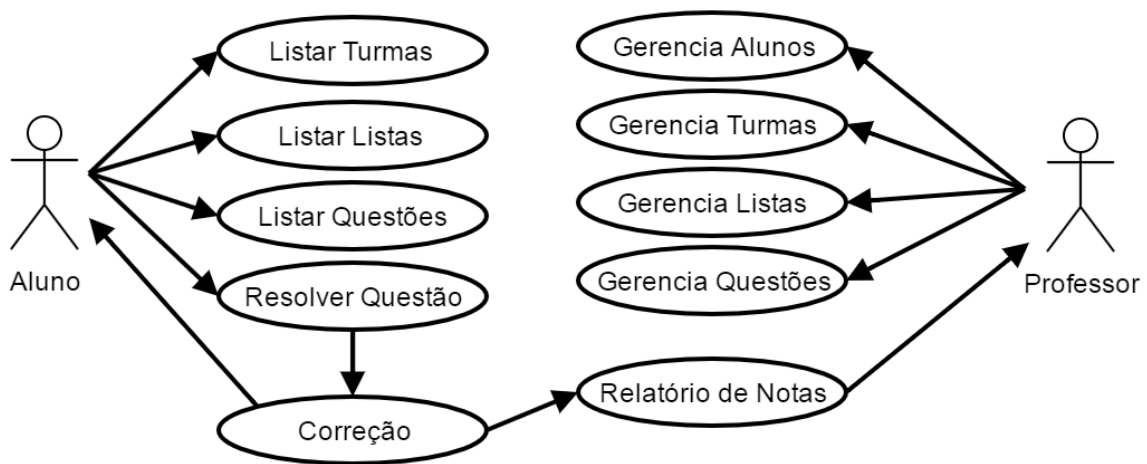


Figura 4 – Diagrama de Caso de Uso.

Para esclarecer como o sistema funciona e mostrar como as informações no sistema fluem, o Diagrama de Fluxo de Dados é apresentado a seguir.

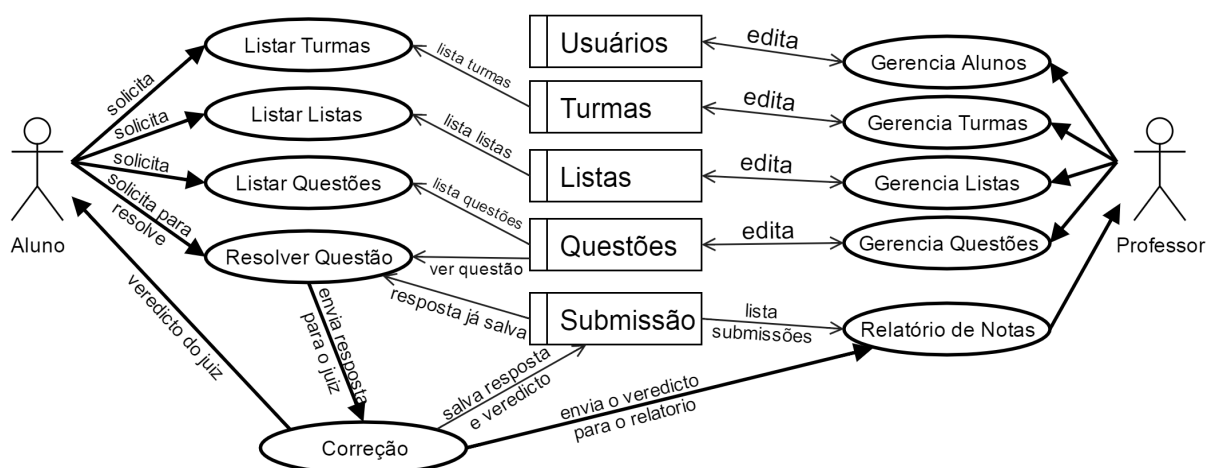


Figura 5 – Diagrama de Fluxo de Dados.

### 5.2.4 Modelos de Dados

Um modelo de dados permite apresentar como foram elaboradas as estruturas dos dados que dão suporte aos processos do sistema, mostrando como os dados são organizados e quais seus relacionamentos. Neste sistema podemos claramente observar que temos 4 entidades, e seus atributos e relacionamentos pode ser visto a seguir no Diagrama de Entidade Relacionamento.

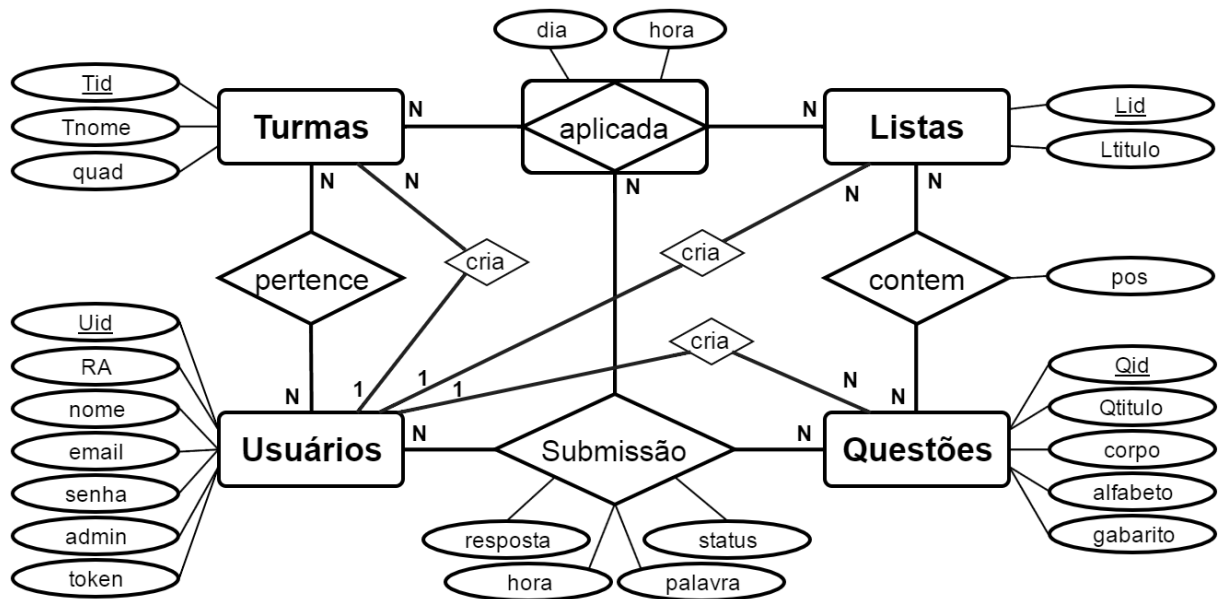


Figura 6 – Diagrama de Entidade Relacionamento.

Este diagrama mostra as relações que determinam a dependência entre as entidades, e também possibilita que o professor gerencie as questões, as listas e as turmas por ele criadas, e também permite que os alunos visualizem os dados conforme as suas relações.

E uma submissão do aluno é a relação entre uma questão de uma lista aplicada na sua turma, na qual armazena seu autômato-resposta e o veredicto do juiz, que é o status da submissão, juntamente com a palavra que o distinguiu, caso necessário.

## 5.3 Análise

Nesta seção vamos analisar os requisitos, afim de tomar as decisões necessárias para seu desenvolvimento.

Como o requisito primário deste sistema é ser um sistema web, então fundamentalmente o sistema foi projetado para esta plataforma. Assim tivemos de tomar algumas decisões, como em qual linguagem o sistema seria implementado e quais recursos estariam disponíveis para ele. E também necessitamos de um banco de dados para o armazenamento de todas as informações que o modelo prevê.

Com o objetivo de baratear o custo para sua execução e garantir sua disponibilidade, o sistema não deve demandar de requisitos caros. Logo, foi escolhida a linguagem *PHP*<sup>2</sup> para a implementação do sistema, pois esta linguagem oferece todas as condições necessárias para sua implementação, e os servidores comerciais oferecem planos mais acessível para suportar o PHP, e juntamente com ele o Sistema de Gerenciamento de Banco de Dados *MySQL*<sup>3</sup> para armazenamos os dados. E assim o sistema construirá páginas para responder às requisições do usuário.

O uso de bibliotecas e *frameworks* inicialmente não foram consideradas, ficando ao encargo do desenvolvedor toda sua codificação, por escolha própria dele, já que ele possui familiaridade com o desenvolvimento web com a linguagem *PHP*, afim também de se aprimorar como um *full stack web developer*<sup>4</sup>.

Não foi possível obter um servidor dedicado do CMCC<sup>5</sup>, assim o professor orientador deste projeto se dispôs a oferecer seu servidor comercial e endereço por ele já contratado, para ser utilizado durante o desenvolvimento e testes deste sistema, na qual não foi possível utilizar o padrão HTTPS<sup>6</sup>, mas isto irá prejudicar em nada o desempenho final do sistema.

O passo seguinte é criar um Modelo Relacional, a partir dos diagramas e modelos de dados vistos anteriormente. O Modelo Relacional permite a criação de um modelo lógico consistente da informação a ser armazenada. Este também pode ser refinado ao passar por um processo de normalização.

Um Objeto de Dados, ou como chamamos anteriormente de Entidade, é a representação de um componente que contém propriedades. Estas propriedades chamamos de Atributos, que qualificam uma entidade e a distingue das outras. E o valor do atributo está definido em um domínio, ou tipo de dado. O Relacionamento é uma ligação lógica entre Entidades, representando a regra de negócio.

Assim, este modelo relacional representa todas as regras de negócio deste sistema, mas adequado a um sistema de gerenciamento de banco de dados. Em que cada tabela é uma entidade, que contem atributos. Cada atributo tem seu tipo de dado especificado. E a ligação de um atributos à outras tabelas, representa um relacionamento, e a cardinalidade indica a quantidade de ocorrências deste relacionamento.

<sup>2</sup> PHP (*PHP: Hypertext Preprocessor*) é uma linguagem de *script open source*, muito utilizada e especialmente adequada para o desenvolvimento web.

<sup>3</sup> O MySQL é um sistema de gerenciamento de banco de dados.

<sup>4</sup> Perfil de desenvolvedor que lida com múltiplas camadas de desenvolvimento e tecnologias envolvidas, desde o *back-end* até o *front-end*, ou seja, do servidor ao cliente.

<sup>5</sup> Centro de Matemática Computação e Cognição da UFABC.

<sup>6</sup> Abreviação de *Hyper Text Transfer Protocol Secure*, em português, Protocolo de Transferência de Hipertexto Seguro, é uma implementação do protocolo HTTP sobre uma camada adicional de segurança.

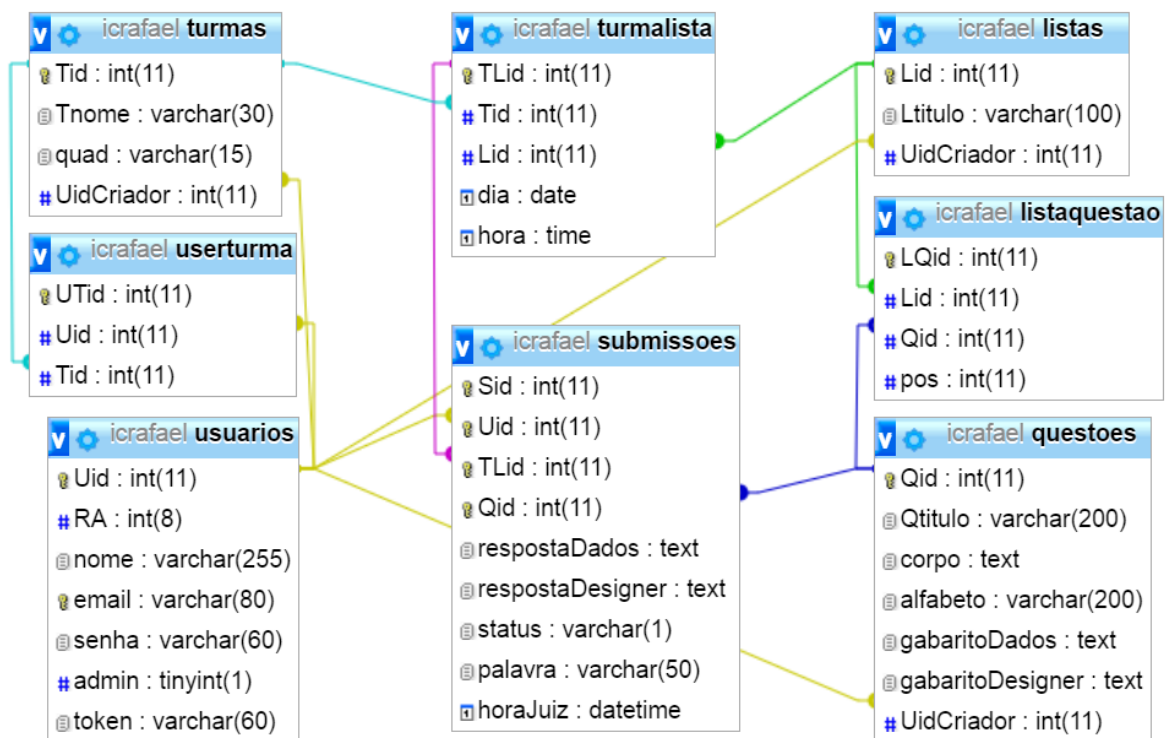


Figura 7 – Modelo Relacional.

O modelo Relacional deste sistema visto acima, na prática, gera as tabelas de um banco de dados relacional. As tabelas usuários, turmas, listas e questões representam as entidades vistas no diagrama da figura 6. E as demais tabelas são utilizadas para apoiar os relacionamentos entre as entidades, geradas após a aplicação das devidas regras de normalização, garantindo assim a integridade dos dados a serem armazenados, o que possibilita a implementação do banco de dados, com todas as regras de negocio do sistema.

## 5.4 Projeto

Nesta seção, tudo o que analisamos dos requisitos nas seções anteriores será detalhado.

Primeiro foi definido como seria a arquitetura do sistema. Em seguida, fizemos o detalhamento de todas as interfaces e suas interligações. E ao fim, elaboramos o Juiz e o DFAdesigner

### 5.4.1 Estilo Arquitetural

O estilo arquitetural em camadas separa o sistema em diferentes partes, que se integram afim de atender o que foi solicitado. Este estilo permite: uma maior independência funcional entre estas partes, facilita o desenvolvimento, particiona problemas complexos

em sequencias de passos, facilita testes de integração e eventuais correções, e permite que um mesmo componente seja usado em diferentes propósitos.

Neste padrão arquitetural de múltiplas camadas, cada camada tem responsabilidades específicas. O mais conhecido desde padrão é o MVC<sup>7</sup>, que separa a lógica do negócio, o modelo de dados e a interface gráfica.

Assim a arquitetura deste sistema foi organizada em camadas, pois oferece maior flexibilidade, padronização e reuso de componentes. Podemos ver a baixo um esquema de como as camadas se integram.

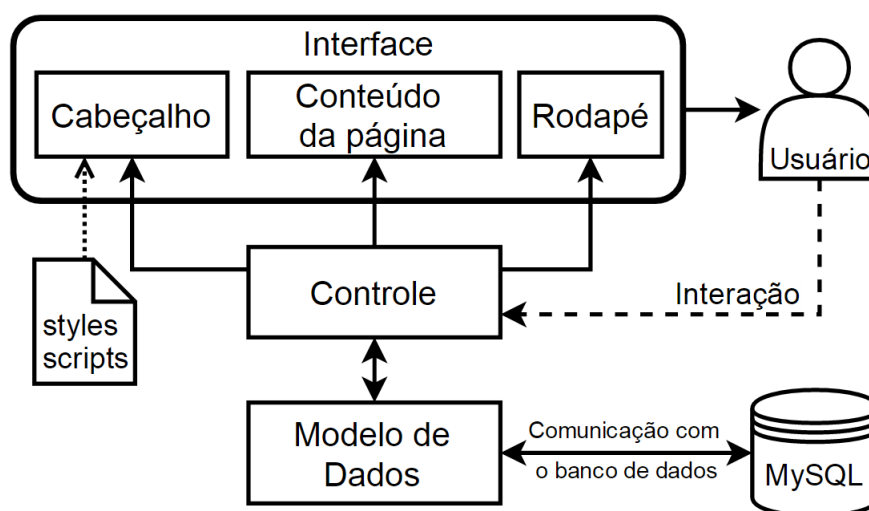


Figura 8 – Diagrama de Camadas.

Quando usuário requisitar algo, o *script* PHP receberá como parâmetro a ação e os dados necessários para poder atender o que foi solicitado. Tomamos como exemplo: o professor requisita a listagem de suas questões. Então a camada Controle recebe a ação de listagem de questões e deveria consultar o banco de dados para obter os dados solicitados, mas esta função é realizada pelo Modelo de Dados. Este se trata de um DAO<sup>8</sup>, ou seja, um objeto que agrupa métodos relacionados ao acesso do banco de dados, como consultas e alterações de dados persistentes. Assim a comunicação direta com o banco de dados é tratada somente em um único lugar, possibilitando a reutilização de métodos em outros componentes. Um método do Modelo de Dados recebe os parâmetros necessários e retorna para o controle os dados ou conclui a ação requisitada. Continuando o exemplo, o Modelo de Dados retornaria a lista das questões daquele professor para o Controle, que agora deve gerar a Interface.

Para formar a interface, o controle irá incluir 3 itens na saída: o cabeçalho, o conteúdo daquela página e o rodapé. O cabeçalho e o rodapé são os mesmos componentes

<sup>7</sup> Abreviação de *Model-View-Control*, em português, Modelo-Visão-Controlador

<sup>8</sup> Abreviação de *Data Access Object*, em português Objeto de Acesso a Dados.

que estarão presentes em todas interfaces. O cabeçalho contém os elementos pré textuais do HTML5, como os elementos: *doctype*, *html*, *head*, *title*, ícones, folhas de estilo e *scripts*. Ele incluirá tudo que é necessário para o propósito daquela interface. Em seguida, ainda no cabeçalho, é iniciada a confecção do corpo da página, incluindo no caso deste sistema, um banner, o menu e uma barra de título com ações que serão apresentados mais a frente.

Já o conteúdo da página é dinâmico em relação aos demais componentes, pois é definido no Controle a partir da ação requisitada, podendo conter formulários ou apresentando dados. No mesmo exemplo, será exibido uma tabela em que cada linha contém informações sobre cada questão e ações referentes a este item. E ao final, o rodapé é incluso, contendo os créditos do desenvolvimento. Um detalhe sobre interface é que o sistema foi projetado para ser acessado somente pelo computador, logo não há a grande necessidade da interface ser responsiva.

Desta forma, o servidor PHP entrega esta interface ao usuário, para que ele interaja e requisite novas ações para o Controle.

## 5.4.2 Interface do Sistema

Garantir uma boa usabilidade e funcionalidade da interface de um sistema pode ser uma tarefa complexa. Uma boa interface deve garantir o rápido acesso a informação ou ação desejada. Deve haver consistência nos elementos da interface, ou seja, um objeto deve estar sempre no mesmo lugar. A interface por si só deve ser autoexplicativa, ou seja, facilmente deve-se entender o objetivo de cada um de seus componentes.

Na Figura 9 da página seguinte, vemos todo o fluxo de acesso aos componentes deste sistema por meio do diagrama “Fluxo de Telas”.

Para entrar no sistema, o usuário deve inserir seu e-mail e uma senha por ele definida, em um pequeno formulário e clicar em login na página de login. Uma vez dentro do sistema, é possível realizar as ações a ele atribuída, como veremos a diante.

Também nesta tela de login, o usuário tem a possibilidade de redefinir a sua senha, caso ele a tenha esquecido. Para requisitar uma nova senha, basta inserir seu e-mail e clicar em enviar. Isto faz com que uma mensagem seja enviada ao e-mail, caso este esteja realmente no sistema, contendo os procedimentos para a definição de uma nova senha. O link enviado por e-mail, direciona o usuário para a tela de Definição de Nova Senha, onde ele pode definir uma nova senha. Este link contém um parâmetro (*token*) criado para aquela requisição específica e é único.

Cada usuário do tipo aluno pode visualizar as turmas em que ele está inscrito. Selecionando uma turma, ele pode visualizar as listas aplicadas a ela, e consultar as respectivas datas limites. Ao adentrar em uma lista, as questões que a compõem serão listadas, e cada uma é acompanhada de uma sinalização que representa o status da mesma.



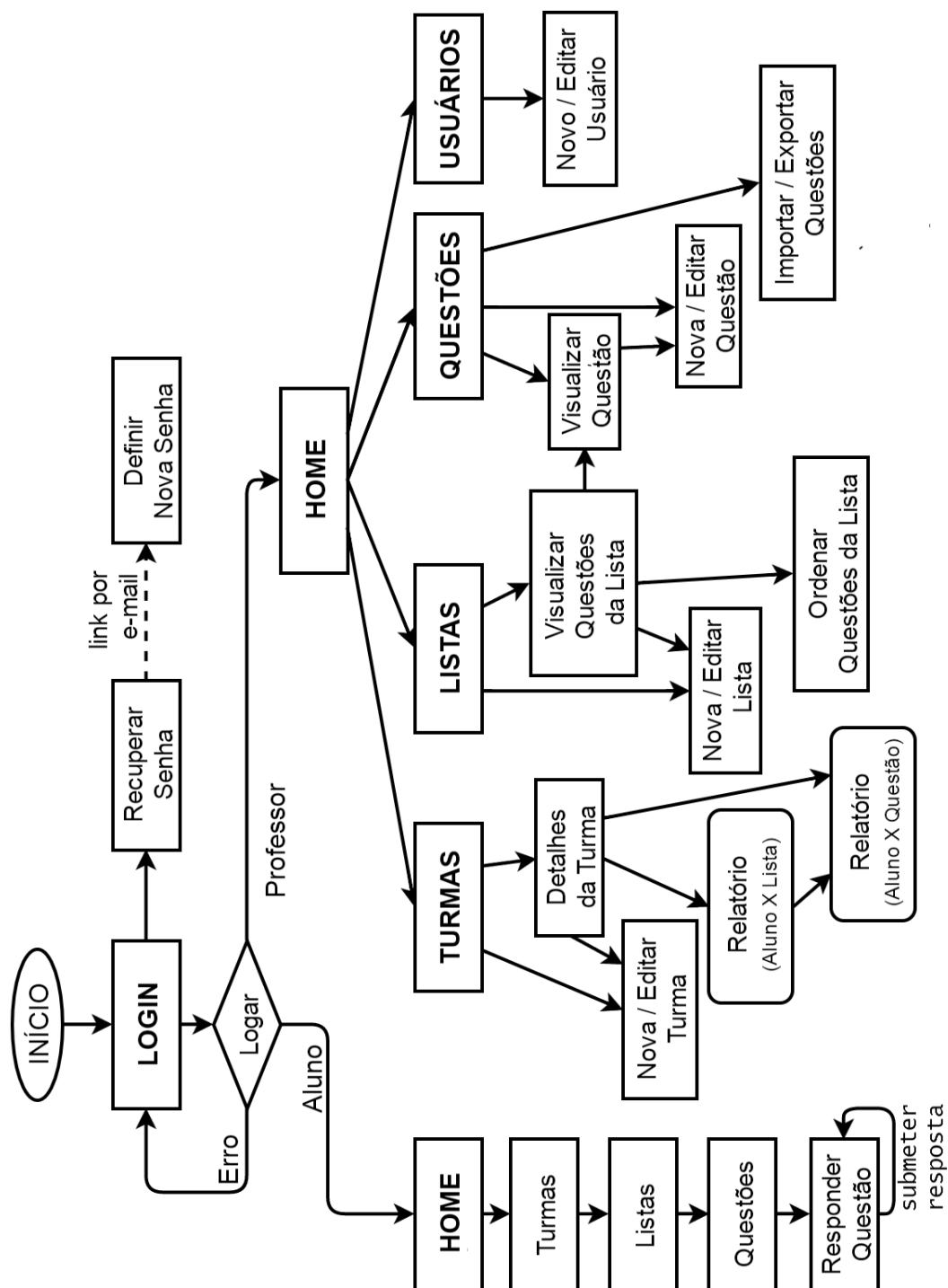


Figura 9 – Diagrama “Fluxo de Telas”.

questão (possivelmente com o veredicto do juiz). Ao clicar em uma questão, o aluno deverá responder à questão em um formulário.

Neste formulário estão inclusos o título e o enunciado da questão, e em seguida o DFAdesigner vazio, preparado para que o aluno desenhe a seu autômato-resposta. E na parte inferior há dois botões de ação: o Salvar e o Submeter. Ao clicar em Salvar, o desenho do aluno é salvo como rascunho, possibilitando a ele retomar a confecção de sua resposta posteriormente. Ao clicar em Submeter, o DFAdesigner analisa o desenho confeccionado, validando se está apto a ser um autômato finito determinístico e o envia ao Juiz. Caso o diagrama tenha problemas, a ação de submeter é abortada e uma mensagem alerta o aluno dos erros cometidos em relação ao desenho.

Com o autômato-resposta corretamente validado, o juiz testa a sua equivalência com o autômato-gabarito daquela questão, e retorna ao aluno, no mesmo formulário de resposta, o seu veredicto: Correto ou Incorreto. Caso o veredicto seja Incorreto, o juiz também retorna a palavra que distinguiu o autômato-resposta do autômato-gabarito.

Já o usuário do tipo professor, ele tem a responsabilidade de gerir todos os objetos envolvidos: usuário, questões, listas e turmas.

Na página de Usuários é possível adicionar novos usuários, editar um usuário já existente e também excluir usuários. Para editar ou adicionar um usuário, basta preencher um formulário com seu RA (no caso do aluno), o nome completo, um e-mail e qual o privilegio daquele usuário (se é administrador ou aluno). Ao adicionar um novo usuário, é enviado a ele um e-mail de boas-vindas, o informando de que foi adicionado ao sistema e contendo instruções para definir uma senha pessoal de acesso.

Na página de Questões, são listadas todas as questões criadas por este professor. Assim, é possível adicionar, editar, remover e visualizar uma questão, e também existe a possibilidade de duplicar uma questão já existente. Esta página também exibe quantas vezes cada questão foi utilizada nas listas do professor. Ao visualizar uma questão, ela é impressa na tela como será visualizada pelo aluno ao responde-la e abaixo o DFAdesigner com desenho do autômato-gabarito, porém bloqueado para edições. Ao adicionar ou editar uma questão, o professor deve preencher um formulário com o título e enunciado da questão, e listar os símbolos do conjunto  $\Sigma$  que compõe a resposta. Logo abaixo, está disponível o DFAdesigner para a confecção ou edição do autômato que representa uma possível resposta correta, ou seja, o autômato-gabarito. Nesse formulário, nos campos de título e enunciado, é possível a utilização de comandos  $\text{\LaTeX}$  e elementos HTML, para uma melhor formatação destes.

Na página de Listas são enumeradas todas as listas criadas por aquele professor, onde é possível visualizar, adicionar, editar ou remover uma lista. Ao visualizar uma lista, são relacionadas todas as questões contidas naquela lista e também existe a possibilidade

de reordena-las. No formulário de edição ou criação de uma nova lista, o professor deve definir um nome para esta lista e, abaixo, são listadas todas as suas questões para que ele possa selecionar quais deverão compor esta lista.

A página de Turmas é mais complexa comparada às anteriores: o professor vai gerenciar as turmas que ele ministra. Então são listadas as suas turmas, e em cada uma delas a que quadrimestre ela está associada, o número de alunos e o número de listas aplicadas. E há possibilidade de visualizar, adicionar, editar e remover uma turma. Ao adicionar ou editar uma turma, o professor deve preencher um formulário com o nome da turma, o quadrimestre, e selecionar os alunos previamente cadastrados. Os alunos também podem ser selecionados em lote por meio de um arquivo CSV<sup>9</sup>, e assim automaticamente adicionando ao sistema alunos que não estão previamente cadastrados. E ao fim escolher dentre suas listas, quais vão ser aplicadas naquela turma, juntamente com a data e hora limite para realizar submissões. Ao visualizar uma turma são listados os alunos nela contidos e as listas aplicadas nela. E nesta mesma página há um redirecionamento para uma outra página em que será gerado e apresentado o relatório daquela turma. O relatório é a listagem de todos os alunos e o número de acertos em cada lista. E há possibilidade também de visualizar para cada lista, qual foi o veredicto do juiz para aquela resposta do aluno. E ao final existe a possibilidade de exportar este relatório como uma planilha no formato CSV, que vai auxiliar este professor a compor a nota final de cada aluno.

Outra página que o sistema disponibiliza é uma seção de ajuda, que contém um tutorial utilizando animações para ensinar o usuário a utilizar o DFAdesigner e contém explicações de outras interações que o aluno pode realizar no sistema. Finalmente há uma página para que o usuário possa visualizar suas próprias informações, possibilitando também a redefinição de sua senha pessoal.

### 5.4.3 O DFAdesigner e o Juiz

Nesta seção, trataremos dos componentes mais importantes deste projeto, que é o método para construir e avaliar uma resposta-autômato. O DFAdesigner e o Juiz estão intimamente ligados, pois o DFAdesigner, além de possibilitar o usuário confeccionar um AFD, ele deve coletar do diagrama os dados necessários para que o Juiz possa avaliar esta resposta com base no gabarito.

Então devemos projetar as modificações necessárias do FSM para utilizá-lo de modo eficiente, pois além de coletar os dados do AFD desenhado pelo usuário, o DFAdesigner deve verificar se o diagrama desenhado condiz de fato com um AFD, para assim enviá-lo para o Juiz. Senão deve retornar mensagens sobre os erros cometidos, que são as seguintes condições em relação a construção deste autômato-resposta:

<sup>9</sup> Abreviação de *Comma-Separated Values*, ou em português, valores separados por vírgula, é um formato de arquivo comumente usados por editores de planilhas eletrônicas.

- Se há estados desenhados;
- Se cada estado está nomeado, senão nomea-los;
- Se há transições com pelo menos um simbolo associado;
- Se há transições com um simbolo inválido;
- Se há transições com símbolos duplicados ou sobrescrevendo outra transição;
- Se há uma transição para cada simbolo a partir de um estado;
- Se há uma e somente uma transição inicial para indicar o estado inicial;

E também adaptar os *scripts* do Evan para que o DFAdesigner possa recuperar do banco de dados o autômato-resposta salvo ou submetido pelo aluno anteriormente.

Após o autômato-resposta ser verificado, o Juiz obtém os dados coletado pelo DFAdesigner e o autômato-gabarito armazenado anteriormente, mas ambos em uma codificação mais concisa, para que, ao receber estes dados, possa converter novamente para uma estrutura que defina estes autômatos para serem utilizados. Estes dados codificados se tratam de uma sequência de número naturais que codifica o autômato da seguinte forma:

- O primeiro número indica a quantidade de estados deste autômato, pois o nome dos estados não nos importa, somente uma enumeração implícita é necessário;
- Depois vem os valores da tabela de transições deste autômato, na qual as linhas estão ordenadas pelo índice dos estados e as colunas conforme os símbolos do conjunto  $\Sigma$  daquela questão;
- Em seguida o índice do representante do estado inicial;
- E, por ultimo, o conjunto de estados que são de aceitação, iniciando pelo número  $m$  de estados de aceitação e, em seguida, uma lista com  $m$  números que correspondem aos índices dos estados de aceitação.

Estes dados são transmitido em uma única linha de números separados por um espaço, pois deste modo facilita o armazenamento e o interpretação do autômato que é representado por esta cadeia de números de forma compacta.

Vejamos um exemplo de como resultaria a codificação do autômato  $M_1$  apresentado na Figura 1 anteriormente:

**2 1 0 0 1 0 1 0**

Figura 10 – Codificação do autômato  $M_1$ .

Podemos observar na Figura 10 a separação desta sequência de números nos exatos 4 blocos que definimos anteriormente. Como  $M_1$  tem dois estados, então o primeiro número

é 2. Os estados de  $M_1$  são enumerados, assim o estado  $S_1$  é o 0 e o estado  $S_2$  é o 1. Depois vem a sua tabela de transições, que é representada pela Tabela 1. Na qual temos dois estados e dois símbolos no conjunto  $\Sigma$ , logo teremos o número de estado multiplicado pelo número de símbolos, que será a quantidade de números desta sequência dos índices dos estados que formam a tabela de transições deste autômato. Em seguida vem o índice do estado inicial, que é o 0. E por fim os estados de aceitação, mas primeiro o número total de estados de aceitação e em seguida os índices destes estados, neste caso temos somente 1 estado de aceitação, e este estado é o de índice 0.

O juiz recebe o autômato-resposta e o autômato-gabarito codificados deste modo, e também o conjunto de símbolos do alfabeto, e os interpretam afim de uni-los numa estrutura de dados em que ele possa finalmente aplicar o Algoritmo 3 e, em seguida, o Algoritmo 4 se necessário. Para ao final retornar seu veredicto.

O veredicto do Juiz é identificado pelo *status* da submissão do aluno, como podemos ver a seguir:

- A** Aguardando o processo de correção do Juiz;
- S** Salvo como rascunho;
- C** Resposta correta;
- I** Resposta incorreta, e acompanha a palavra que os distinguiram;

Deste modo a interface do usuário no navegador, que é o DFAdesigner, se comunica com o Juiz, que está presente no servidor. Então os dados coletados pelo DFAdesigner é salvo no banco de dados na submissão do aluno. Nesta submissão contém esta codificação do autômato e um JSON<sup>10</sup> que representa os objetos da estrutura do desenho do diagrama, que originalmente o Evan implementou para salvar no armazenamento local do navegador, e para obter uma rápida restauração do mesmo. E junto a estes dados da submissão também consta o *status* do Juiz, e a palavra que distinguiu o autômato-resposta do aluno do autômato-gabarito da questão.

Inicialmente, pensamos que arquitetar este Juiz de forma assíncrona às submissões, ou seja, as respostas submetidas pelos alunos aguardarão em uma fila para serem atendidas pelo Juiz posteriormente. Esta é uma estratégia utilizadas por outros sistemas que realizam algum tipo de julgamento, onde realizar esta tarefa necessita de um tempo superior a espera da resposta em um sistema cliente-servidor. Assim, arquitetar nosso sistema para lidar com as submissões de tal modo, ocasionaria o aumento da complexidade do sistema e principalmente de recursos do servidor.

<sup>10</sup> Abreviação de *JavaScript Object Notation*, ou em português, Notação de Objetos JavaScript, é um formato leve de troca de dados com fácil leitura e interpretação tanto de humanos, tanto do computadores.

Então durante o estudo e implementação do Algoritmo 3 e 4 em C++, notamos que a sua execução é de fato muito mais rápida, ao ponto de que em uma requisição, ele possa ser executado e retornar seu veredicto como resposta ao cliente. E oportunamente o algoritmo de Hopcroft e Karp pode ser portado para a linguagem PHP, afim de obter maior integração com o sistema e maior agilidade para responder a requisição.

Assim este sistema deve receber a submissão do aluno e retornar seu veredicto. Mas para melhor explicar como se sucede esta sequencia de processos, veremos o Diagrama de Sequência da submissão a seguir.

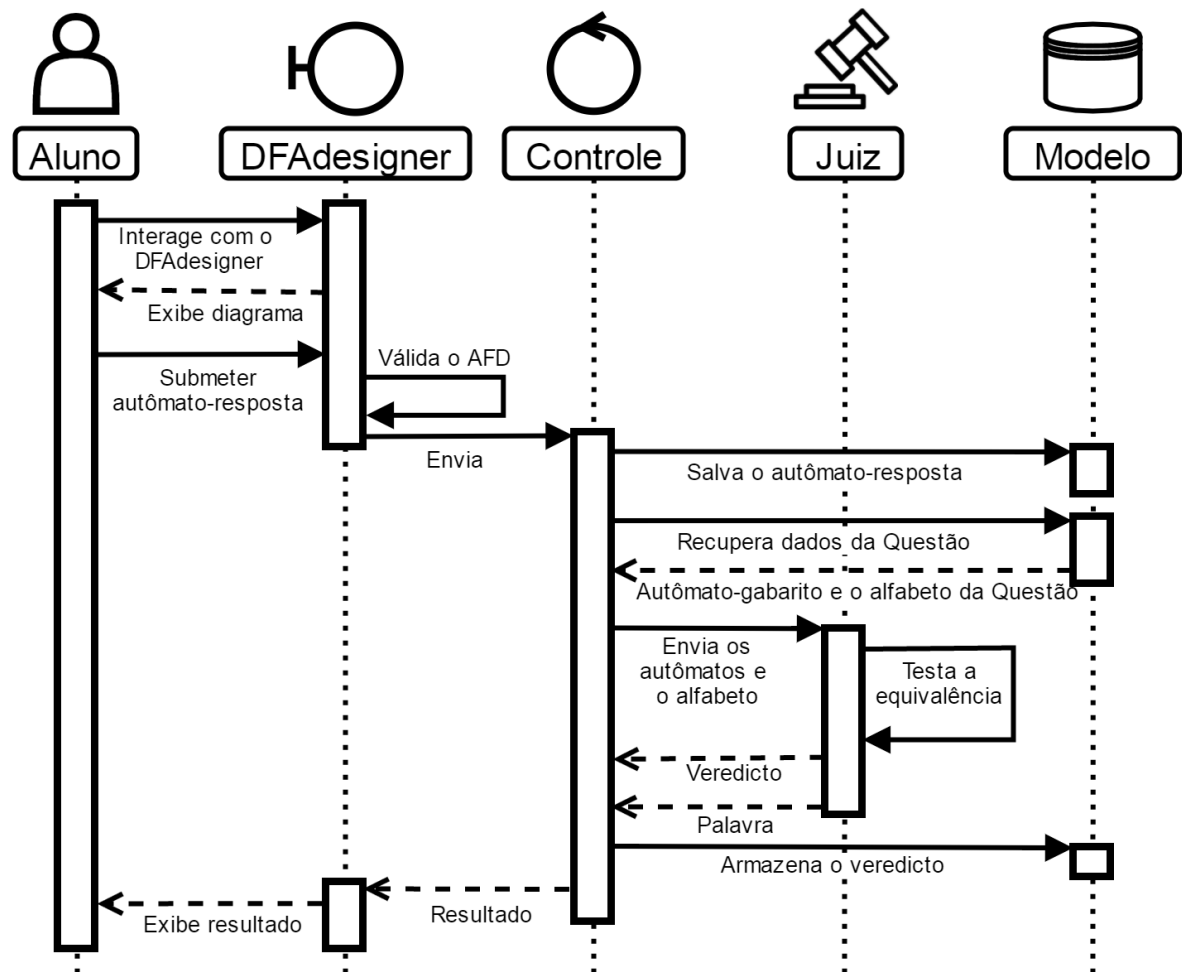


Figura 11 – Diagrama de Sequência da submissão.

O diagrama nos ajuda a entender melhor como se sucede a comunicação do DFA designer com o Juiz sobre a arquitetura de camadas do sistema. Após o Aluno interagir com o DFA designer para confeccionar o seu autômato-resposta, ele pode submetê-lo para a correção, mas primeiro seu diagrama será verificado, e se caso apontar algum erro, então a ação seria abortada e exibiria os erros cometidos. Senão os dados necessários são coletados, da forma codificada apresentada anteriormente, e também a estrutura do desenho do diagrama, para ser enviado ao Controle.

No Controle, ele utiliza o Modelo de Dados para salvar no banco de dados o

autômato-resposta do aluno, que é constituído pelos dados e o desenho. Se caso fosse a ação de salvar como rascunho, este processo terminaria aqui, e retornaria uma mensagem ao usuário informando-o de que o rascunho foi salvo com sucesso.

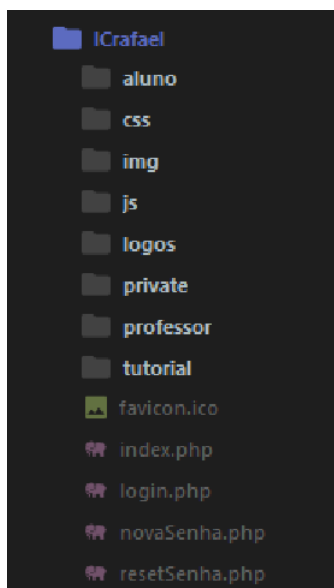
Mas, no caso de uma submissão, o Controle recupera o autômato-gabarito e o alfabeto daquela questão, e entrega ao Juiz o conjunto de símbolos do alfabeto, o autômato-resposta e o autômato-gabarito, para ele realizar o teste de equivalência como descrevermos nos capítulos anteriores.

Ao final, o Juiz retorna seu veredicto, e também a palavra que os distinguiu (se necessária), para o controle que vai alterar o *status* da submissão para o corresponder ao veredicto. E, por ultimo, o controle retorna para o aluno o resultado obtido. Todo esse processo ocorre em uma única requisição do cliente para o servidor.

## 5.5 Implementação

Após feita a análise dos requisitos e o projeto deste sistema, vem a sua codificação. Nesta seção vamos tratar de alguns assuntos referentes ao *design* das interfaces e das implementações realizadas para atender os requisitos deste projeto.

Após estudado o *script* do FSM, algumas de suas funções foram alteradas para possibilitar que o JSON, que representa o diagrama, possa ser salvo em banco de dados e recuperado posteriormente para ser reexibido. Também as novas funções, que discutimos anteriormente, foram adicionadas. Para analisar e coletar os dados do diagrama, bastou varrer a estrutura de dados que o FSM implementa, obtendo então: o índice dos estados e quais são de aceitação, as transições entre os estado que é nomeada com um simbolo e o estado inicial demarcado.



A Figura 12 mostra a estrutura do diretório, onde se encontram os *scripts* do sistema. Na raiz está a página inicial com o login, a recuperação de senha e a definição de senha. Já nas subpastas contém os seguintes arquivos:

- **css** - Os arquivos *Style Sheets*;
- **js** - Os arquivos JavaScript, incluído o DFAdesigner;
- **img** - Ícones para a interface;
- **logo** - Logos do banner e de outras interfaces;
- **tutorial** - Imagens animadas utilizada no tutorial;
- **aluno** - Arquivos do Controle do aluno;
- **professor** - Arquivos do Controle do professor;
- **private** - Classes do DAO, do Juiz e bibliotecas;

Figura 12 – Arquivos.

O banco de dados foi implementado no SGBD MySQL, seguindo os modelos já apresentados. Assim o PHP se conecta a este banco de dados utilizando a biblioteca PDO, que fornece uma interface e os *drivers* necessários para realizar esta conexão de forma segura e garante também a higienização dos valores a serem inseridos no banco de dados.

O *design* das interface se tornou algo importante durante a implementação. Com a rápida prototipagem que RAD proporciona, versões funcionais deste sistema eram demonstrados ao orientador deste projeto, afim dele avaliar e testar as funções implementadas. E suas opiniões ajudavam a melhorar as funcionalidades e também a usabilidade das interfaces do sistema, e até mesmo sugerindo novas funcionalidades.

Então, afim de idealizar as interfaces deste sistema, foram confeccionados os *mockups*, que é equivalente à uma “maquete de uma interface”, não necessariamente de alta fidelidade, para simular o uso da interface.



Figura 13 – Mockup do gerenciamento.

A Figura 13 exemplifica a interface que gerencia os itens das entidades: Turmas, Listas, Questões ou Usuários. No cabeçalho há um banner e um menu de navegação. Em seguida há uma barra com o título da interface e botões de ação pertinentes àquela página.



Abaixo, são listado todos os itens, e em cada linha há botões de ação referentes àquele item. Nesta mesma configuração, é possível, ao invés de listar dados, o sistema exibir um formulário para inserir ou editar algum item.

Figura 14 – Mockup do formulário de resposta.

A Figura 14 exemplifica a interface que o aluno utilizará para resolver uma questão. Este formulário contém: o título da questão e o seu enunciado, o conjunto de símbolos do alfabeto desta questão, o DFA designer para confeccionar o seu autômato-resposta e ao final os botões de Salvar e de Submeter. Este *mockup* serviu como base para todas as outras interfaces que o sistema possui. Seguindo este modelo, pudemos garantir uma padronização que resultou em uma melhor usabilidade do sistema, que discutiremos nos capítulos seguintes.

## 6 O sistema DFAjudge

Neste capítulo será mostrado o resultado final do desenvolvimento deste sistema. A este sistema foi atribuído o nome de **DFAjudge**, abreviação de *Deterministic Finite Automata Judge*, ou em português, Juiz dos Autômatos Finitos Determinísticos. O logo do DFAjudge pode ser visto abaixo, ele é constituído de um autômato de um único estado, que é o inicial e de aceitação, e a partir dele há uma transição para ele mesmo sobre o símbolo “judge”. Simbolizando que o estado é o autômato-resposta por você confeccionado, que ao submete-lo ao Juiz, ele o retorna aceitando-o.



Figura 15 – Logo do DFAjudge.

A partir de agora veremos alguns *screenshots* do DFAjudge, em escala reduzida, e acompanhados de comentários sobre detalhes omitidos nas seções anteriores. Com o apoio da Figura 9 o usuário poderá ver onde ocorre as ações que interligam as páginas. Começamos pela tela de Login.

Sistema de auxílio na  
aprendizagem da disciplina  
Linguagens Formais e Autômatos

E-mail

Senha

Esqueci a senha

LOGAR

UFABC - Universidade Federal do ABC

Figura 16 – Login.

Neste formulário de login é necessário inserir o e-mail e senha para o usuário ter acesso ao sistema. Há também a possibilidade de recuperar a senha, caso esquecida, como vemos na Figura 17.



Figura 17 – Tela de Requisição e Definição de Nova Senha.

A tela a esquerda é a de Requisição de Redefinição de Senha, e ela é acessível pela tela de Login para onde é possível retornar através do link representado pela seta na área superior. Neste formulário é necessário que o usuário digite um e-mail cadastrado no sistema para ser enviado, neste mesmo e-mail, o link de acesso à tela de Definir Nova Senha, que está a direita na figura, onde o usuário deve digitar sua nova senha e redigita-la abaixo.

Um detalhe importante é que, logicamente, para requisitar a redefinição de sua senha, o usuário deve inserir seu e-mail que está de fato cadastrado no sistema. Mas no caso de um usuário mal intencionado, tentar inserir e-mails aleatórios afim de descobrir qual realmente consta no sistema, esta tela se torna porta de entrada para outros tipos de ataques, visto que agora possui o conhecimento de que tal usuário está de fato cadastrado no sistema. Por isto, quando o usuário enviar uma requisição, apenas é exibida a mensagem: “E-mail enviado com sucesso! Cheque seu e-mail”. Deste modo, cadastrado ou não o e-mail, o atacante não receberá uma *feedback* conclusivo sobre sua tentativa.

Como visto anteriormente, o sistema envia e-mails aos usuários em ocasiões específicas. No total são 3 tipos de e-mails, que podemos ver na Figura 18. O primeiro e-mail é enviado quando o usuário é adicionado no sistema, tanto na interface do professor de adicionar novos usuários ou de adicionar alunos em lote em uma Turma, neste e-mail há uma mensagem de boas vindas e instruções de como definir uma nova senha na segunda tela vista na Figura 17. A segunda mensagem diz ao usuário que sua requisição de alteração de senha foi feita, pelo formulário de Requisição de Redefinição de Senha da figura anterior, e abaixo está o link para a tela de Definição de Nova Senha. Uma particularidade deste

link é que contém um *token* que assegura que esta definição de senha é unicamente para aquele usuário, e seu valor foi formado por uma *hash* do horário em microssegundos que foi feita aquela requisição. A terceira mensagem se trata de um alerta que sua senha de acesso foi alterada, afim de avisar o usuário que a operação foi concluída com sucesso.



Figura 18 – Mensagens de e-mail que o sistema envia.

Após o usuário logar no sistema, ele é redirecionado para a Home, como podemos ver na figura a baixo, na qual se trata da Home do usuário Professor.



Figura 19 – Home do Professor.

Nesta tela podemos observar a interface padrão do sistema, onde no cabeçalha há um banner com os logos da UFABC, do DFAjudge e do CMCC. Logo abaixo há o menu de navegação, que logo detalharemos. Ao centro há o conteúdo da página, que é específico em cada interface. E ao final está o rodapé com os créditos do desenvolvimento.

O conteúdo da página Home é semelhante para os dois usuários, nele há o logo do DFAjudge e a esquerda há indicações para os usuários, sobre onde se encontra a Ajuda e como navegar pelo sistema, e no caso do aluno, para ele se atentar a data limite de submissão.

O Menu de Navegação é distinto para cada usuário e apresenta configurações diferentes dependendo do contexto. Abaixo vemos todas as possíveis interações do menu para o usuário Professor.



Figura 20 – Colagem do Menu do Professor.

O Professor tem acesso a todos os itens criados por ele, que são: as questões, as listas com suas questões, as turmas com alunos e suas listas, e também acesso a todos os usuários do sistema. Assim, conforme a página que o professor visualizar, o menu se comporta de modo diferente, ou seja, ele destaca qual dos itens do menu que se relaciona com o conteúdo. Por exemplo, se ele estiver visualizando uma turma, o item Turmas estará em destaque, já se ele estiver editando uma questão, o item Questões se destacará.

Já o usuário do tipo Aluno, tem seu acesso de modo linear, ou seja, ele seleciona primeiramente a sua turma, depois abre uma lista, em seguida escolhe uma questão, e ao fim responde esta questão. Então o menu também se comporta seguindo o contexto, como podemos ver na figura abaixo.



Figura 21 – Colagem do Menu do Aluno.

Mas diferente do professor, os itens do menu do Aluno vão sendo adicionado conforme ele se aprofunda no acesso, visto a dependência entre as entidades, ou seja,

não poderia acessar uma lista de questões sem antes selecionar a sua turma e uma lista. Também deste modo, o menu oferece um rápido regresso entre os itens, não sendo necessário voltar uma a uma.

O item Opções do menu oferece aos dois usuários algumas ações, como: Conta, Ajuda, Sobre e Sair do sistema. Na tela Conta, é possível visualizar dados de sua conta e também redefinir sua senha pessoal como vemos na Figura 22.

### Usuário Aluno

Nome	Rafael Cardoso da Silva
E-mail	rafael.cardoso@aluno.ufabc.edu.br
RA	21048012

**Alterar senha de acesso:**

🔑

🔑

🔑



**🔑 Redefinir Senha**

Figura 22 – Tela da Conta do usuário.



Já na tela de Ajuda, que podemos ver na Figura 23, há um tutorial que utiliza animações e texto para ensinar o usuário a utilizar o DFAdesigner e outros recursos do sistema, como a interpretação das mensagens de *feedback* ou mensagens de erro na construção de seu autômato-resposta. Cada animação demonstra muito bem as ações que devem ser tomadas para realizar certa interação. Para fazer as animações foram utilizadas capturas de tela e ícones que ajudam a representar as ações realizadas. Além disso, há um texto descritivo ao lado de cada desta animação, afim de melhor detalha-la.


## ← Ajuda do DFA judge



Para utilizar o **Construtor de Autômatos Finitos Determinísticos**, basta realizar as seguintes ações descritas a seguir:

**Adicionar Estados**

Para adicionar um estado no seu autômato, basta clicar duas vezes   na área livre do construtor.

A seguir você poderá nomear este estado. Ou a qualquer outro momento, basta somente selecioná-lo com um clique único  assim ele ficará azul indicando que está selecionado.

**Mover um Estado**



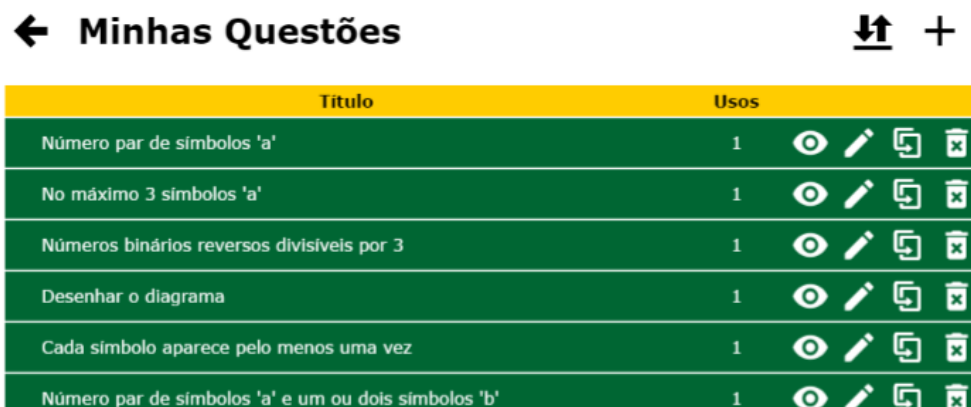
Para mover de posição um estado, basta segurar o clique do mouse  sobre ele e arrasta  para o local desejado.

Figura 23 – Conteúdo da tela de Ajuda.

O Professor desempenha um papel fundamental no sistema, ele deve gerenciar tudo. A seguir veremos o conteúdo de algumas telas que o professor utiliza.

Na Figura 24 são listados as Questões do Professor, por meio de uma lista, em que cada item exibe o título da questão e botões de ação, como: visualizar, editar, duplicar e remover esta questão. A ação de duplicar uma questão foi uma sugestão do próprio orientador durante testes do sistema, em que ele sentiu a necessidade de facilitar a criação de questões que possuam semelhanças, ou seja, quando confeccionar uma questão semelhante a outra já criada, mas que difere em poucos detalhes, é mais fácil poder duplicar-la e modificá-la, do que confeccioná-la do zero. Demonstrando assim que teste de usabilidade do sistema com usuários de verdade resultam em funcionalidade não antes consideradas.

Outra funcionalidade é a possibilidade do professor importar e exportar suas questões em formato JSON, que é muito útil para fazer *backups*, e esta tela é acessível pelo link com ícone de *upload* e de *download* na barra de título desta tela.



← Minhas Questões				
Titulo	Usos			
Número par de símbolos 'a'	1			
No máximo 3 símbolos 'a'	1			
Números binários reversos divisíveis por 3	1			
Desenhar o diagrama	1			
Cada símbolo aparece pelo menos uma vez	1			
Número par de símbolos 'a' e um ou dois símbolos 'b'	1			

Figura 24 – Lista de Questões de uma Lista.

Como o gerenciamento das Turmas, das Listas, das Questões e dos Usuário tem sua interface muito semelhantes, então não apresentaremos elas aqui, pois assim teremos mais oportunidade de explorar outros detalhes mais relevantes.

A Figura 25 a seguir, apresenta o formulário para editar e criar uma nova questão. No primeiro campo deve-se inserir título da questão. No campo do corpo é redigido o enunciado da questão. Tanto no título, quanto no corpo é possível utilizar marcações de comando  $\text{\LaTeX}$  e elementos HTML. Assim ao editar o corpo da questão, logo abaixo na área de borda amarela é exibido o resultado final do enunciado. Em seguida são inseridos os símbolos do conjunto  $\Sigma$ , cada um separado por um espaço, pois este conjunto é utilizado posteriormente pelo DFA designer e pelo Juiz.

Em seguida está o DFA designer, no qual o professor pode confeccionar o autômato-gabarito. A área ocupada pelo DFA designer é grande, o que proporciona criar autômatos de diversos tamanhos e de forma mais espaçosa. Então, após criada a questão, basta clicar no botão de Salvar, caso esteja editando, ou Inserir, caso seja uma nova questão. Em

seguida o professor é direcionado para a tela de Visualizar Questão, onde a questão é exibida da forma que apareceria para o aluno, como vemos posteriormente na Figura 33.

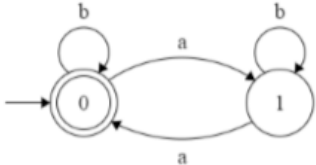
## ← Editar Questão

**Título**

**Corpo** ?

**Alfabeto**   
 $\Sigma = \{ a, b \}$

**Gabarito** ?



```

graph LR
    0((0)) -- b --> 0
    0 -- a --> 1((1))
    1 -- a --> 0
    1 -- b --> 1
    style 0 stroke-width:4px
  
```

**SALVAR**

Figura 25 – Formulário de Criação ou Edição de Questão.

Todas as telas, que editam ou criam um novo item, são as mesmas páginas, como vimos na Figura 9. Porém, se caso for uma edição, então os campos do formulário estarão preenchidos com os antigos dados, senão estará em branco, caso for um novo item.

Na Figura 26 vemos a relação das questões presentes em uma lista, e também o título e a contagem de questões desta lista. Nesta tela é possível acessar a tela de edição desta lista clicando no ícone de lápis no canto superior direito. Também é possível, através



do ícone ao lado, ir para a tela de ordenação de questões, onde com o clicar e arrastar do mouse pode-se reordenar as posições das questões na lista.

Questões da Lista	
Título da Lista	Aquecimento
Número de Questões	10
Escolha a questão para visualizar	
Título da Questão	
<input type="checkbox"/>	Desenhar o diagrama
<input type="checkbox"/>	Número par de símbolos 'a'
<input type="checkbox"/>	Começa e termina com o mesmo símbolo
<input type="checkbox"/>	O terceiro símbolo é um 'b'
<input type="checkbox"/>	Linguagem vazia
<input type="checkbox"/>	Palavra vazia
<input type="checkbox"/>	Lista de palavras I
<input type="checkbox"/>	Contém a subpalavra 'acb'
<input type="checkbox"/>	Contem a subpalavra '0101'
<input type="checkbox"/>	Contém '01101' como subsequência

Figura 26 – Lista de Questões de uma Lista.

Na Figura 27 vemos a tela de Editar Lista, onde temos o campo para inserir um título para esta lista e a relação de todas as questões deste professor, para que ele possa selecionar quais irão compor a lista. A coluna Usos também foi uma sugestão dada pelo orientador durante a implementação, para mostrar quais questões ainda não tinham sido adicionada em uma lista.

Editar Lista

Título da Lista

Aquecimento

Listas

<div></div>	Título da Questão	Usos
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	Número par de símbolos 'a'	1
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	No máximo 3 símbolos 'a'	1
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	Números binários reversos divisíveis por 3	1
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	Desenhar o diagrama	1
<div><div><div></div><div></div></div><div><div></div><div></div></div></div>	Cada símbolo aparece pelo menos uma vez	1

Figura 27 – Editar Lista de Questões.

Na Figura 28 vemos os detalhes de uma turma. Nesta tela está listada todos os alunos da turma e todas as listas aplicadas a ela, onde podemos alternar a visualização destas duas por meio de abas. Nesta tela é possível também acessar a edição desta turma, que é semelhante às das Listas, onde nela são relacionadas todas suas listas e alunos para serem selecionados afim de compor a turma. A partir do Detalhes da Turma pode-se acessar o relatório de desempenho desta turma, pelo ícone que representa uma tabela.

## Detalhes da Turma

Nome da Turma: NAMCTA015-13SA

Quadrimestre: 2016.Q2

Total de Alunos: 45

Total de Listas: 4

VER ALUNOS

VER LISTAS

### Listas aplicadas na Turma

Título da lista	Data Limite		
Aquecimento	24/08/2016 23:59	Relatorio	Remover
Divisibilidade e aritmética	24/08/2016 23:59	Relatorio	Remover
Fáceis	24/08/2016 23:59	Relatorio	Remover
Médias	24/08/2016 23:59	Relatorio	Remover

Figura 28 – Detalhes de uma Turma.

Na Figura 29 vemos um exemplo do relatório de desempenho de uma turma, onde as linhas correspondem aos alunos matriculados e as colunas correspondem às listas aplicadas àquela turma. Cada posição da tabela possui a contagem de acertos que cada aluno obteve em cada lista e, na ultima coluna, está o total de acertos e a porcentagem correspondente. Nesta tela também há a possibilidade de exportar esta tabela em formato CSV, afim de utiliza-la em uma planilha eletrônica para ajudar a compor a nota de cada aluno.

## Relatório da Turma

Nome da Turma: \_\_\_\_\_

Quadrimestre: \_\_\_\_\_

Total de Alunos: 3

Total de Listas: 4

RA	Alunos Nome	Contagem de Acertos				Total (40)
		1 (10)	2 (10)	3 (10)	4 (10)	
_____	_____	0	0	0	1	2.5% (1)
_____	_____	8	0	0	0	20% (8)
_____	_____	10	0	10	10	75% (30)

Figura 29 – Relatório de uma Turma.

A partir de agora falaremos das telas com as quais o usuário do tipo Aluno interage. Após o login, o aluno se depara com uma tela igual à Home do Professor. Em seguida, ao clicar em Turmas no Menu, ele é redirecionado para a tela da Figura 30, onde deve clicar sobre sua turma.

## ← Minhas Turmas

Clique sobre uma Turma para visualizar suas Listas:

Nome	Quadrimestre	Professor
Assistentes	2016.Q2	Daniel M. Martin
Turma Teste	2016.Q1	ADMINISTRADOR

Figura 30 – Turmas do Aluno.

Na Figura 31 vemos todas as lista aplicadas na turma do Aluno. Primeiramente é exibido o nome daquela turma, o quadrimestre em que está sendo ofertada e o professor que criou esta turma. Logo abaixo são listadas as Listas aplicadas a ela. Onde, em cada item, encontra-se o título, a data limite para submissão de resposta e ,na última coluna, o número de acertos do total de questões.

## ← Listas desta Turma

Nome da Turma      Assistentes  
 Quadrimestre      2016.Q2  
 Professor          Daniel M. Martin

Clique sobre uma Lista para visualizar suas Questões:

Título	Data Limite	Acertos
Aquecimento	31/08/2016 23:59	10 / 10
Divisibilidade e aritmética	31/08/2016 23:59	0 / 10
Fáceis	31/08/2016 23:59	10 / 10
Médias	31/08/2016 23:59	10 / 10

Figura 31 – Listas do Turma do Aluno.

Ao selecionar um item da Lista, veremos a tela da Figura 32, que são as questões daquela lista. Temos novamente o nome daquela turma, o quadrimestre, o professor e, agora, informações sobre aquela lista: seu nome, a sua data limite de entrega e a contagem de acertos. Abaixo estão listadas todas as questões daquela lista, onde em cada item temos o título da questão e, na direita, um ícone para simbolizar o seu status. No exemplo da figura, na primeira questão o aluno acertou a resposta, na segunda ele salvou como rascunho, na terceira sua resposta está incorreta e nas outras, sem ícones, ele simplesmente não as respondeu.

Um detalhe importante é que as listas têm data limite de entrega. Assim, caso o aluno tenha ultrapassado esta data, nesta tela e também na anterior haverá um ícone de atenção e uma mensagem dizendo que o prazo para submeter as respostas acabaram, e não será possível submeter e nem salvar. Outro detalhe é que mesmo com a data de tolerância passada, o Aluno poderá acessar todas as questões e somente visualizar suas respostas, afim de poder estudar para a prova.

## ← Questões desta Lista

Nome da Turma	Assistentes
Quadrimestre	2016.Q2
Professor	Daniel M. Martin
Lista	Divisibilidade e aritmética
Data Limite	31/08/2016 23:59
Acertos	1 de 10

Clique sobre uma Questão para responde-la:

Título	Correção
Números binários reversos divisíveis por 3	✓
Números binários divisíveis por 4	🔒
Números binários reversos divisíveis por 4	✗
Números binários divisíveis por 7	
Números binários reversos divisíveis por 7	

Figura 32 – Questões daquela Lista.

Ao selecionar uma questão para resolve-la, a tela da Figura 33 é exibida. Nela, é possível interagir com o DFAdesigner para confeccionar o autômato-resposta para aquela questão e depois submete-lo ao Juiz.

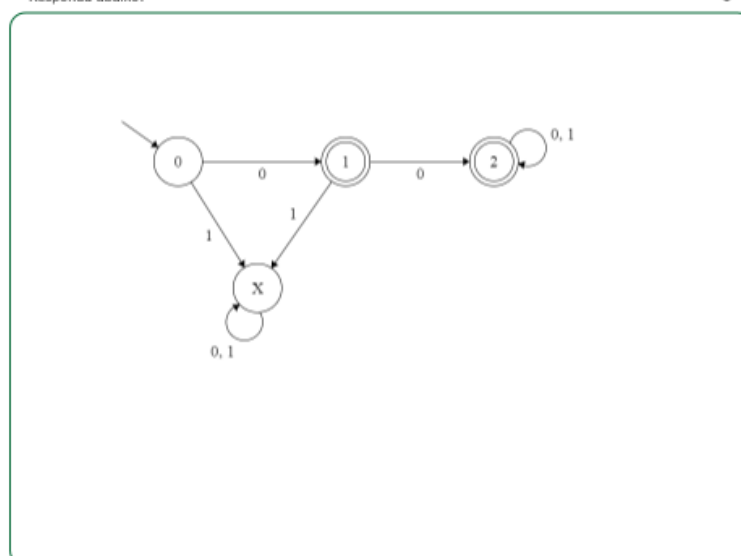
## ← Resolva esta Questão

### Números binários reversos divisíveis por 4

Nesta questão, você deve criar um autômato finito determinístico que aceite precisamente as palavras não-vazias  $w \in \{0, 1\}^*$  que representem números binários divisíveis por 4. Suponha que o dígito **menos** significativo é o primeiro a ser lido pelo autômato, depois o segundo mais significativo e assim por diante.

$\Sigma = \{0, 1\}$

Responda abaixo:



CORRETO !!!

🔒 SALVAR

➡ SUBMITER

Figura 33 – Aluno respondendo uma questão.

Como poderemos observar na Figura 33, o veredicto do Juiz é exibido logo abaixo do DFA designer. Então todos os possíveis veredictos do Juiz e também a mensagem que alerta sobre a data limita, estão na colagem da figura a seguir. No caso da resposta ser incorreta, o *feedback* do juiz será acompanhado da palavra que o distinguiu, como uma dica para o Aluno entender o que ele errou em seu autômato-resposta.

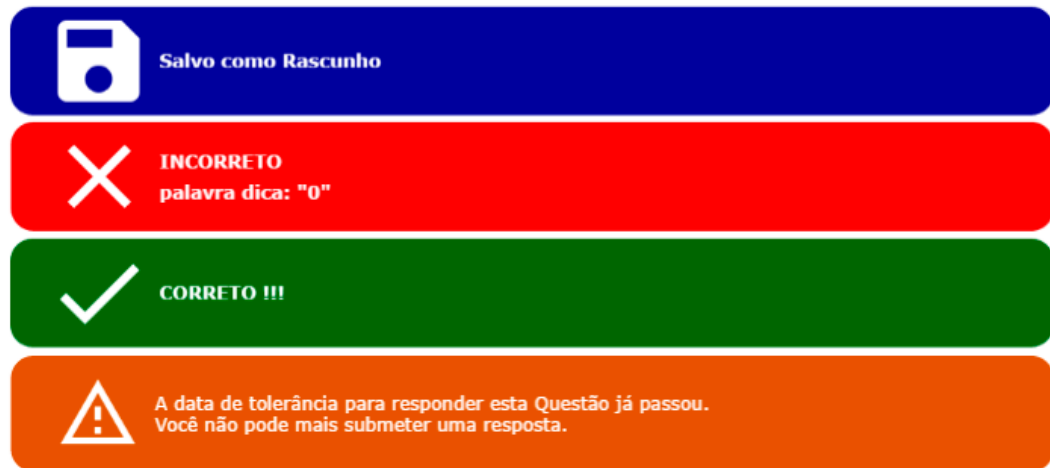


Figura 34 – Colagem do *feedback* da resposta.

Antes de submeter a resposta ao Juiz, o DFA designer realiza uma verificação no diagrama construído, e se houver algum erro, próximo ao *feedback*, será exibido os erros cometidos pelo aluno na confecção do autômato, como vemos no exemplo a seguir.

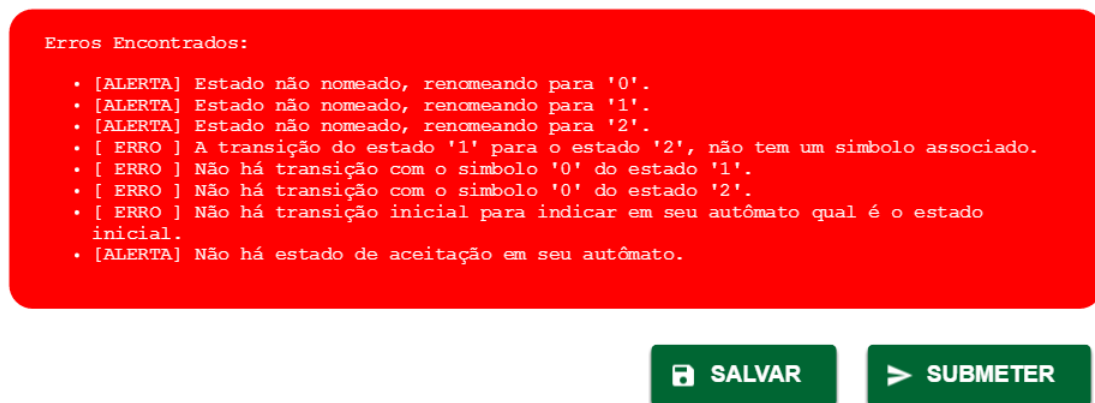


Figura 35 – Erros na construção do autômato-resposta.

São listados dois tipos de mensagens: Alerta e Erro. O alerta não influencia na correta construção do AFD, mas o erro sim. Neste exemplo, o aluno é alertado de que há estados não nomeados e de que estes foram nomeados com números automaticamente. Mas não é necessário os estados terem nomes, porém como eles nomeados ficará mais fácil descrever os outros erros para o usuário. Em seguida são listados erros críticos sobre a construção de um AFD, como: uma transição sem símbolo associado, a falta de transições a partir de um estado e que não foi demarcado o estado inicial. Assim, havendo algum erro, a submissão é abortada e o aluno, ao ler estas mensagens, poderá corrigir os seus erros.

## 6.1 Designer do DFAjudge

Nesta seção analisaremos algumas particularidades a respeito do *designer* das interfaces do DFAjudge. O objetivo de todas interfaces é tornar a interação com o usuário mais o simples e eficaz possível. Então ela deve ser suficientemente clara para que o usuário reconheça facilmente a utilidade e o objetivo dela. Assim, durante o desenvolvimento desse sistema houve a preocupação de maximizar a usabilidade e a experiência do usuário ao utilizar o sistema quando finalizado.

Para começar, o esquema de cores utilizado na interface do DFAjudge, como já foi possível perceber, são as cores verde (#006633) e o amarelo (#FFCC00), pois além de serem as cores oficiais da UFABC, elas têm um ótimo contraste quando associamos o verde com branco e o amarelo com o preto. Assim, proporciona que certos elementos sejam destacados e ainda sejam suficientemente legível seu conteúdo na interface.

Como vimos, a interface do aluno se sucede de maneira simples, onde ele realiza somente uma escolha por página. Desta maneira, os dados são exibidos de modo pregressivo e, conseqüentemente, evitando o excesso de informações exibidos ao aluno. Similarmente ocorre com o Professor, em que cada tela há somente um objetivo a ser realizado.

Como já falamos anteriormente, foi utilizado o artifício do menu se comportar de maneira dinâmica conforme o contexto da página, como vimos na Figura 20 e na Figura 21, e logo abaixo dele, para cada página, há uma barra de título, que nela tem primeiramente um link pra voltar para a página anterior (conforme vimos na Figura 9), um título simples, que ajuda a rapidamente identificar a página, e por último, links para outras páginas que realizam alguma ação sobre aquele objeto. A seguir vemos alguns exemplos de barras de título das páginas do professor.

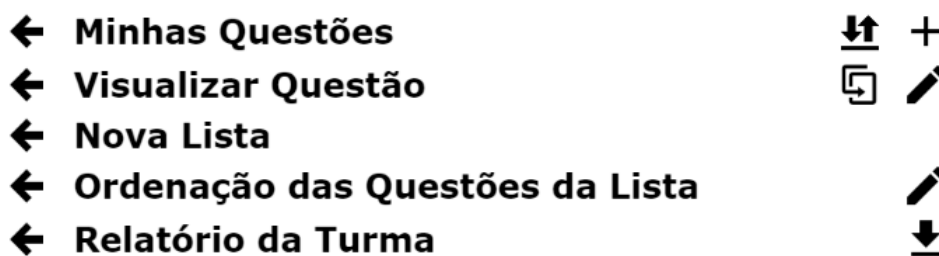


Figura 36 – Barra de título da página.

Assim, a primeira impressão ao ler este título na barra, facilmente leva o usuário a reconhecer o objetivo daquela página, como no exemplo: listar minhas questões, visualizar uma questão, inserir uma nova lista, ordenar as questões de uma lista e exibir o relatório daquela turma. E o link de voltar em destaque na primeira posição, ajuda o usuário a facilmente retornar a página anterior àquela tela. E à direita, como se fossem opções extras, há links representados por imagens, que de forma intuitiva, oferece ao usuário a

possibilidade de redirecioná-lo para uma página que atue sobre aquele objeto, como: editar aquele item, inserir um novo item ou exportar aquela informação.

Então, a união do menu com esta barra, fornece ao usuário uma navegação mais rápida e de fácil identificação de sua localização. E todas as interfaces apresentam este mesmo padrão, resultando em uma boa usabilidade do sistema como um todo.

Outro elemento presente em quase todas as páginas são os botões, como vemos na Figura 37, eles são grandes de cor verde e com ícone e letras de cor branca, assim eles se destacam e o contraste entre as cores facilita a leitura. E também o efeito de sombra passa a sensação dele estar elevado, presumindo assim que são clicáveis. E a utilização de ícones ajuda na rápida interpretação da ação que o botão realiza.



Figura 37 – Botões das interfaces.

Utilizar ícones ao invés de somente texto nos botões e links, ajudam a representar e transmitir ao usuário o objetivo daquele elemento na interface. Resultando assim em uma interface mais limpa e de maior usabilidade. O usuário manipula os objetos de modo simples, mas caso houver alguma dúvida, balões com informações mais detalhadas aparecem sobre o elemento ao parar o *mouse* sobre eles, como vemos nos exemplos a seguir.

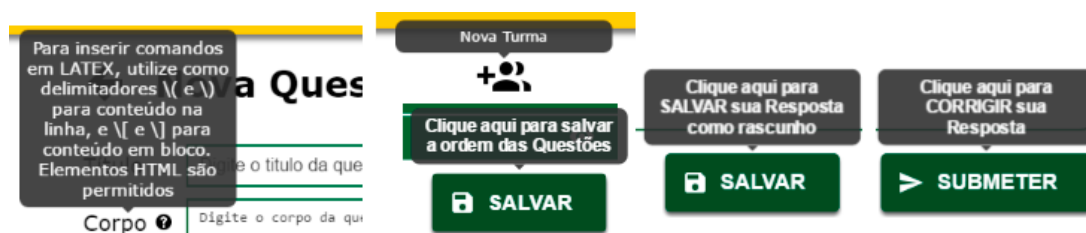


Figura 38 – Bolões com informações a respeito de sua função.

Então podemos concluir que o designer do DFAjudge consegue ser muito satisfatório a respeito da navegação entre as páginas e da usabilidade de suas funcionalidades. O sistema apresenta todos os elementos de forma consistente, ou seja, todos os botões tem aparência semelhante e estão sempre ao final do formulário, os links utilizam imagens para simbolizar sua ação, e todos os outros elementos que constituem as páginas mantêm sempre a mesma aparência, tanto das cores, quanto das suas localizações. Para que assim tenhamos maior clareza na utilização das interfaces deste sistema.

## 7 Teste do sistema em uma turma

Ao final do desenvolvimento deste sistema, houve a oportunidade de utilizá-lo, em caráter experimental, em duas turmas ofertadas da disciplina de Linguagens Formais e Autômatos no segundo quadrimestre de 2016. Então, neste capítulo relataremos os resultados obtidos.

O sistema foi apresentado às turmas pelo professor da disciplina, que é o orientador deste projeto, na primeira semana de aula. Então, seguindo o plano de ensino, após as aulas introdutórias sobre autômato finito determinístico (AFD), os alunos foram adicionados ao sistema. A estas turmas foi aplicada a primeira lista com questões básicas, afim de que os alunos pudessem treinar os conceitos fundamentais de AFD.

Nas semanas seguintes, outras listas foram aplicadas, que abrangiam questões mais complexas do que as primeiras, como divisibilidade e aritmética em autômatos finitos determinísticos. No total, foram aplicadas 4 listas a estas turmas, cada uma delas com 10 questões, totalizando 40 questões que abrangem os tópicos relativos a AFD na disciplina de Linguagens Formais e Autômatos.

Como o sistema estava sendo utilizado caráter experimental nas duas turmas, o professor não considerou obrigatório o uso do sistema e somente 40% dos alunos o utilizaram pelo menos uma vez. Destes, uma fração menor ainda resolveu de fato as listas, como podemos ver no gráfico a seguir, o histograma das notas obtidas no relatório de desempenho das duas turmas unidas.

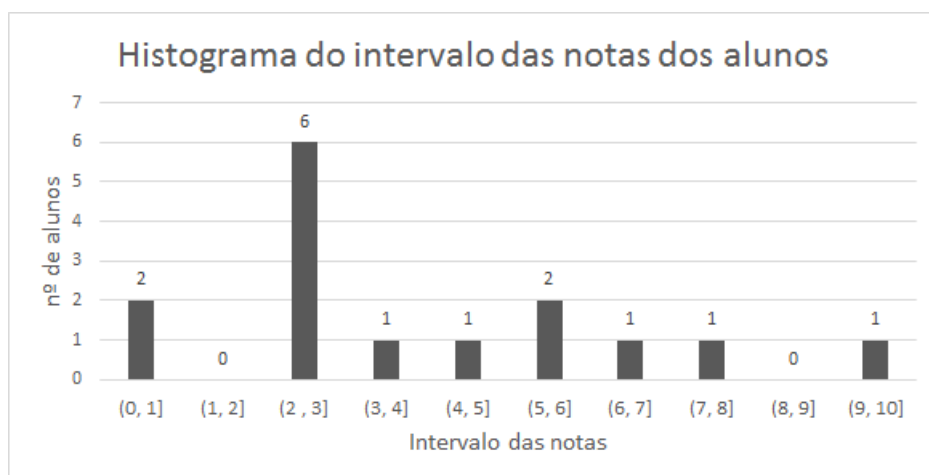


Figura 39 – Histograma das notas que os alunos obtiveram.

Afim de analisar o impacto da utilização do sistema no desempenho dos alunos, calculamos o coeficiente de correlação entre a proporção de exercícios resolvidos e a nota dos alunos. O coeficiente de correlação indica o quanto duas variáveis estão relacionadas



entre si, e será sempre um valor entre  $-1$  e  $1$ . Quanto mais próximo de  $1$ , mais fortemente e positivamente correlacionados os dois dados estão; quanto mais próximo de  $-1$  mais inversamente correlacionados eles estão; e se o coeficiente de correlação estiver próximo de  $0$ , não há correlação entre os dados. Para as turmas ministradas no segundo quadrimestre de 2016, calculamos o coeficiente de correlação entre o percentual de uso do sistema e: a nota da P1, a nota da P2 e a média das provas.

	Correlação
Porcentagem de uso do sistema e P1	0,443637036
Porcentagem de uso do sistema e P2	0,487210236
Porcentagem de uso do sistema e MP	0,575053945

Tabela 5 – Coeficiente de correlação entre a nota do sistema e das provas.

Podemos interpretar, considerando estes valores, que a utilização do sistema para a realização dos exercícios foi benéfica para a fixação dos conceitos, pois há uma clara relação positiva entre resolver as questões e a obter boa nota nas provas.

Outro resultado importante para comentar, é que durante o uso intenso do sistema pelos alunos não houve nenhuma falha.

## 8 Próximos Passos

Embora o sistema tenha sido criado para ser utilizado como uma plataforma de aplicação e correção automática de exercícios que tratam especificamente de autômatos finitos determinísticos, acreditamos que o nosso sistema poderia ser estendido para cobrir outros tipos de exercício envolvendo não só outros tópicos da disciplina de Linguagens Formais e Autômatos, mas também de qualquer outra disciplina, desde que o exercício admita um algoritmo de correção automática. Projetamos o sistema para atender somente uma classe de questões, mas do modo que estruturamos suas regras, seria possível, com poucas adaptações, servir também novas classes de exercícios, bastando somente seguir o mesmo modelo dos componentes implementados: ter um método para o aluno confeccionar sua resposta no *front-end* e um juiz que implemente os devidos algoritmos para corrigir a resposta automaticamente no *back-end*.

Um ponto a ser melhorado no sistema está na listagem dos dados no gerenciamento de algumas entidades, como das listas e das questões pois, com o aumento quantidade de dados, cresce também a necessidade de apresentar os dados de maneira sucinta. O orientador sentiu alguma dificuldade na hora de criar uma lista, pois todas as questões que ele havia introduzido no sistema aparecem de uma vez, em sequência, e ele reportou que fica difícil procurar as questões que deseja. Isso poderia ser melhorado com o auxílio de filtros e ordenações dinâmicas do conteúdo, o que facilitaria a gestão destas entidades.

O DFAdesigner é uma especialização do FSM, mas também poderia ser aprimorado adicionando-se elementos extras, tais como: permitir a definição de um estado como sendo um de morte – estados que não são de aceitação em que todas as transições vão para o próprio estado, porém sem a necessidade de explicitar as transições uma a uma no designer – abranger outros tipos de autômatos, ter uma área de trabalho dinâmica, ou seja, o espaço reservado para a confecção do AFD poderia ser redimensionável no próprio navegador.

Uma das ideias que tivemos no início do desenvolvimento do sistema era permitir que os usuários do tipo professor pudessem compartilhar as questões uns com os outros por meio de um banco público de questões, mas isto aumentaria a complexidade das regras internas do sistema e assim tomamos a decisão de não implementar esta funcionalidade.

Outra ideia interessante era de tornar o sistema público, ou seja, aberto para qualquer usuário que quisesse se cadastrar, permitindo que ele o utilizasse para responder questões e também criar novas questões. Deste modo, o sistema não seria segmentado em turmas de alunos de um professor, mas os usuários teriam papéis similares e poderiam também ter seus desempenhos ranqueados para que pudessem se comparar com os outros usuários.

## 9 Conclusão

Desenvolver o DFAjudge foi um desafio, mas ajudou a melhorar minhas práticas de programação e organização no desenvolvimento de um sistema web. Estudar a fundo os conceitos algorítmicos necessários para sua implementação (que antes eu somente tinha visto em sala de aula) e coloca-los em prática, para resolver problemas reais, foram de grande valia. Para o aluno, ter seus exercícios corrigidos instantaneamente é, de fato, muito útil para ajuda-lo a corrigir as falhas do aprendizado e para fixar os conceitos apresentados em aula, e isso foi demonstrado nas primeiras turmas que utilizaram este sistema. Com o DFAjudge concluído, os próximos passos seriam abranger sua atuação para outras áreas, onde a correção instantânea de exercícios sejam de grande importância, a começar pelos autômatos finitos não determinísticos e pelos demais tópicos da disciplina de Linguagens Formais e Autômatos.

# Referências

CORMEN, T. H. et al. *Introduction to algorithms*. Second. [S.l.]: MIT Press, Cambridge, MA; McGraw-Hill Book Co., Boston, MA, 2001. xxii+1180 p. ISBN 0-262-03293-7. Citado 2 vezes nas páginas 13 e 15.

HOPCROFT, J. An  $n \log n$  algorithm for minimizing states in a finite automaton. In: *Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971)*. [S.l.]: Academic Press, New York, 1971. p. 189–196. Citado na página 8.

HOPCROFT, J.; KARP, R. *A Linear Algorithm for Testing Equivalence of Finite Automata*. [S.l.], 1971. Citado 2 vezes nas páginas 13 e 15.

HOPCROFT, J. E.; KARP, R. M. A linear time algorithm for testing equivalence of finite automata. *Technical report of Cornell University*, p. 71–114, 1971. Citado na página 8.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. Automata theory, languages, and computation. *International Edition*, v. 24, 2006. Citado 3 vezes nas páginas 7, 8 e 13.

HUFFMAN, D. A. The synthesis of sequential switching circuits. I, II. *J. Franklin Inst.*, v. 257, p. 161–190, 275–303, 1954. ISSN 0016-0032. Citado na página 8.

LINZ, P. *An introduction to formal languages and automata*. [S.l.]: Jones & Bartlett Publishers, 2011. Citado na página 7.

MIT. *The MIT Licence*. 1988. <<http://opensource.org/licenses/MIT>>. Citado na página 21.

MOORE, E. F. Gedanken-experiments on sequential machines. In: *Automata studies*. [S.l.]: Princeton University Press, Princeton, N. J., 1956, (Annals of mathematics studies, no. 34). p. 129–153. Citado na página 8.

NIJHOLT, A. The equivalence problem for ll- and lr-regular grammars. In: GécSEG, F. (Ed.). *Fundamentals of Computation Theory*. Springer Berlin Heidelberg, 1981, (Lecture Notes in Computer Science, v. 117). p. 291–300. ISBN 978-3-540-10854-2. Disponível em: <[http://dx.doi.org/10.1007/3-540-10854-8\\_32](http://dx.doi.org/10.1007/3-540-10854-8_32)>. Citado na página 5.

SIPSER, M. *Introduction to the Theory of Computation*. [S.l.]: Cengage Learning, 2012. Citado 2 vezes nas páginas 5 e 7.

WALLACE, E. *Finite State Machine Designer*. 2010. <<http://madebyevan.com/fsm/>>. [Online; acessado 29 de agosto de 2016]. Citado na página 20.