UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS**

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO HIDEYUKI MORIBE

UMA FERRAMENTA DE APOIO AO TREINAMENTO DE LINGUAGENS REGULARES

Bandeirantes

2016

FERNANDO HIDEYUKI MORIBE

UMA FERRAMENTA DE APOIO AO TREINAMENTO DE LINGUAGENS REGULARES

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Me. Wellington A. Della Mura
Co-Orientador: Me. Fábio de Sordi Junior

Bandeirantes
2016

FERNANDO HIDEYUKI MORIBE

UMA FERRAMENTA DE APOIO AO TREINAMENTO DE LINGUAGENS REGULARES

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

COMISSÃO EXAMINADORA

_____
Prof. Me. Wellington Aparecido Della Mura
UENP –*Campus* Luiz Meneghel

_____
Prof. Me. Fábio de Sordi Júnior
UENP –*Campus* Luiz Meneghel

_____
Prof. Me. Thiago Adriano Coleti
UENP –*Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2016

AGRADECIMENTOS

Primeiramente agradeço a Deus, que sempre me guiou e iluminou meu caminho.

Aos meus pais Edson e Meiry, que sempre me apoiaram, e ao meu irmão Fábio, que sempre me aconselhou e ajudou, sendo todos eles fundamentais para o meu triunfo.

Ao meu orientador Della Mura, pelos ensinamentos durante esses 4 anos, pela disponibilidade, dedicação e pelos conselhos que me tornaram um indivíduo profissional e pessoalmente melhor.

Ao meu coorientador Fábio, por aceitar fazer parte desse trabalho, contribuindo de forma crucial. E, pelos ensinos e conselhos que recebi no decorrer deste ano.

A minha namorada Julia, pelo companheirismo, apoio e ajuda em todos os momentos. E, a minha amiga e cunhada Fabiana, que me apoiou e me ajudou na correção deste trabalho.

Aos meus amigos de turma, Andressa, Bruno (Slot), Gabriel (Fada), Letícia, Luan, Janaína, e em especial, Wesley (Dalsin), companheiro de muitos trabalhos, Erick e Felipe (Goku).

Aos grandes amigos que Bandeirantes me deu, Matheus, Victor Hugo, e em especial, Marcus e Tales, que foram meus grandes irmãos nesta cidade.

Aos amigos e amigas do Núcleo de Estudos de Agroecologia e Território, e em especial ao coordenador do Núcleo, Rogério, e ao orientador do projeto, Rafael, que me tornaram uma pessoa e um profissional melhor.

A todos que contribuíram diretamente ou indiretamente para a realização deste trabalho.

RESUMO

Este trabalho tem por escopo conceituar de forma breve, porém concisa, as linguagens formais, tendo um estudo direcionado às linguagens regulares, bem como desenvolver uma ferramenta de apoio ao treinamento de linguagens desse tipo. A teoria das linguagens formais é de ampla importância na área da computação, tendo em vista que esta teoria contém um conjunto de formalismos no qual podem ser aplicados na especificação e reconhecimento de sentenças de uma linguagem de programação. A linguagem regular e os autômatos finitos geralmente são utilizados em um segmento do *front-end* de um compilador, nomeado como análise léxica, além de diversas aplicações em busca e validação de dados. A ferramenta desenvolvida possibilitará que o aluno submeta um autômato finito, seja determinístico, não-determinístico ou com movimentos vazios, de modo que este possa ser corrigido de acordo com as especificações informadas previamente. Ao final do processo, a ferramenta ainda retornará o erro de forma clara a fim de que o aluno possa corrigi-lo e com isso aprender mais sobre o assunto.

Palavras-chave: linguagens regulares, ambiente de treinamento, autômatos finitos.

ABSTRACT

This work aims to provide a briefly introduction about the formal languages, specially related to the regular languages, as well as to develop a tool to support the training of languages of this kind. The theory of formal languages is of great importance in the computation, considering that this theory contains a set of formalisms in which they can be applied in the specification and recognition of sentences of a programming language. Regular languages and finite automata are often used in a front-end segment of a compiler, named as lexer, in addition to a number of applications in text matching and validation. The tool to be developed will enable the student to submit a finite automaton, either deterministic, non-deterministic or with empty movements, so that it can be verified according to the specifications previously reported. At the end of the process, the tool will return the error clearly so that the student can correct it and thereby learn more about it.

Keywords: regular languages, training environment, finite automata.

LISTA DE FIGURAS

Figura 1 - Hierarquia de Chomsky (MENEZES, 2002) 20
Figura 2 - Autômato Finito como uma máquina com controle finito (MENEZES, 2011) 25
Figura 3 - Exemplo de autômato finito determinístico 27
Figura 4 - Exemplo de autômato finito não-determinístico 28
Figura 5 - Exemplo de autômato finito com movimentos vazios 30
Figura 6 - Conversões das linguagens regulares 32
Figura 7 - Base da construção de um AF a partir de uma ER (HOPCROFT et al, 2008) 34
Figura 8 - AF equivalentes às ERs com n+1 operadores (MENEZES, 2011) 34
Figura 9 - AFMV construído a partir da ER (MENEZES, 2011) 35
Figura 10 - AFND N (MENEZES, 2011) 36
Figura 11- AFND M equivalente (MENEZES, 2011) 36
Figura 12 - Situação AFND (RAMOS, 2008) 37
Figura 13 - Situação determinística equivalente à Figura 12 (RAMOS, 2008) 38
Figura 14 - AFND N (MENEZES, 2011) 38
Figura 15 - AFD M equivalente a AFND N 39
Figura 16 - AFD N (RAMOS, 2008) 41
Figura 17- AFM M (RAMOS, 2008) 41
Figura 18 - Ambientes Virtuais de Aprendizagem (SORDI, 2015) 44
Figura 19 - Caracterização dos AVE e AVT (SILVA et al, 2007) 45
Figura 20 - Janela de criação de autômatos finitos 51
Figura 21 - equivalências das linguagens regulares 52
Figura 22 - Arquitetura do método proposto 53
Figura 23 - Método proposto 54
Figura 24 – Representação do TESTE DE ISOMORFISMO 55
Figura 25 – Representação do TESTE DE ACEITAÇÃO 57
Figura 26 – Possibilidade 1 61
Figura 27 – Possibilidade 2 61
Figura 28 – Possibilidade 3 62
Figura 29 – Possibilidade 4 63
Figura 30 – Possibilidade 5 64
Figura 31 - Arquitetura da ferramenta proposta 65
Figura 32 - Casos de uso do FaTLR 66

Figura 33 - Pacotes do FaTLR.....67
Figura 34 - Diagrama de classes do pacote modelo 68
Figura 35 – Diagrama de classe do pacote dao e run 69
Figura 36 - Diagrama de classes do pacote algoritmo.....69
Figura 37 - Diagrama de máquina de estados das interfaces.....70
Figura 38 - Diagrama de sequência da correção do autômato.....71
Figura 39 - resultado SUS por usuário 74
Figura 40 - Menu da FaTLR.....82
Figura 41 - Menu professor 82
Figura 42 - Criar lista e Abrir lista 83
Figura 43 - Verificar resposta.....83
Figura 44 - Menu aluno 84
Figura 45 - Exercícios.....84
Figura 46 - Interface construir autômato 85

LISTA DE TABELAS

Tabela 1 - Tabela de transição do autômato finito determinístico 28
Tabela 2 - Tabela de transição do autômato finito não-determinístico 30
Tabela 3 - Tabela de transição do autômato finito com movimentos vazios 32

LISTA DE QUADROS

Quadro 1 - Vantagens e desvantagens do EaD (Adaptado de LIMA, 2003)..... 43
Quadro 2 - Classe Conteúdo, comparativo entre AVT e AVE..... 46
Quadro 3 - Classe Modelo Pedagógico, comparativo entre AVT e AVE..... 46
Quadro 4 - Classe Modelo de Comunicação, comparativo entre AVT e AVE..... 46
Quadro 5 - Classe Avaliação, comparativo entre AVT e AVE..... 47
Quadro 6 - Casos de uso da FaTLR..... 67

LISTA DE SIGLAS

LF	Linguagens formais
ER	Linguagem regular
GR	Gramática regular
AF	Autômatos finitos
AFD	Autômato finito determinístico
AFND	Autômato finito não-determinístico
AFMV	Autômato finito de movimentos vazios
EaD	Ensino a distância
AVA	Ambiente virtual de aprendizagem
AVT	Ambiente virtual voltado ao treinamento
FaTLR	Ferramenta de Apoio ao Treinamento de Linguagens Regulares
Aa	Autômato finito do aluno
As	Autômato finito do sistema

SUMÁRIO

1. INTRODUÇÃO 13
   1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO 14
   1.2 FORMULAÇÃO DO PROBLEMA 15
   1.3 OBJETIVOS 16
   1.4 JUSTIFICATIVA 16
   1.5 METODOLOGIA DA PESQUISA 17
   1.6 ORGANIZAÇÃO DO TRABALHO 17
2. FUNDAMENTAÇÃO TEÓRICA 19
   2.1 LINGUAGENS FORMAIS 19
   2.2 LINGUAGEM REGULAR 21
      2.2.1 Expressões Regulares 21
      2.2.2 Gramáticas Regulares 23
      2.2.3 Autômato Finito 24
   2.3 Equivalência de linguagens regulares 32
      2.3.1 Equivalência de ER para AFMV 33
      2.3.2 Equivalência de AFMV para AFND 35
      2.3.3 Equivalência de AFND para AFD 36
      2.3.4 Equivalência de AFD para AFM 39
   2.4 Ensino a distância 42
      2.4.1 Ambientes virtuais de aprendizagem 43
      2.4.2 Ambientes Virtuais voltados ao Treinamento 44
3. TRABALHOS RELACIONADOS 49
4. UM MÉTODO AUTOMÁTICO PARA CORREÇÃO DE AUTÔMATOS FINITOS 53
   4.1 Teste de Isomorfismo 54
   4.2 Teste de Aceitação 57
   4.3 Possibilidades de retorno 60
5. DESENVOLVIMENTO DA FERRAMENTA 65
   5.1 Arquitetura da ferramenta 65
   5.2 Análise e projeto 66
6. AVALIAÇÃO DA FERRAMENTA 72

6.1 Resultados da Avaliação 73
7. Considerações finais 75
Referências 76
Apêndice A – interfaces da ferramenta 82
Anexo A – Questionário para avaliação 88

1. INTRODUÇÃO

A linguagem pode ser definida como uma forma de comunicação que utiliza um conjunto de elementos, considerados símbolos, e um conjunto de métodos, ou seja, regras que possibilitam combinar símbolos. Nesse mesmo sentido, as linguagens formais são representadas de forma finita e precisa por meio de um conjunto de sustentação matemática, isto é, modelos matemáticos ou dispositivos formais. Com isso, é importante salientar que a linguagem formal tem extrema relevância na área da Ciência da Computação, pois serve de apoio para outros aspectos desse ramo, como a decidibilidade, a computabilidade e a complexidade computacional. Além disso, ampara inúmeras aplicações computacionais como, o processamento de linguagens e o reconhecimento de padrões (FURTADO, 2002)

A fundamentalidade do estudo da linguagem regular na área da computação deve-se, ainda, ao fato desta poder construir novas linguagens, provar propriedades e construir algoritmos. Ademais, existem vários tipos de software diferentes que utilizam as linguagens regulares, sendo o analisador léxico e os editores de texto os mais importantes a serem mencionados. Vale ressaltar que o analisador léxico é a primeira etapa do *front-end dos* compiladores, conhecidos na área da computação em razão de estarem fortemente relacionados ao desenvolvimento e programação de softwares (MENEZES, 2011; FURTADO, 2002).

Este trabalho tem por finalidade desenvolver uma ferramenta de apoio ao treinamento de linguagens regulares. Devido ao fato de não se ter conhecimento de um software capaz de comparar um autômato finito com outro, surgiu a necessidade de desenvolver uma ferramenta para realizar tal comparação, a fim de facilitar o treinamento de pessoas que possuem interesse na área da teoria da computação. Além dessa comparação, a aplicação proposta realizará também a correção do autômato submetido pelos participantes, com o propósito de propiciar um retorno preciso, no qual auxiliará o aluno na reparação do erro, com a finalidade de proporcionar um aprendizado mais dinâmico ao aluno.

1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO

Com o propósito de elaborar teorias referentes as linguagens naturais, o estudo da teoria das linguagens formais começou a ser desenvolvido na década de 1950. No entanto, logo percebeu-se que a teoria das linguagens formais desempenhava um papel importante no estudo das linguagens artificiais, principalmente nas linguagens provenientes da Ciência da Computação. As pesquisas acerca das linguagens formais aumentaram expressivamente, relacionando-se a diferentes focos, em especial as aplicações em análise léxica e sintática das linguagens de programação (MENEZES, 2000).

Atualmente as linguagens formais podem ser classificadas em linguagens regulares, linguagens livres de contexto, linguagens sensíveis ao contexto e linguagens enumeráveis recursivamente, conforme o entendimento do linguista Noam Chomsky. As linguagens enumeráveis recursivamente são consideradas a de maior abrangência e complexidade, seguidas pelas linguagens sensíveis ao contexto, gramática livres de contexto, e por fim, a linguagem regular, que pode dividir-se em três segmentos: expressão regular, gramática regular e autômato finito.

Toda linguagem regular pode ser escrita por uma expressão, denominada expressão regular. A expressão regular é a forma mais fácil de um ser humano entender a linguagem, em razão da sua proximidade com a matemática. A gramática regular é outro segmento capaz de gerar as linguagens regulares, e contém quatro tipos de regras de produções, sendo elas, gramática linear à direita, gramática linear à esquerda, gramática linear unitária à direita e gramática linear unitária à esquerda. Por sua vez, autômato finito divide-se em autômato finito com movimento vazio, autômato finito não-determinístico e autômato finito determinístico. Vale ressaltar que os autômatos finitos em geral são equivalentes no poder computacional.

Dentro desse contexto, este trabalho busca desenvolver uma ferramenta de apoio ao treinamento de linguagens regulares. Além disso, criar um método de correção automática de um autômato finito, a partir de uma expressão regular dada.

1.2 FORMULAÇÃO DO PROBLEMA

As linguagens formais são aplicadas em diversos tipos de software, como verificação do comportamento de circuitos digitais, editores de texto e principalmente na construção de um compilador. Já a aplicação da linguagem regular, componente das linguagens formais, destaca-se no segmento da análise léxica, considerada o primeiro passo do front-end do compilador e responsável pela divisão do texto de entrada em unidades lógicas, sendo elas identificadores, pontuação e palavras-chave (HOPCROFT; ULLMAN; MOTWANI, 2003).

Os formalismos da linguagem regular podem ser representados por meio de expressões regulares, formados por meio de gramáticas regulares e reconhecidos através de autômatos finitos, sendo que estes últimos dividem-se em determinístico (AFD), não-determinístico (AFND) e com movimentos vazios (AFMV). Os autômatos finitos não-determinísticos e o com movimentos vazios são mais simples de serem criados quando comparados ao autômato finito determinístico. Porém exigem técnicas recursivas (*backtraking*) em suas implementações, implicando em um maior tempo de reconhecimento da sentença. O autômato finito determinístico em geral é mais complexo de ser estabelecido, por outro lado, o reconhecimento de sentenças é mais eficiente (AHO; LAM; ULLMAN, 2008; HOPCROFT; ULLMAN; MONTWANI, 2003).

Qualquer expressão regular pode ser convertida em um autômato finito com movimentos vazios. O autômato finito com movimentos vazios pode se transformar em um autômato finito não-determinístico e este pode converter-se em um autômato finito determinístico. Entretanto, também é sempre possível alcançar um autômato finito mínimo (AFM) equivalente a qualquer AFD. Para que isso ocorra, é necessário reconhecer as mesmas sentenças aceitas pelo AFD e, além disso, ter um número menor ou igual de estados.

Vale ressaltar que o AFM é único para cada linguagem, portanto, se dois autômatos finitos (AF) quaisquer diferentes passarem pelas conversões acima citadas, até chegarem a um AFM cada, é possível indagar se os dois autômatos finitos mínimos obtidos são equivalentes, ou seja, aceitam a mesma linguagem. Este trabalho tem por objetivo criar uma ferramenta de apoio ao treinamento de autômatos finitos que seja capaz realizar correção automática de autômatos, visto que os para os discentes compreenderem esse conteúdo, é necessário que eles praticam, e com

a ferramenta facilitaria o aprendizado. Além disso o estudo das linguagens regulares pode-se se tornar mais dinâmico, devido ao fato que caso o autômato do aluno esteja incorreto, a ferramenta retornará o erro de forma precisa, a fim que o usuário possa corrigi-lo.

1.3 OBJETIVOS

O presente trabalho tem por objetivo desenvolver uma ferramenta de apoio ao treinamento de linguagens regulares. Para atingir o objetivo geral deste trabalho, os seguintes objetivos específicos foram:

- Conceituar as propriedades das linguagens regulares: descrever os três formalismos principais das linguagens regulares: autômatos finitos, gramáticas regulares e expressões regulares.
- Conceituar a equivalência entre os formalismos das linguagens regulares: definir as conversões e equivalências entre os autômatos finitos, as gramáticas regulares e as expressões regulares.
- Desenvolver um método de comparação entre as linguagens regulares: pesquisar, analisar e desenvolver algoritmos de comparação entre os formalismos das linguagens regulares.
- Definir meios de prover retorno preciso para o aluno: definir um retorno ao aluno a fim de facilitar a correção do autômato submetido.
- Determinar uma forma de correção entre formalismos: criar um algoritmo de geração de palavras a partir de uma determinada expressão regular e um algoritmo de isomorfismos entre dois autômatos.

1.4 JUSTIFICATIVA

O presente estudo trará a possibilidade da construção de autômatos finitos a partir de uma expressão regular dada pelo exercício proposto. Esses autômatos construídos passarão por uma validação, na qual haverá um retorno para o aluno corrigi-lo caso esteja incorreto. Isso pode auxiliar o aprendizado, diminuindo a

dificuldade de treinamento das linguagens regulares, visto que essa é uma das maiores adversidades enfrentadas pelos discentes no estudo de linguagens formais.

Assim sendo, esse trabalho trará uma maior dinamização de ensino, proporcionando um aprendizado mais eficiente, visto que acarretará um desafio para os discentes, auxiliando-os a se tornar mais autodidatas. Essa ferramenta de apoio também possibilitará o treinamento não presencial do aluno pelo professor, proporcionando ao docente um parecer da turma, o que lhe trará benefícios, tendo em vista que o controle sobre seus alunos será maior e sua didática poderá ser gradativamente aperfeiçoada.

1.5 METODOLOGIA DA PESQUISA

Este trabalho, quanto a sua natureza, é classificado como uma pesquisa aplicada, uma vez que é uma pesquisa que objetiva gerar conhecimentos para aplicações práticas voltadas à solução de problemas específicos (SILVA, 2004). Quanto aos procedimentos técnicos, essa pesquisa é considerada experimental, posto que o objetivo deste trabalho é desenvolver uma ferramenta de apoio ao treinamento em linguagens regulares. Conforme o entendimento de Gil (2002), pode-se definir como sendo pesquisa experimental o processo que determina o objeto de estudo, identifica as variáveis que seriam capazes de intervi-lo e define as formas de controle e de estudo dos efeitos que a variável gera no objeto.

Quanto a forma de abordagem, o trabalho pode ser classificado como quantitativo, em razão do processo de avaliação de usabilidade e satisfação do usuário ser quantificável. Ainda, segundo Gil (2002), uma pesquisa pode ser considerada quantitativa quando traduz em números informações e opiniões para assim classificá-los e analisá-los.

1.6 ORGANIZAÇÃO DO TRABALHO

A Organização do trabalho obedece a estrutura a seguir. Primordialmente, no Capítulo 2 é apresentado um referencial teórico sobre linguagens formais, linguagens regulares, propriedades das linguagens regulares, ensino a distância e ambientes virtuais de aprendizagem e treinamento. O Capítulo 3 apresenta os trabalhos

relacionados. Já Capítulo 4 contém o método desenvolvido, e este é utilizado para a correção de autômatos finitos. O Capítulo 5 apresenta o desenvolvimento da ferramenta, análises e projeto da mesma. O Capítulo 6 apresenta a avaliação de usabilidade e satisfação dos usuários que utilizaram a ferramenta proposta. E finalmente, o Capítulo 7 traz a conclusão do trabalho e os possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma introdução às linguagens formais e uma minuciosa explanação sobre as linguagens regulares com um foco principal nos autômatos finitos. Além disso, abordará sobre as equivalências e propriedades das linguagens regulares. Ainda, apresentará uma breve consideração sobre o ensino a distância (EaD), bem como denotará sobre os ambientes virtuais de aprendizagem (AVA) e, principalmente, ambientes virtuais de treinamento (AVT). Por fim, observando-se que o objetivo geral deste trabalho é o desenvolvimento de uma ferramenta de apoio ao treinamento, será apresentada uma comparação entre o AVA e AVT.

2.1 LINGUAGENS FORMAIS

Segundo Furtado (2002) linguagem pode ser definida por um conjunto de elemento e um conjunto de regras para agrupar estes símbolos, como, por exemplo, linguagens naturais, linguagens de programação. Neste mesmo sentido, linguagens formais são evidenciadas de modo finito e preciso por meio de dispositivos formais ou modelos matemáticos. Definida uma linguagem L com um alfabeto Σ, pode-se gerado um conjunto de palavras formado por $L \subseteq \Sigma^*$. O estudo dessa linguagem possibilita o reconhecimento e a especificação de linguagens, características, estruturas, propriedades, inter-relacionamentos e classificações.

Segundo Menezes (2002), a classificação para as linguagens formais, nomeada como hierarquia de Chomsky, contém uma relação de expressividade entre as linguagens formais partindo da linguagem regular, a mais simples, até a linguagem enumerável recursivamente, uma das mais completas. O diagrama que representa a hierarquia de Chomsky pode ser visualizada na Figura 1.

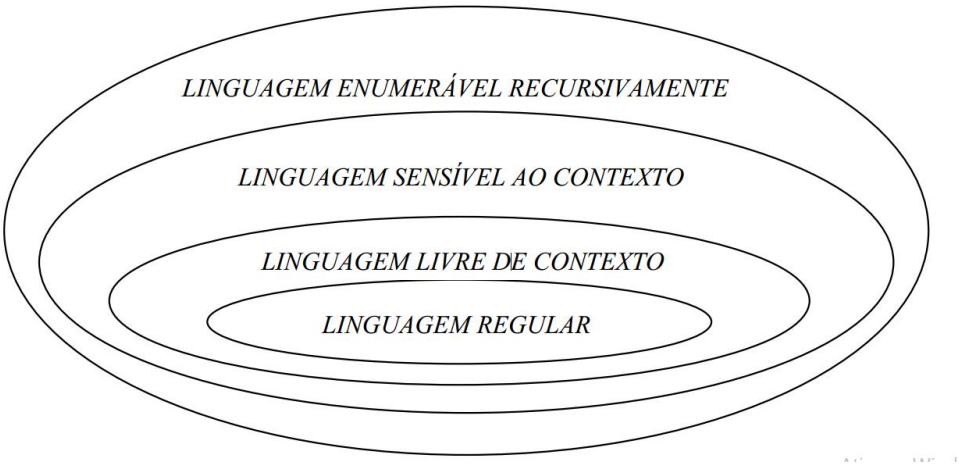

Figura 1 - Hierarquia de Chomsky (MENEZES, 2002)

De acordo com a Hipótese de Church, a máquina de Turing é conceituada o dispositivo mais poderoso da computação, desta forma a linguagem que é aceita por essa máquina é classificada como sendo uma Linguagem Enumerável Recursivamente ou nomeada como linguagem classe 0. Observa-se que essa classe é considerada abastada, uma vez que representa todas as linguagens que podem ser compiladas mecanicamente. A Linguagem Sensível ao Contexto ou classe 1, pode ser aceita por uma Máquina de Turing com limitação finita na dimensão da fita e está incluída precisamente nas Linguagens Recursivas. (MENEZES, 2011).

Seguindo essa linha de raciocínio, a Linguagem Livre de Contexto ou classe 2 tem uma linguagem relativamente restrita, estando contida na Linguagem Sensível a Contexto e indiretamente na Linguagem Enumerável Recursivamente. A Linguagem Livre de Contexto é considerada a mais importante na área de linguagens de programação e compiladores, em razão de especificarem as construções sintáticas utilizadas atualmente de forma eficiente. O estudo da linguagem da classe 2 é abordado utilizando-se da Gramática Livre de Contexto e Autômato com Pilha como formalismos. (FURTADO, 2002).

Por último, a linguagem classe 3, nomeada como Linguagem Regular, é a classe mais simples da Linguagem Formal, possuindo fortes limitações de expressividade. Por outro lado, produz algoritmos reconhecedores, geradores ou transformadores, entre os formalismos com uma complexidade baixa, eficiente e de fácil implementação. Essa linguagem é útil em inúmeros tipos distintos de software, como hipertextos, hipermídias, editores de texto e especialmente no Análise Léxica

de um compilador, sendo está o primeiro passo para a construção de um compilador (MENEZES, 2011).

2.2 LINGUAGEM REGULAR

Nesta seção serão abordados os formalismos da linguagem regular. As linguagens regulares possuem três formalismos principais: autômatos finitos, gramáticas regulares e expressões regulares.

Uma linguagem regular pode ser representada por meio de expressões regulares, formadas por meio de gramáticas regulares e reconhecidas através de autômatos finitos, sendo que estes dividem-se em determinísticos, não-determinísticos e com movimentos vazios. (HOPCROFT; ULLMAN; MONTWANI, 2002).

2.2.1 Expressões Regulares

Linguagens regulares podem ser representadas através de uma expressão titulada como expressão regular (ER), que denotam normas de conjunto de caracteres (*Strings*). Assim como as GRs, as ERs podem ser consideradas geradoras, isto é, possuem a capacidade de formar as palavras de determinada linguagem. As ERs são compatíveis para o comunicação entre humano x humano e especialmente, humano x máquina. Uma expressão regular é estabelecida com base em conjuntos básicos e operações de união e concatenação. (LOUDEN, 2004; MENEZES, 2011);

Em conformidade com Menezes (2011) e Ramos (2008), pode-se concluir através do método indutivo que dada uma expressão regular com um determinado alfabeto Σ:

- $\emptyset$ é uma expressão regular e expressa a linguagem vazia;
- ε é uma expressão regular e expressa a linguagem que inclui somente a palavra vazia $\{ \varepsilon \}$;
- Cada símbolo $x \in \Sigma$ é uma expressão regular e expressa somente a palavra estabelecida pelo símbolo $\{ x \}$.

Segundo Ramos (2008), Maheshwari e Smid (2016), pode-se definir três

operações fundamentais nas expressões regulares. Considerando duas linguagens $L_1 = \{0,1,11,001\}$ e $L_2 = \{\varepsilon, 11\}$ com o mesmo alfabeto ($\Sigma = \{0,1\}$) e w é uma palavra, temos:

União de L_1 e L_2 é definida como:

($L_1 \mid L_2$) ou ($L_1 + L_2$).

é expressão regular e indica a

$L_1 \cup L_2 = \{w : w \in L1 \; ou \; w \in L2\}$

$L_1 \cup L_2 = \{\varepsilon, 0, 1, 11, 011\}$

A concatenação de L_1 e L_2 é definida como:

(L_1L_2).

é expressão regular e indica a

L_1 e $L_2 = \{ww' : w \in L1 \; e \; w' \in L2\}$

$L_1L_2 = \{0, 1, 11, 011, 111, 1111, 01111\}$

Concatenação sucessiva de L_1 é definida como:

(L_1*).s

é expressão regular e indica a

$L_1 = \{v1, v2 \ldots vk : k \geq 0 \in L1 \; \forall \; i = 1,2, \ldots k\}$

$L_1 = \{\varepsilon, 0, 1, 00, 11, 001, 110, 0011, 1100 \ldots\}$

Tendo em vista tornar-se ainda mais prático o uso das expressões regulares, pode-se omitir parênteses contendo sub-expressões que englobam sequencias específicas de operadores, de concatenação ou união, já que trata-se de operações associativas. Assim como acontece na matemática com as expressões aritméticas, os parênteses são aplicados para alterar a precedência. Nesse mesmo sentido destacam-se precedências diferentes para as três operações citadas acima: união, concatenação e concatenação sucessiva. A que tem precedência mais alta é a concatenação sucessiva, logo após vem a concatenação, e por último a união. (RAMOS, 2008; MENEZES, 2011). Para um melhor entendimento da aplicação de precedências, a expressão regular $((0(1)^*)|0)$, descreve palavras que se inicia com 0, seguido de nenhum ou inúmeros 1's, e consegue ser simplificada para $01^*|0$ (LIMA, 2006).

2.2.2 Gramáticas Regulares

De acordo com Menezes (2011), existe a possibilidade de definir tanto linguagens não regulares como linguagens regulares. No entanto, é possível estipular limitações nas regras de produção, de tal modo a delimitar rigorosamente a classe de linguagens regulares. Existem vários formatos de reduzir as regras de produção para a definição de uma gramática regular (GR). Nesse trabalho serão abordados quatro desses formatos, nomeados gramáticas lineares.

Ainda de acordo com Menezes (2011) e Palazzo (2007), Sendo $\mathbf{G = (V, T, P, S)}$ uma gramática, onde $\mathbf{V}$ são variáveis, $\mathbf{T}$ são terminais, $\mathbf{P}$ regras de produção e $\mathbf{S}$ o início. $\mathbf{A}$ e $\mathbf{B}$ pertencem a $\mathbf{V}$, e $\mathbf{w}$ é uma palavra de $\mathbf{T^*}$, portanto, $\mathbf{G}$ pode ser considerada uma gramática linear se todas as suas produções satisfaz-se em um e em somente um dos seguintes formatos:

Gramática linear à direita (GLD). Todas as regras de produção são do formato:

$$A \rightarrow wB \; ou \; A \rightarrow w$$

Gramática linear à esquerda (GLE). Todas as regras de produção são do formato:

$$A \rightarrow Bw \; ou \; A \rightarrow w$$

Gramática linear à direita (GLUD). Todas as regras de produção idêntica a GLD e, além disso:

$$|w| \leq 1$$

Gramática linear à esquerda (GLUE). Todas as regras de produção idênticas a GLE e, além disso:

$$|w| \leq 1$$

Qualquer gramática linear pode ser considerada uma gramática regular. Seja $\mathbf{G = (V, T, P, S)}$ uma gramática, a linguagem gerada por esta é denotada por $\mathbf{L(G)}$ ou $\mathbf{GERA(G)}$. (PALAZZO, 2007).

A linguagem regular $(\boldsymbol{b} + \boldsymbol{c}) * \boldsymbol{aa}(\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c}) *$ é gerada pelas seguintes gramáticas regulares:

Gramática linear à direita (GLD). $G = (\{S, X, Y, \}, \{a, b, c\}, P, S)$ e P possui as seguintes regras de produção:

$$\{S \rightarrow bX \mid cX \mid aa,$$
$$X \rightarrow bS|cS|aa \mid aaY,$$

$$Y \rightarrow aY|bY|cY|\varepsilon\}$$

Gramática linear à esquerda (GLE). $G = (\{S, X, Y\}, \{a, b, c\}, P, S)$ e P possui as seguintes regras de produção:

$$\{S \rightarrow Xa|Xb|Xc|aa,$$
$$X \rightarrow Sa|Sb|Sc|aa|Yaa,$$
$$Y \rightarrow Yb|YC|\varepsilon\}$$

Gramática linear à direita (GLUD). $G = (\{S, X, Y, Z\}, \{a, b, c\}, P, S)$ e P possui as seguintes regras de produção:

$$\{S \rightarrow bX|cX \mid aY,$$
$$X \rightarrow bS|cS|aY \mid aYZ,$$
$$Z \rightarrow aZ|bZ|cZ \mid \varepsilon,$$
$$Y \rightarrow a\}$$

Gramática linear à esquerda (GLUE). $G = (\{S, X, Y, Z\}, \{a, b, c\}, P, S)$e P possui as seguintes regras de produção:

$$\{S \rightarrow Xa|Xb|Xc|Ya,$$
$$X \rightarrow Sa|Sb|Sc|Ya|ZYa,$$
$$Z \rightarrow Zb|Zc|\varepsilon,$$
$$Y \rightarrow a\}$$

Segundo Menezes (2011), dada uma linguagem L, então:

L é gerada por uma GLD se, e somente se,

L é gerada por uma GLE se, e somente se,

L é gerada por uma GLUD se, e somente se,

L é gerada por uma GLUE.

Ou seja, pode-se concluir que as diferentes formas das gramáticas lineares são considerados formalismos equivalentes. Uma gramática G é apontado como uma gramática regular, geralmente abreviada por GR, se G é uma gramática linear. Seja $G = (V, T, P, S)$ uma gramática, a linguagem gerada por G, pode ser denotado por: L(G) ou GERA(G), e tal que: $L(G) = \{ w \in T^* \mid S \Rightarrow^+ w \}$.

2.2.3 Autômato Finito

Autômatos finitos (AF) são ferramentas reconhecedoras de ampla importância na computação. São compostos por um conjunto de estados, e seu controle se

movimenta de estado para estado em *feedback* à fita. Geralmente são utilizados na identificação e classificação dos *tokens* com o objetivo de tratar os erros léxicos de determinada linguagem. A análise léxica é considerada um segmento do *front-end* de um compilador, sendo responsável por desmembrar caractere por caractere do programa-fonte (código-fonte) e classificá-las adequadamente. A performance de um analisador léxico influencia no funcionamento de um compilador (AHO; LAM; ULLMAN, 2008; HOPCROFT; ULLMAN; MOTWANI, 2002).

De acordo com Menezes (2011), esse reconhecedor é constituído por três fragmentos:

- **Fita:** Mecanismo de entrada que recebe a informação a ser processada. Essa é finita, começando mais à esquerda. Dividida em células no qual cada uma armazena um símbolo, que encontra-se em um alfabeto. Não contém memória auxiliar. Inicialmente a fita mantém ocupada pela sentença a ser processada;
- **Unidade de controle:** Pondera o estado corrente do mecanismo. Apresenta um elemento de leitura, o qual acessa uma unidade da fita de cada vez e movimenta-se somente para a direita. A posição inicial da unidade de controle é a célula mais à esquerda possível da fita. Possui uma quantidade finita e predefinida de posições;
- **Função de transição:** Função que dirigi a leitura dos símbolos da fita e define o estado do mecanismo. Dependendo do estado e do símbolo lido, um novo estado é demarcado pelo autômato;

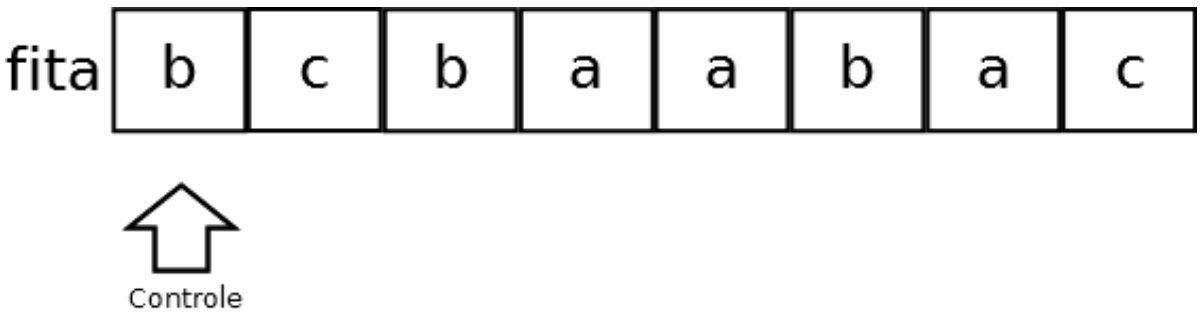

Figura 2 - Autômato Finito como uma máquina com controle finito (MENEZES, 2011)

Neste mesmo sentido, Menezes (2011) dispõe há duas formas de condição de parada do processamento de um autômato finito para determinar entrada **w**. São elas:

1. Aceita a entrada **w**. Após o último símbolo da fita ser processado e o autômato finito encontra-se em um estado final;
2. Rejeita a entrada **w**. São duas alternativas:
   a. Após processar o último símbolo da fita, o autômato encontra-se no estado não final;
   b. Ao longo do processamento de **w**, em algum instante, a função de transição é indefinida para o pressuposto (estado atual e símbolo lido da fita).

Os autômatos finitos podem ser descritos de três formas distintas: diagrama de transição, representação formal e tabelas de transição. O diagrama de transição pode ser considerado um grafo, no qual os estados são nodos e interpretado por círculos e transições são arestas, interligando os correspondentes nodos. Conforme Lima (2006) o estado inicial apresenta uma seta na frente do estado especifico e os estados com um círculo duplo representam o(s) estado(s) final(is). Já a representação formal é uma quíntupla $M = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$. E as tabelas de transição são uma forma comum de representar um autômato finito, na qual as linhas correspondem aos estados (inicial é apontado por uma seta e os finais por asteriscos) e as colunas denotam símbolos de entrada. A conjunção dessas tabelas definem o próximo estado que o mecanismo deverá se encontrar. Por exemplo:(q, a) será igual a "p" se houver $\delta(\mathrm{q}, \mathrm{a}) = p$, caso contrário será indefinível. (MENEZES, 2011; FURTADO, 2002; AHO; LAM; ULLMAN, 2008).

De acordo com Hopcroft, Ullman e Motwani (2002), os reconhecedores dividem-se em:

- **Determinístico**: para cada estado concorrente e o símbolo lido da entrada, o sistema assume um único estado bem determinado;
- **Não determinístico:** para o estado corrente e o símbolo lido da entrada, o sistema assume um estado pertencente a um conjunto de estados alternativos;
- **Movimentos vazios:** para o estado corrente e independentemente de ler um símbolo ou não da entrada, o sistema assume um estado pertencente a um conjunto de estados alternativos (portanto, é não determinístico). O movimento é dito movimento vazio se o sistema muda de estado sem uma correspondente leitura de símbolo.

Autômatos Finitos Determinísticos

A fim de proporcionar um melhor entendimento do conceito de Autômato Finito Determinístico (AFD), tem-se a seguinte representação da expressão regular $(b + c) * aa(a + b + c) *$ em formato de diagrama de transição:

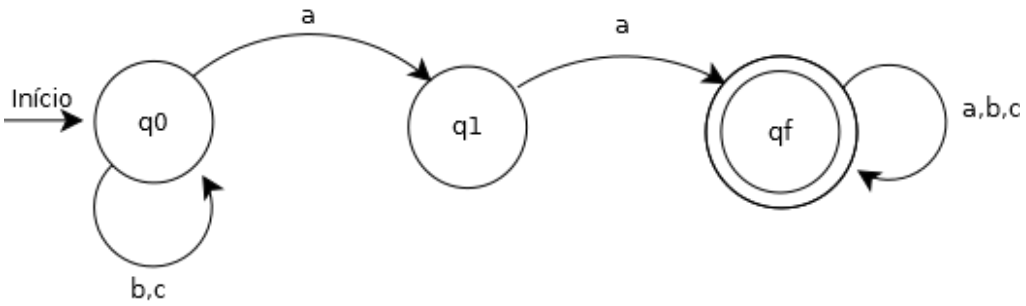

Figura 3 - Exemplo de autômato finito determinístico

Inicialmente o autômato encontra-se no estado $\mathrm{q0}$e pode manter-se no mesmo estado lendo **b** ou **c**, ou poderá transitar para o estado $\mathrm{q1}$ se, e somente se, ler **a**. Em seguida, o estado $q1$ apenas irá mover-se ao estado qf (estado final) se ler **a**. O qf poderá ler inúmeros **a**, **b**, **c** dependendo de sua entrada.

Segundo Hopcroft, Ullman e Motwani (2002), a representação formal de um autômato finito convencional é uma quíntupla $M = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$, na qual:

$Q -$ conjunto finito de estados possíveis do autômato;

$\Sigma -$alfabeto de símbolos de entrada;

$\delta -$ função programa ou função de transição;

δ: $Q \times \Sigma \rightarrow \mathrm{Q}$

$q0 -$denominado estado inicial, onde $q0 \in Q$;

$F -$conjunto de estados finais, no qual($F \subset Q$).

A apresentação do autômato finito correspondente à figura 2 deu-se através da representação formal:

$$Q = \{q0, q1, qf\}$$

$$\Sigma = \{a, b, c\}$$

$$\delta = \{\delta(q0, b) = q0,$$

$$\delta(q0, c) = q0,$$

$$\delta(q0, a) = q1,$$

$$\delta(q1, a) = qf,$$
$$\delta(qf, a) = qf,$$
$$\delta(qf, b) = qf,$$
$$\delta(qf, c) = qf\}$$
$$q0 = \{q0\}$$
$$F = \{qf\}$$

A Tabela 1 exemplifica a tabela de transição do autômato finito correspondente ao mecanismo da figura 3:

Tabela 1 - Tabela de transição do autômato finito determinístico

δ	a	b	c
$\rightarrow * q0$	$q1$	$q0$	$q0$
$q1$	qf	$-$	$-$
qf	qf	qf	qf

Autômatos Finitos Não-Determinísticos

Para proporcionar uma melhor percepção do conteúdo de autômatos finitos não-determinístico (AFND), tem-se a seguinte representação da expressão regular $(b + c) * aa(a + b + c) *$ em forma diagrama de transição:

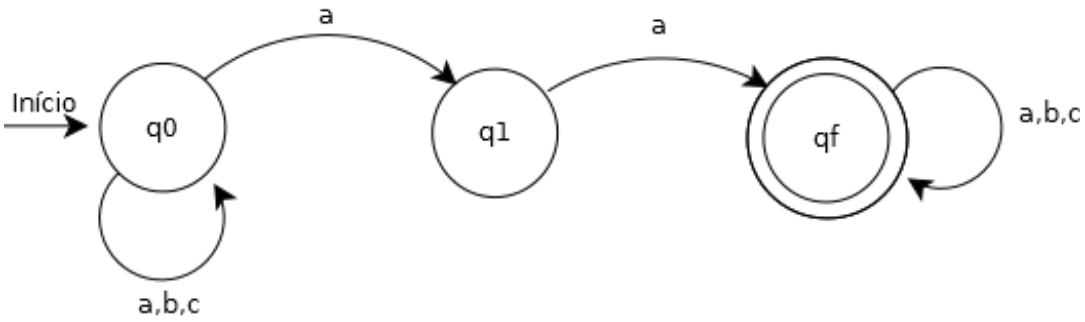

Figura 4 - Exemplo de autômato finito não-determinístico

Inicialmente, o autômato encontra-se no q0 (estado inicial) e irá manter-se no mesmo nó se ler qualquer símbolo do alfabeto, ou poderá, também, deslocar-se para o estado q1 ao ler o símbolo **a,** onde é possível observar o não-determinismo. Na sequência, o estado q1 somente mover-se-á para o estado qf se ler **a**. Após, o estado

qf (estado final) poderá ler nenhum ou infinitos **a**'s, **b**'s e **c**'s, dependendo apenas de sua entrada.

Quando comparado ao autômato finito determinístico, a função de transição é o única característica que difere o AFND do AFD, uma vez que o AFND recebe um símbolo de entrada e um estado como parâmetro (similar a um AFD), contudo poderá ter como retorno um conjunto de zero ou um ou mais estados (ao invés de retornar rigorosamente um único estado, como AFD estabelece). O autômato finito não-determinístico tem a possibilidade de manter-se em inúmeros estados sincronizados, ou seja, existem diversos caminhos possíveis a serem seguidos, definido em uma árvore de opções. Uma entrada **w** somente será aceita se no mínimo um dos caminhos alternativos processar **w**. (MENEZES, 2011; HOPCROFT; ULLMAN; MOTWANI, 2002).

Nessa mesma perspectiva, Hopcroft, Ullman e Motwani (2002) destaca que autômatos finitos determinísticos e autômatos finitos não-determinísticos aceitam rigorosamente as linguagens regulares. Porém, o AFND é mais sucinto e mais fácil de ser projetado. Além disso, há possibilidade de conversão de um AFND em AFD.

Segundo Menezes (2011), o autômato finito não-determinístico pode ser representado formalmente por meio de uma quíntupla $M = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$, na qual:

$Q -$ conjunto finito de estados possíveis do autômato;

$\Sigma -$ alfabeto de símbolos de entrada;

$\delta -$ função programa ou função de transição;

$$\delta\colon \mathrm{Q} \times \Sigma \rightarrow 2^{\mathrm{Q}}$$

$q0 -$ denominado estado inicial, onde $q0 \in Q$;

$F -$ conjunto de estados finais, no qual $(F \subset Q)$.

A representação formal a seguir corresponde ao exemplo de autômato finito não-determinístico:

$$Q = \{q0, q1, qf\}$$
$$\Sigma = \{a, b, c\}$$
$$\delta = \{\delta(q0, a) = q0,$$
$$\delta(q0, b) = q0,$$
$$\delta(q0, c) = q0,$$
$$\delta(q0, a) = q1,$$
$$\delta(q1, a) = qf,$$

$$\delta(qf, a) = qf,$$
$$\delta(qf, b) = qf,$$
$$\delta(qf, c) = qf,$$
$$q0 = \{q0\}$$
$$F = \{qf\}$$

A tabela de transição compatível ao autômato finito não-determinístico da figura 4 é representado na tabela a seguir:

Tabela 2 - Tabela de transição do autômato finito não-determinístico

δ	a	b	c
$\rightarrow * q0$	$\{q0, q1\}$	$q0$	$q0$
$q1$	qf	$-$	$-$
qf	qf	qf	qf

Autômato finito com movimentos vazios

Para proporcionar uma melhor percepção do conteúdo de autômato finito de movimentos vazios (AFMV), tem-se a seguinte representação da expressão regular $(b + c) * aa(a + b + c) *$ em forma de diagrama de transição:

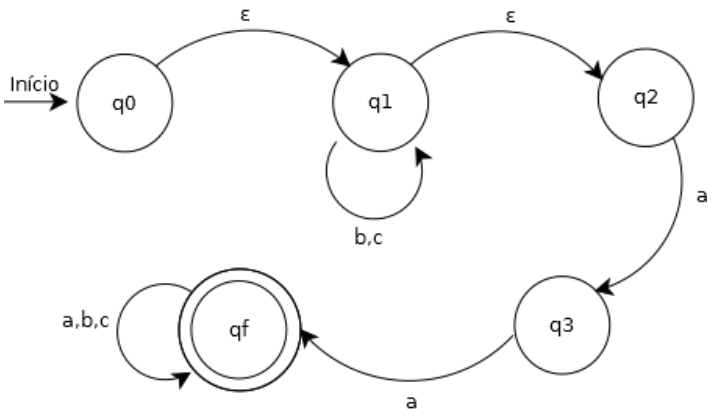

Figura 5 - Exemplo de autômato finito com movimentos vazios

Primeiramente, o autômato encontra-se no estado inicial (q0) e passará para o estado q1 pela transição de movimento vazio. No estado q1, ele poderá manter-se no mesmo estado lendo inúmeros **b**'s ou **c**'s ou mover-se para o estado q2 lendo o símbolo vazio. Posteriormente, no estado q2 e q3, o autômato obrigatoriamente lerá **a**, a fim de atender o formalismo da expressão regular (b + c) ∗ aa(a + b + c) ∗. Já no

estado qf, o autômato poderá ler nenhum ou infinitos **a**'s, **b**'s e **c**'s, dependendo de sua entrada.

Um autômato finito de movimentos vazios possui somente transições com o movimento vazio, quando contraposto a um autômato finito não-determinístico. Essas transições ocorrem eventualmente para uma mudança de estado sem a leitura de um caractere da fita. (MENEZES, 2011).

Nessa perspectiva, assim como os autômato finito não-determinísticos, o autômato finito de movimentos vazios não aumenta o poder de reconhecimento de linguagens que podem ser aceitas por autômatos finitos. Um dos proveitos desse reconhecedor é facilitar algumas criações e demonstrações referentes aos autômatos. (MENEZES, 2011; HOPCROFT, ULLMAN, MOTWANI, 2002).

Segundo Hopcroft, ULLMAN e MOTWANI (2002), a representação formal para um autômato finito de movimento vazio é uma quíntupla ordenada $M = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$, na qual:

$Q -$ conjunto finito de estados possíveis do autômato;

$\Sigma -$ alfabeto de símbolos de entrada;

$\delta -$ função programa ou função de transição;

$$\delta: \mathrm{Q} \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^{\mathrm{Q}}$$

$q0 -$ denominado estado inicial, onde $q0 \in Q$;

$F -$ conjunto de estados finais, no qual ($F \subset Q$).

De acordo com Lima (2006), a função de transição do autômato finito com movimentos vazios é equivalente à função de transição do autômato finito não-determinístico, distinguindo-se somente em relação ao alfabeto de símbolos de entrada, que permite o símbolo ε.

A representação formal a seguir corresponde ao exemplo de autômato finito de movimentos vazios:

$$Q = \{q0, q1, q2, q3, qf\}$$

$$\Sigma = \{a, b, c\}$$

$$\delta = \{\delta(q0, \varepsilon) = q1,$$

$$\delta(q1, b) = q1,$$

$$\delta(q1, c) = q1,$$

$$\delta(q1, \varepsilon) = q2,$$

$$\delta(q2, \mathrm{a}) = q3,$$

$$\delta(q3, \mathrm{a}) = qf,$$
$$\delta(qf, \mathrm{a}) = qf,$$
$$\delta(qf, \mathrm{b}) = qf,$$
$$\delta(qf, \mathrm{c}) = qf\}$$
$$q0 = \{q0\}$$
$$F = \{qf\}$$

A tabela de transição compatível ao autômato finito de movimentos vazios da figura 5 é apresentada na tabela 3.

Tabela 3 - Tabela de transição do autômato finito com movimentos vazios

δ	a	b	c	ε
$q0$	–	–	–	$q0$
$q1$	–	$q1$	$q1$	$q2$
$q2$	$q3$	–	–	–
$q3$	qf	–	–	–
qf	qf	qf	qf	–

2.3 Equivalência de linguagens regulares

Esta seção abordará algumas das conversões e equivalências existentes nas linguagens regulares. Quanto as conversões, existe a possibilidade de conversão entre expressões regulares e autômatos finitos, sendo que estes dividem-se em AFD, AFN e AFMV. (HOPCROFT; ULLMAN; MOTWANI, 2002).

Para um melhor entendimento, a figura 6 demonstra as conversões a serem abordadas neste capítulo:

Figura 6 - Conversões das linguagens regulares

Inicialmente, esta seção abordará a transformação de expressões regulares para autômato finito com movimentos vazios. Em seguida, de autômato finito com movimentos vazios para autômato finito não-determinístico. Na sequência, de autômato finito não-determinístico para autômato finito determinístico. E por final, de

autômato finito determinístico para autômato finito determinístico mínimo.

2.3.1 Equivalência de ER para AFMV

O autor Hopcroft, Ullman e Motwani (2002) explica que a conversão de uma ER em um AFMV demora um período linear. Aho, Lam e Ullman (2008) define que o algoritmo para converter qualquer expressão regular em um AFMV que interprete a mesma linguagem é dirigido por sintaxe, no sentido de que a árvore de derivação é construída recursivamente para as ERs. A cada subexpressão encontrada, é construído um AFMV com um único estado de aceitação. Este algoritmo é denominado algoritmo de McNaughton-Yamada Thompson, que será fundamento a seguir.

Ainda de acordo com Hopcroft, Ullman e Motwani (2002), o primeiro passo para a transformação é desmembrar a ER em subexpressões, visto que a conversão de uma ER para AFMV possui regras básicas que tratam somente de subexpressões com ausência de operadores. Além das regras básicas, existem hipóteses por indução, que são utilizadas para reagrupar as subexpressões intermediarias a fim de construir um único AFMV correspondente a ER.

De acordo com Menezes (2011), as regras básicas de uma expressão regular r com ausência de operadores podem ter somente as seguintes formas:

$$r = \varepsilon$$
$$r = \emptyset$$
$$r = x \ (x \in \Sigma)$$

A fim de proporcionar um melhor entendimento das regras básicas para a construção de um AFMV, a Figura 7 representa os autômatos finitos correspondentes às ERs com ausência de operadores:

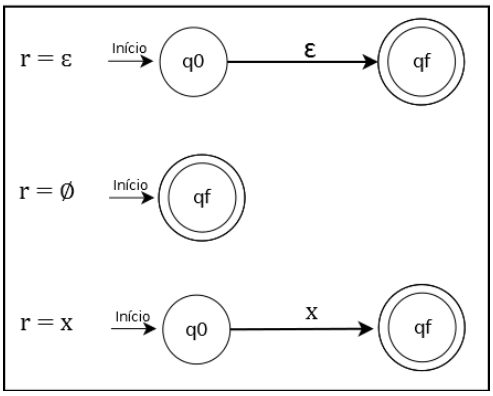

Figura 7 - Base da construção de um AF a partir de uma ER (HOPCROFT et al, 2008)

Em conformidade com Menezes (2011), para agrupar as subexpressões intermediarias existe três casos de indução. Assim, dado r uma ER com $n+1$ operadores, r consegue ser representado pelos três seguintes casos:

$$r = x + y$$

$$r = xy$$

$$r = x^*$$

A Figura 8 demonstra os três casos de indução, representando os autômatos finitos equivalentes às expressões regulares com $n+1$ operadores citados acima:

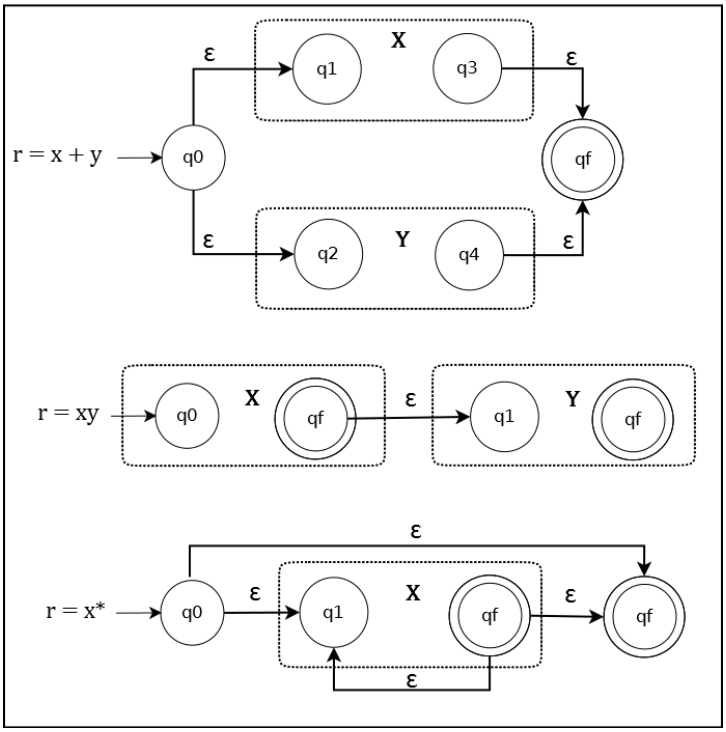

Figura 8 - AF equivalentes às ERs com n+1 operadores (MENEZES, 2011)

Na hipótese de uma ER $a * (aa + bb)$ e com o propósito de demonstrar como se dá a conversão de uma expressão regular qualquer para um autômato finito com movimentos vazios, tem-se o seguinte exemplo:

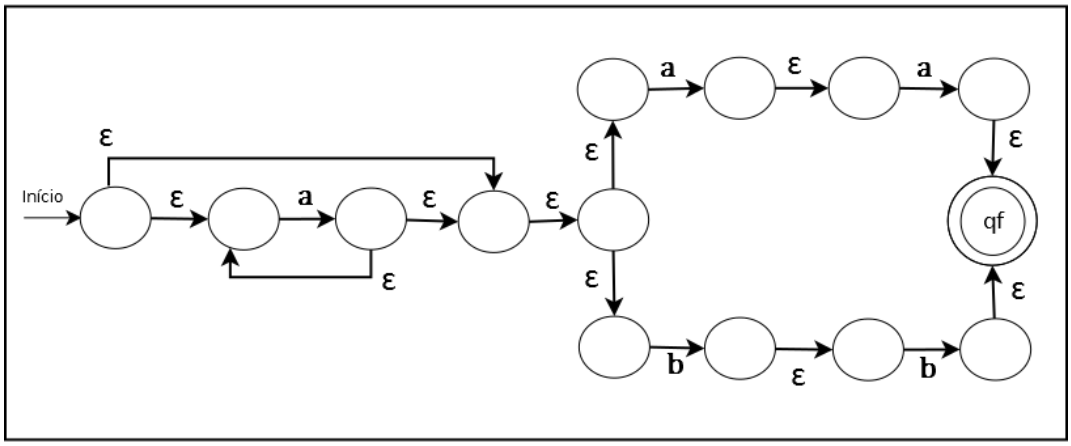

Figura 9 - AFMV construído a partir da ER (MENEZES, 2011)

2.3.2 Equivalência de AFMV para AFND

Consoante entendimento de Menezes (2011), a classe de AFMV é equivalente a classe dos AFND, e como prova de tal entendimento, o autor afirma que a partir de um AFMV N qualquer, existe a possibilidade de construir um AFND M, tal que este reconheça as mesmas sentenças reconhecidas por um autômato finito com movimentos vazios, isto é, M simula N. Nesse mesmo sentido, a demonstração de que, de fato, a simulação acontece, é realizada por meio da indução no tamanho da palavra.

A ideia central do algoritmo de conversão de um autômato finito com movimentos vazios para autômato finito não-determinístico é a construção de uma função de transição isenta de movimentos vazios, onde o conjunto de estados destino de cada transição não vazia é aumentado com todos os estados acessíveis de serem atingidos apenas por transições vazias (MENEZES, 2011).

ENTRADA: um AFMV $N = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$.

SAÍDA: um AFND $M = (\mathrm{Q}', \Sigma', \delta', \mathrm{q0}', \mathrm{F}')$, tal que $M \leftrightarrow N$.

1. $\delta' : Q \times \Sigma \rightarrow 2^{\mathrm{Q}}$ onde:

$$\delta'(\mathrm{q}, \mathrm{y}) = \delta(\{\ \mathrm{q}\ \}, \mathrm{y})$$

2. F' é o conjunto de todos os estados $q \in Q$ tal que:

$$\delta\varepsilon(\mathrm{q}) \cap F \neq \emptyset$$

A fim de proporcionar uma melhor compreensão, a Figura 10 representa um AFMV $N = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$:

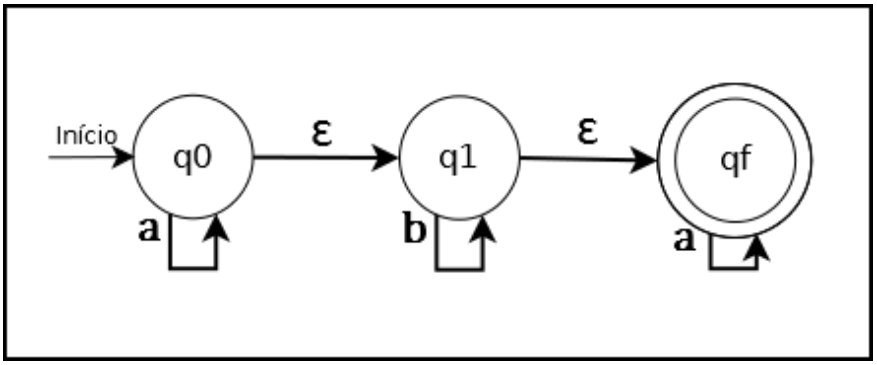

Figura 10 - AFND N (MENEZES, 2011)

O AFND $M = (\mathrm{Q}', \Sigma, \delta', \mathrm{q0}, \mathrm{F}')$ equivalente ao AFMV da Figura 10 pode ser observado na figura 11:

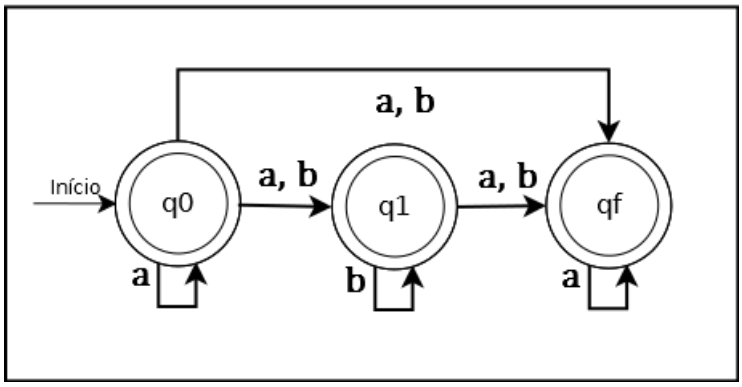

Figura 11- AFND M equivalente (MENEZES, 2011)

2.3.3 Equivalência de AFND para AFD

A classe de autômato finito não-determinístico é equivalente a classe dos autômatos finitos determinístico, o que significa dizer que, a partir de um AFND N qualquer, é possível construir um AFD M que realiza as mesmas simulações, ou seja, M computa N. Embora existam diversas linguagens nas quais o autômato finito não-determinístico seja mais simples de construir do que um autômato finito

determinístico, a concepção e implementação dessa máquina de estado não-determinista é mais custosa e complexa, visto que requer o uso de algoritmos recursivos (*backtraking)* (MENEZES, 2011; HOPCROFT; ULLMAN; MOTWANI, 2002).

Hopcroft, Ullman e Motwani (2002) afirma que, em geral, o AFD dispõe um quantidade de estados próximo a um AFND, entretanto o número de transições normalmente é maior. Contudo, o menor AFD pode ter 2^n estados no pior caso, enquanto o menor AFND tem somente n estados para a mesma linguagem. Segundo Menezes (2011), é, primitivamente, preferível desenvolver uma solução não-determinista e, sobre está, aplicar o algoritmo de conversão para AFD.

O algoritmo de conversão de AFND para AFD basicamente substitui todas as transições não-determinística de um AF, por transições determinísticas para estados novos do AFD em construção. Nesse mesmo sentido, para cada transição não determinística diferente presente no AFND, o algoritmo acrescenta um novo estado e redireciona a transição para o mesmo, repetindo as transições dos estados que seriam atingidos pela realização da transição não-determinística. Esse algoritmo trabalha de forma iterativa, tendo em vista que o processo aplicado por ele pode resultar de novas transições não-determinísticas. O trabalho do algoritmo só termina quando a eliminação do não-determinismo está completa. (RAMOS, 2008).

Objetivando uma melhor compreensão, as Figuras 12 e 13 representam o funcionamento do algoritmo de conversão entre AFND e AFD. A figura abaixo apresenta uma situação AFND:

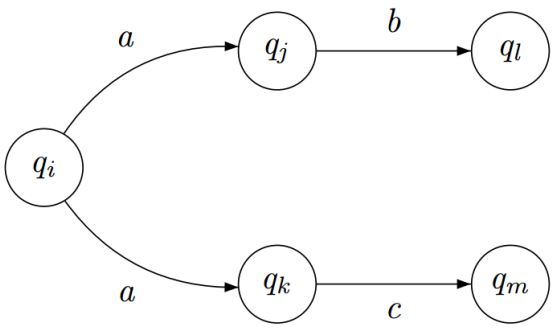

Figura 12 - Situação AFND (RAMOS, 2008)

Em conformidade com Ramos (2008), pressupõe-se que qualquer autômato que apresente, como uma parcela de sua caracterização, uma transição $\delta(\mathrm{qi}, \mathrm{a}) = \{qj, qk\}$, essa será considerada não-determinística. Ao gerar um novo estado $qjqk$, consegue-se fazer com que o autômato alterado aceite a mesma linguagem que o inicial, contudo o autômato será isento de comportamento não-determinístico.

Na Figura 13 é representado de forma ilustrativa a eliminação do não-determinismo. Tanto na Figura 12 quanto na posterior, as versões dos estados representados pelas cadeias ab e ac são idênticas e correspondentes a ql e qm.

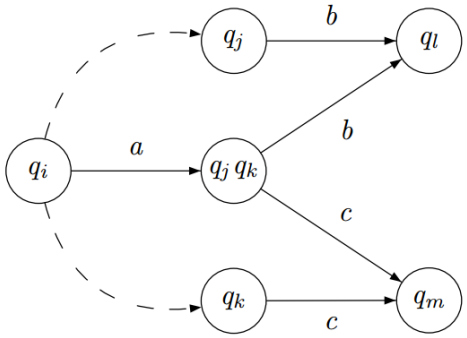

Figura 13 - Situação determinística equivalente à Figura 12 (RAMOS, 2008)

Analogamente, observa-se que, se pelo menos um dos estados qj ou qk for um estado de finalização, o mesmo deverá acontecer com o estado novo, e no caso do exemplo acima, $qjqk$.

Com o propósito de facilitar a absorção da equivalência do AFND e AFD, tem-se na figura 14 um AFND N que representa a ER $(a + b) * aaa$.

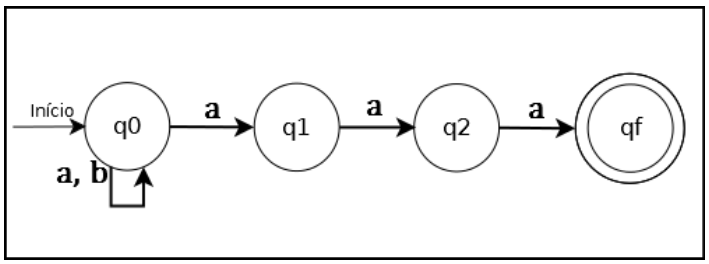

Figura 14 - AFND N (MENEZES, 2011)

O AFD M equivalente ao AFND N da figura 14 pode ser analisado na figura 15.

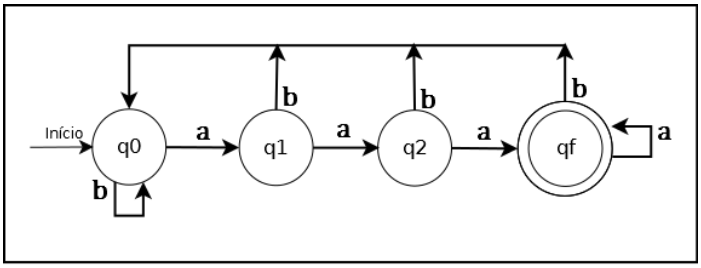

Figura 15 - AFD M equivalente a AFND N

2.3.4 Equivalência de AFD para AFM

A conversão de autômato finito determinístico para autômato finito mínimo tem por objetivo gerar um autômato finito com o menor número de estados possíveis. O AFM é um autômato único, dessa forma, dois AFD's diversos que contém a linguagem idêntica, ao serem minimizados, resultam no mesmo AFM, podendo diferenciar-se apenas na identificação de estados. Destaca-se que tempo de processamento não depende do reconhecimento do autômato, ou melhor, dado um AFD que aceite determinada linguagem terá a mesma duração de processamento de um AFM. Porém, existe uma possível otimização no número de estados, ou seja, a capacidade necessária para armazenar o autômato. (MENEZES, 2011; HOPCROFT; ULLMAN; MOTWANI, 2002).

Ainda de acordo com Aho, Lam e Ullman (2008), a definição de AFM para uma linguagem regular oferecida L, é um autômato finito determinístico $M_{\mathrm{m}} = (\mathrm{Qm}, \Sigma\mathrm{m}, \delta\mathrm{m}, \mathrm{q0m}, \mathrm{Fm})$ tal que **ACEITA ($\boldsymbol{Mm}$)** $= \boldsymbol{L}$ visto que, para algum outro AFD $M = (\mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F})$ tal que **ACEITAM($\boldsymbol{M}$)** $= \boldsymbol{L}$ ocorra:

$$\boldsymbol{Q} \geq \boldsymbol{Q_{\mathrm{m}}}$$

Aho, Lam e Ullman (2008) afirma que para ser possível minimizar um AF, este deverá satisfazer a determinados pré-requisitos:

- Autômato finito que será minimizado deverá ser um AFD;
- Não tolera estados inatingível, ou seja, todos os estados devem ser acessíveis a partir do estado inicial;
- A função de transição deve ser completa, isto é, todo estado precisa conter transições para todos os símbolos do alfabeto.

Conforme explica Martins (2003), o algoritmo de conversão de um AFD para um AFM é basicamente dividido em cinco passos. São eles:

ENTRADA: AFD $N = \mathrm{Q}, \Sigma, \delta, \mathrm{q0}, \mathrm{F}$.

SAÍDA: AFM $M = \mathrm{Q}', \Sigma, \delta', \mathrm{q0}', \mathrm{F}'$, tal que M seja equivalente a N.

PASSO 1: Eliminar os estados inacessíveis:

a) Estabelecer o estado inicial;
b) A cada estado q_{j} apontado, marcar todos os estados q_{k} que podem ser alcançados a partir de $q_{\mathrm{j;}}$
c) Eliminar os estados não apontados.

PASSO 2: Eliminar as transições indefinidas:

a) Incluir um estado novo não-final com transições pra si mesmo, com todos os símbolos do alfabeto de entrada;
b) Alterar todas as transições não previstas por uma transição para o estado novo.

PASSO 3: Determinar os estados equivalentes:

a) Dividir o conjunto de estados Q em dois grupos de equivalência, uma incluindo os estados finais e a outra contendo os demais estados de Q;
b) Dividir iterativamente as classes obtidas até que não se possa obter nenhuma nova classe. Nesta etapa, deve-se considerar que um conjunto de estados q_1,q_2.. e_{j} está na mesma classe de equivalência se todas as transições disponíveis de cada um desses estados levam o autômato para os estados q_{i}, $q_{\mathrm{i+1}}$ ... q_{n} esses últimos estando em uma mesma classe de equivalência.

PASSO 4: Eliminar os estados mortos/inúteis (estados não finais e que a partir deles nenhum estado final é alcançado):

a) Estabelecer os estados finais;
b) Identificar todos os estados q_{k} que alcançam um estado marcado e_{i} por uma transição com algum símbolo de entrada;
c) Eliminar os estados não marcados, isto porque as transições que levam a um estado eliminado são consideradas indefinidas, logo ignoradas.

PASSO 5: Construir $M = \mathrm{Q}', \Sigma', \delta', \mathrm{q0}', \mathrm{F}'$, onde:

Q' – conjunto finito de estados equivalentes conquistados;

Σ' – alfabeto de símbolos de entrada, tal que $\Sigma' = \Sigma$;

δ' – função de transição, delimitada em $Q' \times \Sigma \rightarrow \mathrm{Q}'$, tal que $\delta'\ (\{p\}, x) = \{q\} \leftrightarrow \delta\ \ (p_i, x) = q_i$ é uma transição de N e p_i e q_i são elementos de $\{p\}$ e $\{q\}$, de modo respectivo, onde $\{p\}$ e $\{q\}$ são conjuntos de estados equivalentes;

$\mathrm{q}0'$ – estado equivalente que contém o estado inicial;

F' – conjunto de estados equivalentes $\{q_0\ q_1 \ldots q_n\}$ $(F' \subseteq Q')$ onde algum estado q_i é um estado final de N, isto é, $Q_i \in F$.

A fim de facilitar o entendimento da conversão do AFD N para o AFM M, a Figura a 16 representa um AFD da LR $r = (a + b)(c + d) * e$:

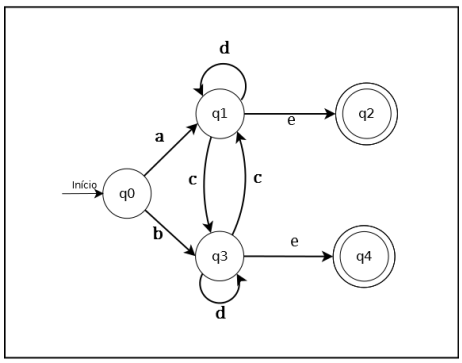

Figura 16 - AFD N (RAMOS, 2008)

A Figura 17 apresenta o AFM da linguagem r, alcançada por meio da conversão do AFD N,que foi obtida através os cinco passos explicados por Martins (2003).

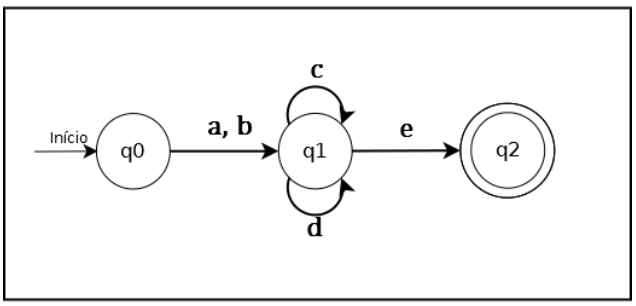

Figura 17- AFM M (RAMOS, 2008)

2.4 Ensino a distância

Segundo Pereira (2010), pode-se definir como sendo ensino a distância (EaD), aquele processo de ensino onde professores e alunos não se encontram geograficamente no mesmo local, no qual o procedimento de ensino aprendizagem se dá através de meios tecnológicos de comunicação. Com o passar do tempo, os meios utilizados foram se alterando ou até se aperfeiçoando, começando pelo correio, depois telefone, fax, televisão, vídeo, CD-ROM até chegar à internet, que foi considerada como um novo formato de ensino a distância, nomeado *e-learning*. Toda essa evolução aconteceu com o intuito de permitir uma interação e participação colaborativa maior para o ensino e aprendizagem (PENTERICH, 2005).

Conforme Bissoni (2012), a EaD começou a ser adotada no Brasil a partir do século XIX, e é um paradigma de ensino que vem crescendo expressivamente no Brasil e no mundo, sendo que essa evolução não se dá apenas às inovações tecnológicas, mas também devido a demanda do mercado. Por consequência, o número de organizações de ensino, tanto particulares quanto públicas, que estão credenciando-se para serem capazes de oferecer algum tipo de ensino ou aprendizagem nesta modalidade, que ainda tem a capacidade de evoluir drasticamente, está aumentando mais a cada dia.

Nesse modelo de ensino, o tutor de um curso de ensino a distância, tem um papel importante na organização do mesmo, principalmente utilizando um conjunto de estratégias pedagógicas com o objetivo de que a experiência dos alunos seja incentivada no ponto de vista da aprendizagem. Os alunos que utilizam esse meio de ensino devem ser motivados a buscarem um papel de investigador, buscando cada vez mais informações do seu interesse, não se limitando a um espaço ou horário especifico. Além do mais, cada aluno pode-se dedicar no seu próprio ritmo de aprendizagem de acordo com suas capacidades e/ou seus interesses. (PEREIRA, 2010).

Em conformidade com Schlemmer (2002), o foco do procedimento educacional está na concepção do conhecimento, na aprendizagem, habilidades e no desenvolvimento de competências, sendo que esses elementos citados formam a cultura da aprendizagem. Nesse mesmo sentido, o uso da tecnologia é considerada uma oportunidade a mais de transmitir conteúdo ou reforça-los.

Os Ambientes Virtuais de Aprendizagem (AVAs) podem ser considerados como sendo um elemento significativo como uma plataforma de *software*, que se fundamenta nas práticas do EaD. Nesse mesmo sentido, o acesso de conteúdo, canais de interação e o suporte oferecido aos participantes em EaD, são disponibilizados através dessas ferramentas. Os AVAs são desenvolvidos tanto por empresas privativas, quanto por instituições educacionais, dentre esses *softwares* privados e *softwares* livres. (SCHLEMMER; SACCOL; GARRIDO, 2007).

De acordo com BrandonHall, 2002; Chute et al. 1999; Khan 2001 e Resenberg 2001, destacado por Lima (2003), existem inúmeras vantagens e desvantagens do EaD. Este trabalho abordará as principais, que serão apresentadas por meio do Quadro 1.

Quadro 1 - Vantagens e desvantagens do EaD (Adaptado de LIMA, 2003)

Ensino a Distância	
Vantagens	**Desvantagens**
Flexibilidade no acesso a aprendizagem	Distanciamento que existe entre formador e os formandos
Economia de tempo	Necessidade de conhecimentos tecnológicos básicos
Tornar-se autodidata	Falta de confiança neste modelo educativo
Atualização rápida de conteúdos	-
Acesso universal a informação	-
Reutilização de conteúdos	-

2.4.1 Ambientes virtuais de aprendizagem

De acordo com Almeida (2003), os AVAs são considerados sistemas computacionais acessíveis pela internet, tendo base através de atividades mediadas pelas Tecnologias da Informação e Comunicação (TICs). A maioria desses sistemas é fundamentada na arquitetura cliente-servidor. Normalmente, o cliente é um navegador *Web*, utilizado para conectar-se as páginas *HyperText Markup Language (HTML)* no servidor, ou seja, uma máquina com *software* específico para isto, com capacidade para implementar, criar e gerenciar páginas dinâmicas de *HTML*, manter base de dados relacionada aos materiais de aprendizagem, classe de usuários e a maioria das informações necessárias para a aplicação (SCHLEMMER, 2002).

Segundo Schlemmer (2002), vários desses sistemas reproduzem a aprendizagem física no ambiente *online*, outros procuram utilizar essa tecnologia para

proporcionar aos alunos novas ferramentas que simplificam a aprendizagem. Sendo que os sistemas AVA procuram suportar uma vasta gama de estilos de aprendizagem distintas, mas com o único objetivo de facilitar o máximo para os discentes. Normalmente, os usuários do AVA são divididos em: estudantes e tutores. Os tutores possuem alguns privilégios a mais no sistema do que os estudantes, como: inserir materiais, acompanhar evolução dos alunos, criar turmas, preparar lista de exercícios, sendo que existem inúmeras funcionalidades adicionais no qual diferenciam-se para cada aplicação.

Em conformidade com Sordi (2015), as ferramentas de interação entre os utilizadores, as ferramentas de controle de usuários, conteúdos e funcionalidades e as ferramenta de edição de conteúdo desfrutado pelos usuários, em síntese, compõe um AVA. A Figura 18 denota uma representação da estrutura fundamental de um AVA.

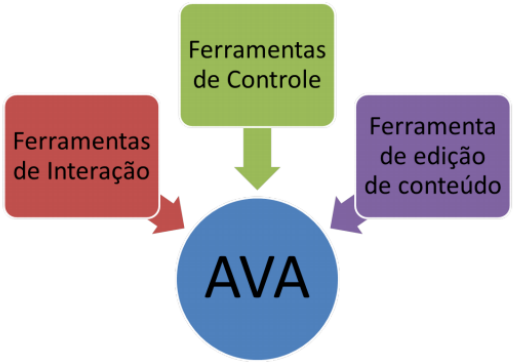

Figura 18 - Ambientes Virtuais de Aprendizagem (SORDI, 2015)

Ainda de acordo com Sordi (2015), os AVAs tem a capacidade de serem utilizados para diferentes fins, e também com propósitos pedagógicos distintos. Um aspecto abordado por Silva et al. (2007), é de ambientes virtuais voltados ao treinamento, que é o aspecto central de análise deste trabalho.

2.4.2 Ambientes Virtuais voltados ao Treinamento

Em conformidade com Silva et al. (2007), os Ambientes Virtuais para Aprendizagem (AVA) podem ser diversificados em dois paradigmas de sistemas: Ambientes Virtuais voltados à Educação (AVE) e Ambientes Virtuais voltados ao Treinamento (AVT). A fim obter um melhor entendimento entre educação e

treinamento, esse mesmo autor pontua que a educação possui um formato mais vasto de aprender, enquanto que o treinamento tem um formato mais especifico de obtenção de conhecimento.

Algumas classes de características para uma melhor diferenciação de AVE e AVT foram identificadas no trabalho de Silva et al. (2007). Esta classificação é abordada na Figura 19, podendo conduzir um parecer melhor de Ambientes Virtuais com o objetivo de aproveitar melhor essa tecnologia, esclarecendo o que, como, quando e o porquê ensinar.

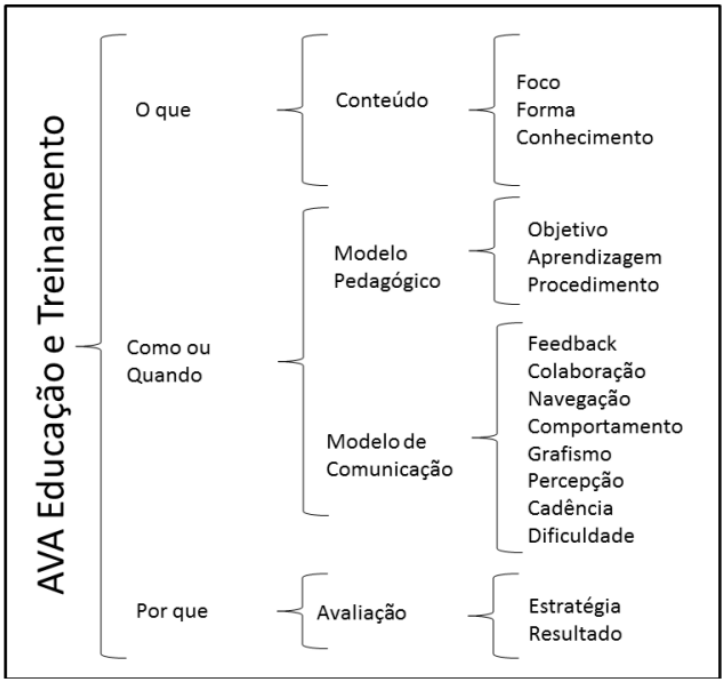

Figura 19 - Caracterização dos AVE e AVT (SILVA et al, 2007)

Para um melhor entendimento as classes contidas na figura acima serão explicadas no decorrer do trabalho, através de quadros comparativos entre AVT e AVE e abordando cada item de sua respectiva classe. São elas: Conteúdo, Modelo Pedagógico; Modelo de Comunicação e Avaliação.

O Conteúdo é a primeira classe a ser abordada, e conforme observa-se na Figura 19, essa classe divide-se em três tópicos: Foco, Forma e Conhecimento.

Quadro 2 - Classe Conteúdo, comparativo entre AVT e AVE (Adaptado)

Conteúdo		
	AVT	**AVE**
Foco	Focam no uso de instruções e operações para aquisição de habilidades.	Utiliza abstrações e compreensão de valores morais.
Forma	Aprendizado é assimilado através da observação ou execução de métodos práticos.	Aprendizados regidos por processos mentais, utilizando técnicas como: aprender através de comparações/definições e reflexões com conceitos e teorias.
Conhecimento	Abordam conteúdos relativos ao experimento industriais e operacionais.	Voltados a conteúdos formais e circulares.

A seguir, a segunda classe será o Modelo Pedagógico, dividindo-se em Objetivo, Aprendizagem e Procedimentos:

Quadro 3 - Classe Modelo Pedagógico, comparativo entre AVT e AVE (Adaptado)

Modelo Pedagógico		
	AVT	**AVE**
Objetivo	Adquirir habilidades específicas e eficiência para a capacitação técnica.	Entendimento e percepção de visões e valores.
Aprendizagem	Foco nas ações e procedimentos técnicos.	Alcançado através da reflexão e tomada de decisões.
Procedimentos	Procedimentos repetitivos, compostos por informações/dados repassados por ordens e comandos.	Pedagógicos não exaustivos (varia) através de explicações, visualizações.

Nesse mesmo sentido, o Modelo de Comunicação é a terceira classe a ser abordada, essa pode segmentar-se em oito tópicos, sendo eles Feedback, Colaboração, Navegação, Comportamento, Grafismo, Percepção, Cadência e Dificuldade.

Quadro 4 - Classe Modelo de Comunicação, comparativo entre AVT e AVE (Adaptado)

Modelo de Comunicação		
	AVT	**AVE**
Feedback	Específico e direto. (Por exemplo: pontuação, status de treinamento).	Abrangente e discursivo.
Colaboração	Geralmente monousuário, visto que aquisição dos objetivos de treinamento (normalmente habilidades físicas e motoras)	Esperado que sejam multiusuários, proporcionando mais interações entre aprendizes, podendo assim

	devem ser obtidos por meio de disciplina do próprio indivíduo.	criar situações de aprendizagem colaborativa.
Navegação	Existem passos pré-estabelecidos (orientação e direcionamento), nestes ambientes o aprendiz deve seguir a sequência dos mesmos.	Geralmente o usuário explora livremente o ambiente.
Comportamento	Ênfase no comportamento específico de alguns objetivos, em busca da fidelidade.	Aproximado ao real em diversas situações, porém, contempla uma ampla gama de situações.
Grafismo	Importante para o treinamento que os objetos sejam representados com realismo, no qual os detalhes são de grande importância para a obtenção de habilidades específicas.	Utilizam objetos caricatos, no qual não é necessário que os aprendizes visualizem o objeto de estudo com fidedignidade para poder entendê-los.
Percepção	Possui a tendência de concentrar esforços em um conjunto mínimo de sentidos, mas em com representação mais confiante.	Quanto maior o número de sentidos humanos envolvidos para a percepção do conteúdo melhor.
Cadência	Cadenciam as atividades de forma real, visando eficácia na realização de tarefas.	Permitem o controle do andamento da comunicação, visto que visam a eficiência.
Dificuldade	Dificuldade computacional reside basicamente na modelagem do fenômeno alvo.	Dificuldade computacional tem sido a modelagem do aprendizado, visto que nestes, o modelo do ambiente, objetos e comportamentos é mais simples.

E por último, a classe conteúdo, que divide-se em dois segmentos, sendo eles: Estratégia e Resultado.

Quadro 5 - Classe Avaliação, comparativo entre AVT e AVE (Adaptado)

Avaliação		
	AVT	**AVE**
Estratégia	Utilizam uma avaliação final, principalmente relacionada a processos manuais, isto é, verificam se a tarefa foi executada com sucesso ou não, mais próxima de uma estratégia de avaliação somativa.	Privilegiam uma avaliação contínua do aprendiz, ou nível a nível, permitindo uma avaliação formativa, analisando principalmente os processos mentais.
Resultado	Busca resultados mais concretos, especialmente relacionados ao condicionamento (ação).	Busca o entendimento dos conceitos pelos aprendizes.

De acordo com Diehl (2004), além das vantagens apontadas, o treinamento por meio de um AVT proporciona que o aprendiz repita as atividades inúmeras vezes, observando e analisando suas respostas. Esse fator estabelece um alicerce de conhecimento muitas vezes superior se comparado a ambiente convencional. Uma outra vantagem é comparar à escalabilidade, um AVT atinge um número maior de usuários treinados em um tempo inferior.

3. TRABALHOS RELACIONADOS

Após a realização de uma pesquisa bibliográfica referente ao uso de ferramentas de correção automáticas e linguagens formais com ênfase em linguagens regulares e autômatos, identificou-se alguns trabalhos relacionados aos temas pesquisados. O autor desse trabalho de conclusão de curso deduz que a existência de alguns trabalhos relacionados às linguagens formais é decorrente do fato do tema ser um assunto discutido há anos. Além disso, as ferramentas corretoras automáticas demonstram ser extremamente práticas e fáceis, tanto para o discente quanto para o docente. Assim, este trabalho pretende abordar dois desses trabalhos, que o autor desta pesquisa conceituou como os de maior importância, sendo eles: URI e Java Formal Language and Automata *Package* (JFLAP).

O trabalho que está sendo desenvolvido pela URI – Universidade Regional Integrada – Campus Erechim é uma ferramenta de ensino e treinamento, disponibilizada em português, inglês e espanhol, que tem como objetivo principal promover a prática de programação e compartilhamento de conhecimento. Essa ferramenta é denominada URI ONLINE JUDGE, e pode ser acessada no meio online através do endereço www.urionlinejudge.com.br (SELIVON, BEZ, TONIN, 2005).

A função do software é, primordialmente, disponibilizar mais de 1000 problemas, divididos em 8 grandes categorias, e tem por finalidade auxiliar os usuários a focarem em um conteúdo especifico. Tais problemas podem ser resolvidos utilizando quatro linguagens de programação: C, C++, Java ou Python. Ao resolver um problema, o usuário irá submetê-lo ao software que, logo após, realizará a correção em tempo real, possibilitando que o usuário saiba se sua resolução está correta ou não. Caso não esteja correta, o software apresentará o erro de forma clara para que o usuário seja capaz de realizar a correção (SELIVON, BEZ, TONIN, 2005).

A interface do URI é limpa e agradável, uma vez que estes são requisitos essenciais para um software de treinamento alcançar o sucesso. Além disso, possui uma arquitetura elástica, isto é, permite que o URI se adapte de acordo com a quantidade de usuários, utilizando múltiplos servidores para a correção das submissões.

O sucesso dessa ferramenta na área da computação é consequente aos estímulos oferecidos pela aplicação, na forma de competição e diferentes tipos de

ranking entre os usuários, como: ranking por problema, por linguagem de programação, ranking exclusivo para cada Universidade. O software ainda os desafia a resolver o maior número de problemas, proporcionando mais conhecimento aos discentes, para que possam aperfeiçoar suas codificações. Além dos benefícios citados acima, em 2013 o URI disponibilizou um módulo único para professores e *coaches*, para que possam criar disciplinas, listas de exercícios e acompanhar a evolução e rendimento dos estudantes em tempo real (SELIVON, BEZ, TONIN, 2005).

Por sua vez, o JFLAP é um pacote de ferramenta gráfica, desenvolvida para a plataforma *desktop*, sendo a primeira versão disponibilizada em 2003. Atualmente, a ferramenta está na versão 8.0, e foi disponibilizada em 2015. Criada por Susan Rodger, Thomas Finley e Peter Linze, o principal objetivo é facilitar o aprendizado linguagens formais e autômatos, utilizado como aparato de auxílio às aulas, como ferramenta de estudo e pesquisa. A ferramenta pode ser obtida no endereço www.jflap.org. A *National Science Foundation* é uma agência governamental dos Estados Unidos independente que sustenta a pesquisa e educação em *todos os campos da* ciência e engenharia, e apoia o JFLAP há anos (RODGER; FINLEY; LINZE, 2006).

O software JFLAP possui inúmeras funcionalidades de experimentação de linguagens formais, sendo elas: a criação e transformação de linguagens livres de contexto, as linguagens enumeráveis recursivamente, a máquina de Mearly, a máquina de Moore, a máquina de Turing, máquina de Turing com múltiplas fitas e os três formalismos das linguagens regulares – a expressão regular, o autômato finito e a gramática regular. Além disso, o JFLAP constrói e testa exemplos para cada uma dessas funcionalidades (RODGER; FINLEY; LINZE, 2006).

Para trabalhar com o JFLAP, inicialmente deverá se escolher o que deseja no menu de entrada, tendo como opções: autômato finito, máquina de Mearly, de Moore, Máquina de Turing, Máquina de Turing com múltiplas fitas, Gramáticas, expressão regular, e linguagens livre de contexto. A escolha do modo de trabalho abre uma tela especifica para cada tipo de estrutura. Para a manipulação dos autômatos finitos, ao iniciar deve-se escolher a opção autômato finito. Logo após, uma janela vazia irá se abrir para criar um autômato, onde o usuário poderá criar estados e transições. Além disso, deverá definir um estado como inicial e final para assim conseguir simular uma entrada no autômato desenvolvido. Assim, a ferramenta retornará se o autômato

reconhece a entrada ou, se for o desejo do usuário, simular a entrada passo a passo para uma análise mais profunda. Ademais, o software possui um módulo para inserir várias entradas, no qual a ferramenta retornará se a lista com palavras foram reconhecidas ou não. A Figura 20 representa a janela de criação de autômatos finitos (RODGER; FINLEY; LINZE, 2006).

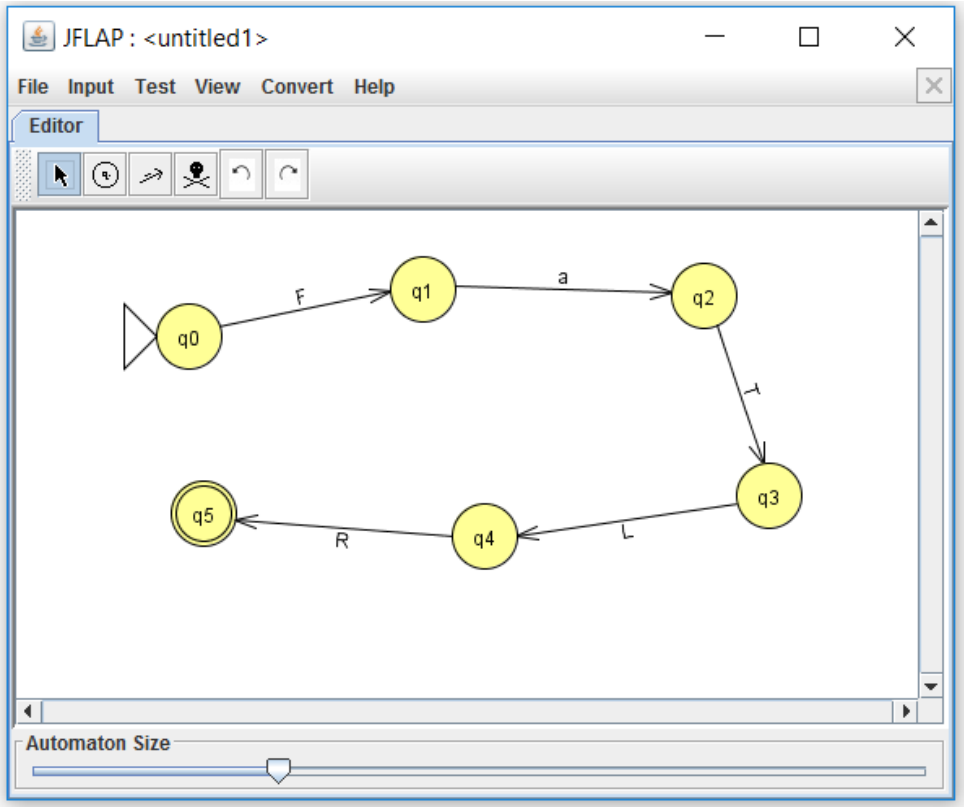

Figura 20 - Janela de criação de autômatos finitos

Além das funcionalidades citadas, o JFLAP permite realizar algumas das conversões das linguagens regulares, tais como a conversão de uma ER para AFMV, de um AFMV para AFD, de um AFD para AFM, de um AFD para ER, de um AFN para uma gramática regular, e vice versa. A Figura 21 apresenta as possibilidades de conversão para as estruturas nas linguagens regulares. (RODGER; FINLEY; LINZE, 2006).

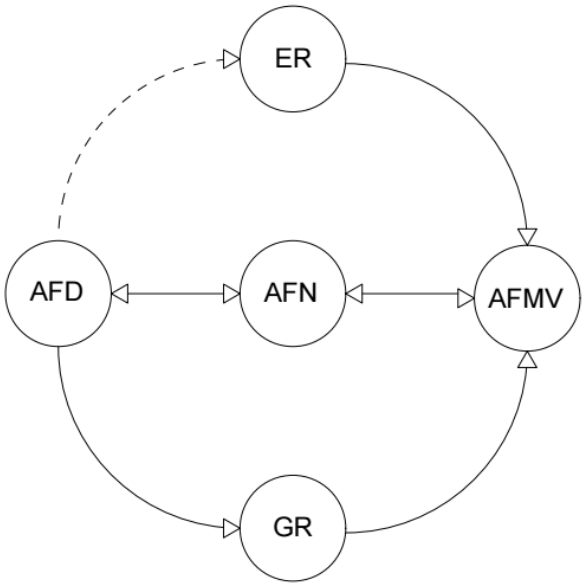

Figura 21 - equivalências das linguagens regulares

Por fim, a documentação do JFLAP é completa, o que facilita a sua utilização. Além disso, é uma ferramenta de código aberto, porém, em caso de modificação, é necessário entrar em contato com os proprietários do software. Pode-se dizer, então, que o JFLAP é uma ferramenta completa na parte das linguagens formais, visto que inúmeras universidades, docentes e pesquisas utilizam-no.

4. UM MÉTODO AUTOMÁTICO PARA CORREÇÃO DE AUTÔMATOS FINITOS

As linguagens regulares são de extrema importância na extensa área da computação, especificamente para a área da teoria da computação. Conforme apresentado no Capítulo 3, alguns trabalhos foram propostos para provar as equivalências das linguagens regulares, porém não se tem o conhecimento de trabalhos que lidem com a correção automática de autômatos finitos. Dessa forma, este capítulo propõe um método de correção automática de autômatos finitos com base em uma expressão regular.

O método proposto consiste em uma lógica que envolve isomorfismo de grafos, busca em profundidade e números aleatórios. O isomorfismo de autômatos consiste em verificar se dois autômatos quaisquer, ao alterar os nomes dos vértices de um deles, ficariam iguais. Já o método de busca em profundidade e geração de palavras aleatórias consiste em percorrer o autômato a fim de gerar palavras aceitas por esse formalismo. O processo de correção automática de autômatos necessita, primeiramente, do autômato a ser corrigido, e que este respeite todos os requisitos de um sistema de estados finitos, isto é, tenha um estado inicial e pelo menos um estado final. Além disso, necessita de uma expressão regular, que servirá como resposta para a correção do autômato. Como foi citado na seção 2.3 todo autômato finito tem uma expressão regular equivalente e do mesmo modo toda expressão regular tem um autômato finito equivalente. Ao realizar o processamento, terá um retorno, se o autômato está correto ou incorreto. A Figura abaixo demonstra as entradas, o processamento e o retorno preciso para o aluno, de modo a facilitar a compreensão.

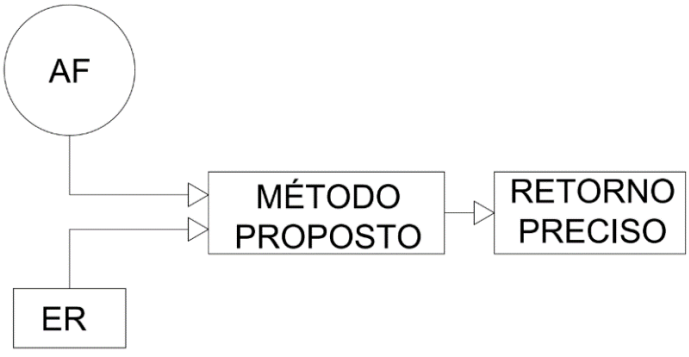

Figura 22 - Arquitetura do método proposto

O método proposto consiste em dois testes, denominados teste de isomorfismo e teste de aceitação, que serão explicados minuciosamente no decorrer teste capítulo. Após a realização dos testes, existem duas possibilidades de retorno, a correta e a incorreta. Caso seja incorreto, apresentará um retorno preciso, isto é, auxílios coerentes que têm por objetivo facilitar a correção do autômato pelo aluno. A fim de facilitar a compreensão do método proposto, a Figura 20 apresenta o processo de correção automática desenvolvida neste trabalho:

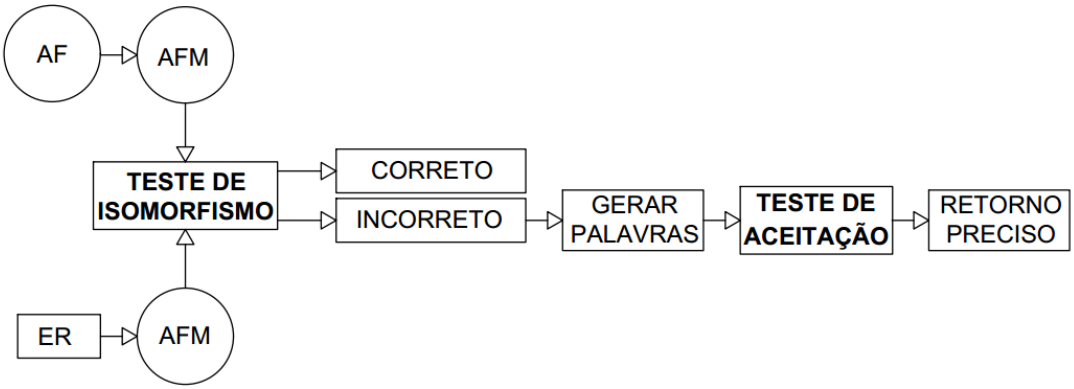

Figura 23 - Método proposto

O AF é o autômato finito a ser corrigido e a ER é a expressão regular resposta, ambas passarão por conversões a fim de realizar o teste de isomorfismo. Ao realizar o teste de isomorfismo, o método retorna se o autômato a ser corrigido está correspondente à expressão regular resposta, isto é, correto. Ou retorna se não está correspondente a expressão regular resposta, ou seja, incorreto. Se o retorno for incorreto, ele irá gerar palavras a fim de realizar o teste de aceitação. o teste de aceitação é realizado para analisar onde pode ocorrer o erro do autômato a ser corrigido, para assim, ter um retorno ao aluno para auxiliá-lo, proporcionando uma correção mais simples.

4.1 Teste de Isomorfismo

A Figura 24 esquematiza o teste de isomorfismo, de modo a proporcionar uma melhor compreensão:

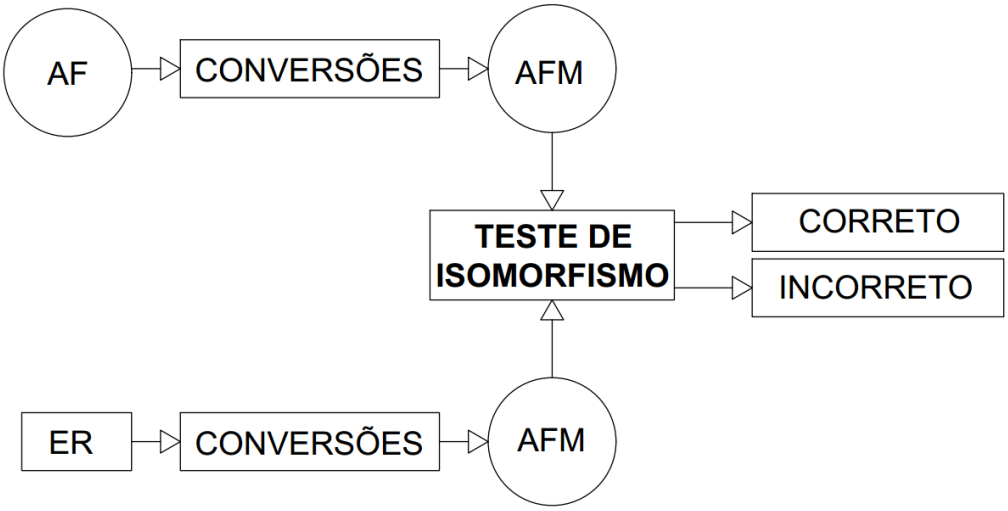

Figura 24 – Representação do teste de isomorfismo

Para a realização do primeiro teste, o autômato a ser corrigido e a expressão regular resposta passarão por um pré-processamento, a fim de obter dois autômatos finitos mínimos. O pré-processamento consiste, basicamente, em aplicar os métodos de equivalência das linguagens regulares, apresentados na seção 2.3 deste trabalho. Abaixo será apresentado as conversões utilizadas no método proposto deste trabalho e o motivos no qual foram utilizadas.

ER para AFMV

A primeira conversão utilizada neste método é a transformação de uma expressão regular para um autômato finito de movimento vazio, como citado na seção 2.3.1. Essa conversão é dada pelo algoritmo de McNaughton-Yamada Thompson e sua realização é necessária devido ao fato de ser o algoritmo que transforma automaticamente uma ER para um AFMV. Além disso, o tempo de execução deste método é linear.

AFMV para AFND

Para a conversão de um AFMV para um autômato finito não determinístico, citada na 2.3.2, basicamente remove-se as transições vazias, lembrando que um AFMV é considerado um AFND. É necessário realizar essa conversão em razão de que o método de conversão de um AFND para um AFD, que é o próximo passo do método, precisa ter como entrada um AFND

AFND para AFD

Esta conversão é necessária em razão que para a conversão de um AFD para um AFM, a entrada é um AFD, e é o próximo passo do método. Além disso, Menezes (2011) afirma que é preferível desenvolver uma solução não determinista e, sobre esta, aplicar o algoritmo de conversão para AFD, visto que é mais fácil desenvolver um AFND do que um AFD. Essa conversão é citada na seção 2.3.3.

AFD para AFM

A conversão de um AFD para um autômato finito mínimo, citada na seção 2.3.4, é necessária pelo motivo de que um AFM é único, ou seja, para cada expressão regular existe um único autômato finito mínimo. Logo, dois AFM quando comparados, podem ser admitidos como isomórficos, diferenciando-se somente na identificação de estados (MENEZES, 2002).

Essas conversões apresentadas acima serão utilizada para realizar o teste de isomorfismo, tanto na parte do autômato a ser corrigido quanto na expressão regular resposta. As conversões que serão utilizadas dependem de qual tipo do autômato será corrigido, lembrando que este pode ser AFMV, AFND ou AFD. Por exemplo, se o autômato a ser corrigido for um AFMV, ele passará por passará por três conversões, de AFMV para AFND, de AFND para AFD e de AFD para AFM. Se for um AFND, passará por duas conversões, de AFND para AFD e de AFD para AFM e se for um AFD passará por uma única conversão, de AFD para AFM. É necessário fazer todas as conversões a fim de obter um autômato finito mínimo, visto que o autômato finito mínimo é considerado único, isto é, para cada expressão regular existe somente um único autômato finito mínimo. Assim, ao obter dois autômatos finitos mínimos é possível realizar o teste de isomorfismos entre eles.

Segundo Dharwadker e Tevet (2009), dois grafos são ditos isomorfos entre si se existir uma correspondência entre os seus vértices e arestas, de tal maneira que pode-se alterar os nomes dos vértices de um deles para que os dois grafos fiquem iguais. Ainda, um autômato pode ser considerado um grafo, visto que possui as mesmas características, ou seja, os estados de um autômato equivalem aos vértices de um grafo, assim como as transições correspondem às arestas.

O método para verificar se dois autômatos são isomórficos é recursivo, isto é, rotinas (funções, métodos) que direta ou indiretamente chamam a si mesmas (ALVES,

2006). Para facilitar a compreensão de como acontece o funcionamento do método será explicado o passo a passo do método:

1. O método verifica se os dois autômatos a serem comparados são autômatos finitos determinísticos.
2. O método verifica se os dois autômatos possuem a mesma quantidade de estados. Se não possuírem, o método já retorna que os dois não são isomórficos, e se sim ele vai para o próximo passo,
3. Obtém um estado do autômato 1 e analisa suas transições comparando-as com um estado do autômato 2, lembrando que sempre irá iniciar com o estado inicial. Se possuir a mesma transição, verifica se é o mesmo caractere, e se não for, o método retorna que não é isomórfico, e caso seja, vai para o próximo passo.
4. O método ficará em um loop entre a etapa três e quatro, até passar por todos os estados possíveis do autômato 1, sempre comparando com o autômato 2.

4.2 Teste de Aceitação

A Figura 25 esquematiza o teste de aceitação, de modo a proporcionar uma melhor compreensão:

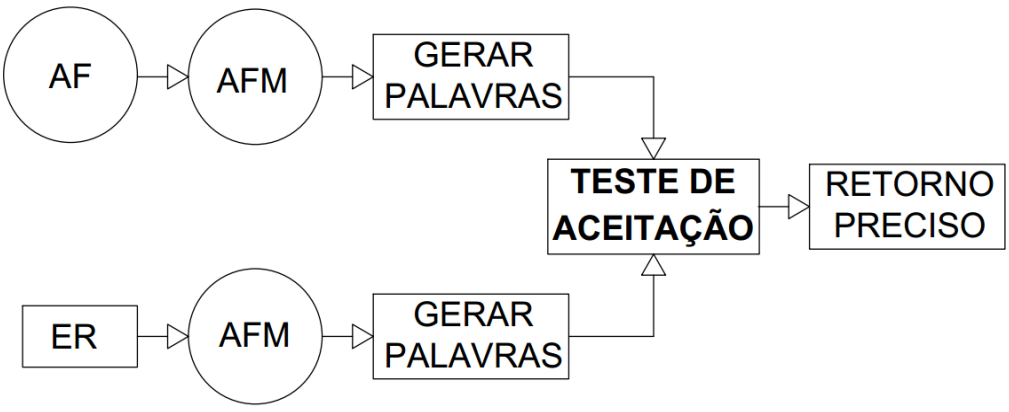

Figura 25 – Representação do teste de aceitação

O teste de aceitação só irá acontecer se o autômato a ser corrigido não for equivalente à expressão regular resposta. Essa verificação de equivalência ocorre no teste de isomorfismo, e, falando em conceitos de programação, o teste de aceitação só ocorre se o método do teste de isomorfismo retornar falso. O teste de aceitação ocorre para demonstrar um retorno preciso para o aluno, visto que o teste de isomorfismo já verifica se o autômato a ser corrigido é equivalente a expressão regular resposta. Esse teste basicamente ocorre para tentar verificar onde o autômato a ser corrigido está errado, a fim de retornar para o aluno que submeteu o autômato auxílios, para que este consiga corrigir e submeter novamente.

O algoritmo do teste de aceitação irá, basicamente, gerar palavras e testá-las no autômato a ser corrigido e no autômato equivalente a expressão regular correta. Essas palavras serão geradas por meio dos métodos de busca em profundidade e método aleatório que serão explicados a baixo.

Busca em profundidade

É considerado um método recursivo, classificado como um método exaustivo ou de força bruta. Utiliza uma técnica nomeada de retrocesso cronológico, isto é, o método irá retornar na árvore de busca, quando um caminho sem saída é encontrado. Em outras palavras, o método segue cada caminho até sua maior profundidade antes de seguir para o próximo caminho. Assim que chegar na folha e não tiver mais caminhos para seguir, a busca retrocederá ao primeiro nó anterior que tenha um caminho não explorado.

Método aleatório

Este método irá gerar cem palavras de forma aleatória com base em um autômato, bem como utilizará a classe *java.util.Random* do Java, uma vez que a ferramenta de apoio a treinamento a linguagens regulares é desenvolvida em Java. Todas as tomadas de decisões do método serão aleatórias, e este ficará em loop até gerar cem palavras diferentes. O passos que o método aleatório segue são basicamente:

1. Obter um estado e as transições deste estado, lembrando que inicialmente o método seleciona o estado inicial;

2. Entrar em um loop, e somente sair deste se uma palavra for adicionada na lista, lembrando que a palavra não pode ser repetida;

3. Verificar se o estado atual, isto é, o estado que o algoritmo se encontra, é final. Se for, ele aleatoriamente verificará se deseja parar ou continuar, e se o resultado for parar, ele irá verificar se a palavra está contida na lista de palavras e caso esteja, não adicionará e voltará para início do método. Se o resultado for continuar, o método irá adquirir as transições e, de forma aleatória, irá escolher uma delas para ser seguida e o estado atual passará a ser o estado destino da transição. Retornará ao início do passo 3.

Por sua vez, as etapas que o teste de aceitação segue são:

1. Primordialmente, o algoritmo obtém o autômato finito mínimo equivalente ao autômato finito a ser corrigido e converte para uma expressão regular equivalente. Desta forma, tem-se uma expressão regular equivalente ao autômato a ser corrigido e uma expressão regular resposta. Essa conversão é necessária, uma vez que é a expressão regular que irá verificar se a linguagem é finita ou infinita;

2. Logo em seguida, o método verifica se as expressões regulares são infinitas ou finitas, lembrando que se houver um asterisco, a expressão regular é infinita, e se não houver um asterisco, a expressão regular é finita. Se a expressão regular foi finita, ela entrará na condição para realizar o método de busca em profundidade para gerar todas as palavras possíveis do autômato, se ela for infinita, ela entrará na condição para realizar o método aleatório a fim de gerar cem palavras;

3. Ao gerar palavras tanto do autômato finito a ser corrigido quanto da expressão regular resposta, utilizando os dois métodos citados acima, busca em profundidade ou de forma aleatória, teremos duas listas de palavra;

4. O método irá obter a lista de palavras geradas pelo autômato a ser corrigido e testar se o autômato equivalente a expressão regular resposta reconhece todas as palavras geradas e vice versa, ou seja, irá obter a lista de palavras geradas pela expressão regular resposta e testar se o autômato a ser corrigido reconhece todas as palavras geradas. Estes dois testes serão feitos a fim de obter o retorno preciso para o aluno. Podendo ter como exemplos de retorno preciso ao aluno: o autômato não reconhece determinada palavra que deveria reconhecer, ou o autômato reconhece palavra a mais, isto é, palavras que não deveriam, mas estão sendo reconhecidas.

4.3 Possibilidades de retorno

Nesta seção serão apresentados todos os retornos possíveis que o método proposto apresentará para o discente, assim como os retornos que o autômato está correto ou incorreto. Caso o autômato a ser corrigido seja incorreto, o método apresentará além do retorno incorreto, auxílios para que a correção ocorra da maneira mais simples possível. Serão apresentados cinco possibilidades de retorno, sendo elas:

1. O autômato a ser corrigido reconhece somente as mesmas palavras que o autômato equivalente à expressão regular correta;
2. O autômato a ser corrigido reconhece todas as palavras geradas com base no autômato equivalente à expressão regular correta, porém reconhece outras palavras que não deveriam ser reconhecidas;
3. O autômato a ser corrigido reconhece algumas palavras geradas; baseado no autômato equivalente à expressão regular correta;
4. O autômato a ser corrigido reconhece algumas palavras geradas a partir do autômato equivalente à expressão regular correta, mas também reconhece outras palavras que não deveriam ser reconhecidas;
5. O autômato a ser corrigido não reconhece nenhuma das palavras geradas segundo o autômato equivalente à expressão regular correta. E reconhece outras palavras que não deveriam ser reconhecidas.

Os retornos serão demonstrados graficamente por meio de operações com conjunto, a fim de facilitar a compreensão. Nas figuras a seguir, o ‘Aa’ representará o autômato a ser corrigido e o ‘As’ o autômato equivalente à expressão regular resposta. Além disso, demonstrarão quais dos dois testes do método proposto define o retorno para o discente. Ainda, denotará quais serão os tipos de mensagens que serão retornadas ao usuário. As possibilidades de retorno serão citadas e explicadas conforme a ordem acima.

Possibilidade 1

O autômato a ser corrigido reconhece somente as mesmas palavras que o autômato equivalente à expressão regular correta reconhece.

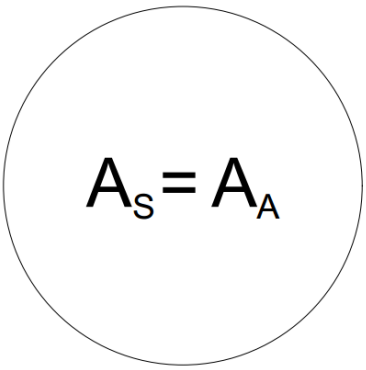

Figura 26 – Possibilidade 1

A Figura 26 representa que o autômato a ser corrigido Aa reconhece somente as mesmas palavras que o autômato equivalente a expressão regular As. Logo, baseado no teste de isomorfismo que utiliza o método de isomorfismo de grafos, o autômato a ser corrigido está equivalente à expressão regular correta. Este é o único retorno correto que o método proposto trará ao aluno. A mensagem que a ferramenta dispara ao discente será que o autômato submetido é equivalente à expressão regular, baseado no isomorfismo de grafos.

Possibilidade 2

O autômato a ser corrigido reconhece todas as palavras geradas com base no autômato equivalente a expressão regular correta, porém reconhece outras palavras que não deveriam ser reconhecidas.

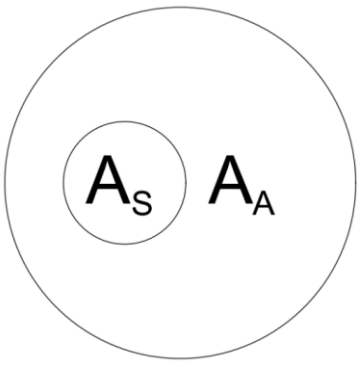

Figura 27 – Possibilidade 2

A Figura 27 representa que o autômato a ser corrigido reconhece todas as palavras possíveis do autômato equivalente a expressão regular, porém reconhece outras palavras que não deveriam ser reconhecidas. Baseando-se no teste de isomorfismo, que utiliza o método de isomorfismo, pôde-se identificar que os dois autômatos não são isomórficos. Assim, irá para o teste de aceitação, onde será verificado em qual parte do autômato pode estar ocorrendo o erro, em razão de sempre facilitar o retorno ao aluno para que ele possa corrigir de maneira mais simples. A mensagem que a ferramenta mostrará ao aluno será que determinada palavra reconheceu no autômato, mas não deveria ser reconhecida.

Possibilidade 3

O autômato a ser corrigido reconhece somente algumas palavras geradas baseada no autômato equivalente a expressão regular correta.

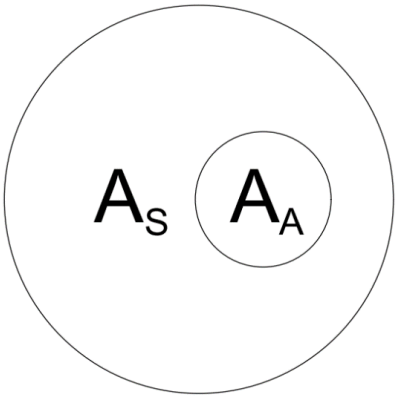

Figura 28 – Possibilidade 3

A Figura 28 representa que o autômato a ser corrigido reconhece somente algumas palavras geradas com base no autômato equivalente à expressão regular. Conforme o teste de isomorfismo, que utiliza o método de isomorfismo, os dois autômatos não são considerados isomórficos. Desta forma, ocorrerá o teste de aceitação, que irá analisar onde ocorre o erro, a fim de obter o melhor retorno possível para o aluno. A mensagem que o método proposto apresentará ao aluno será que o autômato submetido deveria reconhecer tal palavra, mas não a reconhece, sendo que a palavra foi gerada com base no autômato equivalente a expressão regular correta.

Possibilidade 4

O autômato a ser corrigido reconhece algumas palavras geradas a partir do autômato equivalente a expressão regular correta, mas também reconhece outras palavras que não deveriam ser reconhecidas.

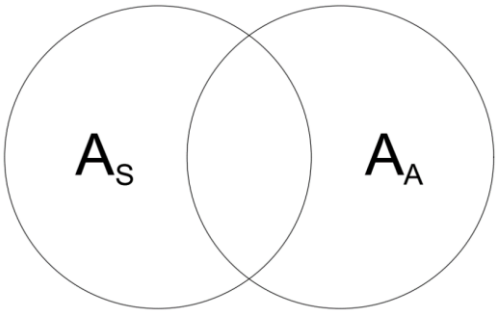

Figura 29 – Possibilidade 4

A Figura 29 representa que o autômato a ser corrigido reconhece algumas das palavras do autômato equivalente a expressão regular, mas também reconhece outras palavras que não deveriam ser reconhecidas. Com base no teste de isomorfismo, que utiliza o método de isomorfismo, afirma-se que os dois autômatos não são isomórficos. Assim, o método irá para o teste de aceitação com o propósito de analisar onde pode ter ocorrido o erro, para retornar o aluno auxílios de modo que este possa corrigir o autômato e submeter novamente. A mensagem que o método trará para o aluno será de que o autômato submetido deveria reconhecer tal palavra que foi gerada pelo autômato equivalente à expressão regular, mas não está reconhecendo. Assim como o autômato feito por ele também reconhece algumas palavras que não deveriam ser reconhecidas.

Possibilidade 5

O autômato a ser corrigido não reconhece nenhuma das palavras geradas segundo o autômato equivalente a expressão regular correta. E reconhece outras palavras que não deveriam ser reconhecidas.

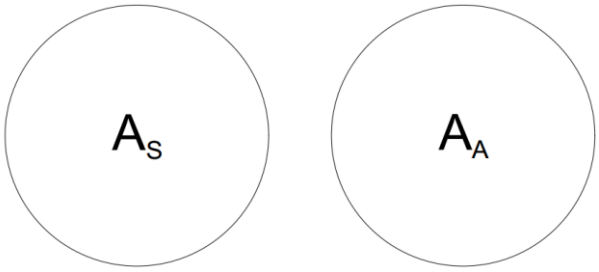

Figura 30 – Possibilidade 5

A Figura 30 representa que o autômato a ser corrigido não reconhece nenhuma das palavras geradas pelo autômato equivalente à expressão regular e reconhece outras palavras que não deveriam ser reconhecidas. Com base no teste de isomorfismo, que utiliza o método de isomorfismo, pode-se concluir que os dois autômatos não são isomórficos. Logo o método desloca-se para o teste de aceitação com o objetivo de analisar onde ocorre o erro para apresentar auxílios coerentes para o aluno. A mensagem que será apresentado ao aluno será de que o autômato submetido não reconhece tal palavra gerada pelo autômato equivalente a expressão regular e deveria reconhecer. Além disso, o autômato submetido reconhece palavras que não deveriam ser reconhecidas.

5. DESENVOLVIMENTO DA FERRAMENTA

Neste Capitulo, inicialmente será apresentado a arquitetura da ferramenta proposta. Em seguida será demonstrado os pacotes, as classes, as interações de interfaces do sistema proposto. E por fim, será detalhado o método de correção do autômato, visto que é o processo principal da ferramenta.

5.1 Arquitetura da ferramenta

O método de correção de autômatos proposto foi desenvolvido através do paradigma orientado a objetos. A análise e o projeto da ferramenta proposta foram realizados com o suporte da linguagem UML e da linguagem de programação Java. Isto porque a ferramenta *Java Formal Language Automata Package* (JFLAP) é desenvolvida na mesma linguagem de programação, além de a ferramenta desenvolvida neste trabalho, denominada FaTLR (Ferramenta de Apoio ao Treinamento de Linguagens Regulares), utilizar alguns métodos de conversões do JFLAP. A Figura 31 representa a arquitetura da ferramenta proposta.

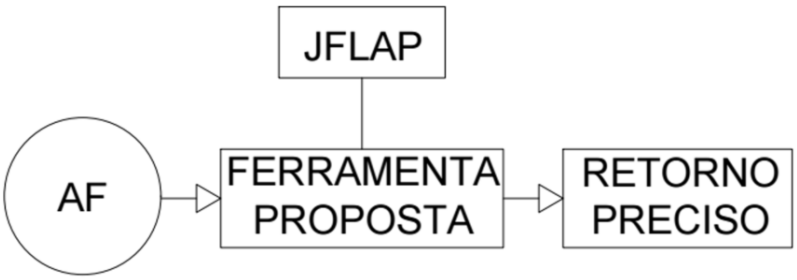

Figura 31 - Arquitetura da ferramenta proposta

A ferramenta proposta trabalha com uma interface utilizando o componente Java Swing, considerado um componente para construir interfaces gráficas na linguagem Java (ORACLE, 2016), onde recebe um autômato finito como entrada, podendo ser autômato finito de movimentos vazios, autômato finito não-determinístico ou autômato finito determinístico. A ferramenta proposta com o auxílio da biblioteca do JFLAP irá realizar as conversões necessárias para assim corrigir o autômato submetido, a fim de obter um retorno preciso.

A ferramenta proposta possui dois módulos, Módulo Aluno e Módulo Professor. Ao escolher o Módulo Aluno, a ferramenta apresentará as opções “Abrir Lista”,

“Exercício Individual” e “Sair”, já no Módulo Professor, terão as opções “Criar Lista”, “Abrir Lista”, “Verificar Respostas” e “Sair”.

5.2 Análise e projeto

A fase de análise e projeto da ferramenta foi realizada com o suporte dos diagramas provenientes da UML. A estrutura dos módulos foi descrita pelos diagramas de classes e de pacotes, enquanto que para a especificação comportamental e integração dos módulos foram utilizados os diagramas de sequência.

Para facilitar a compreensão das funcionalidades do discente e do professor, o diagrama de casos de uso é apresentado na Figura 32. O sistema possui nove casos de uso, sendo que cinco serão por parte do discente e os outros quatro por parte do professor. Os casos de uso definidos para o discente são: Verificar lista de exercício, Salvar lista, Escolher um exercício, Criar autômato e Submeter autômato. Já por parte do professor, os casos de uso são: Criar lista de exercício, Salvar lista, Editar lista de exercício e Verificar lista de exercício.

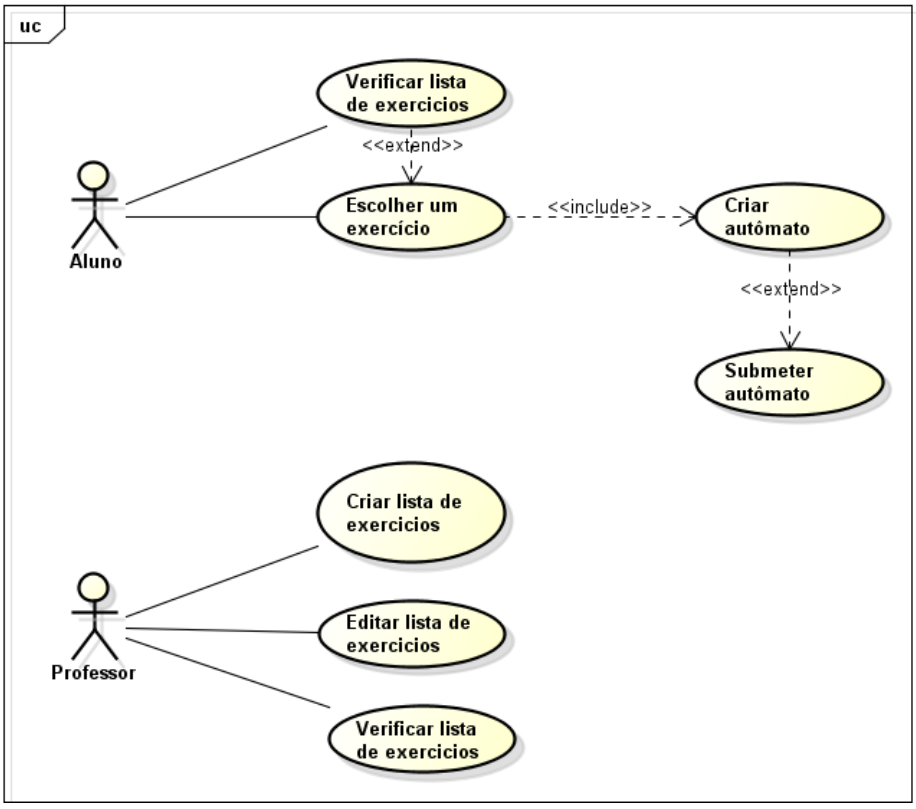

Figura 32 - Casos de uso do FaTLR

Quadro 6 - Casos de uso da FaTLR

Aluno:
Verificar lista de exercício: neste caso de uso o aluno abre a lista de exercícios e verifica os exercícios feitos. **Escolher um exercício:** neste caso de uso o discente poderá escolher um exercício da lista que o professor criou ou escolher um exercício aleatório. **Criar autômato:** o aluno irá criar um autômato finito desejado no espaço pré-estabelecido. Lembrando que o autômato é obrigado ter um único estado inicial, e pelo menos um estado final. E para realizar as transições é necessário selecionar o estado origem e o estado destino. **Submeter autômato:** após o aluno ter escolhido um exercício e criado o autômato, ele poderá submeter o mesmo para correção neste caso de uso. O software retornará, então, se o autômato submetido está correto ou incorreto, visto que se tiver incorreto, mostrará o erro da forma mais simples.
Professor:
Criar lista de exercício: o professor poderá montar quantas listas de exercícios necessitar para as turmas que criou. **Editar lista de exercício:** o professor deverá abrir uma lista de exercício e poderá alterá-la. **Verificar lista de exercício:** o professor verifica a lista de exercício que o aluno resolveu.

A Figura 33 apresenta os pacotes que compõem o projeto da ferramenta FaTLR, em conformidade com a arquitetura apresentada na Figura 31. As conversões utilizadas nos autômatos finitos encontram-se no pacote **algoritmos**. Já o pacote **modelo** contém os objetos utilizados no sistema. O pacote **visão** engloba todas as interfaces que o FaTLR possui e o pacote **run** possui a classe principal, onde ela será executada para compilar o sistema. Já o pacote **dao** contém a classe de persistência de dados. E por fim, o pacote **visao.icons** têm todos os ícones utilizados nas interfaces.

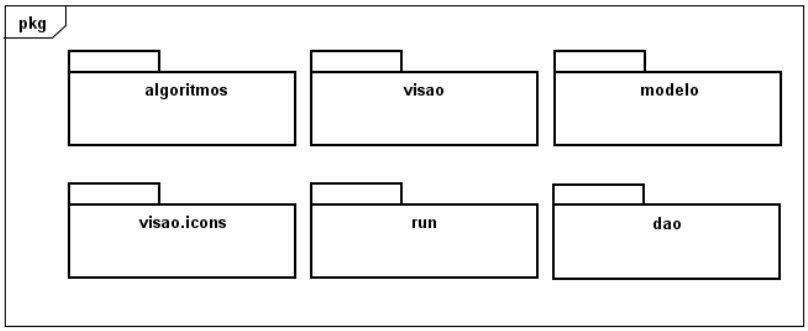

Figura 33 - Pacotes do FaTLR

Com a finalidade de completar os pacotes apresentados acima, todas as classes e métodos utilizados no desenvolvimento da ferramenta FaTLR serão retratados a seguir. A ferramenta FaTLR possui quatro pacotes contendo classes

utilizadas para o desenvolvimento da ferramenta, sendo elas: algoritmos, dao, modelo, run. Já os outros dois pacotes (visao e visao.icons) não contém classes.

As classes serão demonstradas de acordo com seus respectivos pacotes. Primeiramente, será apresentado o pacote **modelo**, sendo que neste pacote só contém os objetos que serão utilizado no sistema, não contendo com nenhum método. A Figura abaixo apresenta as classes do pacote **modelo**.

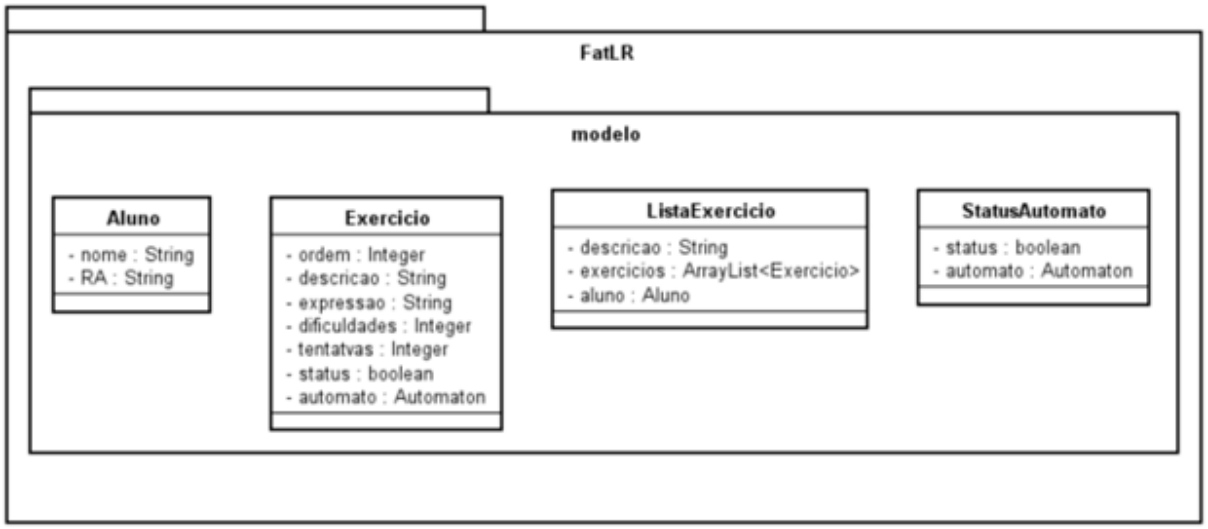

Figura 34 - Diagrama de classes do pacote modelo

Seguidamente, será denotado o pacote **dao** e o pacote **run**, os dois serão apresentados juntos na Figura 35 visto que cada pacote contém uma única classe. A classe contida no pacote dao possui os métodos de gravação e a leitura dos arquivos que serão utilizados no sistema. Já o pacote run, possui a classe responsável para executar o sistema.

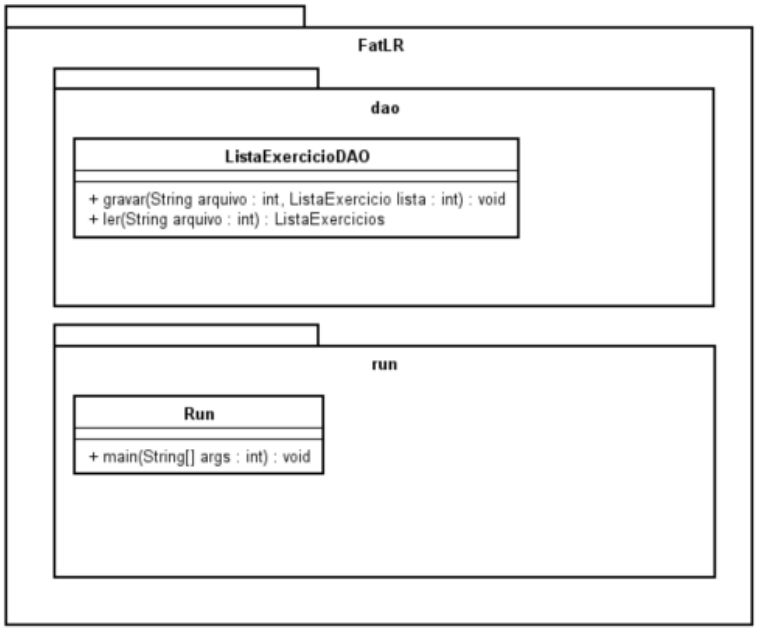

Figura 35 – Diagrama de classe do pacote dao e run

Por fim, o pacote **algoritmos** contém todos os métodos de conversões de autômatos, gerações de palavras e comparações utilizadas na correção do autômato submetido pelo aluno. A Figura 37 denota as classes do pacote **algoritmos**.

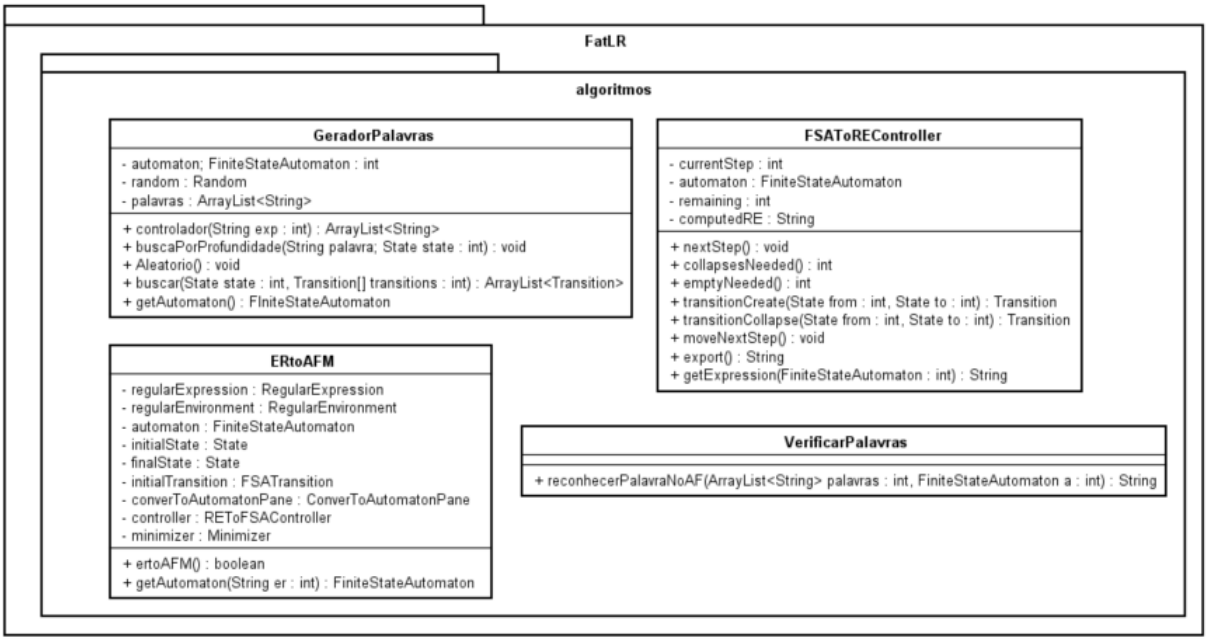

Figura 36 - Diagrama de classes do pacote algoritmo

Para facilitar a compreensão de como acontece as navegações entre as interfaces utilizadas no sistema FaTLR, a Figura 37 apresenta um diagrama de máquina de estados, demonstrando as referidas navegações.

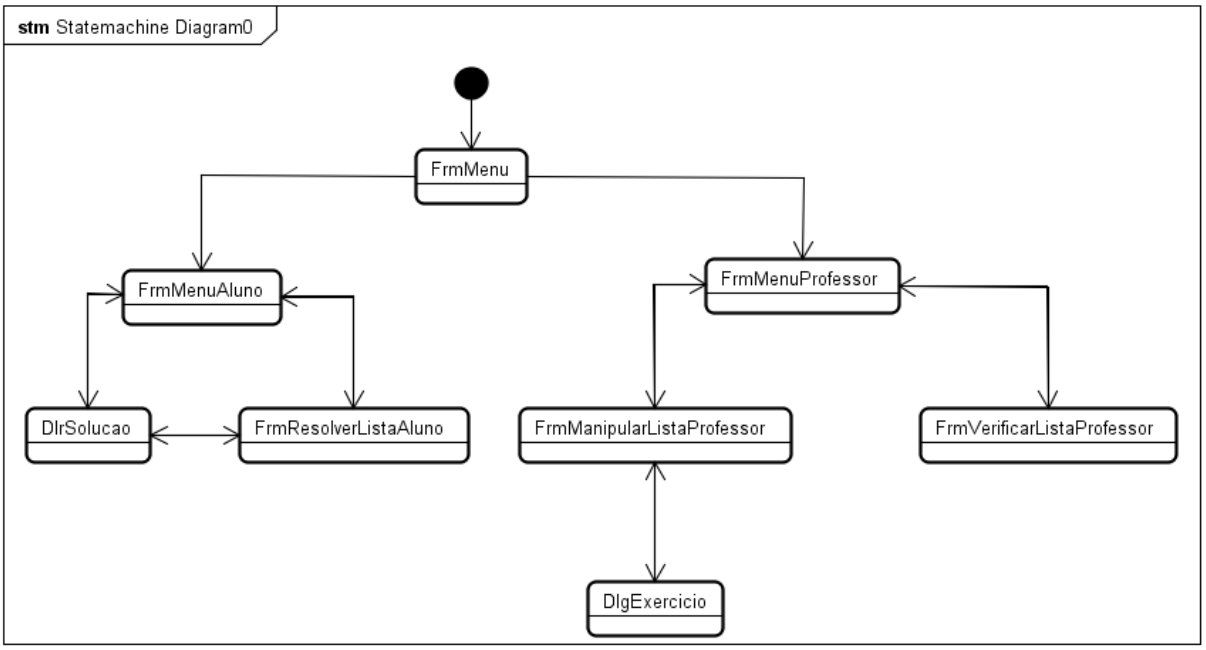

Figura 37 - Diagrama de máquina de estados das interfaces

Partindo do princípio de que a maioria dos casos de usos por parte do aluno, como verificar lista de exercício, escolher um exercício, salvar lista, criar um autômato, são considerados uma tarefa simples de ser implementada, consistindo basicamente em trabalhar com a interface. E de que por parte do professor, os casos de uso como criar lista de exercício, salvar lista, editar lista de exercício, verificar lista de exercício também são considerados uma tarefa simples de ser implementada. Logo, o autor deste trabalho deduz que o interessante a ser explicado detalhadamente é o processo de correção do autômato, ou seja, as classes e os métodos que serão utilizados. Com base nisso, a seguir será explicado como acontece a correção do autômato, bem como o comportamento e a interação entre suas classes.

Correção do autômato

Inicialmente, o processo de correção do autômato utiliza algumas das classes que o JFLAP tem a oferecer, como já foi citado neste trabalho. Além disso, o FaTLR utiliza alguns métodos contidos no JFLAP. Nesse mesmo sentido, será apresentado as classes e os métodos que o autor deste trabalho criou para realizar o método de correção do autômato. A Figura 36 apresenta todas as classes e métodos que foram utilizados para a correção do autômato submetido.

Primeiramente, o processo de correção de autômato, tendo por base o *software* FaTLR, verifica se existe um único estado inicial e se existe pelo menos um estado final. Em seguida, o sistema cria um autômato finito mínimo com base na expressão regular do exercício. Assim, o sistema terá o autômato finito do discente e o autômato finito mínimo do sistema e realizará a comparação. Se o resultado for verdadeiro, retornará ao aluno que está correto, e se não for verdadeiro, a ferramenta utiliza os métodos busca por profundidade, ou o aleatório, para gerar palavras tanto do autômato do aluno quanto do sistema. Após isso, irá processar as palavras geradas pelo autômato do aluno no autômato do sistema, e vice-versa, ou seja, irá também processar as palavras gerada pelo autômato do sistema no autômato do aluno. Esse processo acontece para verificar onde está o erro do autômato do aluno, e possibilitar um retorno mais preciso para o que o aluno consiga corrigir.

Para facilitar a compreensão de como os métodos são utilizado, a Figura 38 apresenta o diagrama de sequência utilizado para realizar a correção do autômato finito.

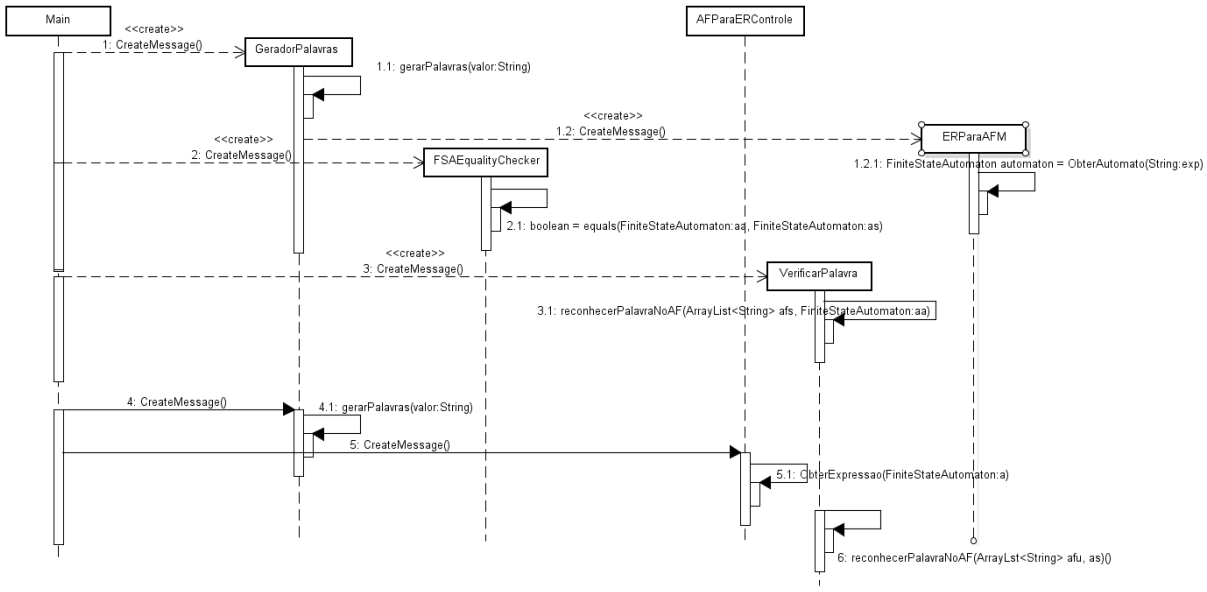

Figura 38 - Diagrama de sequência da correção do autômato

6. AVALIAÇÃO DA FERRAMENTA

Para avaliar a usabilidade e a satisfação do FatLR utilizou-se a aplicação do questionário SUS (*System Usability Scale)*. Este questionário foi desenvolvido pela Digital Equipment CO Ltd., com o objetivo de avaliar a usabilidade dos sistemas e produtos desenvolvidos na empresa e foi descrito por Brooke (BROOKE, 1996, TULLIS; STETSON, 2004). É um questionário simples e de rápida aplicação que demonstra uma visão geral e subjetiva da avaliação da usabilidade de um produto e também avalia a satisfação do usuário em relação ao produto.

O SUS utiliza a escala Likert que, segundo Cunha (2007), é formada por um conjunto de frases (itens), no qual o indivíduo avaliará cada um dos itens, manifestando seu grau de concordância, de modo que as opiniões e atitudes possam ser medidas. Dessa forma, o SUS é formado por 10 questões e cada uma tem uma escala de avaliação que está entre 1 (discordo plenamente), 2 (discordo), 3 (neutro), 4 (concordo) e 5 (concordo plenamente). (Brooke, 1996).

Conforme Simões e Morais (2010), as dez questões avaliam os seguintes itens:

- Frequência de uso do sistema;
- Complexidade do sistema;
- Facilidade de uso;
- Assistência para usar o sistema;
- Funções integradas do sistema;
- Inconsistência do sistema;
- Rápida aprendizagem;
- Sistema é incômodo e complicado para usar;
- Segurança e confiança para usar o sistema;
- Aprendizagem de outras informações para usar o sistema;

Para calcular a pontuação do questionário, deve-se somar a contribuição de cada questão. O valor de cada contribuição muda de acordo com a característica da questão. Para as questões ímpares (1, 3, 5, 7, 9) é a pontuação da escala menos 1. Para as questões de números pares (2, 4, 6, 8, 10) a pontuação na escala é 5 menos para cada resposta marcada. Após determinado o valor de cada questão, é necessário realizar o somatório de todos os valores e multiplicar por 2,6 a fim de obter o resultado

global do SUS. Este resultado global está inserido numa escala de 0 a 100 (Brooke, 1996). Além das dez questões, o questionário apresenta um campo em aberto para que o usuário faça qualquer comentário ou sugestão sobre o sistema.

O SUS foi escolhido pela análise dos resultados obtidos no trabalho de Tullis e Stetson (2004). De acordo com os autores, este método consegue alcançar bons resultados com um baixo número de respostas. Este estudo contou com a participação de 123 respondentes para comparar 5 métodos para avaliação de usabilidade. Ele aponta que com 10 respostas, o SUS consegue um nível de exatidão de 80%, já a partir de 12 respostas, o método alcança um nível de exatidão de 100%, se comparado aos resultados do total de respostas.

Como forma de avaliar a FaTLR, foi criado um questionário on-line que pode ser visualizado no Apêndice deste trabalho. O questionário contém as dez questões propostas no SUS, além de uma questão aberta com o intuito de coletar opiniões e/ou sugestões sobre o sistema.

6.1 Resultados da Avaliação

A avaliação do FaTLR foi realizada por 26 discentes do segundo e terceiro ano do curso de Ciência da Computação da Universidade Estadual do Norte do Paraná (UENP). Inicialmente, foi realizada uma análise por usuário para verificar se existem respostas com uma discrepância muito elevada. Esta análise foi demonstrada através da Figura 39 e mostra que todas as respostas podem ser caracterizadas como válidas visto que não apresentam uma grande discrepância em relação as demais.

A avaliação realizada para testar a usabilidade e a satisfação da FaTLR, através da escala SUS, alcançou 83,17 pontos, essa pontuação foi alcançada através da soma da pontuação dos usuários avaliados dividida pelo número de usuários, ou seja, obtendo a média. De acordo com Cunha (2010), pontuações do SUS abaixo de 60 representam sistemas com experiências relativamente pobres e insatisfação do usuário, e pontuações acima de 80 pontos representam experiências muito boas com alto índice de satisfação dos usuários.

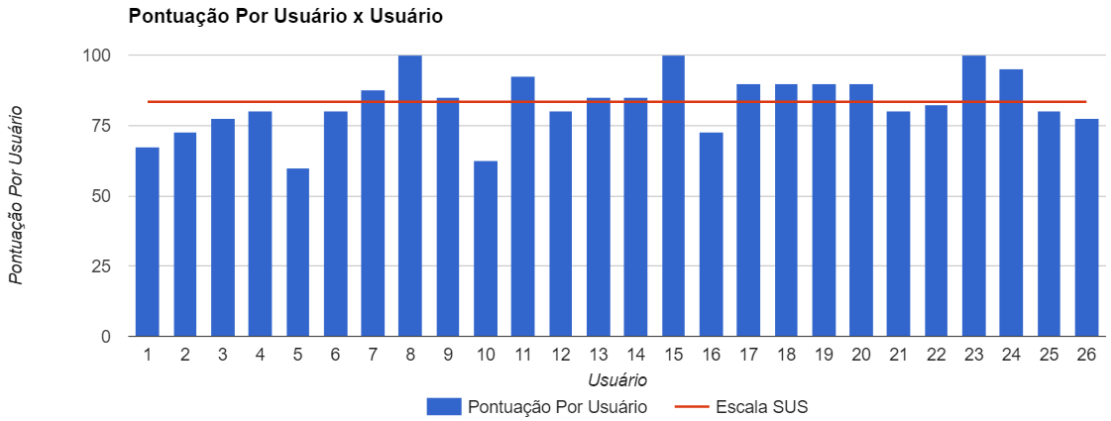

Figura 39 - resultado SUS por usuário

Posteriormente, realizou-se a análise das considerações sobre o ambiente, obtidas através da questão aberta do questionário. A maior parte das respostas refletiu o alto valor obtido no SUS. Algumas delas estão replicadas a seguir:

- “Falta uma opção onde o usuário saiba como inserir a expressão regular”;
- “Ambiente bastante amigável e de fácil utilização”;
- “É um ambiente muito bom para se trabalhar”;
- “Muito bom para o primeiro aprendizado de autômato, bem simples, não apresenta dificuldade por ter uma interface semelhante ao JFLAP”;
- “O ambiente apresentado é muito bom, fácil de usar”.

Algumas outras respostas apontam sugestões, como, por exemplo, deixar um arquivo em PDF com o manual de instruções para o usuário, informando como funciona o programa e outro arquivo de ajuda contendo respostas para possíveis dúvidas (estilo FAQ/SAC).

A avaliação geral do FaTLR se mostrou muito positiva, tanto em relação a pontuação obtida no SUS, quanto em relação às opiniões e sugestões apresentadas pelos usuários que avaliaram a ferramenta. O fato de agradar o usuário é muito interessante, visto que a ferramenta facilitará a alunos e professores da área de linguagens formais.

7. CONSIDERAÇÕES FINAIS

Com a realização deste trabalho, pode-se concluir que a teoria de linguagens formais é um conteúdo de extrema relevância para a área da Ciência da Computação. Tal importância decorre do fato das linguagens formais serem aplicadas em inúmeros tipos de software, tais como editores de texto, verificação do comportamento de circuito digitais e, especialmente, na construção de um compilador, que é um software fundamental para a vasta área de ciências tecnológicas. Já a linguagem regular, que está contida no universo das linguagens formais, é o foco principal deste trabalho e destaca-se no segmento da análise léxica, considerada o primeiro passo do *front-end* do compilador.

O objetivo do trabalho foi alcançado através do projeto e da implementação do ambiente desktop denominado FaTLR. Além disso, esse trabalho possui um método de correção de autômatos finitos. Esse método de correção de autômatos é baseado em dois testes, que foi denominado teste de isomorfismo e teste de aceitação. Basicamente, o primeiro teste verifica se existe isomorfismos entre dois autômatos e o segundo teste analisa os possíveis erros do autômato submetido.

A ferramenta desenvolvida neste trabalho oferece apoio ao treinamento de linguagens regulares, visto que não se tem conhecimento de um software que realize essa correção automática. Ainda, a ferramenta é capaz de prover um retorno preciso para que o aluno consiga corrigir seu autômato de modo mais simples possível.

Com base na pontuação alcançada na avaliação realizada para testar a usabilidade e satisfação da FATLR, a ferramenta pode ser classificada como um ambiente que proporciona experiências muito boas com alto índice de satisfação dos usuários, como foi citado no Capítulo 6. Ademais, algumas sugestões bem relevantes foram adquiridas para uma posterior melhoria do ambiente de apoio ao treinamento.

Para trabalhos futuros pretende-se implementar as melhorias propostas pelos usuários que avaliaram o ambiente. Além disso, pretende-se também transformar a FaTLR em uma ferramenta de treinamento web, visto que uma vez web, os discentes poderão acessá-la em qualquer hora e lugar. Por fim, analisar possíveis *gamificação* para que o sistema fique cada vez mais interativo ao discente, uma vez que a gamificação traz motivação e o engajamento ao ensino, fundamentais para que o discente tenha cada vez mais vontade de aprender (Fadel et al, 2014).

REFERÊNCIAS

AHO, Alfred V.; LAM, Monica S.; ULLMAN, Jeffreu D. **Compiladores:** princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson Addison-wesley, 2008. 634 p.

ALMEIDA, Maria. E. Bianconcini; **Educação a distância na internet:** abordagens e contribuições dos ambientes digitais de aprendizagem. Educação e Pesquisa, v. 29, n. 2: 327-340, jul./dez, 2003.

BISSONI, Estefania. Proposta de utilização do Ava Moodle para formação e treinamento de colaboradores de uma instituição de ensino. In: CLARETIANO, Centro Universitário (Org.). **Educação a distância:** revista científica. 2. ed. Batatais: Centro Universitário Claretiano, 2012. p. 159-180.

BOTELHO, Joacy Machado; CRUZ, Vilma Aparecida Gimenes da. **Metodologia:** Estudo e ensino. São Paulo: Pearson Education do Brasil, 2013. 135 p.

BROOKE, John. Sus-a quick and dirty usability scale. *Usability evaluation in industry*,189(194):4-7, 1996.

CUNHA, Luísa Margarida Antunes da. **Modelos Rasch e Escalas de Likert e Thurstone na medição de atitudes.** 78 f. Dissertação (Mestrado), Universidade de Lisboa, Lisboa, 2007.

CUNHA, Luiz Coelho Cunha. Redes sociais dirigidas ao contexto das coisas. Master's thesis, PUC- RJ, 2010.

DHARWADKER, Ashay; TEVET, John-Tagore. The Graph Isomorphism Algorithm, 2009.

DIEHL, Daniel. C. **Ambiente virtual para manipulação de uma célula robotizada.** Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2004.

FADEL, Liciane Maria; ULBRICHT, Vania Ribas; BATISTA, Claudia Regina; Vanzin Tarcísio (Org.). **Gamificação na educação.** São Paulo: Pimenta Cultural, 2014. 300 p.

FEOFILOFF, Paulo; KOHAYAKAWA, Yoshiharu; WAKABAYASHI, Yoshiko. **Uma introdução Sucinta à Teoria dos Grafos.** 2004. 61. - Curso de Ciência da Computação, Departamento de Ciência da Computação, Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2011.

FURTADO, Olinto José Varela. **Linguagens formais e compiladores.** Universidade Federal de Santa Catarina: Departamento de Informática e de Estatística, 2002. p. 52

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002. 176 p.

GSKINNER. **Regexr:** Learn Build, & Test RegEx. Disponível em: <www.regexr.com>. Acesso em: 01 jul. 2016.

HOPCROFT, John E.; ULLMAN, Jeffrey D.; MOTWANI, Rajeev. **Introdução à Teoria de Autômatos, Linguagens e Computação.** 2. ed. Rio de Janeiro: Editora Campus, 2002.

LIMA, Jorge Reis; CAPITÃO, Zélia. **E-Learning e e-Conteúdos:** Aplicações das teorias tradicionais e modernas de ensino e aprendizagem à organização e estruturação de E-Cursos. 1. ed. Portugal: Lisboa: Centro Atlântico, 2003.

LIMA, Paulo Roberto. **Um software educacional para construção e validação de formalismos utilizados na geração e reconhecimento de sentenças de uma linguagem regular.** 2006. 82 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Educação São José, Universidade do Vale do Itajaí, São José, 2006.

LOUDEN, Kenneth C.. **Compiladores:** Princípios e Práticas. São Paulo: Pioneira Thomson Learning, 2004. 569 p.

MARTINS, Joyce. **Linguagens formais e compiladores.** 2003. 50 f. - Curso de Ciência da Computação, Centro de Educação São José, Universidade do Vale do Itajaí, São José, 2003.

MAHESHWARI, Anil; SMID, Michiel. **Introduction to Theory of Computation.** 2016. - Curso de Computer Science, Carleton University, Ottawa, 2016. 244 p.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos.** 3. ed. Porto Alegre: Sagra Luzzato, 2000. 165 p.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos.** 6. ed. Porto Alegre: Bookman, 2011. 256 p.

ORACLE. Java SE Development Kit 8. Acesso em: 30 de novembro de 2016. Disponível em < http://docs.oracle.com/javase/7/docs/technotes/guides/swing/>.

PALLAZO, Luiz A.M. **Expressões regulares e Gramáticas Regulares.** Rio Grande do Sul: Universidade Católica de Pelotas, 2007. 7 p.

PENTERICH, Eduardo. **Ambientes virtuais de aprendizagem.** Sala de Aula e Tecnologias. São Paulo: Editora da Universidade Metodista de São Paulo, 2005. 92 p.

PEREIRA, Mara Alexandra de Jesus. **O E-Learning na Formação de Pais e Educadores no Apoio à Aprendizagem dos Educandos.** 2010 - Curso de Multimídia em Educação, Departamento de Comunicação e Arte, Universidade de Aveiro, Aveiro, 2010. 249 p.

RAMOS, Marcus Vinícius Midena. **Linguagens formais e autômatos.** 2008. Curso de Engenharia da Computação, Universidade Federal do Vale do São Francisco, São Francisco, 2008. 384 p.

RODGER, Susan; FINLEY, Thomas; LINZE, Peter. **JFLAP.** 2006. Disponível em: <www.jflap.com>. Acesso em: 01 jul. 2016

SCHLEMMER, Eliane. **AVA: Um ambiente de Convivência Interacionista Sistêmico para Comunidades Virtuais na Cultura da Aprendizagem.** Porto Alegre, 2002.

SCHLEMMER, Eliane. **Telepresença**. Curitiba: IESDE Brasil S.A., 2009. 180 p.

SCHLEMMER, Eliane; SACCOL, Amarolinda Zanela; GARRIDO, Susane. **Um modelo sistêmico de avaliação de softwares para educação a distância como apoio à gestão de EAD.** Revista de Gestão USP, São Paulo, v 14, n.1:77-99, jan./mar, 2007.

SELIVON, Michele; BEZ, Jean Luca; TONIN, Neilor A. URI online judge academic: integração e consolidação da ferramentano processo de ensino/aprendizagem. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wei/2015/020.pdf>. Acesso em: 01 jul. 2016.

SILVA, Cassandra Ribeiro de O. e. **Metodologia e Organização do projeto de pesquisa.** 2004. 34 f, Centro Federal de Educação Tecnológica do Ceará, Fortaleza, 2004.

SILVA, Edgar L.; MIRANDA, J. J.; HOUNSELL, M. S. **Diferenças entre educação e treinamento em ambientes virtuais 3D,** 2007.

SIMÕES, Aliana Pereira; MORAES, Anamaria. Aplicação do questionário sus para a avaliação da
satisfação e usabilidade de um software de ead. *Anais do 10º Congresso Internacional de Ergonomia e Usabilidade de Interfaces Humano-Computador*, 10, 2010

SORDI JUNIOR, Fábio de. **Desenvolvimento de um ambiente colaborativo de treinamento preparatório para o POSCOMP.** 2015. 63 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Centro de Ciências Exatas, Universidade Estadual de Londrina, Londrina, 2015.

TULLIS, Thomas S; STETSON, Jacqueline N. A comparison of questionnaires for assessing website
usability. In *Usability Professional Association Conference*, pages 1-12, 2004.

APÊNDICES

APÊNDICE A – INTERFACES DA FERRAMENTA

As interfaces da FaTLR serão apresentados de um modo completo e detalhado. Primordialmente será apresentado o Menu da ferramenta, A Figura 40 apresenta o menu da ferramenta.

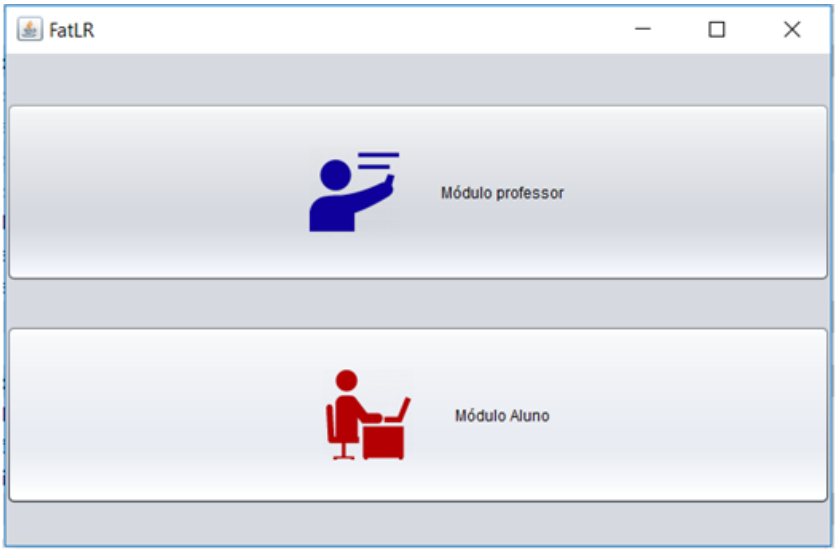

Figura 40 - Menu da FaTLR

Logo em seguida, será detalhado todas as interfaces do módulo do professor. A Figura 41 apresenta o menu do módulo do professor. Onde possui quatro botões, “Criar Lista”, “Abrir Lista”, “Verificar Respostas” e “Sair”. E além disso, possui uma descrição para auxiliar o professor a realizar suas tarefas.

Figura 41 - Menu professor

Se o usuário clicar no botão “Criar Lista” ou “Abrir lista” irá abrir a tela demonstrada n Figura 42 apresenta essa interface. O que difere as duas funcionalidades é que no “Criar Lista” o usuário não escolhe nenhuma lista feita. Já por parte do “Abrir lista” ele deverá escolher uma lista que já foi salva.

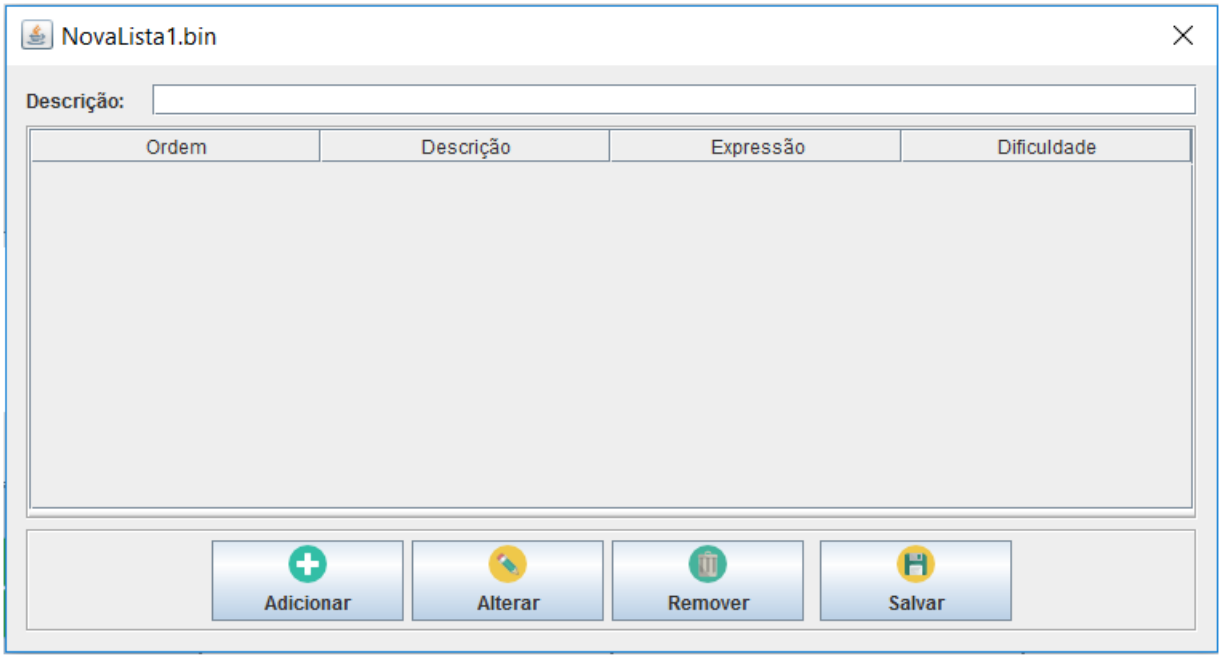

Figura 42 - Criar lista e Abrir lista

Se o usuário clicar no botão “Verificar resposta” abrirá a tela para analisar se o aluno realizou a lista feita, quais os exercícios que ele conseguiu fazer, quantas tentativas. A Figura 43 apresenta a interface para verificar a resposta.

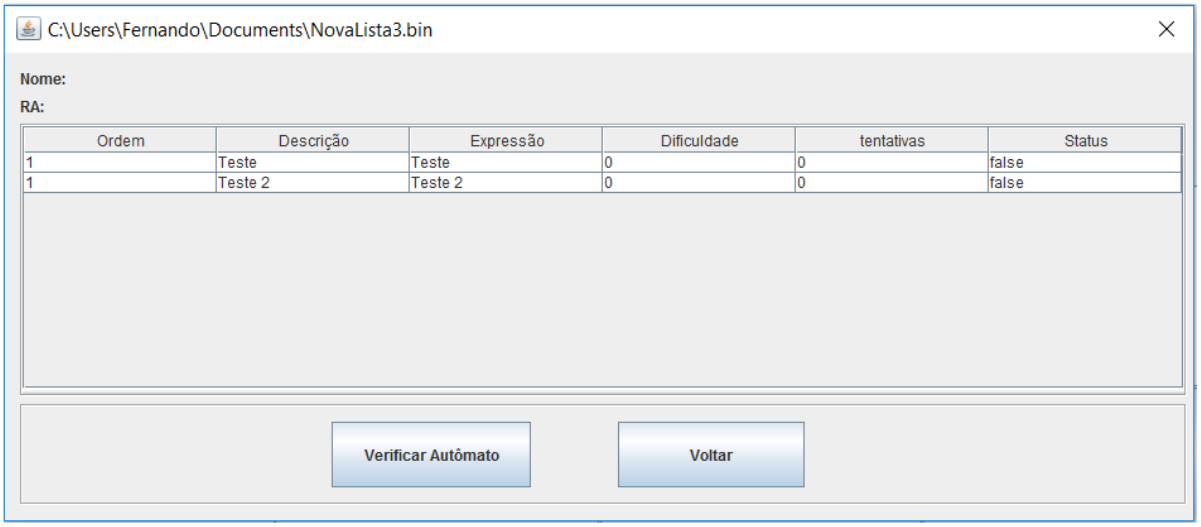

Figura 43 - Verificar resposta

Ao Clicar no botão sair, finaliza o sistema. Já por parte do aluno, inicialmente, a Figura 44 apresenta a interface do menu do aluno. Onde possui o botão “Abrir Lista”, “Exercício individual” e “Sair”. Ainda, possui uma descrição dos botões para auxiliar o aluno a realizar seus objetivos.

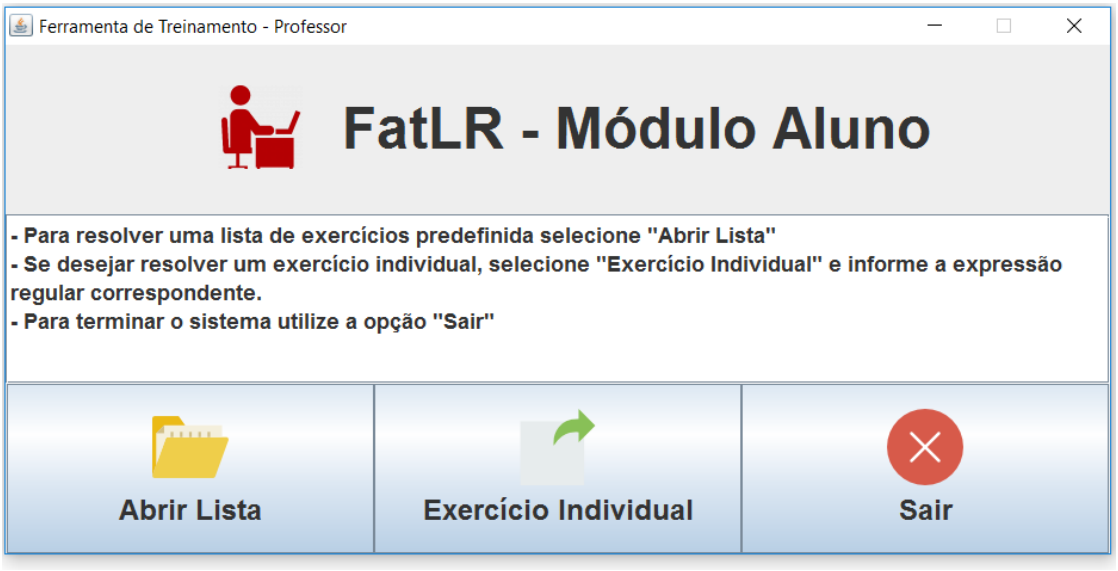

Figura 44 - Menu aluno

Ao clicar no botão “Abrir Lista” o aluno deverá selecionar uma lista a ser feita, logo em seguida, abrirá a interface que contém os exercícios a serem feito. A Figura 45 demonstra essa interface.

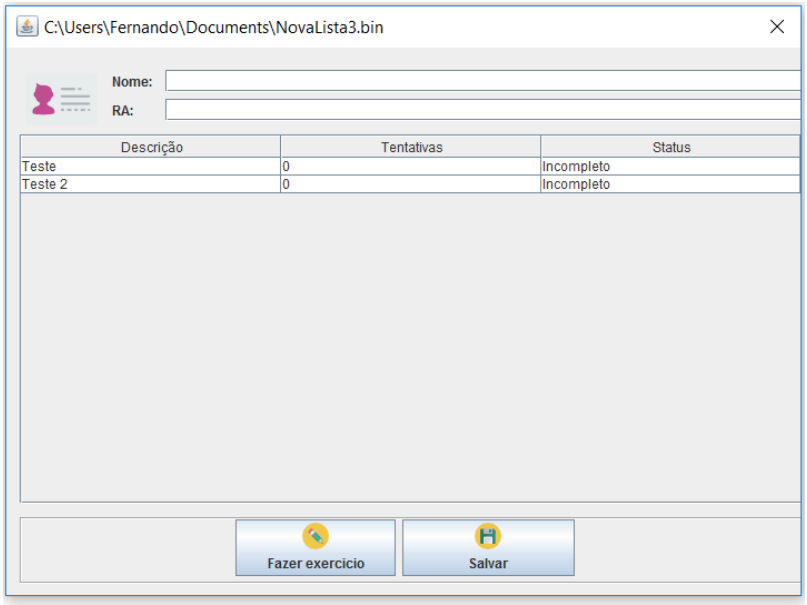

Figura 45 - Exercícios

Na interface que apresenta os exercícios, o discente deverá selecionar um exercício e clicar no botão “Fazer exercício” contido na mesma. Logo, abrirá a interface que ele deverá construir seu autômato. A Figura 46 apresenta a interface de construção do autômato.

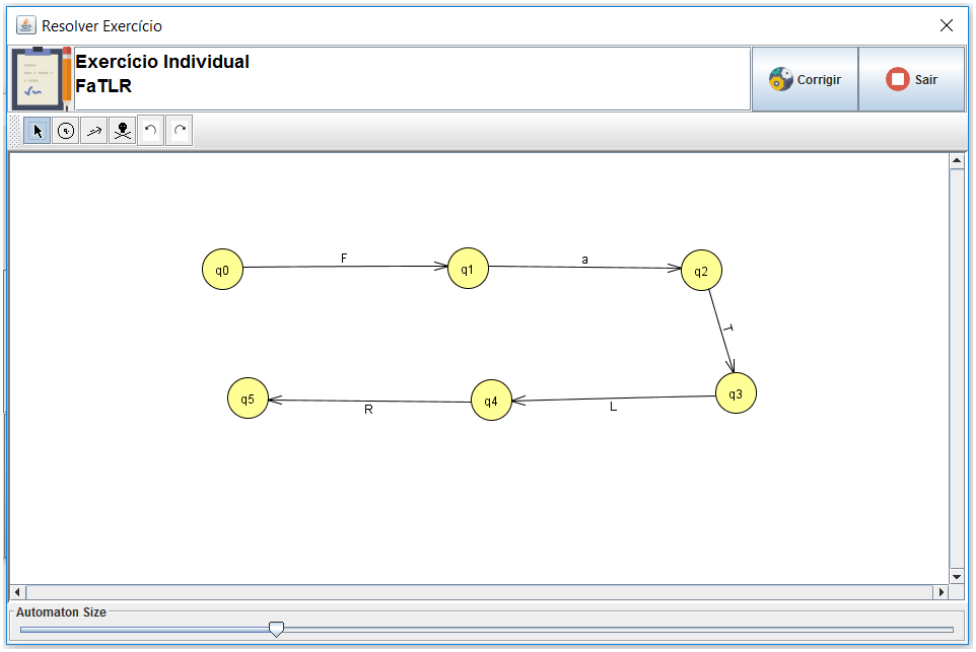

Figura 46 - Interface construir autômato

Ao clicar no botão “Exercício individual” o discente deverá inserir uma expressão regular e abrirá a mesma interface contida na Figura 46. O que difere as funcionalidades é que no “Abrir Lista”, já existe uma expressão regular pré-definida. Já por parte do “Exercício individual” o discente deverá inserir a expressão regular desejada. Na interface construir autômato, possui um botão “Corrigir” no qual ao clicar, ele irá corrigir o autômato construído com base na expressão regular.

ANEXOS

ANEXO A – QUESTIONÁRIO PARA AVALIAÇÃO

Escala de Usabilidade de Serviços da FaTLR

Este questionário avaliará a escala de usabilidade dos serviços oferecidos pela Ferramenta de Apoio de Treinamento de Linguagens Regulares (FaTLR). As respostas deste questionário serão utilizadas no Trabalho de Conclusão de Curso (TCC) do aluno Fernando Hideyuki Moribe.

1. Gostaria de usar este serviço frequentemente
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

2. Achei que o serviço era desnecessariamente complexo
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

3. Achei o serviço simples de usar
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

4. Penso que iria precisar de apoio técnico para usar o serviço
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

5. Achei as várias funcionalidades do serviço bem integradas
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

6. Penso que existe uma alta inconsistência no serviço
   Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

7. Imagino que a maioria das pessoas aprenderiam a usar o serviço rapidamente
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

8. Achei que o serviço não era trivial de usar
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

9. Senti-me muito confiante a usar o serviço
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

10. Preciso aprender muito antes de poder usar este serviço
Marcar apenas uma oval.

	1	2	3	4	5	
Discordo muito	◯	◯	◯	◯	◯	Concordo muito

11. Escreva aqui suas opiniões e sugestões sobre o Ambiente

.....

.....

.....

.....