STRAUS MICHALSKY MARTINS

LFASCHOLAR: UM AMBIENTE DE APRENDIZAGEM E SIMULAÇÃO DE AUTÔMATOS E GRAMÁTICAS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS - BRASIL
2008

STRAUS MICHALSKY MARTINS

LFASCHOLAR: UM AMBIENTE DE APRENDIZAGEM E SIMULAÇÃO DE AUTÔMATOS E GRAMÁTICAS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:
Linguagens Formais e Autômatos
Informática na Educação

Orientador:
Prof. Rudini Menezes Sampaio

LAVRAS
MINAS GERAIS - BRASIL
2008

Ficha Catalográfica preparada pela Divisão de Processos Técnico da Biblioteca Central da UFLA

Martins, Straus Michalsky

LFAScholar: Um Ambiente de Aprendizagem e Simulação de Autômatos e Gramáticas / Straus Michalsky Martins. Lavras – Minas Gerais, 2008. 45p : il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Linguagens Formais e Autômatos. 2. Informática na Educação. I. MARTINS, S. M. II. Universidade Federal de Lavras. III. LFAScholar: Um Ambiente de Aprendizagem e Simulação de Autômatos e Gramáticas.

STRAUS MICHALSKY MARTINS

LFASCHOLAR: UM AMBIENTE DE APRENDIZAGEM E SIMULAÇÃO DE AUTÔMATOS E GRAMÁTICAS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 17/06/2008

_____
Prof. Dr. André Vital Saúde

_____
Prof. Dr. Rêmulo Maia Alves

_____
Prof. Ms. Rudini Menezes Sampaio
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL
2008

Dedico aos meus pais, Andréa e Sergio,

e às minhas irmãs Raíssa e Bianca.

AGRADECIMENTOS

Agradeço aos meus pais, Andréia e Sergio, pelo apoio e ânimo principalmente nas horas difíceis.

Agradeço pelo carinho de minhas irmãs, Raíssa e Bianca que sempre estiveram dispostas a me ajudar.

Agradeço a minha tia Denise pelo apoio, que apesar da distância sempre conseguia guiar minhas decisões com seus conselhos e experiência.

Agradeço ao meu amigo de todas as horas, Tiago Vinícius, o qual trabalhou bastante para que este projeto tornasse realidade.

Agradeço ao professor Rudini pela orientação e confiança em mim depositada.

Agradeço também a todo colegas de turma pelo companheirismo e amizade.

LFASCHOLAR: UM AMBIENTE DE APRENDIZAGEM E SIMULAÇÃO DE AUTÔMATOS E GRAMÁTICAS

O presente trabalho apresenta a construção de um software educativo capaz de simular, validar e converter autômatos e gramáticas para ser utilizado como uma ferramenta de apoio no ensino da disciplina de Linguagens Formais e Autômatos. Este software educativo foi desenvolvido utilizando a tecnologias OpenLaszlo, Java e XML. Ele segue os paradigmas da Web 2.0 e foi chamado de LFAScholar. O resultado foi um ambiente capaz de executar a simulação de autômatos finitos determinísticos, fazer a conversão entre gramáticas regulares e autômatos finitos, converter gramáticas livres de contexto para forma normal de Chomsky e validar gramáticas regulares e autômatos finitos.

Palavras-chave: Linguagens Formais e Autômatos, software educativo, autômatos finitos, gramáticas.

LFASCHOLAR: AN ENVIROMENT FOR LEARNING AND SIMULATION OF AUTOMATONS AND GRAMMARS

The present work presents the building of an educational software capable of simulate, validate and convert automatons and grammar to be used as a support tool in the teaching of the Automata Theory. This educational software was developed using the technologies OpenLaszlo, Java and XML. It follows the web 2.0 paradigm and was called LFAScholar. The result was an environment capable of simulate deterministic finite automaton, do the conversion between regular grammars and finite automatons, convert context-free grammars to the Chomsky normal form and validate regular grammars and finite automatons.

Keywords: Automata Theory, educational software, finite automatons, grammars.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DFA	Deterministic Finite Automaton
FNC	Forma Normal de Chomsky
JSP	JavaServer Pages
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
LFA	Linguagens Formais e Autômatos
NFA	Non Deterministic Finite Automaton
XML	eXtensible Markup Language

1.INTRODUÇÃO

Este trabalho apresenta o ambiente de aprendizado e simulação de autômatos LFAScholar. O LFAScholar surgiu durante as aulas da disciplina de Linguagens Formais e Autômatos no ano de 2005, onde os alunos se motivaram para criar um software que pudesse ser usado na disciplina. O grupo de alunos, Claudiane Maria Oliveira, Diego Nogueira Teixeira, Jaime Daniel Correa Mendes, Straus Michalsky Martins e Tiago Vinícius Paiva do Carmo juntamente com o professor Rudini Menezes Sampaio, resolveram criar um simulador visual para os autômatos da disciplina. Esse grupo de alunos liderados pelo professor Rudini trabalharam em um projeto de pesquisa que iniciou o desenvolvimento do sistema que foi chamado de LFAScholar.

Durante o projeto de pesquisa foram estudadas várias tecnologias existentes e foi decidido que seria um sistema web que seguiria os paradigmas da Web 2.0. Para desenvolver o sistema nos paradigmas da web 2.0 foi escolhida a plataforma OpenLaszlo e a linguagem de programação Java. O grupo de pesquisa também desenvolveu a primeira versão do LFAScholar que possui um editor de autômatos no qual é possível criar autômatos graficamente.

Em 2007 o projeto de pesquisa terminou com esta versão inicial do sistema, porém ela possuía várias correções e melhorias para serem feitas. Foi neste momento em que o aluno Straus Michalsky Martins assumiu o projeto como seu projeto orientado com o objetivo de resolver estas pendências, criar algoritmos para simulação, validação e conversão de autômatos e gramáticas e disponibilizar o sistema para seus primeiros testes por alunos.

1.1. Considerações Iniciais

Com as grandes transformações da sociedade atual incluindo a educação, existe cada vez mais a necessidade de ferramentas pra dinamizarmos o método de ensino mudando a antiga metodologia Professor-Aluno onde o professor possui o conhecimento e o aluno só o absorve de forma que é necessário criar ferramentas que possibilitem a construção do conhecimento em conjunto onde tanto o professor como aluno possa adicionar conhecimento.

FINO(1998) diz que ninguém duvida que os contextos de aprendizagem escolar precisam de ser reestruturados para poderem suportar uma atividade mais centrada no

aprendiz, mais interativa, e estimulando mais a resolução de problemas de forma cooperativa.

Atualmente, com o desenvolvimento de novas tecnologias, é possível criar ambientes dinâmicos capazes de suprir as necessidades tanto de alunos como de professores no ensino.

Algumas disciplinas do curso de ciência da computação possuem esse caráter dinâmico em que apenas um quadro negro não é suficiente para explorar todo seu conteúdo. Entre essas disciplinas, podemos citar Linguagens Formais e Autômatos.

Desta forma, a maior motivação para criação do LFAScholar foi essa dinamicidade que a disciplina de Linguagens Formais e Autômatos possui.

1.2. Natureza e Importância do Assunto

"Antes de existirem computadores, na década de 1930, Alan Turing estudou uma maquina abstrata que tinha todas as características dos computadores atuais, pelo menos no que se refere ao quanto eles poderiam calcular." (HOPCROFT, 2001, P1). Este seria o inicio da teoria de autômatos que é o estudo de dispositivos de computação abstratos.

Na década de 1940 outros pesquisadores estudaram máquinas mais simples conhecidas hoje como autômatos finitos, inicialmente estes autômatos tinham como propósito modelar a função do cérebro, os mesmos se mostraram muito úteis para vários outros propósitos (HOPCROFT, 2001).

Noam Chomsky praticamente fundou a teoria dos autômatos, em meados de 1950, iniciou o estudo de gramáticas formais seu objetivo era desenvolver teorias relacionadas com as linguagens naturais. Entretanto, logo foi verificado que esta teoria era importante para o estudo de linguagens artificiais e, em especial, para as linguagens originárias na Ciência da Computação. Desde então, o estudo das Linguagens Formais desenvolveu-se significativamente e com diversos enfoques, com destaque para aplicações em Análise Léxica e Sintática de linguagens de programação, modelos de sistemas biológicos, desenhos de circuitos e relacionamentos com linguagens naturais.

Já em 1969 surgiram diversos modelos simples de computação: os autômatos finitos. Outros modelos mais complexos se seguiram, como a máquina de Post e a máquina de Turing. Além disso, a teoria foi engrandecida com a introdução de modelos de gramática e linguagens formais.

Atualmente, a área das Linguagens Formais e Autômatos (LFA) está bem estabelecida na ciência da computação e é uma disciplina obrigatória nos cursos acadêmicos.

Uma grande dificuldade no ensino de LFA é o caráter dinâmico dos modelos e sua simulação. Nesse trabalho, desenvolvemos um ambiente educativo e de simulação de LFA, que será útil para melhor fixação do conteúdo na disciplina.

1.3. Objetivos

O principal objetivo do presente trabalho é disponibilizar uma versão funcional do LFAScholar adicionando novas funcionalidades ao mesmo e corrigindo problemas que ele possui, de forma que possa ser usado por alunos que cursam a disciplina de Linguagens Formais e Autômatos. Esta ambiente auxiliará o ensino de Linguagens Formais e Autômatos permitindo a simulação de autômatos e gramáticas, a conversão entre Autômatos e Gramáticas e a validação dos mesmos.

Como objetivo específico, podemos citar a distribuição da biblioteca Java chamada LFALibrary que será desenvolvida como uma ferramenta utilizada no projeto do LFAScholar que poderá ser utilizada em outras aplicações uma vez que ela utiliza arquivos XML como entrada e saída. Para sua utilização em outras ferramentas basta seguir o padrão dos arquivos XML.

2. LINGUAGENS FORMAIS E AUTÔMATOS

Esta é uma disciplina presente na grade curricular de todos os cursos de Ciência da Computação. Nesta disciplina são ensinados conceitos base da teoria da computação tais como:

- Autômatos Finitos Determinísticos
- Autômatos Finitos Não Determinísticos
- Expressões Regulares
- Gramáticas Regulares
- Gramáticas Livres de Contexto
- Autômatos com Pilha
- Máquina de Turing

Em 1959 Noam Chomsky classificou as gramáticas formais em 4 níveis sendo que neste trabalho abordamos mais os dois últimos níveis que são o foco da disciplina de LFA.

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

Nesta seção daremos maior ênfase na definição de conceitos das Linguagens Regulares e Linguagens Livres de Contexto, pois são estes elementos que o sistema irá trabalhar inicialmente.

2.1. Linguagens Regulares

2.1.1. 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. Um autômato finito determinístico normalmente chamado de DFA – Deterministic Finite Automaton, 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 DFA 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 estado em que se encontra.

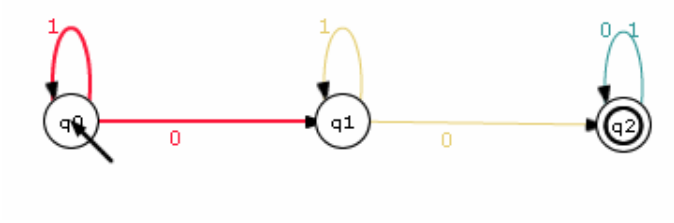

Figura 2.1 - Autômato Finito Determinístico.

E a definição formal do DFA representado graficamente na Figura 2.1 é:

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

Onde:

Q = {q0, q1, q2}

Σ = {0, 1}

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

q_0 = q0

F = {q2}

O exemplo acima é um DFA que aceita apenas palavras que contém no mínimo dois "0". A linguagem de um autômato corresponde a todas as palavras que podem ser aceitas pelo mesmo. E linguagem do exemplo da figura 2.1 é:

L(A) = { w | w tem ao menos dois "0"}

2.1.2. Autômatos Finitos Não-Determinísticos

Existem alguns momentos que é mais fácil a utilização de máquinas não determinísticas, pois é mais simples a generalização de problemas com utilização de autômatos não determinísticos. Nestes autômatos é possível existirem transições vazias (ε – epslon), a ausência de uma transição para certo símbolo e a existência de várias transições saindo de um mesmo estado com valores iguais.

"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).

Para exemplificar o poder de simplificação do autômato finito não determinístico normalmente chamado de NFA - Non deterministic Finite Automaton, observe as figuras 2.2 e 2.3 a linguagem de ambos autômatos é L(A) = { w | w possui "1" em sua antepenúltima posição}, porem fica mais simples o entendimento do NFA do que o DFA.

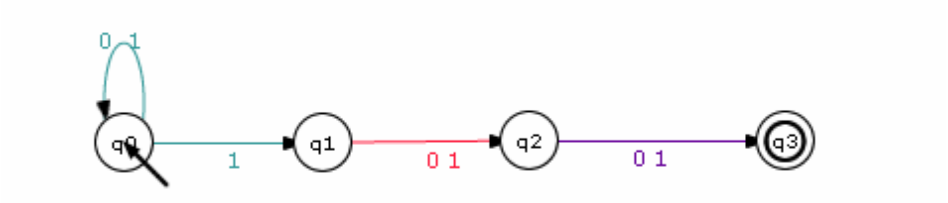

Figura 2.2 - Autômato Finito Não-Determinístico.

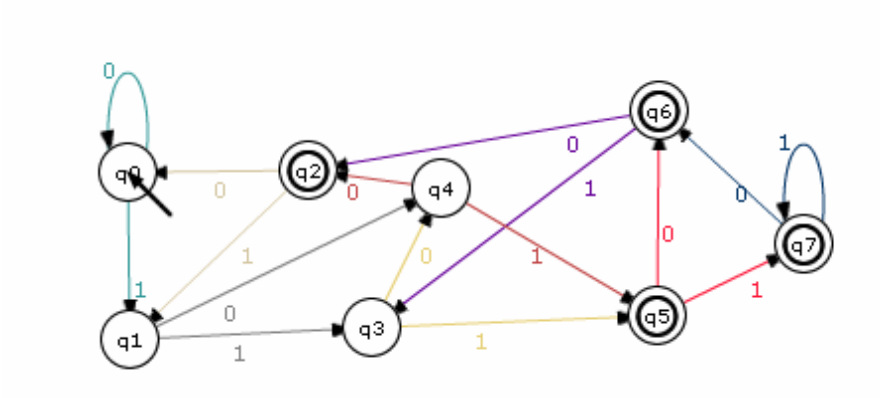

Figura 2.3 - Autômato Finito Determinístico.

O formalização de um NFA é muito semelhante a de um DFA . Sendo que o NFA também é composto por uma 5-upla (Q,Σ,δ,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;

E a definição formal do NFA representado graficamente na Figura 2.2 é:

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

Onde:

Q = {q0, q1, q2, q3}

Σ = {0, 1}

$\delta \Rightarrow$ δ (q0,0)={q0}, δ (q0,1)={q0,q1}, δ (q0, ε)= ø,
δ (q1,0)={q2}, δ (q1,1)={q2}, δ (q1, ε)= ø,
δ (q2,0)={q3}, δ (q2,1)={q3}, δ (q2, ε)= ø,
δ (q3,0)= ø, δ (q3,1)= ø, δ (q3, ε)= ø,

q_0 = q0

F = {q3}

Segundo HOPCROFT(2001), apesar desta simplificação na generalização de problemas, o não-determinismo não possibilita a definição de linguagens que não possam ser definidas por um DFA. Sendo que para qualquer NFA existe um DFA correspondente que é capaz de reconhecer a mesma linguagem que o NFA.

2.1.3. Expressões Regulares

Outra notação utilizada para definir linguagens regulares é a expressão regular. Expressões regulares estão diretamente ligadas a autômatos finitos sendo que muitas vezes são utilizadas como notação de autômatos em softwares.

Os operadores de expressões regulares são:

União representada pelo símbolo "$\cup$" faz a união de duas linguagens, por exemplo, utilizando as linguagens X e Y onde X={0, 1} e Y={ ε , 001} o resultado da operação $X \cup Y$ é { ε, 0, 1, 001}.

Concatenação é uma operação efetuada entre duas linguagens que pode ser representada por X.Y sendo que o símbolo da concatenação "." pode ser omitido. A concatenação das linguagens X e Y onde X={0, 1} e Y={ε , 001} tem como resultado {ε, 0, 1, 001, 0001, 1001}.

Fechamento, que também é chamado de estrela, é uma operação efetuada a uma linguagem a qual consiste na repetição de n vezes das palavras contidas na linguagem. Por exemplo, a linguagem X={0,1} seu fechamento representado por X* é um conjunto infinito que contem repetições das palavras de X, então X*={ ε, 0, 1, 00, 01, 10, 11, 001, 010, ... }. A palavra vazia é aceita por ser uma repetição de zero vezes de qualquer palavra da linguagem X. É comum encontrar a notação X^+, o que assemelha a X*, porém o número de repetições deve ser maior que zero vezes, desta forma ε não será parte do conjunto resposta.

A representação do NFA da figura 2.2 em uma expressão regular é $\Sigma^*1(0 \cup 1)(0 \cup 1)$. A precedência dos operadores é estrela, concatenação e união e caso seja necessário a execução de alguma operação de ordem diferente a essa deve-se utilizar parênteses.

2.1.4. Gramáticas Regulares

Além dos Autômatos finitos e as expressões regulares existe outro formalismo que possibilita a representação de linguagens regulares. Este formalismo é conhecido como Gramática Linear. Uma Gramática é uma quádrupla ordenada G = (V, T, P, S) onde:

V é o conjunto finito de símbolos variáveis ou não-terminais;

T é o conjunto finito de símbolos terminais disjunto de V;

P conjunto finito de pares, denominados regras de produção tal que a primeira componente é uma variável pertencente a V em seguida temos o símbolo da produção → e em seguida o corpo da produção que é a união de variáveis e terminais da seguinte forma $(V \cup T)^*$;

S é um elemento de V denominado variável inicial.

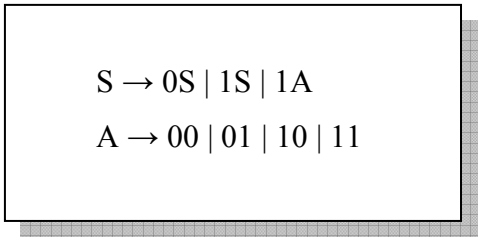

Figura 2.4 - Gramática Linear à direita.

Como LEWIS (1981) afirma, uma Gramática Regular é qualquer Gramática Linear. E uma gramática pode ser linear à direita ou linear à esquerda. No caso da gramática da figura 2.4 ela é linear à esquerda e ela corresponde ao NFA da figura 2.2.

A representação formal da gramática da figura 2.4 é G = (V, T, P, S) onde:

V ={S, A}

T = {0, 1}

P = { S→0S, S→1S, S→1A, A→00, A→01, A→10, A→11}

S = S

2.2. Linguagens Livres de Contexto

As linguagens regulares são muito úteis para a resolução de vários problemas, porém chega a certo ponto de complexidade o qual não é mais possível generalizar problemas em forma de NFA ou DFA devido algumas limitações na forma da computação destes autômatos. Como por exemplo, a criação de uma linguagem em que todas suas palavras são palíndromos.

Para resolver problemas como este é que utilizamos linguagens livres de contexto, que é uma classe de linguagem mais ampla que a das linguagens regulares e que utiliza como notações as gramáticas livres de contexto e os autômatos com pilha.

2.2.1. Gramáticas Livres de Contexto

Como LEWIS (1981) afirma uma Gramática Livre de Contexto é uma Gramática de Duplo balanceamento. E uma gramática que é ao mesmo tempo linear à direita e linear à esquerda.

Para representarmos gramáticas livres de contexto utilizamos à mesma formalização das gramáticas regulares, ou seja, ela é composta por uma quádrupla ordenada G = (V, T, P, S) onde:

V é o conjunto finito de símbolos variáveis ou não-terminais;

T é o conjunto finito de símbolos terminais disjunto de V;

P conjunto finito de pares, denominados regras de produção tal que a primeira componente é uma variável pertencente a V em seguida temos o símbolo da produção → e em seguida o corpo da produção que é a união de variáveis e terminais da seguinte forma $(V \cup T)^*$;

S é um elemento de V denominado variável inicial.

$$S \rightarrow 0S0 \mid 1S1 \mid \$$$

Figura 2.5 - Gramática livre de contexto.

A figura 2.5 representa a gramática que reconhece a linguagem dos palíndromos, ou seja, {w | w seja um palíndromo}. Graficamente será utilizado o símbolo cifrão ($) para representar transições vazias (ε).

As gramáticas livres de contexto são amplamente utilizadas na computação, como por exemplo, na descrição de linguagens de programação na forma de um analisador sintático.

Para utilização das linguagens livres de contexto muitas vezes é necessário à simplificação da mesma para que ela possa ser trabalhada de forma mais ampla. Existem várias formas de simplificações gramaticais uma das mais utilizadas é a Forma Normal de Chomsky – FNC.

Uma gramática G está na FNC se todas suas produções estão entre umas das formas abaixo:

$A \rightarrow BC$

$A \rightarrow a$

$S \rightarrow \$$

De forma que toda variável produza outras duas variáveis ou um terminal, caso seja a variável inicial pode gerar o símbolo vazio. G também não pode ter nenhum símbolo inútil.

De acordo com HOPCROFT(2001), qualquer linguagem livre de contexto pode ser gerada por uma gramática livre de contexto na Forma Normal de Chomsky. E se G é uma gramática livre de contexto na Forma Normal de Chomsky, então, para qualquer palavra w pertencente a linguagem de G que tenha comprimento n>=1, são requeridos exatamente 2n-1 passos para qualquer derivação de w.

2.2.2. Autômatos com pilha

É possível definir uma linguagem livre de contexto a partir de um autômato com pilha. O autômato com pilha assemelha-se a um autômato finito porem ele possui uma pilha na qual ele pode ler, escrever e remover sempre o último elemento inserido da forma que o protocolo de pilha funciona.

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;

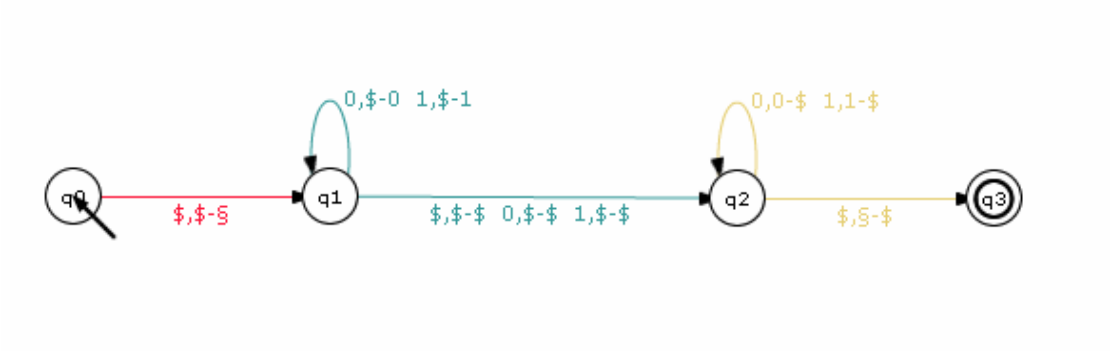

Figura 2.6 - Autômato com Pilha.

A Figura 2.6 mostra graficamente um autômato com pilha que reconhece a linguagem dos palíndromos. O símbolo cifrão ($) representa o símbolo vazio (ε). O símbolo § foi utilizado para verificar o fim da pilha. Os três valores que se encontra em cada transição correspondem respectivamente ao valor que é lido da palavra de entrada, o valor que é removido da pilha e o valor a ser escrito na pilha. Formalmente o autômato da Figura 2.6 dever ser representado da seguinte forma:

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

Q = {q0, q1, q2, q3}

Σ = {0, 1}

Γ = {0, 1, §}

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

δ (q1,ε,ε)= (q2,ε), δ (q1,0,ε)= (q2,ε), δ (q1,1,ε)= (q2,ε),

δ (q2,0,0)= (q2,ε), δ (q2,1,1)= (q2,ε), δ (q2,ε,§)= (q2,ε)

q_0 = q0

F = {q3}

3. INFORMÁTICA NA EDUCAÇÃO E TECNOLOGIAS UTILIZADAS

Neste capítulo, serão apresentados alguns conceitos relacionados a informática na educação e as tecnologias utilizadas para o desenvolvimento do projeto.

3.1. Informática na Educação

A introdução do computador na educação tem provocado uma verdadeira revolução na nossa concepção de ensino e de aprendizagem (Carlos, 2004). As primeiras tentativas eram de fazer o computador imitar a atividade que acontece em sala de aula, e com o aprimorar das tecnologias outras modalidades de uso do computador na educação vem se desenvolvendo.

Como VALENTE(1993) afirma em 1924, o Dr. Sidney Pressey usou da idéia de ensino através da computação na elaboração de uma máquina para corrigir teste de múltipla escolha. Mais tarde, no início da década de 50, B. F. Skinner elaborou uma máquina para ensinar usando o conceito de instrução programada.

A instrução programada consiste em dividir o material a ser ensinado em pequenos segmentos seqüenciais denominados módulos. A instrução programada foi muito usada no final de 1950 e inicio dos anos 60. Entretanto, esta idéia não popularizou pela dificuldade da produção do material instrucional e da falta de padrão dos materiais existentes.

No inicio dos anos 60, notou-se a possibilidade de se implementar os módulos do material instrucional no computador. E nesse momento surgia a instrução auxiliada por computador ou software educativo.

Softwares educativos são softwares desenhados para facilitar o ensino e o aprendizado. De acordo com GIRAFFA(1999) todo programa que utiliza uma metodologia que o contextualize no processo ensino e aprendizagem, pode ser considerado educacional.

VALENTE(1993) afirma que os softwares educativos podem ser classificados de acordo com a sua função. Assim os softwares educativos podem ser dos seguintes tipos:

- tutoriais,
- de exercitação e prática,
- simuladores,
- jogos educacionais;

Os softwares educativos tutoriais possuem o conteúdo a ser ensinado predefinido em seu escopo numa estrutura parecida com a utilizada em uma sala de aula. Já os de exercitação e prática são utilizados para revisão e memorização de algum assunto já estudado pelo aluno. Os simuladores são utilizados na construção de situações que se assemelham com a realidade e enfatizam a exploração autodirigida envolvendo a criação de modelos dinâmicos e simplificados do mundo real, dentro do contexto abordado, oferecendo ainda a possibilidade de o aluno desenvolver hipóteses, testá-las, analisar resultados e refinar conceitos. E os jogos educacionais se assemelham aos simuladores, porém são desenvolvidos de forma que entreterem o aluno.

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.

É importante salientar que existem outras formas de classificar softwares educativos como, por exemplo, TAYLOR(1980) classifica os softwares educativos em tutor, tutelado e ferramenta. O software tutor assemelhasse ao descrito anteriormente onde o computador ensina o aluno, já o software tutelado é o contrario, o aluno é quem ensina o computador como nas linguagens de programação e as ferramentas são onde o aluno pode manipular as informações.

3.2. Web 2.0

O termo Web 2.0 foi criado por Dale Dougherty inicialmente como nome dado a uma série de conferências sobre o tema, que foi chamada de web 2.0 conference que aconteceu em Outubro de 2004 e foi realizada pela O'Reilly Media e pela MediaLive International.

A Web 2.0 é um paradigma muito recente cuja principal característica é a utilização da Internet como plataforma para aplicações e não apenas como meio de entrega expressa de conteúdo (plataforma Web 1.0).

Como afirma O'REILLY(2005) Este paradigma é baseado em duas qualidades fundamentais: simplicidade e conteúdo. A simplicidade está na tentativa de tornar a navegação na internet fácil e intuitiva ao usuário comum, evitando gráficos pesados e tabelas desnecessárias, facilitando ao máximo o acesso ao conteúdo. Por outro lado, o conteúdo não é baseado somente no que é disponibilizado pelo webwriter[1], mas há

[1] São redatores que trabalham especificamente na área de desenvolvimento de matérias e textos para internet.

colaboração dos próprios usuários, de forma parcial (uma parte é do webwriter e outra dos usuários) ou total. Por exemplo, a enciclopédia gratuita Wikipedia segue este modelo.

A partir destas características iniciais, sugiram outras características como formação de comunidades, aplicações ricas e uso do conteúdo em aplicações externas.

A formação de comunidade dá-se na identificação de interesses comuns, na disponibilização ou busca de conteúdo. Quanto mais usuários, mais conteúdo produzido e, com isso, a qualidade do serviço aumenta.

Nas aplicações ricas, deixa-se de desenvolver "páginas" para se desenvolver aplicações. Com o avanço das tecnologias hoje é possível disponibilizar aplicações de interface mais robusta e ambiente mais parecido com aplicativos de desktop, com alto nível de interação entre o usuário e a aplicação.

O uso do conteúdo em aplicações externas, como as tecnologias XML, RSS, API, é muito importante para deixar a aplicação mais independente e mais portável.

3.3. Java

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. Dentre as linguagens existentes hoje foi escolhida a linguagem java uma vez que ela possui essas características e ainda vários outros benefícios que serão citados nesta seção do trabalho.

A linguagem Java foi desenvolvida na década de 90 pela empresa Sun Microsystems, e a equipe de programadores foi chefiada por James Gosling. Ela e 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 máquina virtual Java. Desta forma, 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 várias bibliotecas ou pacotes como também são conhecidas forma 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. Entre as API's utilizadas para o desenvolvimento da ferramenta podemos citar o JDOM que encapsula o trabalho de arquivos XML. E como

MCLAUGHLIN (2001) afirma, JDOM acessa os dados XML através da estrutura de uma árvore tornando sua utilização bem intuitiva.

Existem também os servelets Java que é tecnologia que veio para substituir o uso de CGI's - Common Gateway Interface. Os servelets são programas que rodam no servidor web entre a requisição do cliente HTTP e a base da aplicação no servidor HTTP. Na ferrmenta desenvolvida não será utilizado os serveletes diretamente porem será utilizado o JSP que utiliza dos servelets.

E finalmente temos o JSP que é a tecnologia Java para desenvolvimento de aplicações Java. Como ROCKWELL (2001) diz, JSP que significa JavaServer Page, ele mistura o HTML estático com a dinamicidade de conteúdo gerado pelos servelets possibilitando criar documentos dinamicamente. Quando um arquivo JSP é executado pela primeira vez ele é transformado em um programa Java (Servlet).

Desta forma existem muitos benefícios para se utilizar JSP para construção de documentos dinamicamente. Pois JSP é escrito em Java, então ele possui orientação a objetos podendo importar bibliotecas Java, ele também é independente de plataforma e possui todas as qualidades da linguagem Java.

3.4. OpenLaszlo

OpenLaszlo uma plataforma de desenvolvimento que utiliza a linguagem de programação LZX. Esta é uma plataforma de código aberto desenvolvida pela Laszlo Systems Inc e possibilita a criação de aplicativos web de forma dinâmica muito semelhante a aplicativos desktop.

A linguagem LZX foi desenvolvida de maneira a utilizar sintaxes e convenções de nomes familiares a desenvolvedores web com intuito de facilitar o aprendizado, porém introduz novos conceitos e capacidades que possibilitam interfaces com usuário mais agradáveis e interativas.

Em LZX, tags XML são usadas para criar objetos JavaScript, e JavaScript usado dentro de programas LZX têm função de manipular objetos criados pelas tags. Na maioria dos casos tudo que pode ser feito pelas tags XML pode ser feito usando JavaScript e vice-versa. Mas essa equivalência não é total, existem funcionalidades que só podem ser desenvolvidas com JavaScript e outras que só são possíveis com tags XML. A LZX, exatamente, une as sintaxes do XML e JavaScript.

A disponibilização de aplicações em OpenLaszlo podem ser feitas de duas formas. A primeira é onde o código é pré-compilado pelo compilador do OpenLaszlo de forma que são gerado arquivos binários que podem ser disponibilizados em qualquer servidor web. A outra forma é a que a aplicação é intermediada pelo servidor, a aplicação fica em um servidor Java J2EE e o código é compilado pelo servidor e enviado para ser executado no cliente e toda comunicação cliente e outros servidores é gerenciada pelo servidor onde a está aplicação de forma que a manipulação dos dados é feita pelo servidor.

Para que o cliente poder executar aplicação basta ele possuir o plugin Macromedia Flash Player 5 ou superior. Desta forma a execução da aplicação fica independente do sistema operacional do cliente e do browser utilizado podendo ser executado até em aparelhos móveis que possua suporte ao plugin Macromedia Flash Player.

3.5. XML

Antes da existência do XML o World Wide Web Consortium, W3C, que foi fundado em 1994 de acordo com Deitel et al. (2003), já tentava criar alguns protocolos comuns para Web entre os protocolos existia o SGML (Standard Generalized Markup Language) e o HTML(HyperText Markup Language).

Devido algumas limitações que o SGML e o HTML possuíam, foi desenvolvido um comitê para revisá-los a fim de melhorá-los. E conforme BRAY, PAOLI & MALER(2006) foi este comitê que criou o XML em 1996 a partir da revisão destas tecnologias.

Os principais problemas do SGML eram sua complexidade e seu tamanho que inviabilizavam sua utilização na Web. Já o HTML não possui uma separação exata dos dados estruturais, semânticos e de formatação além de não ser muito rígido com suas próprias regras sintáticas.

XML é uma abreviação de eXtensible Markup Language. RAY (2001) afirma que o XML é uma forma flexível de criar dados que se auto descrevem e de disponibilizá-los na web.

Segundo RAMALHO (2002), o XML tem como finalidade marcar um determinado texto que sofrerá algum tipo de processamento. Em outras palavras, pode-se dizer que a XML é uma linguagem para criar dados estruturados baseados em um texto. A estruturação dos dados deve-se ao fato de que ela permite identificar de forma inequívoca uma peça individual de informação.

HAROLD(2004) afirma que este padrão de publicação que é o XML é formado por um documento que possui tags, que são palavras encapsuladas por sinais “<” e “>”, que descrevem os dados da mesma forma que outras linguagens de marcação como HTML e SGML, sendo que ele pode ser utilizado na representação de estruturas de dados estruturados e semi-estruturados.

Em nenhum momento o XML fornece dados para a sua formatação. Para que ele seja trabalhado de qualquer forma é necessária uma aplicação que seja capaz de lê-lo, interpretá-lo e apresentá-lo da melhor forma para o usuário. Isso mostra que muitas pessoas estão enganadas ao acreditarem que XML é um HTML evoluído com novos recursos.

Atualmente o XML já é um formato padrão para transferência de dados entre computadores. De acordo com DEITEL et al. (2003), diversos domínios de aplicação como comércio eletrônico e cadastro bibliográfico, vêm definindo protocolos XML para os seus dados e cada vez mais aplicações e pessoas disponibilizam e manipulam informações XML na Web. O uso cada vez mais intensivo de XML vem despertando o interesse de diversas áreas da ciência da computação, como Linguagens de Programação, Engenharia de Software e Bancos de Dados, todas preocupadas com a definição de mecanismos para o tratamento de dados neste formato.

BRAY (2006) afirma que a flexibilidade do XML provém da possibilidade de transportar qualquer tipo de dados, mantendo-os estruturalmente coesos e inteligíveis. Devido a esta estrutura, é também possível combinar em um mesmo documento, vários objetos com tipos de dados diferentes.

Devido a essas características, o XML foi escolhido para fazer a troca de dados entre o LFAScholar e a ferramenta desenvolvida de forma que qualquer pessoa que necessite utilizar a ferramenta apenas precisa entender qual o formato de XML utilizado para poder utilizá-la. Na Figura 3.1 pode-se observar um XML utilizado na ferramenta para representar um NFA sendo que qualquer pessoal que conheça um NFA é capaz de compreendê-lo.

```
- <automato tipo="nfa">
  - <informacoes>
      <data>25/05/2008</data>
      <autor>Straus</autor>
      <descricao>Exemplo de automato </descricao>
    </informacoes>
  - <alfabeto>
      <simbolo valor="0"/>
      <simbolo valor="1"/>
    </alfabeto>
  - <estados>
    - <estado rotulo="q0">
      - <informacoes>
          <corTransicao valor="0x660099"/>
          <coordenadas x="200" y="150"/>
          <tipo valor="inicial"/>
        </informacoes>
      - <transicoes>
        - <transicao rotulo="t1" destino="q0">
          - <simbolos>
              <simbolo valor="0"/>
            </simbolos>
          </transicao>
        - <transicao rotulo="t2" destino="q1">
          - <simbolos>
              <simbolo valor="1"/>
            </simbolos>
          </transicao>
        </transicoes>
      </estado>
    - <estado rotulo="q1">
      - <informacoes>
          <corTransicao valor="0x339999"/>
          <coordenadas x="350" y="150"/>
          <tipo valor="final"/>
        </informacoes>
      - <transicoes>
        - <transicao rotulo="t1" destino="q0">
          - <simbolos>
              <simbolo valor="0"/>
            </simbolos>
          </transicao>
        - <transicao rotulo="t2" destino="q1">
          - <simbolos>
              <simbolo valor="1"/>
            </simbolos>
          </transicao>
        </transicoes>
      </estado>
    </estados>
  </automato>
```

Figura 3.1 - Exemplo de XML que representa um NFA

4. METODOLOGIA

Neste capitulo será apresentada o tipo de pesquisa feita neste trabalho e forma em que foi feita a construção o LFAScholar a da LFALibrary.

4.1. Natureza da Pesquisa

O presente trabalho tem como base desenvolver uma ferramenta capaz de ser acoplada ao ambiente de ensino LFAScholar, portanto esta pesquisa é do tipo tecnológica, uma vez que se utiliza de técnicas existentes em engenharia de software para o desenvolvimento desta ferramenta que será responsável pela simulação, validação e conversão de autômatos e gramáticas no LFAScholar.

Quanto ao objetivo, esta é uma pesquisa exploratória, pois sua finalidade é a produção de um software e como JACOBSON (1992) diz, é a partir de experimentações exploratórias que se produzem inovações tecnológicas.

O procedimento utilizado para desenvolver a presente pesquisa foi pesquisa experimental, uma vez que foi utilizado bastante o empirismo nas tomadas de decisões relativas ao projeto.

Sendo que a pesquisa é Bibliográfica envolvendo, basicamente, consultas a livros de referências e a artigos científicos.

4.2. Materiais

A plataforma de desenvolvimento da ferramenta foi um microcomputador com processador AMD Sempron(tm) 2600+ de1,60GHz, 512 MB de RAM e placa de vídeo nVidia® GeForceTM Series com 256 MB de RAM, o sistema operacional Microsoft® Windows® XP Professional.

A plataforma do servidor onde o sistema está disponível é um microcomputador com processador AMD Athlon(tm) XP 1500+, 128 MB de RAM e o sistema operacional Ubuntu Linux 7.04. Para que o servidor funcione corretamente foi instalado o Apache Tomcat 5.0.24, o OpenLaszlo 3.3.3 e o J2SE(Java 2 Standard Edition) 5.0. Ele está ligado diretamente à rede do departamento de ciência da computação da Universidade Federal de Lavras – UFLA e pode ser acessado no endereço mathematica.dcc.ufla.br .

A principal ferramenta de desenvolvimento foi o Integrated Development Environment (IDE) NetBeans 6.0.1 que já vem integrado ao J2SE. O NetBeans foi utilizado tanto para desenvolver a biblioteca Java que faz todas conversões, simulações e validações de autômatos e gramáticas, quanto para a criação dos arquivos JSP que serão responsáveis pela integração do LFAScholar com a Biblioteca Java.

A utilização do LFAScholar foi testada em várias máquinas, sendo que os requisitos mínimos das máquinas clientes são: possuir um web browser com o plug-in do Flash Player e uma conexão à internet. A resolução da tela ideal para sua visualização é de 1024x768 pixels.

4.3. Desenvolvimento

Inicialmente foi desenvolvida uma pesquisa bibliográfica sobre os assuntos tratados anteriormente, de forma que essa pesquisa embasasse as decisões tomadas durante o desenvolvimento do projeto. Nesta etapa todas as ferramentas utilizadas no projeto foram estudadas.

Com este embasamento teórico foi possível definir como seria a arquitetura do sistema e como a comunicação será feita. A Figura 4.1 exemplifica essa arquitetura.

À partir deste momento foi possível iniciar o desenvolvimento, pois toda teoria já tinha sido levantada e a forma em que o ambiente seria desenvolvido já tinha sido decidida. Assim, foi criada a LFALibrary que é uma biblioteca Java, uma camada de comunicação em JSP e a interface em OpenLaszlo que já existia foi melhorada. E a divisão em camadas lavando em consideração a camada de interface, camada de negócios e camada de persistência também foi decidia e a camada de interface seria composta pela interface em OpenLazlo, a camada de comunicação e a LFALibrary compõem o camada de negócios sendo que o ambiente não possui a camada de persistência uma vez que nenhum dado do ambiente será armazenado.

A LFALibrary é a parte responsável por todo o processamento dos autômatos e gramáticas, ela é que faz as validações, conversões e simulações. Ela foi desenvolvida em Java e possui duas classes principais que são autômatoFinito e gramática. A classe autômatoFinito foi modelada de forma a representar o autômato da forma em que ele foi descrito acima, ou seja, um conjunto de cinco elementos $(Q, \Sigma, \delta, q_0, F)$. Da mesma forma, a classe gramática foi modelada para representar a gramática com todos seus quatro elementos que são (V, T, P, S). O construtor de ambas as classes recebem como parâmetro

uma string XML que representa o autômato ou a gramática. Já no caso da simulação de um DFA, por exemplo, a função recebe como entrada a palavra e retorna uma string XML com o caminho que é percorrido para que esta palavra seja aceita. Todos as classes da LFALibrary foram devidamente comentadas a fim de que outras pessoas possam utilizar desta biblioteca e que ela possa ser melhorada com o passar do tempo.

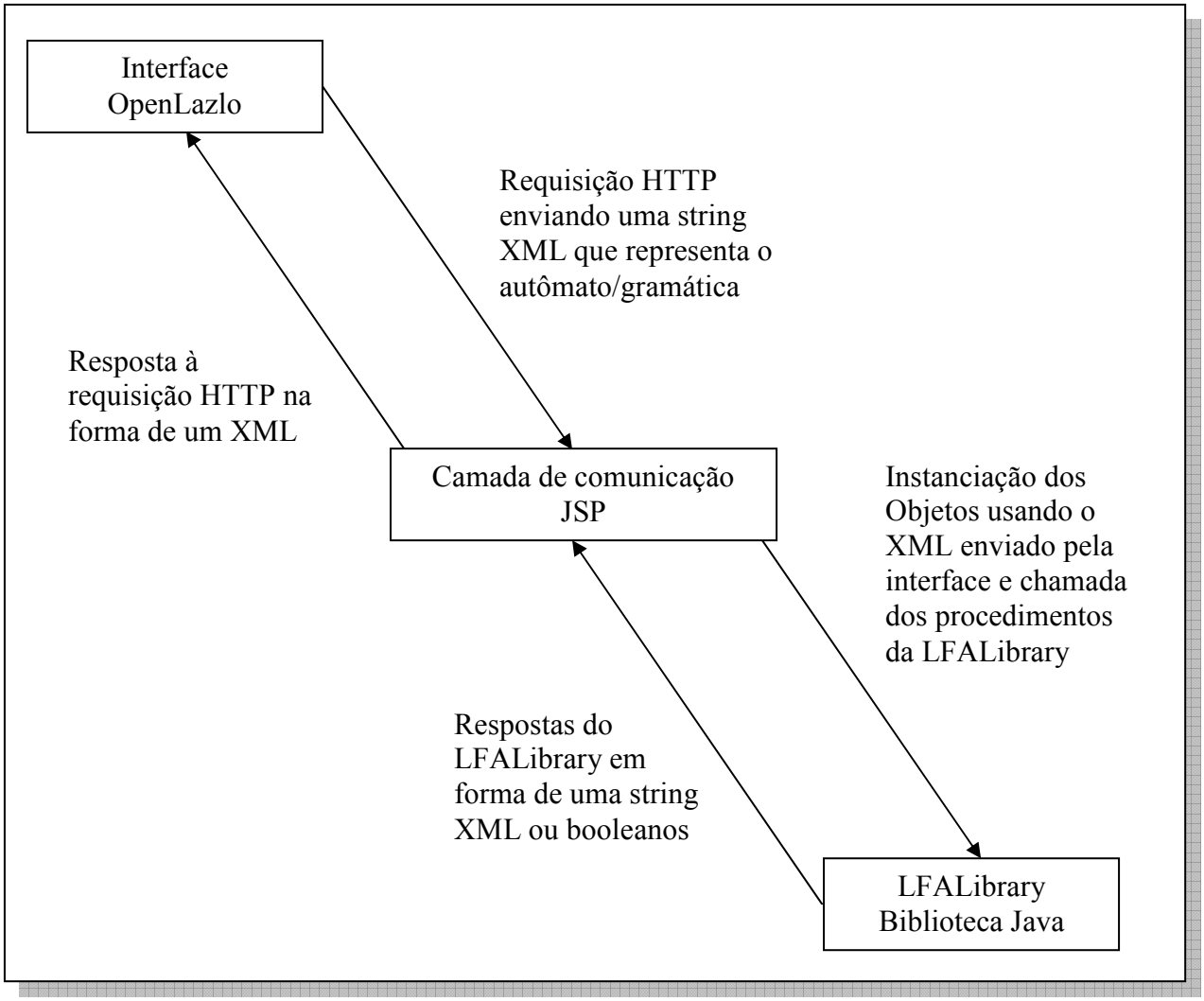

Figura 4.1 - Arquitetura do sistema.

A parte desenvolvida em JSP é responsável por fazer a ligação entre a interface em OpenLaszlo e a LFALibrary. Desta forma, ela e acionada através de uma requisição HTTP que tem como parâmetro uma string XML que representa o autômato ou a gramática que ela irá trabalhar e pode conter também outros parâmetros como a palavra que é para ser simulada no autômato. Este arquivo JSP irá instanciar os objetos necessários na

LFALibrary e usar suas funções de forma que seja criada a resposta que devera ser apresentada ao LFAScholar. E finalmente o JSP responde na forma de um arquivo XML que será interpretado pela interface.

A interface do LFAScholar, por sua vez, será responsável por fazer a requisição no momento adequado enviando os parâmetros necessários e interpretar o arquivo XML de resposta mostrando ao usuário a resposta de forma mais simples possível.

Findado o desenvolvimento do ambiente, ele foi disponibilizado para testes no seguinte endereço mathematica.dcc.ufla.br e o resultado do desenvolvimento desta ferramenta será apresentado no próximo capítulo.

5. RESULTADOS E DISCUSSÃO

Este Capítulo tem por objetivo apresentar de forma sucinta o LFAScholar e os principais algoritmos desenvolvidos. Para isso, serão apresentadas imagens de cada elemento importante que compõe a interface do LFAScholar além de alguns pseudocódigos dos algoritmos desenvolvidos na LFALibrary.

De acordo com a classificação dos softwares educativos o LFAScholar é um software de simulação e sua interface é composta por dois módulos principais que são os:

- Editor de Diagramas
- Editor de Gramáticas

Com Editor de Diagramas é possível criar autômatos finitos de forma gráfica utilizando recursos de arrastar e soltar. Para criar um estado de um autômato, basta arrastar a figura do estado na barra de ferramentas da janela do editor de diagramas e soltar a mesma no local em se deseja criar o estado. Para deletar ou configura um estado, basta selecioná-lo com um clique e depois clicar no respectivo botão na barra de ferramentas da janela do editor de diagramas. Para criar uma transição entre dois estados, deve-se selecionar o estado que será a origem da transição, clicar no símbolo da seta na barra de ferramentas da janela do editor de diagramas e em seguida clicar no estado destino da transição. Para criar transições vazias basta utilizar como símbolo da transição o cifrão "$". Para se configurar uma transição, basta clicar no seu símbolo que será aberta uma janela de configuração. As opções de simular, validar e converter NFA para DFA e converter autômato finito para gramática estão nos menus do editor de diagramas.

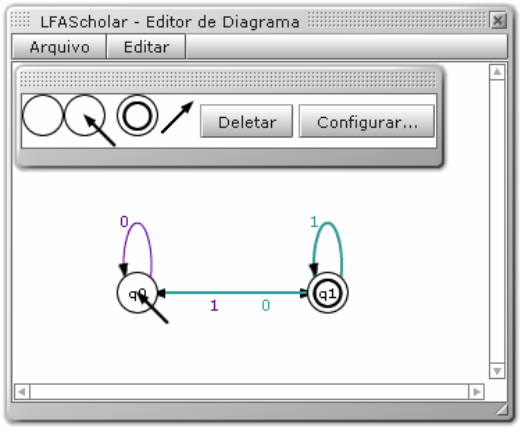

Figura 5.1 - Editor de Diagramas.

No Editor de Gramáticas é possível descrever gramáticas na forma de um texto simples onde cada linha representa as regras de produção de uma variável. O primeiro símbolo da linha é a variável a qual suas regras de produção serão representadas, o símbolo maior “>” representa o símbolo da produção, ou seja, separa a variável das suas produções e em seguida as produções são apresentadas separadas pela barra vertical “|”. Nos menus da janela do editor de gramáticas se encontram as opções para validar a gramática, transformá-la em um autômato finito e transformá-la para forma normal de Chomsky.

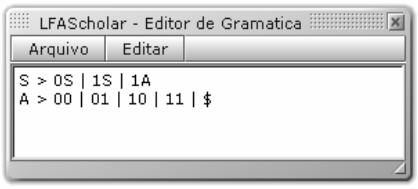

Figura 5.2 - Editor de Gramáticas.

5.1. Validar um Autômato Finito Determinístico

A operação de validação de autômato consiste em verificar se ele é autômato finito determinístico ou não. Para que ele seja considerado determinístico, deve existir uma e apenas uma transição de saída para cada símbolo do alfabeto em cada estado do autômato.

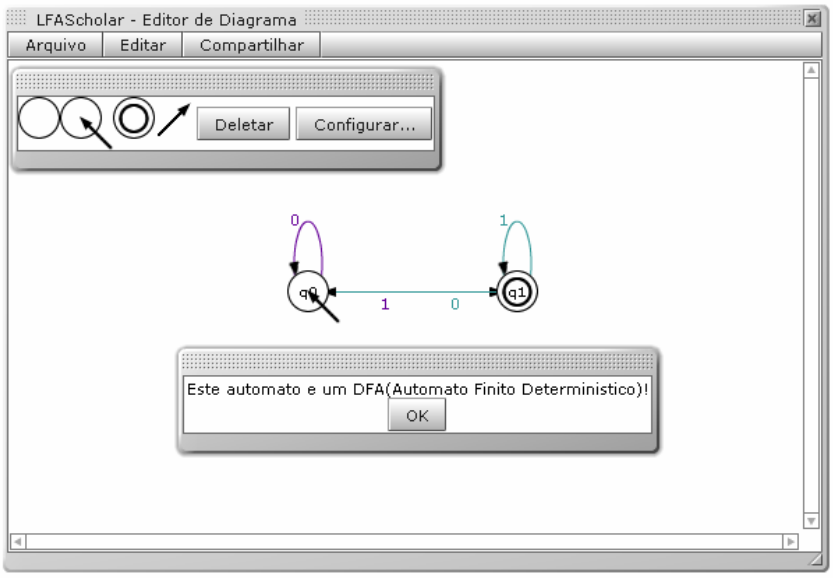

Figura 5.3 - Tela de validação de um DFA

5.2. Validar Gramática regular (linear à direita ou esquerda)

Para que uma gramática linear seja validada, é necessário verificar se ela é linear à direita ou à esquerda, ou seja, todas as regras da gramática são verificadas de forma que elas podem gerar apenas um não terminal à direita se ela for linear a direita, ou à esquerda, caso ela seja linear à esquerda.

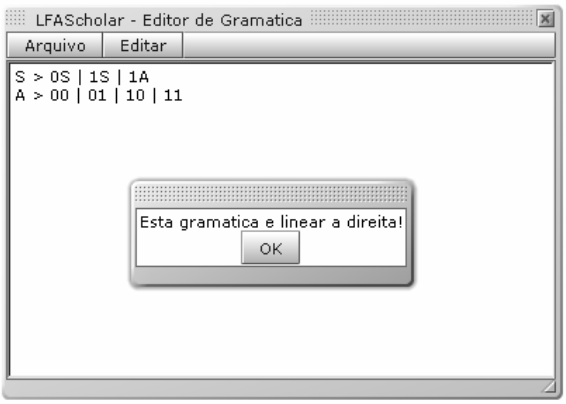

Figura 5.4 - Tela de Validação de uma Gramática Linear.

5.3. Transformar um Autômato Finito Não-Determinístico para um Autômato Finito Determinístico

Como foi apresentado anteriormente, todo NFA possui um NFA correspondente. Para ser efetuada essa transformação, é montado o conjunto potência de todos os estado do NFA, estes conjuntos serão os estados do DFA e suas transições serão criadas de acordo com as transições do NFA.

```
DFA.estados = conjunto potência NFA.estados
DFA.estadoInicial = estados alcançáveis do estado inicial do NFA com o símbolo $;
para cada estado do DFA{
  se estado DFA contém estado final do NFA {
    DFA.adicionaEstadoFinal(estado DFA);
  }
  se é o estado vazio{
    para cada símbolo do alfabeto{
      estadoOrigem = DFA.estado;
      estadoDestino = DFA.estado;
      valor = símbolo;
      NFA.adicionaTransição(estadoOrigem, estadoDestino, valor);
    }
  } senão {
    para cada estado do NFA pertencente aos estados do DFA{
      para cada símbolo do alfabeto{
        estadoOrigem = DFA.estado;
        estadoDestino = estados alcançáveis do estado do NFA com o símbolo ou $;
        valor = símbolo;
        NFA.adicionaTransição(estadoOrigem, estadoDestino, valor);
      }
    }
  }
}
```

Figura 5.5 - Pseudocódigo da conversão de NFA para DFA.

5.4. Transformar uma gramática regular em autômato finito

A operação de transformar uma gramática regular em um autômato finito é descrita em pseudocódigo na figura 5.6. Esta função recebe uma gramática regular e retorna o autômato finito equivalente a essa gramática. A operação cria um estado no autômato correspondente a cada variável da gramática e cria as transições do autômato de acordo com as regras de produção da gramática.

```
inteiro i = 0;
para cada variavel faça {
  se é variável inicial {
    automato.adicionaEstadoInicial(variável.nome);
  } senão {
    se variável possui produção vazia ou produção de terminais {
      automato.adicionaEstadoFinal(variável.nome);
    } senão {
      automato.adicionaEstado(variável.nome)
    }
  }
  para cada produção da variável faça{
    para cada símbolo da produção faça{
      i = i + 1;
      automato.adicionaSimbolo(simbolo);
      se próximo símbolo da produção é uma variável{
        string estadoOrigem = concatena("q",i-1);
        string estadoDestino = próximo símbolo da produção;
        string valor = símbolo;
        automato.adicionaTransição(estadoOrigem, estadoDestino, valor);
      } senão {
        se símbolo é terminal {
          string estadoOrigem = concatena("q",i-1);
          string estadoDestino = concatena("q",i);
          string valor = símbolo;
          automato.adicionaEstado(estadoDestino);
          automato.adicionaTransição(estadoOrigem, estadoDestino, valor);
        }
      }
    }
  }
```

Figura 5.6 - Pseudocódigo da conversão de gramática linear à direita para DFA.

5.5. Transformar um autômato finito em gramática regular

Esta função consiste em construir uma gramática linear à direita a partir de um autômato finito. A operação de transformar uma gramática regular em um autômato finito cria um estado no autômato correspondente a cada variável da gramática e cria as transições do autômato de acordo com as regras de produção da gramática. Na figura 5.7 encontra-se o pseudocódigo utilizado nesta transformação.

```
para cada estado faça {
  gramática.adicionaVariavel(estado.nome);
  se é estado inicial {
    gramática.alteraVariavelInicial(estado.nome);
  }
  se é estado final {
    gramática.adicionaProdução(estado.nome, $);
  }
  para cada transição do estado faça{
    string produção = concatena(transição.valor, transição.destino);
    gramática.adicionaTerminal(transição.valor);
    gramática.adicionaProdução(estado.nome, produção);
  }
}
```

Figura 5.7 - Pseudocódigo da conversão de DFA para gramática linear.

5.6. Transformar Gramática livre de contexto para a Forma Normal de Chomsky

Qualquer gramática livre de contexto pode ser transformada para forma normal de Chomsky. Para que ela seja transformada são utilizados quatro passos, são eles:

Passo 1: Adicionar a regra A → S, onde A é uma nova variável inicial e S é a variável inicial antiga.

Passo 2: Eliminar regras B → $ onde A não é o símbolo inicial A.

Passo 3: Eliminar regras do tipo A → B. Se A = B simplesmente removemos a regra, senão para cada regra B → u adicionamos A → u, a menos que essa regra já tenha sido removida.

Passo 4: Sobram as regras A → u1, u2, u3, ... uk, onde k ≥ 2 são substituídas por regras A → u1, A1

A1 → u2, A2

A2 → u3, A3

:

:

:

Ak-3 → uk-2, Ak-2

Ak-2 → uk-1, Ak-1

Desta forma a transformação da gramática S > 0S0 | 1S1 | $ para forma normal de Chomsky é apresentada passo a passo na figura 5.8.

****Primeiro Passo****
A > S
S > 0S0 | 1S1 | $

****Segundo Passo****
A > S | $
S > 0S0 | 1S1 | 00

****Terceiro Passo****
A > $ | 0S0 | 1S1 | 00
S > 0S0 | 1S1 | 00

****Quarto Passo****
A > $ | EE | EB | DC
S > EE | EB | DC
B > SE
C > SD
D > 1
E > 0

Figura 5.8 - Passos da transformação de gramática para FNC.

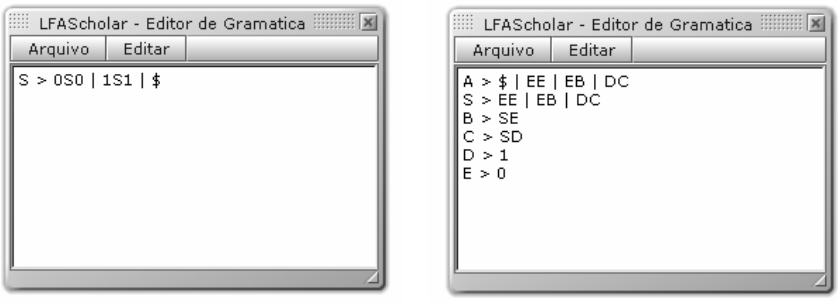

Figura 5.9 - Telas de execução da transformação da gramática para FNC.

5.7. Simulação de DFA

A simulação de um DFA consiste em apresentar o caminho em que uma palavra percorre no autômato e se ao final do percurso a palavra terminar em um estado final, ela é aceita.

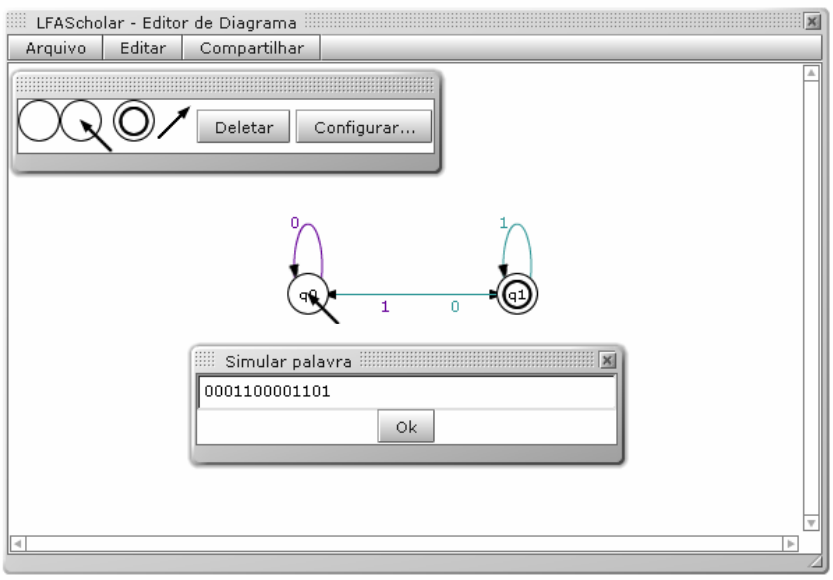

Figura 5.10 - Tela de simulação de um DFA.

Exemplo de execução da simulação do autômato da figura 5.10 com a palavra de entrada 0001100001101. O XML de retorno apresenta todo o caminho que foi percorrido pelo autômato e a interface do LFAScholar se encarrega de apresentar a simulação ao usuário.

```
- <resposta tipo="Palavra aceita">
  - <caminho>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="1" proximo="q2"/>
      <movimento atual="q2" simbolo="1" proximo="q2"/>
      <movimento atual="q2" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="1" proximo="q2"/>
      <movimento atual="q2" simbolo="1" proximo="q2"/>
      <movimento atual="q2" simbolo="0" proximo="q1"/>
      <movimento atual="q1" simbolo="1" proximo="q2"/>
    </caminho>
</resposta>
```

Figura 5.11 - XML com o caminho a ser percorrido na simulação do DFA.

6. CONCLUSÕES

Neste capítulo serão apresentadas as considerações finais do desenvolvimento do trabalho e também algumas idéias para trabalhos futuros.

6.1. Considerações Finais

O objetivo principal deste trabalho foi a disponibilização da versão funcional do LFAScholar assim como a adição de novas funcionalidades ao mesmo, além de corrigir alguns problemas que ele possuía. Assim, pode-se observar que o objetivo principal do trabalho foi alcançado com sucesso. Pois o ambiente encontra-se no ar e pode ser acessado pelo endereço mathematica.dcc.ufla.br. As funcionalidades de simulação, validação e conversão de gramáticas e autômatos foram adicionadas, o editor de gramáticas também foi desenvolvido e o editor de diagramas foi melhorado.

Outro ponto positivo do trabalho foi a criação da LFALibrary, pois com ela será possível que outras pessoas desenvolvam aplicativos que utilizem autômatos e gramáticas de forma mais simples necessitando apenas que elas obedeçam os padrões adotados nos arquivos de entrada de XML, sendo possível usar a LFALibrary sem precisar de XML pois seu código está todo comentado e existem funções específicas que manipulam os objetos sem a necessidade de XML.

6.2. Trabalhos Futuros

Como sugestões para trabalhos futuros, apresento algumas idéias que, se bem trabalhadas, podem contribuir bastante para o LFAScholar e a LFALibrary.

Uma dessas idéias é estender o funcionamento para autômatos com pilha e máquinas de turing. Desta forma, será possível utilizar este sistema não apenas na disciplina de LFA, mas também em aulas de teoria da computação e compiladores.

Outra idéia interessante é a padronização e a internacionalização dos arquivos XML no sentido de criar um padrão para a representação de autômatos e gramáticas no estilo do que foi feito com o MathML, que é um padrão para representação de problemas matemáticos. Desta forma, torna-se possível o intercâmbio entre dados de qualquer outro programa que trabalhe com autômatos e gramáticas que utilizem este padrão.

7. REFERÊNCIAS BIBLIOGRÁFICAS

BRAY, Tim. PAOLI, Jean. MALER, Eve: **Extensible Markup Language (XML)** 1.0 (Fourth Edition). Disponível em http://www.w3.org/TR/2006/REC-xml-20060816, atualizada em 2006, consultada em 30 maio 2008.

CARLISLE, David. ION, Patrick. MINER, Robert. POPPELIER, Nico.: **Mathematical Markup Language (MathML)** Version 2.0 (Second Edition). Disponível em http://www.w3.org/TR/2003/REC-MathML2-20031021/, atualizada em 2003, consultada em 30 maio 2008.

CARLOS, F. J.: **Metodologia para Pesquisa e Desenvolvimento.** Primeira Edição. 2004, 328 p.

DEITEL, H. M.; DEITEL, P. J.: **Java, Como Programar**, tradução de Carlso Arthur Lang Lisboa, Porto Alegre: Bookman, 2003.

DEITEL, H. M.; DEITEL, P. J.; NIETO, T. R.; LIN, T. M.; SADHU, P.. **XML, Como Programar**, tradução de Luiz Augusto Salgado e Edson Furmankiewicz, Porto Alegre: Bookman, 2003.

FINO, Carlos N.: **Um software educativo que suporte uma construção de conhecimento em interacção (com pares e professor).** Departamento de Ciências da Educação da Universidade da Madeira In Actas do 3º Simpósio de Investigação e Desenvolvimento de Software Educativo. Évora: Universidade de Évora. 1998

GIRAFFA , Lúcia M.M. **Uma arquitetura de tutor utilizando estados mentais.** Tese de Doutorado. Porto Alegre: CPGCC/UFRGS, 1999.

HAROLD, E. R**.: XML 1.1 Bible**, 3rd Edition, Indianapolis: Wiley Publishing, 2004, 1022p.

HOPCROFT, J. E.; ULLMAN, J.D.: **Introdução à Teoria de Autômatos, Linguagens e Computação.** Segunda Edição, 2001, 560 págs.

JACOBSON, I. **Object-Oriented Software Enginnering. Addison-Wesley**, 1992, 512p.

LEWIS, H. R.; PAPPADIMITRIOU, C. H.: **Elements of the Theory of Computation**. Englewood Cliffs: Prentice-Hall, 1981.

MACKAY D. J.C.: **Information Theory, Inference, and LearningAlgorithms**. Ed.Cambridge University Press, 2003.

MCLAUGHLIN, Brett**: Java & XML**, 2nd Edition, O'Reilly 2001, 528p.

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

O'REILLY, Tim, **What Is Web 2.0.** Disponível em http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, atualizada em 2006, consultada em 30/ maio/ 2008.

SALOMA, A.: **Formal Languages**. New York: Academic Press, 1973.

SHIELDS, M. W.: **An Introduction to Automata Theory**. Oxford: Blackwell Scientific Publications, 1987.

RAMALHO, J. A.: **XML Teoria e Prática**, São Paulo: Berkeley, 2002, 146p.

RAY, Erik T.: **Learning XML**, First Edition, O'Reilly, 2001, 368p.

ROCKWELL, W.: **XML, XSLT, Java, and JSP: A Case Study in Developing a Web Application**, First Edition, Indianapolis: New Riders, 2001, 746p.

TAJRA, S. F. **Informática na Educação: novas ferramentas pedagógicas para o professor da atualidade.** 2.ed. São Paulo: Érica, 2000, 181p.

TAYLOR, R.P. **The Computer in the School: Tutor, Tool, Tutee,** Teachers College Press, New York, 1980, 274 p.

VALENTE, J. A.: **Diferentes usos do computador na educação.** Campinas, SP: Gráfica da UNICAMP, 1993, 28p.

VELOSO, P. A. S.: **Máquinas e Linguagens: Uma introdução à teoria de autômatos**. São Paulo/SP,1979, 242 págs.