Pablo Vinícius Neves Oliveira

LFA Virtual – Uma ferramenta de ensino para a disciplina de linguagens formais e autômatos

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2009

Pablo Vinícius Neves Oliveira

LFA Virtual – Uma ferramenta de ensino para a disciplina de linguagens formais e autômatos

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

Orientador:
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2009

Pablo Vinícius Neves Oliveira

LFA Virtual – Uma ferramenta de ensino para a disciplina de linguagens formais e autômatos

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências da disciplina Projeto Orientado para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em __ do _____ de 2009

_____
Prof. Bráulio Adriano de Mello

_____
Prof. Marluce Rodrigues Pereira

_____
Prof. Joaquim Quinteiro Uchôa
UFLA
(Orientador)

Lavras
Minas Gerais - Brasil
2009

À família Neves.

Agradecimentos

Durante esse trabalho e durante a vida acadêmica, vários personagens estiveram presentes pelo caminho, e colaboraram positivamente de forma direta ou indireta.

Seja na pura motivação em relação aos estudos, seja na hora de ensinar aquela disciplina difícil. A vocês dedico minha gratidão.

Ao prof. Joaquim Quinteiro Uchôa, pela paciência na orientação desde trabalho.

A todos os professores do Departamento de Ciência da Computação e, por que não dizer, a Richard Dawkins.

E principalmente à minha família.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

1. AFD – Autômato Finito Determinístico
2. AFND – Autômato Finito Não Determinístico
3. GLC – Gramática Livre de Contexto
4. LLC – Linguagem Livre de Contexto
5. GLS – Gramática Sensível ao Contexto
6. LSC – Linguagem Sensível ao Contexto
7. GR – Gramática Regular
8. ER – Expressão Regular
9. GLD – Gramática Linear à Direita
10. GLE – Gramática Linear à Esquerda
11. GLUD – Gramática Linear Unitária à Direita
12. GLUE – Gramática Linear Unitária à Esquerda
13. IDE – Integrated Development Environment
14. JVM – Java Virtual Machine
15. JRE – Java Runtime Environment

LFA VIRTUAL – UMA FERRAMENTA DE ENSINO PARA A DISCIPLINA DE LINGUAGENS FORMAIS E AUTÔMATOS

Pablo Vinícius Neves Oliveira
Joaquim Quinteiro Uchôa

RESUMO

Linguagens Formais e Autômatos (LFA) é uma das disciplinas mais importantes do currículo de Ciência da Computação. Entretanto, devido à sua complexidade, os tópicos da disciplina não são facilmente absorvidos pelos alunos. Tendo isso em vista, neste projeto será realizada uma abordagem dos principais conceitos da matéria, bem como a implementação de uma aplicação *online* para o ensino de alguns desses tópicos.

Palavras-Chave: autômatos, ensino, linguagens.

LFA VIRTUAL – A LEARNING ENVIRONMENT FOR THE DISCIPLINE OF FORMAL LANGUAGES AND AUTOMATA.

ABSTRACT

Formal Languages and Automata (FLA) is one of the most important disciplines in the curriculum of the Computer Science course. However, due to its complexity, the content is not easily absorbed by the students. Focusing on this issue, this project will approach the main concepts of the discipline, as well as the development of a web application that will teach some of these concepts.

Keywords: automata, learning, languages.

1. INTRODUÇÃO

1.1. Contextualização e motivação

Este trabalho apresenta o LFA Virtual, uma aplicação de aprendizado e simulação de autômatos. A idéia para o trabalho surgiu durante o primeiro período de 2008. O material disponível para estudo da disciplina é excessivamente rebuscado e técnico, dificultando o entendimento dos tópicos abordados. Apesar de algumas aplicações já existirem com o mesmo propósito, geralmente estavam limitadas pela barreira da língua e eram de difícil manipulação. Com o passar do tempo se originou a idéia que uma aplicação visual e interativa diminuiria a dificuldade de absorção, facilitando o aprendizado para pessoas que nunca tiveram contato com algo parecido.

1.2. Problema e objetivos

Durante o projeto de pesquisa foram estudadas várias tecnologias existentes, aptas ao desenvolvimento de uma interface gráfica rica, além de manter a compatibilidade com outros sistemas operacionais. Foi decidido que este deveria ser um sistema possível de ser armazenado e executado na máquina local, mas também que estivesse disponível *online* para download. Além disso, a interface deveria manter a interatividade com o usuário. A escolha da linguagem de programação para desenvolver este ambiente foi de grande importância no trabalho uma vez que ela precisa ser robusta e portável. A linguagem Java, por atender todos esses requisitos, foi selecionada.

O objetivo deste trabalho é que ao final desse projeto fosse obtida uma aplicação de alta confiabilidade e maleabilidade, disponível para os alunos da disciplina de Linguagens Formais e Autômatos utilizarem como complemento às aulas.

2. LINGUAGENS FORMAIS E AUTÔMATOS

Esta disciplina está presente na grade curricular do curso de Ciência da Computação. Nela são ensinadas as bases teóricas para a disciplina de Teoria da Computação. Neste trabalho serão abordados os seguintes tópicos:

- Gramáticas
- Expressões Regulares
- Autômatos Finitos Determinísticos
- Autômatos Finitos Não Determinísticos
- Autômatos Com Pilha

2.1. Gramáticas

Procedimentos são uma seqüência finita de operações claramente descritas. Esses procedimentos podem ser usados para definir linguagens de duas maneiras essenciais: como *geradores* ou *reconhecedores*. Geradores são procedimentos que enumeram os elementos da linguagem. Reconhecedores indicam quando uma seqüência faz parte da linguagem.

O tipo mais comum de gerador é a gramática. A idéia original de gramática vem do estudo de linguagens naturais, e as definições que apresentaremos aqui são essencialmente devidas ao linguista Noam Chomsky.

Fundamentalmente, uma gramática é composta por *regras de produção*, ou *regras de re-escrita*, através das quais é possível obter todos os elementos da linguagem a partir de um símbolo inicial, usando as regras para re-escrever (produzir) os elementos. Formalmente, define-se uma *gramática* G como sendo uma construção <N, Σ, P, S>, onde:

- N é um alfabeto de símbolos auxiliares, chamados de símbolos não terminais, ou, simplesmente, de não terminais.
- Σ é o alfabeto no qual a linguagem é definida, cujos elementos são os símbolos terminais, ou, simplesmente, terminais.
- P é o conjunto de regras de re-escrita, chamadas simplesmente de regras ou produções
- S é o símbolo inicial.

Define-se o *vocabulário* de G, como sendo $V = N \cup \Sigma$, o alfabeto composto pelos símbolos terminais e não terminais. O conjunto de regras P é uma relação binária no conjunto V* de cadeias de símbolos quaisquer (terminais ou não terminais), isto é, $P \subseteq V^* \times V^*$, correspondendo cada regra individual a um par de cadeias (α, β). Entretanto, em vez de $(\alpha, \beta) \in P$, a notação habitual para a regra que permite a reescrita de α como β é simplesmente $\alpha \rightarrow \beta$. Além disso, reúnem-se regras com o mesmo lado esquerdo α, tais como $\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, …, $\alpha \rightarrow \beta_n$, na abreviação $\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$.

Por exemplo, se define uma gramática G = < N, Σ, P, S > = < { S }, { 0, 1 }, { (S, 0 S 1), (S, ε) }, S > onde N = { S } é o conjunto de não terminais, Σ = { 0, 1 } é o conjunto de terminais, e P = { (S, 0 S 1), (S, ε) } = { $S \rightarrow 0S1$, $S \rightarrow \varepsilon$ } = { $S \rightarrow 0S1 \mid \varepsilon$ } é o conjunto de regras.

Para mostrar que a cadeia 000111 faz parte da linguagem associada à gramática, deve-se seguir a partir de S, os seguintes passos intermediários:

S
0S1
00S11
000S111
000111.

Assim, por três vezes S é substituído por 0S1, e finalmente, S é substituído pela sequência vazia ε. Como veremos a seguir, aplicar uma regra $\alpha \rightarrow \varepsilon$ é equivalente a simplesmente remover α.

2.1.1. A linguagem de uma gramática

Define-se a linguagem da gramática G = <N, Σ, P, S> por: L(G) = { x ∈ Σ* | S ⇒* x }. Ou seja, a linguagem da gramática G é constituída pelas sequências x, que são compostas apenas de símbolos terminais e que podem ser obtidas em um número arbitrário de passos de derivação a partir do símbolo inicial S, usando as regras de P.

Por exemplo, na linguagem L(G) = { x ∈ 0^i1^i | i ∈ **N** }, cada sequência da forma 0^i1^i pode ser derivada a partir de S, ou seja, L(G) ⊇ { 0^i1^i | i ∈ **N** }. Basta observar que uma derivação a partir de S, em que se utiliza i vezes a regra S → 0S1 e uma vez a regra S → ε, gera exatamente 0^i 1^i . Uma sequência de terminais derivada de S tem a forma 0^i1^i, ou seja, L(G) ⊆ { 0^i1^i | i ∈ **N** }.

2.1.2. A hierarquia de Chomsky

Hierarquia de Chomsky é a classificação de gramáticas formais descrita em 1959 pelo linguista Noam Chomsky. Esta classificação possui 4 níveis, sendo que os dois últimos níveis são amplamente utilizados na descrição de linguagem de programação e na implementação de interpretadores e compiladores.

A classificação começa pelo tipo 0, com maior nível de liberdade em suas regras, e aumentam as restrições até o tipo 3. Cada nível é um super conjunto do próximo. Logo, uma gramática de tipo *n* é conseqüentemente uma gramática de tipo *n - 1*.

- **Gramáticas tipo 0** (*Recursivamente enumeráveis*).
  Exatamente as gramáticas vistas na definição anterior. As regras de uma gramática tipo 0 são regras da forma α → β, com α e β quaisquer.

- **Gramáticas tipo 1** (*Sensíveis ao contexto*, ou *GSC*).
  As gramáticas tipo 1 são as gramáticas com regras da forma α → β, em que se exige | α | ≤ | β |; é entretanto permitida uma regra que viola esta restrição: uma gramática tipo 1 pode ter a regra S → ε, se S não aparece do lado direito de nenhuma regra.

- **Gramáticas tipo 2** (*Livres de contexto*, ou *GLC*).

  As gramáticas tipo 2 são as gramáticas com regras da forma $A \rightarrow \beta$, onde A é um símbolo não terminal, e β é uma sequência qualquer de V*, possivelmente vazia.

- **Gramáticas tipo 3** *(Regulares)*.

  As gramáticas tipo 3 só podem ter regras dos três tipos descritos a seguir:

  $A \rightarrow a$ B, onde A e B são não terminais, e *a* é um terminal;

  $A \rightarrow a$, onde A é um não terminal, e *a* é um terminal;

  $A \rightarrow \varepsilon$, onde A é um não terminal.

Se uma linguagem é gerada por uma gramática tipo 0, ela é uma linguagem tipo 0; se tem uma gramática tipo 1, ela é uma linguagem tipo 1, ou uma linguagem sensível ao contexto (LSC); se tem uma gramática tipo 2, ela é uma linguagem tipo 2, ou uma linguagem livre de contexto (LLC). Se tem uma gramática do tipo 3, ela é uma linguagem do tipo 3 ou Regular. A tabela 2.1 a seguir detalha a hierarquia de Chomsky, assim como a figura 2.1.

Tabela 2.1 - Hierarquia de Chomsky

Teoria de Autômatos: Linguagem Formal e Gramática Formal			
Hierarquia de Chomsky	**Gramática**	**Linguagem**	**Reconhecedor**
Tipo-0	Estrutura de frase	Recursivamente enumerável	Máquina de Turing
	Estrutura de frase	Recursiva	Máquina de Turing
Tipo-1	Sensíveis ao contexto	Sensíveis ao contexto	Máquina de Turing com memória limitada
Tipo-2	Livre de contexto	Livre de contexto	Autômato com pilha
Tipo-3	Regular	Regular	Autômato finito

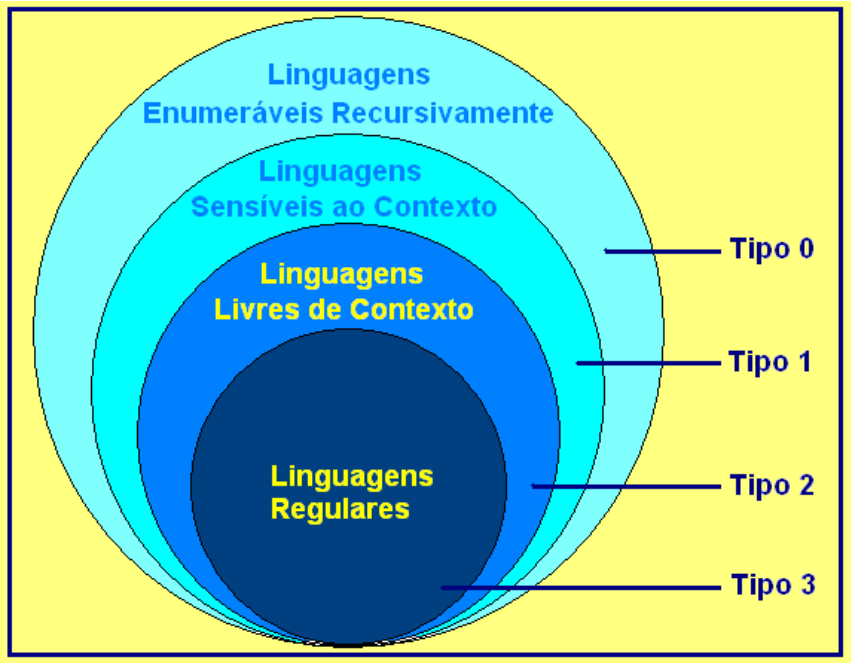


Figura 2.1 – Hierarquia de Chomsky

Neste projeto trabalharemos especificamente com gramáticas regulares, do tipo 2, e gramáticas livres de contexto, do tipo 3.

2.2. Linguagens Regulares

Segundo a hierarquia de Chomsky, a Linguagem Regular trata-se de uma linguagem mais simples, sendo possível desenvolver algoritmos de reconhecimento ou de geração de pouca complexidade, grande eficiência e fácil implementação.

Rangel (1999) descreve o estudo das Linguagens Regulares (ou tipo 3) como visto através de vários formalismos:

- Operacional ou reconhecedor – Uso dos autômatos finitos (determinístico, não determinístico)
- Axiomático ou gerador – Gramática Regular
- Denotacional – Expressão regular

Nos tópicos a seguir, serão abordados cada um destes individualmente.

2.2.1. Autômatos Finitos Determinísticos

Autômatos são dispositivos de computação abstratos que foram alvos de estudos antes mesmo de surgirem os computadores. Dentre estes dispositivos os mais simples são os autômatos finitos, eles são capazes de descrever linguagens regulares.

Um autômato é um conjunto de estados que possui um controle que desloca entre os estado de acordo com uma entrada externa. Segundo Prado (2008), um autômato finito determinístico normalmente chamado de AFD é formado por um conjunto de cinco elementos $(Q, \Sigma, \delta, q_0, F)$ onde :

Q é o conjunto finito de estados;

Σ é conjunto finito de símbolos de entrada conhecido também como alfabeto;

δ é a função de transição que retorna um estado ao receber um estado e um símbolo como argumentos.

q_0 é o estado inicial e pertence ao conjunto Q;

F é o conjunto finito de estados finais ele é um sub-conjunto do conjunto Q;

O AFD tem a capacidade de processar palavras de forma que ela pode ser aceita ou não, sendo que para qualquer palavra de entrada sempre existira uma resposta “aceito” ou “não aceito”. Para que uma palavra seja aceita ela deve terminar em um estado final, caso contrário a palavra não é aceita. Durante o processamento da palavra o autômato aplica a função de transição para cada símbolo da palavra de entrada de acordo com o valor e a posição em que esta se encontra. A Figura 2.2 detalha os símbolos que podem pertencer a um autômato.

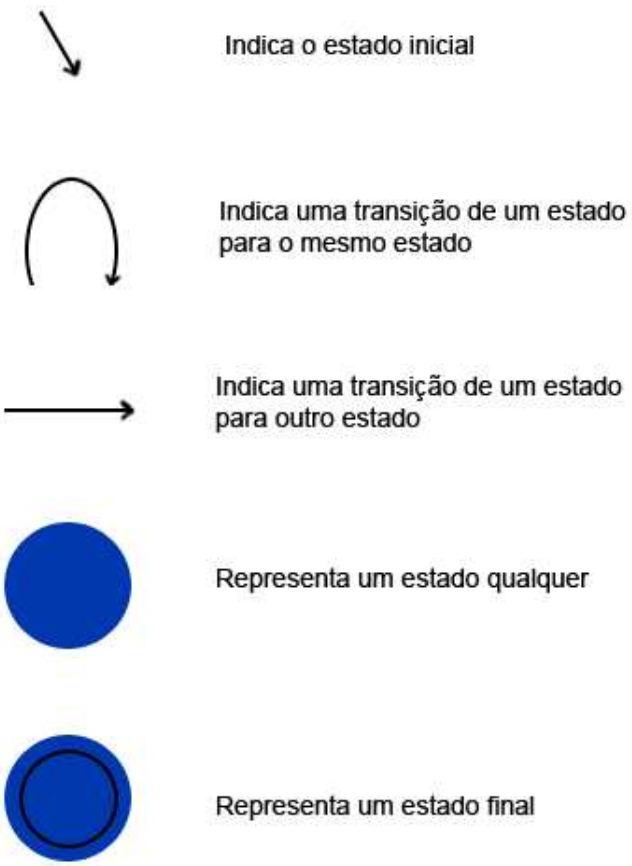


Figura 2.2 – Símbolos que representam um autômato.

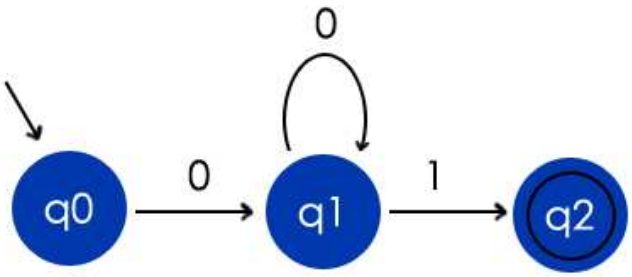


Figura 2.3 - Autômato Finito Determinístico.

A definição formal do AFD representado graficamente na Figura 2.3 é:

$A = (Q, \Sigma, \delta, q_0, F)$

Onde:

Q = {q0, q1, q2}

Σ = {0, 1}

$\delta \Rightarrow$ δ (q0,0)=q1, δ (q1,0)=q1, δ (q1,1)=q2

q_0 = q0

F = {q2}

O exemplo acima é um AFD que começa com vários “0” mas sempre terminará com um único número “1”. A linguagem de um autômato corresponde a todas as palavras que podem ser aceitas pelo mesmo.

2.2.2. Autômatos Finitos Não Determinísticos

Sipser (2007) menciona que se uma máquina está em determinado estado, e ela lê o próximo símbolo de entrada, sabemos qual será o próximo estado. Essa seria chamada de computação determinística.

"O não-determinismo é uma importante generalização dos modelos de máquinas, sendo de fundamental importância no estudo da teoria da computação e da teoria das linguagens formais." (MENEZES, 2002).

Em uma máquina não-determinística, várias escolhas para o próximo estado podem existir em qualquer ponto. O não-determinístico é uma generalização do determinismo. Logicamente então, todo autômato finito determinístico é também um autômato finito não-determinístico.

O conceito de não determinismo de um autômato de acordo com Rangel (1999) significa que:

- A função de transição pode conduzi-lo a múltiplos estados, ou mesmo a nenhum estado.
- Podem ocorrer transições por ε .

Autômatos Finitos Não Determinísticos, geralmente chamados de AFND, possuem um grande poder de simplificação. Para exemplificar isso, as Figuras 2.4 e 2.5 representam um AFND e sua conversão para AFD, respectivamente.

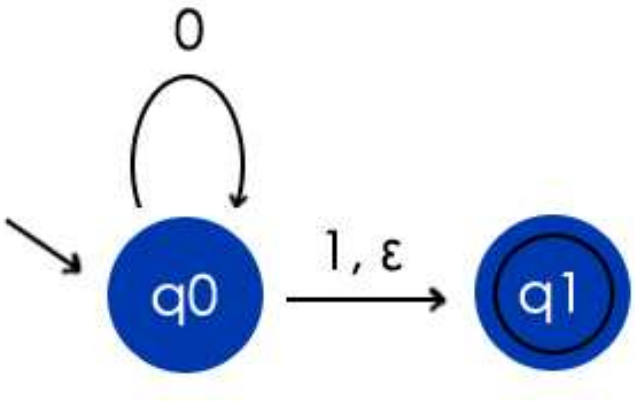


Figura 2.4 - Autômato Finito Não Determinístico.

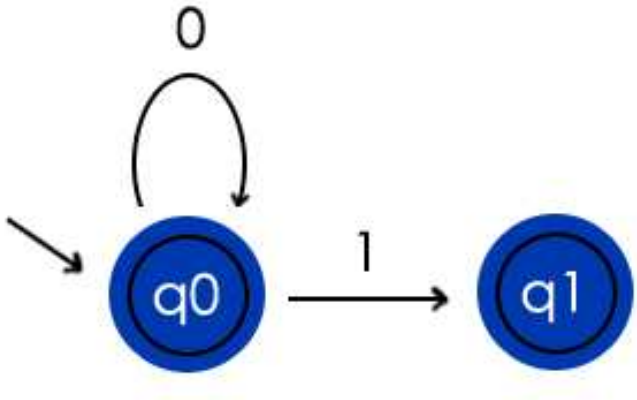


Figura 2.5 – AFD originado do AFND.

A formalização de um AFND é parecida com a de um AFD. Sendo que o AFND também é composto por uma 5-upla $(Q, \Sigma, \delta, q_0, F)$, onde:

Q é o conjunto finito de estados;

Σ é conjunto finito de símbolos de entrada conhecido também como alfabeto;

δ é a função de transição que retorna um conjunto de estados ao receber um estado e um símbolo como argumentos, sendo que o símbolo passado como argumento para a função de transição pode ser ε.

q_0 é o estado inicial e pertence ao conjunto Q;

F é o conjunto finito de estados finais ele é um sub-conjunto do conjunto Q;

A definição formal do AFD é:

$A = (Q, \Sigma, \delta, q_0, F)$

Onde:

Q = {q0, q1 }

Σ = {0, 1, 2}

$\delta \Rightarrow \delta$(q0,0)={q0}, δ(q0,1)={q2}, δ(q0, 2)= {q2}

q_0 = q0

F = {q1}

Apesar desta simplificação na generalização de problemas o não-determinismo não possibilita a definição de linguagens que não possam ser definidas por um AFND.

2.2.3. Equivalência entre AFND e AFD

Sabe-se que um autômato M_1 é equivalente a um autômato M_2 se, e somente se, $L(M_1) = L(M_2)$. Então se pode transformar um AFND em AFD para que a linguagem lida pelo AFND seja do tipo regular.

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFND qualquer. Seja $M' = (Q', \Sigma', \delta', < q_0 >, F')$ um AFD construído à partir de M como segue:

Q' é o conjunto de todas as combinações, sem repetições, de estado de Q as quais são denotados por $<q_1, q_2,...,q_n>$, onde q_j pertence à Q para j em {1, 2, ..., n}. Nesse caso a ordem dos elementos não distingue mais combinações. Por exemplo, $<q_1q_2> = <q_2q_1>$.

δ' é tal que $\delta'(<q_1...q_n>,a) = <p_1...p_m>$ se e somente se, $\delta(\{q_1, ..., q_n\},a) = \{p1, ..., p_m\}$. Ou seja, um estado de M' representa uma imagem dos estados de todos os caminhos alternativos de M.
$<q_0>$ é o estado inicial.

F' é o conjunto de todos os estados $<q_1q_2...qn>$ pertencentes à Q' tal que alguma componente q_j pertence a F, para j em {1, 2, ..., n}

Sendo assim, para qualquer AFND existe um AFD correspondente que é capaz de reconhecer a mesma linguagem que o AFND.

2.2.4. Gramática Regular

Seja G = < N, Σ, P, S > uma gramática e sejam A e B elementos de N e w uma palavra de Σ*.

Então G é uma:

a) Gramática Linear à Direita (GLD) se todas as regras de produção são da forma:
   A -> wB ou A -> w

b) Gramática Linear à Esquerda (GLE) se todas as regras de produção são da forma:
   A $\rightarrow$ Bw ou A $\rightarrow$ w

c) Gramática Linear Unitária à Direita (GLUD) se todas as regras de produção são como na GLD e adicionalmente |w| <= 1.

d) Gramática Linear Unitária à Direita (GLUE) se todas as regras de produção são como na GLE e adicionalmente |w| <= 1:

Por exemplo, a gramática G a seguir é uma GLD.

G = <{S}, {x, y}, P, S>

P = {S $\rightarrow$ xyS, S $\rightarrow$ x}

"Uma **gramática regular** é qualquer gramática linear... Se L é uma linguagem gerada por uma gramática regular, então L é uma **linguagem regular**". (PRADO, 2008)

2.2.5 Expressões Regulares

Outra notação utilizada para definir linguagens regulares é a expressão regular. Usa-se essa notação em construção de compiladores, editores, sistemas operacionais, protocolos, entre outros.

"As expressões regulares têm um papel importante em aplicações da ciência e da computação. Em aplicações envolvendo texto, os usuários podem querer fazer busca por cadeias que satisfaçam certos padrões. Expressões regulares provêem um método poderoso para descrever tais padrões. Linguagens modernas tais como PERL, e editores de texto, todos eles provêem mecanismos para a descrição de padrões usando expressões regulares". (SIPSER, 2007).

Uma expressão regular é definida a partir de conjuntos básicos e operações de concatenação e união. Sendo assim, formalmente uma Expressão Regular (ER) sobre um alfabeto $\sum$ é definida como:

a) Φ é uma ER e denota uma linguagem vazia.

b) λ é uma ER e denota a linguagem contendo exclusivamente uma palavra vazia, ou seja, λ.

c) x (símbolo do alfabeto $\sum$) é uma ER e denota a linguagem contendo a palavra {x}.

d) Se r e s são ER e denotam as linguagens R e S, respectivamente, então:

d.1) (r) é uma ER.

d.2) (r + s) é uma ER e denota a linguagem R $\cup$ S.

d.3) (r . s) é uma ER e denota a linguagem RS = {uv | u $\in$ R e v $\in$ S}.

d.4) r* é uma ER e denota a linguagem R*.

Obs.: A concatenação sucessiva (*) tem precedência sobre a concatenação (.) e a união (+). Também, a concatenação tem precedência sobre a união. A Tabela 2.2 mostra exemplos de expressões regulares.

Se r e s são ER e denotam as linguagens R e S, respectivamente, então:

a) L (r + s) = L(r) $\cup$ L(s)

b) L (r . s) = L(r) . L(s) ou L(r s) = L(r) L(s)

c) L((r)) = L(r)

d) L(r*) = (L(r))*

Tabela 2.2 – Exemplos de expressões regulares

Exemplos de expressões regulares

ER	Linguagem representada
aa	Somente a cadeia “aa”.
ba*	Cadeias que iniciam com “b”, seguida por zero ou mais “a”.
(a + b)*	Todas as cadeias sobre {a,b}.
(a + b)*aa(a + b)*	Todas as cadeias contendo “aa” como subcadeia.
(a + λ)(b+ba)*	Todas as cadeias que não possuem dois “a” consecutivos.

2.3. Linguagens Livres de Contexto

Linguagens Regulares são extremamente úteis no contexto de vários problemas, porém a forma de computação limitada destes autômatos tem um limite de complexidade. Em dado ponto, não é mais possível generalizar alguns problemas em forma de AFND ou AFD. Por exemplo, o balanceamento de parênteses em compiladores de linguagens de programação.

Para resolver problemas como este é que são utilizadas linguagens livres de contexto, que é uma classe de linguagem mais ampla que a das linguagens regulares e que utiliza como notações à partir de dois formalismos:

- Axiomático ou gerador – Gramática livre de contexto
- Operacional ou reconhecedor – Uso dos autômatos com pilha

2.3.1. Gramáticas Livres de Contexto

Gramáticas livres de contexto, geralmente referidas como GLC, são um poderoso método para descrever linguagens. Elas podem descrever certas características que tem uma estrutura recursiva, o que as torna úteis em uma variedade de aplicações.

Como fala Sipser (2007), foram primeiramente utilizadas no estudo de linguagens humanas. Uma maneira de entender o relacionamento de termos tais como nome, verbo e preposição e suas respectivas frases leva a uma recursão natural, pois frases nominais podem aparecer dentro de frases verbais, e vice-versa. GLCs capturam aspectos importantes desses relacionamentos.

“Uma aplicação importante de gramáticas livres de contexto ocorre na especificação e compilação de linguagens de programação” (SIPSER, 2007). Quando um compilador é projetado, os projetistas geralmente começam obtendo uma gramática para a linguagem. A maioria dos compiladores e interpretadores contém um analisador, que seria um componente que extrai o significado do programa antes de executar a interpretação ou gerar o código compilado.

Como descreve Prado (2008), as regras de uma gramática livre de contexto, são da forma $A \rightarrow \alpha$, onde α é uma cadeia qualquer de terminais e não terminais. O que caracteriza a gramática livre de contexto é a propriedade de que o símbolo não terminal A pode ser substituído pela cadeia α do lado direito da regra, onde quer que A ocorra, não importando o contexto. Isto é, do resto da cadeia que está sendo derivada. Por essa razão é possível representar derivações em GLC através de árvores de derivação.

Definindo formalmente, uma gramática G = < N, Σ, P, S > é dita ser uma **Gramática Livre de Contexto** se todas as suas produções, em P, são da forma:

$A \rightarrow \alpha$, onde $A \in V$ e $\alpha \in (V \cup T)^*$

“Como $A \in V$, então do lado esquerdo da produção, aparece somente uma variável. Entretanto, do lado direito, podem aparecer quaisquer combinações, já que $\alpha \in (V \cup T)^*$, ou seja, quaisquer combinações entre as variáveis e os símbolos terminais.” (PRADO, 2008).

Toda linguagem que utiliza uma gramática livre de contexto é uma **Linguagem Livre de Contexto**, geralmente definida como LLC.

A linguagem $L_1 = \{a^n b^n \mid n \geq 0\}$, por exemplo, é uma Linguagem Livre de Contexto. Sendo G_1 a gramática de L_1:

$G_1 = < \{S\}, \{a,b\}, S, P >$

P = {

S → aSa

S → bSb

S → λ

}

Então as cadeias { λ , ab, aabb, aaabbb, ...} pertencer a L_1.

2.3.2. Autômatos com pilha

Os aceitadores, ou reconhecedores, das linguagens livres de contexto são os chamados autômatos de pilha, ou APs. O autômato com pilha é semelhante a um autômato finito, porém ele possui uma pilha, não qual pode-se ler, escrever e remover sempre o último elemento inserido. Ou seja, seguindo o comportamento formalmente definido de uma pilha. A Figura 2.6 nos mostra detalhadamente os elementos que fazem parte da representação de autômatos com pilha.

A representação formal de um autômato com pilha é composta por seis elementos $(Q, \Sigma, \Gamma, \delta, q_0, F)$ onde :

Q é o conjunto finito de estados.

Σ é conjunto finito de símbolos de entrada conhecido também como alfabeto de entrada.

Γ é o alfabeto da pilha.

δ é a função de transição que retorna um estado e um símbolo do alfabeto da pilha a ser escrito na pilha ao receber um estado, um símbolo do alfabeto de entrada e um símbolo do alfabeto da pilha argumentos. O formato da função de transição é esse $\delta(Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon) = (Q \times \Gamma_\varepsilon)$.

q_0 é o estado inicial e pertence ao conjunto Q.

F é o conjunto finito de estados finais ele é um sub-conjunto do conjunto Q.

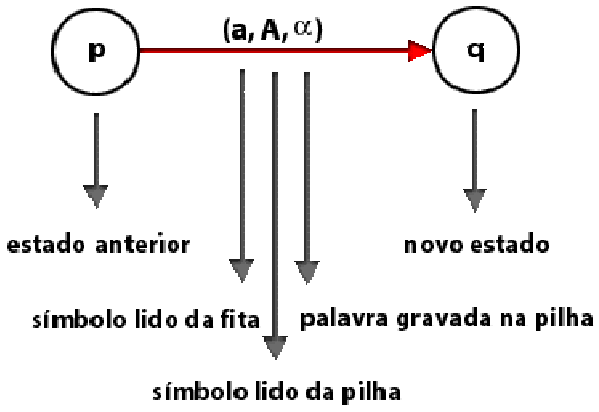


Figura 2.6 – Representação do um autômato com pilha.

3. REFERENCIAL TEÓRICO

3.1. Conceitos Básicos

3.1.1. Informática na educação

Segundo Freire (1987), no Brasil o modelo educacional infelizmente se baseia na transmissão de conhecimento, no acúmulo de saberes e na supremacia de um professor que detém o saber constituindo uma pedagogia bancária, da omissão e da passividade.

A introdução do computador na educação provocou e tem provocado uma verdadeira revolução na concepção de ensino e de aprendizagem. Na era contemporânea, a informação chega de todos os lados e formas. A informática entra no papel decisivo de acelerar o ensino e absorção de tudo isso. O estímulo ao aluno pela tarefa de aprender passa a ir além das salas de aula.

Segundo Valente (1993), os *softwares* educativos podem ser classificados em diferentes categorias:

- Tutoriais.
- De exercitação e prática.
- Simuladores.
- Jogos educacionais.

"Quando o computador é usado para passar a informação ao aluno, o computador assume o papel de máquina, e a abordagem pedagógica é instrução auxiliada por computador. Geralmente os *softwares* que implementam essa abordagem são os tutorias, os *softwares* de exercício-e-prática de jogos. Os tutorias enfatizam a apresentação das lições ou a explicitação da informação. No exercício-e-prática a ênfase está no processo de ensino baseado na realização de exercício com grau de dificuldade variado. Nos jogos educacionais a abordagem pedagógica utilizada é a exploração livre e o lúdico ao invés da instrução explícita e direta" (VALENTE, 1993).

"Esses *softwares* podem ser incrementados com características de inteligência como os *intelligent tutorial sytems*, capazes de identificar os erros mais freqüentes e ajudar os alunos a superá-los (como o sistema Buggy), auxiliar a resolução de problemas

específicos (como os sistemas especialistas), ou *softwares* para auxiliar o professor a planejar suas aulas ou a monitorar o desempenho dos alunos" (WENGER, 1987).

Essa classificação dos *softwares* educativos é mais de caráter didático para facilitar seu estudo sendo que um mesmo *software* educativo pode ser classificado em mais de um tipo.

3.1.2. Java

A linguagem Java foi desenvolvida na década de 90 pela empresa Sun Microsystems e a equipe de programadores foi chefiada por James Grosling. Em 1999 surgiu a plataforma para desenvolvimento e distribuição corporativa batizado de Java 2 Enterprise Edition (J2EE) e a plataforma Java 2 Mobile Edition (J2ME) para dispositivos móveis, celulares, PDAs e outros aparelhos limitados.

Java é uma linguagem computacional completa, adequada para o desenvolvimento de aplicações baseadas na rede internet, redes fechadas ou ainda programas *stand-alone*.

A linguagem obteve sucesso em cumprir os requisitos de sua especificação, mas apesar de sua eficiência não conseguiu sucesso comercial. Com isso, a linguagem conseguiu uma popularização fora de série, passando a ser usada amplamente na construção de documentos *web* que permitam maior interatividade. Os principais *web browsers* disponíveis comercialmente passaram a dar suporte aos programas Java, e outras tecnologias em áreas como Computação Gráfica e banco de dados também buscaram integrar-se com o novo paradigma proposto pela linguagem, que seriam aplicações voltadas para o uso de redes de computadores.

Atualmente, a linguagem Java é a força propulsora por trás de alguns dos maiores avanços da computação mundial. Ela é uma linguagem orientada a objetos e sua execução é diferente das outras linguagens de programação, pois ela é compilada para um formato chamado *bytecode* e é executada pela maquina virtual Java. Desta forma a ela torna-se portável uma vez que sua máquina virtual é adaptada ao sistema operacional.

Uma característica importante de Java é a existência de várias bibliotecas que possibilitam a utilização de componentes desenvolvidos por outros programadores. A

junção de varias bibliotecas ou pacotes, como também são conhecidas, formam uma API (*Application Programming Interface*) que tem como função encapsular uma atividade de forma que o programador não precise envolver em detalhes de sua implementação mas apenas em usar seus serviços.

3.1.3. Netbeans

O NetBeans começou como um projeto estudantil, na República Checa em 1996. O objetivo foi o de escrever um programa similar ao Delphi para ser um IDE em Java. Foi o primeiro Java IDE (*Integrated Development Environment*) escrito em Java, com seu primeiro pré-lançamento em 1997.

O plano inicial para o negócio estava em desenvolver componentes *web*. Jarda Tulach, que projetou o IDE básico da arquitetura, surgiu com o nome NetBeans para descrever o que eles fariam. A partir de então o Netbeans se tornou um sucesso, e hoje é tido como o principal IDE disponível para a linguagem Java, ao lado do Eclipse. É uma ferramenta eficiente para escrever, compilar, depurar e implantar programas. É escrito em Java, mas pode suportar qualquer linguagem de programação. Existe também um enorme número de módulos para aprimorar o NetBeans. É um produto gratuito sem restrições de como ser utilizado.

Ele foi escolhido para ser o ambiente de desenvolvimento desse trabalho por ser extremamente simples de ser utilizado. Além disso, já possui uma série de ferramentas que serão necessárias ao desenvolvimento da aplicação final, e um construtor de interface extremamente intuitivo.

3.2. Estado da Arte

Atualmente, o âmbito de linguagens formais e autômatos tem sido muito explorado, com um destaque especial para a produção de artigos científicos. Como exemplo, no congresso da SBC de 2008 foi publicado um artigo da autoria de alunos da Universidade Estadual de Maringá (UEM), descrevendo uma ferramenta para auxílio didático no ensino de Teoria da Computação. Da Costa (2008) descreve uma ferramenta que possui uma interface gráfica e implementa modelos para linguagens regulares, no caso,

Autômatos Finitos Determinísticos (AFD), Autômatos Finitos Não Determinísticos (AFND) e Autômatos Finitos de Movimento Vazio (AFMV). Esta possui interface rebuscada e de difícil entendimento.

Outro exemplo recente é o artigo da Universidade Católica de Brasília (UCB), em que Neuhauss (2008) descreve um editor/simulador de autômatos em um Desktop compartilhado sobre Grid. Visando uma abordagem "mão na massa", editores/simuladores de autômatos permitem o trabalho de forma mais ativa para os estudantes. Tais ferramentas, porém, não incorporam conhecimento do domínio nem observam a evolução do estudante a fim de prover retroalimentação personalizada. Também não promovem a aprendizagem colaborativa. O projeto proposto surge como solução a tal lacuna, buscando prover colaboração síncrona na aprendizagem de LFA. Para tanto, parte de trabalhos prévios considerando (i) a aprendizagem colaborativa no espectro mais amplo da Ciência da Computação e (ii) a interação em um Desktop Compartilhado sobre Grid. Apesar disso, o *software* possui poucas funções, pouca interatividade e não possui material de referência apropriado.

Existem simuladores como o JFLAP podem contribuir no processo de aprendizagem, porém não incorporam conhecimento do domínio nem observam a evolução do estudante afim de prover retroalimentação personalizada. Além disso o JFLAP restringe o aprendizado do usuário comum através da barreira da língua. Endereçando tal lacuna, um Sistema Tutor Inteligente pioneiro para o ensino de LFA é o FLUTE. Além de permitir o trabalho de forma mais atrativa e intuitiva, as motivações para o projeto deste sistema recaem sobre a relevância do domínio para a formação do profissional de Ciência da Computação ou engenharia e na carência de ambientes computacionais de suporte à aprendizagem de LFA. Sua principal limitação é a interface pouco amigável, que dificulta o entendimento imediato do usuário, e também é limitado pela língua.

JFLAP e FLUTE podem ser explorados de forma complementar por professores e estudantes. Entretanto, nenhuma dessas ferramentas provê a aprendizagem colaborativa. O ambiente JFLAP foi proposto, porém de forma assíncrona. A colaboração síncrona, entretanto, mostra-se necessária para possibilitar a realização de exercícios em sala por pequenos grupos, ou ainda dar suporte aos trabalhos dos grupos a

distância. Na literatura, encontram-se relatos de projeto, implementação e uso de alguns ambientes computacionais, brevemente descritos a seguir, que provêem colaboração síncrona na aprendizagem mais ampla da Ciência da Computação.

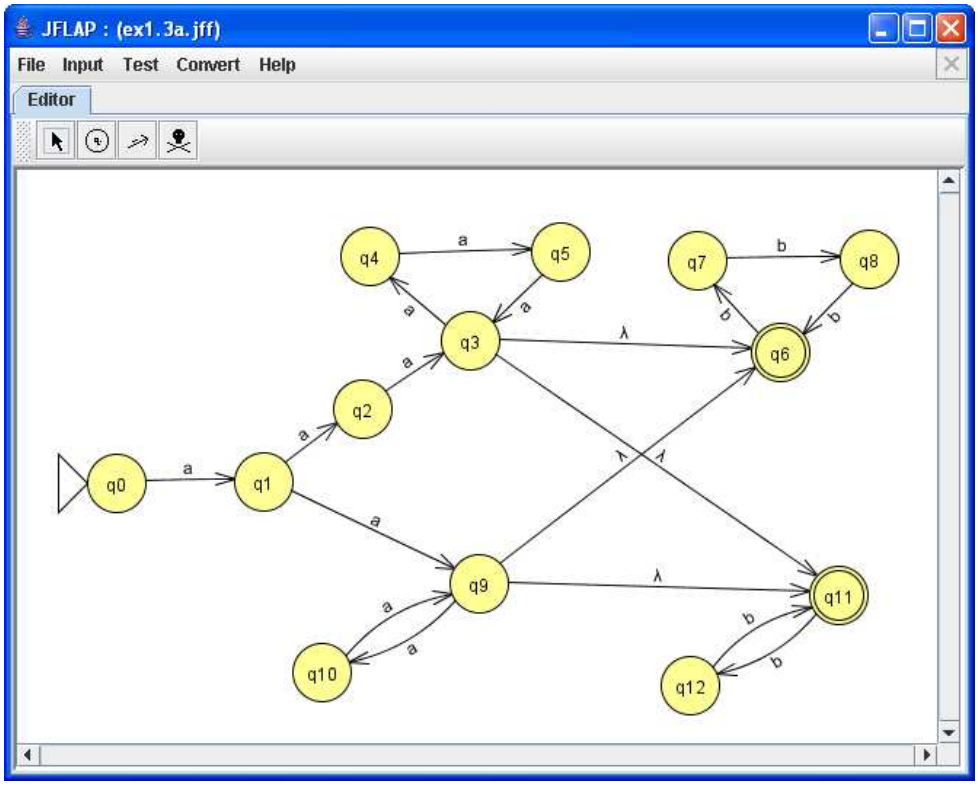


Figura 3.1 – Exemplo de autômato feito no JFLAP
Fonte: www.jflap.org

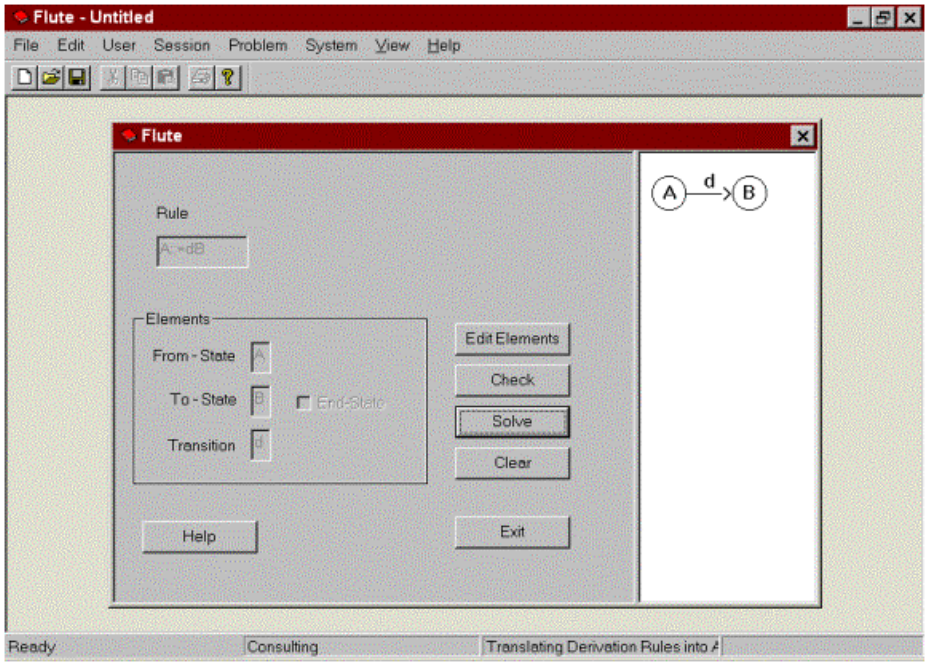


Figura 3.2 – Tela inicial do Flute
Fonte: www.ifets.info/journals/3_2/devedzic.html

Já Jukemura (2003) descreve a GAM, que é um simulador para auxiliar o ensino de Linguagens Formais e de Autômatos. Um autômato finito é um modelo matemático de uma máquina de estados finitos que lê símbolos de uma fita e aceita ou rejeita sua entrada. A Teoria de Autômatos Finitos é importante para a modelagem de máquinas reais com um conjunto finito de estados, para compactação de dicionários, e para várias outras aplicações. A ferramenta fornece condições de simular diferentes tipos de máquinas abstratas a fim de executar diversas operações e conversões que são comuns nessa área de estudo.

A GAM foi implementada com as funções de simulação de AFDs e AFNDs, com opções de análise passo a passo e análise total para *strings* de entrada com até vinte caracteres. Com o auxílio da interface gráfica que expõe a função programa, fica evidente a facilidade de estudar essas classes de uma forma muito clara e prática. A principal vantagem da ferramenta encontra-se nas funções de simulação de AFD e AFND, juntamente com as operações de conversão de AFNDs para AFDs e minimização de AFDs, as quais fornecem uma excelente opção prática à exposição teórica do estudo desses itens, além de economizar muito tempo em simulações de cadeias de caracteres de entrada para os autômatos durante demonstrações em sala de

aula. O ponto fraco deste se resume ao fato de ser pouco difundido e à ausência de código fonte disponível, impossibilitando a continuidade do trabalho.

Seguindo a mesma linha, Assis (2007) apresenta o SisLR, que foi concebido para ser uma aplicação que possibilitasse ao usuário manipular AFDs, AFNDs e ER s e suas equivalências, que seriam as transformações entre esses formalismos. O *software* foi modelado em três camadas, permitindo a modularização do sistema. Além disso, dispõe de um tutorial sobre a disciplina que explica cada tópico, com exemplos e exercícios, além dos documentos de ajuda. Esse *software* deixa a desejar no sentido de possuir poucas funcionalidades, e as que existem necessitam de um bom conhecimento prévio dentro do tema.

Como se pode observar, existem muitas referências disponíveis sobre o tema. O trabalho será baseado nessas referências, procurando desenvolver o tema com uma abordagem nova e de acordo com os objetivos descritos. O diferencial do trabalho a ser desenvolvido será a fácil usabilidade e o código aberto, que permitirá que o trabalho continue a ser desenvolvido por outros interessados.

4. METODOLOGIA

4.1. Tipo de pesquisa

O presente trabalho tem como objetivo desenvolver o aplicativo LFA Virtual, que é uma ferramenta interativa capaz de ser acoplada ao ambiente de ensino. Portanto, esta pesquisa é tecnológica, uma vez que se utiliza técnicas existentes em engenharia de *software* para o desenvolvimento desta ferramenta que será responsável pela simulação, validação e conversão de autômatos e gramáticas no software.

Quanto ao objetivo, esta é uma pesquisa exploratória, pois sua finalidade é criar maior familiaridade com certo fenômeno, visando torná-lo mais explícito. Para isso será construído um software.

O procedimento utilizado para desenvolver a presente pesquisa foi pesquisa experimental, uma vez que o empirismo foi determinante para a compreensão e evolução dos trabalhos. A partir de exemplos sobre estruturas gráficas em Java, como o de Buzzato (2008), a interface gráfica foi formulada, visando a facilidade de manipulação. Com essa estrutura pronta, foi possível programar as outras funcionalidades que utilizariam desta para alcançar os resultados propostos. Através da tentativa e erro os resultados aos poucos foram aparecendo.

Sendo que a pesquisa é bibliográfica, pois será feita a leitura, análise e interpretação de livros, periódicos, documentos científicos, etc. Todo o material recolhido será submetido a uma triagem, e o que for de utilidade ao trabalho será descrito, implementado e documentado.

4.2. Materiais utilizados

A plataforma de desenvolvimento do aplicativo é um computador com um processador Intel Pentium IV de 3,0 GHz, 4GB de RAM e placa de vídeo NVídia GeForceTM Series com 512MB de memória.

A principal ferramenta de desenvolvimento foi o Integrated Development Environment NetBeans 6.5, que já vem integrado ao J2SE. O NetBeans será utilizado tanto para desenvolver a biblioteca Java que faz todas as conversões, simulações e

validações de autômatos e gramáticas, quanto para a criação dos arquivos KSP que serão responsáveis pela integração do LFA Virtual com a Biblioteca Java.

A configuração mínima exigida para uma máquina executar o programa é possuir a JVM (Java Virtual Machine) instalada na máquina. Ela pode ser adquirida instalando o JRE (Java Runtime Environment).

5. RESULTADOS E DISCUSSÃO

Durante este capítulo serão apresentadas as principais funcionalidades e os algoritmos desenvolvidos durante o trabalho. A seguir estão as funcionalidades implementadas, assim como a descrição da forma de utilização de cada uma delas.

5.1. Salvar e recuperar autômatos

Primeiramente, para se criar um estado deve-se clicar no espaço de “Visualização e teste”, representado na figura 5.1. A interação entre esses estados como a criação de transições e características funcionais pode ser feita na aba de “Edição de estados”. Caso o usuário queira salvar algum autômato produzido dentro do software, ele pode utilizar do botão “Salvar”. Os dados serão gravados em um arquivo, e poderão ser recuperados utilizando o botão “Abrir”. Caso queira limpar a tela, o usuário deve pressionar o botão “Novo”. A aba de visualização e teste ficará em branco, e todos os elementos anteriores serão apagados.

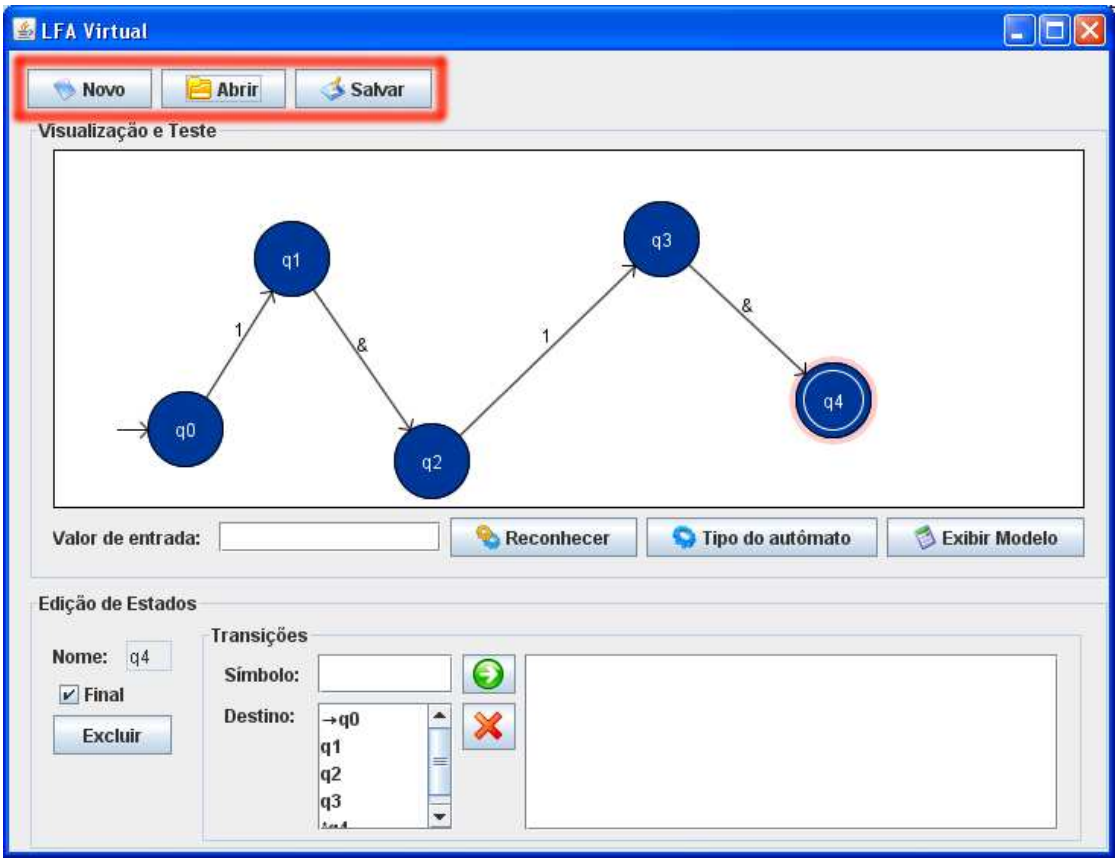


Figura 5.1 – Barra de tarefas destacada no ambiente do LFA Virtual.

5.2. Converter um autômato em modelo formal

O usuário tem a liberdade de criar autômatos usando o ambiente gráfico do sistema. A qualquer momento ele converter o resultado em um modelo formal, exibida no software. Para isso, após desenhado o autômato, deve-se clicar em “Exibir Modelo”.

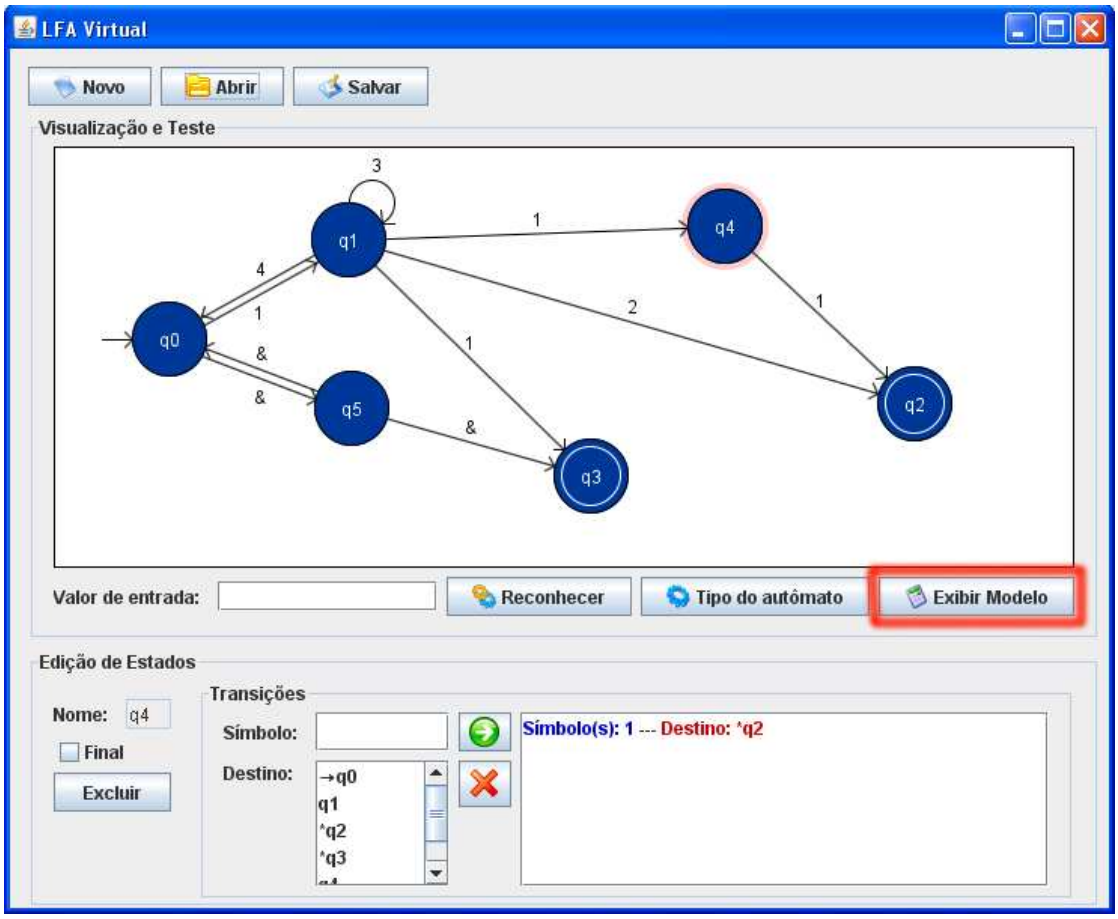
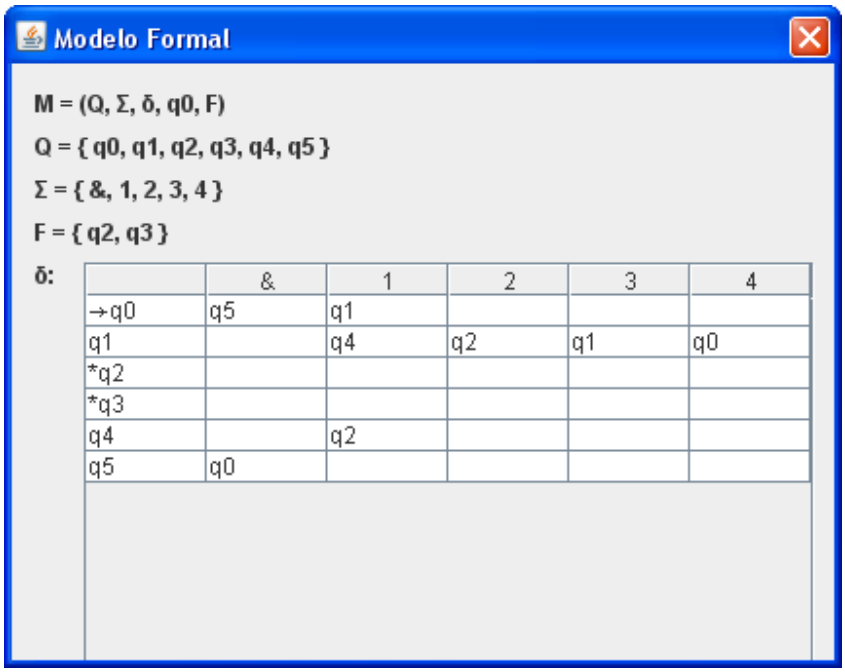


Figura 5.2 – Autômato desenhado no LFA Virtual, com o botão “Exibir Modelo” em destaque.



	&	1	2	3	4
→q0	q5	q1			
q1		q4	q2	q1	q0
*q2					
*q3					
q4		q2			
q5	q0				

Figura 5.3 – Modelo Formal gerado a partir do autômato da figura 5.2

5.3. Reconhecer tipo do autômato

O *software* é ser capaz de reconhecer qual o tipo do autômato desenhado pelo usuário. Estes poderão ser autômatos finitos determinísticos (AFD) ou não determinísticos (AFND).

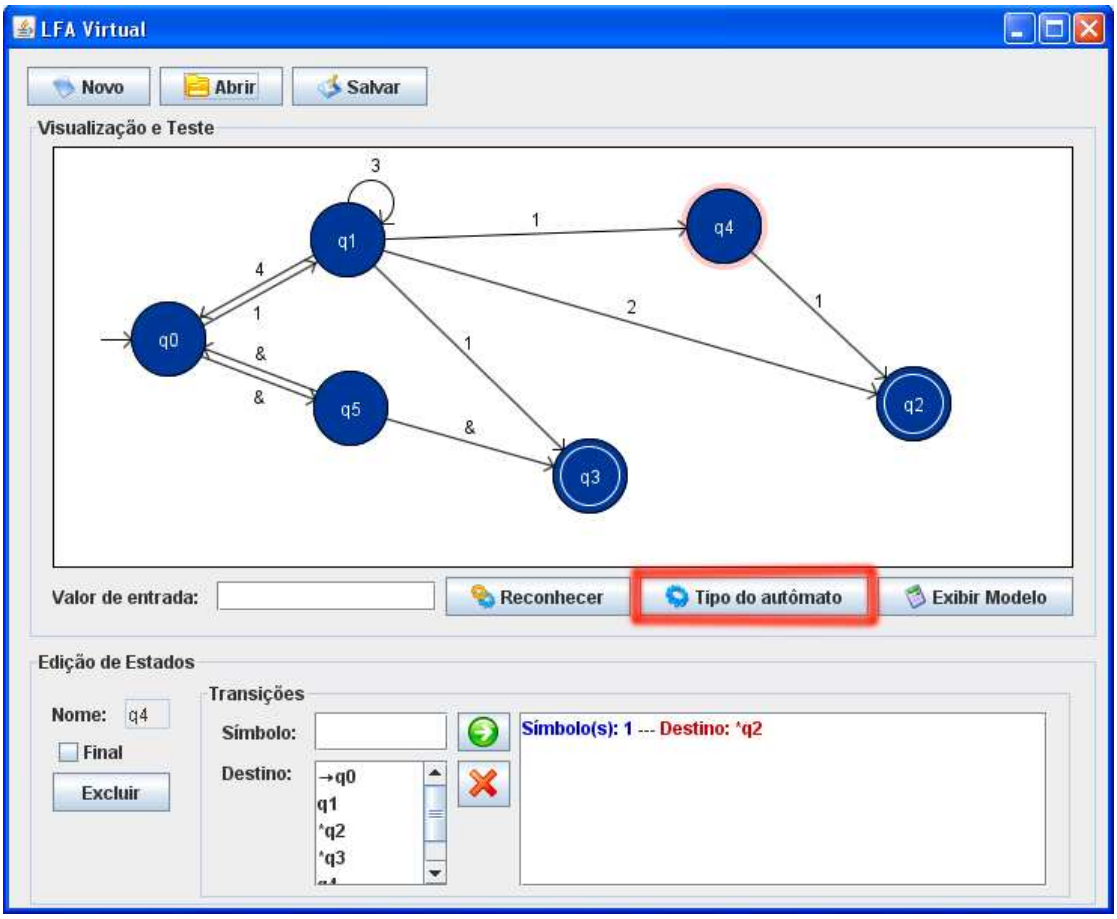


Figura 5.4 – Funcionalidade de reconhecimento do tipo de autômato em destaque

5.4. Reconhecer valores de entrada

O aplicativo deverá ser capaz de detectar se o autômato desenhado aceita o um valor de entrada inserido pelo usuário. Caso isso aconteça, o caminho possível será destacado no autômato.

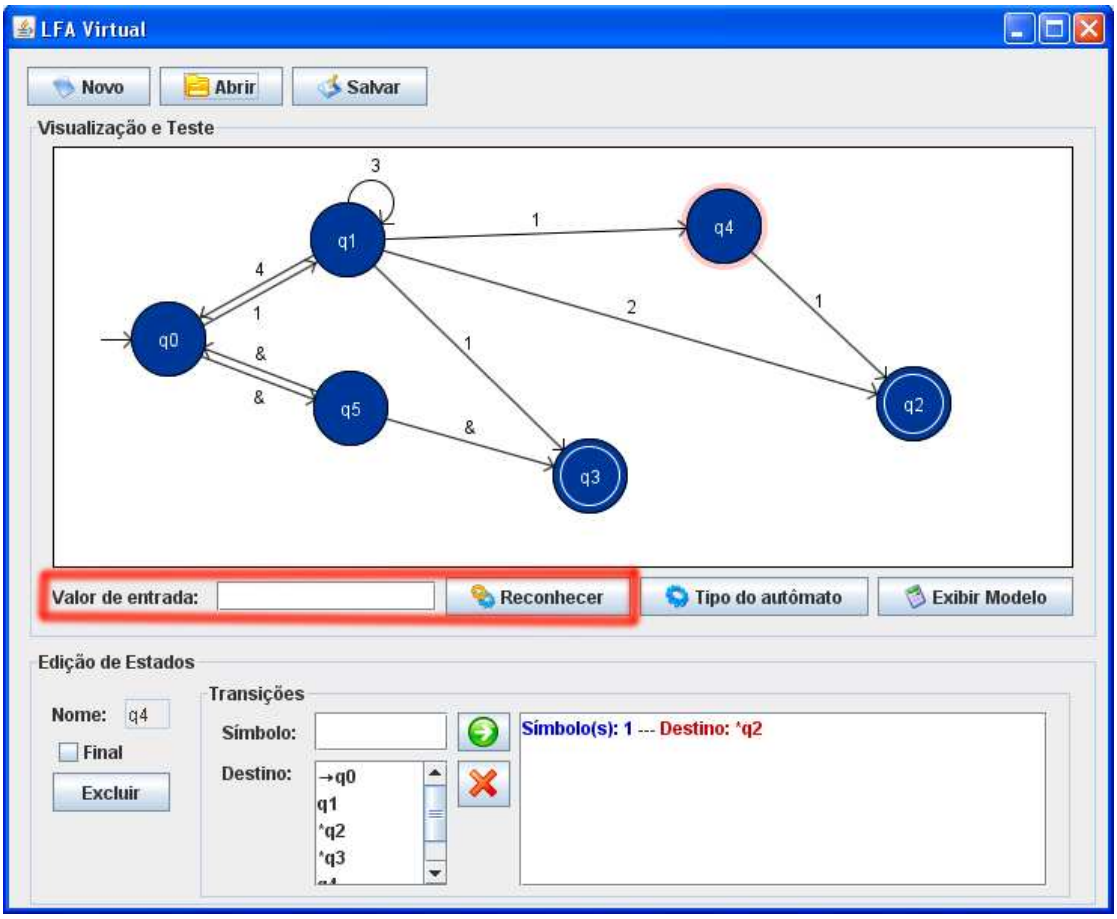


Figura 5.5 – Funcionalidade de reconhecimento de valores de entrada em destaque

5.5. Comentários finais

Ao final do desenvolvimento do projeto foram realizados testes com quatro usuários para avaliar a utilização do sistema, sua facilidade de uso e se atingiu os objetivos propostos. Após o uso do sistema, um questionário foi aplicado para que os usuários avaliassem o sistema de acordo com as seguintes características:

- Usabilidade
- Portabilidade
- Abrangência do Escopo

Estas deveriam ser avaliadas com apenas uma das seguintes atribuições: insatisfatório, satisfatório, bom, muito bom e excelente. No resultado final, o aplicativo foi avaliado como “muito bom” em todas as áreas, mostrando que os objetivos foram atingidos.

6. CONCLUSÕES

O objetivo principal deste trabalho foi a disponibilização de uma versão funcional do LFA Virtual. Pode-se dizer então que este objetivo foi alcançado. As funcionalidades de conversão, validação e simulação do trabalho foram concluídas com sucesso. Espera-se que o projeto desenvolvido possa auxiliar o aprendizado de interessados nesta área do conhecimento computacional, que é a de Linguagens Formais e Autômatos.

Outro ponto crucial do trabalho é que ele foi desenvolvido com o objetivo de ser progressivamente melhorado. Esse foi um dos motivos da escolha da linguagem Java, uma das mais populares do mundo, e por isso seu código se encontra aberto e à disposição de possíveis interessados em colaborar com o projeto.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ASSIS, R. J; SILVA, M. O. **Projeto e implementação de um software para auxílio ao ensino de linguagens regulares**. Barbacena/MG, 2007.

BITTENCOURT, J.R.; GIRAFFA, L.M. **Role-Playing Games, Educação e Jogos Computadorizados na Cibercultura,** In: I Simpósio de RPG em Educação**.** Rio de Janeiro: CCEAD/PUC-Rio, 2003.

BUZZATO, D. **Exemplo de Java 2D.** Disponível em http://www.guj.com.br/posts/list/118828.java. Consultado em 15/09/2008.

DA COSTA Y. M. e G., MENESES R. C., UBER F. R.: **Uma ferramenta para auxílio didático no ensino de Teoria da Computação**, In: XXVIII Congresso da SBC, 2008, Fortaleza. XXVIII anais do Congresso da SBC, 2008.

DOGNINI, M. J.; RAABE, Alice, A. L..**Eduling – Software Educacional para linguagens regulares**. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 14., 2003. Anais... Rio de Janeiro: [s.n.], 2003. p. 1-10.

FREIRE, Paulo. **Pedagogia do oprimido.** Disponível em: http://cms.mit.edu/games/education/. Consultado em 10/09/2009.

JUKEMURA, Anibal Santos ; NASCIMENTO, H. A. D. ; UCHOA, J. . **GAM - Um Simulador para Auxiliar o Ensino de Linguagens Formais e de Autômatos. In: Sociedade Brasileira de Computação**, São Leopoldo. Anais SBC 2005 - XIII Workshop sobre Educação em Computação, 2005.

MARTINS, S. M..: **LFA Scholar: Um ambiente de aprendizado e simulação de autômatos e gramáticas.** Monografia de graduação, Universidade Federal de Lavras, 2008.

MENEZES, P. F. B.: ***Linguagens Formais e Autômatos***. Porto Alegre: Sagra Luzzatto, 2002,165 págs.

NEUHAUSS, F. ; ELPÍDIO, F. ; NÓBREGA, G. M. ; CRUZ, F. W. ; SANTANA, C. . **Projetando a colaboração através de um editor/simulador de autômatos em um**

Desktop Compartilhado sobre Grid. In: Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza. XXI anais do Simpósio Brasileiro de Informática na Educação, 2008.

PRADO, S. G. D.; **Teoria da Computação e Linguagens Formais**. Disponível em http://wwwp.fc.unesp.br/~simonedp/disctcBCC.htm#C3. Consultado em 16/07/2009.

RANGEL, J. L.; **Linguagens Formais**. Disponível em www-di.inf.puc-rio.br/~rangel/lf.html. Consultado em 16/07/2009.

VALENTE, José Armando. **Diferentes usos do computador na educação.** Em Aberto. Brasília, ano 12, n.57, jan./mar. 1993. p.3-16.

WENGER, E. **Artificial Intelligence and Tutoring System: Computational and Cognitive Approaches to the Communication of Knowledge.** Califórnia: Morgan Kaufmann publishers, 1987.

ZAMBALDE, A. L.; SILVA E PÁDUA, C. I. P. **O documento científico em ciência da computação – suas partes e sua redação: estudo e análise em uma instituição federal de ensino superior (IFES).** Belo Horizonte/MG: DCC-UFMG, 2005.