UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

INCLUSÃO DO ALGORITMO DE TRANSFORMAÇÃO DE AUTÔMATO FINITO EM EXPRESSÃO REGULAR NO "EDITOR DE AUTÔMATOS FINITOS"

FERNANDO RAFAEL PICCINI

BLUMENAU
2006

2006/1-13

FERNANDO RAFAEL PICCINI

INCLUSÃO DO ALGORITMO DE TRANSFORMAÇÃO DE AUTÔMATO FINITO EM EXPRESSÃO REGULAR NO "EDITOR DE AUTÔMATOS FINITOS"

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. José Roque Voltolini da Silva – Orientador

BLUMENAU
2006

2006/1-13

INCLUSÃO DO ALGORITMO DE TRANSFORMAÇÃO DE AUTÔMATO FINITO EM EXPRESSÃO REGULAR NO "EDITOR DE AUTÔMATOS FINITOS"

Por

FERNANDO RAFAEL PICCINI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. José Roque Voltolini da Silva – Orientador, FURB
Membro:	Prof. Joyce Martins, MsC. – FURB
Membro:	Prof. Mauricio Capobianco Lopes, MsC. – FURB

Blumenau, 07 de julho de 2006

Dedico este trabalho a meus pais.

AGRADECIMENTOS

Agradeço a Deus, pela saúde e força de vontade que o mesmo me concedeu para ir em busca do conhecimento.

A meus pais e familiares que sempre me deram apoio, inclusive minhas primas as quais me aturaram durante alguns anos.

A todos os professores que participaram da minha formação acadêmica, inclusive meu orientador professor José Roque Voltolini da Silva e também a professora Joyce Martins.

Aos meus chefes e colegas da Wheb-Sistemas que deram-me uma oportunidade de trabalho e conseqüentemente adquiri conhecimento para poder efetuar a implementação do Editor de Autômatos Finitos.

Aos meus amigos de tantas festas que muitas vezes tive que deixá-los bebendo sem a minha presença, por motivos acadêmicos.

Aos escritores dos livros da editora Érica, por escreverem livros um tanto didáticos e ilustrarem os mesmos com uma grande quantidade de Figuras, fazendo prender minha atenção quando os liam.

E por fim fica o agradecimento para uma pessoa a qual admirei durante muitos anos, porém a mesma me abandonou no momento em que mais precisei. Por maior que tenha sido a desilusão em relação a este fato, não desisti do objetivo da minha formação acadêmica e da conclusão deste trabalho. Sendo assim, pude superar um grande desafio em minha vida e seguir adiante.

Só há um bem: o conhecimento. Só há um mal: a ignorância.

Sócrates

RESUMO

O presente trabalho descreve a implementação do algoritmo proposto em Silva (2006) para transformação de autômato finito em expressão regular no Editor de Autômatos Finitos (MORASTONI, 2002). Melhorias e implementação de novas funcionalidades, como apresentação da tabela de transição e inclusão de opção para salvar e abrir um arquivo contendo estados e transições de um autômato finito, foram também realizadas. O Editor de Autômatos Finitos foi inicialmente desenvolvido com objetivo de facilitar a compreensão da teoria estudada pelos acadêmicos da disciplina de Linguagens Formais do curso de Ciências da Computação da FURB. Na extensão da ferramenta foram especificadas três novas classes: TTabela, TExpressaoRegular e TTabelaTransicao. Utilizou-se orientação a objetos, usando os diagramas de casos de uso, diagrama de atividades e diagrama de classes e foi implementada no ambiente Delphi.

Palavras-chave: Autômatos finitos. Expressões regulares. Linguagens regulares. Teoria da computação.

ABSTRACT

The present work describes the implementation of the algorithm of a finite automata transformation in regular expression proposed in Silva (2006) in the finite automata editor (MORASTONI, 2002). Improvements and implementations of new functionalities as transistion table presentation, to allow to save and to open an archive contend the structure (graph) of a Finite Automaton and inclusion of the RxLib library also had been carried through. The Editor of Finite Automata initially was developed with objective to facilitate the understanding of the theory studied for the academics of disciplines of Formal Languages of the course of Computer sciences of the FURB. In the extension of the tool three new classes had been specified: TTabela, TExpressaoRegular and TTabelaTransicao. The tool was specified with orientation to objects, using the diagrams of use cases, diagrams of activities and diagram of class and was implemented in the environment Delphi.

Key-words: Finite automata. Regular languages. Theory of the computation. Regular expressions.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AF – Autômato Finito

AFD – Autômato Finito Determinístico

AFN – Autômato Finito Não determinístico

ε-AFN – Autômato Finito com movimento vazio

EAF – Editor de Autômatos Finitos

ER – Expressão Regular

MDI – *Multiple Document Interface*

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

A Teoria da Computação é fundamental para a Ciência da Computação. Não só proporciona um adequado embasamento teórico necessário para um correto e amplo entendimento da ciência envolvida na computação, como também propicia o desenvolvimento de um raciocínio lógico e formal, cada vez mais necessário em todas as subáreas da computação. Segundo Terada (2005), “pesquisas em Teoria da Computação têm produzido estruturas de dados eficientes e algoritmos que foram incorporados a muitas ferramentas de software e produtos de hardware”. De acordo com Menezes (1998, p. 4), a Teoria da Computação introduz conceitos fundamentais que são desenvolvidos em outras áreas como, por exemplo: biologia computacional (modelos para redes de neurônios), matemática (lógica), lingüística (gramáticas para linguagens naturais), entre outras. A Teoria da Computação também aborda o processamento de funções e o reconhecimento de linguagens (base de estudo de linguagens formais), bem como, trata da análise e do estudo da complexidade de algoritmos. Com isso pode-se expressar de forma abstrata a eficiência de um algoritmo, descrevendo o seu tempo de execução e calculando o tamanho do problema (quantidade de dados) (WANGENHEIM, 1997).

Com o surgimento da Teoria da Computação, criou-se uma nova sub-área, a Teoria de Linguagens Formais, que originou-se na década de 1950 com o objetivo de desenvolver teorias relacionadas com as linguagens naturais. Foi verificado que essa teoria era extremamente importante para linguagens artificiais, em especial, para as linguagens de programação. Dessa forma, as linguagens formais assumem um importante papel na ciência da computação (MENEZES, 1998, p. 1). Dentro dessa teoria, encontra-se a teoria dos autômatos finitos que tem as mais diversas aplicações. Como exemplos podem ser citados: analisadores léxicos (parte integrante de compiladores); editores de textos; sistemas de

pesquisa e atualização em arquivos (em geral, do tipo busca e substituição de informação); sistemas para verificação de circuitos digitais e protocolos de comunicação; entre outras aplicações (HOPCROFT; ULLMAN; MOTWANI, 2002, p. 2).

Considerando o estudo dos autômatos, verifica-se que existe uma carência de ferramentas para auxílio do conteúdo lecionado nas disciplinas relacionadas ao assunto, quais sejam Linguagens Formais e Compiladores. Observa-se também que os alunos encontram dificuldades em entender o conceito de autômatos e até mesmo em não conseguir associar teoria e prática. Em conseqüência disto, foi desenvolvido o Editor de Autômatos Finitos (EAF) (MORASTONI, 2002) com a finalidade de apoiar o processo de ensino-aprendizagem das disciplinas mencionadas. Porém, essa ferramenta possui limitações, entre elas: não permite salvar um autômato finito (AF) especificado; não permite transformar um AF para uma expressão regular (ER); não mostra os estados que geram o não determinismo e também não converte um autômato finito não-determinístico em um autômato finito determinístico.

Nesse sentido, é importante o aprimoramento desta ferramenta, acrescentando novas funcionalidades, entre as quais destaca-se a inclusão de um algoritmo descrito em Silva (2006), para transformação de autômato finito em uma expressão regular. As melhorias descritas visam adequar a ferramenta às necessidades dos acadêmicos em compreender melhor a teoria estudada. Observa-se também que a verificação da validação do algoritmo descrito em Silva (2006) também é proposta, visto que o mesmo ainda não foi validado através de uma implementação.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é acrescentar novas funcionalidades ao EAF, destacando-se a inclusão do algoritmo de transformação de um AF em uma ER.

Os objetivos específicos do trabalho são:

a) apresentar a tabela de transição correspondente ao diagrama de transição de um AF;
b) disponibilizar as funções para abrir e salvar a estrutura (diagrama de transição) de um AF;
c) verificar a validade do algoritmo descrito em Silva (2006);
d) apresentar uma ER correspondente a um AF, utilizando o algoritmo de transformação proposto por Silva (2006).

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em 4 capítulos. No capítulo 2 é descrita a fundamentação teórica utilizada para embasar este trabalho, apresentando alguns conceitos sobre expressões regulares. Além de expressões regulares, são apresentados autômatos finitos, equivalências entre AFs e ERs, algoritmo de transformação de um AF em uma ER proposto por Silva (2006), biblioteca RxLib e trabalhos correlatos.

O capítulo 3 descreve o desenvolvimento do trabalho e a especificação da ferramenta, representada por diagramas de casos de uso e atividades, bem como pelo diagrama de classes. Ainda neste capítulo encontram-se descritas algumas rotinas implementadas na ferramenta.

Finalizando, o capítulo 4 apresenta a conclusão do trabalho, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho, tais como: expressão regular; autômatos finitos, equivalências entre ERs e AFs e a biblioteca RxLib. Na última seção são descritos alguns trabalhos correlatos.

2.1 EXPRESSÃO REGULAR

De acordo com Menezes (1998, p. 50), uma expressão regular denota um formalismo para construir palavras de uma linguagem e é definida a partir de um conjunto básico de operações de concatenação, união e fechamento. Para construção de uma ER, sobre um conjunto de símbolos Σ, pode-se usar as seguintes regras:

a) ∅ para denotar a linguagem vazia;

b) ε para denotar a linguagem L que contém a palavra vazia (L = {ε});

c) x para denotar a linguagem L que contém a palavra x (L = {x}, onde x é um símbolo que pertence a Σ);

d) (x | y) para denotar a união da linguagem X com a linguagem Y (X ∪ Y), onde x e y são ERs;

e) (x . y) para denotar a concatenação da linguagem X com a linguagem Y, onde x e y são ERs;

f) (x*) para denotar o fechamento da linguagem X, onde x é uma ER.

O Quadro 1 apresenta algumas expressões regulares e as linguagens denotadas, considerando o alfabeto composto pelas letras **a** e **b** (Σ = {a, b}).

expressão regular	linguagem denotada
aa	somente a palavra aa
ba*	todas as palavras que iniciam com b seguidas por zero ou mais a
(a | b)*	todas as palavras sobre { a , b }
a*b(a | b)*	todas as palavras contendo um ou mais b como subpalavra
a*ba*ba*	todas as palavras contendo exatamente dois b
(a | b)*(aa | bb)	todas as palavras que terminam com aa ou bb
(a | ε)(b | ba)*	todas as palavras que não possuem dois a consecutivos

Fonte: adaptado de Menezes (1998, p. 51).

Quadro 1 – Exemplo de expressões regulares sobre Σ = {a, b}

2.2 AUTÔMATO FINITO

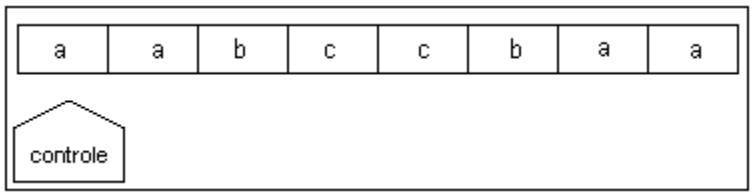
Segundo Menezes (1998, p. 33), um autômato finito pode ser visto como uma máquina, composta de três partes:

a) fita: é o dispositivo de entrada que contém a informação a ser processada. É finita e dividida em quadros, sendo que cada um armazena um símbolo. Estes símbolos pertencem a um alfabeto de entrada;

b) unidade de controle: indica o estado atual da máquina e é responsável por ler o dispositivo de entrada;

c) programa ou função de transição: comanda as leituras e define o estado da máquina em função do símbolo escrito em qualquer posição na fita de entrada.

Na Figura 1 é mostrado um modelo de máquina de estados finitos.



Fonte: Menezes (1998, p. 33).

Figura 1 - Autômato finito como uma máquina com controle finito

No processamento de uma palavra, o autômato lê um símbolo da fita e entra em um novo estado a partir do estado atual e do símbolo que acabou de ser lido. Após a leitura de um

símbolo de entrada, a cabeça de leitura movimenta um quadro para a direita na fita de modo que, no próximo movimento, ela lerá o símbolo no próximo quadro. Esse processo é realizado até que a cabeça de leitura chegue no final da palavra de entrada ou até ocorrer uma condição de parada. O autômato indica sua aprovação ou não quanto ao que leu, baseado no estado em que se encontra no fim. Se o resultado é um estado do conjunto de estados de aceitação ou estados finais[1], a palavra de entrada foi aceita. A linguagem aceita pela máquina é o conjunto de palavras que ela reconhece.

Na descrição acima pode-se perceber que o autômato finito não possui memória de trabalho. Portanto, para armazenar as informações deve-se utilizar o conceito de estado.

Segundo Menezes (1998, p. 33), um AF é definido através da 5-upla $M = (\Sigma, Q, \delta, q_0, F)$, onde:

a) Σ é denominado alfabeto de entrada e corresponde a um conjunto finito não vazio dos símbolos de entrada;
b) Q é um conjunto finito não vazio de estados do autômato;
c) δ é uma função de transição de estados, onde esta função toma como argumentos um estado e um símbolo de entrada e retorna um estado;
d) q_0 é denominado estado inicial e corresponde a um elemento do conjunto Q. É o estado para o qual o reconhecedor deve ser levado antes de iniciar suas atividades;
e) F é um subconjunto do conjunto Q, que contém todos os estados finais do autômato.

Os autômatos finitos, também citado em Menezes (1998), podem ser representados por meio de diagramas de estados (Figura 2) ou através de tabelas de transição (Quadro 2).

[1] Estes estados são aqueles em que o autômato deve terminar quando do reconhecimento das palavras.

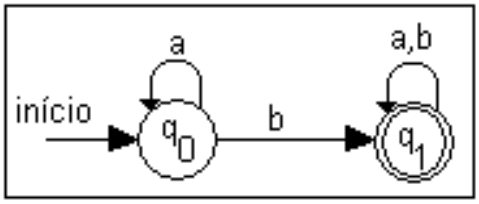


Figura 2 – Exemplo de diagrama de transição para a ER **a*b(a|b)***

	a	b
→ q0	q0	q1
*q1	q1	q1

Quadro 2 – Exemplo de tabela de transição correspondente a Figura 2

Existem três tipos de autômatos finitos: os determinísticos (AFD), os não determinísticos (AFN) e os autômatos finitos como epsilon-transições ou movimento vazio (ε-AFN).

No AFD cada estado tem no máximo uma transição para cada símbolo do alfabeto. Na Figura 3 pode ser visualizado um exemplo de AFD.

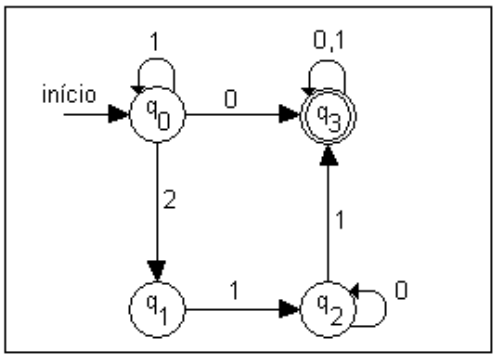


Figura 3 – Exemplo de AFD (Σ = {0, 1, 2})

Em um AFN os estados podem ou não ter uma transição para cada símbolo do alfabeto, ou podem ter até mesmo múltiplas transições para cada símbolo (Figura 4). O autômato aceita uma palavra se existir pelo menos um caminho a partir do estado inicial para um estado final, rotulado com a palavra da entrada.

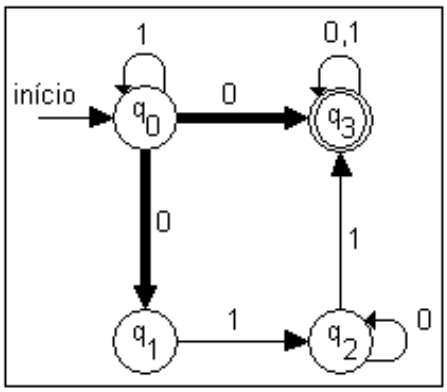
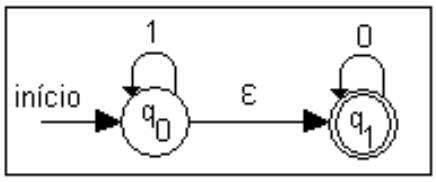


Figura 4 – Exemplo de AFN (Σ = {0, 1})

ε-AFN (Figura 5) constituem uma generalização dos modelos de máquinas não-determinística, ou seja, a partir de um ε-AFN é possível construir um AFN que realiza o mesmo processamento. Segundo Menezes (1998, p. 44), “um movimento vazio é uma transição sem leitura de símbolo algum da fita”, onde o mesmo possui a vantagem de facilitar algumas construções e demonstrações relacionadas com os autômatos.



Fonte: Menezes (1998, p. 46).

Figura 5 – Exemplo de ε-AFN (Σ = {0, 1})

2.3 EQUIVALÊNCIAS ENTRE EXPRESSÕES REGULARES E AUTÔMATOS FINITOS

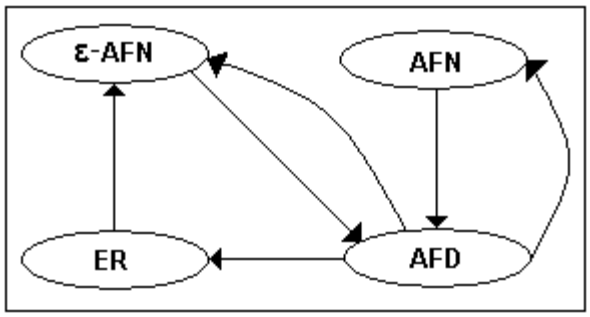
Todos os formalismos apresentados (ER, AFD, AFN e ε-AFN) são equivalentes entre si. Hopcroft, Ullman e Motwani (2002, p. 97) afirmam que:

a) toda linguagem reconhecida por um autômato também é definida por uma expressão regular;

b) toda linguagem definida por uma expressão regular é reconhecida por um autômato.

Na Figura 6 são mostradas as equivalências existentes entre as quatro notações

definidas para as linguagens regulares. Observa-se que o grafo é fortemente conectado (isto é, pode-se ir de cada um dos quatro nós a qualquer outro nó). Dito de outra forma, pode-se converter um AFN em um AFD; é possível construir um AFN a partir de um ε-AFN e é possível gerar um autômato finito a partir de uma expressão regular (e vice versa), todos realizando o mesmo processamento.



Fonte: Hopcroft, Ullman e Motwani (2002, p. 98).
Figura 6 – Equivalências entre as quatro notações para linguagens regulares

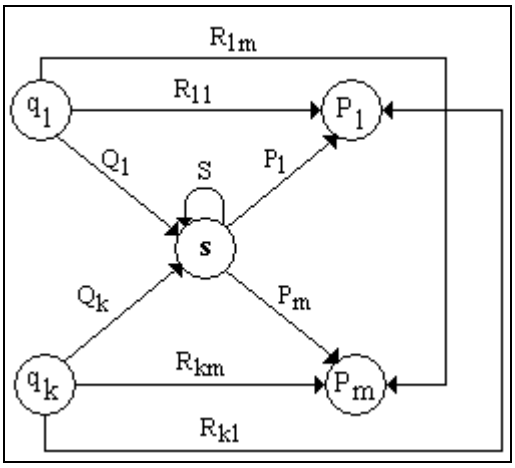
2.3.1 Conversão de autômatos finitos em expressões regulares

Nesta seção são explorados dois algoritmos para conversão de AF em ER, descritos em Hopcroft, Ullman e Motwani (2002) e em Silva (2006).

Segundo Hopcroft, Ullman e Motwani (2002, p. 98), o processo de construção de uma expressão regular para definir uma linguagem de qualquer AFD é extremamente complicado, pois são criadas expressões que descrevem conjuntos de cadeias de caracteres que identificam certos caminhos no diagrama de transição do AFD. Porém, só se permite que os caminhos passem por um subconjunto limitado dos estados. Em alguns casos, os algoritmos que efetuam essa conversão geram ERs muito extensas, as quais pode chegar a ordem de $\mathbf{4^n}$ símbolos.

Aborda-se também a técnica de eliminação de estados: quando é eliminado um estado **s,** todos os caminhos que passam por **s** deixam de existir no autômato. De acordo com Hopcroft, Ullman e Motwani (2002, p. 104), a linguagem do autômato é a união sobre todos

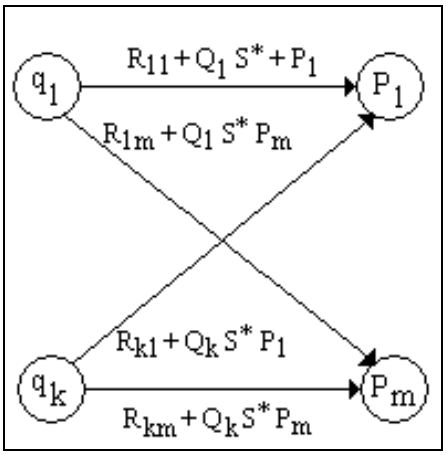
os caminhos desde o estado inicial até o estado de aceitação/final da linguagem formada pela concatenação das linguagens das expressões regulares ao longo deste caminho. Na Figura 7 é mostrado um AF com um estado **s** a ser eliminado.



Fonte: adaptado de Hopcroft, Ullman e Motwani (2002, p. 105).

Figura 7 – Autômato finito com estado **s** a ser eliminado

A Figura 8 mostra o AF após a eliminação do estado **s**. Todas as arestas que envolvem o estado **s** foram eliminadas.



Fonte: adaptado de Hopcroft, Ullman e Motwani (2002, p. 105).

Figura 8 – Autômato finito equivalente após a eliminação do estado **s**

Segundo Hopcroft, Ullman e Motwani (2002, p. 106), a estratégia para construir uma expressão regular a partir de um autômato finito é a seguinte:

a) para cada estado de aceitação/final **q** efetuar o processo de redução citado anteriormente, isto para produzir um autômato equivalente com rótulos de expressões regulares nas arestas. Eliminar todos os estados, exceto **q** (estados de

aceitação/final) e $\mathbf{q_0}$ (estado inicial);

b) se ($\mathbf{q_0}$ = **q**), deve-se executar uma eliminação de estados no autômato original, exceto o estado inicial. Após executado este processo restará um único estado no autômato;

c) a ER a ser gerada é a soma (união) de todas as expressões derivadas dos autômatos reduzidos.

Em resumo, a eliminação de estados é aplicada quando um estado não é o inicial e nem um estado de aceitação/final, ou seja, pode-se eliminar todos os estados que não são o estado inicial nem estados finais, sendo um de cada vez.

O segundo algoritmo apresentado é descrito em Silva (2006), o qual descreve a transformação de um AF em uma ER, até então, ainda não foi validado através de uma implementação. Os passos do algoritmo proposto são descritos no Quadro 3.

Passo	Descrição do Passo
1	Identificar os estados do AF com números distintos, iniciando-se com o número um (1). Sugere-se começar a partir dos estados iniciais.
2	Construir uma tabela com três colunas, onde cada linha representa uma transição no formato **X *er* Y**, onde **X** (coluna 1) identifica o estado de origem, o ***er*** (coluna 2) a expressão regular representada na aresta, que inicialmente conterá apenas o símbolo que é associado à transição e **Y** (coluna 3) o estado de destino.
3	Colocar todas as transições do AF na tabela, identificando os estados iniciais (denominados aqui de estados iniciais antigos) e finais (denominados aqui de estados finais antigos).
4	Criar um novo estado inicial, identificando-o com o número zero (0). Ligar este estado inicial aos antigos estados iniciais através de arestas marcadas com a palavra vazia (ε). Os antigos estados iniciais deixam de ser iniciais.
5	Criar um novo estado final, identificando-o com o próximo número disponível (ainda não usado). Ligar os antigos estados finais ao novo estado final criado através de arestas marcadas com a palavra vazia (ε). Os antigos estados finais deixam de ser finais.
6	Identificar união ("|"): identificar arestas (linha 1 e linha 2) com os mesmos **X** e os mesmos **Y**.
6.1	Criar uma nova linha, sendo o **X** da linha 1 ou linha 2 e o **Y** da linha 1 ou 2. Marcar a aresta com a junção das ***er***s da linha 1 e da linha 2 separadas pelo conectivo "ou" ("|"), colocando entre parênteses, isto é, (***er*** linha 1 | ***er*** linha 2).
6.2	Eliminar as linhas 1 e 2.
6.3	Retornar ao passo 6.
7	Identificar concatenação ("."): identificar um estado (denominado de estado intermediário) onde chega apenas uma aresta (linha 1) e sai apenas outra (linha 2) para outro estado.
7.1	Criar uma nova linha, sendo o **X** da linha 1 e o **Y** da linha 2. Marcar a linha com a junção das ***er***s da linha 1 e da linha 2 separadas pelo conectivo "." (ou deixar em branco), ficando ***er er*** ou ***er . er***.
7.2	Eliminar as linhas 1 e 2.
7.3	Retornar ao passo 6.

8	Identificar fechamento: identificar um estado (denominado de estado intermediário) onde chega apenas uma aresta de um outro estado (linha 1), uma única aresta que sai e chega (neste estado intermediário) (linha 2) e sai apenas uma aresta para um outro estado (linha 3).
8.1	Criar uma nova linha, sendo o **X** da linha 1 e o **Y** da linha 3. Marcar a aresta com a junção das ***er***s da linha 1, da linha 2 acrescida do operador de fechamento (*) e da linha 3, ficando ***er (er)* er***.
8.2	Eliminar as linhas 1, 2 e 3.
8.3	Retornar ao passo 6.
9	Identificar desdobramento ou concatenação múltipla: identificar um estado (denominado de estado intermediário) onde chega uma aresta (linha A) e sai no mínimo duas outras (linhas B_i, onde $1 < i \leq n$, sendo $n \in \mathbb{N}$) para outros estados.
9.1	Identificar a existência de linha B_i onde o **X = Y**.
9.1.1	Assuma que **E1 = (*er*)*,** onde ***er*** é a expressão que está na linha onde identificou o **X =Y**.
9.1.2	Concatenar a **E1** conseguida no passo anterior na frente das demais linhas B_i .
9.1.3	Excluir a linha B_i onde identificou a existência de **X = Y**.
9.2	Na linha onde identificou o desdobramento (**X *er* Y**), assuma que **N1 = X *er***.
9.3	Identifique todas as outras linhas (exceto linhas A e B_i) onde os $\mathbf{Y_j}$ foram iguais ao **X** das linhas B_i (que é igual ao **Y** da linha A) e denomine de linhas C_j. Para cada linha C, duplicá-la criando uma nova linha para cada linha B existente, denominado-as de linhas D. Para cada linha D, fazer a concatenação da ***er*** de cada linha B respectiva e substituir o **Y** da linha D pelo respectivo **Y** da linha B (desdobramento). Excluir a linha C em questão, deixando apenas as novas criadas.
9.4	Em todas as outras linhas Bs onde os **X**s são iguais a **Y** da linha A (onde identificou a concatenação), com formato **X *er* Y,** substituir os **X**s pelo **N1** conseguido no passo 6.1.
9.5	Se o **Y** da linha A é final no AF, deixe-a na tabela, retirando este **Y** (deixe em branco a coluna do estado de destino). Caso contrário elimine esta linha da tabela.
9.6	Retornar ao passo 6.
10	Coloque as ***er***s de todas as linhas que sobraram entre parênteses.
11	Caso exista alguma linha que o **X** não é um estado inicial do AF, elimine esta linha, pois este é um estado inacessível do AF.
12	Caso exista alguma linha que o **Y** (estado destino) não é vazio ou o não é final do AF, elimine esta linha, pois a partir desta linha jamais será alcançado um estado final.
13	Caso restar mais de uma linha na tabela de transformação, junte as ***er***s das linhas, separando-as com o conectivo "ou" ("|").

Fonte: Silva (2006).

Quadro 3 – Algoritmo para transformar AF em ER proposto por Silva (2006)

No Quadro 4 é mostrado um modelo de tabela para efetuar a transformação.

(LE) linha excluída	NÓ DE ORIGEM (X)	EXPRESSÃO REGULAR (er)	NÓ DE DESTINO (Y)

Fonte: adaptado de Silva (2006).

Quadro 4 – Modelo de tabela proposta por Silva (2006), para efetuar a transformação de AF para uma ER

O algoritmo divide-se em quatro procedimentos principais:

a) união;

b) concatenação;

c) fechamento;

d) desdobramento.

A Figura 9 e o Quadro 5 apresentam um AF e a tabela de transformação com os valores iniciais correspondentes ao AF. As duas transições que partem de um mesmo estado de origem (estado 0) para um mesmo estado de destino (estado 1) caracteriza ocorrência de uma união, conforme algoritmo de transformação.

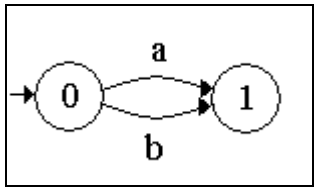


Figura 9 – Exemplo de união em um AF

(LE)	(X)	(ER)	(Y)
F	0	a	1
F	0	b	1

Quadro 5 – Tabela de transformação correspondente ao AF representado na Figura 9

A Figura 10 e o Quadro 6 mostram o AF e a tabela de transformação depois de executada a rotina de união definida no algoritmo proposto em Silva (2006).

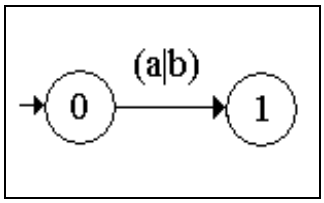


Figura 10 – AF após execução do processo de união

(LE)	(X)	(ER)	(Y)
~~T~~	~~0~~	~~a~~	~~1~~
~~T~~	~~0~~	~~b~~	~~1~~
F	0	(a|b)	1

Quadro 6 – Tabela de transformação após execução do processo de união

O processo de concatenação caracteriza-se pela existência de um estado intermediário onde chega apenas uma transição e sai apenas uma outra para outro estado. A Figura 11 e o

Quadro 7 mostram um AF e a tabela de transformação com os valores iniciais correspondentes ao AF. Observa-se que o estado 1 é o estado intermediário descrito no algoritmo, pois chega uma única transição e sai apenas uma outra transição (para outro estado).

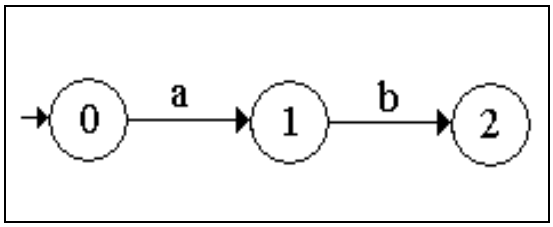


Figura 11 – Exemplo de concatenação em um AF

(LE)	(X)	(ER)	(Y)
F	0	a	1
F	1	b	2

Quadro 7 – Tabela de transformação correspondente ao AF representado na Figura 11

A Figura 12 e o Quadro 8 mostram o AF e a tabela de transformação depois de executada a rotina de concatenação definida no algoritmo proposto em Silva (2006).

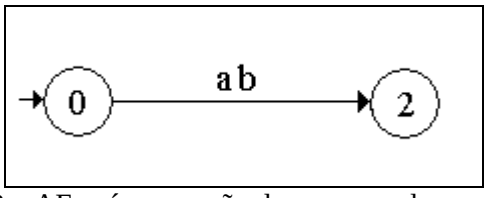


Figura 12 – AF após execução do processo de concatenação

(LE)	(X)	(ER)	(Y)
~~T~~	~~0~~	~~a~~	~~1~~
~~T~~	~~1~~	~~b~~	~~2~~
F	0	a b	2

Quadro 8 – Tabela de transformação correspondente ao AF representado na Figura 12

A Figura 13 e o Quadro 9 apresentam um AF e a tabela de transformação com os valores iniciais correspondentes ao AF. Observa-se que existe a ocorrência de um fechamento no AF, pois existe um estado intermediário (estado 1) onde chega apenas uma transição e sai apenas duas outras transições, sendo que uma delas possui o estado 1 como origem e destino.

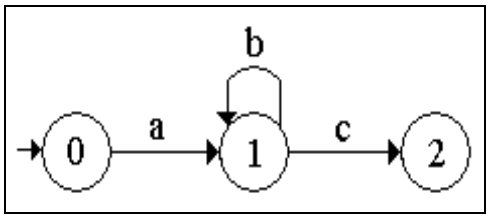


Figura 13 – Exemplo de fechamento em um AF

(LE)	(X)	(ER)	(Y)
F	0	a	1
F	1	b	1
F	1	c	2

Quadro 9 – Tabela de transformação correspondente ao AF representado na Figura 13

A Figura 14 e o Quadro 10 mostram o AF e a tabela de transformação depois de executada a rotina de fechamento definida no algoritmo proposto em Silva (2006).

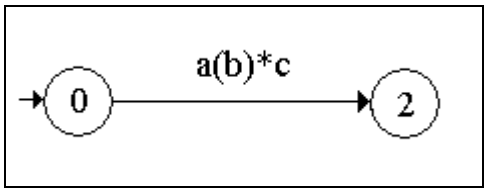


Figura 14 – AF após execução do processo de fechamento

(LE)	(X)	(ER)	(Y)
~~T~~	~~0~~	~~a~~	~~1~~
~~T~~	~~1~~	~~b~~	~~1~~
~~T~~	~~1~~	~~c~~	~~2~~
F	0	a(b)*c	2

Quadro 10 – Tabela de transformação após execução do processo de fechamento

A última rotina descrita no algoritmo de transformação é a de desdobramento. A mesma identifica um estado onde chega uma transição e sai duas ou mais transições para outros estados. A Figura 15 e o Quadro 11 apresentam um AF e a tabela de transformação com os valores iniciais correspondentes ao AF.

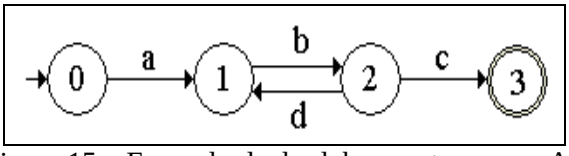


Figura 15 – Exemplo de desdobramento em um AF

(LE)	(X)	(ER)	(Y)
F	0	a	1
F	1	b	2
F	2	c	3
F	2	d	1

Quadro 11 – Tabela de transformação correspondente ao AF representado na Figura 15

A Figura 16 e o Quadro 12 mostram o AF e a tabela de transformação depois de executada a rotina de desdobramento definida no algoritmo proposto em Silva (2006).

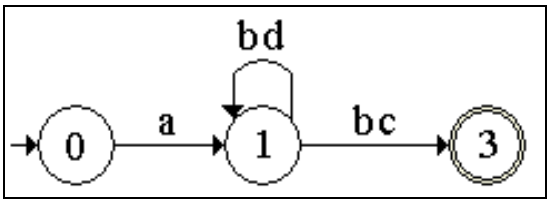


Figura 16 – AF após execução do processo de desdobramento

(LE)	(X)	(ER)	(Y)	
F	0	a	1	
~~T~~	~~1~~	~~b~~	~~2~~	A
F	~~2~~ 1	bc	3	B
F	~~2~~ 1	bd	1	B

Quadro 12 – Tabela de transformação após execução do processo de desdobramento

A Figura 17 apresenta de forma ilustrativa como ficaria o processo de transformação de AF em uma ER utilizando o algoritmo de transformação descrito em Silva (2006).

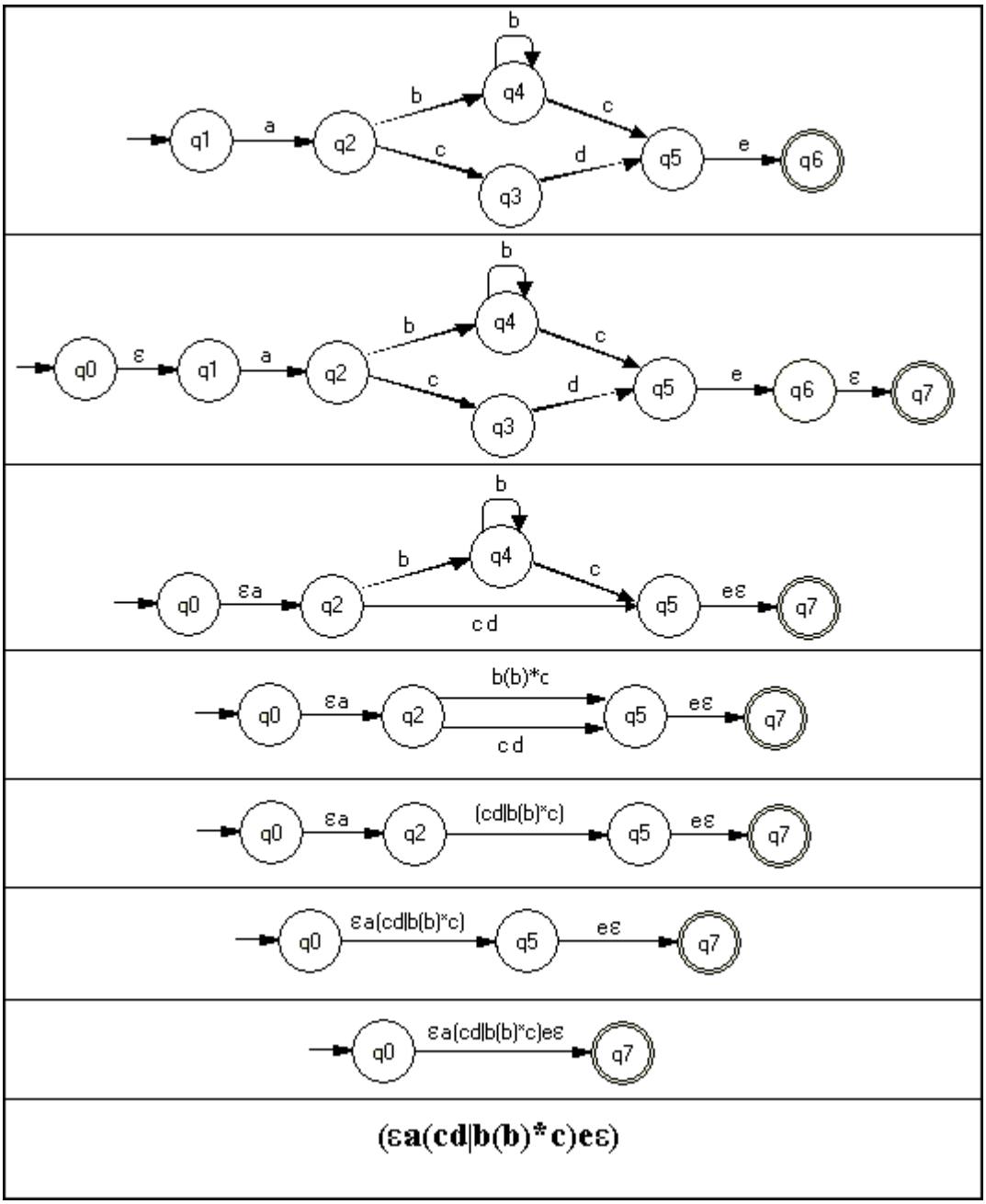


Figura 17 – Exemplo de transformação de um AF para uma ER

2.4 BIBLIOTECA RXLIB

A biblioteca `RxLib` é uma *suite* (pacote) de componentes visuais disponíveis, sendo que a mesma possui a vantagem de ser *free* e vem com código fonte completo. Ela é utilizada para aperfeiçoamento gráfico de aplicações desenvolvidas no ambiente Borland Delphi.

Segundo Sonnino (1998), a biblioteca está dividida em três partes:

a) componentes visuais (`RxControls`): encontram-se muitos `EditBox` especializados, como `ComboEdit` (`edit` com um botão), `FileName` e `DirectoryEdit` (para selecionar arquivos e diretórios), `DateEdit` (com um calendário *popup*), `CurrencyEdit` (para edição de números e valores monetários), `RxCalcEdit` (com uma calculadora *popup*) e `SpinEdit` (para editar valores em ponto flutuante);

b) componentes *data aware* (`RxDBAware`): nesta categoria encontram-se componentes para efetuar acesso e controle a banco de dados, destacando dentre eles: versões *data aware* dos `EditBox` (`DBComboEdit, DBDateEdit, DBCalcEdit`), versões melhoradas dos `Lookups` e `Combos`, com mais eventos e procura incremental e também `Datasets` como `RxQuery` que permitem efetuar consultas a um determinado banco de dados;

c) componentes não visuais (`RXTools`): nesta categoria encontram-se os componentes `TrayIcon`, que disponibiliza um novo ícone na barra de tarefas ao lado do relógio, `Speedbar`, que é uma barra de ferramentas posicionável e customizável, `PageManager`, que gerencia aplicações no formato *wizard* e `DualListBox`, que é uma caixa de diálogo que mostra duas *listboxes*, permitindo transferência de itens entre elas.

Os componentes citados nas três categorias são apenas alguns entre dezenas de outros componentes presentes na biblioteca.

Na Figura 18 são visualizadas quatro diferentes formas de uma tela de cadastro utilizando a biblioteca `RxLib`. Observa-se que as telas possuem diferentes padrões em relação aos seus componentes.

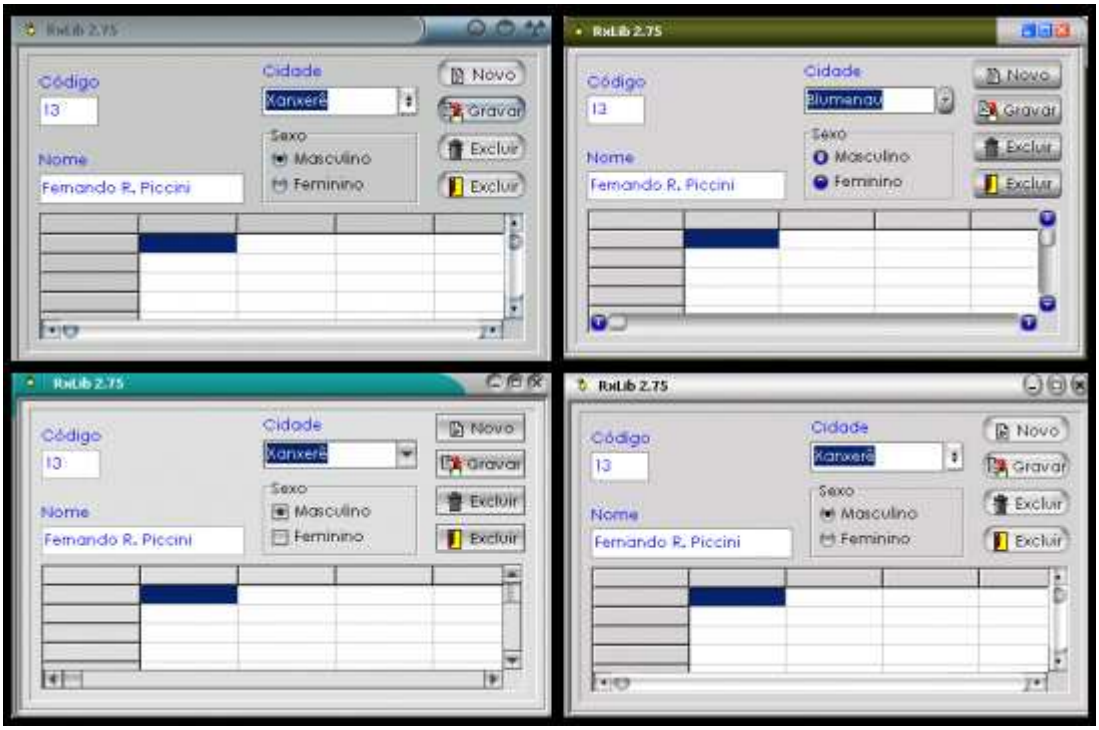


Figura 18 – Exemplos de interface utilizados em telas de cadastro

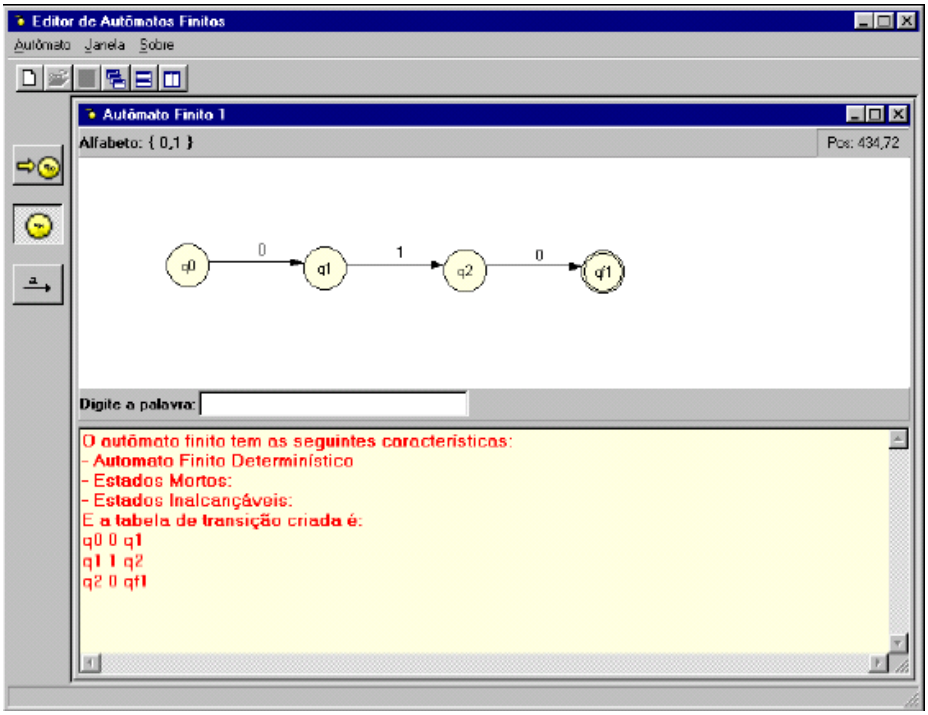
2.5 TRABALHOS CORRELATOS

Trabalhos correlatos ao desenvolvimento desta proposta são descritos a seguir, os quais são: Editor de Autômatos Finitos (MORASTONI, 2002); EduLing (DOGNINI; RAABE, 2003); Protótipo para Transformação de uma Expressão Regular em um Autômato Finito (GLATZ, 2000) e o AutomataEditor (VALE, 1999).

2.5.1 Editor de Autômatos Finitos

O Editor de Autômatos Finitos (MORASTONI, 2002) permite efetuar a construção do diagrama de transição de autômatos finitos determinísticos e não-deterministicos, a validação dos autômatos especificados e o reconhecimento de palavras. A interface do ambiente gráfico foi construída em uma aplicação *Multiple Document Interface* (MDI), de tal forma que é possível especificar vários diagramas de transição ao mesmo tempo, ou seja, pode-se

descrever um ou mais diagramas para cada janela do EAF. A Figura 19 mostra a interface do EAF efetuando a validação de um autômato finito determinístico.



Fonte: Morastoni (2002).

Figura 19 – Validação de um autômato finito determinístico

Para o reconhecimento de palavras a autora cita dois algoritmos: algoritmo específico e o algoritmo genérico recursivo, sendo esse último implementado no Editor de Autômatos Finitos. Para implementação dos estados e transições na parte gráfica foi utilizado os componentes `TDesenha` e `TDesenhaLinha`. No entanto, o EAF possui limitações, as quais já foram citadas na introdução. Em Morastoni (2002) são citadas algumas possíveis extensões, entre elas, tem-se:

a) melhorar a interface gráfica;

b) converter um AFN em um AFD;

c) mostrar os estados que geram o não determinismo;

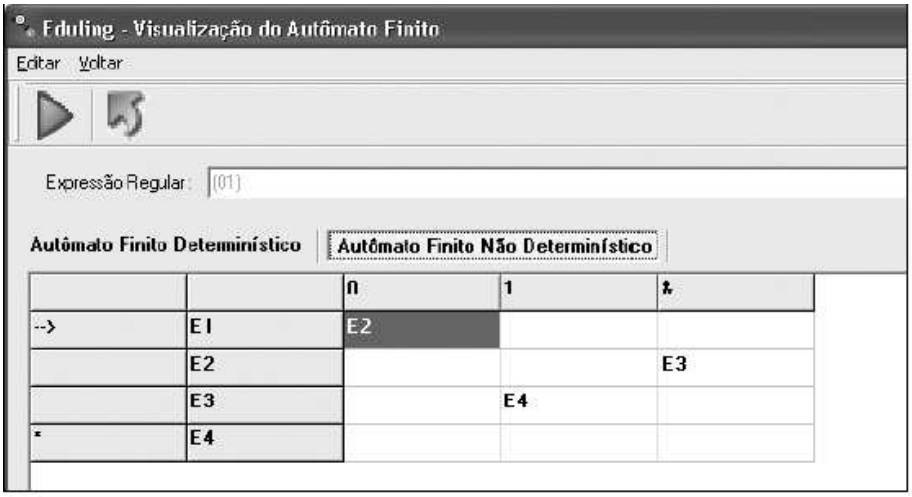
d) implementar um algoritmo de minimização para um AFD.

2.5.2 EduLing

O EduLing é uma ferramenta que possibilita a realização de atividades práticas envolvendo o uso de formalismos para especificação e reconhecimento de linguagens regulares, seja através de autômatos finitos determinísticos, não-determinístico ou expressões regulares (DOGNINI; RAABE, 2003, p. 2). O software também permite verificar a equivalência entre as representações (ER, AFD, AFN). O EduLing está dividido em três módulos:

a) tutorial: aborda conceitos de teoria das linguagens regulares;
b) experimentação livre: possibilita a especificação de linguagens regulares usando expressões regulares ou diagramas de transição. Após a construção da linguagem, é possível visualizar outras representações equivalentes (AFD, AFN, ER e tabela de transição);
c) desafio: busca incentivar o aluno na resolução de problemas relacionados a autômatos.

A Figura 20 apresenta o módulo de Experimentação Livre que possibilita a construção de linguagens regulares. Neste módulo o aluno pode especificar sua linguagem através de expressão regular ou diagrama de transição. Em ambos os casos, depois de terminar a especificação da linguagem, é possível visualizar as outras representações equivalentes (AFD, AFN e ER).



Fonte: Dognini e Raabe (2003, p. 5).

Figura 20 – Equivalência de ER, AFD e AFN

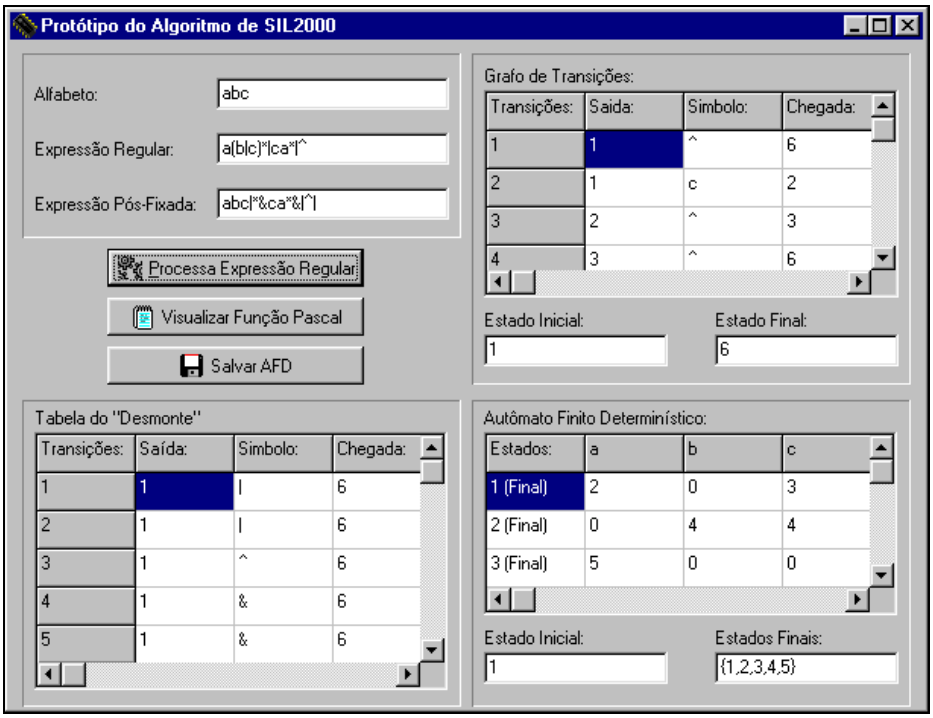
Se o aluno optar por construir a ER, ele deve fazê-lo na área de texto destinada para isso. Já se optar pela construção do diagrama de transição, o EduLing disponibiliza um módulo específico para efetuar o desenho dos estados e das transições.

2.5.3 Protótipo para Transformação de uma Expressão Regular em um Autômato Finito

Em Glatz (2000) é descrita uma ferramenta para transformar uma expressão regular em um autômato finito equivalente. Para a expressão regular submetida à análise é criada uma função de reconhecimento equivalente, na linguagem de programação Pascal. Neste trabalho o autor utilizou o algoritmo proposto em Silva (2000), sendo que até então este ainda não tinha sido validado. Também é implementado o algoritmo descrito em Hopcroft e Ullmann (1979, p. 29). Os dois algoritmos baseiam-se no Teorema de Kleene para transformar expressões regulares em autômatos finitos determinísticos. No mesmo trabalho, Glatz (2000) também faz um estudo do tempo de processamento de algumas expressões regulares submetidas na ferramenta desenvolvida, fazendo uma comparação entre os números obtidos ao final de cada processamento, ou seja, o tempo que o algoritmo leva para transformar uma expressão regular em um autômato finito. Diante deste estudo o autor conclui que o algoritmo

descrito em Silva (2000) é uma nova solução para minimizar problemas como falta de desempenho e alto consumo de recursos computacionais. Sendo assim, o mesmo mostra-se amplamente superior ao algoritmo descrito por Hopcroft e Ullmann (1979).

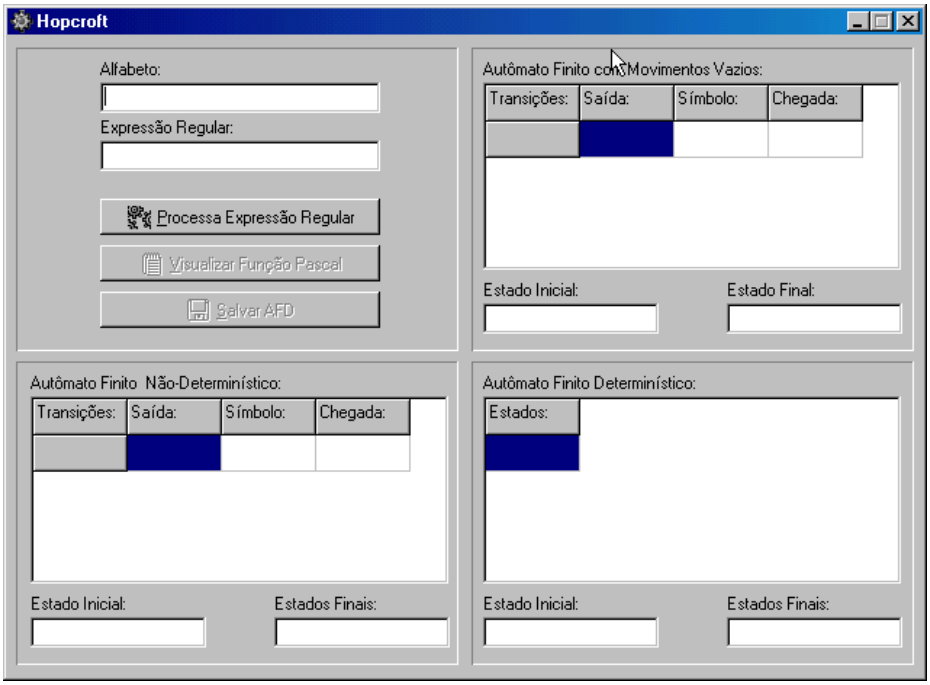
A Figura 21 apresenta a tela principal da ferramenta a qual implementa o algoritmo de transformação de AF para ER proposto por Silva (2000).



Fonte: Glatz (2000, p. 79).

Figura 21 – Tela da ferramenta que transforma uma ER para um AF usando o algoritmo descrito em Silva (2000)

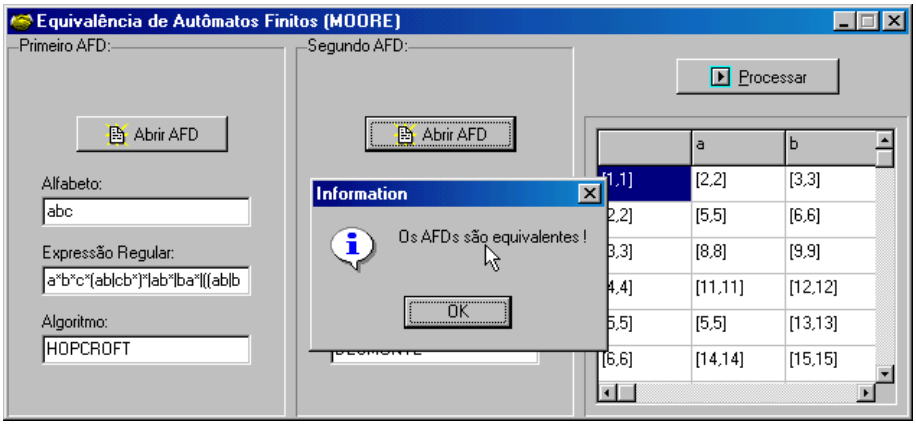
A Figura 22 apresenta a tela principal da ferramenta a qual implementa o algoritmo de transformação de uma ER para um AF descrito por Hopcroft e Ullmann (1979).



Fonte: Glatz (2000, p. 70).

Figura 22 – Tela da ferramenta que transforma uma ER para um AF usando o algoritmo descrito em Hopcroft e Ullmann (1979)

Glatz (2000) ainda descreve em seu trabalho que as várias classes de autômatos finitos podem permitir a equivalência entre si. O autor define que através da equivalência de Moore é possível determinar se dois autômatos finitos determinísticos são equivalentes ou não. A Figura 23 apresenta a tela principal do programa que efetua a equivalência entre dois AFDs.

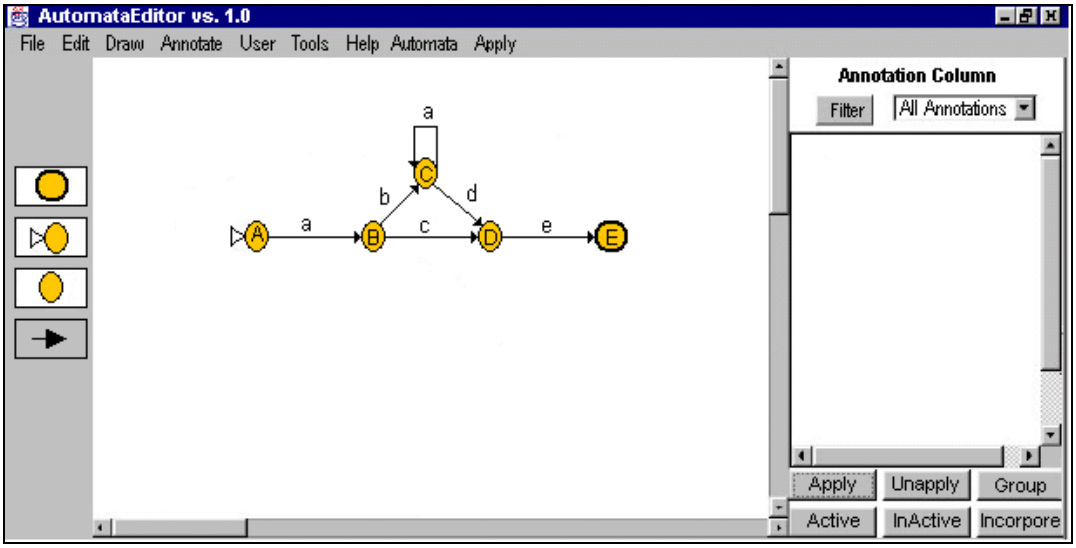


Fonte: Glatz (2000, p. 99).

Figura 23 – Tela principal do protótipo que implementa a equivalência de Moore

2.5.4 AutomataEditor

Em Vale (1999) é descrita uma ferramenta para auxílio no desenvolvimento de autômatos finitos. O editor AutomataEditor é um ambiente para desenvolvimento de autômatos finitos, o qual foi implementado em Java. Como principal característica deste ambiente encontra-se a capacidade de criar, validar e executar os autômatos. Neste trabalho o autor utilizou-se de um *framework*[2] para auxiliar o desenvolvimento do AutomataEditor. A Figura 24 mostra a tela de edição da ferramenta.



Fonte: adaptado de Vale (1999, p. 49).

Figura 24 – Tela de edição do AutomataEditor

A Figura 25 mostra a tela de execução do AutomataEditor, a qual permite que o usuário informe um conjunto de caracteres e a ferramenta verifica se essa cadeia é reconhecida pelo autômato especificado na Figura 24.

[2] O *framework* funciona como um molde para a construção de aplicações ou subsistemas dentro de um domínio, sendo que todas as aplicações construídas a partir de um mesmo *framework* apresentam a mesma estrutura, diferenciando-se em seu comportamento, que é dependente da aplicação. Isso torna as aplicações desenvolvidas a partir de *framework* mais fáceis de manter e mais consistentes para os usuários, que não precisam aprender diferentes aplicações.

Figura 25 – Diálogo de execução do AutomataEditor

A tela de execução do AutomataEditor (Figura 25) é responsável por reconhecer uma determinada cadeia de caracteres, conforme o AF desenhado na tela de edição da ferramenta (Figura 24). Ela também permite que o usuário visualize o andamento da execução, sendo que o usuário também pode escolher para qual estado ir quando o autômato for não-determinístico, bem como retroceder em suas decisões.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são descritos os requisitos, a especificação, a implementação e a operacionalidade do EAF. Ainda ao final, os resultados são avaliados usando a ferramenta proposta em Glatz (2000).

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os principais requisitos a serem desenvolvidos no projeto são:

a) permitir abrir e salvar um arquivo contendo a estrutura (grafo) de um autômato finito em arquivo com extensão **.af** (requisito funcional - RF);
b) possuir um módulo para apresentação das tabelas de transição (RF);
c) gerar uma ER a partir de um AF (RF);
d) utilizar o algoritmo proposto em Silva (2006) para gerar uma ER a partir de um AF (requisito não funcional - RNF);
e) adicionar ao projeto a biblioteca RxLib, visando melhorar a interface e aparência da aplicação (RNF);
f) ser desenvolvido no ambiente Borland Delphi 7.0 (RNF).

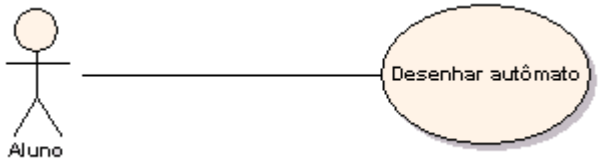
3.2 ESPECIFICAÇÃO

Para especificação do ambiente é utilizada orientação a objetos, representado através dos diagramas de casos de uso, diagramas de atividades e diagrama de classes, sendo que para isto é utilizada a *Unified Modeling Language* (UML) e a ferramenta *Enterprise Architect* para a descrição dos diagramas citados. Os tópicos a seguir apresentam a especificação da

ferramenta.

3.2.1 Casos de uso e diagramas de atividades

Os três casos de usos apresentados a seguir são baseados em três processos efetuados no EAF: desenhar autômato, apresentar tabela de transição e apresentar expressão regular. O Quadro 13 e a Figura 26 apresentam o caso de uso e o diagrama de atividades para desenhar um autômato finito no EAF.



UC.1. Desenhar autômato: permite ao aluno desenhar o autômato na tela de edição do EAF.	
Pré-condições	Não possui.
Fluxo principal	1. O aluno seleciona a opção **Novo**. 2. A ferramenta apresenta a tela para edição do autômato. 3. O aluno desenha o autômato finito na tela de edição do EAF. 4. O aluno seleciona a opção **Salvar**, informando um nome para o arquivo que contém a estrutura (grafo) do autômato finito.
Fluxo alternativo	1. Existe a opção de abrir um arquivo contendo a estrutura (grafo) de um autômato finito. Para isso, deve-se selecionar a opção **Abrir** e informar o nome do arquivo que contém o autômato.
Fluxo de exceção	1. Se não houver ao menos um estado do autômato desenhado na tela de edição no momento em que for salvo o autômato (grafo) será apresentada uma mensagem de aviso.
Pós-condições	Serão habilitadas as opções de Tabela de Transição e Expressão Regular.
Requisitos atendidos	1. Permitir **Abrir** e **Salvar** a estrutura (grafo) do Autômato Finito em Arquivo.

Quadro 13 – Caso de uso: desenhar autômato

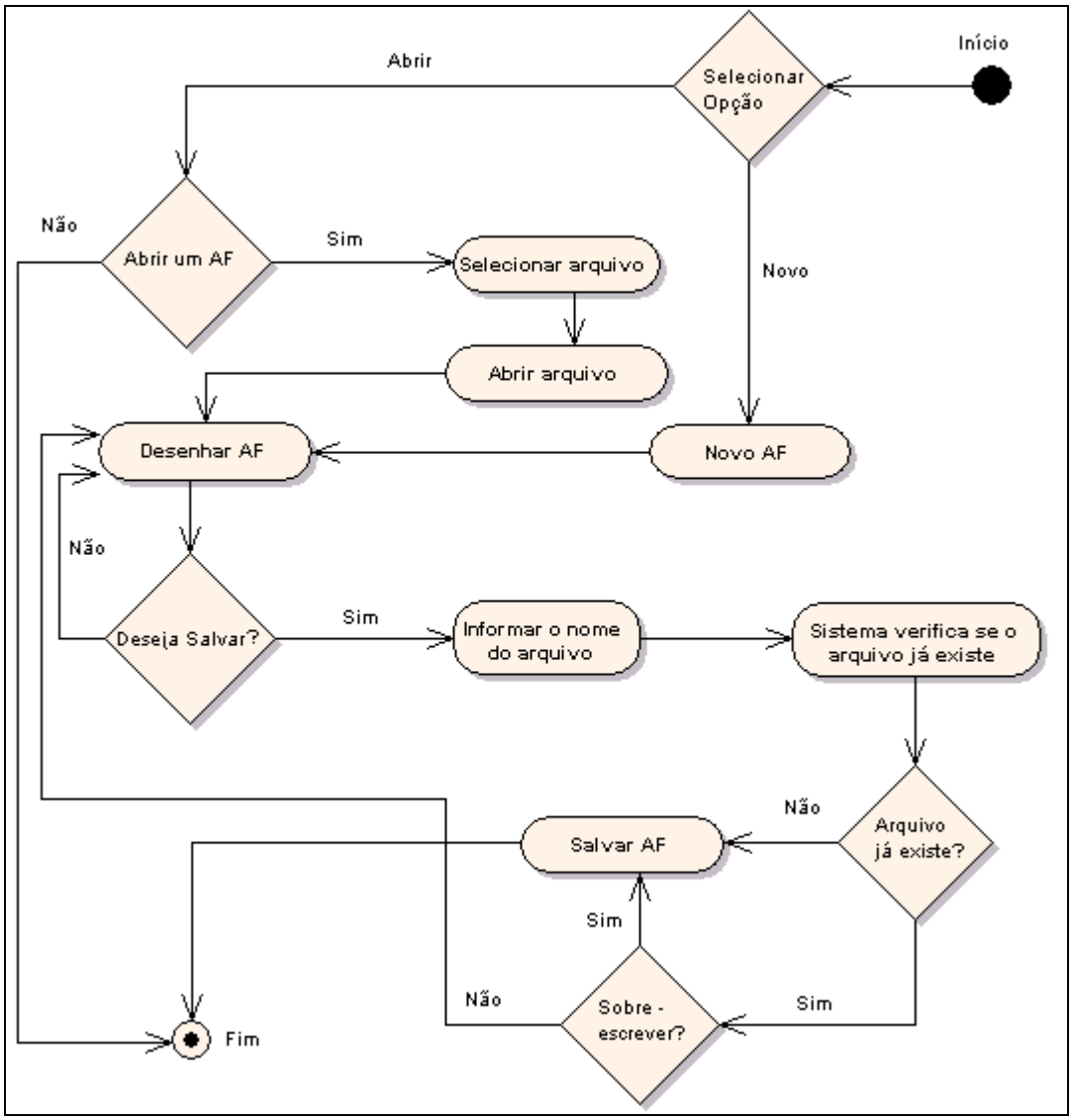
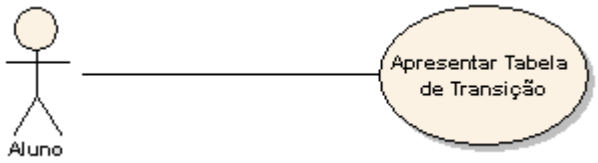


Figura 26 – Diagrama de atividades: desenhar autômato

Para visualizar uma tabela de transição no EAF é necessário que um AF esteja já definido na tela de edição. O Quadro 14 e a Figura 27 apresentam o caso de uso e o diagrama de atividades para apresentação da tabela de transição de um AF no EAF.



UC.2. Apresentar Tabela de Transição: permitir ao aluno visualizar uma tabela de transição correspondente ao autômato finito que está sendo editado no EAF	
Pré-condições	Possuir um autômato finito desenhado em uma janela do EAF.
Fluxo principal	1. O aluno pressiona o botão **Tabela de Transição** localizado na parte inferior da janela de edição da aplicação. 2. A ferramenta percorre a lista de vértices (estados) e arestas (transições) do AF que está sendo editado e apresenta uma mensagem caso não seja um AF válido. 3. O EAF apresenta uma nova janela contendo a tabela de transição correspondente ao AF desenhado na janela de edição (ativa).
Fluxo alternativo	Não possui.
Fluxo de exceção	1. Se não houver ao menos um estado do autômato na tela de edição do EAF será apresentada uma mensagem de aviso.
Pós-condições	Não possui.
Requisitos atendidos	Permitir visualizar uma **Tabela de Transição** correspondente a um autômato finito.

Quadro 14 – Caso de uso: apresentar tabela de transição

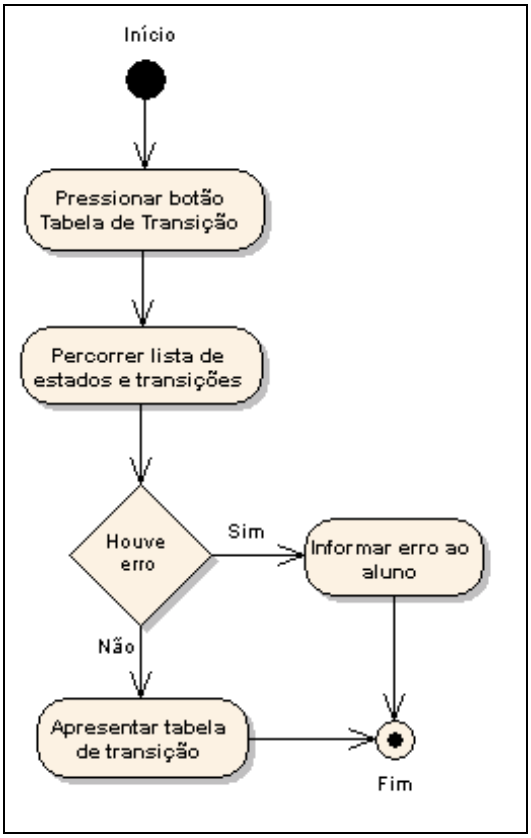
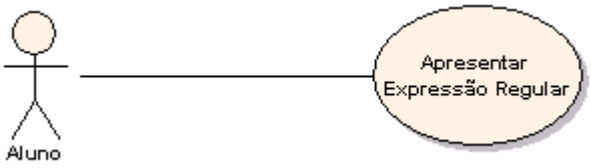


Figura 27 – Diagrama de atividades: apresentar tabela de transição

O terceiro caso de uso deste projeto apresenta o processo que é efetuado no EAF para

apresentação de uma expressão regular referente a um autômato finito. O Quadro 15 e Figura 28 apresentam o caso de uso e o diagrama de atividades para apresentação de uma expressão regular na ferramenta.



UC.2. Apresentar Expressão Regular: permitir ao aluno visualizar uma expressão regular correspondente ao autômato finito que está sendo editado no EAF	
Pré-condições	Possuir um autômato finito desenhado em uma janela do EAF.
Fluxo principal	1. O aluno pressiona o botão **Expressão Regular** localizado na parte inferior da janela de edição da aplicação. 2. A ferramenta executa o algoritmo para transformação de AF em ER. 3. O EAF apresenta uma janela contendo a expressão regular.
Fluxo alternativo	Não possui.
Fluxo de exceção	1. Se não houver ao menos um estado do autômato na tela de edição do EAF será apresentada uma mensagem de aviso.
Pós-condições	Não possui.
Requisitos atendidos	Permitir transformar e visualizar uma **Expressão Regular** correspondente a um autômato finito.

Quadro 15 – Caso de uso: apresentar expressão regular

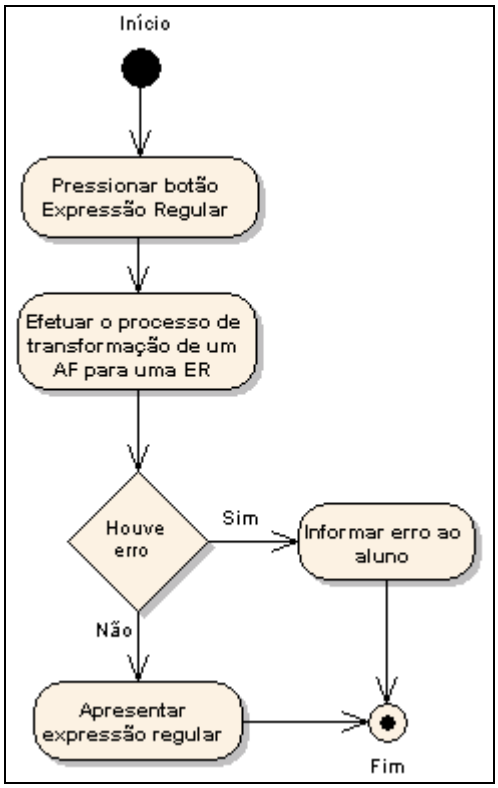


Figura 28 – Diagrama de atividades: apresentar expressão regular

Os casos de uso reconhecer palavra e validar autômato foram especificados em Morastoni (2002). Os casos de uso apresentar expressão regular e apresentar tabela de transição (em destaque na Figura 29) foram especificados neste trabalho e o caso de uso desenhar autômato foi refeito, a partir da implementação apresentada em Morastoni (2002).

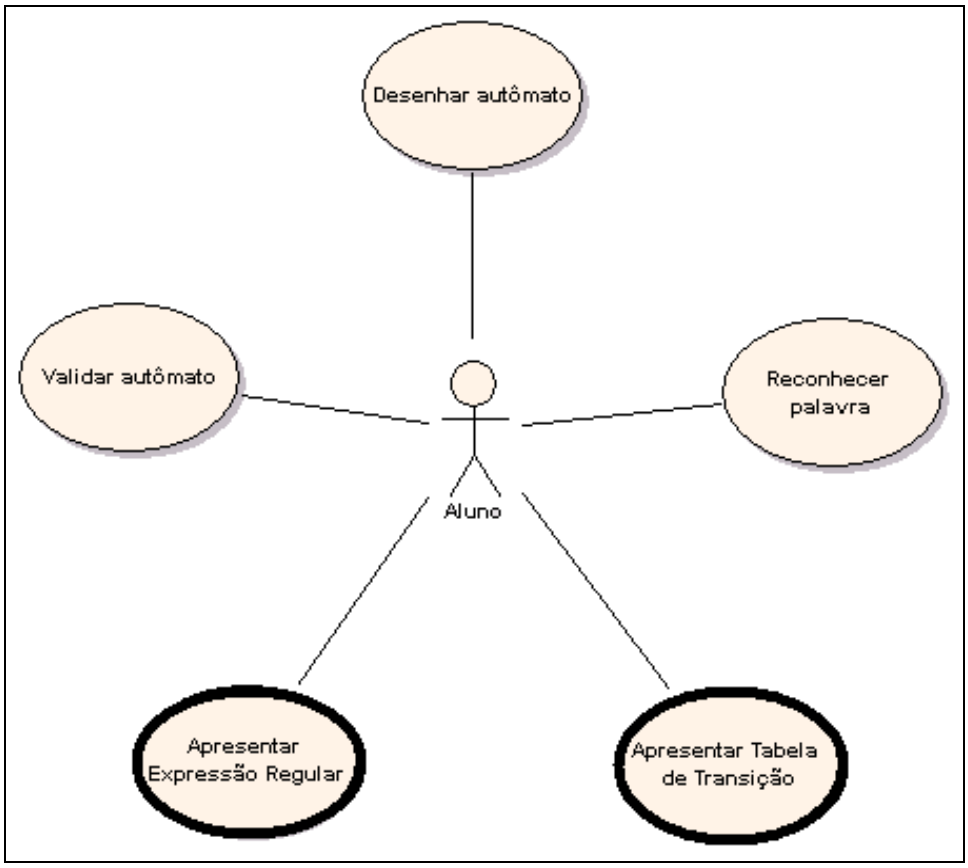


Figura 29 – Casos de usos especificados para o EAF

3.2.2 Diagrama de classes

As classes implementadas na ferramenta são apresentadas na Figura 30.

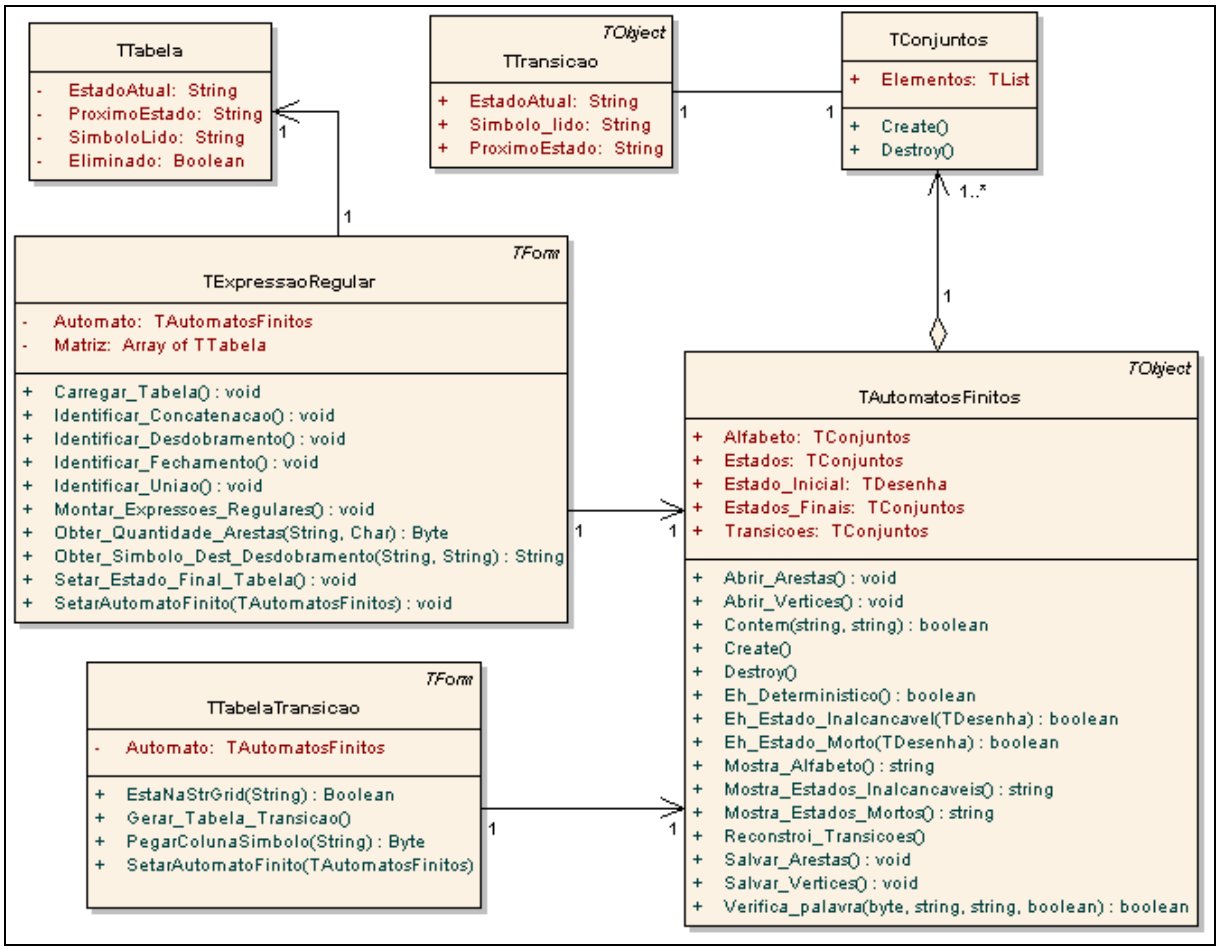


Figura 30 – Diagrama de Classes do EAF

Observa-se que as classes `TAutomatosFinitos`, `TConjuntos` e `TTransicao` já tinham sido especificadas por Morastoni (2002) e as classes `TTabela`, `TExpressaoRegular` e `TTabelaTransicao` foram especificadas neste trabalho.

As classes `TTabelaTransicao` e `TExpressaoRegular` são classes de interface, porém estão sendo consideradas como classes de domínio da aplicação. Ambas as classes foram alteradas, ou seja, abstraiu-se os atributos e métodos desnecessários, considerando apenas as rotinas que foram implementadas neste trabalho.

A classe `TTabelaTransicao` é responsável pela apresentação de uma tabela de transição correspondente ao AF desenhado na tela de edição do EAF e a classe `TExpressaoRegular` é responsável pela transformação e apresentação de uma ER . Ambas as classes possuem um atributo do tipo `TAutomatosFinitos`, porém possuem diferentes rotinas em suas estruturas.

Observa-se que a classe `TAutomatosFinitos` possui o atributo `Matriz` que é um vetor do tipo `TTabela`. A classe `TTabela` é uma das novas classes especificadas na ferramenta, sendo que a mesma foi especificada objetivando implementar a tabela descrita em Silva (2006).

A classe principal da aplicação é a `TAutomatosFinitos`. Esta classe disponibiliza algumas rotinas como: `salvar_vertices`, `salvar_arestas`, `abrir_vértices`, `abrir_arestas` entre outras. Além de seus métodos a classe `TAutomatosFinitos` possui os atributos: estado inicial, alfabeto, estados, estados finais e transições. O atributo estado inicial é do tipo `TDesenha`, que é a classe referente ao componente utilizado para efetuar o desenho dos estados. Considera-se que na ferramenta (EAF) é somente permitido desenhar apenas um estado inicial em cada janela filha (tela de edição). Os outros atributos mencionados da classe `TAutomatosFinitos` são do tipo `TConjuntos`.

A classe `TConjuntos` possui apenas um atributo que se chama elementos, sendo do tipo `TList`, (classe definida no ambiente Delphi). A classe `TList` é uma estrutura caracterizada como sendo uma lista dinâmica a qual possui atributos e métodos para armazenar, remover e controlar instâncias de objetos alocados em memória.

A classe que efetua o armazenamento das transições de um AF é a classe `TTransicao`. Essa classe possui os seguintes atributos: estado atual, símbolo lido e próximo estado. Estes três atributos são do tipo *String*. É através desta classe que se torna possível efetuar a ligação entre os vértices (estados) do AF.

Observa-se que não foi necessária a especificação do algoritmo de transformação de AF para ER, pois a implementação do mesmo foi baseada na descrição do Quadro 3.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentados aspectos da implementação do EAF, bem como o algoritmo de transformação de autômato finito para uma expressão regular.

3.3.1 Técnicas e ferramentas utilizadas

O EAF foi implementado na linguagem *Object Pascal* no ambiente Delphi. Na parte gráfica do editor foi utilizado dois componentes, entre eles o `TDesenha`, que desenha os estados e o `TDesenhaLinha`, que desenha as transições. Ambos componentes citados foram importados do EAF descrito em Morastoni (2002). Para aprimoramento da interface foi utilizada a biblioteca RxLib 2.75 para Delphi 7 e para gerar uma expressão regular a partir de um autômato finito foi utilizado o algoritmo apresentado em Silva (2006).

3.3.1.1 Algoritmo para transformação de autômato finito em expressão regular

A seguir são apresentados alguns procedimentos implementados na ferramenta para efetuar o processo de análise e transformação de AF para ER, conforme o algoritmo de transformação descrito em Silva (2006).

A *procedure* `Identificar_Uniao` (Quadro 16) verifica a existência de duas linhas com a mesma origem e o mesmo destino, ou seja, o valor da coluna **X** e **Y** (linha 1) necessitam ser iguais em uma outra linha (linha 2). Caso exista outra linha com os mesmos valores é criada uma nova linha com o **X** da linha 1 ou 2 e o **Y** da linha 1 ou 2. O Quadro 16 mostra a rotina responsável para efetuar a união das expressões regulares e em destaque o código responsável por criar uma nova linha na tabela de transformação.

```
procedure TExpressaoRegular.Identificar_Uniao;
var
  I, J : Byte;
  Simbolo : String;
begin
  // Percorre matriz (coluna X)
  for I := 0 to High(Matriz) do
    begin
      Simbolo := '';
      // Percorre matriz (coluna Y)
      for J := 0 to High(Matriz) do
        if (I <> J) and (not Matriz[J].Eliminado) and
           (Matriz[I].EstadoAtual = Matriz[J].EstadoAtual) and
           (Matriz[I].ProximoEstado = Matriz[J].ProximoEstado) then
           begin
             // Verifica se é a primeira união
             if (Simbolo = '') then
                Simbolo := Matriz[I].SimboloLido + '|' + Matriz[J].SimboloLido
             else
                Simbolo := Simbolo + '|' + Matriz[J].SimboloLido;
             // Marca linha como eliminada
             Matriz[J].Eliminado    := True;
           end;
      // Verifica se símbolo esta vazio
      if (Simbolo <> '') then
        begin
          // Marca linha como eliminada
          Matriz[I].Eliminado := True;
          // Seta novo tamanho para a matriz
          SetLength(Matriz, High(Matriz) + 1);
          // Atribui valores dos estados para novo índice da matriz
          Matriz[High(Matriz) + 1].EstadoAtual   := Matriz[I].EstadoAtual;
          Matriz[High(Matriz) + 1].ProximoEstado := Matriz[I].ProximoEstado;
          Matriz[High(Matriz) + 1].SimboloLido   := simbolo;
          Matriz[High(Matriz) + 1].Eliminado     := False;
        end;
    end;
end;
```

Quadro 16 – Rotina responsável pela identificação e tratamento da união

O Quadro 17 apresenta a rotina de concatenação das expressões. O processo efetuado na *procedure* `Identificar_Concatenacao` (Quadro 17) efetua uma análise na tabela de transformação identificando a existência de um estado intermediário que possui apenas uma transição de chegada e uma única transição de saída. Caso está análise seja satisfeita é criada uma nova linha na tabela para efetuar a concatenação das **er** (linha 1) e **er** (linha 2). Após o processo de concatenação as linhas 1 e 2 devem ser eliminadas (Quadro em destaque). Observa-se que ao final da *procedure* `Identificar_Concatenacao` é chamada a *procedure* `Identificar_Uniao` descrita no Quadro 16.

```
procedure TExpressaoRegular.Identificar_Concatenacao;
// Declaração das variáveis
var
  I1, J1, Count1,
  I2, J2, Count2: Byte;
begin
  // Percorre matriz (Coluna X)
  for I1 := 0 to High(Matriz) do
    begin
      Count1:= 0;
      // Percorre matriz (Coluna Y)
      for J1 := 0 to High(Matriz) do
        if (not Matriz[I1].Eliminado) and
           (Matriz[I1].EstadoAtual = Matriz[J1].EstadoAtual) then
           Inc(Count1);
      // Verifica se achou apenas um estado igual ao analisado
      if (Count1 = 1) then
        begin
          for I2 := 0 to High(Matriz) do
            begin
              Count2 := 0;
              for J2 := 0 to High(Matriz) do
                if (not Matriz[I2].Eliminado) and
                   (Matriz[I2].ProximoEstado = Matriz[J2].ProximoEstado) then
                   Inc(Count2);
              // Verifica se achou apenas um estado igual ao analisado
              if (Count2 = 1) then
                 begin
                  // Seta novo tamanho para a matriz
                  SetLength(Matriz, High(Matriz) + 1);
                  // Atribui valores dos estados para novo índice da matriz
                  Matriz[High(Matriz) + 1].EstadoAtual   := Matriz[I2].EstadoAtual;
                  Matriz[High(Matriz) + 1].ProximoEstado:=Matriz[I1].ProximoEstado;
                  Matriz[High(Matriz) + 1].SimboloLido := Matriz[I2].SimboloLido +
Matriz[I1].SimboloLido;
                  Matriz[High(Matriz) + 1].Eliminado     := False;
                  Matriz[I1].Eliminado    := True;
                  Matriz[I2].Eliminado    := True;
                  // Chama a rotina de União
                  Identificar_Uniao;
                end;
            end;
        end;
    end;
end;
```

Quadro 17 – Rotina responsável pela identificação e tratamento da concatenação

O algoritmo descrito em Silva (2006) também identifica e trata o fechamento em um AF. Conforme descrição do algoritmo de transformação, o processo para identificar um fechamento no AF encontra uma transição cuja origem parte de um estado **z** e seu destino também encontra-se no estado **z**. Para efetuar a conversão de um AF para uma ER foi criada a *procedure* `Identificar_Fechamento`**.** Inicialmente a rotina identifica um estado intermediário **z**, sendo que chega apenas uma aresta de outro estado e sai apenas uma aresta para outro estado. A segunda etapa consiste em verificar a existência de uma única aresta que tem como origem e destino o próprio estado **z**. Caso a rotina identifique o estado

intermediário **z,** cria-se uma nova linha na tabela de transformação e atribui **X** com o valor do estado de origem em relação ao estado intermediário **z** e atribui **Y** com o valor do estado de destino em relação ao estado intermediário **z**. Ainda nesta etapa é necessário atribuir o operador de fechamento (*) após a ER e eliminar as linhas de origem, destino e a linha que consistia o fechamento. Ao final da *procedure* `Identificar_Fechamento` é chamada a rotina `Identificar_Uniao`. O Quadro 18 apresenta o processo executado pela rotina responsável por identificar e tratar fechamento no EAF. Os trechos em destaque no Quadro 18 mostram as validações efetuadas pela rotina `Identificar_Fechamento` para identificação de um estado intermediário.

```
procedure TExpressaoRegular.Identificar_Fechamento;
// Declaração das variáveis
var
  I, J,
  I1, J1, Count1,
  I2, J2, Count2: Byte;
begin
  // Percorre a matriz utilizando dois índices
  for I := 0 to High(Matriz) do
    for J := 0 to High(Matriz) do
      // Verifica se o valor do primeiro índice "I" é igual ao segundo "J"
      if (I = J) and
        (Matriz[I].EstadoAtual = Matriz[J].proximoEstado) then
        begin
          for I1 := 0 to High(Matriz) do
            begin
              // Inicializa contador
              Count1:= 0;
              for J1 := 0 to High(Matriz) do
                // Verifica se a linha já foi eliminada e os índices são iguais
                if (not Matriz[I1].Eliminado) and
                   (Matriz[I1].EstadoAtual = Matriz[J1].EstadoAtual) then
                   // Incrementa primeiro contador auxiliar
                   Inc(Count1);
              if (Count1 = 1) then
                begin
                  // Percorre novamente a matriz
                  for I2 := 0 to High(Matriz) do
                    begin
                      // Inicializa segundo contador
                      Count2 := 0;
                      for J2 := 0 to High(Matriz) do
                        if (not Matriz[I2].Eliminado) and
                         (Matriz[I2].ProximoEstado = Matriz[J2].ProximoEstado) then
                          // Incrementa segundo contador auxiliar
                          Inc(Count2);
                      // Verifica uma segunda condição
                      if (Count2 = 1) then
                        begin
                          // Seta novo tamanho para matriz
                          SetLength(Matriz, High(Matriz) + 1);
                          // Atribui valores dos estados para novo índice da matriz
                          Matriz[High(Matriz) + 1].EstadoAtual   :=
Matriz[I2].EstadoAtual;
                          Matriz[High(Matriz) + 1].ProximoEstado :=
Matriz[I1].ProximoEstado;
                          Matriz[High(Matriz) + 1].SimboloLido   :=
Matriz[I2].SimboloLido + Matriz[I1].SimboloLido;
                          Matriz[High(Matriz) + 1].Eliminado     := False;
                          // Marca os dois estados como eliminado
                          Matriz[I1].Eliminado    := True;
                          Matriz[I2].Eliminado    := True;
                          // Chama a rotina de União
                          Identificar_Uniao;
                        end;
                    end;
                end;
            end;
        end;
end;
```

Quadro 18 – Rotina responsável pela identificação e tratamento de fechamento

Conforme o passo 9 do Quadro 3, Silva (2006) descreve o processo para identificar desdobramento ou concatenação múltipla. Este processo foi implementado pela p*rocedure*

`Identificar_Desdobramento`. Esta rotina (*procedure*) inicia com a identificação de um estado intermediário, sendo que chega uma aresta (linha A) e sai no mínimo duas outras (linhas B_i) para outros estados. Para controle do processo existe uma variável auxiliar chamada **N1** a qual assume **X er** para a linha onde foi identificado o desdobramento. Após a identificação do desdobramento é necessário verificar todas as linhas (exceto as linhas A e B_i) onde os $\mathbf{Y_j}$ forem iguais ao **X** das linhas B_i , e denominá-las de linhas C_j. Conforme descrição do algoritmo de transformação, existe uma verificação para as linhas B_i, sendo que esta verificação consiste em identificar uma linha B_i onde **X = Y**. Caso existir um B_i que acontecer esta situação assuma-se que **E1 = (ER)*** (ER é a expressão que está na linha onde foi identificado **X = Y**) e em seguida concatena-se a **E1** conseguida no passo anterior na frente das demais linhas B_i. Em seguida é necessário excluir a linha B_i onde identificou a existência **X = Y**. Depois da verificação das linhas B_i, é percorrida novamente a matriz, a qual duplica cada linha C (para cada linha B_i existente). Ainda ao final da *procedure* `Identificar_Desdobramento` é feita a chamada da rotina `Identificar_Uniao` apresentada no Quadro 16. O Quadro 19 apresenta o código fonte do procedimento `Identificar_Desdobramento` implementado na ferramenta. O trecho em destaque é responsável pela identificação de um estado **z** o qual possui duas ou mais transições que saem do estado **z** e uma ou mais transições que chegam no estado **z**.

```
procedure TExpressaoRegular.Identificar_Desdobramento;

  // Utiliza o valor "X" para identificar um status auxiliar p/ controle da Matriz
  procedure Limpar_Letra_Matriz;
  var
    Ind : Byte;
  begin
    for Ind := 0 to High(Matriz) do
      Matriz[Ind].letra := 'X';
  end;

  // Verificar se o Estado "X" é igual ao estado "Y"
  function Identificar_Origem_Destino_B: String;
  var
    Ind : Byte;
    ER  : String;
  begin
    ER := '';
    for Ind := 0 to High(Matriz) do
      if (not Matriz[Ind].Eliminado) and
         (Matriz[Ind].Letra = 'B') and
         (Matriz[Ind].EstadoAtual = Matriz[Ind].ProximoEstado) then
         ER := ('(' + Matriz[Ind].SimboloLido + ')*');
    Result := ER;
  end;

  // Function responsável por obter a linha “A” da Matriz (Tabela)
  function Obter_Indice_A(Est: String) : Byte;
  var
    Ind: Byte;
  begin
    for Ind := High(Matriz) - 1 downto 0 do
      if (not Matriz[Ind].Eliminado) and
         (Matriz[Ind].ProximoEstado = Est) then
         Result := Ind;
  end;

// Declaração das variáveis
var
  Tab : Tabela;
  N1  , Smb1, ER_B : String;
  I1, I2, I3, J1, J2, Ind_A: Byte;
begin
  // Efetua limpeza da Matriz
  Limpar_Letra_Matriz;
  // Efetuar o passo 9 da Algoritmo de transformação
  for I1 := 0 to High(Matriz) do
    if (not Matriz[I1].Eliminado) then
      if (Obter_Quantidade_Arestas('C',Matriz[I1].EstadoAtual) > 0) and
         (Obter_Quantidade_Arestas('S',Matriz[I1].EstadoAtual) > 1) and
         (Matriz[I1].EstadoAtual <> Matriz[I1].ProximoEstado) then
         begin
           // Encontrar a 1ª transição que chega no estado testado (Determinar a
Linha "A")
           Ind_A := Obter_Indice_A(Matriz[I1].EstadoAtual);
           Matriz[Ind_A].Letra := 'A';
           // Efetuar o passo 9.1 da Algoritmo de trasnformação
           N1   := Matriz[Ind_A].EstadoAtual;
           Smb1 := Matriz[Ind_A].SimboloLido;
           for J1 := 0 to High(Matriz) do
             begin
             // Verifica se a linha não foi eliminada
             if (not Matriz[J1].Eliminado) then
               begin
                 // Verifica se a linha é uma do tipo "B"
                 if (Matriz[J1].EstadoAtual = Matriz[Ind_A].ProximoEstado) then
                    Matriz[J1].Letra := 'B';
                 if not (Matriz[J1].Letra[1] in ['A','B']) and
```

```
                    (Matriz[J1].ProximoEstado = Matriz[Ind_A].ProximoEstado) then
                    Matriz[J1].letra := 'C';
              end;
            end;
          ER_B := Identificar_Origem_Destino_B;
          // Efetuar o passo 9.2 da Algoritmo de trasnformação
          for I2 := 0 to High(Matriz) do
            if (Matriz[I2].letra = 'C') then
              begin
                for J2 := 0 to High(Matriz) do
                  if (Matriz[J2].letra = 'B') then
                    begin
                      if (Matriz[J2].EstadoAtual = Matriz[J2].ProximoEstado) then
                        Matriz[J2].Eliminado := True
                      else
                        begin
                          if (ER_B <> '') then
                            Matriz[J2].SimboloLido := ER_B +
Matriz[J2].SimboloLido;
                          // Cria nova linha na Matriz
                          SetLength(Matriz, (Length(Matriz) + 1));
                          Tab.EstadoAtual   := Matriz[I2].EstadoAtual;
                          Tab.ProximoEstado := Matriz[J2].ProximoEstado;
                          Tab.SimboloLido   := Matriz[I2].SimboloLido +
Matriz[J2].SimboloLido;
                          Tab.Eliminado     := False;
                          Matriz[Length(Matriz) - 1] := Tab;
                        end;
                    end;
                  Matriz[I2].Eliminado := True;
              end;
          // Efetuar o passo 9.3 da Algoritmo de trasnformação
          for I3 := 0 to High(Matriz) do
            begin
              if (Matriz[I3].letra = 'B') then
                begin
                  Matriz[I3].EstadoAtual := N1;
                  Matriz[I3].SimboloLido := Smb1 + Matriz[I3].SimboloLido;
                end;
              // Efetuar o passo 9.4 da Algoritmo de trasnformação
              if (Matriz[I3].letra = 'A') and
                 (TDesenha(Matriz[I3].ProximoEstado).Estado <> esFinal) then
                 Matriz[I3].Eliminado := True;
            end;
          // Chama a rotina  responsável por identificar e tratar união
          Identificar_Uniao;
          Exit;
        end;
end;
```

Quadro 19 – Rotina responsável pelo tratamento de desdobramento

Assim como o Quadro 19 apresenta a rotina para tratar o desdobramento no algoritmo de transformação, existem outros procedimentos que complementam o algoritmo descrito em Silva (2006), como: `Montar_Expressões_Regulares`, `Setar_Estado_Final_Tabela` e `Carregar_Tabela`.O objetivo foi mostrar apenas algumas rotinas que são importantes no processo de transformação de AF para uma ER.

No menu principal do EAF estão localizadas várias funcionalidades da ferramenta,

dentre as quais encontra-se a opção de abrir e salvar a estrutura (grafo) de um autômato finito.

Para efetuar o processo da gravação é necessário ter ao menos uma transição desenhada no EAF. Caso esta condição seja atendida, após clicar na opção de salvar é apresentada uma tela para informar o nome do arquivo e local onde o mesmo será gravado. O arquivo gravado possuirá a extensão ".af".

Nos Quadros 20 e 21 são apresentadas as implementações dos algoritmos que efetuam a gravação de arquivo no EAF. As rotinas apresentadas foram desenvolvidas na classe `TAutomatosFinitos.`

O Quadro 20 apresenta a *procedure* `Salvar_Vertices` que é responsável por percorrer e salvar todos estados (inicial, finais, não inicial e não finais) do autômato que está sendo editado no EAF.

No Quadro 21 tem-se a *procedure* `Salvar_Arestas` que efetua o processo semelhante ao algoritmo anterior, porém percorre a lista das transições (arestas) e conseqüentemente salva-as em um arquivo que é definido pelo usuário no momento de efetuar a gravação do arquivo.

```
procedure TAutomatosFinitos.Salvar_Vertices;
// Declaração das  variáveis
var
   Ind, IndAux: Byte;
   Desenho: TDesenha;
begin
   // Associar nome físico a nome lógico
   AssignFile(ArqVertice,Self.Caption);
   // Cria arquivo
   Rewrite(ArqVertice);
   // Inicializa contador
   Ind := 0;
   // Atribuição dos valores do estado inicial para o tipo registro
   RegVertice.Vertices[Ind + 1].Estado   :=
Obter_Char_Estado(Automato.Estado_Inicial.Estado);
   RegVertice.Vertices[Ind + 1].Caption := Automato.Estado_Inicial.Caption;
   RegVertice.Vertices[Ind + 1].Top       := Automato.Estado_Inicial.Top;
   RegVertice.Vertices[Ind + 1].Left       := Automato.Estado_Inicial.Left;
   // Grava informações do tipo registro no arquivo
   Write(ArqVertice, RegVertice.Vertices[Ind + 1]);
   // Verifica se a lista dos estados normais se não está vazia
   if (Automato.Estados.Elementos.Count > 0) then
  // Percorre lista dos estados normais do autômato finito
   for Ind := 0 to Automato.Estados.Elementos.Count - 1 do
      begin
          Desenho:= Automato.Estados.Elementos[Ind];
          RegVertice.Vertices[Ind + 2].Estado   := Obter_Char_Estado(Desenho.Estado);
          RegVertice.Vertices[Ind + 2].Caption := Desenho.Caption;
          RegVertice.Vertices[Ind + 2].Top       := Desenho.Top;
          RegVertice.Vertices[Ind + 2].Left       := Desenho.Left;
          // Posiciona cabeça de gravação no final do arquivo
          seek(ArqVertice, FileSize(ArqVertice));
          // Efetua novamente gravação dos estados no arquivo
          Write(ArqVertice, RegVertice.Vertices[Ind + 2]);
      end;
   // Incrementa contador
   IndAux := Ind + 2;
   // Verifica se a lista dos estados finais se não está vazia
   if (Automato.Estados_Finais.Elementos.Count > 0) then
      // Percorre lista dos estados finais
      for Ind:= 0 to Automato.Estados_Finais.Elementos.Count - 1 do
         begin
            Desenho := Automato.Estados_Finais.Elementos[Ind];
            RegVertice.Vertices[IndAux + Ind].Estado   :=
Obter_Char_Estado(Desenho.Estado);
            RegVertice.Vertices[IndAux + Ind].Caption := Desenho.Caption;
            RegVertice.Vertices[IndAux + Ind].Top       := Desenho.Top;
            RegVertice.Vertices[IndAux + Ind].Left       := Desenho.Left;
            seek(ArqVertice, FileSize(ArqVertice));
            // Grava registro no arquivo
            Write(ArqVertice, RegVertice.Vertices[IndAux + Ind]);
        end;
   // Fecha o arquivo
   CloseFile(ArqVertice);
                                                End;
```

Quadro 20 - Rotina responsável por salvar os estados (vértices) do AF

```
procedure TAutomatosFinitos.Salvar_Arestas;
// Declaração das variáveis
var
     Ind: Word;
     TransAux: TTransicao;
begin
   // Associar nome físico a nome lógico
   AssignFile(ArqAresta, ExtractFileDir(self.Caption) + '\' +
   copy(ExtractFileName(self.Caption),1,length(ExtractFileName(self.Caption))-3) +
'.art');
   // Cria arquivo
   Rewrite(ArqAresta);
   // Verifica se a lista de transições esta vazia
   if (Automato.Transicoes.Elementos.count > 0) then
      // Percorre lista de transições
      for Ind := 0 to Automato.Transicoes.Elementos.count - 1 do
         begin
            // Atribui valores das transições para o registro do tipo TTransicao
            RegAresta.EstadoAtual      :=
TTransicao(Automato.Transicoes.Elementos[ind]).EstadoAtual;
            RegAresta.SimboloLido     :=
TTransicao(Automato.Transicoes.Elementos[ind]).Simbolo_Lido;
            RegAresta.ProximoEstado :=
TTransicao(Automato.Transicoes.Elementos[ind]).ProximoEstado;
            // Grava informações do tipo registro no arquivo
            Write(arqAresta,regAresta);
        end;
   // Fecha arquivo
   CloseFile(ArqAresta);
end;
```

Quadro 21 - Rotina responsável por salvar as transições (arestas) do AF

Nos Quadros 22, 23 e 24 são apresentados os algoritmos para efetuar a abertura de arquivos com extensão **.af**, ou seja, arquivos os quais contenham a estrutura (grafo) de um autômato finito. Observa-se que as rotinas apresentadas foram implementadas na classe `TAutomatosFinitos`.

```
function Obter_Estado(Nome: String): TDesenha;
// Declaração das variáveis
var
    I: byte;
  begin
    Result := nil;
    // Verifica o nome do estado inicial
    if (EstadoInicial.Caption = Nome) then
      Result := EstadoInicial
    else
      begin
      // Verifica se a lista dos estados normais não esta vazia
      if (Automato.Estados.Elementos.Count > 0) then
        begin
          i := 0;
          while ((Result = nil) and (i < (Automato.Estados.Elementos.Count))) do
            begin
              if (TDesenha(automato.Estados.Elementos[i]).Caption = nome) then
                 // Retorna o objeto estado encontrado
                 Result:= TDesenha(automato.Estados.Elementos[i]);
              // Incrementa o indice da lista
              inc(i);
            end;
        end;
      // Verifica se a lista dos estados finais não esta vazia
      if (Result = nil) and (Automato.Estados_Finais.Elementos.Count > 0) then
        begin
        // Inicializa a variavel "i" com zero
        i := 0;
        while ((Result = nil) and (i<(Automato.Estados_Finais.Elementos.Count))) do
          begin
            if (TDesenha(automato.Estados_Finais.Elementos[i]).Caption = nome) then
               // Retorna o objeto estado encontrado
              Result:= TDesenha(automato.Estados_Finais.Elementos[i]);
            // Incrementa o indice da lista
            inc(i);
          end;
        end;
      end;
  end;
```

Quadro 22 – Função utilizada para retornar estados pesquisados

A *function* `Obter_Estado` apresentada no Quadro 22 é chamada pela *procedure* `Abrir_Arestas` (Quadro 24)**,** a qual percorre um "**arquivo.art**" e efetua a leitura das transições (arestas) que ligam os estados (vértices) do autômato.

```
Procedure TAutomatosFinitos.Abrir_Vertices;
// Declaração das variáveis
var
  Aux: TVetorVertice;
  Ind: Byte;
begin
  // Associar nome físico a nome lógico
  AssignFile(ArqVertice, Self.Caption);
  // Abrir o arquivo
  Reset(ArqVertice);
  Ind := 0;
  // Verifica se a quantidade de registros é maior que zero
  if FileSize(ArqVertice) > 0 then
    begin
      // Posiciona cabeça de leitura no inicio do arquivo
      Seek(ArqVertice, 0);
     // Seta a lista com o quantidade de registros existentes no arquivo
     SetLength(Aux.Vertices, (FileSize(ArqVertice) + 1));
     // Percorre o arquivo enquanto não chegar ao seu final
     while (not Eof(ArqVertice)) do
        begin
          Inc(Ind);
          read(ArqVertice, Aux.Vertices[ind]);
          FrmPrincipal.SBEstInicial.Down     := (Aux.vertices[ind].Estado = 'I') or
(Aux.vertices[ind].Caption =     'qf0');
          FrmPrincipal.SBEstTransitorio.Down := (Aux.vertices[ind].Estado in
['N','F']) and (Aux.vertices[ind].Caption <> 'qf0');
          // Atribui valor da coordenada X do vértice (Estado)
          XPos:= Aux.vertices[ind].Left;
          // Atribui valor da coordenada Y do vértice (Estado)
          YPos:= Aux.vertices[ind].Top;
          // Chama procedure a qual instância e desenha os estados na tela de edição
          FormClick(ScrBoxEditor);
          // Verifica se é o estado inicial ou é o primeiro estado
          if (Aux.vertices[ind].Caption = 'qf0') and (Ind = 1) then
            begin
              // Atributi o valor de esInicial para o estado inicial do AF
              Automato.Estado_Inicial.Estado := esInicial;
              FormDblClick(Automato.Estado_Inicial);
            end
          else if (Aux.vertices[ind].Estado = 'F') and (Aux.vertices[ind].Caption <>
'qf0')then
            FormDblClick(TDesenha(Automato.estados.Elementos.Last));
        end;
    end;
  // Fecha arquivo
  CloseFile(ArqVertice);
end;
```

Quadro 23 - Rotina responsável pela abertura e criação dos estados (vértices)

A *procedure* `Abrir_Vértices` (Quadro 23) efetua uma varredura no arquivo "**.af**" verificando a existência de estados (vértices) do AF. Cada estado encontrado no arquivo é instanciado em memória e desenhado na tela de edição do EAF. Após o processo de leitura dos estados (vértices) é chamada a *procedure* `Abrir_Arestas` apresentada no Quadro 24.

```
procedure TAutomatosFinitos.Abrir_Arestas;
// Declaração das variáveis
var
  Ind: Byte;
  Aux: TVetorAresta;
  strArq : string;
begin
  // Atribui o nome do arquivo a uma variável auxiliar
  StrArq := ExtractFileDir(self.Caption) + '\' +

copy(ExtractFileName(self.Caption),1,length(ExtractFileName(self.Caption))-3) +
'.art';
  // Associar nome físico a nome lógico
  AssignFile(ArqAresta, StrArq);
  // Verifica se o arquivo existe
  if (FileExists(StrArq)) then
    begin
      // Efetua a abertura do arquivo
      reset(ArqAresta);
      Ind:= 0;
      // Verifica se existe algum registro no arquivo
      if FileSize(ArqAresta) > 0 then
        begin
          // Pociona no inicio do arquivo
          Seek(ArqAresta,0);
          // Percorre o arquivo enquanto não encotrar o seu final
          while (not Eof(ArqAresta)) do
            begin
            SetLength(Aux.Arestas, (FileSize(ArqAresta) + 1));
            read(ArqAresta,Aux.Arestas[ind]);
            // Atribui o estado atual para uma variável global (EstadoSelecionado1)
            EstadoSelecionado1 := Obter_Estado(Aux.Arestas[ind].EstadoAtual);
            // Atribui o simbolo lido para uma variável global (SimboloGlobal)
            SimboloGlobal         := Aux.Arestas[ind].SimboloLido;
            if (Aux.Arestas[ind].EstadoAtual = Aux.Arestas[ind].ProximoEstado) then
               EstadoSelecionado2 := nil
            else
              EstadoSelecionado2 := Obter_Estado(Aux.Arestas[ind].ProximoEstado);
            // Procedure a qual desenha as transições na tela de edição do EAF
            FrmPrincipal.SBTransicaoClick(FrmPrincipal.SBTransicao);
            // Incrementa a variável utilizada como índice
            Inc(Ind);
            end;
          end;
       // Fecha arquivo
      CloseFile(arqAresta);
      SimboloGlobal:= '';
    end;
end;
```

Quadro 24 - Rotina responsável pela abertura e criação das transições (arestas)

A *procedure* `Abrir_Arestas` (Quadro 24) percorre toda estrutura do arquivo "**.art**" em busca de arestas (transições) para instanciá-las e apresentá-las na tela de edição do EAF.

3.3.2 Operacionalidade da implementação

O EAF apresenta uma interface de fácil usabilidade. Na parte superior do aplicativo

encontra-se a barra de ferramentas composta por botões, dentre os quais estão dispostos a opção de criação de um novo arquivo, capaz de armazenar a estrutura (grafo) de um autômato finito e também a opção para abrir um arquivo com extensão “.af”.

A Figura 31 apresenta a interface do EAF. Cada botão que aparece na imagem possui uma única funcionalidade. Os botões em destaque estão à disposição da ferramenta apenas para controle de interface, ou seja, efetuam a manipulação das janelas do EAF, sendo que, cada uma destas janelas pode conter um único autômato finito.

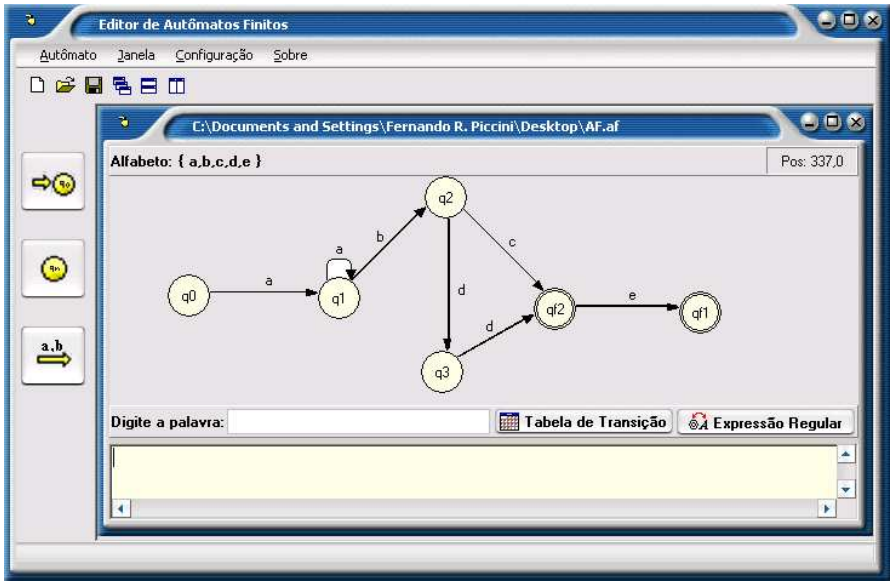


Figura 31 – Interface do EAF com a utilização da biblioteca RxLib

O EAF disponibiliza dez opções de interface utilizando a biblioteca `RxLib`, sendo que estas opções podem ser acessadas no menu “Configuração” localizado na parte superior da aplicação. Quando selecionada uma opção de interface o EAF efetua a chamada de um arquivo correspondente à opção escolhida, sendo que todos arquivos possuem a mesma extensão “.skn”. Observa-se que os arquivos “.skn” não foram implementados neste trabalho.

A Figura 32 mostra duas janelas dispostas verticalmente dentro do EAF. Isto vem a ser útil para o usuário quando o mesmo desejar visualizar dois ou mais autômatos finitos na mesma aplicação.

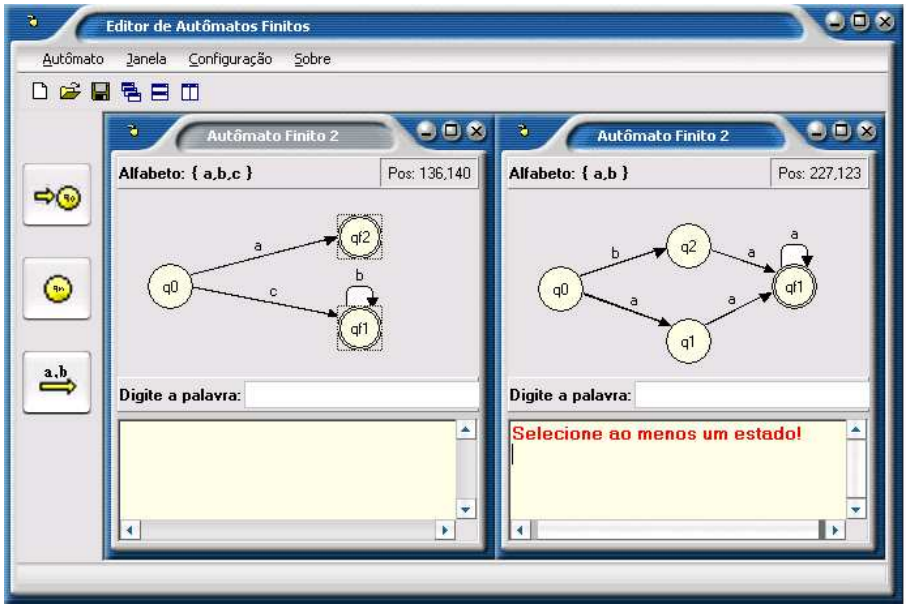


Figura 32 – Interface do EAF com duas telas de edição

Cada janela “filha” do EAF possui um botão com a descrição “Tabela de Trasição”, onde o mesmo agrega a funcionalidade de apresentação de uma tabela de transição correspondente ao autômato definido na janela correspondente. A Figura 33 mostra o EAF com apenas uma tela de edição.

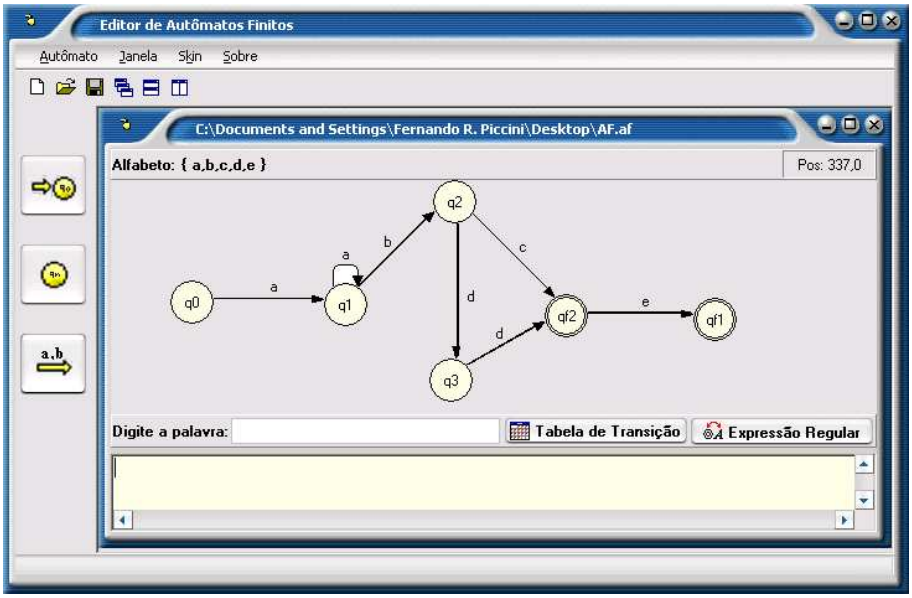


Figura 33 – Interface do EAF com apenas uma janela “filha”

A Figura 34 apresenta a tabela de transição correspondente ao autômato finito desenhado na Figura 33, conseguida a partir da opção “Tabela de Transição” localizada na parte inferior da ferramenta.

Figura 34 – Apresentação da tabela de transição

A tabela de transição mostrada na Figura 34 é formada por símbolos e estados, sendo que as colunas representam os símbolos do AF e as linhas os estados. Ainda, outros símbolos são usados, os quais são:

a) “ - ”: representa que não há nenhuma ligação entre um estado e o símbolo corrente;

b) “ --> ”: representa o estado inicial do AF, ou seja, é estado onde deve ser inicializado o processo de análise do AF;

c) “ * ”: indica que é um estado final do AF.

Com a inclusão do algoritmo de transformação de AF para ER foi acrescentado um novo botão “Expressão Regular” na janela de edição do EAF, sendo que o mesmo agrega a funcionalidade de apresentação de uma expressão regular correspondente ao autômato finito definido na janela de edição. A Figura 35 apresenta um AF desenhado na tela de edição antes da apresentação da ER.

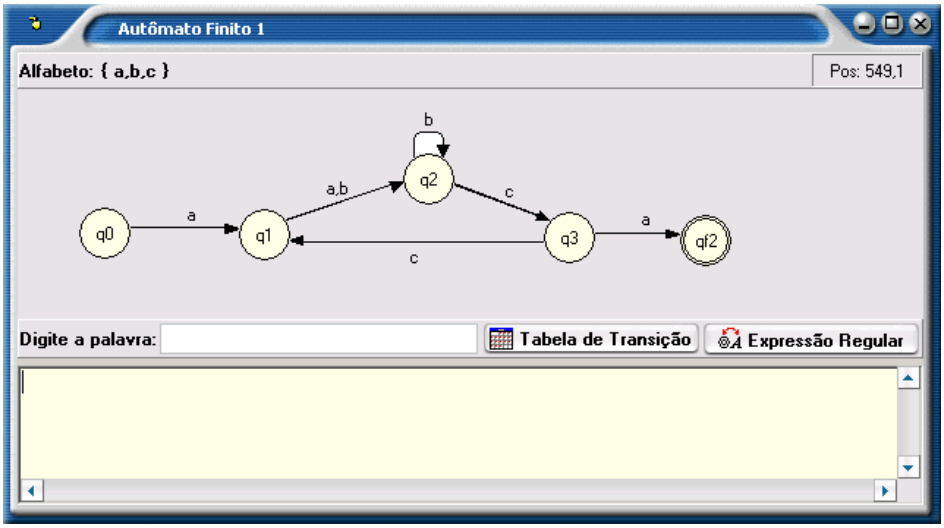


Figura 35 – AF utilizado para apresentação de uma expressão regular

A Figura 36 apresenta a expressão regular correspondente ao autômato finito desenhado na Figura 35.

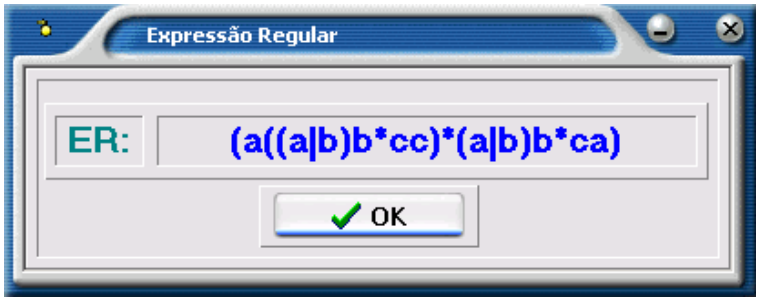


Figura 36 – Apresentação de uma expressão regular no EAF

A Figura 36 mostra a forma na qual as expressões regulares serão apresentadas para o usuário do EAF.

Observa-se também que o AF (Figura 35) foi desenhado com uma determinada forma e estrutura, sendo que o objetivo era utilizar todas as rotinas do algoritmo de transformação, ou seja, para gerar a ER (Figura 36) foi necessário utilizar as rotinas de: união, concatenação, fechamento e desdobramento ou concatenação múltipla.

3.4 RESULTADOS E DISCUSSÃO

Um dos resultados obtidos durante o desenvolvido do trabalho é mostrado nas Figuras 37 e 38, as quais ilustram a interface do EAF antes e depois da utilização da biblioteca RxLib.

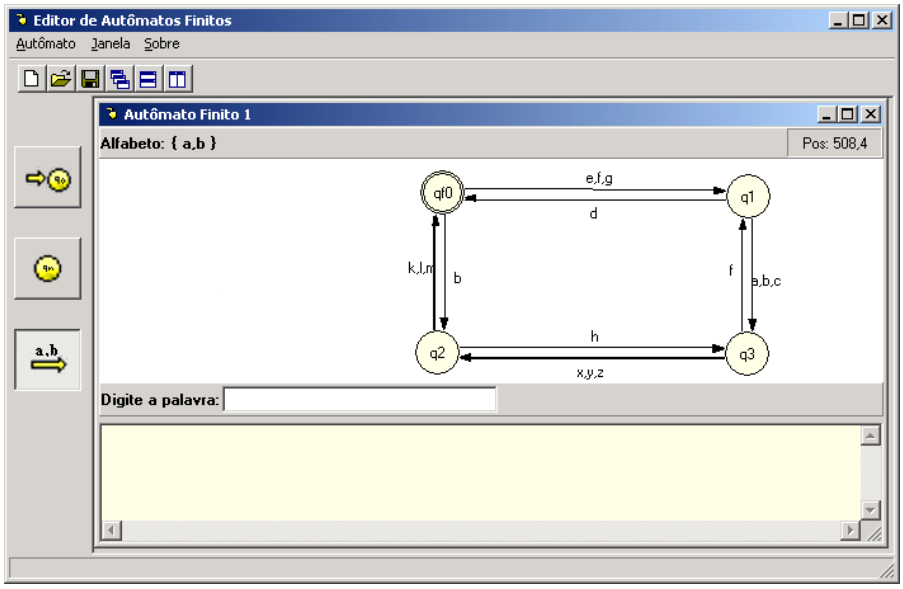


Figura 37 – Interface do EAF sem a biblioteca RxLib

A Figura 38 mostra o EAF após a utilização da Biblioteca RxLib. Observa-se que a interface gráfica ficou mais agradável e sofisticada.

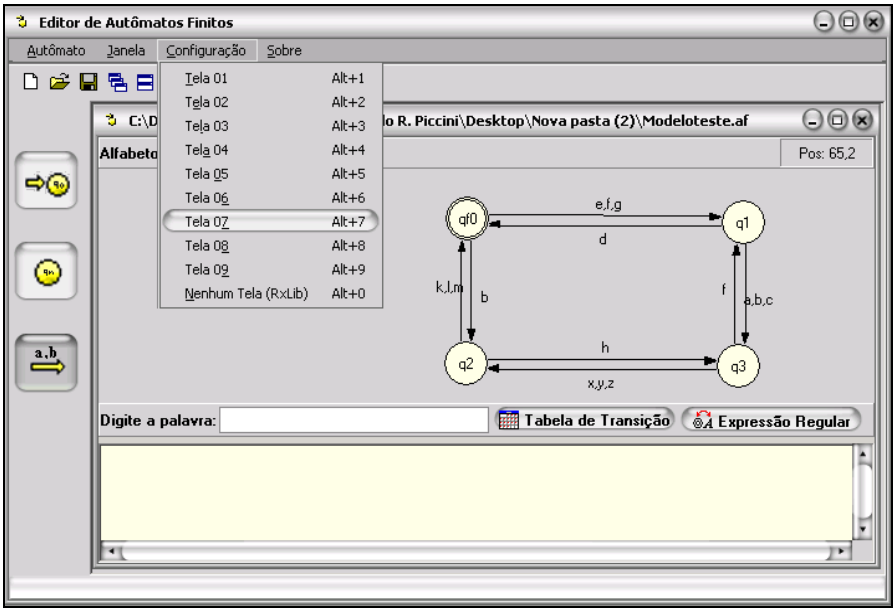


Figura 38 – Interface do EAF após a utilização da biblioteca RxLib

Discute-se também neste trabalho a forma efetuada para validação do algoritmo proposto por Silva (2006), para a qual foi utilizada a ferramenta para transformar uma expressão regular em um autômato finito equivalente (GLATZ, 2000).

O processo utilizado para a validação é descrito da seguinte forma:

a) desenhar AF_1 no EAF;

b) gerar ER_1 a partir do AF_1 desenhado na ferramenta;

c) submeter ER_1 gerada no EAF no protótipo de Glatz (2000) para gerar um AF_2;

d) com o AF_2 gerado no protótipo de Glatz (2000), redesenhar o mesmo no EAF;

e) gerar ER_2 a partir do AF_2 desenhado no EAF;

f) submeter ER_2 gerada no EAF no protótipo de Glatz (2000) para gerar um AF_3;

g) validar a equivalência do AF_2 e AF_3 na ferramenta de equivalência de Moore, descrita em Glatz (2000).

Com objetivo de exemplificar de forma ilustrativa o processo de validação, a Figura 39 mostra a tela de edição do EAF com um AF_1 desenhado, sendo que este AF_1 será submetido a validação.

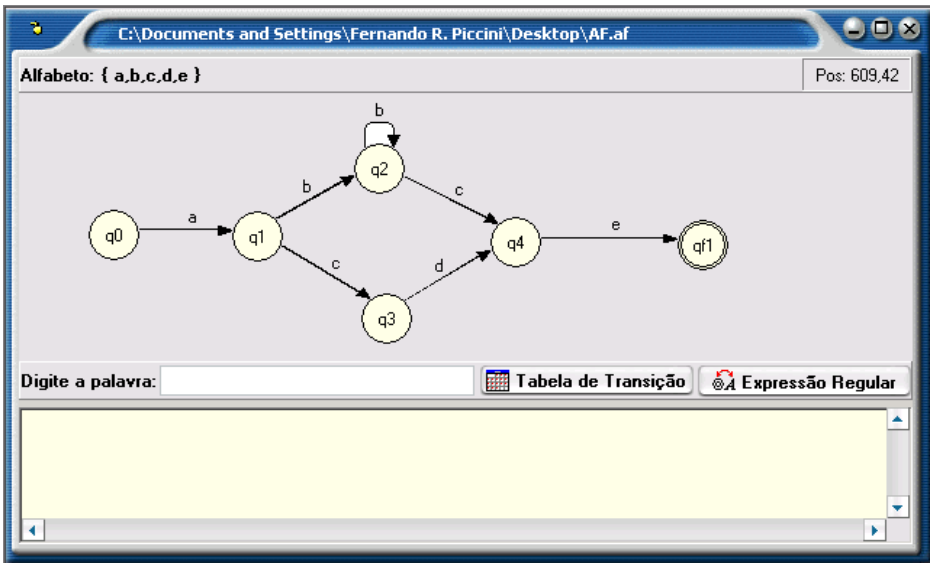


Figura 39 – AF_1 submetido para apresentação de uma expressão regular no EAF

Depois de desenhado o AF_1 na tela de edição do EAF gera-se a ER_1 apresentada na Figura 40.

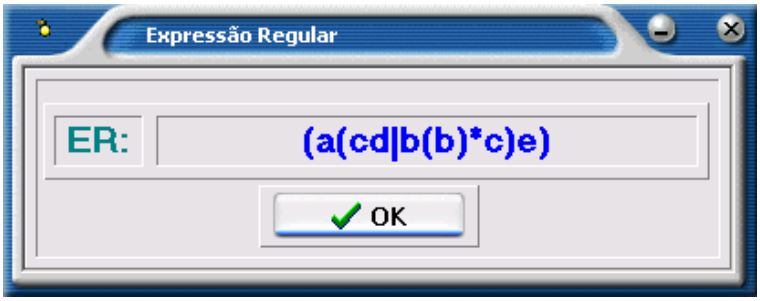


Figura 40 – ER_1 equivalente ao AF_1 apresentado na Figura 39

Com a ER_1 "**(a(cd|b(b)*c)e)**" gerada a partir do AF_1 (Figura 39), submete-se a mesma

ao protótipo de Glatz (2000) (Figura 41) para geração do AF_2 .

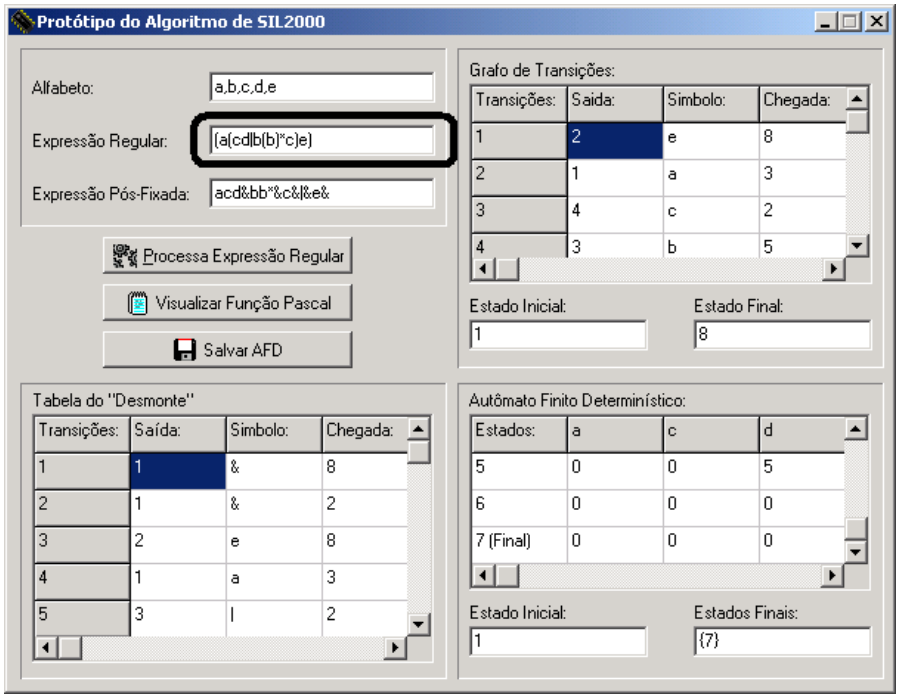


Figura 41 – ER_1 submetida a geração do AF_2 na ferramenta de Glatz (2000)

O protótipo gera um tabela de transição equivalente ao AF_2. Esta tabela localiza-se na parte superior direita da ferramenta de Glatz (2000). Na Figura 42 é mostrado o AF_2 corresponde a tabela de transição gerada (Figura 41).

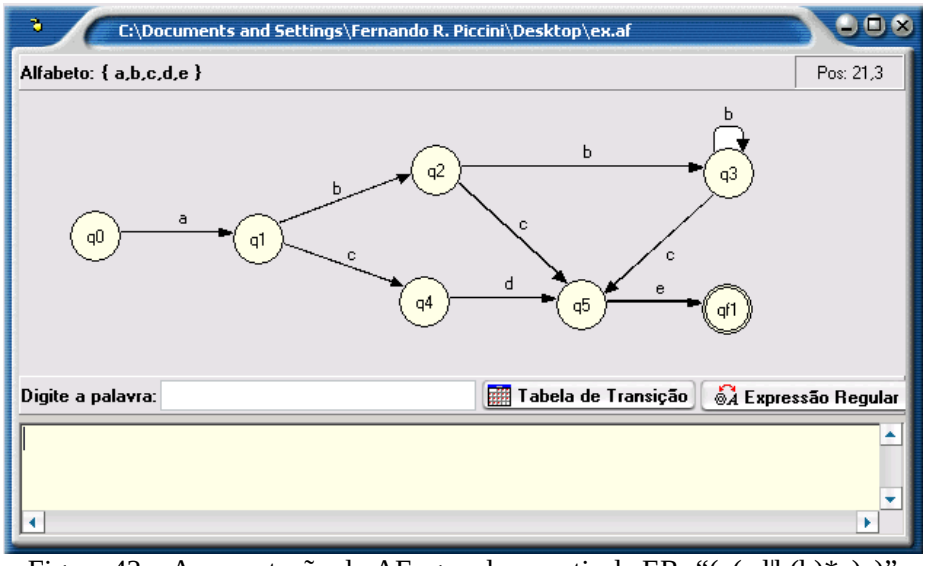


Figura 42 – Apresentação do AF_2 gerado a partir da ER_1 “(a(cd|b(b)*c)e)”

Após desenhar o AF_2 na tela de edição do EAF é necessário gerar a ER_2 correspondente ao AF_2 .

A Figura 43 mostra a ER_2 correspondente ao AF_2 desenhado no EAF.

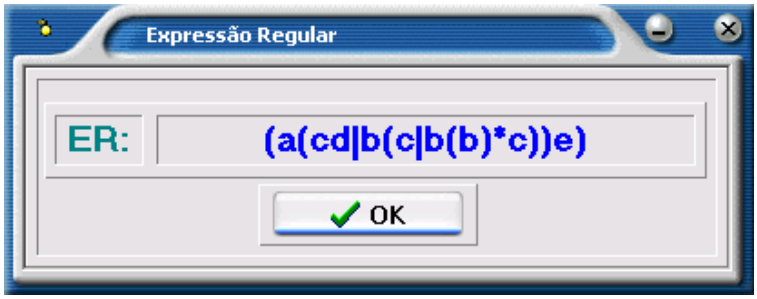


Figura 43 – ER_2 equivalente ao AF_2 apresentado na Figura 31

Após gerada a ER_2 pelo EAF tem-se a necessidade de informá-la no protótipo de Glatz (2000) para gerar o AF_3 (Figura 44).

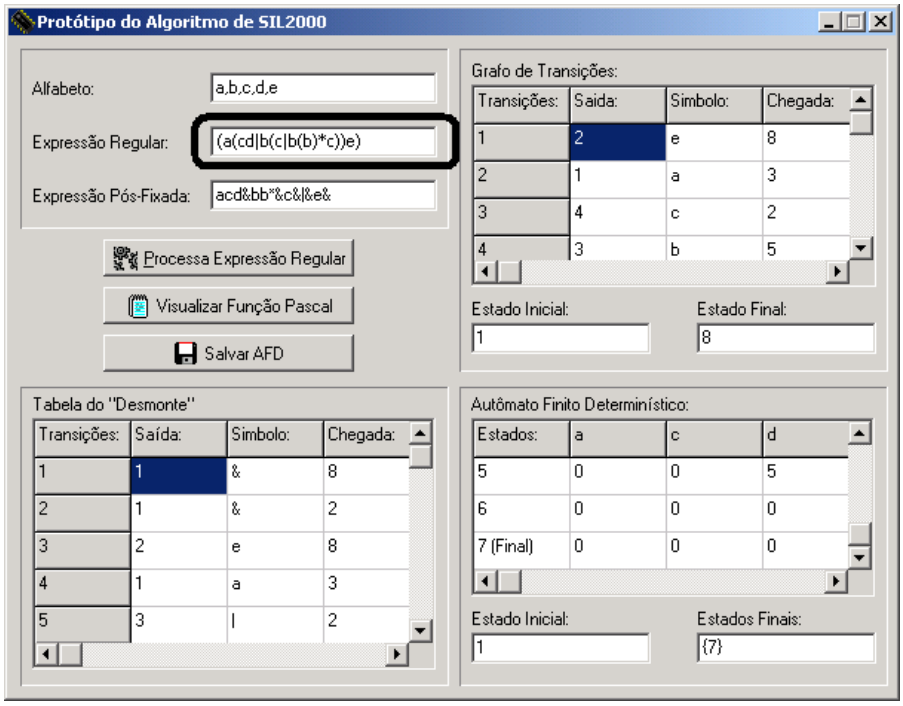


Figura 44 – ER_2 submetida para geração do AF_3 na ferramenta de Glatz (2000)

Após obter o AF_2 (Figura 42) e o AF_3 (Representado através da tabela de transição; Figura 44) é necessário validá-los na ferramenta de equivalência de Moore, descrita em Glatz (2000). A Figura 45 apresenta a validação dos dois AFs.

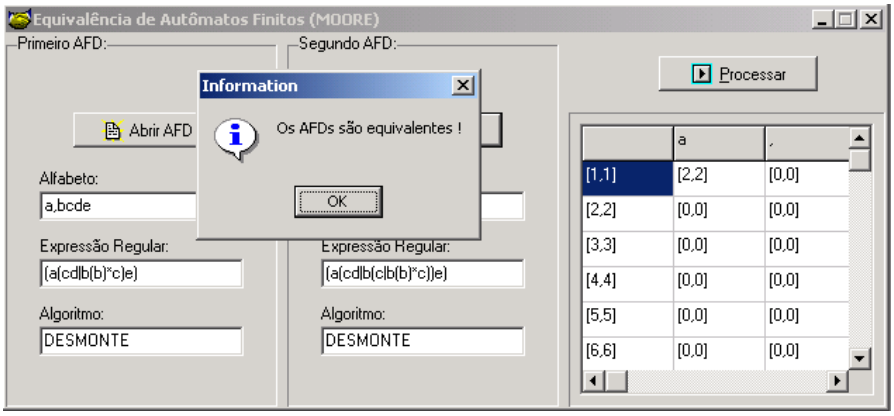


Figura 45 – Validação do AF_2 e AF_3 através da equivalência de Moore

4 CONCLUSÕES

Com o desenvolvimento deste trabalho foram atingidos os objetivos propostos, tendo como resultado final a implementação do algoritmo proposto em Silva (2006), que permite transformar um autômato finito em expressão regular. O algoritmo apresenta-se como uma nova solução para transformar um AF para ER, sendo que as avaliações realizadas neste trabalho comprovaram a validação do mesmo. Além do algoritmo de transformação foram implementadas funcionalidades que permitem salvar e abrir arquivos contendo estruturalmente um autômato finito, bem como a representação de um autômato através de uma tabela de transição no EAF.

Durante o desenvolvimento das atividades foram utilizadas duas ferramentas: *Enterprise Architect* para fazer a especificação e *Borland Delphi* 7 para implementar as novas funcionalidades no editor de autômatos finitos.

No decorrer do projeto foram identificadas algumas limitações que o EAF deixou de disponibilizar, entre elas: um AF não poderá ter mais do que um estado inicial; as configurações de interface só poderam ser alteradas antes de abrir a tela de edição e também não será possível excluir mais do que um estado ao mesmo tempo.

4.1 EXTENSÕES

Como possíveis extensões desse trabalho cita-se:

a) implementar algoritmo para minimizar a expressão regular, gerada pelo algoritmo proposto em Silva (2006);
b) determinar a complexidade do algoritmo proposto por Silva (2006);
c) comparar o algoritmo proposto por Silva (2006) com outro algoritmo, objetivando verificar qual deles é mais eficiente.

REFERÊNCIAS BIBLIOGRÁFICAS

DOGNINI, Marlon J.; RAABE, André L. A. EduLing: software educacional para linguagens regulares. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 14., 2003, Rio de Janeiro. **Anais...** Rio de Janeiro: UFRJ, 2003. p. 1-10. Disponível em: <http://www.nce.ufrj.br/sbie2003/publicacoes/paper24.pdf>. Acesso em: 08 set. 2005.

GLATZ, Ronald. **Protótipo para transformação de uma expressão regular para uma função equivalente em Pascal, utilizando dois algoritmos baseados no teorema de Klenne.** 2000. 104 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HOPCROFT, John E.; ULLMAN, Jeffrey D.; MOTWANI, Rajeev. **Introdução à teoria de autômatos, linguagens e computação.** Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus, 2002.

HOPCROFT, John. E; ULLMANN, Jeffrey. D. **Introduction to autômato theory, languages and computation**. Massachusets: Addison Wesley, 1979.

MENEZES, Paulo F. B. **Linguagens formais e autômatos**. Porto Alegre: Sagra-Luzzatto, 1998.

MORASTONI, Josiane P. **Editor de autômatos finitos**. 2002. 39 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVA, José R. V. da. **Proposta de um novo algoritmo para transformação de uma expressão regular em um autômato finito determinístico**. [Blumenau], 2000. Artigo não publicado – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVA, José R. V. da. **Proposta de um novo algoritmo para transformação de autômato finito em expressão regular**. [Blumenau], 2006. Artigo não publicado – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SONNINO, Bruno. **RxLib.** [Sem Local], 1998. Disponível em: <http://www.revolution.com.br/delphistop/rxrevp.htm>. Acesso em: 06 maio 2006.

TERADA, Routo. **Teoria da computação**: pesquisa em algoritmos e complexidade. [Brasília], [2005]. Disponível em: <http://www.cnpq.br/areas/sociedadeinformacao/protem-cc/fase2-03.html>. Acesso em: 28 nov. 2005.

VALE, Rodrigo F. **AutomataEditor:** uma ferramenta para o auxílio do desenvolvimento de autômatos. 1999. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém. Disponível em: <http://xote.akwan.com.br/~rodrigov/artigos/tcc.pdf>. Acesso em: 08 set. 2005.

WANGENHEIM, Aldo V. **Análise de algoritmos.** [Florianópolis], 1997. Disponível em: <http://www.inf.ufsc.br/~ine5384-hp/estrut07.html>. Acesso em: 16 jan. 2006.