UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NANDERSON BRUNO MENEZES ROCHA

APLICATIVOS DE ENSINO NA DISCIPLINA DE LINGUAGENS FORMAIS

São Luís
2017

NANDERSON BRUNO MENEZES ROCHA

APLICATIVOS DE ENSINO NA DISCIPLINA DE LINGUAGENS FORMAIS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Ivo José Cunha Serra

São Luís

2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Rocha, Nanderson Bruno Menezes.
Aplicativos de ensino na disciplina de linguagens formais / Nanderson Bruno Menezes Rocha. – 2017.
87 f.

Orientador(a): Ivo José da Cunha Serra.
Monografia (Graduação) – Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2017.

1. Ensino. 2. Linguagens formais. 3. Software. I. Serra, Ivo José da Cunha. II. Título.

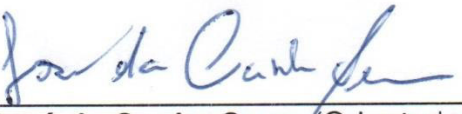
NANDERSON BRUNO MENEZES ROCHA

APLICATIVOS DE ENSINO NA DISCIPLINA DE LINGUAGENS FORMAIS

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

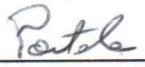
Aprovada em: 12 / 07 / 2017

BANCA EXAMINADORA



Prof. Ivo José da Cunha Serra (Orientador)
Doutor em Ciência da Computação
Universidade Federal do Maranhão

Prof. Samyr Beliche Vale
Doutor em Ciência da Computação
Universidade Federal do Maranhão



Prof. Carlos Eduardo Portela Serra de Castro
Mestre em Ciência da Computação
Universidade Federal do Maranhão

Dedico esse trabalho primeiramente a Deus, aos meus pais, minha irmã e ao meu orientador e amigos.

AGRADECIMENTOS

Primeiramente a Deus, por sempre está comigo em todos os momentos, pois sem ele, não teria a capacidade de alcançar esse objetivo em minha vida, por sempre me direcionar e ensinar o caminho certo.

Aos meus pais Raimundo Nonato Rocha Filho e Linete de Maria Menezes Rocha, pessoas fundamentais em minha vida que me apoiaram e apoiam nessa minha trajetória, mesmo com todas as dificuldades e problemas nunca me abandonaram sempre me ensinando a lutar pelos meus ideais, confiando sempre em mim e nos meus objetivos, ficando sempre do meu lado em minhas decisões. A minha irmã Larissa Karine, que me deu força pra continuar e nunca desistir, que ao longo desse caminho estive ao meu lado.

Ao meu orientador, professor Ivo Serra, por ter aceitado me ajudar nesse trabalho, pela dedicação, competência, que soube ter paciência, e sabedoria em me transmitir segurança a conseguir realizar esse trabalho que tanto temia, foi muito importante na realização desse trabalho.

Agradeço a todos os meus amigos, que direta ou indiretamente me deram forças para concluir mais essa etapa em minha vida, acreditando e me fazendo acreditar em minha capacidade.

“Tudo posso naquele que me fortalece”

Filipenses 4:13

RESUMO

Ensino e aprendizagem em todas as disciplinas são complexos, quando não existem métodos adequados para um bom entendimento e desenvolvimento dos conteúdos. Como em todas as disciplinas de ensino, Linguagens Formais tem suas peculiaridades. Uma delas é que não atrai muito interesse de muitos estudantes, principalmente por causa de sua estrutura e abstração. O beneficio deste trabalho é a procura por diferentes ferramentas de software e metodologias para dar uma solução para tal problema.

Este trabalho inclui a pesquisa sobre métodos de aprendizagem e avaliação, bem como software e abordagens para Ensino de Línguas formais para as áreas de computação e afins. Seu principal objetivo é apresentar e discutir ferramentas de software que podem ser usados e apresentem conteúdos da disciplina de Linguagens Formais e, assim, facilitar o processo de ensino/ aprendizagem.

Foram utilizadas as bibliografias disponíveis, tais como artigos, livros e outros meios que abordam o tema para o qual o trabalho está relacionado. Essas informações serviram como base para o seu desenvolvimento. O trabalho foi realizado dentro do âmbito da pesquisa bibliográfica. Discutimos como o ensino e aprendizagem de linguagens formais podem se beneficiar do uso de software educacional.

Palavras – chave: ensino, linguagens formais, software.

ABSTRACT

Teaching and learning in all disciplines is complex, when there are no adequate methods, for a good understanding and development of the contents. As in all disciplines the teaching of Formal Languages has its peculiarities. One of these is that it does not attract much interest of many students, mainly because of its structure and abstraction. This leeds to searching for different methodologies and software tools to give a solution to such problem.

This work includes research on methods of learning and evaluation, as well as software approaches for teaching of Formal Languages for the areas of computing and related. Its main objective is to present and discuss software tools that can be used to present contents of the discipline of Formal Languages and thus facilitate the teaching / learning process.

Bibliographies available were used, such as articles, books and other means that address the topic to which the work is related. This information served as a basis for its development. The work was carried out within the scope of this bibliographical research. We discussed how teaching and learning formal languages can benefit of using educational software.

Keywords: teaching, formal languages, software.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Por centenas de anos o quadro negro, o giz e o livro foram marcados por serem os instrumentos mais utilizados no ensino para mediação pedagógica. No entanto a expansão de artefatos tecnológicos, ferramentas educacionais, vem se expandindo e revolucionando a educação tradicional.

Masetto (2000) entende que o uso de novas tecnologias em educação como: computador, internet, hipermídia, multimídia e de outros recursos de linguagens digitais de que dispomos, colaboram significantemente pra tornar a educação mais eficiente e eficaz.

Na educação a utilização dessas novas ferramentas de ensino utilizando ferramentas educacionais ajuda a despertar nos alunos a capacidade de raciocínio, observação e crítica, podendo despertar assim a vontade de aprender e estudar que alguns alunos perderam ou "nunca tiveram". Para isso devem ser apresentadas aplicações práticas da temática a fim de tornar o assunto mais instigante aos alunos.

Posto isto, os professores têm a difícil tarefa de com a ajuda desses novos recursos educacionais, criar alternativas para o ensino do conteúdo com o propósito de torná-lo mais atrativo aos alunos.

Como instrumento tecnológico na educação evidencia-se o uso dos softwares educacionais por possibilitar ao usuário uma relação entre a realidade e a teoria que lhe é ensinada, exercendo assim a função de aliados no processo de aquisição de conhecimento.

Sendo assim, pode-se dizer que a mudança proporcionada pela inserção das tecnologias na educação é extremamente relevante para romper com paradigmas impostos pela educação tradicional, contribuindo desta forma para a criação de novas propostas metodológicas e para o enriquecimento do processo de ensino/aprendizagem (TIMBOÍBA et al, 2011).

O objetivo do presente trabalho se dá na análise dos prováveis benefícios, apontar algumas limitações e a importância do uso destes softwares para o ensino de umas das disciplinas teóricas da Computação, a disciplina de Linguagens Formais e Autômatos (LFA).

1.1 Motivação

O foco principal desse trabalho é a discussão sobre o uso de softwares no processo de aprendizagem para o ensino de umas das disciplinas clássicas da Computação, a disciplina de Linguagens Formais e Autômatos (LFA).

A disciplina de LFA que é um conteúdo essencial para a formação dos profissionais da área da Computação e afins, mas por ser uma disciplina bastante teórica, com definições que utilizam formalismos matemáticos e demonstrações de teoremas, pode acabar sendo considerada pelos alunos um conteúdo bastante abstrato e complexo, tendo como consequência diminuir o interesse e estimulo dos alunos. Desta forma, esse trabalho foi motivado por ressaltar a importância do uso de softwares educacionais no contexto de LFA, destacando-se a sua serventia notória em relação à compreensão das informações e a atenuar as dificuldades de aprendizagem.

1.2 Objetivos do Trabalho

Este trabalho tem como objetivo:

- Evidenciar a relevância da utilização de softwares educativos na disciplina de Linguagens Formais.
- Apresentar alguns softwares de ensino em Linguagens Formais.
- Discutir cada um dos softwares apresentados ressaltando aspectos positivos e limitações.

1.3 Estrutura do trabalho

O capítulo 2 trata de alguns marcos da informática na educação, definições de LFA, a importância do estudo da disciplina LFA para o curso de Computação e a relevância de utilizar softwares educativos na disciplina de Linguagens Formais, mostrando o que professores e alunos podem ensinar/aprender usando novos métodos e assim obter resultados mais satisfatórios do que usando somente o ensino convencional.

O capítulo 3 apresenta alguns softwares de ensino em Linguagens Formais presentes no mercado, discutindo cada um deles e ressaltando seus aspectos positivos e limitações.

No capítulo 4 é apresentada a discussão sobre a relevância dos softwares educacionais, particularmente aqueles da área de Linguagens Formais.

E finalmente no capítulo 5 são apresentadas às conclusões, contribuições e possíveis trabalhos futuros.

2 Softwares educacionais e introdução a Linguagens Formais

Antes de pensar nos softwares educacionais sendo implantados no projeto pedagógico das escolas, Valente (1992) discute a revolução que foi quando decidiram colocar computadores nas escolas e transformá-los em ferramentas de ensino. O autor faz um relato da evolução deste processo. Em 1955, cerca de uma década após sua invenção, o computador era utilizado no âmbito acadêmico apenas para resolver problemas nos cursos de pós-graduação e somente em 1958 foi utilizado como uma máquina de auxílio ao ensino no Centro de Pesquisas Watson da IBM e na Universidade de Illinois, nos Estados Unidos. Entretanto, tinha uma função diferente das muitas que encontramos hoje nos computadores das escolas, pois eram vistos apenas como armazenadores de informação que ficavam à disposição quando fosse necessário.

Em outro artigo, Valente (1997a) destaca o trabalho de Skinner, B. F.(1968), um promissor professor da universidade americana *Harvard Business School*, que idealizava uma máquina que pudesse auxiliar no processo de construção do conhecimento “usando o conceito de instrução programada”. Skinner seguia uma linha de raciocínio em que o conteúdo passado deveria ser dividido em fases que seguiriam uma sequencia lógica e iriam aumentar o grau de dificuldade gradativamente; os alunos responderiam perguntas referentes ao que estava sendo estudado e, se acertassem, passariam de fase, se não, poderiam voltar onde sentiram dificuldades e começar novamente a construção de determinado conhecimento. Esta ideia foi projetada entre o fim dos anos 1950 e começo dos anos 1960 de forma impressa, porém não conseguiam disseminar este meio de estudo.

Amante (2011) comenta, com base na ideia de Skinner, B. F., a implantação do EAC - Ensino Assistido por Computador que, apesar de idealizar um método de ensino diferenciado e com ênfase na tecnologia, não se destacava muito de como era a sala de aula, pois o computador apenas transmitia informações. Seguido deste programa, outra tentativa de integrar tecnologia e educação, a Alfabetização Informática, trazia para a grade curricular o ensino efetivo da Informática e aulas específicas para que os alunos pudessem ter um tempo dedicado ao uso da máquina.

Porém, conforme Valente (1997), apenas com a chegada dos computadores em algumas instituições de ensino no início da década de 1960, é que softwares de ensino voltados ao uso da tecnologia puderam ser integrados e utilizados para fins pedagógicos. Nos Estados Unidos surgiu o CAI - *Computer-Aided Instruction*, que, no Brasil, foi chamado PEC- Programas Educacionais por Computador. Desta época em diante, empresas famosas, como a IBM, começaram a investir e comercializar softwares educativos, até então usados apenas em universidades que disponibilizavam aqueles equipamentos caros da época: os *mainframes* (computadores de grande porte).

Estes novos recursos e esta nova metodologia, no entanto, só puderam chegar às escolas quando os microcomputadores ganharam o mercado no início da década de 1980. Com estes equipamentos no comércio, a produção de softwares educativos chegou a 125 programas colocados à venda por mês naquele período. Este número cresceu tanto, que hoje é difícil afirmar quantos destes programas são criados e comercializados.

Nas décadas de 1980 e 1990, Amante (2011) descreve a expansão das funções que o computador foi ganhando com softwares que permitiam a digitação de textos, a obtenção de informações a partir de bases de dados, o processamento de imagens, o tratamento de dados através de planilhas e gráficos, dentre outras. Esta expansão atingiu seu ápice com a implantação da Internet e as TICs - Tecnologias da Informação e da Comunicação, que permitiam um acesso maior a diversas informações e uma maior interatividade do usuário não somente com o computador, mas com outras pessoas ligadas à rede, permitindo a troca e a distribuição de informações em quantidade e em velocidade nunca antes experimentadas.

Segundo Oliveira (2017) percebe-se que a educação atualmente precisa refletir sobre todo sistema educacional, corrigir lacunas que ainda permeiam a mesma. E, encarar a questão da mídia na educação como um processo dinâmico e que necessita ultrapassar a visão de ensino tradicional. Desse modo, construindo novos paradigmas, onde eles sejam capazes de formar cidadãos críticos e reflexivos para viver nessa cultura tecnológica.

2.1 Noções sobre Linguagens Formais e Autômatos

A disciplina de Linguagens Formais e Autômatos (LFA) tem como propósito o entendimento e compreensão de assuntos complexos que fazem parte da computação como a decidibilidade (isto é, a questão da existência de um método capaz de determinar um conjunto de questões para as quais a resposta para um problema *P* são sim ou não) e a complexidade computacional (que consiste em classificar problemas computacionais, passiveis de resolução por um computador, de acordo com sua dificuldade inerente, e relacionar essas classes entre si) que abrangem amplamente as práticas computacionais.

Para Menezes (2008) é indispensável para compreender a disciplina de LFA, alguns conceitos como: alfabeto, palavra e linguagem. Alfabeto é um conjunto finito não vazio de símbolos, onde símbolo (ou caractere) é uma entidade abstrata a qual é definida formalmente. Letras e dígitos são exemplos de símbolos. Uma palavra, ou sentença sobre um alfabeto, é uma sequencia finita de símbolos (do alfabeto) justapostos. Segundo Bonfim (2009) uma linguagem formal é um conjunto, finito ou infinito, de cadeias de tamanho finito, compostas pela concatenação de elementos de um alfabeto finito e não vazio. Uma gramática é uma ferramenta formal capaz de gerar conjuntos de cadeias de certa linguagem. As gramáticas são instrumentos utilizados na definição das características das linguagens.

Menezes (2008) evidencia que LFA abrange o conhecimento de assuntos essenciais. Tais como:

- Hierarquia de Chomsky ou Hierarquia de classes de linguagens
- Linguagens regulares
- Linguagens livres de contexto
- Linguagens recursivamente enumeráveis e sensíveis ao contexto

Alguns dos formalismos utilizados para representar as linguagens dessas classes são: Gramáticas, Autômatos e Expressões Regulares.

2.1.1 Hierarquia de Chomsky

Hierarquia de Chomsky foi uma categorização de gramáticas formais criada em 1959 pelo linguista Noam Chosky. Uma gramática formal consiste em um conjunto de regras pra elaboração de cadeias de caracteres (strings) em uma linguagem formal, isto é, ela tem por finalidade caracterizar uma linguagem. Tal classificação é composta por quatro níveis distintos e são usadas na Computação na definição de linguagens de programação e na implementação de interpretadores e compiladores. Para ser mais especifico são empregadas na análise sintática e na análise léxica.

A Hierarquia de Chomsky seguem na tabela 1:

Tabela 1 - Hierarquia de Chomsky

Tipo	Gramática	Linguagem	Reconhecedor (Autômato)
0	Irrestrita	Recursivamente enumerável	Máquina de Turing
2	Sensível ao contexto	Sensível ao contexto	Autômato linearmente limitado
3	Livre do contexto	Livre do contexto	Autômato de pilha
3	Regular	Regular	Autômato finito

Fonte: Adaptado de Padro (2016).

Tal Hierarquia é composta por tipos, sendo que cada tipo indica o nível de autonomia em relação as regras, ela se inicia do tipo 0, que possui o maior nível de autonomia e vai aumentando até ao tipo 3 com a maior nível de limitação. A seguir será explicado o que é gramática e cada tipo de gramática (Irrestrita, Sensível ao contexto, Livre do contexto e Regular) além de o que é autômato e os tipos de Autômatos (Máquina de Turing, Autômato linearmente limitado, Autômato de pilha, Autômato finito) seguindo a Hierarquia de Chomsky.

.

2.1.2 Gramática

Rangel (2016) diz que em teoria das linguagens formais, uma gramática é um conjunto de regras de produção de cadeias em uma linguagem formal, ou seja, um objeto que permite especificar uma linguagem ou língua. As regras descrevem como formar cadeias — a partir

do alfabeto da linguagem — que são válidas de acordo com a sintaxe da linguagem.

Formalmente, as gramáticas são caracterizadas como quádruplas ordenadas G = (V, T, P, S) onde:

- V representa o vocabulário não terminal da gramática. Este vocabulário corresponde ao conjunto de todos os símbolos dos quais a gramática se vale para definir as leis de formação das sentenças da linguagem.
- T é o vocabulário terminal, contendo os símbolos que constituem as sentenças da linguagem. Dá-se o nome de terminais aos elementos de T.
- P representa o conjunto de todas as leis de formação utilizadas pela gramática para definir a linguagem. Para tanto, cada construção parcial, representada por um não-terminal, é definida como um conjunto de regras de formação relativas à definição do não terminal a ela referente. A cada uma destas regras de formação que compõem o conjunto P dá-se o nome de produção da gramática. Assumimos $V \cap T = \varnothing$. Convencionamos que V U T = *V* Cada produção P tem a forma:

$$\alpha \rightarrow b\ \alpha \in V+;\ b \in V*$$

- S ϵ V denota a principal categoria gramatica de G; é dito o símbolo inicial ou o axioma da gramática. Indica onde se inicia o processo de geração de sentenças.

Ex.1: G = ({S, A, B}, {a, b}, P, S)

P: {S → AB

A → a

B → b}

2.1.2.1 Gramática regulares

Para o entendimento sobre gramáticas regulares precisamos ter conhecimento sobre o que é uma linguagem linear. Uma gramática G = {V, T, P, S} diz-se linear se todas as produções são da forma:

A → wB

A → Bw

A → w

onde $A,B \in V$ e $w \in T^*$

Assim uma gramática é linear à esquerda se todas as produções são da forma A → Bw ou A → w, assim como uma gramática linear á direita são todas produções da forma A → wB ou A → w.

Uma gramática regular é uma gramática linear á direita ou á esquerda. Suas restrições são de grande importância ao estudo dos compiladores por possuir propriedades adequadas para a aquisição de reconhecedores simples, que são dispositivos formais que nos permitem verificar se uma determinada sentença pertence ou não a uma determinada linguagem. Esses dispositivos são autômatos; autômatos finitos, autômatos de pilha e máquinas de Turing, por exemplo, que são destacados como importantes classes de autômatos.

2.1.2.2 Gramática livre do contexto

A gramatica livre de contexto (GLC) se caracteriza na teoria de linguagem formal, como a gramática ao quais todas as regras de produção seguem são da forma:

$$A \rightarrow \alpha$$

Sendo *A* um símbolo não terminal, e α uma cadeia de terminal e/ou não terminais (α podendo ser palavra vazia). Essa linguagem formal é dita “livre de contexto” justamente pelo fato de suas regras de produção ser empregadas livres do contexto do símbolo não terminal. E esta é a característica que diferencia a GLC de todas as demais gramáticas.

Para Menezes (2008) as linguagens Livres de Contexto tem um grande papel na computação, tratando de algumas questões com o uso de parênteses balanceados e blocos-estruturados que estão presentes nas linguagens de programação, além de estarem presentes em aplicações como editores de texto, tradutores de linguagens e analisadores sintáticos.

Um exemplo é a gramática G = ({S}, {a,b}, P, S) com as produções:

$S \rightarrow aSa$,

$S \rightarrow bSb$,

$S \rightarrow \varepsilon$,

G é uma gramática livre de contexto uma vez que suas produções são independentes ao contexto do símbolo não terminal.

2.1.2.3 Gramática sensível ao contexto

Uma gramática sensível ao contexto (GSC), também conhecida como Tipo 1 da Hierarquia de Chomsky, é uma gramática formal em que os lados esquerdo e direito de qualquer regra de produção podem ser cercados por um contexto de símbolo terminal e símbolo não-terminal. Símbolos terminais são caracteres literais que podem aparecer nas entradas ou saídas das regras de produção de uma gramática formal e não podem ser quebradas em unidades "menores". Para ser preciso, símbolos terminais não podem ser modificados usando as regras da gramática, já os Símbolos *não-terminais*, ou simplesmente *não-terminais*, são os símbolos que podem ser substituídos; portanto existem cadeias compostas por uma combinação de terminais e não-terminais símbolos.

Gramáticas sensíveis ao contexto são mais gerais do que as gramáticas livres de contexto, mas ainda ordenadas o suficiente para serem verificadas por um autômato linearmente limitado (isto é, uma Máquina de Turing com memória limitada por possuir uma fita limitada e portanto finita)

Usando uma definição formal, a gramática sensível ao contexto é uma gramatica formal $G = (V,T,P,S)$:

- V é um conjunto finito de variáveis;
- T é um conjunto finito, disjunto de V, denominado terminais;
- P é um conjunto finito de regras, onde cada regra é uma variável e uma cadeia de variáveis terminais;
- S é a variável inicial;

Onde V(variável) não-Terminal e T(terminal) é sensível ao contexto se todas as regras em P são da forma:

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

Em que $A \in V$ (isto é, A é um único não-terminal), $\alpha, \beta \in (V \cup T)^*$ (ou seja, α e β são sequências de não-terminais e símbolo terminal) e $\gamma \in (V \cup T)^+$ (isto é, γ é uma sequência não vazia de não-terminais e terminais).

2.1.2.4 Gramática irrestrita

É a gramatica que apresenta total liberdade em relação a regras. A elas nenhuma restrição é imposta. O universo das linguagens definidas a partir do processo de geração estabelecido por esta gramática equivale ao conjunto das linguagens que essa classe gramatical é apta a criar (Linguagem Recursivamente Enumerável).

Segundo Menezes (2008):

> A linguagens Enumeráveis Recursivamente (também conhecidas como Tipo 0) são aquelas que podem ser reconhecidas por uma Máquina de Turing. Considerando que, segundo a Hipótese de Church, a Máquina de Turing é o mais geral dispositivo de computação, então a Classe das Linguagens Enumeráveis Recursivamente representa o conjunto de todas as linguagens que podem ser reconhecidas mecanicamente e em tempo finito.

2.1.3 Expressão Regular (ER)

As linguagens regulares pertencem à classe das linguagens mais simples. As cadeias (palavras) da linguagem regular podem ser reconhecidas através de algoritmos com pouca complexidade. A expressão regular é considerada geradora, pois se pode inferir como construir (“gerar”) as palavras da linguagem.

Uma ER sobre Σ pode ser definida como:

- ∅ é uma ER e denota a linguagem vazia
- ε é uma ER e denota a linguagem {ε}
- x ∈ Σ é uma ER e denota a linguagem {x}
- Se r e s são ER e denotam as linguagens R e S
  - (r+s) é ER ⇒ R ∪ S
  - (rs) é ER ⇒ RS = {uv | u ∈ Rev ∈ S}
  - (r∗) é ER ⇒ R∗
  - (r|s) é ER ⇒ R ou S

Qualquer símbolo x ∈ Σ ou ε é uma ER e denota a linguagem contendo exclusivamente a palavra x ou a palavra vazia respectivamente. Também podemos definir as ER através de união e concatenação como visto no slide. A união representa de duas linguagens R e S é o conjunto de cadeias que estão em R ou em S ou em ambas. O fechamento (ou estrela ou

fechamento de Kleene) de uma linguagem é denotado L^* e representa o conjunto de cadeias que podem ser formados tomando-se qualquer número de cadeias de L.

Existem algumas convenções para as expressões regulares, são elas:

- Concatenação sucessiva tem precedência sobre a concatenação e a união

  abc+a ⇒ (((ab)c)+(a))

- A concatenação tem precedência sobre a união.

  ab+a ⇒ ((ab)+(a))

Exemplo de Expressões Regulares:

- 001(0+1) denota L = {0010, 0011}
- 01(10+11) denota L = { 0110, 0111}
- (0 + 01)1 denota L = { 01, 011}
- ab* a denota L = {aa, aba, abba, abbba, ...}, isto é, L = { w | w inicia com um a seguido de zero ou mais b's e termina com um a}
- a(bc)*a denota L = {aa, abca, abcbca, abcbcbca, ...}, isto é, L = { w | w inicia-se com um a seguido de zero ou mais sequências de bc e termina com a}
- ab+ a denota L = {aba, abba, abbba, ...}, isto é, L = { w | w inicia-se com um a seguido de um ou mais b's e termina com a}
- (ab + bc)a denota L = {aba, bca}
- (a+b)* denota L = {a,b}* , isto é todas as palavras possíveis de serem formadas a partir do símbolos a e b, inclusive a palavra vazia.
- (a+b)* c(a+b)* denota L = { w | w possui exatamente um c sobre Σ = {a,b,c}* }

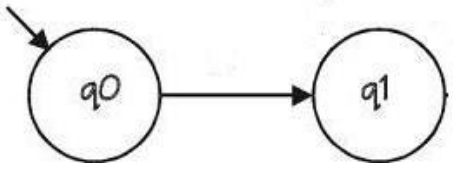
2.1.4 Autômato finito

Um autômato e um modelo de computador digital que pode ser usado para representar linguagens. Um autômato possui algumas características essenciais. Ele possui um mecanismo para ler entradas. Assumindo que essas entradas são cadeias sobre um dado alfabeto, escritas em uma fita de entrada, a qual o autômato pode ler, porem não alterar. O mecanismo de entrada pode ler a fita de entrada da esquerda para a direita, um símbolo por vez. O mecanismo de entrada pode detectar o fim da cadeia de entrada.

Uma maneira conveniente de se visualizar um autômato é através de grafos. Um grafo e um construto construído de dois conjuntos finitos, um conjunto V = {v_1, v_2,..., v_n} de vértices e o conjunto E = {c_1, c_2,..., c_n} de arestas. Cada aresta é um par de vértices de V, por exemplo, e1 = {v_j, v_k};

Os grafos são representados através de diagramas nos quais os vértices são representados como círculos e as arestas como linhas com setas conectando os vértices. O grafo com vértices {q_0, q_1} e aresta {(q_0, q_1)} é ilustrado na figura 1.

Figura 1 - Autômato finito.



Fonte: Adaptado de Rangel e Guedes (2017).

Uma sequencia de arestas (v_i, v_j), (v_j, v_k),...(v_m, v_n) diz-se um caminho de vi a v_n. O comprimento de um caminho e o número total de arestas que ele atravessa indo do vértice inicial ao vértice final. Um caminho no qual nenhuma aresta e repetida diz-se uma trilha. Uma trilha e simples se nenhum vértice e repetido. Se as arestas de um grafo são rotuladas, podemos falar do rotulo do caminho. Este rótulo e a sequencia de rótulos das arestas encontradas quando se percorre a trilha.

Os autômatos finitos são divididos em duas classes, são elas:

• **Autômatos Finitos Determinísticos (AFD):**

Um autômato finito determinístico e definido pela quíntupla $\{\Sigma, Q, \delta, q_0, F\}$,

onde:

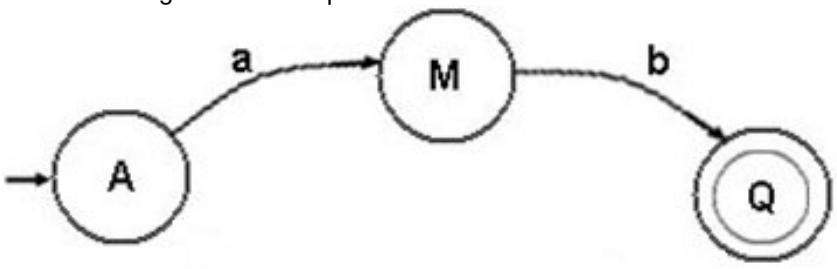
- Σ é um conjunto de símbolos chamado alfabeto de entrada,
- Q é um conjunto finito de estados internos,
- δ é a função programa ou função de transição,

$$\delta: Q \times \Sigma \rightarrow 2^{Q}$$

- q_0 e o estado inicial ($q_0 \in Q$),
- F e o conjunto de estados finais. Para visualizar e representar autômatos usa-se grafos de transições nos quais os vértices representam estados e as arestas representam transições.

Os rótulos dos vértices são os nomes dos estados enquanto os rótulos das arestas são os valores correntes dos símbolos de entrada. A figura 2 mostra um autômato finito determinístico. Os autômatos determinísticos possuem papel relevante nesse trabalho, uma vez que uma linguagem é regular se e somente se existe um autômato finito determinístico para reconhecer todas as suas cadeias conforme Acioly, Bedregal & Lyra (2002), portanto uma linguagem é o conjunto de todas as cadeias reconhecidas por um autômato finito determinístico. Desta forma, se $M = \{\Sigma, Q, \delta, q_0, F\}$ é um reconhecedor finito determinístico, então a ele esta associado um grafo de transição G tendo exatamente |Q| vértices, cada um rotulado com um q_i | Q diferente. Para cada regra de transição $\delta(q_i,a) = q_j$, o grafo tem uma aresta (q_i,q_j) rotulada por a.

Figura 2 - Exemplo de autômato determinístico.



Fonte: Adaptado de Tavares (2017)

Autômatos finitos determinísticos possuem a característica importante que o caracteriza, que é, para cada estado, deve consumir algum símbolo de entrada a fim de atingir algum estado do autômato.

A introdução da função programa estendida δ^*: Q x $\Sigma^* \rightarrow$ Q neste momento se faz conveniente. Formalmente podemos definir δ^* recursivamente por:

1. $\delta^*(q,\Lambda) = q$
2. $\delta^*(q,wa) = \delta(\delta^*(q,w),a)$.

A linguagem aceita por um AFD $M = \{Q, \Sigma, \delta, q_0, F\}$ é o conjunto de todas as cadeias sobre Σ aceita, de maneira formal temos:

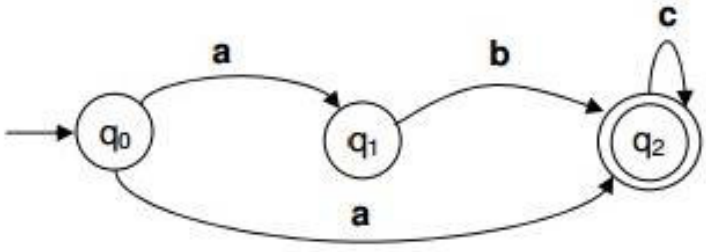
$$L(M) = \{w \in \Sigma^* / \delta^*(q_0, w) \in F\}, \text{ onde:}$$

Seja $M = \{Q, \Sigma, \delta, q_0, F\}$ um AFD, e seja G seu grafo associado. Então para todo q_i, $q_j \in Q$ e $w \in \Sigma+$, $\delta^*(q_i, w) = q_j$ se e somente se existe em G um caminho com rotulo w de q_i a q_j.

• Autômatos Finitos Não-Determinísticos (AFND):

Outro autômato que merecem atenção especial são os autômatos finitos não determinísticos que tem por característica principal a possibilidade de se atingir um conjunto de estados a partir de um determinado estado consumindo um determinado símbolo de entrada, ou sem consumir nenhum símbolo, dai a expressão “não determinística”, pois não se pode prevê o próximo estado a ser alcançado a partir de um símbolo em um determinado momento (figura 3). Autômatos finitos determinísticos não possuem esta característica, pois, para cada estado, ele deve consumindo algum símbolo de entrada a fim de atingir algum estado do autômato.

Figura 3 - Autômato Finito Não-Determinístico.



Fonte: Silva (2007).

Segundo Acioly, Bedregal & Lyra (2002), não determinismo significa uma escolha de movimentos para um autômato. Em vez de prever um único movimento em cada situação, permitimos um conjunto de movimentos possíveis. Formalmente, conseguimos isso definindo a função de transição δ tal que sua imagem seja um conjunto de estados possíveis. Segundo Acioly, Bedregal & Lyra (2002), um autômato finito não determinístico (AFN) e definido pela quíntupla {Q, Σ, δ, q_0, F} onde Q, Σ, q_0, F são definidos como para um autômato finito determinístico, porém:

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

Existem duas diferenças principais entre essa definição e a de autômato finito determinístico. Aqui a imagem de δ e o conjunto das partes de Q. Então seu valor não e um único elemento de Q, mas um subconjunto dele. Este subconjunto define o conjunto de todos os estados possíveis que podem ser alcançados pela transição.

Analogamente a autômatos finitos determinísticos, os autômatos finitos não determinísticos podem ser representados por grafos de transição. Também da mesma forma, os autômatos finitos não determinísticos também reconhecem linguagens, porem as linguagens regulares somente podem ser reconhecidas por AFD´s, daí a importância de se estabelecer uma equivalência entre AFD´s e AFN´s. Esta equivalência é verdadeira e segundo Acioly, Bedregal & Lyra, (2002), dois reconhecedores são equivalentes se eles reconhecem a mesma linguagem. O procedimento de transformação de equivalência é mostrado a seguir na seção 2.1.4.1.

• **Autômatos Finitos Não-Determinísticos com Movimentos Vazios (AFND-ε)**

É uma extensão (variação) de um autômato finito não-determinístico (AFND), que permite a transição para um novo estado sem consumir qualquer caractere da entrada. As transições que não consomem nenhum caractere de entrada são chamadas de *transições ε* ou *transições λ*.

Usando formalismo, um AFε é definido pela quíntupla:

M = {Σ, Q, δ, q_0, F} onde:

- Σ – alfabeto de símbolos de entrada
- Q – conj. de estados possíveis do autômato o qual ´e finito

- δ – função de transição (δ: Q x (Σ ∪ {ε}))
- q_0 – estado inicial ($q_0 \in Q$)
- F – conjunto dos estados finais tal que F está contido em Q

2.1.4.1 Equivalência entre AFD e AFN

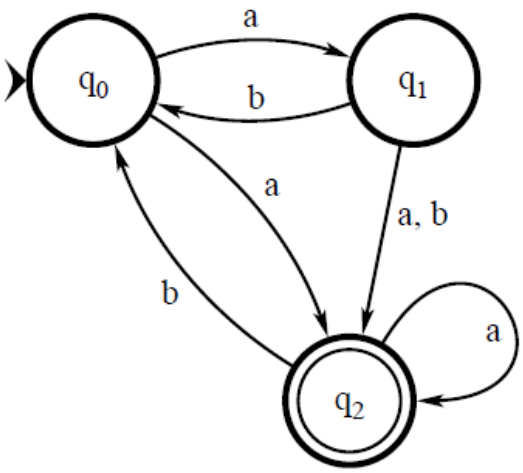
Um autômato finito não-determinístico pode se transformar em um autômato finito equivalente, isso é, dois autômatos que aceitam a mesma linguagem.

Para construir um AFD a partir de um AFN qualquer, devemos realizar os seguintes passos:

1. Construir a tabela de transições do AFN (φ), uma tabela de transição de estados é uma tabela que mostra para qual estado (ou estados, como nesse caso, por ser um autômato finito não-determinístico) a máquina de estados finitos irá se mover, com base no estado atual e em outras entradas.
2. Construir a tabela de transições do AFD (δ) através do produto cartesiano dos estados de φ, incluindo como último conjunto o vazio.
3. Mostrar todos os conjuntos que contém como elemento estados finais como novo estado final de δ.
4. Verificar a ocorrência de cada conjunto de δ em relação a um símbolo e colocar como resultado o conjunto correspondente que pertence a _. Quando existir mais de um elemento no conjunto a ocorrência passa a ser a união das ocorrências de todas as transições.
5. Eliminar as linhas que possuem transições somente com saídas (não existe transição que chega até ela, isto é, estado inacessível).
6. Montar o AFD a partir de δ.
7. Eliminar os estados que não possuem saída para outro estado e não são finais.

Exemplo: Considere o AFN seguinte na figura 4:

Figura 4 – Autômato Finito Não-Determinístico.



Fonte: Sakata (2007).

1. Construir a tabela de transições do AFN (φ).

φ	a	b
q_0	$\{q_1, q_2\}$	-
q_1	$\{q_2\}$	$\{q_0, q_2\}$
q_2	$\{q_2\}$	$\{q_0\}$

Fonte: Sakata (2007).

2. Construir a tabela de transições do AFD (δ) através do produto cartesiano dos estados de φ, incluindo como _ultimo conjunto o vazio:

δ	a	b
$S_0 = \{q_0\}$		
$S_1 = \{q_1\}$		
$S_2 = \{q_2\}$		
$S_3 = \{q_0, q_1\}$		
$S_4 = \{q_0, q_2\}$		
$S_5 = \{q_1, q_2\}$		
$S_6 = \{q_0, q_1, q_2\}$		
$S_7 = \{\ \}$		

Fonte: Sakata (2007).

Sempre existirá 2^k combinações, onde k é o número de estados do AFN.

3. Mostrar todos os conjuntos que contém como elemento estados finais como novo estado final de δ:

δ	a	b
$S_0 = \{q_0\}$		
$S_1 = \{q_1\}$		
$\mathbf{S_2 = \{q_2\}}$		
$S_3 = \{q_0, q_1\}$		
$\mathbf{S_4 = \{q_0, q_2\}}$		
$\mathbf{S_5 = \{q_1, q_2\}}$		
$\mathbf{S_6 = \{q_0, q_1, q_2\}}$		
$S_7 = \{\ \}$		

Fonte: Sakata (2007).

4. Verificar a ocorrência de cada conjunto de δ em relação a um símbolo e colocar como resultado o conjunto correspondente que pertence a δ. Quando existir mais de um elemento no conjunto a ocorrência passa a ser a união das ocorrências de todas as transições.

δ	a	b
$S_0 = \{q_0\}$	S_5	S_7
$S_1 = \{q_1\}$	S_2	S_4
$\mathbf{S_2 = \{q_2\}}$	S_2	S_0
$S_3 = \{q_0, q_1\}$	S_5	S_4
$\mathbf{S_4 = \{q_0, q_2\}}$	S_5	S_0
$\mathbf{S_5 = \{q_1, q_2\}}$	S_2	S_4
$\mathbf{S_6 = \{q_0, q_1, q_2\}}$	S_5	S_4
$S_7 = \{\ \}$	S_7	S_7

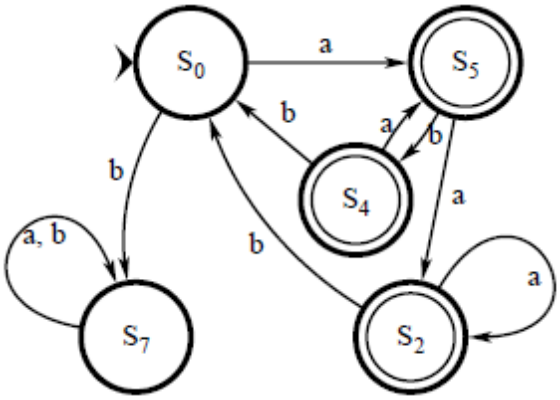
Fonte: Sakata (2007).

5. Eliminar as linhas que possuem transições somente com saídas, oi seja, não existe nenhuma transição que chega até ela (estados inacessíveis).

δ	a	b
$S_0 = \{q_0\}$	S_5	S_7
$\mathbf{S_2 = \{q_2\}}$	S_2	S_0
$\mathbf{S_4 = \{q_0, q_2\}}$	S_5	S_0
$\mathbf{S_5 = \{q_1, q_2\}}$	S_2	S_4
$S_7 = \{\ \}$	S_7	S_7

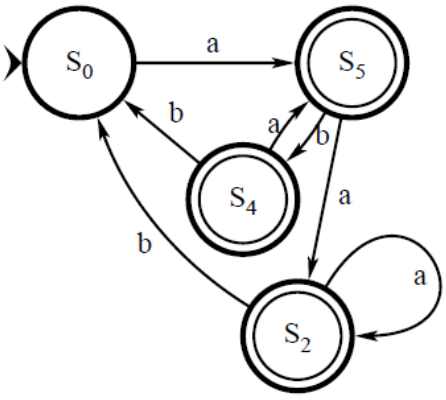
Fonte: Sakata (2007).

6. Montar o AFN a partir de δ:



Fonte: Sakata (2007).

7. Eliminar os estados que não possuem saída para outro estado e não são finais:



Fonte: Sakata (2007).

2.1.4.2 Minimização de Autômato Finito

Ao construirmos um AFD a partir de um AFN, percebe-se que o número de estados pode crescer. Para facilita a implementação devemos deixar o AFD com o mínimo de estados possíveis, esse processo é chamado de Minimização de um AFD, isto é, construir um AFD equivalente com o mínimo de estados possíveis.

O algoritmo de minimização unifica os estados equivalentes de um autômato. Dois estados q e p são ditos equivalentes se, e somente se, para qualquer cadeia w pertencente a Σ*, δ(q,w) e δ(p, w) resultam simultaneamente em estados finais, ou não-finais, ou seja, o processamento de uma entrada qualquer a partir de estados equivalentes gera, em qualquer caso, o mesmo resultado: aceitação ou rejeição. Se dois estados não são equivalentes, eles são ditos distinguíveis.

Um autômato Finito a ser minimizado deve satisfazer aos seguintes pré-requisitos:

1. Deve ser determinístico;
2. Não pode ter estados inacessíveis (não-atingíveis a partir de estado inicial);
3. A função programa deve ser total (a partir de qualquer estado são previstas transições para todos os símbolos do alfabeto). Para transformar a função programa em total, é necessário introduzir um novo estado não-final *d* e incluir as transições não-previstas, tendo *d* como estado destino. Por fim incluir um ciclo em *d* para todos os símbolos do alfabeto.

Supondo que um autômato Finito Determinístico $M = (\Sigma, Q, \delta, q_0, F)$ que satisfaz aos pré-requisitos de minimização. Os passos do algoritmo de minimização são os seguintes:

1 Tabela. Construir uma tabela relacionando os estados distintos, onde cada par de estados ocorre somente uma vez.

2. Marcação dos estados trivialmente não-equivalentes. Marcar todos os pares do tipo {estado *final, estado não-final}*, pois, obviamente, os estados finais não são equivalentes a não-finais;
3. Marcação dos estados não-equivalentes — para cada par (q_u, q_v) não marcado e para cada símbolo a ∈ Σ, δ(q_u, a) = p_u e δ(q_v, a) = p_v:

```
para cada símbolo a do alfabeto
  se pu == pv então
     qu é equivalente a qv e não deve ser marcado
  se pu != pv e o par {pu,pv} não está marcado
    então
    {qu,qv} é incluído em uma lista a partir de {pu,pv}
    para posterior análise
  se pu != pv e o par {pu,pv} está marcado então
    {qu,qv} não é equivalente e deve ser marcado e
     consequentemente
        se {qu,qv} encabeça uma lista de pares então
            marcar todos os pares da lista e assim
             recursivamente se algum par da lista
              encabeça outra
```

Fonte: Fredrich et al.(2009).

4. fazer a unificação dos estados equivalentes, que são aqueles pares que não estão marcados na tabela, levando em consideração seus tipos:

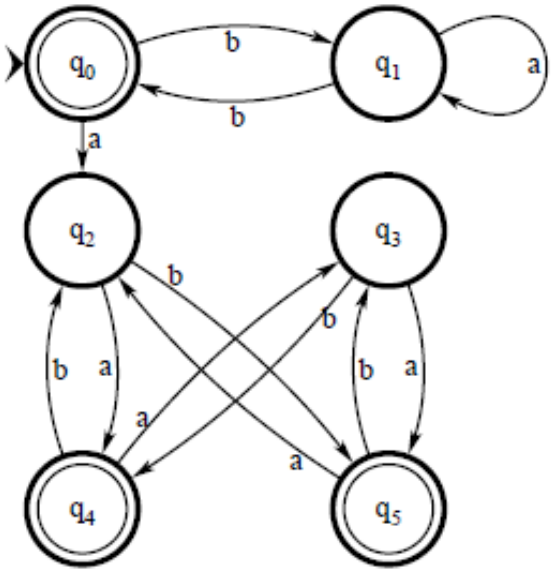
```
se são não-finais então
  tornam-se um único estado não-final
se são finais então
  tornam-se um único estado final
se um deles é inicial então
  tornam-se um único estado inicial
```

Fonte: Fredrich et al.(2009).

Por fim excluir os estados inúteis, sendo um estado inútil aquele que é do tipo não-final e a partir dele não é possível atingir um estado final.

Será utilizado o AFD da figura 5 para exemplificar os passos da minimização. Este autômato satisfaz os pré-requisitos de minimização e consequentemente não é necessário incluir o estado d.

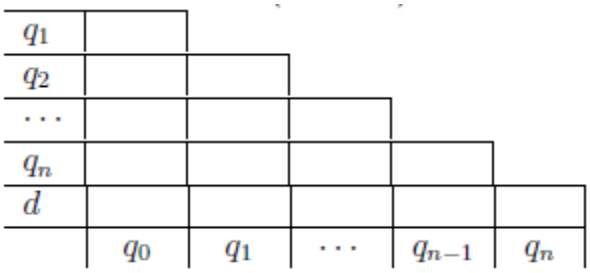
Figura 5 – Autômato Finito Determinístico.



Fonte: Sakata (2007).

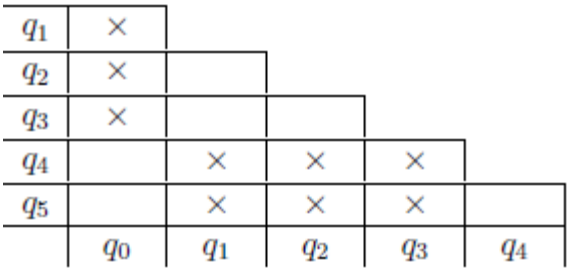
Os passos do algoritmo são como segue:

1. Tabela de estados (genérica)



Fonte: Sakata (2007).

2.



Fonte: Sakata (2007).

3.

a. Análise do par $\{q_0, q_4\}$

$\delta(q_0, a) = q_2$ $\delta(q_0, b) = q_1$

$\delta(q_4, a) = q_3$ $\delta(q_4, b) = q_2$

Como $\{q_1, q_2\}$ e $\{q_2, q_3\}$ são não-marcados, então $\{q_0, q_4\}$ é incluído nas listas encabeçadas por $\{q_1, q_2\}$ e $\{q_2, q_3\}$.

b. Análise do par $\{q_0, q_5\}$

$\delta(q_0, a) = q_2$ $\delta(q_0, b) = q_1$

$\delta(q_5, a) = q_2$ $\delta(q_5, b) = q_3$

Como $\{q_1, q_3\}$ é não-marcado e $\{q_2, q_2\}$ é equivalente, então $\{q_0, q_5\}$ é incluído na lista encabeçada por $\{q_1, q_3\}$.

c. Análise do par $\{q_1, q_2\}$

$\delta(q_1, a) = q_1$ $\delta(q_1, b) = q_0$

$\delta(q_2, a) = q_4$ $\delta(q_2, b) = q_5$

Como $\{q_1, q_4\}$ é marcado, então $\{q_1, q_2\}$ também deve ser marcado. Como $\{q_1, q_2\}$ encabeça uma lista, o par $\{q_0, q_4\}$ deve ser marcado.

d. Análise do par $\{q_1, q_3\}$

$\delta(q_1, a) = q_1$ $\delta(q_1, b) = q_0$

$\delta(q_3, a) = q_5$ $\delta(q_3, b) = q_4$

Como $\{q_1, q_5\}$ bem como $\{q_0, q_4\}$ são marcados, então $\{q_1, q_3\}$ também deve ser marcado. Como $\{q_1, q_3\}$ encabeça uma lista, o par $\{q_0, q_5\}$ deve ser marcado.

e. Análise do par $\{q_2, q_3\}$

$\delta(q_2, a) = q_4$ $\delta(q_2, b) = q_5$

$\delta(q_3, a) = q_5$ $\delta(q_3, b) = q_4$

Como $\{q_4, q_5\}$ é não-marcado, então $\{q_2, q_3\}$ é incluído na lista encabeçada por $\{q_4, q_5\}$.

f. Análise do par $\{q_4, q_5\}$

$\delta(q_4, a) = q_3$ $\delta(q_4, b) = q_2$

$\delta(q_5, a) = q_2$ $\delta(q_5, b) = q_3$

Como $\{q_2, q_3\}$ é não-marcado, então $\{q_4, q_5\}$ é incluído na lista encabeçada por $\{q_2, q_3\}$.

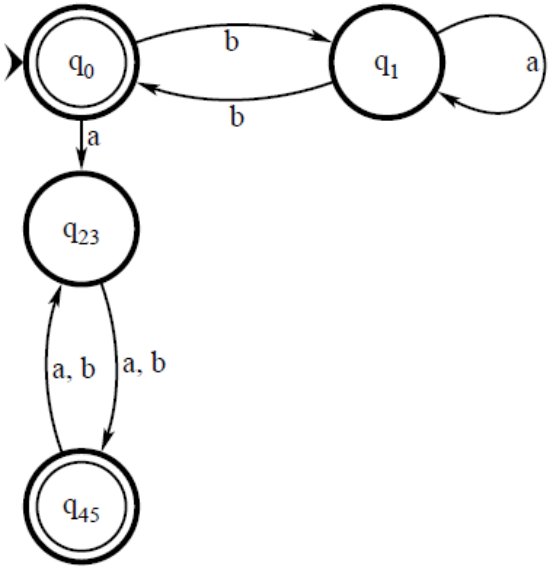
Tabela de estados resultante:

q_1	×				
q_2	×	⊗			
q_3	×	⊗			
q_4	⊗	×	×	×	
q_5	⊗	×	×	×	
	q_0	q_1	q_2	q_3	q_4

Fonte: Sakata (2007).

4. Como os pares $\{q_2, q_3\}$ e $\{q_4, q_5\}$ são não-marcados, as seguintes unificações podem ser feitas:
   a. q_{23} representa a unificação dos estados não-marcados, q_2 e q_3;
   b. q_{45} representa a unificação dos estados finais q_4 e q_5;

O autômato mínimo para o AFD do exemplo é:



Fonte: Sakata (2007).

5. Nesse exemplo, não houve um estado inútil para ser excluído.

2.1.4.3 Conversão de expressões regulares para Autômato Finito

O Algoritmo de Thompson foi criado por Kenneth Thompson (Nova Orleans, 4 de fevereiro de 1943) um cientista da computação, conhecido pela sua influência no sistema operacional UNIX e Dennis MacAlistair Ritchie (Bronxville, 9 de setembro de 1941 — Berkeley Heights, 12 de outubro de 2011) um cientista da computação estadunidense, notável pela sua influência em linguagens de programação como ALTRAN, B, BCPL e C, e em sistemas operacionais como o Multics e o UNIX.

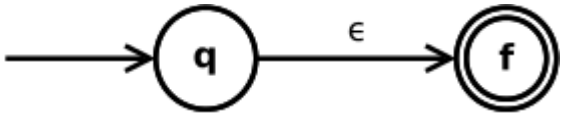
Em ciência da computação, Algoritmo de Thompson é um algoritmo para transformar uma expressão regular em um equivalente autômato finito não determinístico (AFN). Este AFN pode ser usado para casar palavras com expressões regulares.

O algoritmo funciona de forma recursiva dividindo uma expressão em sub expressões constituintes, das quais o AFN será construído usando um conjunto de regras. Mais precisamente, a partir de uma expressão regular E, o autômato A obtido com o δ da função de transição respeita as seguintes propriedades:

- *A* tem exatamente um estado inicial q_0, que não é acessível a partir de nenhum outro estado. Ou seja, para qualquer estado *q* e qualquer letra *a*, δ(q,a) não contém q_0.
- *A* tem exatamente um estado final q_f, que não é co-acessível a partir de nenhum outro estado. Isto é, para qualquer letra *a*, $\delta(q_f,a) = \emptyset$.
- Seja *c* o número de concatenação da expressão regular *E* e seja *s* o número de símbolos à parte de parênteses — Isto é, |, *, *a* e *ε*. Em seguida, o número de estados de A é 2s − c (linear no tamanho de E). Em seguida, o número de estados de A é 2s − c (linear no tamanho de E).
- O número de transições que deixam qualquer estado é de no máximo dois
- Uma vez que um AFN de m estados e no máximo e transições de cada estado pode corresponder a uma cadeia de comprimento n em tempo O, um AFN de Thompson pode fazer a correspondência de padrões em tempo linear, assumindo um alfabeto de tamanho fixo.

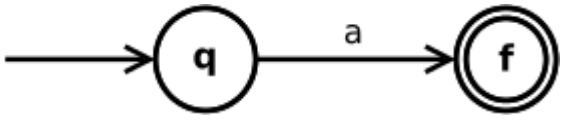
Regras:

A expressão-vazia (ε) é convertida em



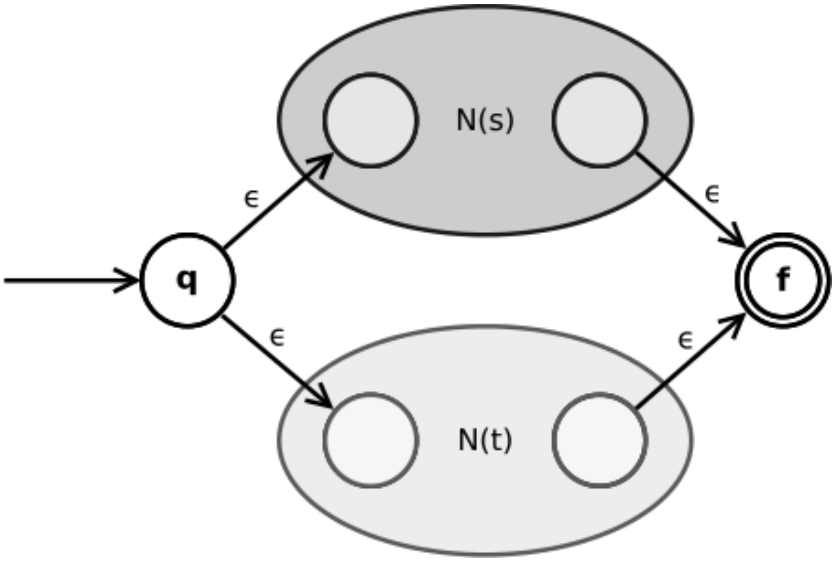
Fonte: Xing (2007).

Um símbolo '*a*' de um alfabeto de entrada é convertido em



Fonte: Xing (2007).

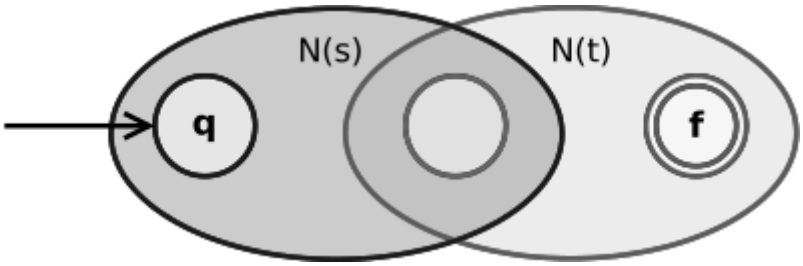
A expressão de união *s*|*t* é convertida em



Fonte: Xing (2007).

O estado *q* passa através de ε para o estado inicial de *N*(*s*) ou *N*(*t*). Seus estados finais se tornam estados intermediários de todo AFN e mescla através de duas transições ε para o estado final do AFN.

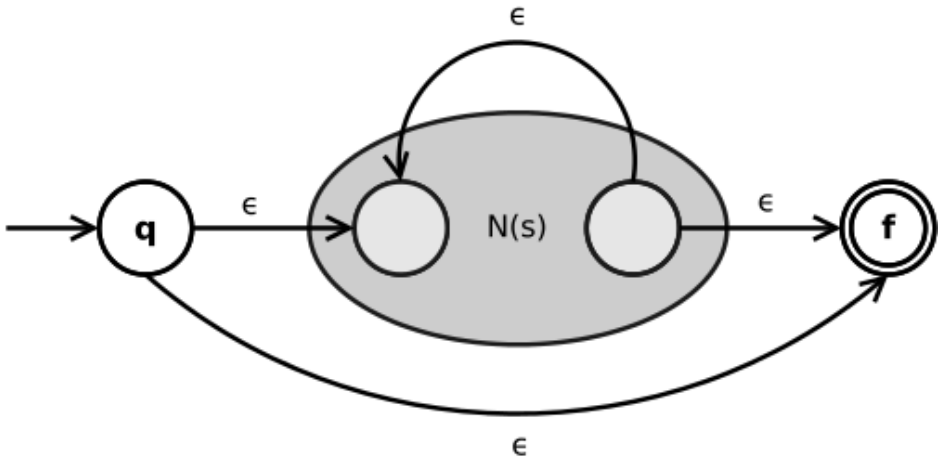
A expressão de concatenação *st* é convertida em



Fonte: Xing (2007).

O estado inicial de *N*(*s*) é o estado inicial de todo o AFN. O estado final de *N*(*s*) torna-se o estado inicial de *N*(*t*). O estado final de *N*(*t*) é o estado final de todo o AFN.

A expressão *Fecho de Kleene* s^* é convertida em



Fonte: Xing (2007).

Uma transição ε conecta o estado inicial e final do AFN com o sub-AFN *N*(*s*) no meio. Outra transição ε do estado final interior para o estado inicial interior de *N*(*s*) permite a repetição da expressão *s* de acordo com o operadora estrela.

2.1.5 Autômatos de Pilhas ou Autômatos *push-down* (APD)

Um autômato com pilha (PDA) é uma sêxtupla M = (Q, Σ, Γ, δ, q_0, F), onde

• Q é um conjunto finito de estados;

• Σ é o alfabeto de entrada;

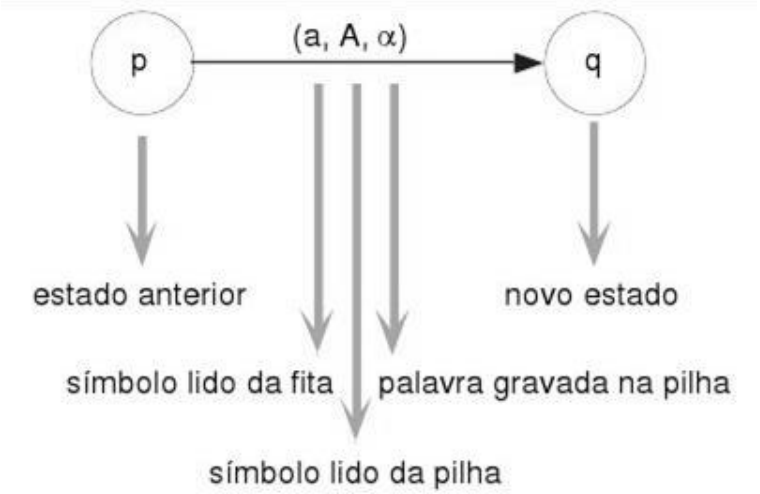
• Γ é o alfabeto da pilha;

• $q_0 \in Q$ é o estado inicial;

• F é o subconjunto de estados finais de Q;

• δ é uma função δ: Q × (Σ ∪ {λ}) × (Γ∪ {λ}) → P(Q × (Γ∪ {λ})), chamada de função de transição. (figura 6)

Figura 6 – Representa a função programa de um APD como um grafo.



Fonte: Menezes (2008).

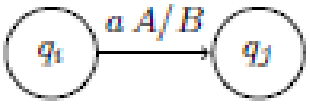
O alfabeto da pilha será representado por letras maiúsculas e uma pilha vazia será denotada pela string vazia λ. No início da computação, o PDA começa no estado inicial, com a string de entrada na fita e com a pilha vazia.

A função de transição será representada da seguinte maneira: $\delta(q_i, a, A) = \{(q_j, B),(q_k, C)\}$. A transição $(q_j , B) \in \delta(q_i , a, A)$, por exemplo, pode ser interpretada usando a ideia de uma máquina com pilha. A transição faz com que a máquina:

- mude o estado de q_i para q_j ;
- processe o símbolo a;
- remova o símbolo A do topo da pilha;
- coloque o símbolo B no topo da pilha.

O diagrama de estados do PDA terá como rótulos das arestas o símbolo de entrada a ser processado e as operações da pilha (qual elemento sai do topo/qual elemento entra no topo). A transição $\delta(q_i , a, A) = \{(q_j , B)\}$, por exemplo, é representada como mostrado na figura 7.

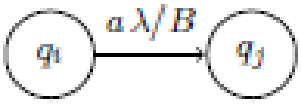
Figura 7 – Operação de pilha: processa ‘a’, tira ‘A’ do topo, entra ‘B’ no topo



Fonte: Silva (2014).

O PDA muda do estado q_i para o estado q_j processando o símbolo a $\in \Sigma$. Ainda, o símbolo A $\in \Gamma$ é removido do topo da pilha enquanto o símbolo B $\in \Gamma$, em seguida, é colocado no topo. Na função de transição, podemos ter λ como símbolo de entrada e como símbolo da pilha. Uma transição como $(q_j, B) \in \delta(q_i, a, \lambda)$ entra o estado q_j e adiciona B no topo da pilha (figura 8).

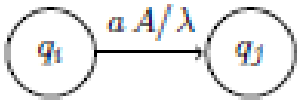
Figura 8 - Operação de pilha: processando o símbolo a, remove 'A' do topo da pilha e coloca 'B' no topo.



Fonte: Silva (2014).

A transição $(q_j, \lambda) \in \delta(q_i, a, A)$ entra o estado q_j e remove A do topo da pilha (figura 9).

Figura 9 - Operação de pilha: remove 'A' do topo da pilha.



Fonte: Silva (2014).

Exemplo de Autômato de Pilha:

$$L_1 = \{a^n b^n \mid n \geq 0\}$$

$$M_1 = (\{q_0, q_1, q_f\}, \{a, b\}, \delta_1, q_0, \{q_f\}, \{B\})$$

$$\delta_1(q_0, a, \epsilon) = \{(q_0, B)\}$$

$$\delta_1(q_0, b, B) = \{(q_1, \epsilon)\}$$

$$\delta_1(q_0, ?, ?) = \{(q_f, \epsilon)\}$$

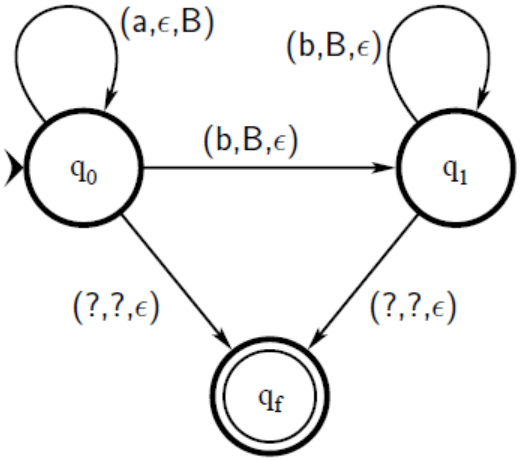
$$\delta_1(q_1, b, B) = \{(q_1, \epsilon)\}$$

$$\delta_1(q_1, ?, ?) = \{(q_f, \epsilon)\}$$

O autômato M_1 é determinístico. No estado q_0, para cada símbolo a lido da fita é armazenado um símbolo B na pilha. No estado q_1, é realizado um batimento, verificando se para cada símbolo b da fita, existe um correspondente B na pilha. O algoritmo somente aceita se ao terminar de ler

toda a palavra de entrada a pilha estiver vazia. O autômato pode ser representado pelo grafo da figura 10.

Figura 10 - Autômato de Pilha *M1*.



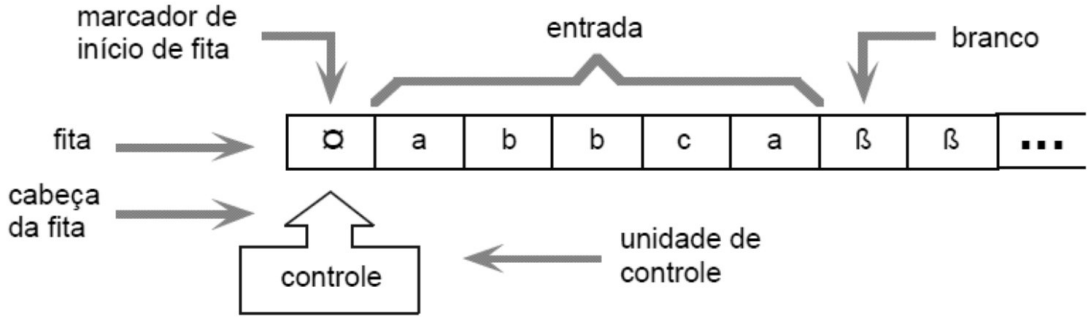
Fonte: Silva (2014).

2.1.6 Máquina de Turing (MT)

O modelo proposto por Turing e 1936, conhecido com Máquina de Turing, consiste basicamente em 3 partes (figura 11):

- Fita: Usada simultaneamente como dispositivo de entrada, saída e memória de trabalho;
- Unidade de Controle: Reflete o estado corrente da máquina. Possui uma unidade de leitura e gravação (cabeça da fita) a qual acessa uma célula da fita de cada vez e movimenta-se para esquerda ou direita;
- Programa ou Função de Transição: Função que comanda as leituras e gravações, o sentido de movimento da cabeça e define o estado da máquina.

Figura 11 - Modelo Máquina de Turing.



Fonte: Casillo (2017).

Usando uma definição formal podemos dizer que:

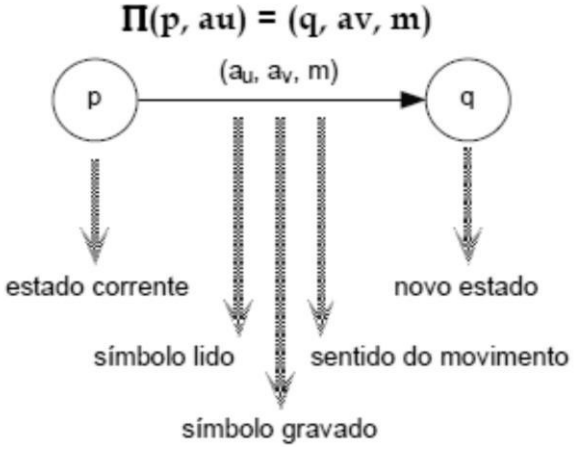
Uma Máquina de Turing é uma 8-upla: M = (∑, Q, Π, q_0, F, V, β, ¤)

- ∑ alfabeto de símbolos de entrada;
- Q conjunto de estados possíveis da máquina, o qual é finito;
- Π programa ou função de transição: (é uma função parcial)
- q_0 estado inicial da máquina, tal que q_0 é elemento de Q;
- F conjunto de estados finais, tal que F está contido em Q;
- V alfabeto auxiliar;
- β símbolo especial branco;
- ¤ símbolo especial marcador de início da fita

Símbolo de início de fita ¤, ocorre exatamente uma vez e sempre na célula mais à esquerda da fita, auxiliando na identificação de que a cabeça da fita se encontra na célula mais à esquerda da fita. A função programa Π, considera o estado corrente e o símbolo lido da fita para determinar o novo estado, o símbolo a ser gravado e o sentido de movimento da cabeça da fita que pode ser esquerda (E) ou direita (D).

O programa pode ser representado como um grafo finito, como na figura 12.

Figura 12 - Representa a função programa de uma MT como um grafo.



Fonte: Casillo (2017).

Ou pode ser representado por uma Tabela de Transição, como na figura 13.

Figura 13 - Tabela de Transição de MT.

$$\Pi(p, au) = (q, av, m)$$

Π	¤	...	au	...	av	...	ß
p			(q, av, m)				
q							
...							

Fonte: Casillo (2017).

O processamento de uma Máquina de Turing M = (∑, Q, Π, q_0, F, V, β, ¤) para uma palavra de entrada w consiste na sucessiva aplicação da função programa, a partir do estado inicial q0 e da cabeça posicionada na célula mais à esquerda da fita até ocorrer uma condição de parada. O Processamento de uma máquina M para a entrada w pode parar ou ficar em loop infinito.

A parada de uma Maquina de Turing pode ser de duas maneiras: aceitando ou rejeitando a entrada w. As condições de parada são as seguintes:

Estado Final: A máquina assume um estado final: a máquina para, e a palavra de entrada é aceita;

Função Indefinida: A função programa é indefinida para o argumento (símbolo lido e estado corrente): a máquina para, e a palavra de entrada é rejeitada;

Movimento Inválido: O argumento corrente da função programa define um movimento à esquerda e a cabeça da fita já se encontra na célula mais à esquerda: a máquina para, e a palavra de entrada é rejeitada.

2.2 Relevância de Linguagens Formais e Autômatos para Computação

Linguagens Formais e Autômatos têm um grande destaque e importância na área da Computação, principalmente na Teoria da Computação, por oferecer a fundamentação teórica e vários subsídios para aplicações práticas.

Segundo Ramos (2009) o estudo de matérias teóricas como é o caso de Linguagens Formais e Autômatos, não devem ser menosprezadas na

grade curricular, já que elas são essenciais para formação de profissionais preparados para lidar não apenas os desafios atuais da computação, mas especialmente aqueles que ainda estão por vir.

De acordo com José Neto (2009) a importância do estudo de LFA se evidencia de diversas formas:

- Propicia uma olhar amplo dos fundamentos científicos da computação;
- Propicia uma base teórica para a área, por meio do estudo de assuntos como decidibilidade, computabilidade e complexibilidade computacional;
- Oferece um forte embasamento na criação de várias aplicações computacionais, principalmente nos relacionados a processamento de linguagem, reconhecimento de padrões e a modelagem de sistemas;
- Firma uma forte ligação entre a teoria e a pratica computacional;
- Possibilita que os conceitos e os resultados teóricos correspondentes a linguagem, seus geradores, reconhecedores e analisadores sejam empregados de forma rígida;
- É um tema que se apresenta vasto e menos distanciado dos demais assuntos estudados na Teoria da Computação.

Ramos (2009) e José Neto (2009) evidenciam a ampla aplicabilidade dos conceitos de LFA nas demais disciplinas dos cursos de computação, qualificando o profissional a desenvolver argumentações matemáticas formais e rigorosas, fornecendo uma familiaridade com as bases e fundamentos da Ciência da Computação.

3 Utilização de softwares educativos na disciplina de Linguagens Formais

A Teoria de Linguagens Formais e Autômatos como visto no capítulo 2, é o estudo de modelos matemáticos que permitem a especificação e o reconhecimento de linguagens, suas classificações, estruturas, propriedades, características e inter-relacionamentos, sendo assim uma disciplina de bastante teórica.

E justamente por ser uma disciplina teórica com predominância de formalismos matemáticos e demonstração de teoremas, acaba por se tornar uma disciplina com um grau elevado de dificuldade e de escasso interesse por parte dos alunos. Além disso, um fato que contribui para esse desinteresse é o método tradicional no qual é ensinada, através de aulas expositivas e teóricas, com resolução de problemas que são bastante suscetíveis a erros, com grandes grafos, cheios de estados e transições, tornado a tarefa cansativa e tediosa. Sendo assim a árdua tarefa de tentar deixar esse conteúdo mais atrativo para os alunos fica a cargo dos professores, que tem que buscar novos métodos para o ensino da disciplina.

Para conseguir esse objetivo de atrair alunos, a principal medida deve ser a transmissão do conteúdo de forma clara, traduzindo toda a linguagem matemática em termos mais coloquiais. Além do uso de aplicações práticas do conteúdo que é predominantemente teórico, demostrando a importância da teoria lecionada. E é aí que o uso de softwares como ferramentas educacionais se encaixam como um facilitador no objetivo de motivar o interesse dos alunos pela disciplina.

> As ferramentas computacionais, utilizadas como auxiliares do processo de ensino-aprendizagem (...) rendem largas oportunidades para a construção crítica do conhecimento (...) e podem oportunizar, no contexto acanhado da sala de aula e para além dele, a dinâmica da experimentação. [Oliveira, 2004].

Existem atualmente algumas ferramentas que dão suporte no ensino de Linguagens Formais e Autômatos tais com o JFLAP (2017), um software desenvolvido e mantido na Universidade de Duke (Durham, Carolina do Norte, EUA), com o apoio da National Science Foundation desde 1993, Auger (2017),

criado por Charbel Szymanski e posteriormente utilizado como ferramenta de apoio ao ensino na disciplina de Linguagens Formais e Autômatos do Curso de Ciência da Computação da Universidade do Sul de Santa Catarina e o Simulador de Autômatos (2017), resultado de trabalho de conclusão de Curso (em Engenharia da Computação), na Universidade de Uberaba - UNIUBE entre outros softwares.

Essas ferramentas são capazes de ajudar o entendimento por parte dos alunos, possibilitando a construção de modelos durante exercícios e permitindo a verificação dos mesmos.

3.1 Softwares de ensino em Linguagens Formais

Estão presentes no mercado alguns softwares educativos para o auxilio no aprendizado de diversos conteúdos da Computação. O uso destes softwares se destaca por motivar o aluno para o estudo da disciplina, permitindo testes rápidos de novas ideias ou conceitos ensinados.

Desta forma nas sessões 3.1.1 a 3.1.3 serão apresentados alguns softwares educativos que auxiliam no estudo de Linguagens Formais e Autômatos nos cursos da área de Computação, atendendo desta maneira a necessidade por um ambiente de ensino a esta disciplina.

3.1.1 JFLAP (Java Formal Languages and Automata Package)

O JFLAP é um pacote de ferramentas gráficas desenvolvido em Java que pode ser utilizado no auxilio e na aprendizagem dos conceitos básicos de Linguagens Formais e Teoria dos Autômatos.

JFLAP começou como uma série de ferramentas no Rensselaer Polytechnic Institute por volta de 1990, com os alunos trabalhando sob a direção da professora Susan Rodger e contando com o apoio da National Science Foundation desde 1993. O projeto JFLAP mudou para a Duke University em 1994, quando Rodger se mudou para lá. Apartir daí vários alunos trabalharam no JFLAP e/ou ferramentas relacionadas ao longo dos anos desde quando começaram a trabalhar na ferramenta

Atualmente o JFLAP esta na versão JFLAP 8.0 BETA que já está disponível para download em *http://www.jflap.org/*, logo após o preenchimento de um formulário. Sendo que a fabricante do software, a Duke University,

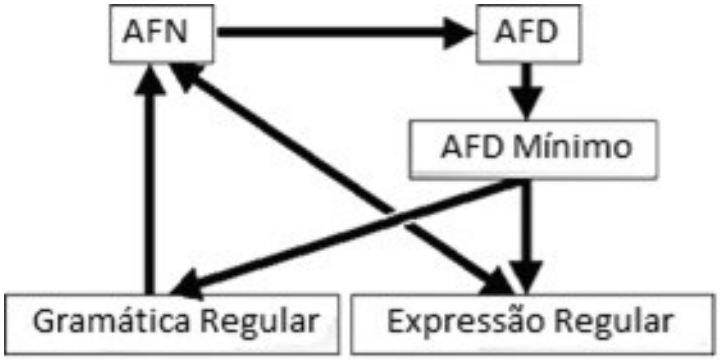
recomenda o uso da Versão 7 para utilização em cursos, por ser a versão mais estável até o momento. Vale destacar que em dezembro de 2007 a versão 6.1 do JFLAP foi candidata finalista no Premier Award (*https://premieraward.org/*) por Excelência em Educação em Engenharia.

O JFLAP conta com um tutorial no site *http://www.jflap.org/tutorial/* oferecendo um manual básico sobre muitos conceitos do JFLAP para ajudar a começar em um primeiro contato com a ferramenta. Além disso, para ajudar nesse tutorial é disponibilizado para download todos os arquivos utilizados no manual de forma a facilitar o aprendizado da forma mais rápida e eficiente possível ao usuário.

O estágio inicial para o uso do JFLAP é a criação de autômatos. O usuário utiliza a interface visual para criar um grafo representando os estados, o diagrama de transições e os seus labels. Em seguida, pode definir a palavra de entrada e então visualiza a execução de cada passo do autômato, verificando se o seu projeto é coerente. Isso pode ser feito com três escolhas de execução, um modo direto, onde a resposta é imediata, um modo passo a passo, que mostra os estados percorridos, e o modo múltiplo, que mostra o teste de diversas palavras, da mesma maneira que o modo direto.

JFLAP se destaca por ser um software para aprendizado e prática de tópicos de linguagens formais, incluindo autômatos finitos não determinísticos, autômatos de pilha não determinísticos, máquinas de Turing, vários tipos de gramáticas e análises. Além de construir e testar exemplos para estes, o JFLAP permite experimentos com provas de construção de uma forma para outra, como converter um AFN para um AFD, um AFD para um AFD mínimo, um AFD para uma expressão regular, um AFN para uma gramática regular e uma gramática regular para um AFN. A Figura 14 mostra as possibilidades de conversão entre estruturas nas linguagens regulares.

Figura 14 - Modos de conversão para as representações das linguagens regulares.



Fonte: Adaptado de JFLAP (2017).

O JFLAP pode ser utilizado em três formas:

- **Experimentação das estruturas das linguagens:** simular palavras de entrada em autômatos. Os autômatos principais são o autômato finito deterministico (AFD) e não deterministico (AFN), autômatos a pilha (PDA) e maquinas de Turing multi-tape.
- **Conversão entre estruturas equivalentes:** ele tem capacidade de converter expressões regulares para AFD, AFN para AFD, PDA para gramática, gramática para PDA, minimização de AFD, gramática livre de contexto para a forma normal de Chomsky.
- **Ferramentas de análise:** O JFLAP oferece algumas ferramentas para mostrar as características das estruturas de linguagens, como por exemplo equivalência entre dois autômatos finitos.

3.1.1.1 Métodos de uso

O modo inicial de trabalho do JFLAP pode ser escolhido através do menu de entrada de acordo com a figura 15.

Figura 15 - Menu principal.

Fonte: JFLAP (2017).

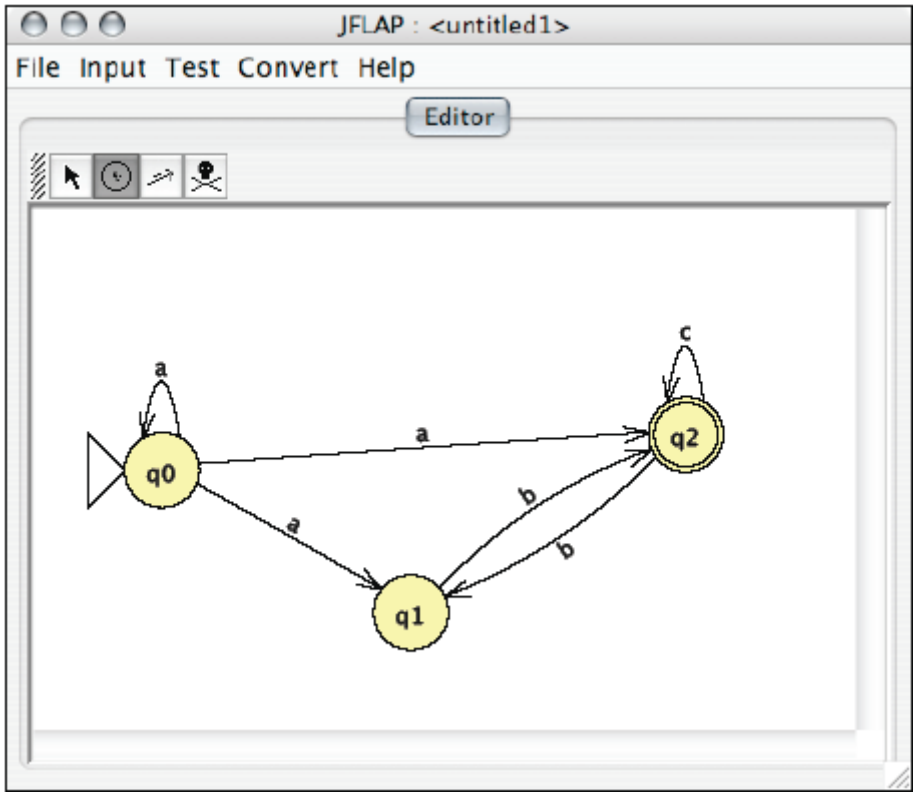
A escolha do modo de trabalho abre uma tela especifica para cada tipo de autômato ou estrutura de linguagem.

Autômatos Finitos

Pode-se realizar varias operações com Autômatos finitos no JFLAP como verificar se palavras são reconhecidas pelo autômato, comparar equivalência entre autômatos que sejam do mesmo tipo, além de conversões do autômato para AFD, minimizar, converter em gramatica, converter para expressão regular e combinar o autômato com outro.

Na tela de autômatos finitos é feita toda a manipulação do Autômato: criação de estados, transições, identificação de estados iniciais e finais. Sem a realização dessas definições o JFLAP acusara a impossibilidade de testar o funcionamento do autômato. A figura 16 mostra a tela de manipulação de autômatos, com a criação de três estados (q_0, q_1 e q_3), três transições (a.b e c), com o estado inicial definido em q_0 e estado final definido em q_2.

Figura 16 - Tela de manipulação de Autômato.



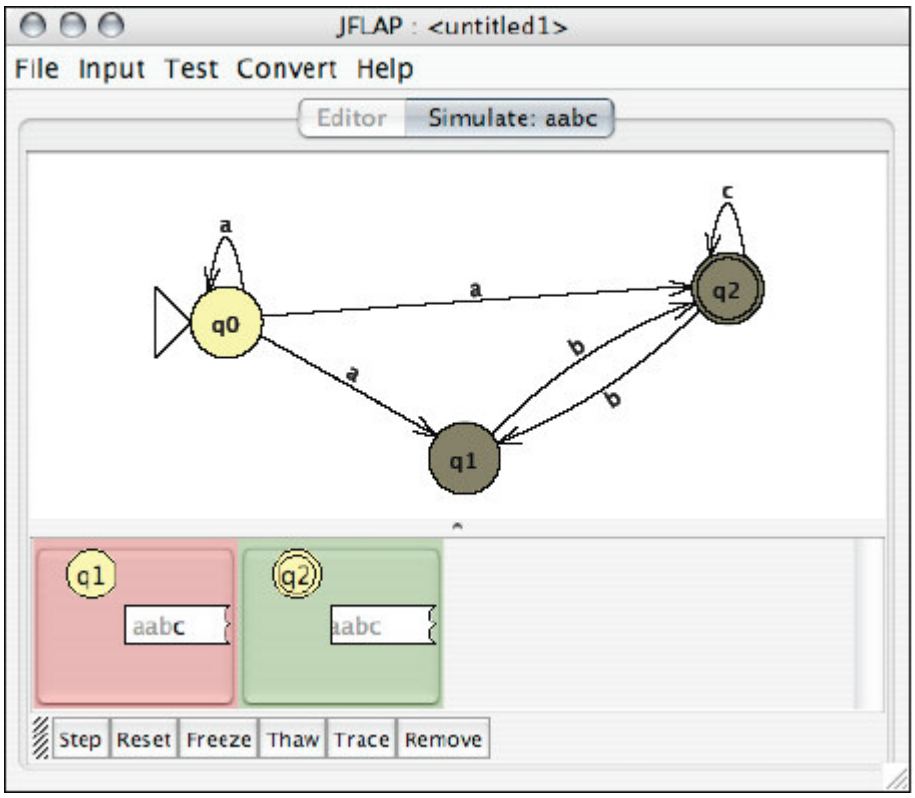
Fonte: JFLAP (2017).

Depois de criar o autômato, é possível a realização do procedimento de verificação. O JFLAP verifica se uma palavra de entrada é a aceita ou não pelo autômato.

Enquanto são realizadas as simulações e teste a interface apresenta o passo a passo do tratamento da palavra, dando destaque aos estados em uso com a cor mais escura.

A verificação das transições é realizada até o instante que é possível ocorrer a simulação de transições. Nesse momento, a transição em particular é considerada invalida e as posteriores continuam. Nos casos aonde a transição chega ao estado final, o simulador informa que o autômato aceita a palavra. A recusa é representada pela cor vermelha e a aceitação da palavra é representada pela cor verde. A figura 17 mostra o processo de verificação de uma entrada, verificando se a mesma pertence ou não ao autômato, sendo este o mesmo da figura 16. Nela a entrada que é verificada é “aabc” e ela é aceita pelo autômato.

Figura 17 - Tela de verificação de Autômato.

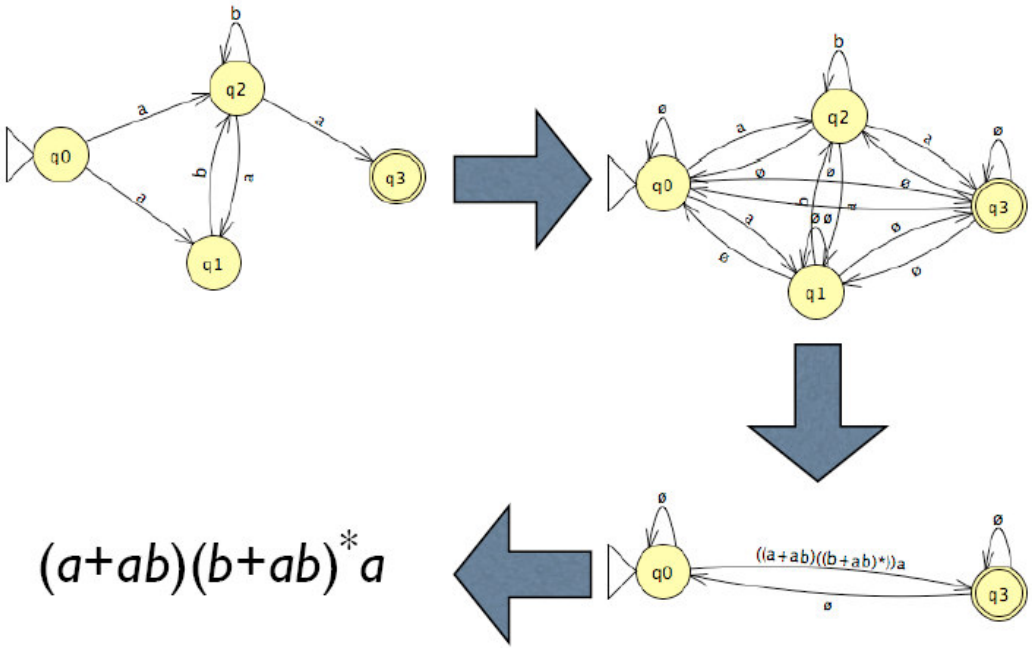


Fonte: JFLAP (2017).

Conversão para expressões regulares

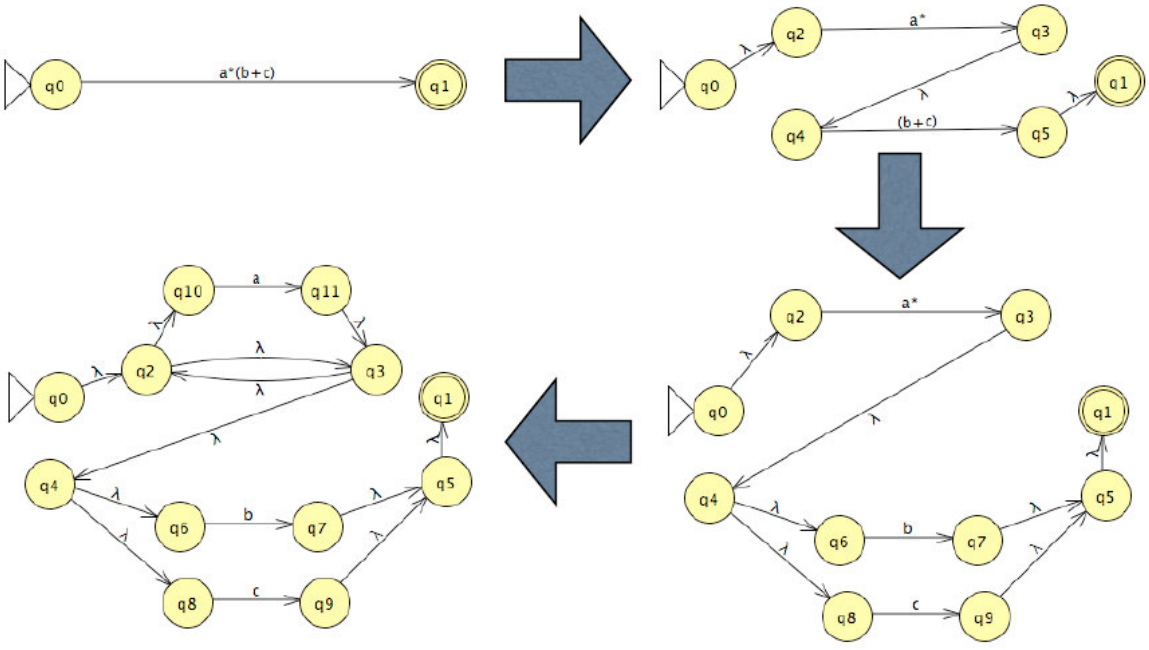
Outro ponto importante é a conversão do autômato finito para a respectiva expressão regular. O processo pode ser feito de maneira direta, com o resultado mostrado rapidamente, ou interativamente, com o objetivo de se apresentar o algoritmo passo a passo. Simplificando, a partir do AFD inicial, o JFLAP mostra a obrigação da inclusão de transições lambda extras (transição para um novo estado sem ler nenhum símbolo de entrada), para em seguida habilitar o usuário a escolher quais estados devem ser excluídos e substituídos por trechos de expressões regulares até gerar a expressão regular final. A figura 18 mostra a conversão de AFD para ER e a figura 19 mostra o processo inverso, conversão de ER para AFN.

Figura 18 - Conversão de AFD para ER.



Fonte: JFLAP (2017).

Figura 19 – Conversão de ER para AFN.



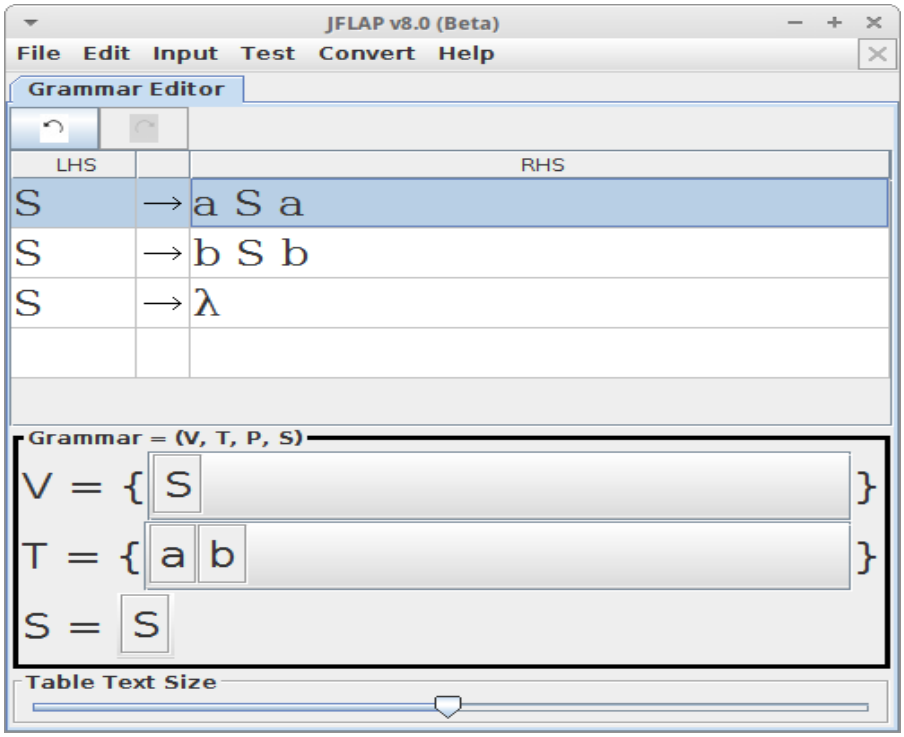
Fonte: JFLAP (2017).

No movimento contrário, o JFLAP realiza a conversão, de maneira automática ou interativa, da ER para um AFN. Podendo este em seguida ser convertido para um AFD e ainda ser minimizado.

Gramáticas

O modo de criação de uma gramática no JFLAP abrange todas as gramáticas previstas na seção 2.1.2. A entrada dos dados consiste em colocar as variáveis na coluna da esquerda e na direita, as variáveis e os terminais como representada na figura 20.

Figura 20 – Entrada de dados de uma gramática.

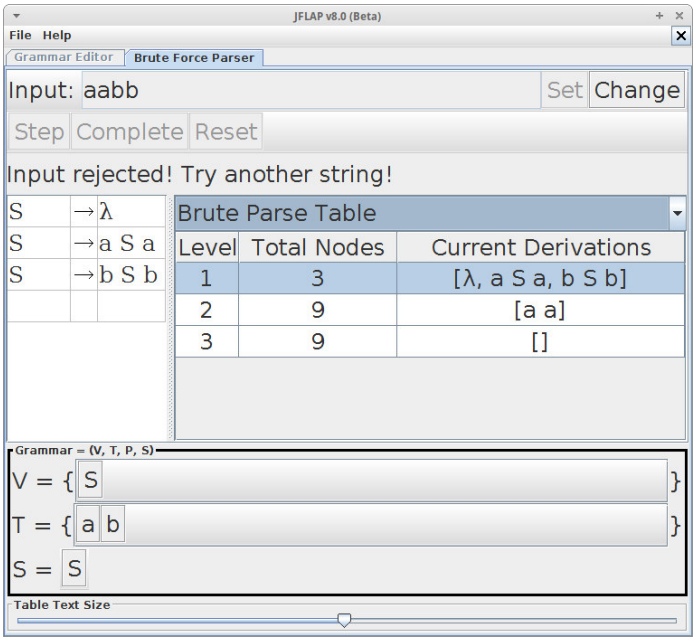


Fonte: JFLAP (2017).

Dada a gramatica e incluindo as suas regras de produção no JFLAP. Pode-se verificar se sentenças são válidas. Desta maneira podem-se realizar vários testes e verificar a validade de sentenças, nos dois casos o softwares informa o usuário atrás de uma mensagem se esta é válida ou invalida. As figuras 21 e 22 mostram duas verificações na gramática da figura 20 na qual a figura 21 apresenta a sentença de entrada “aabb” sendo

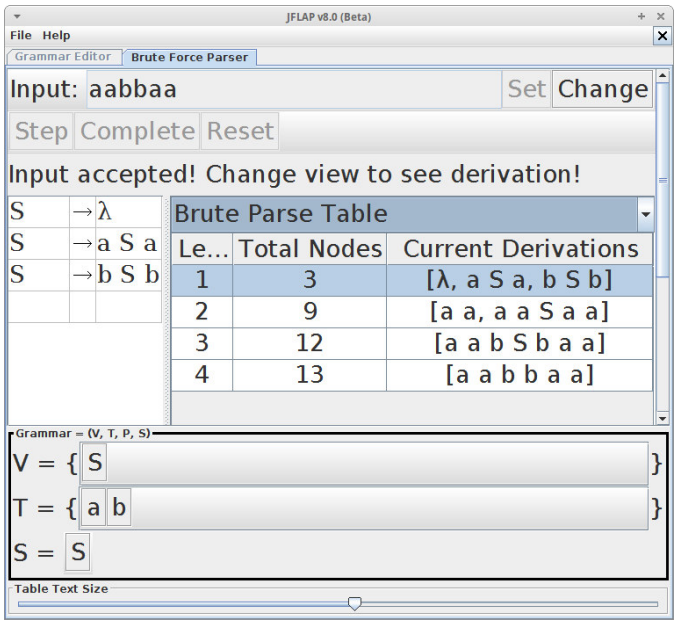
rejeitada e a figura 22 apresenta a sentença de entrada “aabbaa” sendo aceita pela gramática.

Figura 21 - Caso no qual a sentença não é valida para a gramatica



Fonte: JFLAP (2017).

Figura 22 - Caso onde a sentença é valida para a gramatica.

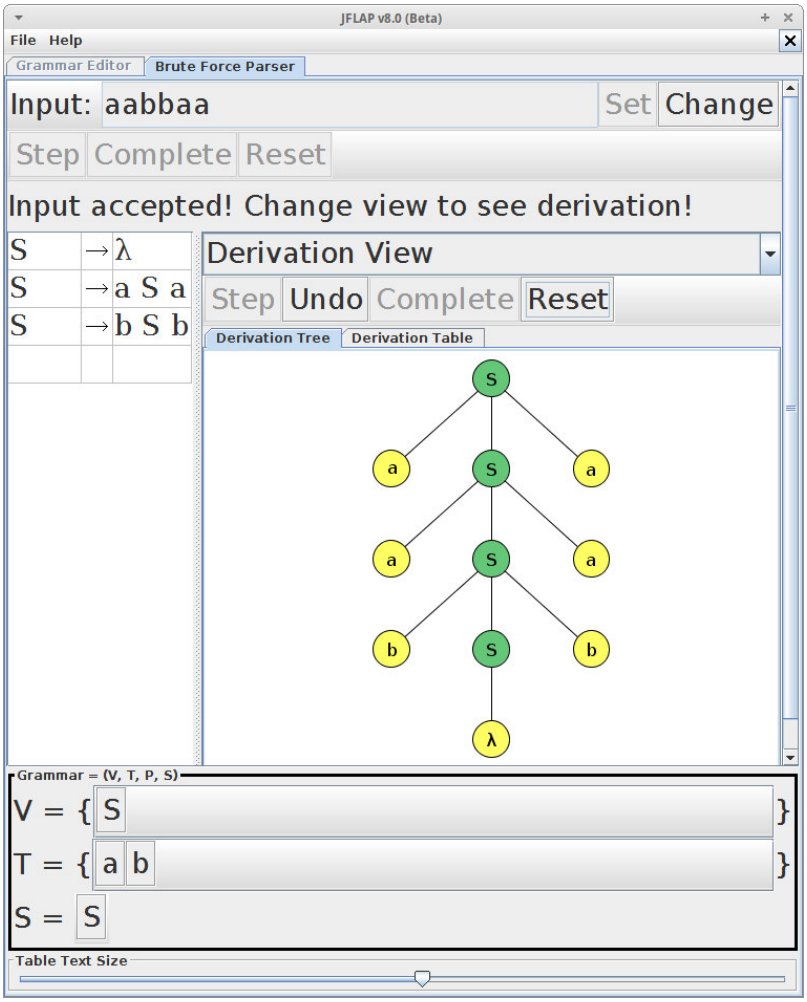


Fonte: JFLAP (2017).

Um recurso interessante e que vale ressaltar no JFLAP é o *Derivation View* (Ver derivação), que permite visualização a árvore de

derivação de determinada sentença. A derivação da sentença de entrada “aabbaa” aceita pela gramática representada na figura 22 pode ser vista na figura 23.

Figura 23 - Recurso Derivation View (Ver derivação).



Fonte: JFLAP (2017).

Através do recurso *Derivation View* é possível acompanhar todo o resultado da derivação de uma gramática, podendo acompanhar cada etapa, desfazer, redefinir ou finalizar a derivação.

3.1.2 Auger – Ambiente para construção e simulação de autômatos finitos

Auger é um ambiente para construção e simulação de autômatos finitos, ele iniciou como projeto de conclusão do curso de graduação em Ciência da Computação do Sr. Charbel Szymanski. Após a apresentação do projeto o Auger passou a ser utilizado como ferramenta de apoio ao ensino na

disciplina de Linguagens Formais e Autômatos do Curso de Ciência da Computação da Universidade do Sul de Santa Catarina, estando disponível para download em várias versões, sendo a versão mais atual a 3.1.

O objetivo principal do Auger é o apoio didático no ensino de linguagens formais, mais especificamente no ensino de autômatos finitos.

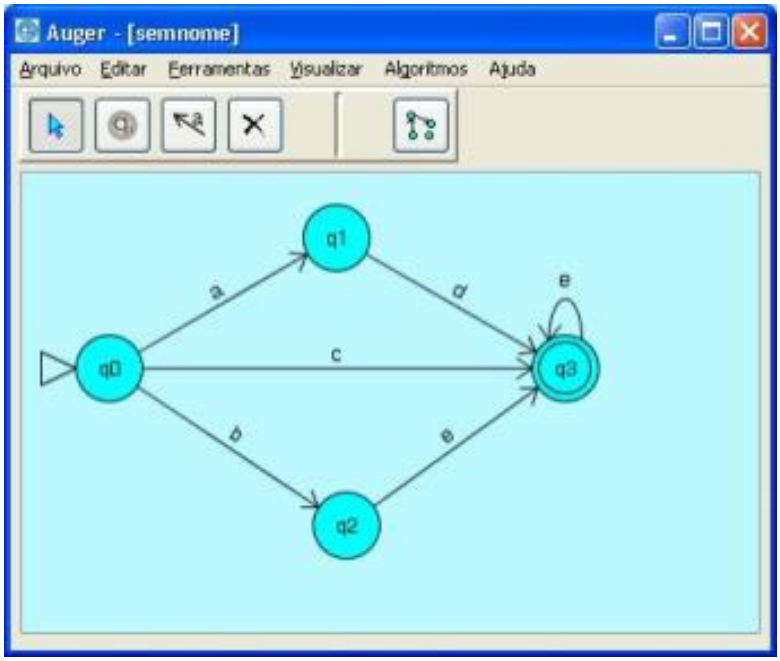
Através do Auger é possível:

- Criar autômatos finitos de forma gráfica (a partir do diagrama de estados);
- Executar algoritmos de manipulação de autômatos finitos;
- Testar autômatos finitos através do recurso de simulação de cadeias de entrada;
- Utilizar expressões regulares e gramáticas regulares para gerar autômatos finitos;
- Gerar expressões regulares e gramáticas regulares a partir de autômatos finitos;
- Gerar programas em linguagem Java para testar os autômatos criados ou criar analisadores léxicos;
- Criar diagramas de estados e utilizá-los como imagens em outros softwares.

3.1.2.1 Métodos de Uso

Basicamente o Auger possui 4 módulos de uso bem definidos: manipulação do autômato, Importações, Exportações e Algoritmos. Além disso, o Auger conta também com um manual presente no site *http://www.evoluma.com/auger/* anteriormente citado. O manejo do autômato é realizado através do diagrama de estados ou através da aplicação dos algoritmos específicos, a figura 24 ilustra a tela de manipulação de autômatos com a criação de quatro estados (q_0, q_1, q_2, q_3) e cinco transições (a, b, c, d, e).

Figura 24 - Tela de manipulação de autômatos.



Fonte: Auger (2017).

Para manipular diretamente o autômato o usuário deve utilizar as ferramentas presentes na barra superior e ilustrada na figura 25.

Figura 25 - Barra de ferramentas para manipulação de autômatos.

Fonte: Auger (2017).

Através da barra de ferramentas toda a manipulação do autômato é feita, traves dela e possível:

- Selecionar objetos (um ou vários estados).
- Criar estados do autômato.
- Marcar um estado como inicial ou final do autômato.
- Renomear um estado.
- Criar transições.
- .Exclusão de objetos.
- Posicionar estados aleatoriamente, posicionando estados para uma posição qualquer.

Através da função "Importar" o usuário poderá transformar uma expressão regular (ER) ou uma gramática regular (GR) em um autômato.

Para importar uma expressão regular o usuário deverá digitar ou colar uma expressão na caixa de texto. Na figura 26 mostra a caixa de entrada para uma ER (abc*) ser importada em um autômato.

Figura 26 – Importar uma expressão regular.

Fonte: Auger (2017).

Para importar uma gramática regular o usuário deverá abrir um arquivo salvo. O arquivo deve ser um arquivo de texto que esteja em notação BNF. A notação BNF(*Backus Naur Form* ou *Backus Normal Form*) é uma metassintaxe usada para expressar gramáticas livres de contexto, isto é, um modo formal de descrever linguagens formais. Ela é usada como uma notação para as gramáticas de linguagens de programação, conjuntos de instruções e protocolos de comunicação, e também como notação para representar partes de gramáticas de linguagens naturais. Uma especificação BNF é um conjunto de regras de derivação, escritas como:

<símbolo> ::= <expressão com símbolos>

Onde <símbolo> é um não terminal, e a expressão consiste em sequências de símbolos e/ou sequências separadas pela barra vertical, '|', indicando uma escolha. Esta notação indica as possibilidades de substituição para símbolo da esquerda. Símbolos que nunca aparecem no lado esquerdo são ditos terminais. A figura 27 apresenta um exemplo de gramática regular aceita pelo Auger.

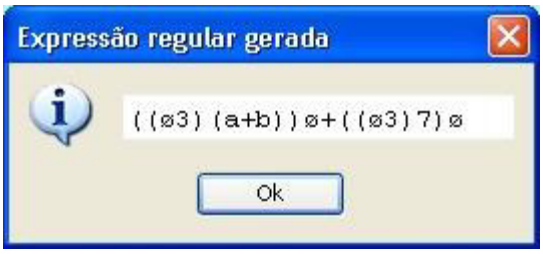
Figura 27 - Arquivo de texto em notação BNF aceito pelo Auger.

```
<q0> ::= 3<q1>
<q0> ::= 6<q3>
<q1> ::= w<q2>
<q2> ::= ø
<q3> ::= 2<q2>
```

Fonte: Auger (2017).

A função "Exportar" oferece ao usuário um mecanismo para criação de uma expressão regular ou uma gramática regular a partir de um autômato. Tanto a gramática quanto a expressão regular podem ser salvas em arquivo após a exportação. A figura 28 mostra um exemplo de uma ER criada a partir de um autômato e a figura 29 uma gramatica regular em notação BNF criada a partir de um autômato.

Figura 28 - Expressão regular exportada.



Fonte: Auger (2017).

Figura 29 - Gramática regular exportada.

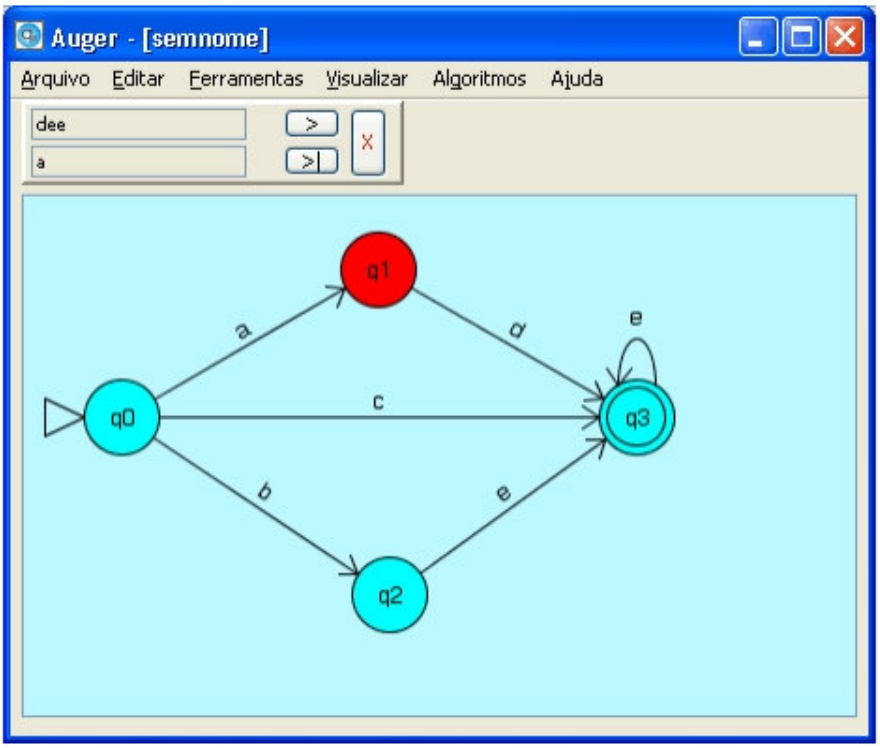
Fonte: Auger (2017).

Na função Algoritmo se encontra a execução dos algoritmos do Auger, como:

- AFND => AFD: transformação para autômato finito determinístico

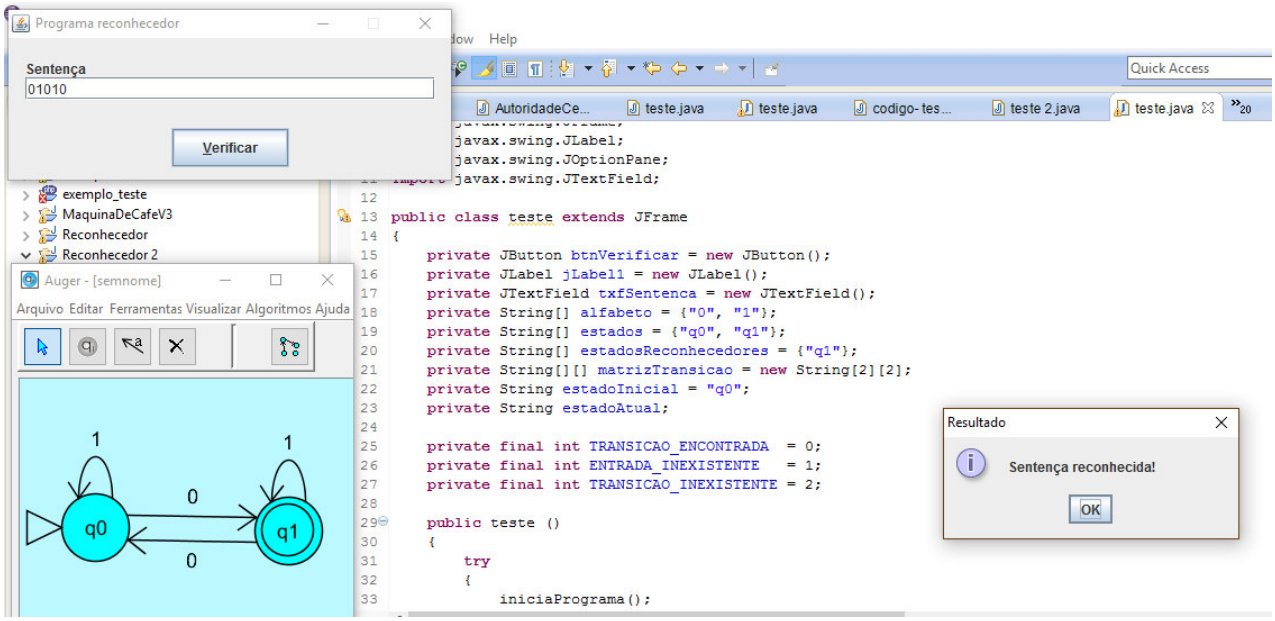
- Excluir e-transições: retirar as transições vazias presentes.
- Excluir estados inativos
- Excluir estados desnecessários ou inacessíveis
- Minimizar o autômato: retirando estados que se equivalentes, inativos e inacessíveis.
- Gerar programa reconhecedor**:** gera um programa em Java que permite receber sentenças de entrada e verificar se pertencem à linguagem reconhecida pelo autômato que está sendo manipulado, esse reconhecimento pode ser gráfico ou através da geração de código fonte em linguagem Java, sendo o autômato utilizado para gerar o programa reconhecedor precisa ser determinístico. (figura 30 e figura 31).
- Teste do Autômato: Verifica se uma sentença dada como entrada pertence à linguagem aceita pelo autômato.

Figura 30 – Reconhecedor do Auger (Forma gráfica): verificando se a sentença pertence à linguagem aceita pelo autômato da figura 24.



Fonte: Auger (2017).

Figura 31 – Código Java gerado pelo Auger a partir do AFD da figura.



Fonte: Autor.

Na figura 31 podemos pode-se visualizar o código fonte linguagem Java, onde podemos perceber a criação dos estados (q_0 e q_1) e transições (0 e 1), estado inicial e final (estado reconhecedor) e criação da matriz de transição.

3.1.3 Simulador de Autômatos

O Simulador de Autômatos é um software para criação, simulação e conversão de modelos formais, foi criado com o proposito de contribuir no processo de aprendizagem de Linguagens Formais e Autômatos.

O simulador de Autômatos se originou de um projeto de Iniciação Cientifica da disciplina de Teoria da Computação tendo posteriormente sua continuação no Trabalho de conclusão de Curso (em Engenharia da Computação) na Universidade de Uberaba – UNIUBE. O autor do projeto criador do Simulador de Autômato foi, o na época aluno, Neilton Gonçalves Ribeiro Junior, e o download pode ser feito no site: *www.simuladordeautomatos.com/* gratuitamente.

O software possui um vídeo tutorial que demonstra como realizar algumas funções do simulador como desenhar diagramas, realizar simulações e conversões e inserir as produções de uma gramática.

O Simulador de Autômatos reconhece e simula:

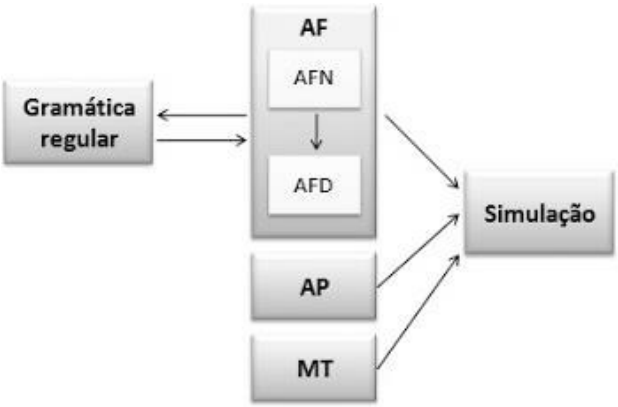
- Autômatos Finitos Determinísticos
- Autômatos Finitos Não-Determinísticos
- Autômatos com pilha
- Máquina de Turing
- Gramáticas Regulares

Além disso, realiza conversões entre formalismos como:

- AFD -> AFN;
- AF -> GR;
- GR -> AF.

As funcionalidades citadas são mostradas na forma gráfica na figura 32.

Figura 32 – Funcionalidades do Simulador de Autômatos.

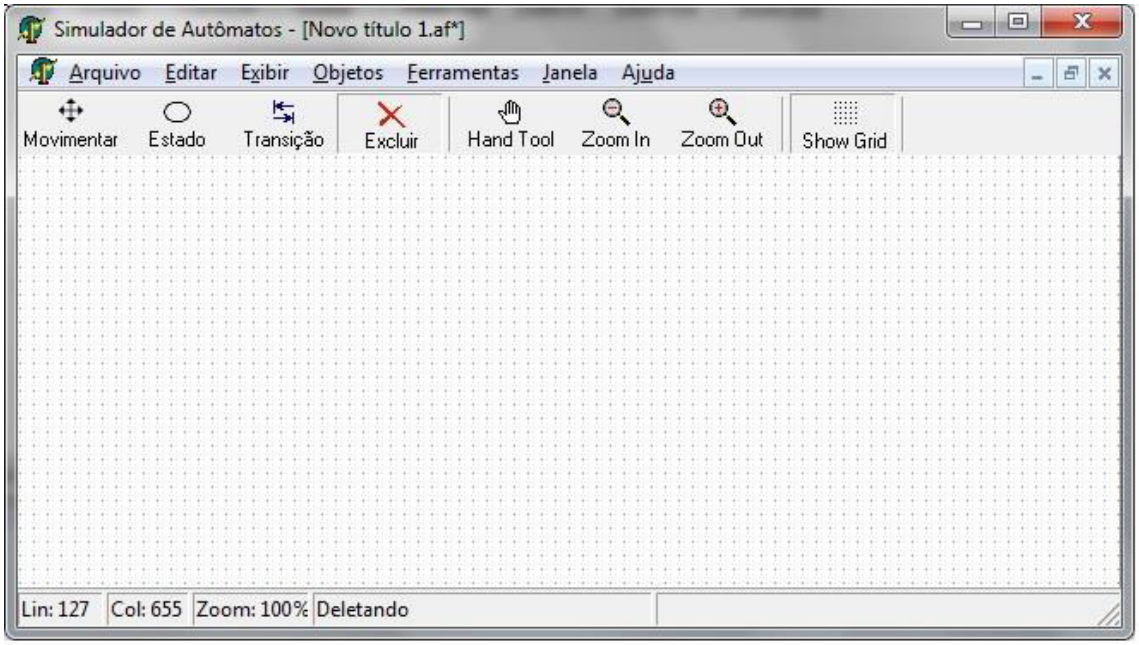


Fonte: Simulador de Autômatos (2017).

3.1.3.1 Métodos de Uso

O Simulador de Autômatos possui uma interface simples e a manipulação do autômato é feita através de diagrama de estados. Para manipular diretamente o autômato o usuário deve utilizar a barra de ferramentas. A tela de manipulação e mostrada na figura 33.

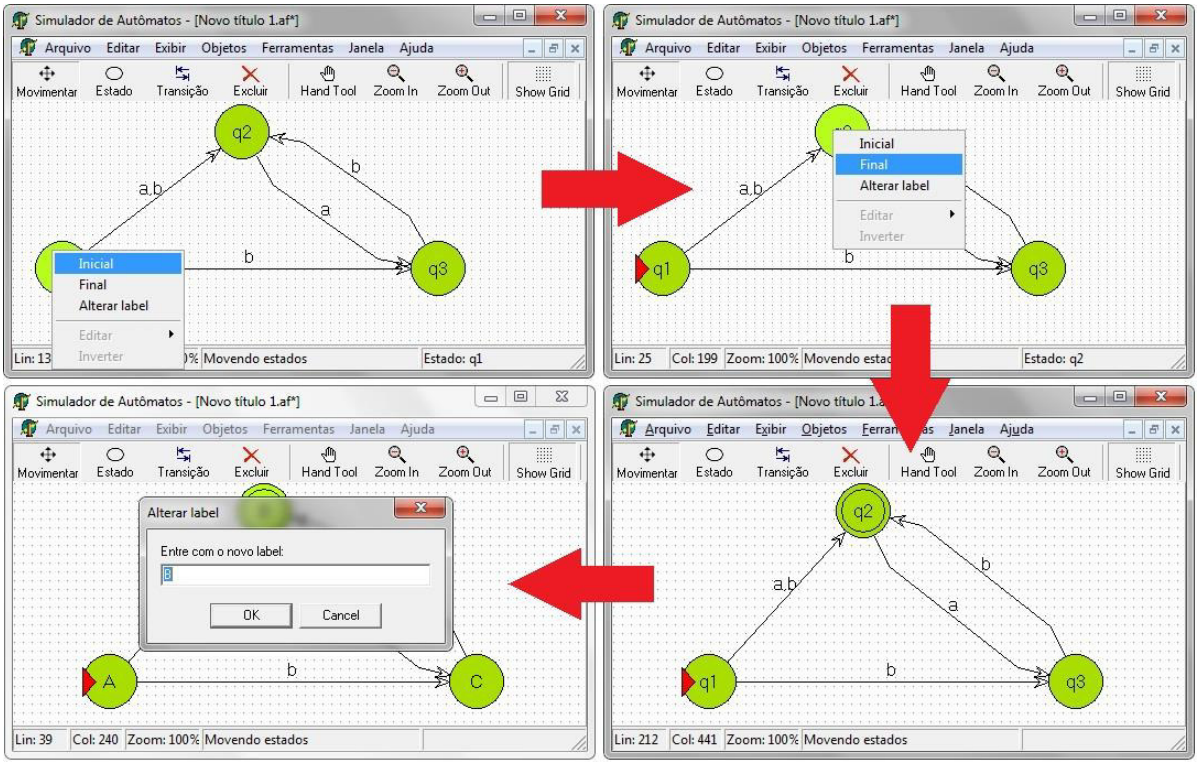
Figura 33 - Tela principal do Simulador de Autômatos



Fonte: Simulador de Autômatos (2017).

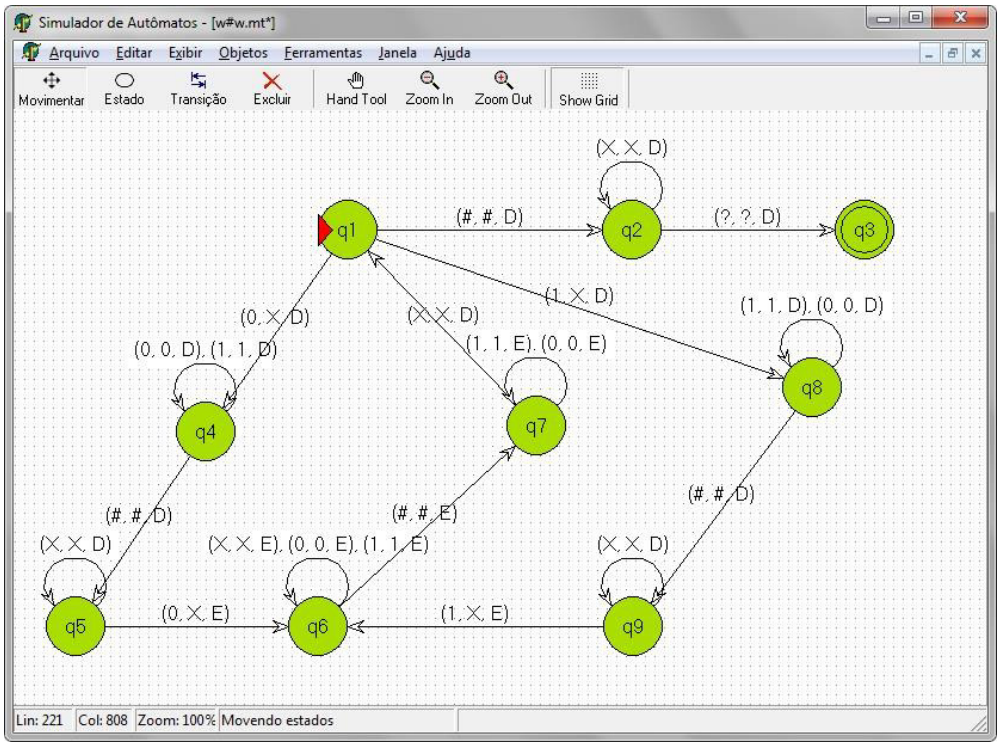
Através das ferramentas presentes na tela inicial é possível toda a manipulação dos autômatos, como os desenhos de um autômato, definição de estados iniciais e finais, criação de diagramas de estados de um Autômato Finito como mostrado na figura 34, onde ocorre a criação de três estados (q_1,q_2,q_3), criação das transições e definição do estado q_1 como inicial e q_3 como final, além da criação de diagramas de um autômato com Pilha e criação de diagramas de uma Maquina de Turing (Figura 35) .

Figura 34 – Manipulação de Autômato no Simulador de Autômatos.



Fonte: Simulador de Autômatos (2017).

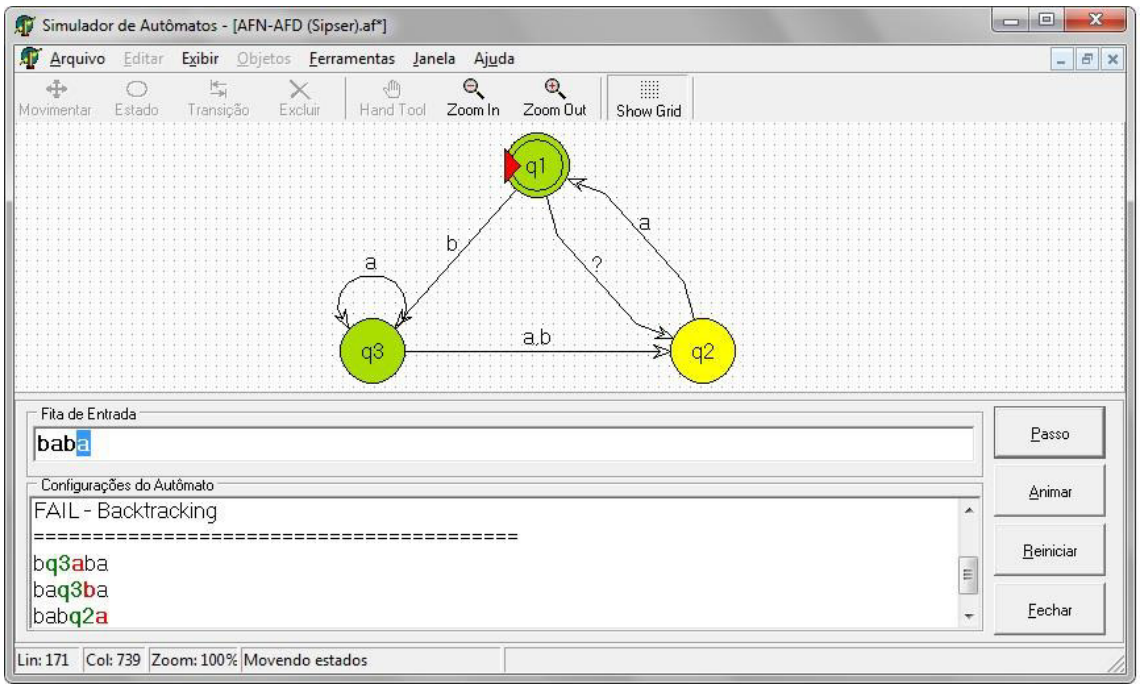
Figura 35 - Diagrama de uma Máquina de Turing.



Fonte: Simulador de Autômatos (2017).

O software conta com uma função de simulação onde é possível verificar se uma sentença de entrada pertence à linguagem aceita pelo autômato. Essa etapa pode ser realizada de forma automática, como o software realizando os testes passo a passo de forma rápida ou de forma manual, aonde o usuário vai passando cada passo da simulação da forma que desejar. A figura 36 apresenta um AFD realizando a função de verificação. Na parte central é possível entrar com a sentença que se deseja verificar se pertence à linguagem aceita pelo autômato, na parte inferior é mostrado o algoritmo que é realizado durante a verificação e na parte superior o mesmo algoritmo é mostrado, mas de forma gráfica através do próprio autômato desenhado para uma melhor visualização e entendimento do usuário.

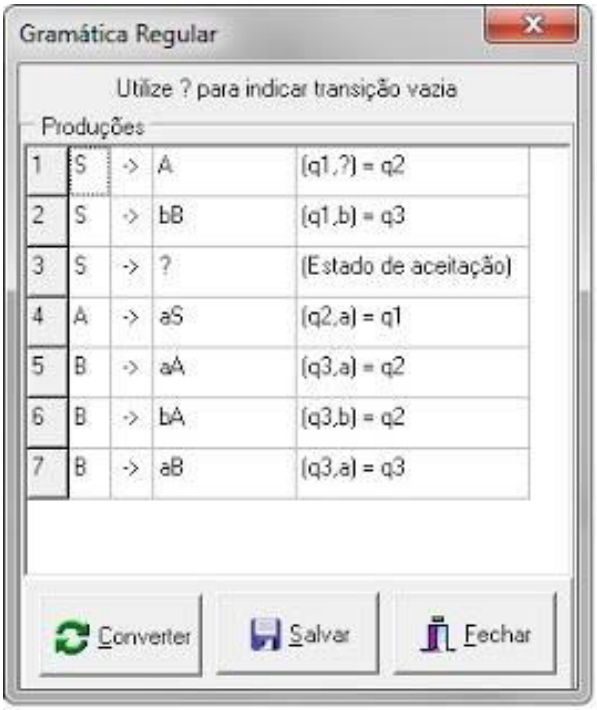
Figura 36 - Tela de simulação da ferramenta.



Fonte: Simulador de Autômatos (2017).

Além disso, é oferecido ao usuário um mecanismo para conversão de Autômato Finito em uma Gramática Regular. Na figura 37 é mostrado um exemplo dessa conversão em GR, onde o autômato criado na figura 34 é convertido em uma GR.

Figura 37 - Conversão de Autômato Finito em uma Gramática Regular.



Fonte: Simulador de Autômatos (2017).

4 Discussão sobre os softwares de LFA

Com o avanço da tecnologia e a sua presença em vários aspectos de nossas vidas, a criação de softwares tomou uma posição de destaque no mercado. Hoje já é possível se deparar como uma grande variedade de softwares educacionais disponíveis, mas nem todos alcançam seu principal objetivo, dar suporte de forma eficaz ao ensino/aprendizagem, ou nem tem fins pedagógicos.

Diante disso, neste trabalho, verificou-se a necessidade de debater sobre softwares de ensino em Linguagens Formais presentes no mercado, discutindo sobre cada um dos softwares educativos de LFA citados: JFLAP (2017), Auger (2017) e Simulador de Autômatos (2017), ressaltando seus aspectos positivos e limitações, verificando que há a necessidade de realizar ponderações sobre estes programas antes de utiliza-los com os alunos.

Segundo GOUVÊA e NAKAMOTO (2015) A maioria das discurssões encontradas sobre softwares educativos é focada em avaliar apenas o desempenho do programa em si, desconsiderando a parte pedagógica do software. Este fato dificulta aos professores e alunos na hora de escolher um Software Educacional, por expor bastantes termos técnicos e não levar em consideração suas dúvidas em relação às atividades, como *feedback* ao aluno, sequencia didática, grau de dificuldade, interação aluno-software, entre outros fatores pedagógicos.

Vale ressaltar que todo Software Educacional deve ser desenvolvido por uma equipe multidisciplinar, com a presença de profissionais da educação, como pedagogos e professores e avaliados por uma comissão pedagógica.

A primeira análise que se faz ao escolher um Software Educacional é se o software está condizente com as diretrizes do projeto pedagógico e se todas as pessoas envolvidas no processo educativo devem estar sensibilizadas ao uso da informática no ambiente escolar. O uso do computador como ferramenta pedagógica requer quebras de antigos paradigmas ainda presentes, como sistema de avaliação, práticas pedagógicas ultrapassadas, concepção de ensino/ aprendizagem.

Para Rocha (2001) é necessário cautela com os Softwares Educacionais em relação aos seguintes aspectos:

- Características pedagógicas: atributos que evidenciam a conveniência e a viabilidade de uso do software em situações educacionais
- Facilidade de uso: atributos que evidenciam a facilidade de uso do software.
- Características da interface: atributos que evidenciam a presença de recursos e meios que facilitam a interação do usuário com o software.
- Adaptabilidade: atributos que evidenciam a capacidade do software adaptar-se às necessidades e preferências do usuário e ao ambiente educacional.
- Documentação: atributos que evidenciam que a documentação para instalação e utilização do software está completa, consistente, legível, e organizada.
- Portabilidade: atributos que evidenciam a adequação do software aos equipamentos onde serão instalados.
- Retorno do investimento: atributos que evidenciam a adequação do investimento na aquisição do software.

Já segundo Oliveira (2001), as características importantes de um Software Educacional são as seguintes:

- Definição e presença de uma fundamentação pedagógica que permeie todo o seu desenvolvimento;
- Finalidade didática, por levar o aluno/usuário a "construir" conhecimento relacionado com seu currículo escolar;
- Interação entre aluno/usuário e programa, mediada pelo professor;
- Facilidade de uso, uma vez que não se devem exigir do aluno conhecimentos computacionais prévios, mas permitir que qualquer usuário, mesmo que em primeiro contato com a máquina, seja capaz de desenvolver suas atividades;

Sendo assim, segue uma discussão sobre os softwares presentes nesse trabalho, utilizando algumas ponderações e levando em consideração,

vários critérios existentes, critérios esses que se apresentam de forma relevante por Rocha (2001) e Oliveira (2001) anteriormente e privilegiam o processo de ensino e aprendizagem. Tais critérios são: características pedagógicas, facilidade de uso, características da interface, adaptabilidade, documentação, portabilidade, retorno do investimento e finalidade didática.

4.1 Características pedagógicas:

Características pedagógicas são atributos que evidenciam a conveniência e a viabilidade de uso do software em situações educacionais. (ROCHA, 2001)

Em relação as características pedagógicas, todos os softwares citados (JFLAP, Auger e Simulador de Autômatos) apresentam atributos capazes de proporcionar o seu uso no ambiente educacional (clareza, simplicidade e associação da ideia/imagem permitindo aprender com mais facilidade). Os softwares se apresentaram bastante úteis e proveitosos no processo de ensino-aprendizagem de LFA, através da realização de teste pelo autor. Contudo vale ressaltar que estes softwares, assim como todos os softwares educativos, não devem ser considerados como algo independente do acompanhamento do professor e/ou tutores.

4.2 Facilidades de uso

Facilidade de uso são atributos que evidenciam a facilidade de uso do software. (ROCHA, 2001)

A facilidade de uso é evidente em todos os softwares, bem como o seu caráter aplicativo, possibilitando ao usuário realizar de forma prática e rápida várias tarefa relacionada a LFA. Um aspecto negativo em um dos softwares fica por conta do JFLAP, por só apresentar o idioma inglês, causando certa dificuldade aos alunos que não dominam o idioma. Além deste, outro aspecto negativo que deixa a desejar fica por conta do Auger por possuir a função exportar (gerar uma expressão regular ou uma gramática regular a partir de um autômato) e Importar (transformar uma expressão regular ou uma gramática regular em um autômato) onde o usuário para realizar tais operações deve ter certo conhecimento prévio sobre a notação BNF, pois ao realizar tais funções o resultado é exibido nesta notação, o que a princípio dificulta para um

usuário leigo no assunto, em um primeiro contato. O ponto negativo do Simulador de Autômatos está presente na modelagem dos autômatos que em alguns momentos se apresenta inflexível na manipulação dos autômatos e as vezes um pouco confusa em relação a criação de estados e transições, poderia ser um pouco mais intuitiva, já que a maioria das ação é feita através de botões, além de não possuir uma comando de retorno de ação, como no JFLAP e Auger, para correção de possíveis erros cometidos. Os aspectos motivadores estão presentes em todos os softwares (JFLAP, Auger e Simulador de Autômatos) em vários pontos por meio de desenhos, ilustrações, gráficos, variedade de cores, colocando o aluno frente a desafios e a oportunidades de identificar seus erros na realização de atividades.

4.3 Características da interface

Características de interface são atributos que evidenciam a presença de recursos e meios que facilitam a interação do usuário com o software. (ROCHA, 2001)

Um bom software educativo deve possuir meios que facilitem a interação do usuário com o software, dessa forma ele deve aliar conceitos ergonômicos e de design coerentes com a perspectiva pedagógica adotada. Seguindo esse conceito podemos afirmar que os softwares discutidos neste trabalho são softwares que apresentam uma boa interface, possibilitando através do design das mesmas mediar uma comunicação com os alunos, por estar intimamente relacionada a abordagens pedagógicas e aos conceitos presentes em LFA. Todos os softwares auxiliam os alunos, facilitando a assimilação de conteúdos e tornando o aprendizado mais interessante e possuem recursos gráficos devidamente justificados e contextualizados com as atividades de aprendizagem da disciplina LFA.

4.4 Adaptabilidade

Adaptabilidade são atributos que evidenciam a capacidade do software adaptar-se às necessidades e preferências do usuário e ao ambiente educacional selecionado. (ROCHA, 2001)

Desta forma a adaptabilidade serve para melhorar o nível de usabilidade de um software, sendo a flexibilidade um critério importante na adaptabilidade.

A flexibilidade é um recurso que permite ao usuário personalizar a interface de acordo com a variação do contexto e estratégia de trabalho para a realização de uma tarefa. Pode-se qualificar a flexibilidade em dois tipos: estrutural e personalização. A Flexibilidade Estrutural caracteriza-se por oferecer maneiras distintas de realizar uma tarefa, customizando a sequência de entrada de dados conforme a necessidade, e nesse critério todos os softwares: JFLAP, Auger e Simulador de Autômatos se destacam positivamente apresentando ferramentas bastante didáticas, mostrando o passo a passo dos testes realizados nos autômatos dada uma entrada, podendo ser realizada de acordo com a vontade do usuário para o seu melhor entendimento. A Flexibilidade de Personalização já é uma adaptabilidade para usuários mais experientes e possibilita ao usuário incluir ou retirar dados ou comandos da tela de acordo coam a exigência da tarefa, estratégia ou hábito de trabalho, além disso, permite alterar valores padrões do sistema conforme a exigência ou necessidade. Nesse critério de personalização o Simulador de Autômatos e o Auger apresentam possibilidades restritas ou quase nulas, com o Simulador de Autômatos possibilitando apenas a mudança da fonte das letras utilizadas para nomear os estados e transições de autômatos nos diagramas e o Auger apenas na mudança da cor de fundo do ambiente de modelagem do autômato. Já o JFLAP diferente dos demais apresenta várias possibilidades de personalização, merecendo destaque neste quesito. O JFLAP possibilita definir o caractere de cadeia vazia possibilitando o usuário escolher o símbolo de cadeia vazia entre lambda ou epsilon de acordo com o que desejar. O JFLAP também possibilita definir o número de ações que o usuário é capaz de desfazer quando estiver trabalhando nos editores de diagramas de estados, permite que o usuário especifique se as máquinas de Turing aceitarão uma entrada quando estiverem em um estado final ou aceitaram quando pararem, ambas ou nenhuma delas, sendo que por padrão se nenhuma for selecionada ele sempre rejeitará, além disso, é possível restringir os movimentos da cabeça da fita para a esquerda ou direita na Maquina de Turing.

4.5 Documentação

Documentação são atributos que evidenciam que a documentação para instalação e utilização do software está completa, é consistente, legível, e organizada. (ROCHA, 2001)

A documentação é essencial para qualquer software desenvolvido, desde o projeto à implementação, ela deve especificar de forma detalhada o software para ser criado um manual do usuário, contendo informações referentes à utilização do software e exemplos de atividades pedagógicas. O JFLAP neste quesito se destaca por apresentar um tutorial completo, explicando suas principais funcionalidades, além de disponibilizar para download todos os arquivos utilizados no manual de forma a facilitar o aprendizado da forma mais rápida e eficiente ao usuário. O Auger já apresenta um tutorial modesto, com apenas uma noção de como se utiliza o software, muitas partes do tutorial disponível, como no menu “algoritmos”, que dá acesso à aplicação dos algoritmos, há apenas citações sem apresentar nenhum exemplo ou figura. O Simulador de Autômatos já trás um tutorial que pode ser mais interessante para o usuário, o tutorial do software está disponível através de um vídeo no qual são apresentadas as principais funcionalidades da aplicação, o ponto negativo está na falta de áudio no vídeo o que poderia ajudar em um melhor entendimento de algumas partes explicadas.

4.6 Portabilidade

Portabilidade são atributos que evidenciam a adequação do software aos equipamentos onde serão instalados. (ROCHA, 2001)

Em relação a portabilidade, isso é, a possibilidade de utilização dos softwares em questão em diversas plataformas com pequeno esforço de adaptação, o JFLAP e o Auger por serem desenvolvidos em Java, são compatíveis com os sistemas operacionais Windows, MAC OS e UNIX/LINUX, já o software Simulador de Autômato está disponível para download em arquivo executável e informa em seu site oficial que é compatível com o Windows, e que teste com MAC OS e UNIX/LINUX ainda não foram realizados.

4.7 Retorno do investimento

Retorno de investimento são atributos que evidenciam a adequação do investimento na aquisição do software. (ROCHA, 2001)

Em relação ao retorno de investimento, todos os softwares são gratuitos e estão disponíveis após preenchimento de formulários. Os softwares apresentados cumprem com o que se propõem a fazer de forma satisfatória, de forma a serem boas opções para o auxílio do ensino de LFA.

4.8 Finalidade didática

Finalidade didática é a capacidade de levar o aluno/usuário a "construir" conhecimento relacionado com seu currículo escolar. (OLIVEIRA, 2001)

Todos os softwares são bem similares neste quesito, sendo que dois softwares merecem destaque, o JFLAP, por possuir um maior número de opções de manipulação de formalismos (Gramáticas, Autômatos e Expressões Regulares) de maneira a possibilitar experimentação das estruturas das linguagens, conversão entre estruturas equivalentes e análises como mostrado no tópico 3.1.1 (Métodos de Uso - JFLAP) e representada na figura 4.1. Outro software que possui uma característica didática que se pode destacar é o Auger com a possibilidade de gerar um programa (código) em Java que permite receber sentenças de entrada e verificar se pertencem à linguagem reconhecida pelo autômato que está sendo manipulado, sendo essa uma opção importante, principalmente por ser um software da área de Computação.

4.9 Comparação e avaliação dos softwares

Na tabela 2 é exiba uma tabela mostrando algumas características dos softwares discutidos no trabalho: Sistema Operacional suportado, tipo de licença e as funções realizadas por cada um para uma melhor comparação.

Tabela 2 - Quadro Comparativo dos Softwares.

Características		Ferramentas		
		JFLAP	Auger	Simulador de Autômatos
Sistema Operacional	Windows	√	√	√
	GNU/Linux	√	√	
	MAC/OS	√	√	
Licença	GLP	√	√	√
	Comercial			
Funções	Simular AFD	√	√	√
	Simular AFND	√	√	√
	Simular AFND- ε	√	√	√
	AFD → AFND	√	√	√
	AFD → ER	√	√	
	AF → GR	√	√	√
	GR→ AF	√	√	√
	ER → AFND- ε	√	√	
	Minimizar AFD	√	√	
	Simular APD	√		√
	Simular MT	√		√
	Gerar Código		√	

Fonte: Autor.

Para uma melhor visualização e entendimento dos critérios de avaliação considerados relevante por Rocha (2001) e Oliveira (2001) para um software educacional, cada um desses critérios é conceituado, conforme a tabela 3, recebendo uma gradação qualitativa para eles.

Tabela 3 – Níveis de gradação qualitativa dos softwares.

Avaliação qualitativa
Insatisfatório
Regular
Bom
Excelente

Fonte: Autor

- Insatisfatório: Não atende ou atende minimamente às expectativas quanto ao esperado de desempenho.

- Regular: Atende parcialmente às expectativas quanto ao esperado de desempenho.

- Bom: Atende às expectativas quanto ao esperado de desempenho. Alcança o padrão esperado, demonstrando resultado plenamente satisfatório.

- Excelente: Supera as expectativas quanto ao padrão esperado e definido para a competência. Excede ao padrão esperado, demonstrando desempenho além das expectativas.

As tabelas 4 e 5 apresentam o resultado da análise comparativa entre os softwares JFLAP, Auger e Simulador de autômatos. A tabela 4 apresenta a "avaliação" em cada critério e a tabela 5 as "considerações finais" de acordo com a avaliação comparativa de cada programa. Veja as avaliações:

Tabela 4 - Análise comparativa entre os softwares.

Características	Ferramentas		
	JFLAP	Auger	Simulador de Autômatos
Características pedagógicas	Excelente	Excelente	Excelente
Facilidade de uso	Bom	Bom	Regular
Características da interface	Excelente	Excelente	Excelente
Adaptabilidade	Bom	Regular	Regular
Documentação	Excelente	Insatisfatório	Regular
Portabilidade	Regular	Regular	Insatisfatório
Retorno do investimento	Insatisfatório	Insatisfatório	Insatisfatório
Finalidade didática	Excelente	Excelente	Bom

Fonte: Autor

Tabela 5 – Considerações finais de acordo como a análise comparativa da tabela 4.

Ferramenta	**Avaliação**
JFlap	Excelente
Auger	Bom
Simulador de Autômatos	Regular

Fonte: Autor

Com isso podemos concluir que todos os softwares apresentam uma avaliação no mínimo satisfatória para auxiliar de forma eficaz o aprendizado de LFA, estando assim todos aptos para utilização no ambiente escolar. Sendo que o JFlap se destacou na comparação com os demais por ser um software mais completo em vários pontos como documentação, adaptabilidade e quantidade de funções capaz de realizar.

5 CONCLUSÃO

Estamos em uma época de grandes mudanças nas práticas educacionais, na qual o que se busca é alcançar diferentes resultados. A Computação, não está fora dessas mudanças, pelo contrário, ela vem se transformando a fim de propiciar modificações na educação, e por que não dizer na sociedade, em ritmo acelerado através de novos métodos, com a utilização de recursos tecnológicos, ou seja, softwares educacionais.

Apesar das vantagens que esses novos métodos podem trazer no processo de ensino – aprendizagem, muitos professores não se utilizam dos mesmos, por falta, às vezes, de compromisso, de interesse em melhorar suas aulas, e outras vezes por não saberem como adequar esses métodos e abordagens às suas aulas e/ou conteúdos. Infelizmente poucos professores percebem a necessidade de estar em constante atualização e aperfeiçoamento.

O conteúdo de Linguagens Formais e Autômatos (LFA) na Computação é visto por muitos como difícil de entender, mas isso ocorre quando não existe um trabalho prazeroso aos olhos de quem vê, tanto para quem ensina, quanto para quem aprende, sendo assim cabe ao professor usar métodos alternativos, no ensino de LFA, para que assim no decorrer de sua vida acadêmica, esses alunos não tenham tanta dificuldade em aprender novos conteúdo.

A forma tradicional de o professor lecionar suas aulas apresentando o conteúdo de forma estritamente expositiva e após pedir aos alunos que façam os exercícios, não há nada que estimule o aluno a gostar de LFA, e outras disciplinas, isso é um tanto quanto desagradável.

Com o objetivo de aumentar a qualidade da aprendizagem em LFA, por meio de softwares educacionais, muitos autores, pesquisadores defenderam e buscaram encontrar métodos inovadores para a abordagem de conteúdos que por meios destes convidam os alunos a desenvolver seu raciocínio, resolver problemas, como também criar, inovar. Possibilitando despertar um interesse maior pela disciplina e assim um melhor desempenho na mesma.

O estudo de Linguagens Formais e Autômatos são importantes para a formação de profissionais em Computação e em áreas de conhecimento afins. As definições e os problemas estudados são a base para compreender conceitos mais complexos em Teoria da Computação, tais como decidibilidade e complexidade de problemas. Além disso, a Teoria de Autômatos tem aplicações diretas na modelagem de vários sistemas físicos de estados finitos (como em circuitos lógicos, como por exemplo, um controle de semáforo) e na construção de compiladores e interpretadores para linguagens de programação.

Assim, a abordagem do ensino de LFA deve estimular a atenção e concentração dos alunos afim de que estes percebam que esses conceitos computacionais são essenciais para o processo de aprendizagem por possibilitar uma familiarização com os conceitos base da Computação.

Utilizando algumas ponderações e levando em consideração os critérios existentes relevantes por Rocha (2001) e Oliveira (2001) que privilegiam o processo de ensino e aprendizagem foram feitas algumas comparações entre os softwares existentes e como resultado pode-se perceber que todos os softwares tem um potencial no mínimo satisfatório para auxiliar a aprendizagem da disciplina de LFA, com o JFLAP apresentando uma certa superioridade em relação aos outros softwares (Auger e Simulador de Autômatos) em pontos importantes como documentação, adaptabilidade e quantidade de funções capaz de realizar.

A principal contribuição deste trabalho foi apresentar e discutir a relevância de algumas abordagens, como neste caso o uso de softwares, para o ensino e avaliação no processo de aprendizagem da LFA nos cursos de Computação, apresentando algumas ferramentas pedagógicas que podem auxiliar na quebra de antigos paradigmas ainda presentes na educação, como sistema de avaliação, práticas pedagógicas ultrapassadas, concepção de ensino, aprendizagem.

Assim, depende de cada professor, particularmente aos de LFA, encontrar alternativas para aumentar a concentração, atenção e motivação para a aprendizagem do conteúdo de forma eficiente e eficaz.

Visando trabalhos futuros, poderia ser realizados estudos de outras características importantes em softwares educacionais não abordadas nesse trabalho.

REFERÊNCIAS

ACIÓLY, Bedregal e Lyra. **Introdução à Teoria das Linguagens Formais, dos Autômatos e da Computabilidade**. Natal, RN.: Edições UNP, 2002

AMANTE, Lúcia. **As tecnologias digitais na escola e na educação infantil**. Pinhais: Editora Melo, 2011.

Auger – Ambiente para construção e simulação de autômatos finitos. Disponível em: http://www.evoluma.com/auger/index.html. Acesso em: janeiro de 2017.

BONFIM, Viviane Duarte. **Apostila de aula**: disciplina de Linguagens Formais e Autômatos. Lages, Agosto de 2009.
Disponível em: https://pt.scribd.com/document/64288252/Apostila-LFA. Acesso em: novembro de 2016.

CASILLO, Danielle. **Teoria da Computação**: Máquina de Turing. Disponível em:
http://www2.ufersa.edu.br/portal/view/uploads/setores/166/arquivos/CienciaComputacao/M%C3%A1quina%20de%20Turing.pdf. Acesso em: fevereiro de 2017.

FREDRICH. Ana Paula et al. **Módulo de minimização de autômatos finitos para o piramide** – programa interpretador e resolvedor de autômatos e máquinas de estados finitos. Anais do XVIII EAIC, 2009.

GOUVÊA, Marianna Centeno Martins; NAKAMOTO, Paula Teixeira. **Avaliação de software educacional**: Uma oportunidade de reflexão da educação na sociedade do conhecimento. Uberaba, IFTM–Campus, 2015.

JFLAP. Disponível em: http://www.jflap.org/. Acesso em: janeiro de 2017.

JOSE NETO, João. **A Teoria da computação e o profissional de informática**. RECET (Revista de Computação e Tecnologia da PUC-SP), Departamento de Computação/FCET/PUC-SP. Vol 1. Nº 1, p. 4- 21, Outubro de 2009.

LITWIN, E. **Tecnologia Educacional**: Política, história e propostas. Porto Alegre: Artes Médicas, 1997.

MASETTO, Marcos T.; MORAN, José Manuel; BEHRENS, Marilda Aparecida. **Novas tecnologias e mediação pedagógica**. 12 ed. Campinas: Papirus, 2000. 173p.

MENEZES, Paulo Blauth. **Linguagens Formais e autômatos**. 5. Ed. – Porto Alegre: Bookman: Instituto de informática da UFRGS. 2008.

OLIVEIRA, Celina Couto de; COSTA, José Wilson da; MOREIRA, Mercia. **Ambientes informatizados de aprendizagem**: Produção e avaliação de software educativo. Campinas: Papirus, 2001.

OLIVEIRA, Gerson Pastre de. **Discussões nos ambientes virtuais de aprendizado colaborativo**: a relevância do espaço proporcionado pelo fórum. In: XII ENDIPE. *Conhecimento local e conhecimento universal*. Curitiba, ago/set. 2004.

PRADO, Simone Domingues. Disponível em: http://wwwp.fc.unesp.br/~simonedp/zipados/TC01.pdf. Acesso em: Dezembro 2016.

RAMOS, Marcus Vinícius Midena. **Ensino de linguagens formais e autômatos em cursos superiores de computação**. RECET (Revista de Computação e Tecnologia da PUC-SP), Departamento de Computação/FCET/PUC-SP. Vol 1. Nº 1, p.22- 34, Outubro de 2009.

RANGEL, J. L.; GUEDES L. C. Disponível em: https://www.qconcursos.com/questoes-de-concursos/questao/7ead3c23-82. Acesso em: novembro de 2016.

RANGEL, José Lucas. Disponível em: http://www.inf.puc-rio.br/~inf1626/Apostila/lf3.pdf. Acesso em : dezembro de 2016.

ROCHA, Ana Regina Cavalcanti da. **Qualidade de Software**. São Paulo: Prentice Hall, 2001.

SAKATA, Tiemi Christine. **Tópicos em Computação 1**. Sorocaba, 2007.

SILVA, Ricardo Dutra da. **Teoria da Computação**. Departamento Acadêmico de Informática - DAINF-UTFPR. Curitiba, 2014.

SILVA, Rômulo César. **Apostila:** Teoria da Computação. Maio de 2007.

Simulador de Autômatos. Disponível em: http://www.simuladordeautomatos.com/. Acesso em: fevereiro de 2017.

SKINNER, B. F. **The technology of teaching**. New York: Appleton-Century-Crofts, 1968.

TAVARES, Orivaldo de Lira. Disponível em: https://inf.ufes.br/~tavares/labcomp2000/aula51.htm. Acesso em: janeiro de 2017.

TIMBOÍBA, C. A. P.; RIBON, I. S.Paim, I.P. de O.; Monteiro, S. R. Monteiro, S. A.; Guirardi, M.M.M. **A inserção das tics no ensino fundamental:** limites e possibilidades. Revista Científica de Educação à distância,v.2, n.4,2011.

VALENTE, José A. **Análise dos diferentes tipos de software usados na Educação**. Campinas, SP: Gráfica da UNICAMP, 1997b.

VALENTE, José A. **Diferentes usos do computador na educação**. Campinas, SP: Gráfica da UNICAMP 1992.

VALENTE, José A. **O uso inteligente do computador na educação**. Pátio - revista pedagógica, Porto Alegre, Editora Artes Médicas Sul Ano 1, Nº 1, pp, 1997a.

XING, Guangming. **Minimized Thompson NFA**, Disponível em: http://people.wku.edu/guangming.xing/thompsonnfa.pdf. Acesso em: fevereiro de 2017.