UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALINE TEIXEIRA

GRAMÁTICAS REGULARES E EXPRESSÕES REGULARES:

UM AMBIENTE DE MANIPULAÇÃO NO AFLAB

CRICIÚMA, JULHO DE 2008

ALINE TEIXEIRA

GRAMÁTICAS REGULARES E EXPRESSÕES REGULARES:

UM AMBIENTE DE MANIPULAÇÃO NO AFLAB

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientadora: Profª. MSc. Christine Vieira Scarpato

CRICIÚMA, JULHO DE 2008

Ao meu pai, para que volte logo e participe comigo da festa de formatura.

AGRADECIMENTOS

À minha mãe, por ter a paciência de me ver várias tardes inteiras na mesa da cozinha sem ajudá-la a secar um prato de comida e por não me pedir para fazer isso.

Da mesma forma, ao meu namorado, que precisou fazer o sacrifício de ficar longe de mim nessas horas intermináveis dos finais de semana.

Às minhas sobrinhas, que mesmo atrapalhando a minha concentração com bagunças, choros e pedidos de colo em horas tão preciosas, me fazem sempre sorrir e ver que a vida certamente vale a pena.

Ao meu pai, minha irmã e meu irmão, que agradeço aqui pelo simples fato de existirem e pelos quais sou tão grata por tudo.

À minha querida orientadora, pelas conversas, pelas caronas, pelos e-mails, pelo apoio, pelo carinho, e, é claro, pela orientação.

À Deus e à Nossa Senhora, nos quais tenho fé e sempre agradeço pela vida, pela força para continuar, pela saúde e por tudo que tenho em minha vida.

"Os sete pecados capitais responsáveis pelas injustiças sociais são: riqueza sem trabalho, prazeres sem escrúpulos, conhecimento sem sabedoria, comércio sem moral, política sem idealismo, religião sem sacrifício e ciência sem humanismo."

Mahatma Gandhi

RESUMO

O AFLAB é um software que permite a construção de autômatos finitos determinísticos e não determinísticos em sua forma gráfica ou tabular, permitindo ainda o reconhecimento de sentenças por meio destes autômatos. Este trabalho apresenta uma continuação do AFLAB e consta da adição de dois módulos nesta ferramenta: um para transformação de expressões regulares em autômato finito e o outro para obtenção de autômato finito a partir de gramáticas regulares. Contudo, para que estes novos módulos pudessem ser implementados, foi necessário primeiramente fazer um estudo acerca de autômatos finitos, gramáticas regulares, expressões regulares, bem como sobre os algoritmos presentes na literatura que realizam a transformação desejada entre os formalismos. Este trabalho traz também a descrição dos módulos adicionados, detalhes sobre a implementação e demonstra suas funcionalidades. De uma forma geral os novos módulos funcionam perfeitamente, atendendo as expectativas e mantendo as características iniciais do AFLAB, com textos explicativos e interface simples e fácil de utilizar.

Palavras-Chave: Autômato Finito, Gramáticas Regulares, Expressões Regulares, Transformação entre Formalismos.

`

ABSTRACT

AFLAB is a software that allows the deterministic and non-deterministic finite automata construction on two different representation forms: diagrams and tables. It also allows recognizing sentences through these automata. This paper presents the addition of two new features on AFLAB: the transformation of a regular expression into a finite automaton and the conversion of a regular grammar into a finite automaton. However, to achieve the goals of the new modules, a study about finite automata, regular grammar, regular expressions and algorithms to do the transformation between formalisms was made. Both added modules are described here, as well as the implementation details about them and their features. In general, the new modules work perfectly, coming up to expectations and keeping the AFLAB characteristics, with explanatory texts and a simple and easy to use interface.

Key-words: Finite Automata, Regular Grammar, Regular Expressions, Transformation between Formalisms

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AFD	Autômato Finito Determinístico
AFDs	Autômatos Finitos Determinísticos
AFND	Autômato Finito Não Determinístico
AFNDs	Autômatos Finitos Não Determinísticos
AFNDε	Autômato Finito com Movimentos Vazios

SUMÁRIO

1 INTRODUÇÃO

Linguagem formal é um conjunto de palavras sobre o alfabeto de uma linguagem (MENEZES, 2005). As Linguagens Formais têm utilidade na área da matemática e em outras áreas que as utilizam: engenharia, física, química e computação. Elas estão presentes na computação tanto nos níveis mais altos de programação, quanto nos mais baixos e se tornaram importantes nessa área devido ao fato de que os profissionais trabalham com uma ou mais linguagens (C, Java, Pascal, entre outras) diariamente (VIEIRA, 2006).

Uma linguagem pode ser reconhecida por um autômato finito, que é uma máquina abstrata de estados finitos, com número pré-definido de estados. Ele tem apenas dois resultados possíveis: aceitação ou rejeição da entrada, sendo que uma linguagem aceita por um autômato finito é chamada Linguagem Regular. Existem duas notações bastante utilizadas para especificar este tipo de linguagem: expressões regulares e gramáticas regulares.

Expressão regular é um formalismo denotacional que define a linguagem por meio de uma expressão propriamente dita, a partir de conjuntos básicos e operações de concatenação, união e repetição. A gramática regular, por sua vez, utiliza regras para especificar a linguagem regular, sendo que por meio dessas se consegue gerar todas e somente as palavras da linguagem.

A partir de uma expressão regular é possível construir o autômato finito reconhecedor da linguagem que ela denota. Da mesma forma, por meio de uma gramática regular é possível obter o autômato finito que reconhece a linguagem por ela gerada.

Todos esses conceitos são abordados na disciplina de Linguagens Formais, que vem sendo estudada sem auxílio de ferramentas de software, o que limita professor e alunos ao uso de papel e caneta, resultando no estudo de autômatos finitos e linguagens regulares não muito complexos. Observou-se então, quão interessante seria ter uma ferramenta que pudesse ser utilizada em aula para manipulação desses autômatos e linguagens. Partindo dessa necessidade, foi criado um grupo de pesquisa em Linguagens Formais do curso de Ciência da Computação da Unesc, com o objetivo de desenvolver um software para auxiliar na disciplina no que diz respeito ao ensino de autômatos finitos. Esse software denomina-se AFLAB.

Considerando que a primeira versão do AFLAB não contempla o tratamento de expressões regulares, um dos objetos de estudo desta pesquisa é exatamente este: o desenvolvimento de um módulo na *shell* para obter autômatos finitos a partir de expressões regulares.

O início do módulo que trata de gramáticas regulares no AFLAB foi proposto por outro acadêmico, abrangendo apenas a geração da gramática regular a partir do autômato finito. Observou-se então a necessidade de aumentar a funcionalidade do software quanto a gramáticas regulares e para isso será acrescentada a transformação da gramática regular em autômato finito.

1.1 OBJETIVO GERAL

Desenvolver o módulo para manipulação de expressões regulares e ampliar o módulo para manipulação de gramáticas regulares na ferramenta AFLAB.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa são:

a) compreender autômato finito determinístico e não determinístico, gramáticas regulares e expressões regulares;
b) pesquisar algoritmos de transformação de gramáticas regulares e expressões regulares para autômatos finitos;
c) construir na *shell* AFLAB o módulo para transformação da expressão regular em autômato finito;
d) implementar na ferramenta o módulo para geração do autômato finito a partir da gramática regular.

1.3 JUSTIFICATIVA

A disciplina de Linguagens Formais hoje não dispõe de variedades de softwares que possam auxiliá-la no processo de aprendizagem. Unindo a necessidade de se ter um software para isso e a disponibilidade de alunos para desenvolvê-la, decidiu-se criar uma *shell* denominada AFLAB para manipulação de autômatos finitos na forma gráfica ou tabular.

A ferramenta foi concebida a partir da criação de autômatos finitos para reconhecimento de sentenças, porém há diversos assuntos relacionados a autômatos finitos além desse. Por essa razão, outros módulos passaram a ser colocados em pauta, como o módulo de expressões regulares e o módulo de gramáticas regulares.

Um dos assuntos tratados na disciplina de Linguagens Formais mostra que se *G* é uma gramática regular, então existe um autômato finito que reconhece a

linguagem gerada por ela, podendo este ser obtido a partir da gramática. O mesmo acontece com expressões regulares, sendo possível gerar o autômato finito caso a expressão regular seja conhecida (CRESPO, 2001). Esses tópicos hoje são ensinados na teoria e desenvolvidos no papel em sala de aula, mas poderão ser visualizados na prática com os módulos de gramáticas regulares e expressões regulares do AFLAB.

O cenário atual de falta de ferramentas na disciplina de Linguagens Formais no que diz respeito à manipulação de autômatos finitos, pode ser mudado aos poucos, conforme vai se aprimorando a *shell* AFLAB ao introduzir novos módulos à mesma. Com isso, os professores terão a possibilidade de expor exemplos mais complexos, não se limitando somente ao uso de papel e caneta, podendo ainda utilizar o software para corrigir eventuais exercícios. Quanto aos alunos, estes poderão ter um entendimento melhor de como tudo funciona e por se tratar de uma ferramenta de simples manipulação, a disciplina pode vir a se tornar mais interessante do ponto de vista de quem a está estudando.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho tem como objetivo o desenvolvimento de dois novos módulos na ferramenta AFLAB e está disposto em sete capítulos.

O Capítulo 2 aborda máquinas de estados, envolvendo autômatos finitos determinísticos, não determinísticos e com movimentos vazios. Este capítulo traz também uma comparação acerca do determinismo e não determinismo dos autômatos. O capítulo seguinte traz uma visão geral sobre gramáticas, detalhando gramáticas regulares, seus tipos e as regras que são utilizadas para converter uma gramática regular em um autômato finito. O Capítulo 4 abrange expressões regulares, demonstrando as

possíveis operações de serem realizadas e os passos práticos para a obtenção de um autômato finito a partir de uma expressão regular.

Os trabalhos correlatos são apresentados no Capítulo 5. O Capítulo 6 dispõe sobre a primeira versão da ferramenta AFLAB e sobre a metodologia, desenvolvimento e resultados obtidos dos novos módulos. Finalmente, na conclusão são apresentadas as considerações finais, bem como as dificuldades encontradas e sugestões para trabalhos futuros.

2 MÁQUINAS DE ESTADOS

Segundo Santos (2004, p. 11) “máquinas de estados são estruturas lógicas compostas por um conjunto de estados e um conjunto de regras de transição entre os estados”. Essas máquinas podem auxiliar no controle de processos, descrevendo as situações em que eles se encontram a cada momento. Tais situações são representadas pelos estados da máquina, enquanto as condições necessárias para o processo sair de uma situação e partir para a próxima, dependem das regras de transição.

O funcionamento da máquina de estados depende de dados de entrada, que combinados com as regras de transição, gerarão dados de saída como resultados. Para iniciar seu processamento, deve ser definido um estado inicial e um ou vários estados finais. A combinação de um dado de entrada com o estado corrente pode gerar ou não uma transição, de acordo com as regras. No término de sua execução, a máquina aceita a entrada a que foi submetida se o estado corrente for um estado final, caso contrário, ela a rejeita (SANTOS, 2004).

As máquinas de estados são um conceito importante em várias áreas da ciência. Elas têm um papel importante na Teoria da Computação, onde são utilizadas na especificação e construção do analisador léxico de compiladores, em softwares para examinar grandes corpos de texto, com o propósito de encontrar palavras ou frases, e em outras áreas que envolvem o reconhecimento de linguagens (HOPCROFT; ULLMAN; MOTWANI, 2002; SANTOS, 2004).

2.1 AUTÔMATO FINITO

Autômato finito é uma máquina de estados com um número finito e predefinido de estados, que pode ser visto como sendo constituído basicamente por três partes (MENEZES, 2005):

a) **fita**: divida em células, contém a informação a ser processada (entrada);

b) **unidade de controle**: indica o estado atual da máquina. Sua unidade de leitura (cabeça da fita) acessa uma célula por vez e move-se sempre para a direita;

c) **programa**, **função programa** ou **função de transição**: função para comandar as leituras e definir o estado da máquina.

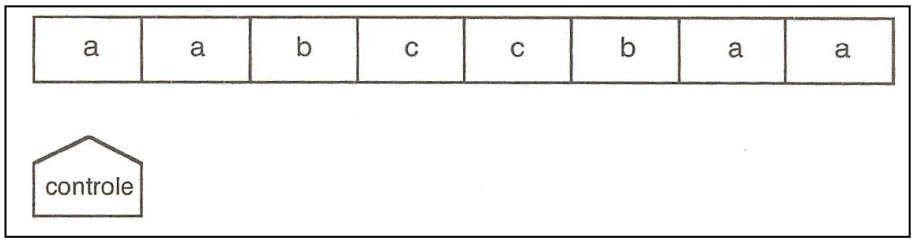


Figura 1. Autômato finito visto como uma máquina com controle finito
Fonte: MENEZES, P. (2005)

Como é possível observar na Figura 1, a fita é divida em um número finito de células e cada célula armazena um símbolo do alfabeto de entrada. A unidade de controle fica inicialmente posicionada no início da fita (célula mais à esquerda). Ela acessa uma célula por vez, lendo seu símbolo e movendo-se para a célula seguinte à direita, com o propósito de realizar a próxima leitura. O funcionamento de um autômato finito inicia a partir do seu estado inicial e é baseado na aplicação sucessiva das transições de acordo com cada símbolo de entrada lido, da esquerda para a direita (MENEZES, 2005).

Um autômato finito sempre terá uma condição de parada, pois qualquer palavra que ele vir a processar, será finita, não existindo então uma possibilidade de ciclo infinito (*loop*). A parada de seu processamento acontece de duas maneiras (MENEZES, 2005):

a) aceitação da entrada:
   - a fita inteira é processada e o último símbolo lido leva a um estado final;

b) rejeição da entrada:
   - a fita inteira é processada e o último símbolo lido não leva a um estado final,
   - em alguma posição da fita, é encontrado um símbolo não pertencente ao alfabeto do autômato, ou
   - não existe uma transição para um símbolo lido.

Se ao final do processamento de um autômato finito, a linguagem for aceita por ele, então ela é chamada de linguagem regular. Por outro lado, se não houver algum autômato finito que a reconheça, então ela não pertence a essa classe de linguagens (VIEIRA, 2006).

Uma forma de representação de um autômato finito é o diagrama de estados, também chamado de diagrama de transição. Segundo Menezes (2005) esta representação é da seguinte forma:

a) os círculos (nodos) representam os estados do autômato;

b) as arestas entre dois nodos são as transições entre eles. Estas arestas estarão sempre direcionadas e rotuladas com um símbolo de entrada do alfabeto, como representado na Figura 2;

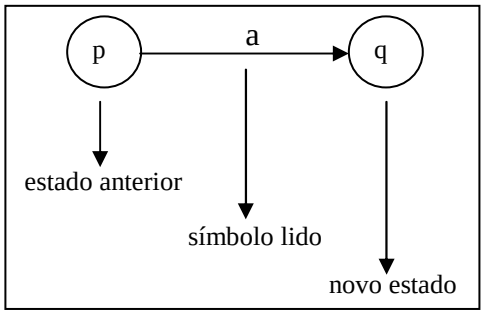


Figura 2. Representação de uma transição em um autômato
Fonte: MENEZES, P. (2005)

c) uma flecha vinda de fora do diagrama indica o estado inicial, como pode ser visto na Figura 3;

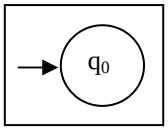


Figura 3. Representação do estado inicial de um autômato
Fonte: MENEZES, P. (2005)

d) um círculo duplo ou de maior espessura que os demais caracteriza um estado final, representado na Figura 4.

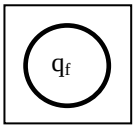


Figura 4. Representação de um estado final de um autômato
Fonte: MENEZES, P. (2005)

A Figura 5 mostra a representação de um autômato finito na forma de diagrama, com todos os componentes acima citados.

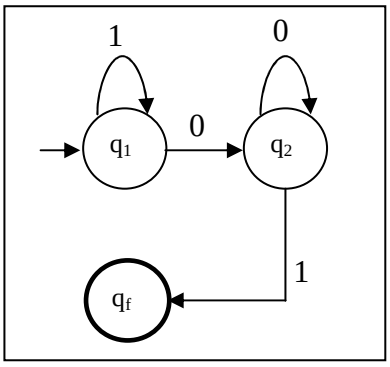


Figura 5. Representação de um autômato finito na forma de um diagrama

Uma outra forma de representar um autômato finito é por meio de sua tabela de transições, que é uma representação tabular das transições do autômato, onde cada transição recebe dois argumentos (estado atual e símbolo de entrada) e retorna um valor (próximo estado). Por meio desta tabela de transições, é possível retirar todas as informações necessárias para se especificar um autômato finito (HOPCROFT; ULLMAN; MOTWANI, 2002).

A tabela de transições de um autômato finito possui as seguintes características:

a) a primeira linha contém os símbolos do alfabeto de entrada do autômato finito;
b) a primeira coluna contém os estados;
c) o cruzamento das linhas com as colunas representam as transições do autômato;
d) o estado inicial é marcado com uma seta;
e) os estados finais são marcados com um asterisco (*).

Estas características podem ser melhor observadas na Tabela 1, que mostra o autômato finito da Figura 5 representado por meio de sua tabela de transições.

Tabela 1. Representação de um autômato finito por meio de sua tabela de transições

δ	0	1
→**q_1**	q_2	q_1
q_2	q_2	q_3
***q_3**	-	-

2.2 AUTÔMATOS FINITOS DETERMINÍSTICOS

Segundo Vieira (2006) um autômato finito determinístico (AFD) é definido como uma quíntupla:

$$M_D = (E, \Sigma, \delta, q_0, F)$$

Onde:

a) E: conjunto finito de estados;

b) Σ: alfabeto ou símbolos de entrada;

c) q_0: estado inicial, pertencente a E;

d) F: conjunto de estados finais, também pertencentes a E;

e) δ: função de transição[1] ou função programa: $\delta : E \times \Sigma \rightarrow E$. Essa função demonstra que o resultado da combinação entre um estado pertencente a E e um símbolo de entrada pertencente a Σ, resulta em um estado novo, ou não, de E. Exemplo: $\delta (q_0, a) = q_1$.

A Tabela 2 representa um autômato por meio de sua tabela de transições:

Tabela 2. Tabela de transições do autômato M = ({a, b}, {q_0, q_1, q_f}, δ, q_0, {q_f})

δ	a	b
→q_0	q_1	q_0
q_1	q_1	q_f
***q_f**	q_f	q_f

A fim de simular o funcionamento deste autômato finito, supõe-se uma entrada *aabab*, então o processo é iniciado no estado q_0 e o primeiro símbolo lido é o

[1] Determina sempre o próximo estado do autômato, dependendo do símbolo lido (MENEZES, 2006).

mais à esquerda, ou seja, o *a*. De acordo com sua função programa, com a combinação de q_0 com o símbolo *a* um novo estado é atingido, neste caso, o q_1. Então o próximo símbolo *a* é lido, dessa vez a combinação de *a* com q_1, faz com que o autômato permaneça no mesmo estado. Quando *b* é lido, o estado q_f é alcançado e independente dos próximos símbolos de entrada, não haverá mais trocas de estados, pois seja *a* ou *b*, as transições de q_f sempre apontarão para q_f. Então o final da fita é alcançado e como q_f é um estado final, diz-se que a entrada é aceita.

A Figura 6 mostra a representação na forma de um diagrama do autômato finito exemplificado anteriormente.

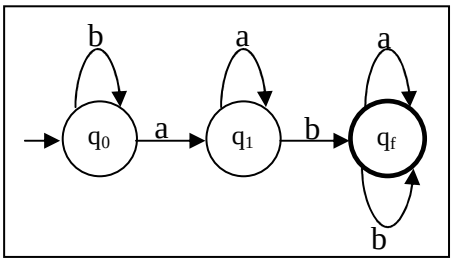


Figura 6. Representação do autômato finito M na forma de um diagrama
Fonte: MENEZES, P. (2005)

Segundo Vieira (2006) se a função de transição do autômato finito mapeia a combinação do estado corrente com um símbolo de entrada para um, e apenas um novo estado (podendo ser o atual), então esse autômato é dito determinístico. Desta forma, para aceitar uma determinada palavra de entrada, é possível apenas seguir uma única seqüência de estados partindo-se do estado inicial do autômato.

2.3 AUTÔMATOS FINITOS NÃO DETERMINÍSTICOS

Assim como o AFD, o autômato finito não determinístico (AFND) também é definido por uma quíntupla (HOPCROFT; ULLMAN; MOTWANI, 2002):

$$M_N = (E, \Sigma, \delta, q_0, F)$$

Onde:

a) E: conjunto finito de estados;

b) Σ: conjunto finito de símbolos de entrada;

c) δ: função de transição ou função programa: $\delta : E \times \Sigma \rightarrow 2^E$;

d) q_0: estado inicial, pertencente a E;

e) F: conjunto de estados finais, também pertencentes a E.

Os componentes de um AFND são os mesmos de um AFD, porém o não determinismo é dado pela sua função de transição, que neste caso retornará um conjunto (vazio ou não) de estados para cada combinação de estado corrente com o símbolo de entrada, ao contrário do AFD, que indica apenas um próximo estado possível, para cada símbolo lido (HOPCROFT; ULLMAN; MOTWANI, 2002).

Devido ao não determinismo, várias computações podem ser efetuadas para aceitar uma determinada palavra de entrada, ou seja, diferentes seqüências de estados podem ser seguidas para que se chegue à aceitação (VIEIRA, 2006).

Sendo que a diferença entre AFDs e AFNDs é na sua função de transição, então o processamento da entrada ocorre de forma semelhante, aplicando a função programa para cada símbolo de entrada da esquerda para a direita, até ocorrer uma condição de parada.

Exemplo: $M = (\{q_0, q_1, q_2, q_f\}, \{a, b\}, \delta, q_0, \{q_f\})$, onde o autômato validará se a entrada possui uma seqüência *aa* ou *bb* e terá a função programa conforme ilustra a Tabela 3.

Tabela 3. Tabela de transições do autômato finito

δ	**a**	**b**
→ **q_0**	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_f\}$	-
q_2	-	$\{q_f\}$
***** **q_f**	$\{q_f\}$	$\{q_f\}$

O não determinismo do autômato da Tabela 3 está no fato de que para o estado q_0 com símbolos de entrada *a* ou *b*, o autômato pode assumir dois caminhos. Para o símbolo *a*, por exemplo, ele pode permanecer no estado q_0 ($\delta (q_0, a) = q_0$) ou avançar para o estado q_1 ($\delta (q_0, a) = q_1$). Isso pode ser observado claramente na Figura 7, que mostra o autômato na forma de um diagrama.

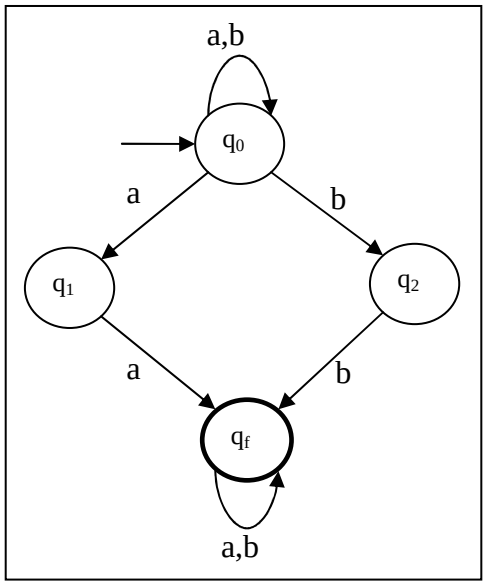


Figura 7. Representação do autômato finito M na forma de um diagrama
Fonte: MENEZES, P. (2005)

Uma semântica utilizada por Menezes (2005) para explicar a característica não determinística, diz que o AFND quando visto como uma máquina composta por fita, unidade de controle e programa, admite alguns estados alternativos, como se sua

unidade de controle se multiplicasse para cada alternativa e passasse a processar de forma independente das demais. Desta forma, o processamento de cada uma das opções não interfere no estado, símbolo de entrada e posição da cabeça da fita das demais opções assumidas.

Outra semântica para explicar o processamento de uma máquina não determinística é que quando houver dois caminhos possíveis, ela precisa assumir um caminho de cada vez. Caso o primeiro caminho percorrido não seja capaz de aceitar a sentença, então o autômato percorre outros caminhos até que a sentença seja reconhecida ou até que se esgotem todos as alternativas possíveis e só então se pode afirmar que a sentença não é aceita pelo autômato finito (GAIDZINSKI, 2007).

2.4 AUTÔMATOS FINITOS DETERMINÍSTICOS VERSUS AUTÔMATOS FINITOS NÃO DETERMINÍSTICOS

Em Linguagens Formais, determinismo não é sinônimo de aumento no poder de reconhecimento de linguagens de uma classe de autômato, tanto é que qualquer AFND pode ser simulado por um AFD, pois é possível construir um AFD equivalente a um AFND que realiza o mesmo processamento (MENEZES, 2005).

Segundo Aho; Sethi; Ullman (1995) existe uma diferença entre tempo e espaço no que diz respeito ao determinismo e ao não determinismo, pois os AFDs permitem a construção de reconhecedores mais rápidos, mas em contrapartida eles também podem ser muito maiores do que um AFND equivalente.

Uma das desvantagens dos AFDs é exatamente que o número de estados sempre vai ser igual ou maior ao número de estados de seu AFND equivalente. No pior caso, em que um AFND possui n estados, um AFD para a mesma linguagem pode ter 2^n

estados (HOPCROFT; ULLMAN; MOTWANI, 2002). A diferença no número de estados de um AFND equivalente a um AFD pode ser observada na Figura 8.

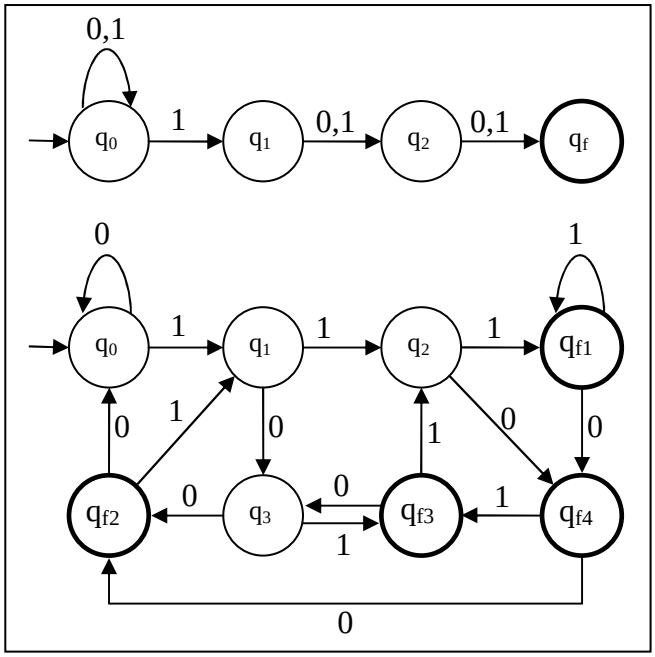


Figura 8. AFD e AFND para a linguagem $\{0,1\}^{*}$ {1}{0,1}{0,1}
Fonte: VIEIRA, N. (2006)

Outra característica de um AFND é que ele possui algumas vezes uma implementação mais complexa que um AFD equivalente, porém possui a capacidade de representar mais claramente algumas linguagens regulares. Já um AFD, tem uma implementação mais simples, porém dependendo do seu número de estados, deixa a representação de algumas linguagens regulares não muito claras (VIEIRA, 2006). Essa característica pode ser observada na Figura 9.

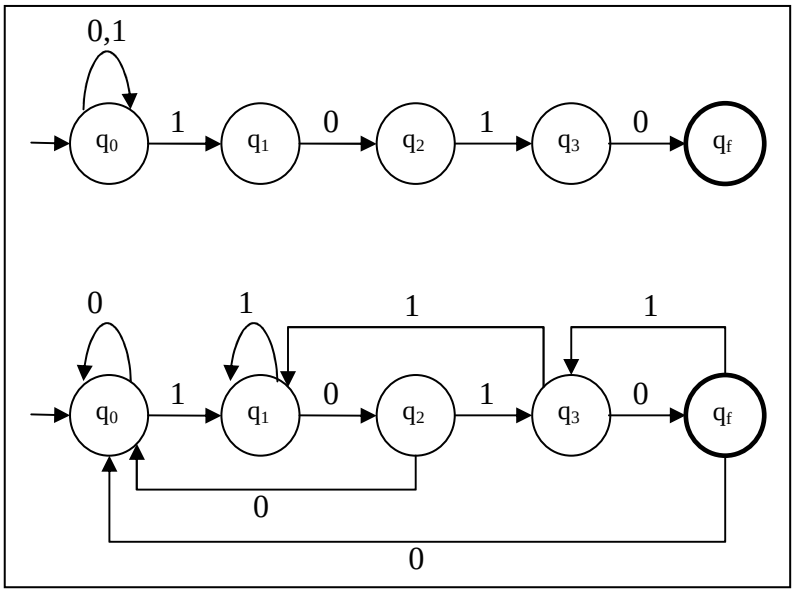


Figura 9. AFD e AFND para a linguagem $\{0,1\}^*$ {1010}
Fonte: VIEIRA, N. (2006)

A partir dessas características, é possível observar que o não determinismo não acrescenta nada a um autômato finito, a não ser praticidade.

2.5 AUTÔMATO FINITO COM MOVIMENTOS VAZIOS

Os autômatos finitos com movimentos vazios (AFNDε), como o próprio nome sugere, podem ter movimentos vazios, ou seja, uma transição em que nenhum símbolo da fita é lido (MENEZES 2005). Isso acontece porque um AFND pode fazer uma transição de forma espontânea, não precisando necessariamente receber um símbolo de entrada (HOPCROFT; ULLMAN; MOTWANI, 2002).

Assim como os AFDs e AFNDs, o AFNDε é uma quíntupla:

$$M_\varepsilon = (E, \Sigma, \delta, q_0, F)$$

Onde todos os componentes são iguais ao de um AFND, exceto a sua função de transição, que agora passa a receber como argumento um estado de E e um elemento de

$\Sigma \cup \varepsilon$, ou seja, um símbolo de entrada ou um movimento vazio, representado pelo símbolo ε (HOPCROFT; ULLMAN; MOTWANI, 2002).

Porém, assim como o não determinismo, essa característica não aumenta ou diminui seu poder de reconhecimento de linguagens, apenas traz algumas vantagens como maior facilidade na construção e demonstração dos autômatos finitos no estudo de Linguagens Formais (MENEZES 2005). O exemplo de um AFNDε é mostrado na Figura 10 por meio do seu diagrama de estados.

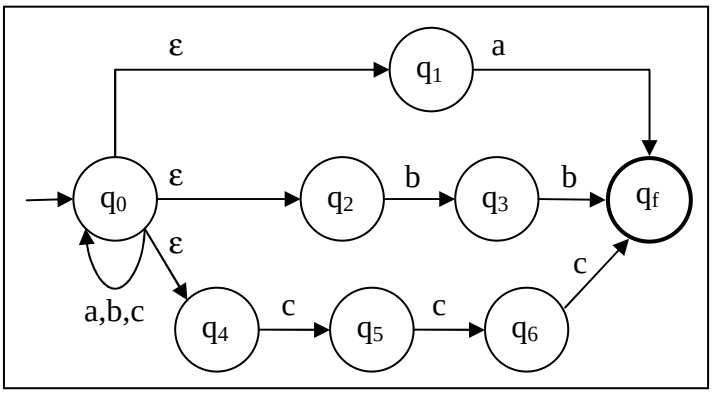


Figura 10. Autômato finito com movimentos vazios
Fonte: MENEZES, P. (2005)

Quando um movimento vazio é encontrado ao longo do caminho de um autômato finito, é como se ele fosse invisível, pois nada acrescenta na palavra formada ao longo do processamento (HOPCROFT; ULLMAN; MOTWANI, 2002). Ao se encontrar num estado que possua uma próxima transição com movimento vazio, o autômato precisa determinar todos os estados que podem ser atingidos exclusivamente por movimentos vazios e somente depois continuar o processamento (MENEZES 2005).

A computação deste tipo de autômato é equivalente à do AFND, portanto um pode ser simulado pelo outro, o que pode ser observado na Figura 11.

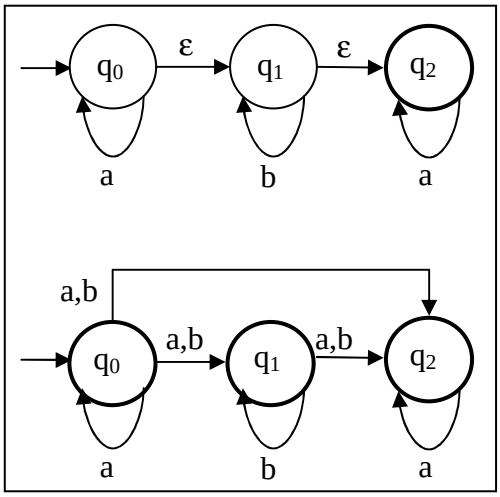


Figura 11. AFND e AFNDε equivalentes
Fonte: MENEZES, P. (2005)

3 GRAMÁTICAS

As gramáticas são um formalismo utilizado para a definição de linguagens. Elas são conjuntos finitos de regras que aplicadas sucessivamente, geram palavras, sendo que o conjunto de todas essas palavras define a linguagem (MENEZES, 2005; VIEIRA, 2006).

As regras de uma gramática demonstram como podem ser geradas as palavras da linguagem e são pares ordenados escritos na forma $u \rightarrow v$, onde u e v são palavras que podem conter símbolos de dois alfabetos: um com símbolos não terminais, que auxiliam na geração das palavras e outro com símbolos terminais, que nada mais são do que o próprio alfabeto da linguagem definida (VIEIRA, 2006).

Chama-se derivação a aplicação de uma regra, sendo que quando as regras são aplicadas sucessivamente, todas as palavras da linguagem representadas pela gramática são geradas. Portanto, derivação nada mais é do que a substituição de uma subpalavra seguindo uma regra de produção (VIEIRA, 2006).

Exemplo de regra: $aAB \rightarrow baA$, onde os não terminais aparecem em letras maiúsculas e os terminais em letras minúsculas.

Uma regra especifica que a palavra à esquerda da flecha pode ser substituída pela palavra à direita, dessa forma, no exemplo acima citado, a palavra *aAB* pode ser substituída por *baA*, assim, a partir de *aABBaAB* pode-se obter *bbaAbaA* (VIEIRA, 2006):

aABBaAB ⇨ **baA**BaAB (aplicando-se a regra aAB → baA)

⇨ b**baA**aAB (aplicando-se a regra aAB → baA)

⇨ bbaA**baA** (aplicando-se a regra aAB → baA).

Segundo Menezes (2005) uma gramática pode ser chamada também de Gramática de Chomsky, sendo uma quádrupla ordenada:

$$G = (V, T, P, S)$$

Onde:

a) V: conjunto de símbolos não terminais;

b) T: conjunto de símbolos terminais (alfabeto);

c) P: regras de produção;

d) S: símbolo inicial (é um símbolo não terminal).

O símbolo inicial da gramática indica onde iniciará a derivação. O processo de derivação de uma palavra alcançará um ponto em que não haverá mais não terminais na palavra, isso é conhecido como sentença. O conjunto dessas sentenças geradas pela gramática define a linguagem por ela definida (VIEIRA, 2006).

Exemplo: A sentença *aaabbbccc* pertence à linguagem gerada por G = ({S, A, C}, {a, b c}, P, S), onde P contém as regras:

1. S → aAbc
2. A → aAbC
3. A → λ
4. Cb → bC
5. Cc → cc

S ⇨ aAbc (regra 1)

⇨ aaAbCbc (regra 2)

⇨ aaaAbCbCbc (regra 2)

⇨ aaabCbCbc (regra 3)

⇨ aaabbCCbc (regra 4)

⇨ aaabbCbCc (regra 4)

⇨ aaabbCbcc (regra 5)

⇨ aaabbbCcc (regra 4)

⇨ aaabbbccc (regra 5).

Existem quatro classes gramaticais que geram quatro classes correspondentes de linguagens (MENEZES, 2005):

a) **gramática irrestrita ou do tipo 0**: trata-se de uma gramática cujas regras de produção não possuem nenhuma restrição. A linguagem gerada por este tipo de gramática é chamada Linguagem Recursivamente Enumerável;

b) **gramática sensível ao contexto ou do tipo 1**: é uma gramática cujas regras de produção são todas da forma $\alpha \rightarrow \beta$, onde:

   - β pode ser composto por símbolo(s) terminais e / ou símbolo(s) não terminais da gramática, bem como pelo vazio (ε) $(V \cup T)^*$;
   - α pode ser composto por símbolo(s) terminais e / ou símbolo(s) não terminais da gramática $(V \cup T)^+$, sendo que $|\alpha| \leq |\beta|$, ou seja, a palavra à direita da flecha será sempre maior que a palavra à esquerda da flecha, exceto quando houver uma produção do tipo $S \rightarrow \varepsilon$, onde para este caso S não poderá aparecer no lado direito de alguma produção.

   A linguagem gerada por este tipo de gramática é chamada Linguagem Sensível ao Contexto ou Linguagem Tipo 1;

c) **gramática livre do contexto ou do tipo 2**: trata-se de uma gramática cujas regras de produção são da forma $A \rightarrow \alpha$, onde:

   - A é um único símbolo não terminal da gramática, pertencente a V;

- α pode ser composto por símbolo(s) terminais e / ou símbolo(s) não terminais da gramática, bem como pelo vazio (ε) $(V \cup T)^*$.

A linguagem gerada por este tipo de gramática é dita Linguagem Livre do Contexto ou Linguagem Tipo 2;

d) **gramática regular ou do tipo 3**: é uma gramática cujas regras de produção possuem mais restrições do que as gramáticas apresentadas anteriormente. Este tipo de gramática será aprofundado a seguir, pois são estas gramáticas que possuem as regras que especificam o tipo de linguagem aceita por um autômato finito: a Linguagem Regular ou Linguagem Tipo 3.

3.1 GRAMÁTICAS REGULARES

Assim como um autômato finito é um tipo de reconhecedor de uma linguagem regular, uma gramática regular especifica esta linguagem por meio de um gerador, ou seja, esse tipo de gramática mostra como gerar todas e apenas as palavras de uma linguagem regular (VIEIRA, 2006).

As restrições das regras de produção desse tipo de gramática permitem definir exatamente a classe das linguagens regulares. Tais restrições são representadas pelos quatro diferentes tipos de gramáticas lineares. Diz-se então que uma gramática é regular se suas regras possuem o formato de uma das gramáticas lineares, ou seja, se ela é uma gramática linear (MENEZES, 2005).

3.1.1 Gramáticas Lineares

As gramáticas lineares representam as restrições das regras de produção de uma gramática para que ela possa ser considerada gramática regular (MENEZES, 2005).

Nas definições e exemplos a seguir, as letras em maiúsculo são símbolos não terminais (elementos de V) e as letras em minúsculo são símbolos terminais (elementos de T).

3.1.1.1 Gramática Linear à Direita

Uma gramática é dita linear à direita (GLD) se suas regras de produção são da seguinte forma (MENEZES, 2005):

$$A \rightarrow wB \text{ ou } A \rightarrow w$$

Exemplo: $S \rightarrow aA$

$A \rightarrow baA \mid \varepsilon$

3.1.1.2 Gramática Linear à Esquerda

Uma gramática é dita linear à esquerda (GLE) se suas regras de produção são da seguinte forma (MENEZES, 2005):

$$A \rightarrow Bw \text{ ou } A \rightarrow w$$

Exemplo: $S \rightarrow Sba \mid a$

3.1.1.3 Gramática Linear Unitária à Direita

Uma gramática é dita linear unitária à direita (GLUD) se suas regras de produção são da seguinte forma (MENEZES, 2005):

$$A \rightarrow wB \text{ ou } A \rightarrow w, \text{ onde } w \leq 1$$

Exemplo: $S \rightarrow aA$

$A \rightarrow bB \mid \varepsilon$

$B \rightarrow aA$

3.1.1.4 Gramática Linear Unitária à Esquerda

Uma gramática é dita linear unitária à esquerda (GLUE) se suas regras de produção são da seguinte forma (MENEZES, 2005):

$$A \rightarrow Bw \text{ ou } A \rightarrow w, \text{ onde } w \leq 1$$

Exemplo: $S \rightarrow Aa \mid a$

$A \rightarrow Sb$

3.1.2 TRANSFORMAÇÃO DE GRAMÁTICAS REGULARES EM AUTÔMATOS FINITOS

Uma linguagem regular é gerada por alguma gramática regular e para mostrar que uma linguagem é regular, basta que se construa seu autômato finito reconhecedor (MENEZES, 2005). Ou seja, se G = (V, T, P, S) é uma gramática regular, então existe um autômato finito M = (E, Σ, δ, q_0, F) tal que a linguagem aceita por M é a mesma linguagem gerada por G (L(M) = L(G)).

Sendo assim, é possível construir um autômato finito *M* que reconheça uma determinada linguagem regular denotada por uma gramática *G*:

ACEITA (M) = GERA (G)

Essa gramática pode então ser vista quase que como uma forma linear de representar o autômato (VIEIRA, 2006).

Ao se ter um autômato finito que reconhece a linguagem de uma gramática regular, então os componentes do autômato são definidos conforme segue (MENEZES, 2005):

a) o alfabeto de entrada são os símbolos terminais da gramática;
b) os estados correspondem aos símbolos não terminais da gramática e mais algum outro estado necessário que foi criado de acordo com as regras;
c) o estado inicial é o símbolo inicial da gramática;
d) os estados finais são todos os estados que produzem o vazio (ε) juntamente com novos estados possivelmente criados de acordo com as regras da gramática;
e) as transições dependem de cada regra da gramática.

Sendo assim, as regras que a gramática possuir na forma $X \rightarrow aY$ corresponderão a transições da forma $\delta (X, a) = Y$. Já as que aparecerem na forma $X \rightarrow a$ originarão transições do tipo $\delta (X, a) = Z$, onde Z é um novo estado do autômato finito e é um estado final. E por fim, as regras $X \rightarrow \varepsilon$ determinarão que X é um estado final, não gerando nenhuma transição (MENEZES, 2005). Essas regras podem ser melhor compreendidas por meio da Tabela 4.

Tabela 4. Regras da gramática regular e correspondências no autômato finito

Regra da gramática	**Autômato finito**
$X \rightarrow aY$	$\delta (X, a) = Y$
$X \rightarrow a$	$\delta (X, a) = Z$, onde Z é um estado novo e final
$X \rightarrow \varepsilon$	X é estado final

De acordo com as regras citadas anteriormente, a Tabela 5 mostra as transições do autômato reconhecedor da gramática regular G = ({A,B}, {0,1}, P, A), onde P:

a) $A \rightarrow 0A \mid 1B \mid 0$

b) $B \rightarrow 1B \mid \varepsilon$

Tabela 5. Regras da gramática regular e transições do autômato finito

Regras	**Transições**	**Observação**
$A \rightarrow 0A$	$\delta (A, 0) = A$	
$A \rightarrow 1B$	$\delta (A, 1) = B$	
$A \rightarrow 0$	$\delta (A, 0) = Z$	Z é estado final
$B \rightarrow 1B$	$\delta (B, 1) = B$	
$B \rightarrow \varepsilon$		B é estado final

Fonte: VIEIRA, N. (2006)

A Figura 12 mostra a representação na forma gráfica do autômato finito reconhecedor da linguagem gerada pela gramática *G*.

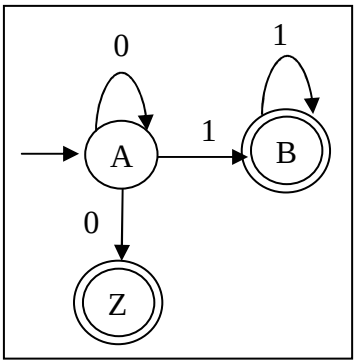


Figura 12. Autômato finito gerado a partir da gramática regular
Fonte: VIEIRA, N. (2006)

Os passos a seguir mostram um resumo das correspondências entre uma gramática regular G = (V, T, P, S) e o autômato finito M = (E, Σ, δ, q_0, F) reconhecedor da linguagem gerada por ela.

a) E = V ∪ {Z}, onde Z é um novo estado;

b) Σ = T;

c) q_0 = S;

d) F = (E | E = ε ∈ P} ∪ {Z (estado novo) }, ou seja, todos os estados que produzem ε, juntamente com novo(s) estado(s);

e) construir δ (função de transição) obedecendo as seguintes regras:

- para cada produção da forma X → a ∈ P, criar a transição t (X, a) = Z (estado novo);
- para cada produção da forma X → aY ∈ P, criar a transição t (X, a) = Y;
- para cada produção da forma X → ε, não é criada nenhuma transição e X passa a ser um estado final.

4 EXPRESSÕES REGULARES

Expressões regulares são expressões propriamente ditas que definem uma linguagem, permitindo representá-la por meio de uma descrição algébrica. Elas podem definir exatamente a mesma linguagem que os autômatos aceitam e que as gramáticas regulares geram: as linguagens regulares (CRESPO, 2001; HOPCROFT; ULLMAN; MOTWANI, 2002).

Uma expressão regular é formada por expressões regulares mais simples usando-se um conjunto de regras de definição (AHO; SETHI; ULLMAN, 1995).

Segundo Louden (2004) expressões regulares são montadas com os caracteres de um alfabeto. Porém, há dois símbolos adicionais que devem ser considerados para situações especiais: o epsilon (ε) para denotar a cadeia vazia, que é uma cadeia composta por zero (0) caracteres, e o ∅ para representar o conjunto vazio, que não contém cadeia de caracteres.

4.1 OPERADORES DE EXPRESSÕES REGULARES

As álgebras de todos os tipos permitem construir expressões com constantes ou variáveis, alguns operadores e símbolos como parênteses para agrupamento e definição de prioridades. Assim como nessas álgebras, as expressões regulares seguem esse padrão para denotar linguagens, sendo que as três operações básicas por elas utilizadas são: união, concatenação e repetição (também chamada fechamento) (HOPCROFT; ULLMAN; MOTWANI, 2002; LOUDEN, 2004).

A operação de união é representada pelo símbolo “ + ”. Dessa forma, se E e F são expressões regulares, E + F é uma expressão regular que denota a linguagem

definida por E (L(E)) unida com a linguagem definida por F (L(F)), ou seja, L (E+F) = L (E) ∪ L(F) (HOPCROFT; ULLMAN; MOTWANI, 2002). A expressão regular $a+b$ possui a operação de união e é representada na Figura 13 por um autômato finito, onde pode-se observar que os autômatos M1 e M2 processam de forma não determinística, assim sendo, basta um dos dois autômatos aceitar a entrada, para que o autômato M a aceite.

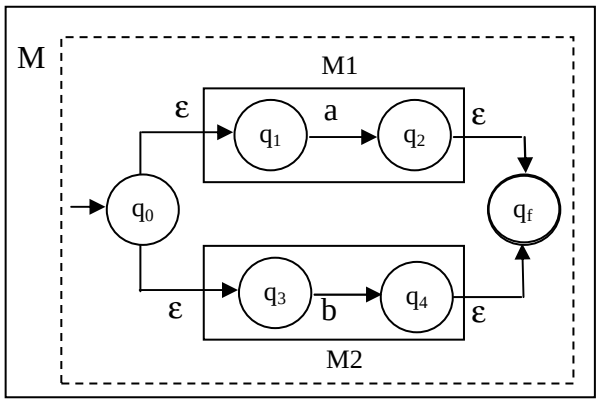


Figura 13. Representação da operação de união
Fonte: Adaptado de HOPCROFT, J.; ULLMAN, J.; MOTWANI, R. (2002)

Por sua vez, a concatenação de E e F é denotada como EF, então L (EF) = L(E)L(F) (HOPCROFT; ULLMAN; MOTWANI, 2002). Isso significa que se E for a expressão *a* e F for a expressão *c*, então a concatenação de EF será *ac*. A operação de concatenação é representada na Figura 14 e demonstra que o autômato M somente aceitará a entrada se os autômatos M1 e M2 a aceitarem, obrigatoriamente.

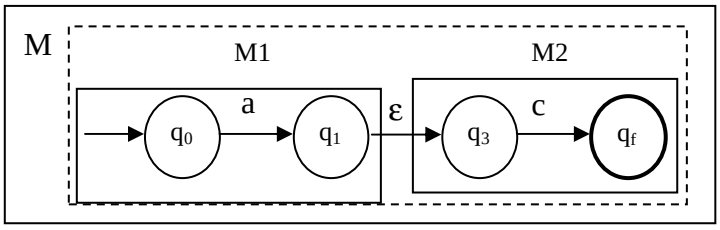


Figura 14. Representação da operação de concatenação
Fonte: Adaptado de HOPCROFT, J.; ULLMAN, J.; MOTWANI, R. (2002)

Enfim, a repetição é representada pelo símbolo “ * ”. Neste caso, a expressão regular a^* pode resultar em *a*, *aa*, *aaa*, e assim por diante, bem como resultar numa cadeia vazia (HOPCROFT; ULLMAN; MOTWANI, 2002).

A Figura 15 demonstra o autômato reconhecedor da linguagem denotada pela er = a^*, onde pode-se obter 0 (zero) ou mais *as*, sendo que para resultar na cadeia vazia basta que não seja processado nenhum símbolo de entrada, pois o único estado do autômato já é um estado de aceitação (estado final).

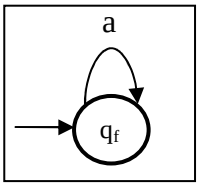


Figura 15. Representação da operação de repetição

Assim como na álgebra tradicional, os operadores das expressões regulares também possuem prioridades diferentes, sendo o fechamento o que possui a maior prioridade, seguido da concatenação e da união (CRESPO, 2001).

Além das operações básicas, outros símbolos foram acrescentados a fim de facilitar a definição das expressões regulares (CRESPO, 2001; LOUDEN, 2004):

a) fecho transitivo ($^+$): funciona de forma semelhante ao fechamento (repetição), porém não permite a cadeia vazia;

b) parênteses: possibilitam mudar a precedência dos operadores.

A Figura 16 representa o fecho transitivo para a expressão regular a^+.

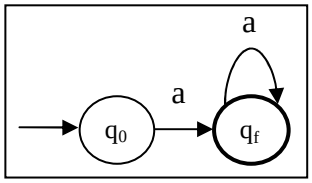


Figura 16. Representação do fecho transitivo

A Tabela 6 mostra alguns exemplos de expressões regulares e a linguagem gerada por cada uma delas.

Tabela 6. Expressões regulares e correspondentes linguagens geradas

Expressão Regular	***Linguagem Gerada***
aa	Somente a palavra *aa*
ba^*	Todas as palavras que iniciam por *b*, seguindo por zero ou mais *a*
$(a+b)^*$	Todas as palavras sobre {*a*,*b*}
$(a+b)^*aa(a+b)^*$	Todas as palavras contendo *aa* 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*
$(a+\varepsilon)(b+ba)^*$	Todas as palavras que não possuem dois *a* consecutivos

Fonte: MENEZES, P. (2005)

Embora as expressões regulares sejam uma outra forma de representar as linguagens regulares, além dos autômatos finitos e gramáticas regulares, elas não são capazes de gerar todas as cadeias de caracteres que se possa imaginar. Um exemplo disso é uma cadeia de caracteres que possui um único *b*, com mesmo número de *a* antes e depois: *b* ou *aba* ou *aabaa* ou *aaabaaa* e assim por diante. Isso porque as expressões regulares não conseguem contar e assim não se tem como garantir que o número de *as* antes seja o mesmo número de *as* depois. O mais próximo que se consegue chegar de uma cadeia com uma regra como essas, é com a expressão a^*ba^*, porém não se terá a garantia de que o número de *as* será igual (LOUDEN, 2004).

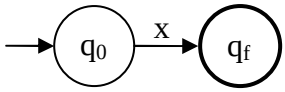
4.2 TRANSFORMAÇÃO DE EXPRESSÕES REGULARES EM AUTÔMATOS FINITOS

Uma linguagem é dita regular somente se for possível construir um autômato finito capaz de reconhecê-la e as expressões regulares definem exatamente a classe de linguagens regulares. Assim sendo, por meio de qualquer expressão regular se pode construir um autômato finito capaz de reconhecer a linguagem denotada por ela (MENEZES, 2005):

ACEITA (M) = GERA (r).

Caso uma expressão regular *r* não possua operadores, fica simples definir seu autômato finito correspondente, como pode ser observado na Tabela 7.

Tabela 7. Autômatos finitos correspondentes às expressões regulares com zero operadores

ER	***Autômato Finito Correspondente***
r = ∅	
r = ε	
r = x	

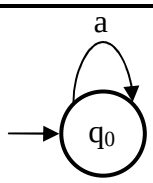
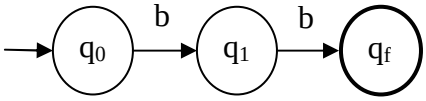
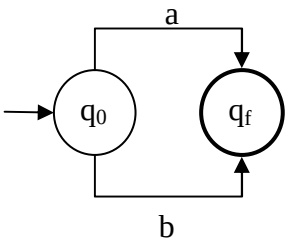
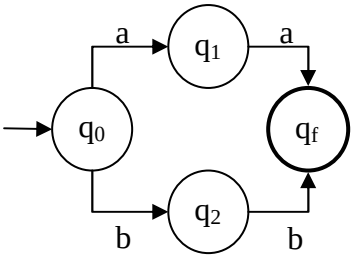
Fonte: MENEZES, P. (2005)

Na Tabela 7, pode-se observar que a linguagem do primeiro autômato é ∅, pois não existe nenhum caminho que inicia em um estado inicial e termina em um estado final. Já no segundo autômato, a linguagem por ele definida é {ε}, pois ele não lê nenhum símbolo de entrada e mesmo assim termina em um estado de aceitação.

Finalmente, no terceiro autômato, fica simples observar que sua linguagem consiste em um único *string x*.

A obtenção de expressões regulares com um ou mais operadores, é realizada por meio da combinação das operações das expressões regulares nos autômatos finitos, como demonstrado na Tabela 8.

Tabela 8. Autômatos finitos correspondentes às expressões regulares com um ou mais operadores

ER	***Autômato Finito Correspondente***	***Operação***
$r = a^*$		**Repetição**
r = bb		**Concatenação**
r = a + b		**União**
r = aa + bb		**Concatenação e União**

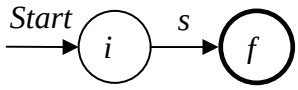
Fonte: MENEZES, P. (2005)

A seguir são demonstrados os passos de um algoritmo para geração de um AFND a partir de uma expressão regular (CRESPO, 2001).

Entrada: Uma expressão regular.

Resultado: Um AFND.

1. Identificar os símbolos de entrada do alfabeto usados na expressão regular. Para cada símbolo *s*, gerar o seguinte AFND:

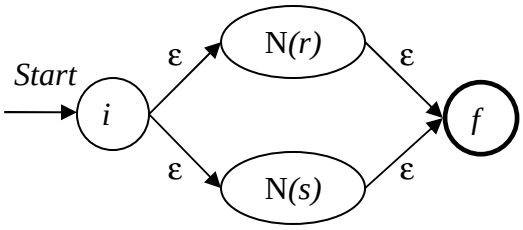


$(\{ i , f \}, s , \{ < i , s > \rightarrow f \} , i , \{ f \})$;

2. O AFND da expressão regular (*r*) é idêntico ao AFND da expressão *r*;

3. Dados dois AFND para expressões regulares *r* e *s*, respectivamente, $N(r) = (E(r), \Sigma(r), \delta(r), q_0(r), F(r))$ e $N(s) = (E(s), \Sigma(s), \delta(s), q_0(s), F(s))$

   a) para a expressão regular *r* + *s* é construído o seguinte AFND:



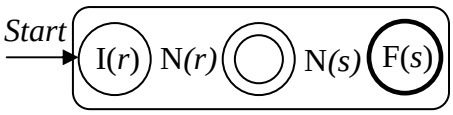
$(E(r) \cup E(s) \cup \{i, f\}, \Sigma(r) \cup \Sigma(s),$

$\delta(r) \cup \delta(s) \cup \{<i, \varepsilon> \rightarrow q_0(r), <i, \varepsilon> \rightarrow q_0(s),$

$<f(r), \varepsilon> \rightarrow f, <f(s), \varepsilon> \rightarrow f\}, \{f\})$

no qual $f(r) \in F(r)$ e $f(s) \in F(s)$;

   b) para a expressão regular *rs* é construído o seguinte AFND:

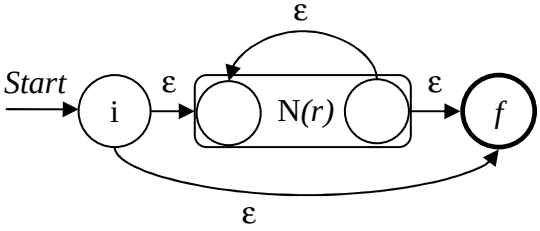


$(E(r) \cup E(s), \Sigma(r) \cup \Sigma(s), \delta(r) \cup \delta(s),$

$I(r), F(s))$ no qual os estados $F(r)$

coincidem com os estados $q_0(s)$;

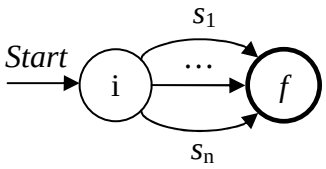
   c) para a expressão regular r^* é construído o seguinte AFND:



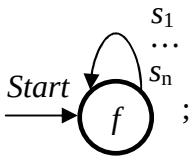
$(\{i,f\} \cup E(r), \Sigma(r), \delta(r) \cup \{<i,\varepsilon> \rightarrow q_0(r), <i,\varepsilon> \rightarrow f, <f(r),\varepsilon> \rightarrow q_0(r),$

$<f(r),\varepsilon> \rightarrow f\}, i, \{f\})$ no qual $f(r) \in F(r)$;

d) para a expressão regular s_1+...+ s_n, em que s_1, ..., s_n são símbolos do alfabeto, o AFND gerado pode ser simplificado para:



e) para a expressão (s_1+...+ s_n)*, em que s_1, ..., s_n são símbolos do alfabeto, o AFND gerado pode ser simplificado para:



A transformação de uma expressão regular em um autômato finito é demonstrada a seguir. Com o propósito de facilitar o entendimento, cada parte da expressão regular será representada por um autômato independente e todas as partes serão unidas num só autômato no final.

er = (0 + 1)* 1 (0 + 1)

1º passo: Construir o autômato que reconhece (0 + 1)*;

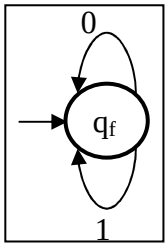


Figura 17. Autômato finito que reconhece (0 + 1)*

2º passo: Construir o autômato que reconhece (1);

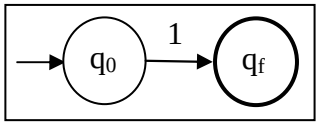


Figura 18. Autômato finito que reconhece (1)

3º passo: Construir o autômato que reconhece (0 + 1);

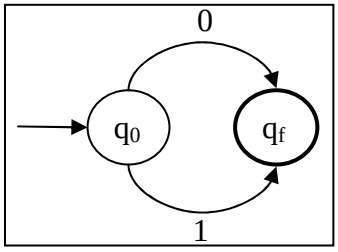


Figura 19. Autômato finito que reconhece (0 + 1)

4º passo: Unir todas as partes acima desenvolvidas num único autômato finito.

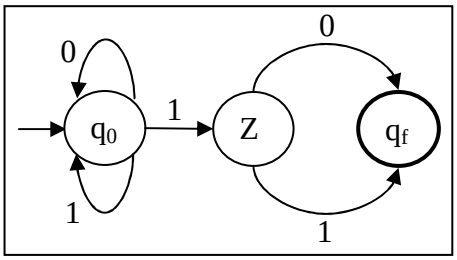


Figura 20. Autômato finito a partir de uma expressão regular

Assim como o exemplo citado acima, outros autômatos finitos podem ser gerados a partir de expressões regulares, basta combinar as operações das expressões no autômato finito.

5 TRABALHOS CORRELATOS

No decorrer das pesquisas referentes a este trabalho foram encontrados alguns trabalhos semelhantes na área de autômatos finitos, expressões regulares e gramáticas regulares que a seguir estão relacionados.

5.1 EDULING - SOFTWARE EDUCACIONAL PARA LINGUAGENS REGULARES

Trabalho de Conclusão do Curso Superior de Ciência da Computação da Universidade do Vale do Itajaí (UNIVALI) apresentado no ano de 2002.

A ferramenta denominada EduLing é um software educacional que auxilia na aprendizagem de linguagens regulares. Ela possui uma interface gráfica que permite ao usuário construir autômatos finitos e expressões regulares, podendo verificar a equivalência destas duas formas de representação de uma linguagem regular. Além disso, a ferramenta permite simular o reconhecimento de sentenças, validando desta forma a linguagem construída. O objetivo do software EduLing é reduzir a dificuldade de alunos no aprendizado de Linguagens Formais (DOGNINI, 2003).

5.2 LANGUAGE EMULATOR

Trabalho realizado no curso de pós-graduação na Universidade Federal de Minas Gerais (UFMG) no ano de 2002.

O Language Emulator é uma ferramenta desenvolvida na linguagem Java (ambiente multiplataforma) que possibilita a manipulação de AFDs, AFNDs, AFNDs com transições lambda, todos na forma tabular, expressões regulares, gramáticas

regulares, máquinas de Moore e de Mealy, tendo várias funcionalidades (VIEIRA; VIEIRA; VIEIRA 2003):

a) minimizar autômatos finitos determinísticos;
b) transformar autômato finito não determinístico em autômato finito determinístico;
c) transformar autômato finito não determinístico com transições lambda em autômato finito não determinístico;
d) transformar uma gramática regular em um autômato finito não determinístico;
e) transformar autômato finito não determinístico em uma gramática regular;
f) transformar uma expressão regular em um autômato finito não determinístico com transição lambda;
g) transformar autômato finito não determinístico em uma expressão regular;
h) derivar todas as palavras até um tamanho *n* que pertençam a uma gramática regular;
i) determinar se uma palavra pertence a uma gramática regular;
j) determinar se uma palavra pertence à linguagem especificada por um autômato finito determinístico;
k) determinar se uma palavra pertence à linguagem especificada por uma expressão regular;
l) transformar uma máquina Mealy em uma máquina de Moore;
m) transformar uma máquina de Moore em uma máquina Mealy;
n) dar a saída de uma máquina de Moore para uma entrada dada;

o) determinar se uma palavra pertence à uma máquina Mealy ou de Moore;

p) união, interseção e diferença de dois AFDs;

q) complemento de um AFD;

r) união das diferenças entre dois AFDs. $A_F = (A_1 - A_2) \cup (A_2 - A_1)$;

s) salvar e carregar AFDs;

t) exportar o AFD para um arquivo no formato utilizado pela ferramenta GraphViz[8]. Esta ferramenta recebe uma descrição textual de um grafo e gera uma representação visual do mesmo.

O objetivo deste software é auxiliar os estudantes da Teoria da Computação a compreender os conceitos desta área, permitindo um melhor entendimento por meio de uma forma prática (VIEIRA; VIEIRA; VIEIRA, 2003).

A Figura 21 mostra a interface da ferramenta Languagem Emulator.

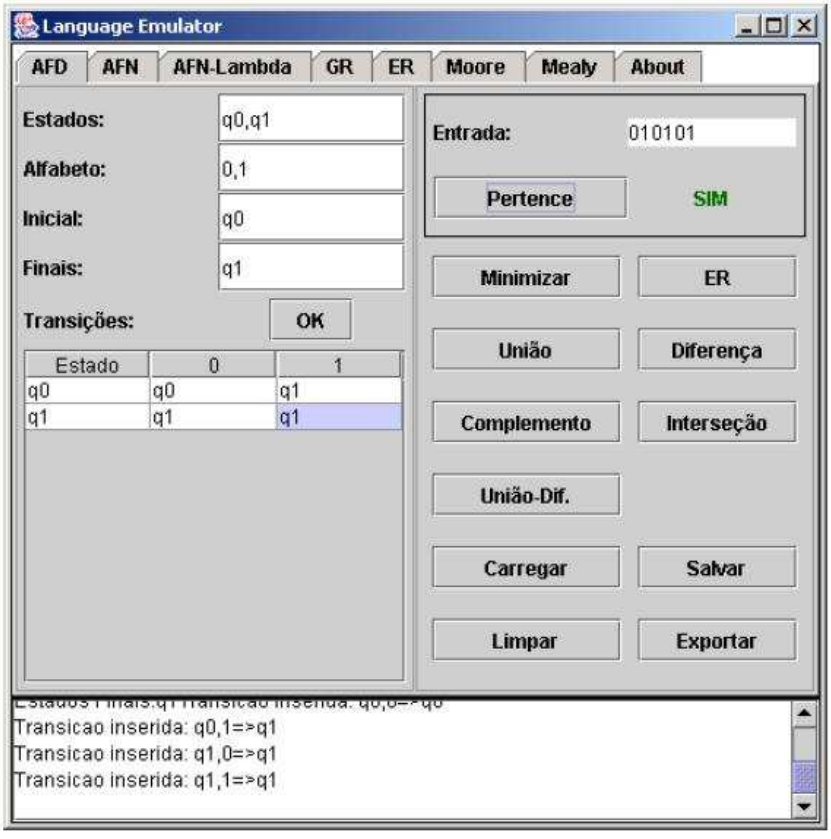


Figura 21. Language Emulator
Fonte: VIEIRA, L.; VIEIRA, M.; VIEIRA N. (2003)

5.3 JAVA FORMAL LANGUAGE AND AUTOMATA PACKAGE

O Java Formal Language and Automata Package (JFLAP) iniciou como uma série de ferramentas no Rensselaer Polytechnic Institute, USA, por volta de 1990. Mais tarde, em 1994, o projeto se mudou para Duke University, onde alunos e professores ainda trabalham no desenvolvimento de novas funções da ferramenta.

Trata-se de um software com interface gráfica que permite criar e simular diversos tipos de autômatos e fazer conversões entre diferentes representações de linguagens, tais como:

a) autômato finito para gramática regular;

b) gramática regular para autômato de pilha;

c) expressão regular para autômato finito;

d) autômato finito para expressão regular; entre outras.

O JFLAP é um pacote de ferramentas simples de usar e que pode ser utilizado como um auxílio na aprendizagem de conceitos básicos de Linguagens Formais e Teoria dos Autômatos (RODGER, 1990).

Ao executá-lo, surgirá uma tela inicial onde o usuário tem a opção de escolher o que deseja construir: autômato finito, autômato de pilha, gramática regular, expressão regular, etc., como pode ser observado na Figura 22.

Figura 22. Tela inicial do JFLAP
Fonte: RODGER, S. (1990)

A próxima tela surgirá de acordo com a opção selecionada pelo usuário. A Figura 23, por exemplo, mostra a tela de autômatos finitos, onde o usuário pode desenhá-los e fazer conversões que julgue necessário, como transformar um AFND em um AFD, minimizar um AFD, converter um AFND ou AFD em gramática regular ou em expressão regular, testar sentenças, entre outras ações.

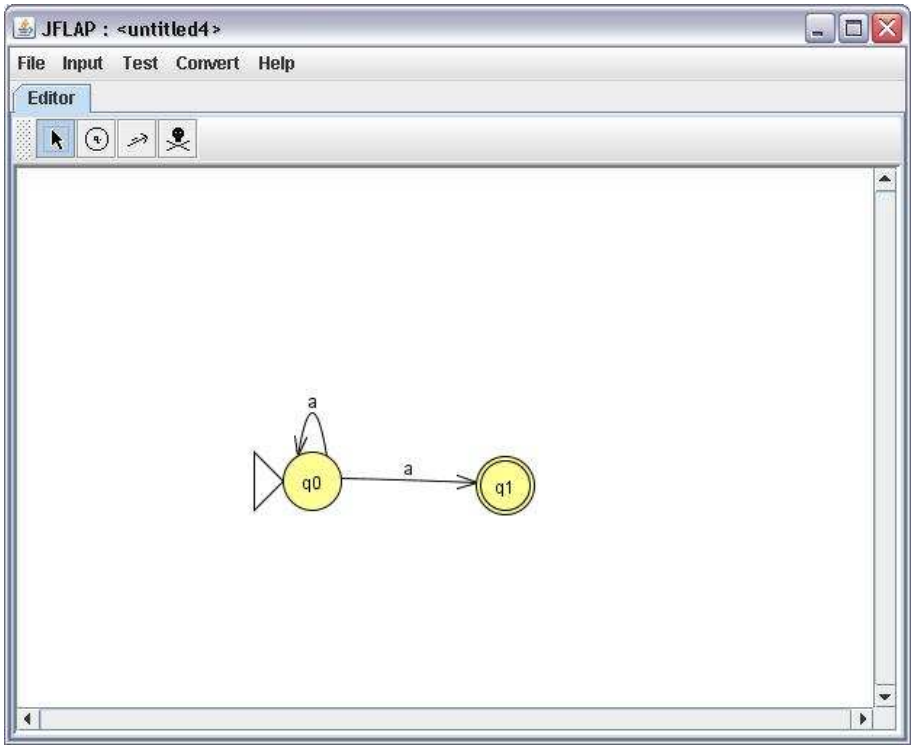


Figura 23. Tela de autômatos finitos do JFLAP
Fonte: RODGER, S. (1990)

Caso o usuário escolha a tela de expressões regulares, então uma expressão regular será solicitada, conforme pode ser observado na Figura 24. A única ação que se pode solicitar nesse caso é a transformação da mesma em um autômato finito.

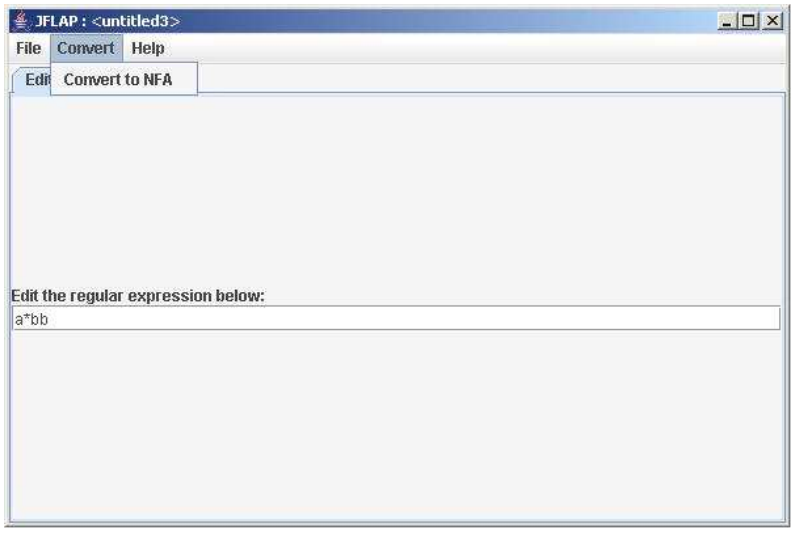


Figura 24. Tela de expressões regulares do JFLAP
Fonte: RODGER, S. (1990)

A Figura 25 mostra a interface da opção de gramáticas. O usuário informa as regras da gramática e logo depois pode escolher uma ação a ser tomada. O JFLAP permite transformar a gramática informada em um autômato de pilha, permite construir um autômato finito a partir de uma gramática linear à direita, entre outras coisas.

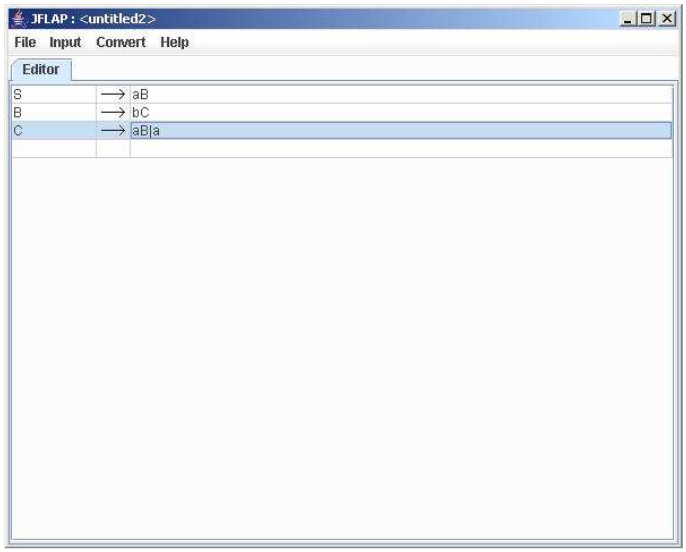


Figura 25. Tela de gramáticas do JFLAP
Fonte: RODGER, S. (1990)

5.4 REANIMATOR

A ferramenta reAnimator proporciona uma visualização interativa de autômatos finitos determinísticos e não determinísticos equivalentes, obtidos a partir de uma expressão regular que pode ser informada pelo usuário.

O *front end* da ferramenta foi criado com a plataforma openLazlo, compilada com Flash e usa AJAX and JSON para construir os autômatos. O *back end é* escrito em C e Python (STEELE, 2006).

A Figura 26 mostra a ferramenta reAnimator, que é executada diretamente de uma página da web, no endereço http://osteele.com/tools/reanimator .

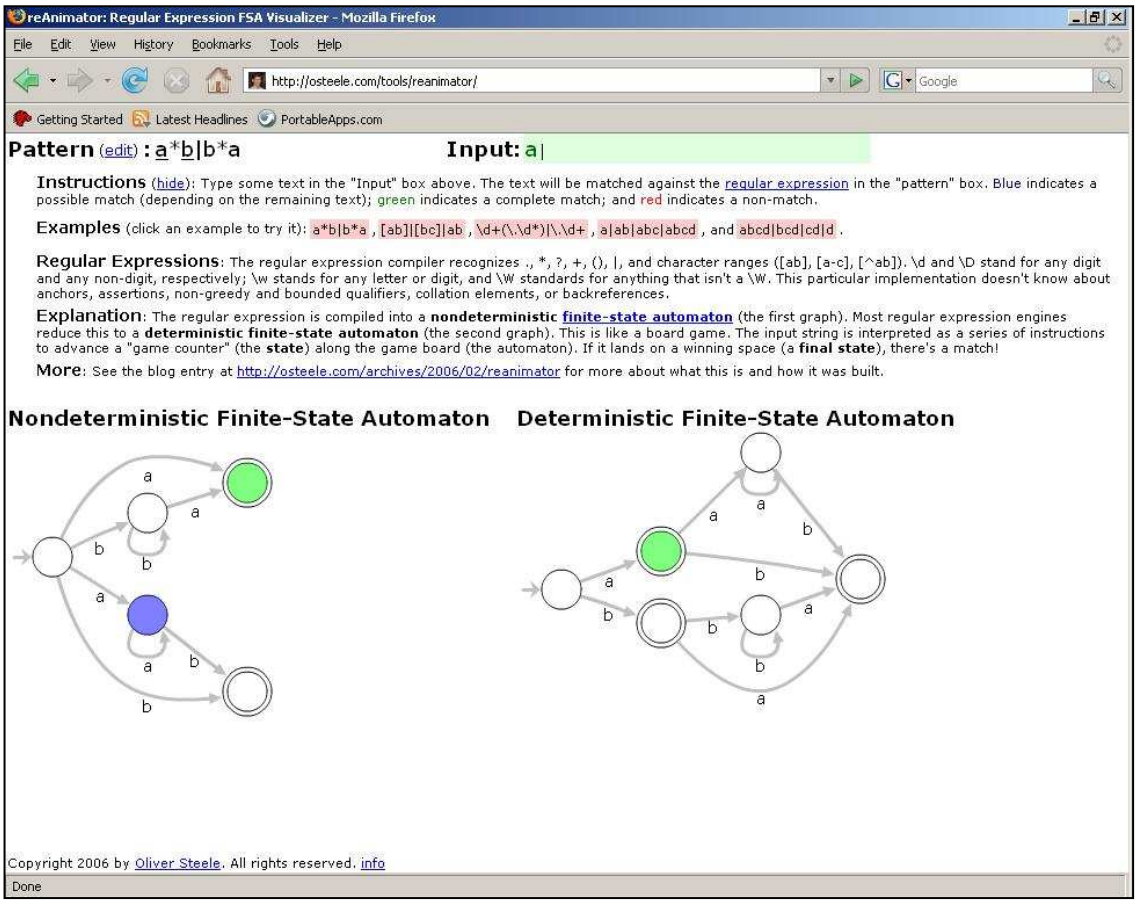


Figura 26. reAnimator
Fonte: STEELE, O. (2006)

5.5 GRAIL

O Grail é um ambiente escrito em C++ para manipulação de autômatos finitos, expressões regulares e outros objetos da teoria de linguagem formal. Com essa ferramenta é possível transformar autômatos finitos em expressões regulares e vice-versa, minimizar autômatos, torná-los determinísticos, além de outras operações (DARREL; WOOD, 1995).

6 A FERRAMENTA AFLAB

O AFLAB é um ambiente de criação e manipulação de autômatos finitos que surgiu da necessidade de se ter uma ferramenta para esse fim nas disciplinas de Linguagens Formais, Compiladores e Teoria da Computação do curso de Ciência da Computação da Unesc.

A primeira versão do software contemplou os módulos de reconhecimento de sentenças por meio de autômatos finitos determinísticos ou não determinísticos definidos na forma gráfica ou tabular, proposto por Gaidzinski (2007).

O sistema foi desenvolvido em *object pascal*, por meio da plataforma Borland® Delphi™ Enterprise, versão 7.0, pelo fato de ser um ambiente já utilizado no meio acadêmico, permitindo que o código fonte seja estudado por outros alunos e modificado conforme suas necessidades. Além disso, gera programas bastante leves, não exigindo assim o consumo de muitos recursos do computador (GAIDZINSKI, 2007).

Internamente, a ferramenta possui uma estrutura única para o armazenamento de autômatos finitos, que podem ser determinísticos ou não determinísticos, na sua forma gráfica ou tabular, permitindo dessa forma, que o autômato finito construído na forma tabular seja visualizado também na forma gráfica e vice versa. Essa estrutura foi dividida basicamente em três classes (GAIDZINSKI, 2007):

a) *Testados*: responsável por armazenar atributos dos estados, tais como: nome, marcação de estado inicial ou final e as coordenadas (x,y) para aparecer na tela;

b) *Talfabeto*: responsável por armazenar os símbolos de entrada do autômato finito;

c) *Tconexao*: responsável por armazenar as transições do autômato, com os seguintes atributos: estado de origem (saída da transição, pertencente a *Testados*), símbolo de entrada (um único caractere pertencente a *Talfabeto*) e estado destino (chegada da transição, pertencente a *Testados*).

O armazenamento de estados, alfabetos e transições é feito por meio de *arrays* dinâmicos das classes declaradas, pois dessa forma não fica estipulado um número máximo de cada um desses elementos na aplicação, ficando a critério do usuário.

Na primeira etapa da ferramenta, é possível informar o autômato (gráfica ou tabularmente) ou carregá-lo de um arquivo salvo anteriormente com extensão do tipo INI.

No módulo de Forma Tabular é necessário informar quais são os estados do autômato finito, seus símbolos de entrada, o estado inicial e o conjunto de estados finais. Depois do preenchimento dessas informações, prossegue-se informando as transições entre os estados e os respectivos símbolos de entrada. Finalmente a sentença a ser testada pode ser informada no campo correspondente, que trará como resposta a palavra “VERDADEIRA” se tal sentença for reconhecida pelo autômato informado ou “FALSA” caso contrário. Um botão auxiliar permite a migração do autômato finito construído na Forma Tabular para ser visualizado e / ou manipulado na sua forma gráfica. Outro botão permite que o autômato finito informado seja excluído, afim de que um novo possa começar a ser manipulado. O módulo de Forma Tabular pode ser observado na Figura 27.

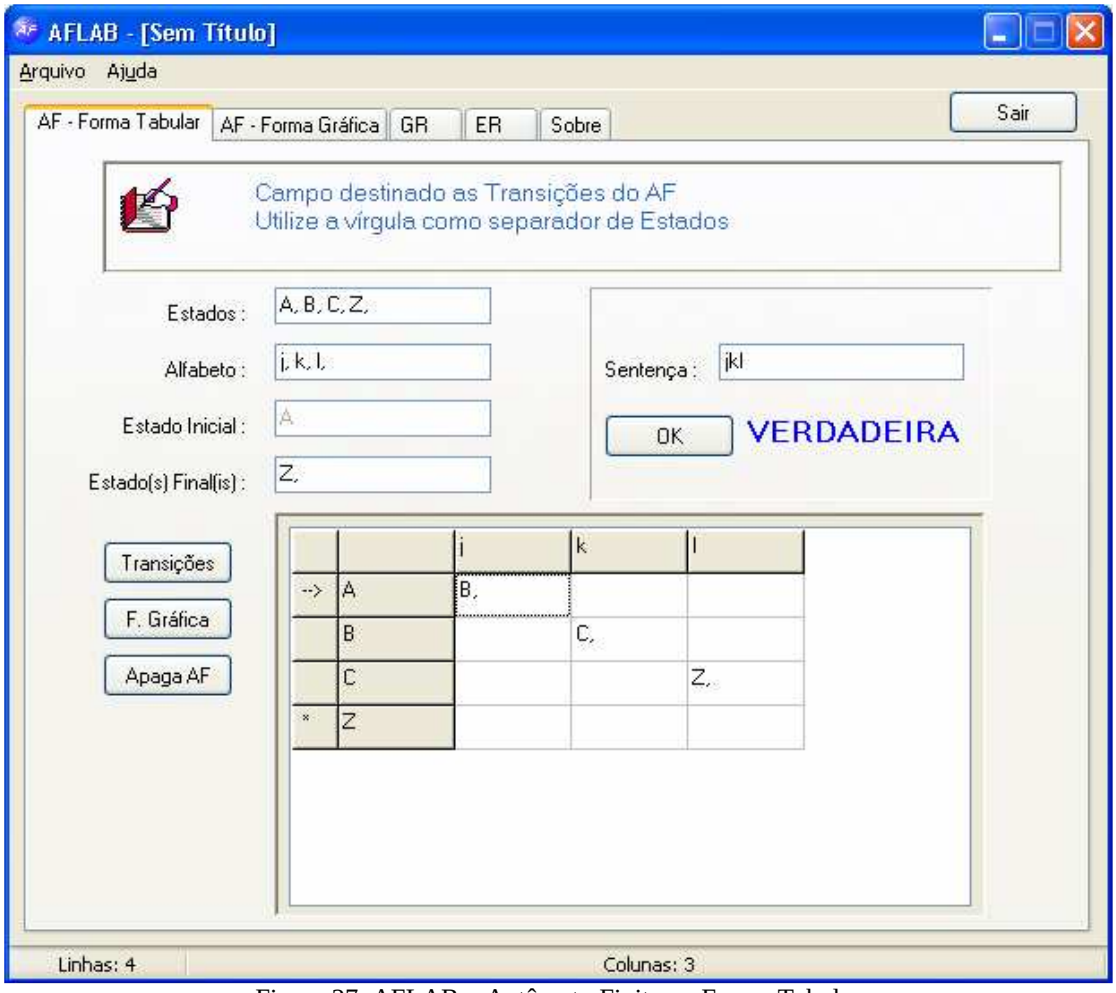


Figura 27. AFLAB – Autômato Finito na Forma Tabular
Fonte: GAIDZINSKI, M. (2007)

O módulo de Forma Gráfica do AFLAB permite que o autômato finito seja desenhado na tela, utilizando-se as opções de adicionar estados, adicionar estados finais, remover e mover estados, sendo que o primeiro estado adicionado é considerado por padrão o estado inicial do autômato. Os símbolos de entrada são adicionados juntamente com as transições, que podem ser informadas após o clique no botão de “Transições”. Para excluir uma transição, deve-se clicar no botão “Apagar Trans.” e selecionar as transições desejadas. Finalmente, com o autômato informado, pode-se informar a sentença a ser testada no campo correspondente e, assim como no módulo de Forma Tabular, o software dará então como resposta a palavra “VERDADEIRA” se tal sentença for reconhecida pelo autômato ou “FALSA” caso contrário. Um botão auxiliar

“F.Tabular” permite a migração do autômato finito construído neste módulo para ser visualizado e / ou manipulado na sua forma tabular. Outro botão auxiliar, o “Apagar AF”, permite que o autômato finito informado seja excluído, afim de que um novo possa ser construído. O módulo de Forma Gráfica pode ser observado na Figura 28.

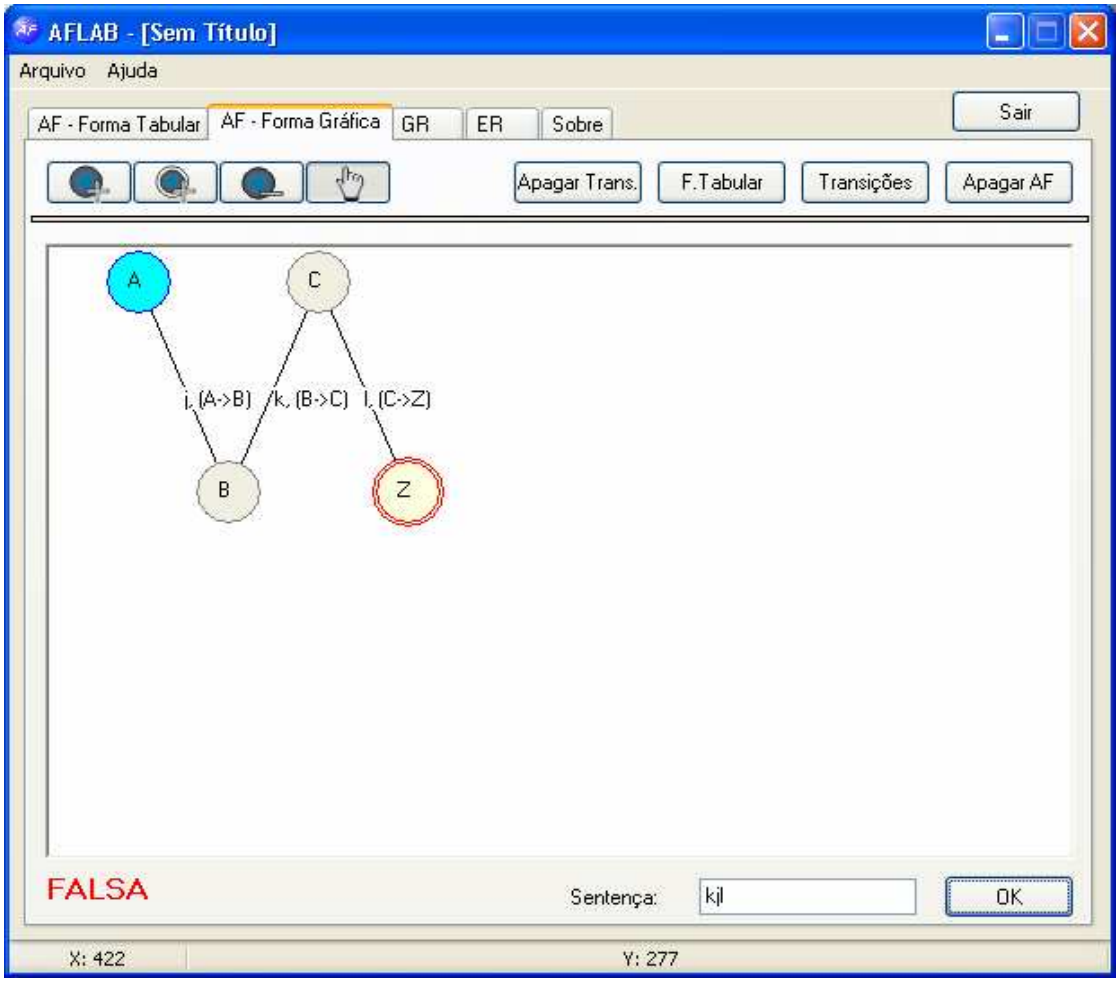


Figura 28. AFLAB – Autômato Finito na Forma Gráfica
Fonte: GAIDZINSKI, M. (2007)

Após o início do desenvolvimento da ferramenta AFLAB, percebeu-se que ela poderia ser expandida, pois existem outros assuntos relacionados a autômatos finitos do que simplesmente o reconhecimento de sentenças. Por este motivo foi dado início ao desenvolvimento dos módulos de Gramáticas Regulares e Expressões Regulares, que possibilitam a transformação de tais formalismos em um autômato finito.

6.1 GRAMÁTICAS REGULARES E EXPRESSÕES REGULARES: UM AMBIENTE DE MANIPULAÇÃO NO AFLAB

O módulo de geração de autômatos finitos a partir de gramáticas regulares tem por objetivo receber uma gramática regular informada pelo usuário e aplicar um algoritmo para gerar o autômato finito reconhecedor da linguagem gerada por esta gramática.

A construção de autômatos finitos a partir de expressões regulares, por sua vez, tem por objetivo receber uma expressão regular informada pelo usuário e a partir dela gerar o autômato finito reconhecedor da linguagem denotada por tal expressão.

Os dois módulos utilizam as classes já existentes de autômatos finitos da versão 1.0.0 do AFLAB no que diz respeito aos estados, alfabeto e conexões. Desta forma, é possível haver uma integração entre os novos módulos e os módulos pré-existentes do AFLAB (AF - Forma Tabular e AF – Forma Gráfica), ou seja, a gramática regular ou a expressão regular informada no seu respectivo módulo será tratada e permitirá a criação do autômato finito que poderá ser visualizado tanto em sua forma tabular quanto na sua forma gráfica.

Ao término deste e de outros trabalhos relacionados ao AFLAB, pretende-se ter uma ferramenta bastante completa no que diz respeito a manipulação de autômatos finitos, podendo ser utilizada em sala de aula para o ensino de matérias relacionadas a este assunto.

6.2 METODOLOGIA

Antes de iniciar a parte prática do projeto, foi necessário realizar um levantamento bibliográfico referente a AFD, AFND, gramáticas regulares e expressões regulares, além de uma pesquisa mais aprofundada na busca de algoritmos para obtenção de autômatos finitos a partir de gramáticas regulares e geração de autômatos finitos a partir de expressões regulares.

A modelagem dos dois módulos foi feita por meio de UML com a ferramenta Pacestar UML Diagrammer[2], o que permitiu demonstrar as ações que cada entidade pode exercer no sistema por meio do diagrama de Caso de Uso (*Use Case*) e o fluxo das atividades, por meio do diagrama de Atividades. As Figuras 29 e 30 mostram os diagramas de Caso de Uso referente ao módulo de Gramáticas Regulares e as Figuras 31 e 32 são referentes ao módulo de Expressões Regulares.

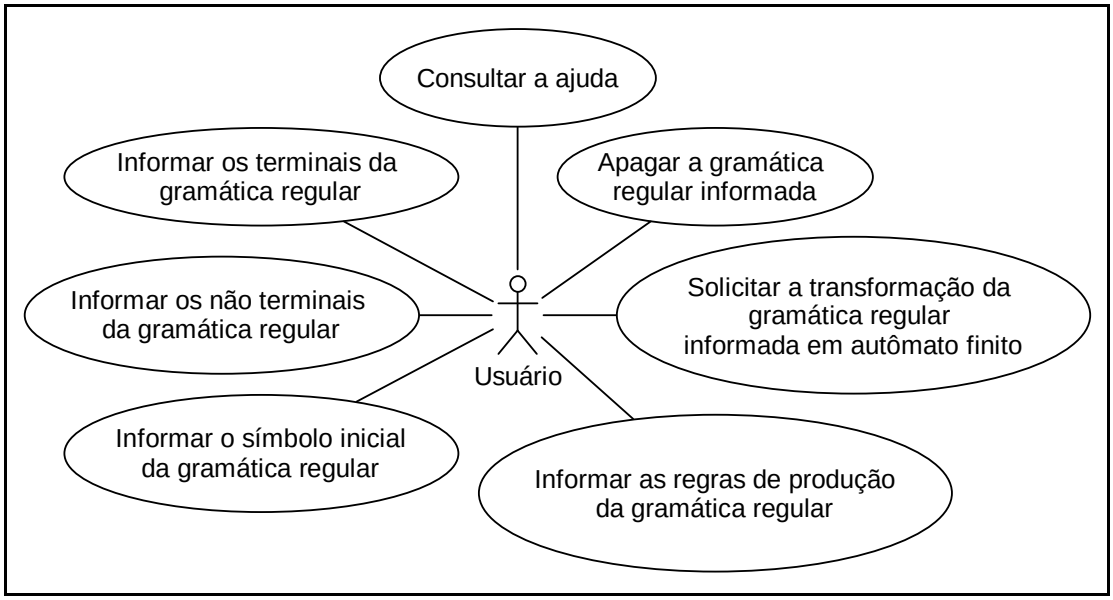


Figura 29. Diagrama de Caso de Uso do ponto de vista do usuário – Módulo de Gramáticas Regulares

[2] Ferramenta gráfica com interface simples que permite criar diagramas UML de diferentes tipos.

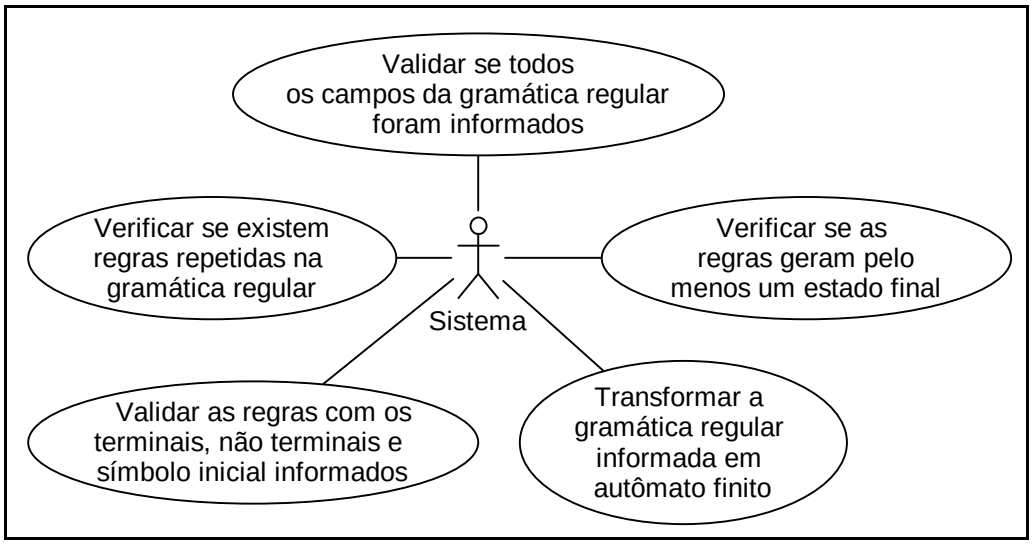


Figura 30. Diagrama de Caso de Uso do ponto de vista do sistema – Módulo de Gramáticas Regulares

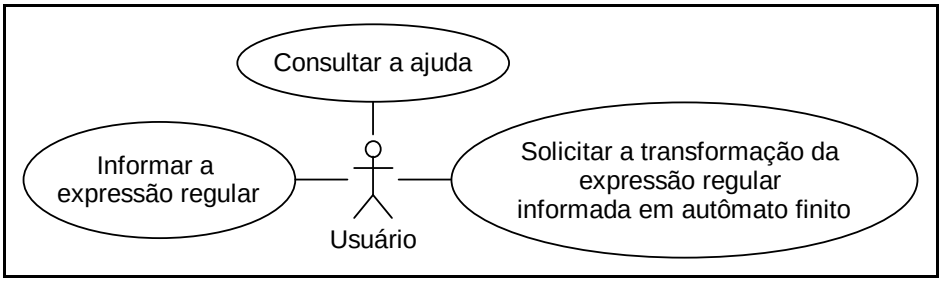


Figura 31. Diagrama de Caso de Uso do ponto de vista do usuário – Módulo de Expressões Regulares

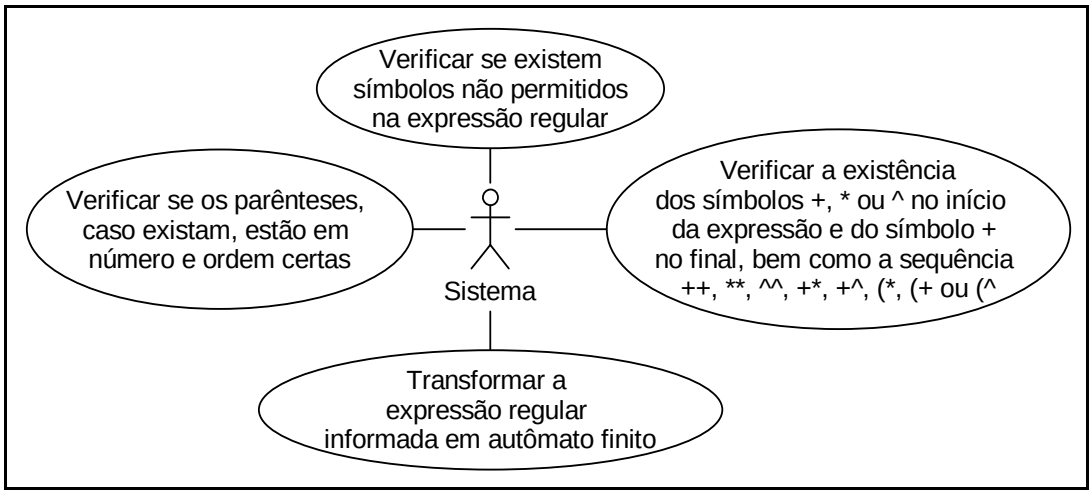


Figura 32. Diagrama de Caso de Uso do ponto de vista do sistema – Módulo de Expressões Regulares

Os diagramas de Atividades, por sua vez, que demonstram o fluxo de cada módulo, podem ser visualizados nas Figuras 33 e 34.

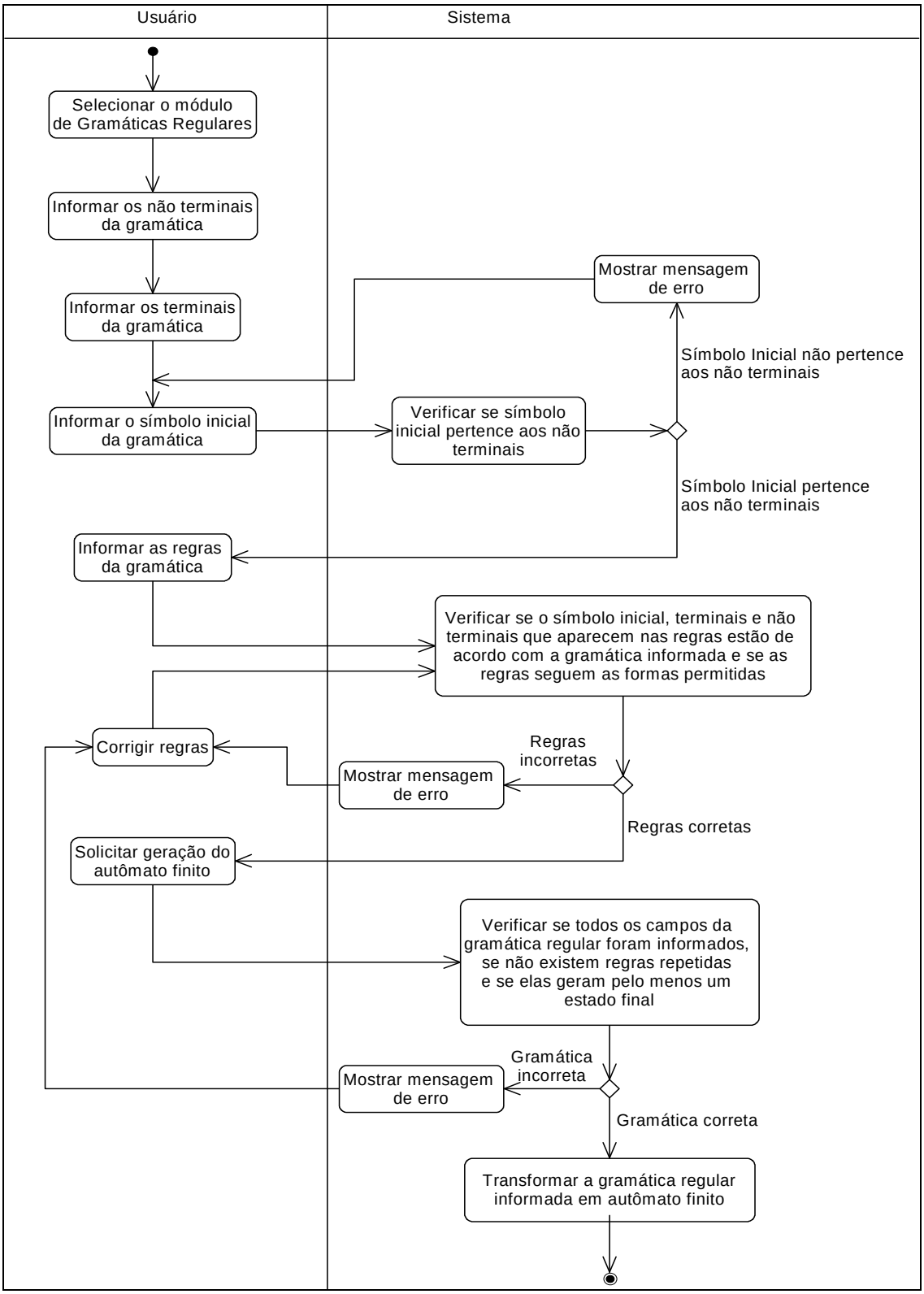


Figura 33. Diagrama de Atividades – Módulo de Gramáticas Regulares

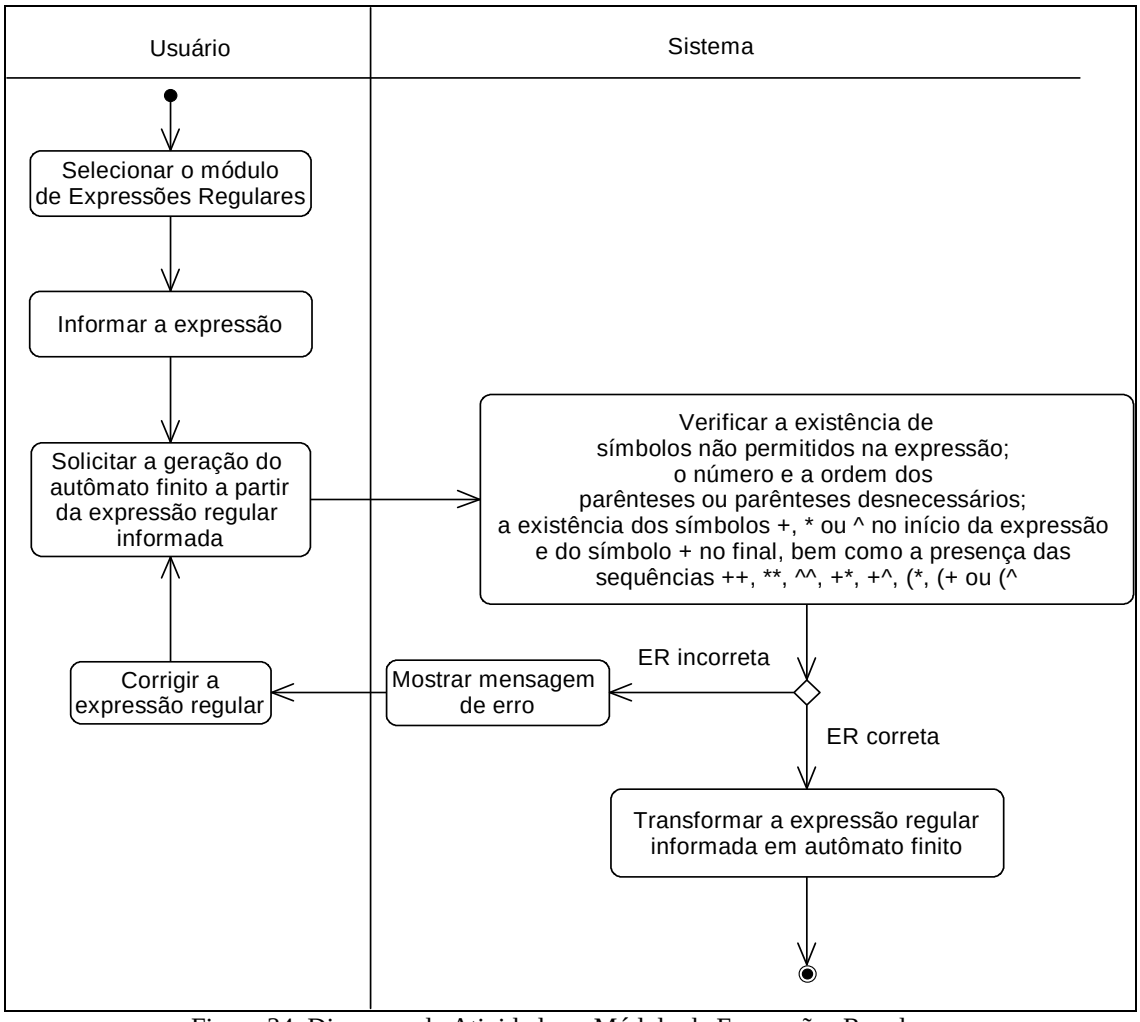


Figura 34. Diagrama de Atividades – Módulo de Expressões Regulares

Após o levantamento bibliográfico e a modelagem em UML, partiu-se para a implementação dos módulos em *object pascal*, por meio da plataforma Borland® Delphi™ Enterprise, versão 7.0, pelo fato de ser uma continuação da Shell AFLAB. Dessa forma, foi necessário também possuir o código fonte da ferramenta, versão 1.0.0. Quanto ao sistema operacional, foi utilizado o Microsoft® Windows XP.

6.3 DESENVOLVIMENTO

6.3.1 Módulo de Gramáticas Regulares

O raciocínio para a implementação deste módulo partiu do que foi estudado sobre a transformação de gramática regular em autômato finito. Foi então aplicado na prática, o que havia sido visto na teoria:

a) os símbolos terminais da gramática regular se transformam no alfabeto de entrada do autômato finito;

b) os não terminais da gramática se tornam os estados do autômato;

c) o símbolo inicial da gramática passa a ser o estado inicial do autômato;

d) regras na forma $X \rightarrow aY$ (onde X e Y são símbolos não terminais da gramática e a é símbolo terminal) dão origem no autômato a uma transição de X para Y com símbolo de entrada a;

e) regras na forma $X \rightarrow a$ originam no autômato uma transição de X para um novo estado Z, com símbolo de entrada a, sendo Z um estado final; e

f) regras na forma $X \rightarrow \varepsilon$ determinam apenas que X é um estado final.

Aplicando-se estas regras, obtém-se um autômato finito reconhecedor da linguagem gerada pela gramática regular informada.

A Figura 35 mostra a parte do código que utiliza estas regras para a geração do autômato finito a partir da gramática regular informada pelo usuário. A função *determinaPalavras*, cuja chamada pode ser visualizada no código, recebe a parte à esquerda e à direita das produções e retorna três palavras separadas por vírgula (X,a,Y) correspondentes a $X \rightarrow aY$, que serão tratadas conforme a sua forma. Cada regra irá

gerar uma transição no autômato finito e desta forma ele vai sendo construído da primeira até a última produção.

```
for i := 0 to strgrid_gr1.RowCount - 1 do
  if strgrid_gr1.Cells[0,i] <> ''
then begin
  palavra0 := '';
  palavra1 := '';
  palavra2 := '';
  achou1virg := false;
  achou2virg := false;

  palavra := determinaPalavras(strgrid_gr1.Cells[0,i],
             strgrid_gr1.Cells[2,i]);

  for j := 1 to Length(palavra) do
  begin
    if (palavra[j] <> ',')  and
       (achou1virg = false) and
       (achou2virg = false)
    then palavra0 := palavra0 + palavra[j]
    else if palavra[j] <> ','
         then if achou2virg = false
              then palavra1 := palavra1 + palavra[j]
              else palavra2 := palavra2 + palavra[j];

    if palavra[j] = ','
    then begin
      if achou1virg = false
      then achou1virg := true
      else achou2virg := true;
    end;
  end;

  if (palavra1 = 'í') or
     (palavra1 = 'Í')
  then begin
    if (estFinais <> '')
    then estFinais := estFinais + ',';

    estFinais := estFinais + strgrid_gr1.Cells[0,i];
  end
  else if (palavra2 = '') then
  begin
    novoestado := 'Z';

    estAutomato := estAutomato + ',' + novoestado;

    if (estFinais <> '')
    then estFinais := estFinais + ',';

    estFinais := estFinais + novoestado;

    if ListaConexoes.VerificaConexoesRepetidas(Trim(palavra0),
       Trim(novoestado), Trim(palavra1)) = false
    then ListaConexoes.NovaConexao(Trim(palavra0), Trim(novoestado),
         Trim(palavra1));
  end
  else if ListaConexoes.VerificaConexoesRepetidas(Trim(palavra0),
          Trim(palavra2), Trim(palavra1)) = false
       then ListaConexoes.NovaConexao(Trim(palavra0), Trim(palavra2),
            Trim(palavra1));
   if (palavra1 <> '') and //Cria o alfabeto do autômato
      (palavra1 <> 'í') and
      (palavra1 <> 'Í')
   then ListaAlfabeto.NovoAlfabeto(palavra1);
end;
for i := 1 to Length(estAutomato) do //Cria os estados do autômato
begin
  if estAutomato[i] <> ','
  then estado := estado + estAutomato[i];

  if (estAutomato[i] = ',') or
     (i = Length(estAutomato))
  then begin
    posY := ListaEstados.ContaEstados+1;

    if (posY mod 2 = 0)
    then posY := posY + 30;

    ListaEstados.NovoEstado(Trim(estado), Trim(estFinais),
                            Trim(ed_GrSimbIni.Text),
                            ValorX(ListaEstados.ContaEstados+1),
                            ValorY(posY));

    estado := '';
  end;
end;
```

Figura 35. Código para geração do autômato finito a partir da gramática regular

6.3.2 Módulo de Expressões Regulares

O algoritmo citado na fundamentação teórica que trata da geração de um autômato finito a partir de uma expressão regular proposto por Crespo (2001) não foi aplicado no desenvolvimento deste módulo. O autor deste algoritmo apresenta a transformação em forma de figuras e quando se pretende adaptá-lo para uma linguagem de programação, aparecem questões complexas e difíceis de transformar em algoritmo. Por este motivo, optou-se por criar uma nova lógica.

Quando o usuário solicita a geração do autômato finito a partir da expressão regular informada são feitas todas as validações necessárias. Caso existam erros, mensagens são disparadas e não é possível prosseguir até que tudo esteja correto. Caso não existam erros, a *procedure chamaConstrucao,* representada na Figura 36, é invocada. Esta *procedure* recebe uma palavra, que no primeiro momento é a expressão regular informada pelo usuário e dependendo das operações presentes, tal *procedure* pode ser invocada em momentos posteriores, recebendo partes dividas da expressão.

```
vaiProOu := false;
i := 1;
par1 := 0;
par2 := 0;

while (i < Length(palavra) + 1) and  // verifica se é ou fora dos parênteses
      (vaiProOu = false ) do         // a(ab)*+c, a+c, (a+b)+c
begin
  if palavra[i] = '('
  then par1 := par1 + 1
  else if palavra[i] = ')'
  then par2 := par2 + 1;

  if (par1 = par2) and
     (palavra[i] = '+')
  then vaiProOu := true;

  i := i + 1;
end;

if vaiProOu = false
then begin
  if pos('(',palavra) = 0
  then  // expressão é 'e'  aabbcc
    criaER(palavra, 'e', nIniAux, nFimAux, ind)
  else  // tem parenteses
    criaER(palavra, '(', nIniAux, nFimAux, ind);
end
else begin     // expressão é ou   a+b
  if nFim = -1 // define um estado final para cada parte do ou terminar
  then begin   // neste estado
    nFimAux := 111111;

    while pos(IntToStr(nFimAux), auxFimOu) <> 0 do
      nFimAux := nFimAux + 1;

    auxFimOu := auxFimOu + IntToStr(nFimAux);

  end
  else nFimAux := nFim;

  criaER(palavra, 'ou', nIniAux, nFimAux, ind);
end;
```

Figura 36. Parte da *procedure chamaConstrucao*

A *procedure chamaConstrucao* verifica três possibilidades na palavra recebida, nesta ordem: operação de união fora dos parênteses (*a+(cd)*+e*), presença de parentes (*a(b+c)*d*) e concatenação (*abc*d^e*). Caso seja uma operação de união, a *procedure criaER* é invocada com a palavra e a indicação de união, caso a operação principal não seja união e haja parênteses, a *procedure criaER* é invocada com a palavra e a indicação de existência de parênteses e o mesmo procedimento para a concatenação.

A *procedure criaER* realiza o tratamento da palavra de acordo com a indicação de operação recebida pela *chamaConstrucao*. Caso seja uma operação de união fora dos parênteses, a palavra é divida em partes, pois esta operação é composta de expressões menores. As partes podem ser apenas um símbolo, concatenações e repetições com ou sem parênteses. Por se tratarem de novas pequenas expressões, cada parte é enviada separadamente para a *procedure chamaConstrucao* para que seja tratada novamente e volte para a criaER de forma mais reduzida. Além disso, no autômato finito, estas partes devem sair do mesmo estado de partida, por exemplo, a expressão *ab+(cd)*+e* é composta por três expressões: *ab*, (*cd)** e *e* e no autômato finito todas elas devem partir do mesmo estado. Isto também é tratado na *criaER*. O tratamento da operação de união fora dos parênteses nesta *procedure* está representado na Figura 37.

```
else if opcao = 'ou' // a+b
then begin
  p := 0;
  par1 := 0;
  par2 := 0;

  SetLength(parte,1);

  for i := 1 to Length(palavra) do    // separa as partes do 'ou'
  begin
    if palavra[i] = '('
    then par1 := par1 + 1
    else if palavra[i] = ')'
    then par2 := par2 + 1;

    if ((palavra[i] = '+') and
        (par1 <> par2))     or
        (palavra[i] <> '+')
    then parte[p] := parte[p] + palavra[i]

    else if par1 = par2
    then begin
      p := p + 1;
      SetLength(parte,p+1);
    end;
  end;

  for i := 0 to p do // trata as partes do 'ou'
  begin
    ListaConexoes.NovaConexao('Q' + IntToStr(nIni),
                              'Q' + IntToStr(ListaEstados.ContaEstados), 'î')
    ListaEstados.NovoEstado('Q' + IntToStr(ListaEstados.ContaEstados),'','',
                            0,0);
    ListaEstados.NovoEstado('Q' + IntToStr(ListaEstados.ContaEstados),
                            '','',0,0);

    // altera nomes dos estados definidos como > 111111
    for i := 0 to ListaConexoes.ContaConexoes - 1 do
    begin
      if Conexao[i].estado_origem = 'Q' + IntToStr(nFim)
      then Conexao[i].estado_origem := 'Q' +
                                       IntToStr(ListaEstados.ContaEstados - 1);

      if Conexao[i].estado_destino = 'Q' + IntToStr(nFim)
      then Conexao[i].estado_destino := 'Q' +
                                       IntToStr(ListaEstados.ContaEstados - 1);

    end;
  end
```

Figura 37. Operação de união na *procedure criaER*

O autômato finito reconhecedor da linguagem gerada pela expressão *ab+(cd)*+e* pode ser observado na Figura 38.

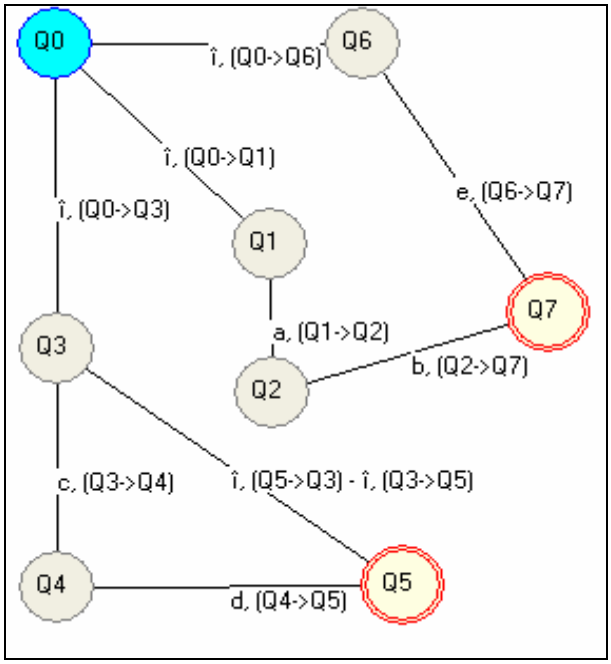


Figura 38. Autômato finito reconhecedor da linguagem gerada pela expressão *ab+(cd)*+e*

Caso seja uma palavra sem operação de união fora dos parênteses e que tenha parênteses, a operação principal é uma concatenação e é realizada novamente a divisão das partes da expressão, por exemplo, a expressão *a(b+c)*d* é tratada como sendo composta por três partes: *a* concatenado com *(b+c)** concatenado com *d*. Após dividas, cada parte é encaminhada para a *chamaConstrucao*, que verificará a operação em questão e encaminhará novamente para a criaER. Neste caso, o caractere *a* é encarado como uma concatenação, a expressão *b+c* como uma união, o caractere * é tratado como uma repetição do início até o final do *ou* e o caractere *b* é encarado também como uma concatenação. O tratamento de parênteses nesta *procedure* está representado nas Figuras 39 e 40.

```
  else if opcao = '(' // parênteses - mudam precedência dos operadores
  then begin
    SetLength(parte,1);
    p := 0;
    par1 := 0;
    par2 := 0;
    for i := 1 to Length(palavra) do // separa as partes do '('
    begin
      if palavra[i] = '('
      then par1 := par1 + 1
      else if palavra[i] = ')'
      then par2 := par2 + 1;
      if ((palavra[i-1] = '*')  or
          (palavra[i-1] = '^')) and // quebra depois de armazenar o asterisco
         (palavra[i-2] = ')')   and
         (par1 = par2)
      then begin
        p := p + 1;
        SetLength(parte,p+1);
      end;
      parte[p] := parte[p] + palavra[i];
      if ((palavra[i+1] = '(')   or
         ((palavra[i] = ')')     and
          (palavra[i+1] <> '*')  and
          (palavra[i+1] <> '^')) and
          (i <> Length(palavra)))
      then begin
        p := p + 1;
        SetLength(parte,p+1);
      end;
    end;
    for i := 0 to p do   // trata as partes do '('
    begin
      par1 := 0;
      repete := false;

      for k := 1 to Length(parte[i]) do
      begin
        if parte[i][k] = '('
        then par1 := par1 + 1;
      end;
      if pos(')*',parte[i]) <> 0 // se tem *,cria loop
      then begin
        Delete(parte[i],pos(')*',parte[i]),1);
        repete := true;
      end;
      fecho := false;
      if pos(')^',parte[i]) <> 0 // se tem ^, vai criar uma vez, depois fazer loop
      then begin
        Delete(parte[i],pos(')^',parte[i]),1);
        fecho := true;
        repete := true;
      end;
      if (par1 = 1)           and
         (parte[i][1] = '(') and
         (parte[i][Length(parte[i])] = ')')
      then begin
        Delete(parte[i],1,1);
        Delete(parte[i],Length(parte[i]),1);
      end;
      if fecho // cria pelo menos uma vez
      then chamaConstrucao(parte[i], ListaEstados.ContaEstados-1, -1, 0);
      nIniAux := ListaEstados.ContaEstados-1;
      chamaConstrucao(parte[i], ListaEstados.ContaEstados-1, -1, 0);
      if repete   // transição de ida para passar direto e de volta
      then begin //    para poder fazer o loop
        ListaConexoes.NovaConexao('Q' + IntToStr(nIniAux), 'Q' +
                                  IntToStr(ListaEstados.ContaEstados-1), 'î');
        ListaConexoes.NovaConexao('Q' + IntToStr(ListaEstados.ContaEstados-1),
                                  'Q' + IntToStr(nIniAux), 'î');
      end;
    end;
  end;
end;
```

Figura 39. Tratamento dos parênteses na *procedure criaER*

O autômato finito reconhecedor da linguagem gerada pela expressão *a(b+c)*d* pode ser visualizado na Figura 40.

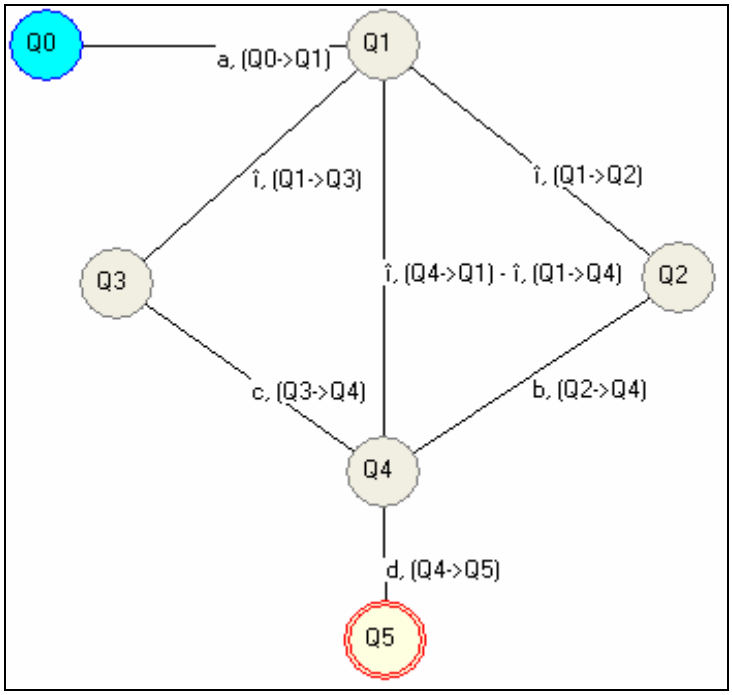


Figura 40. Autômato finito reconhecedor da linguagem gerada pela expressão *a(b+c)*d*

A operação de concatenação é a mais simples de ser tratada pela *criaER*. A *procedure* varre a palavra recebida e cria uma conexão após outra no autômato finito para cada caractere da palavra. Ao encontrar uma repetição (*), é criada uma conexão com mesmo estado de partida e de chegada, ao invés da conexão para o próximo estado. O mesmo acontece para o fecho transitivo (^), porém é criada uma conexão com o caractere primeiramente para o próximo estado e depois é criada a repetição. O tratamento das operações de concatenação com ou sem repetição e / ou fecho transitivo nesta *procedure* está representado na Figura 41.

```
if opcao = 'e'     // concatenação
then begin
  for i := 1 to Length(palavra) do
  begin
    if (palavra[i] <> '*') and
       (palavra[i] <> '^')
    then begin
      if (ind = 1) and // veio do ou pra jogar o estado final no nFim
        ((i = Length(palavra)) or
         ((i = Length(palavra) -1) and
          ((palavra[i+1] = '*') or
           (palavra[i+1] = '^'))))
      then begin
        if (palavra[i+1] = '*') or
           (palavra[i+1] = '^')
        then begin // cria a letra em um loop
          if palavra[i+1] = '^'  // cria no mínimo uma repetição
          then begin
            ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(ListaEstados.ContaEstados),
                           palavra[i]);

            ListaEstados.NovoEstado('Q' + IntToStr(ListaEstados.ContaEstados),
                                    '', '',0,0);
          end;
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           palavra[i]);
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(nFim), 'î');
        end
        else begin
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(nFim),palavra[i]);
        end;
      end
      else begin
        if (palavra[i+1] = '*') or
           (palavra[i+1] = '^')
        then begin // cria a letra em um loop
          if palavra[i+1] = '^'  // cria no mínimo uma repetição
          then begin
            ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(ListaEstados.ContaEstados),
                           palavra[i]);

            ListaEstados.NovoEstado('Q' + IntToStr(ListaEstados.ContaEstados),
                                    '', '',0,0);
          end;
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(ListaEstados.ContaEstados-1),
                           palavra[i]);
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(ListaEstados.ContaEstados), 'í');
        end
        else begin
          ListaConexoes.NovaConexao(
                           Estados[ListaEstados.ContaEstados-1].nome_estado,
                           'Q' + IntToStr(ListaEstados.ContaEstados),
                           palavra[i]);
        end;

        ListaEstados.NovoEstado('Q' + IntToStr(ListaEstados.ContaEstados),'',
                                '',0,0);
      end;
      ListaAlfabeto.NovoAlfabeto(palavra[i]);
    end;
  end;
end
```

Figura 41. Operação de concatenação, repetição e fecho transitivo na *procedure criaER*

A expressão regular *abc*d^e* representa concatenação com repetição e fecho transitivo e o autômato reconhecedor da linguagem gerada por ela é apresentado na Figura 42.

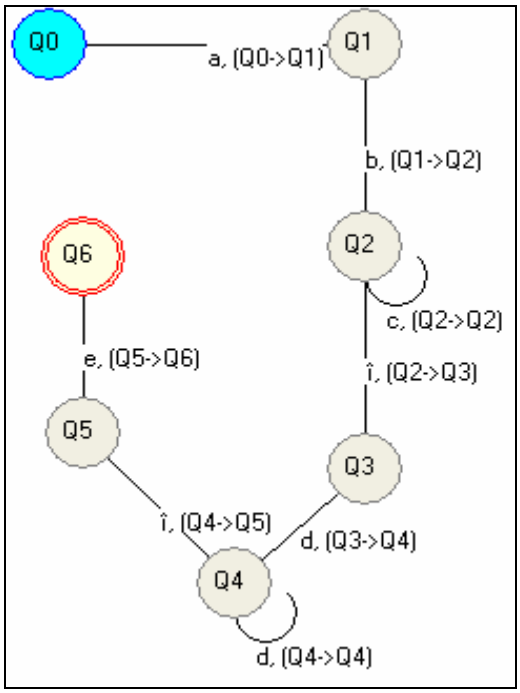


Figura 42. Autômato finito reconhecedor da linguagem gerada pela expressão *abc*d^e*

Os movimentos vazios (î) encontrados nos autômatos apresentados visam facilitar a construção e proporcionar uma visualização mais simplificada dos mesmos, pois como foi demonstrado na Figura 11, comparando um autômato finito com movimentos vazios e um sem, o que possui movimentos vazios é mais simples de construir e entender.

6.4 RESULTADOS OBTIDOS

6.4.1 Módulo de Gramáticas Regulares

Neste módulo foi criada a estrutura da gramática para que o usuário possa informá-la, sendo o primeiro campo para a definição dos símbolos não terminais, o segundo para os terminais e o terceiro para o símbolo inicial. As regras de produção devem ser definidas em uma *String Grid*, sendo uma linha para cada produção da gramática, conforme pode ser observado na Figura 43.

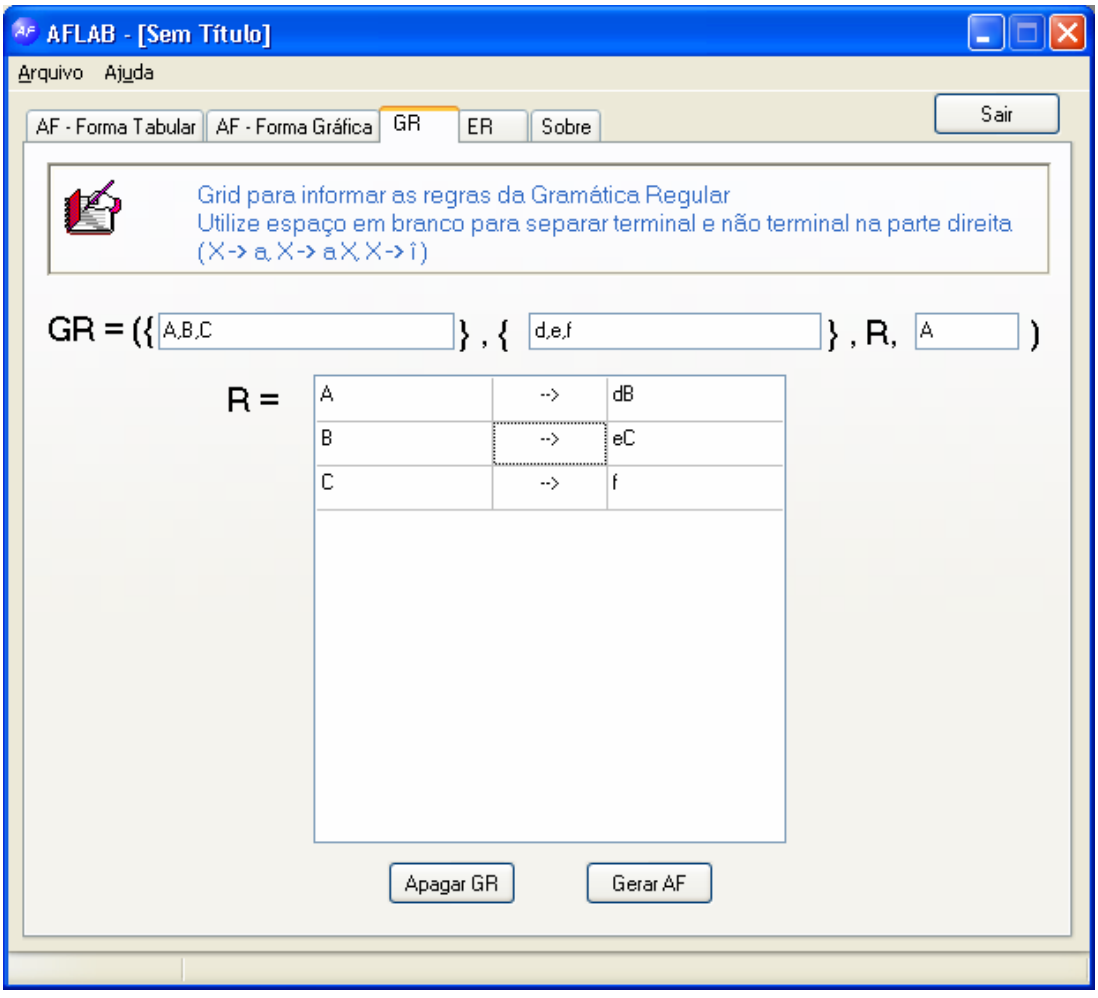


Figura 43. AFLAB – Módulo de gramáticas regulares

Os não terminais da gramática devem ser definidos, obrigatoriamente, com letras maiúsculas. Os terminais, por sua vez, devem ser formados por apenas um caractere, sendo permitido letras minúsculas, números e símbolos. Nenhum dos dois campos pode conter espaços vazios. Tais restrições foram validadas na implementação.

O usuário deve informar primeiramente os não terminais e terminais, seguidos do símbolo inicial. Ao sair do campo de símbolo inicial, o programa verificará se o símbolo inicial informado pertence aos não terminais da gramática, caso não pertença dará uma mensagem de erro e focará novamente no campo de símbolo inicial. É obrigatório informar primeiramente estes três campos antes de seguir para o preenchimento das regras da gramática.

Após isso, o usuário inicia a declaração das regras, que serão validadas:

a) em relação ao primeiro símbolo da regra, que deve ser o símbolo inicial da gramática;
b) quanto a presença de espaços em branco e símbolos não pertencentes aos terminais e não terminais da gramática;
c) quanto as formas permitidas para as regras ($X \rightarrow aY$, $X \rightarrow a$ e $X \rightarrow \hat{\imath}$), onde X e Y são não terminais, a é terminal e $\hat{\imath}$ é o símbolo que representa o vazio.

Estando tudo certo em relação à definição da gramática, o usuário poderá solicitar a criação do autômato finito.

Após criação do autômato, o programa mostrará o módulo “AF – Forma Gráfica” para que o autômato possa ser visualizado graficamente, podendo ser selecionada também a guia “AF – Forma Tabular” possibilitando a visualização na forma tabular, integração que o AFLAB já garantia na primeira versão.

Caso o usuário opte por desistir da gramática em que estiver trabalhando, o software possui um botão para apagá-la (“Apagar a GR”), que limpará todos os campos informados, bem como o autômato finito, caso este já tenha sido criado.

O autômato finito gerado a partir da gramática regular da Figura 43 pode ser visualizado na sua forma tabular na Figura 44 e na sua forma gráfica na Figura 45.

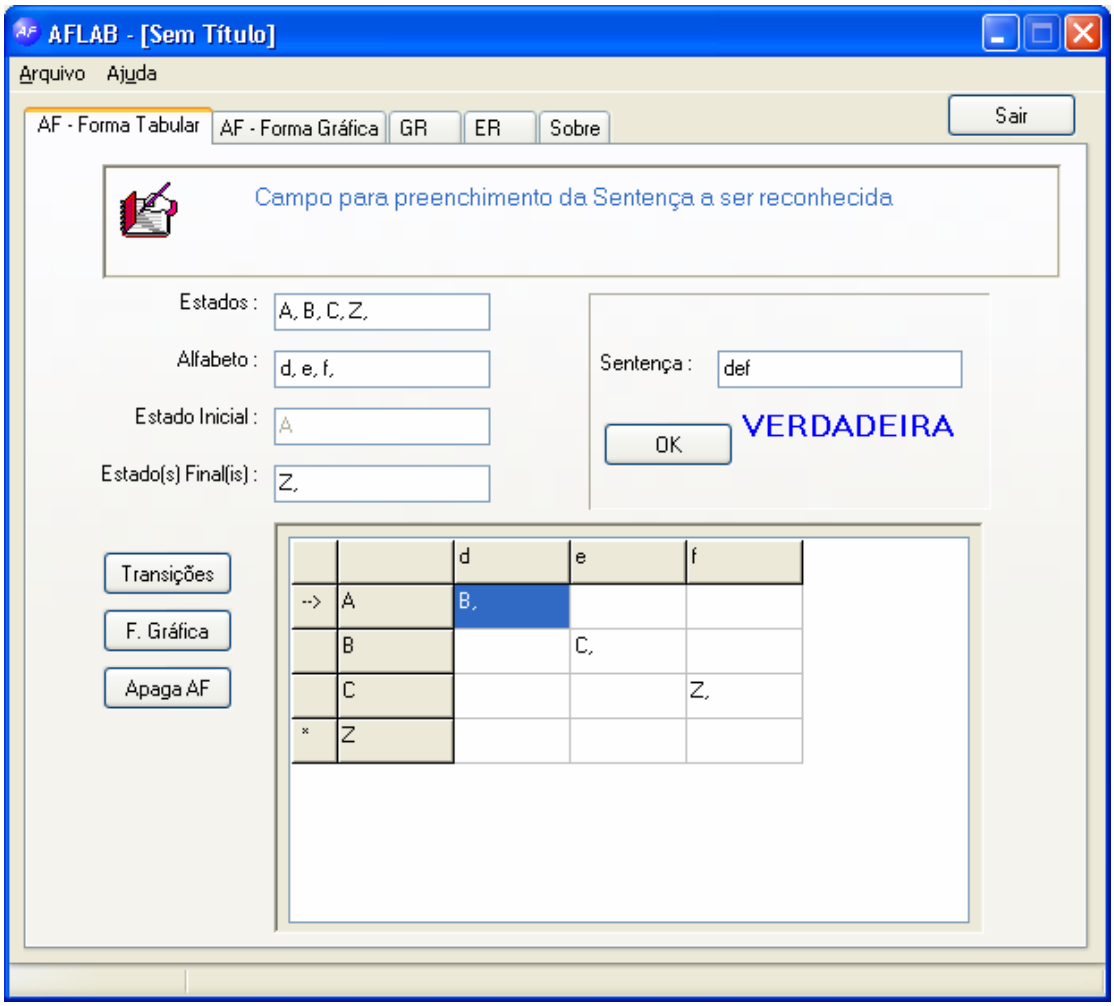


Figura 44. Autômato finito a partir da gramática regular – Forma Tabular

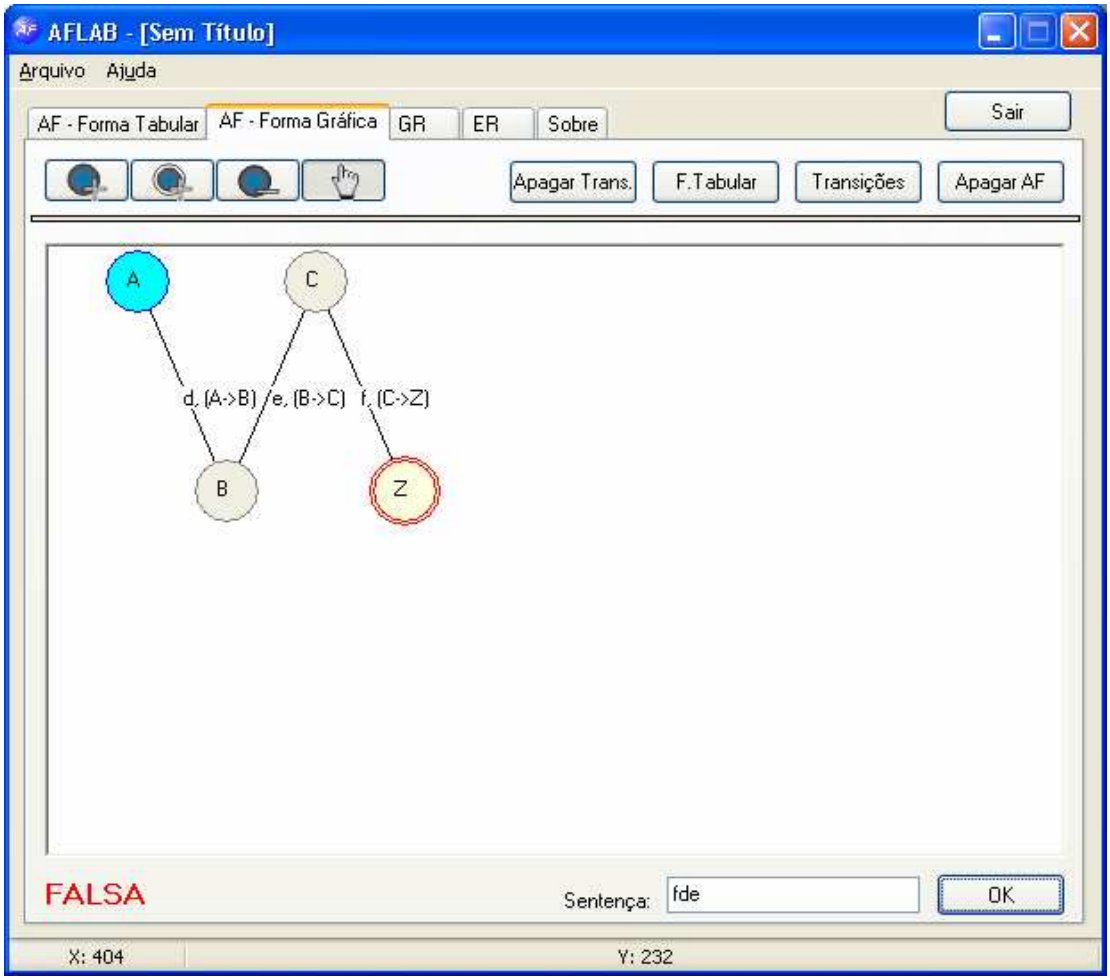


Figura 45. Autômato finito a partir da gramática regular – Forma Gráfica

Caso o usuário deseje salvar a gramática regular informada, é possível fazer uso do menu para isso. Esta opção permite salvar a gramática e o autômato finito, se este já tiver sido gerado.

Vários testes foram realizados no módulo de Gramáticas Regulares com o propósito de verificar o seu correto funcionamento. Com isto, pôde-se constatar que os objetivos quanto a este módulo foram alcançados, já que não foram encontrados erros na construção de autômatos finitos reconhecedores das linguagens geradas pelas gramáticas regulares informadas.

Após o término da implementação do módulo de Gramáticas Regulares, foi dado início ao módulo de Expressões Regulares.

6.4.2 Módulo de Expressões Regulares

A tela para informar a expressão regular na ferramenta AFLAB é composta apenas por um campo para que a expressão seja informada e um botão no qual o usuário solicita a geração do autômato finito reconhecedor da linguagem definida pela expressão. O módulo de Expressões Regulares pode ser observado na Figura 46.

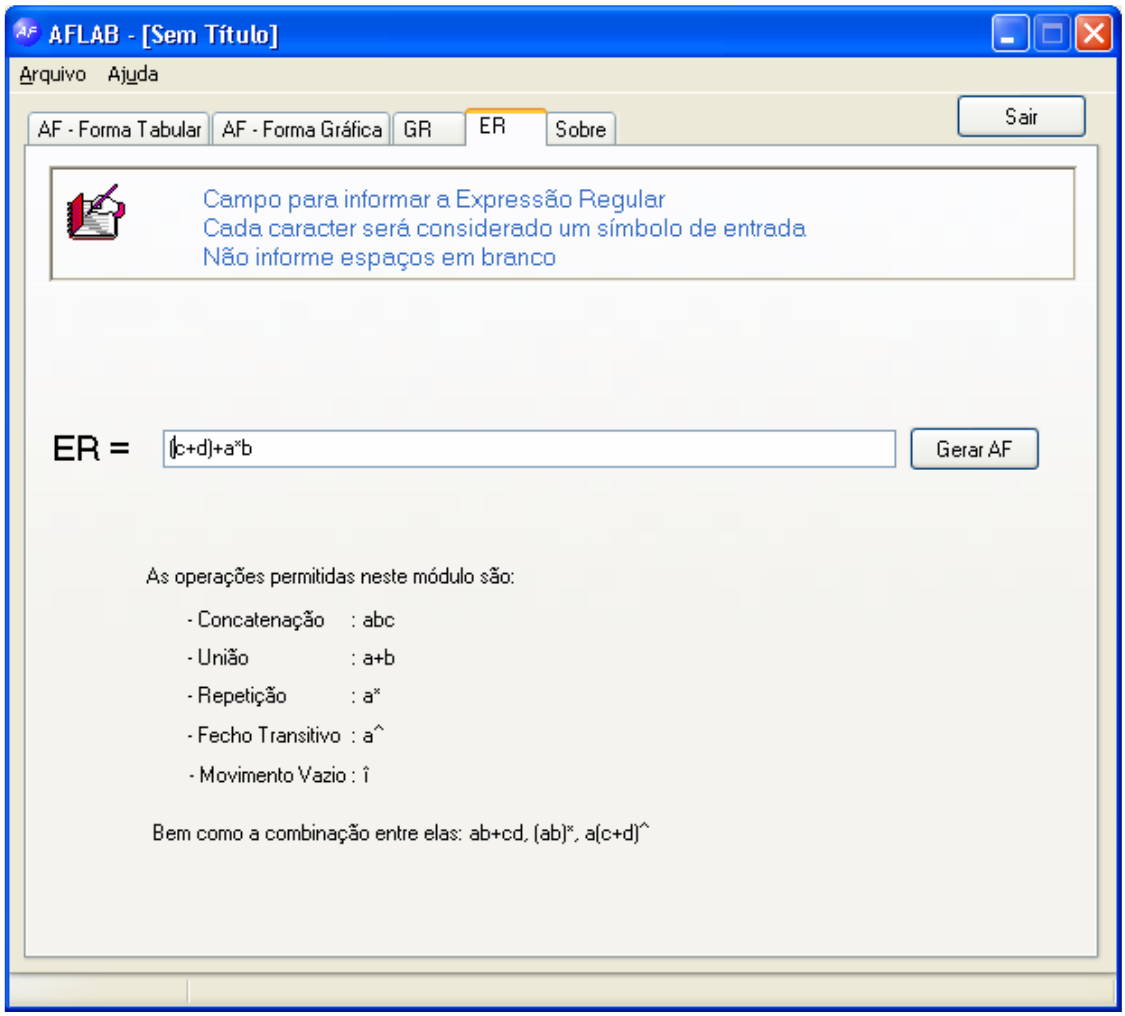


Figura 46. AFLAB – Módulo de Expressões Regulares

As operações permitidas neste módulo são:

a) concatenação: não utiliza nenhum caractere adicional, apenas a sucessiva utilização das letras e / ou números desejados. Exemplos: abc, 123, ab12;

b) união: é caracterizada pela combinação dos caracteres da expressão regular com o símbolo + , lido como ‘ou’ e significa que o autômato finito percorrerá um caminho ou outro. Exemplos: a+b, ab+cd, 12+ab;

c) repetição: caracteriza-se pela presença do símbolo * e significa a ausência ou repetição de uma determinada parte da expressão regular. Exemplos: ab*, a(ab)*, 0(1+2)*;

d) fecho transitivo: nesta ferramenta será representado pelo símbolo ^ devido a impossibilidade de representar o $^{+}$. Sua função é semelhante a repetição, porém garante a ocorrência da parte estipulada pelo menos uma vez. Exemplos: ab^, a(ab)^, 0(1+2)^.

Nas seguintes situações, após o clique no botão “Gerar AF”, o sistema mostrará uma mensagem de erro, não sendo possível prosseguir até que o erro na expressão regular seja resolvido:

a) presença de um caractere diferente dos permitidos nas operações;

b) número incorreto de parênteses, por exemplo, dois parênteses abrindo e apenas um fechando;

c) ordem incorreta de parênteses, ou seja, aparecer primeiro um parêntese que fecha e depois um parêntese que abre;

d) presença de um parêntese que abre no início e um parêntese que fecha no final da expressão, sem necessidade. Por exemplo, os parênteses em *(ab)* não precisam aparecer nesta expressão;

e) presença dos símbolos +, * ou ^ no início da expressão regular e do símbolo + no final;

f) seqüências **, ^^, ++, +* , +^, *(*, *(^* ou *(+* em qualquer parte da expressão;

g) haver parênteses antes e depois de um único símbolo, pois não há necessidade;

h) concatenação com parênteses antes e depois sem haver repetição, por exemplo, a expressão *(abcd)+e* é igual a *abcd+e*, ou seja, sem necessidade de parênteses.

Além das mensagens de erro causadas pelos tópicos apresentados, existe ainda outra, proveniente de uma restrição que foi feita neste módulo: não é possível informar parênteses dentro de parênteses, por exemplo, *a(a+(ab)*)*.

Após todas as validações, inicia-se então o tratamento da expressão regular para geração do autômato finito reconhecedor da sua linguagem. Assim como no módulo de Gramáticas Regulares, após a criação do autômato, o foco irá para o módulo "AF – Forma Gráfica", para que o usuário possa conferir o resultado graficamente, podendo ele alternar também para o módulo "AF – Forma Tabular" conforme sua necessidade.

O resultado da transformação da expressão demonstrada na Figura 46 pode ser visto na Figura 47, que mostra o autômato gerado em sua forma tabular e na Figura 48, que mostra o mesmo autômato em sua forma gráfica.

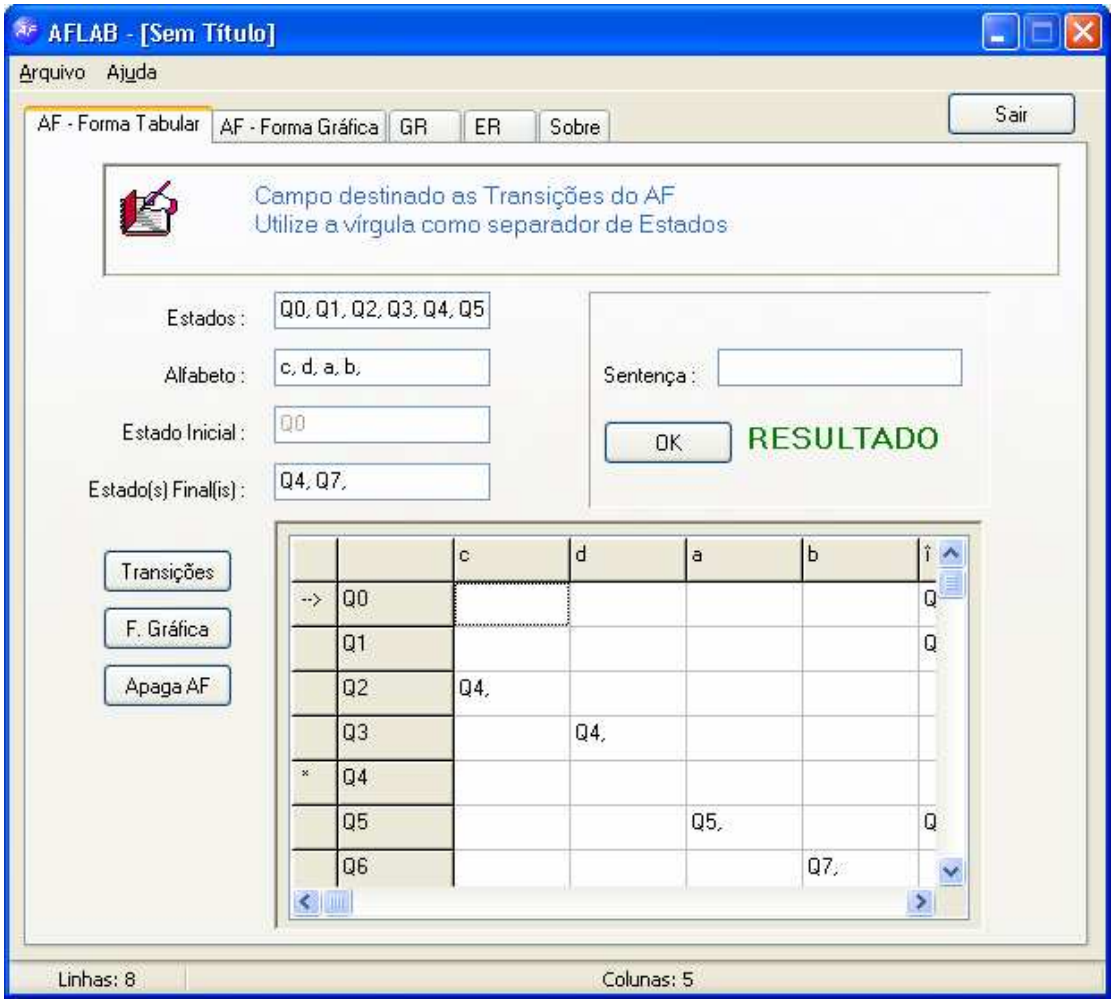


Figura 47. Autômato finito a partir da expressão regular – Forma Tabular

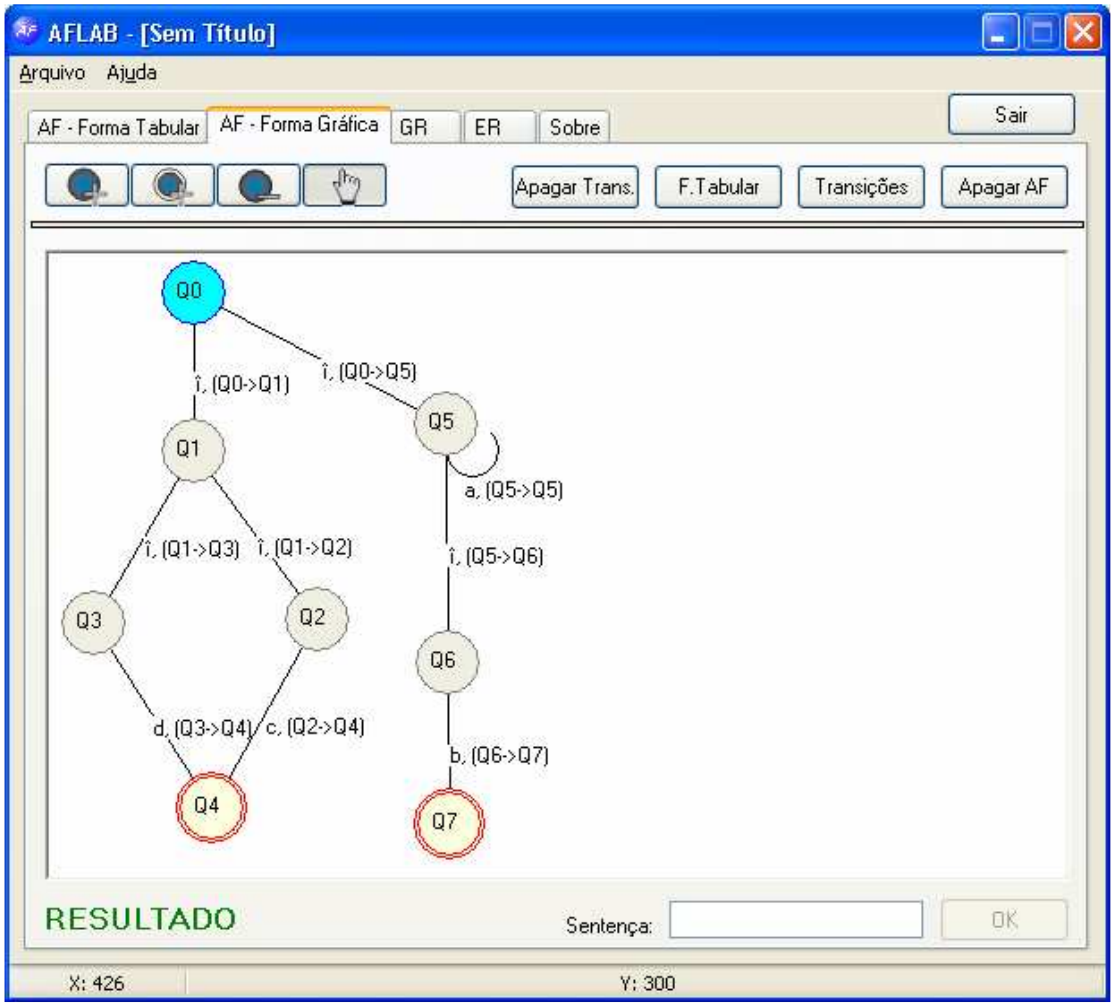


Figura 48. Autômato finito a partir da expressão regular – Forma Gráfica

Os autômatos finitos gerados no módulo de Expressões Regulares que possuem movimentos vazios não são capazes de reconhecer sentenças. Isso acontece devido ao fato de que a primeira versão do AFLAB não contemplava movimentos vazios e eles não foram tratados na *procedure* que realiza o reconhecimento de sentenças.

Após o usuário informar a expressão regular é possível salvá-la por meio do menu. Esta opção salvará não somente a expressão em questão, mas também o autômato finito, se já houver sido gerado.

Durante e após a implementação, foram realizados diversos testes neste módulo para verificar o funcionamento e a possível existência de erros. O

funcionamento ficou como o esperado e erros não foram encontrados. Ficando também constatado que para este módulo os objetivos foram alcançados.

CONCLUSÃO

Ainda é notável nos dias atuais a falta de ferramentas em determinadas áreas de conhecimento. Por esta razão, vários novos softwares para fins educacionais têm surgido em diversas instituições para tentar suprir esta necessidade e aos poucos ir mudando este cenário.

O AFLAB partiu do princípio de se criar um software que suprisse a necessidade da manipulação de autômatos finitos e aplicação prática dos conceitos relacionados a eles. Após a conclusão de sua primeira versão, que abrangeu autômatos finitos na forma gráfica e tabular e o reconhecimento de sentenças por meio destes autômatos, outras necessidades foram levantadas e novas funcionalidades passaram a ser implementadas.

Os novos módulos desenvolvidos no AFLAB que transformam uma gramática regular ou expressão regular em um autômato finito reconhecedor da linguagem gerada por estes formalismos enriqueceram ainda mais a ferramenta, contribuindo para que ela ficasse mais completa. Além disso, há também uma contribuição no que diz respeito aos softwares existentes nesta área específica, pois não são muitos os que realizam estas transformações de forma satisfatória.

Pelo fato de ser continuação de uma ferramenta, foram mantidas as características da primeira versão no que diz respeito ao uso das classes existentes (Testados, Tconexao, Talfabeto) e também quanto à interface gráfica.

O módulo de Gramáticas Regulares foi implementado com maior facilidade, pois havia de forma clara na fundamentação teórica os passos necessários para se obter um autômato finito partindo de uma gramática regular. Esses passos mostram o que

deve ser criado no autômato a partir dos símbolos e regras de produção da gramática regular.

Em relação à obtenção de autômatos finitos a partir de expressões regulares, foram encontrados alguns passos que demonstram na prática como fazer essa transformação, mas que ficaram um tanto confuso no momento da implementação. Por este motivo, foi criada uma lógica própria para o desenvolvimento do módulo de Expressões Regulares e foi possível também atingir um resultado satisfatório quanto a este módulo.

Após o término de todo o estudo acerca do tema e da implementação dos novos módulos, testes foram realizados e pôde-se perceber então que os objetivos deste trabalho foram alcançados. A partir de agora o AFLAB conta com dois novos módulos que funcionam perfeitamente e devem atender as expectativas dos usuários.

Finalizando, sugere-se para trabalhos futuros a expansão do módulo de Expressões Regulares, para que seja possível fazer o caminho inverso, ou seja, a obtenção da expressão regular a partir de um autômato finito informado. Ainda neste módulo, sugere-se implementar o reconhecimento de sentenças em autômatos finitos com movimentos vazios. Outra sugestão é que este software, que foi desenvolvido em *object pascal*, por meio da plataforma Borland® Delphi™ Enterprise, seja transformado para uma linguagem multiplataforma, permitindo assim que todos os usuários interessados possam utilizá-lo independente do sistema operacional que fazem uso.

REFERÊNCIAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores**: princípios, técnicas e ferramentas. Rio de Janeiro: Livros Técnicos e Científicos, 1995. Tradução de Daniel de Ariosto Pinto.

CRESPO, Rui Gustavo. **Processadores de linguagens**: da concepção à implementação. Lisboa: IST Press, 2001.

DARRELL, Raymond; WOOD, Derick. **Grail**. University of Waterloo, Canadá. 1995. Disponível em: <http://www.cse.ust.hk/faculty/dwood/.grail>. Acesso em: 17 out. 2007.

DOGNINI, Marlon José. **EduLing - Software Educacional para Linguagens Regulares.** In: XIV Simpósio Brasileiro de Informática na Educação, 2003, Rio de Janeiro, RJ. Disponível em: <http://www.nce.ufrj.br/sbie2003/publicacoes/paper24.pdf>. Acesso em: 19 set. 2007.

GAIDZINSKI, Marco Aurélio. **Ambiente de criação e manipulação de autômatos finitos na forma gráfica ou tabular para o reconhecimento de sentenças**. 2007. 61 f. Trabalho de Conclusão de Curso (Bacharelado) – Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2007.

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

LOUDEN, Kenneth C. **Compiladores:** princípios e práticas. São Paulo: Pioneira Thomson Learning, 2004. Tradução de Flávio Soares Correia da Silva.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos**. 5. ed. Porto Alegre: Sagra Luzzatto, 2005.

RODGER, Susan H. **Java Formal Language and Automata Package (JFLAP)**. Rensselaer Polytechnic Institute, RPI, Estados Unidos, 1990. Disponível em: <http://www.jflap.org>. Acesso em: 19 set. 2007.

SANTOS, Gilliard Lopes dos. Máquinas de Estados Hierárquicas em Jogos Eletrônicos. In: _____. **Máquinas de Estados em Jogos Eletrônicos**. Rio de Janeiro,

2004. Cap 2. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/4711_3.PDF?NrOcoSis=11468&CdLinPrg=pt>. Acesso em: 30 ago. 2007.

STEELE, Oliver. **reAnimator**. Amherst, Massachusetts. 2006. Disponível em: <http://osteele.com/tools/reanimator>. Acesso em: 17 out. 2007.

VIEIRA, Luiz Filipe Menezes; VIEIRA, Marcos Augusto Menezes; VIEIRA, Newton José. **Language Emulator, uma ferramenta de auxílio no ensino de Teoria da Computação**. In: II Workshop de Educação em Computação e Informática do Estado de Minas Gerais, 2003, Poços de Caldas, MG. Disponível em: <http://www.inf.pucpcaldas.br/eventos/weimig2003/ArtigosWEIMIG2003/WEIMIG2003LuizFilipeMenezes.pdf>. Acesso em: 14 mar. 2007.

VIEIRA, Newton José. **Introdução aos Fundamentos da Computação**: Linguagens e Máquinas. São Paulo: Pioneira Thomson Learning, 2006.

BIBLIOGRAFIA COMPLEMENTAR

CANTÚ, Marco. **Dominando o Delphi 6**: a bíblia. São Paulo: Makron Books, 2002. Tradução de João Eduardo Nóbreg Tortello.

FOWLER, Martin; SCOTT, Kendal. **UML essencial**: um breve guia para a linguagem padrão de modelagem de objetos. Porto Alegre: Bookman, 2000. Tradução de Vera Pezerico e Christian Thomas Price.

JARGAS, Aurélio Marinho. **Expressões Regulares**: uma abordagem divertida. São Paulo: Novatec, 2006.

LEWIS, Harry R.; PAPADIMITRIOU, Christos H. **Elementos de teoria da computação**. 2. ed. Porto Alegre: Bookman, 2004.

STACHOVOSKI, Lislaine. **AGASTUDIO**: ambiente de criação e manipulação de animações baseadas em autômatos finitos. 2005. 81 f. Trabalho de Conclusão de Curso (Bacharelado) – Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2005.

TAYLOR, Ralph Gregory. **Models of computation and formal languages**. New York: Oxford University Press, 1998.

APÊNDICE – ARTIGO

Gramáticas regulares e expressões regulares: Um ambiente de manipulação no AFLAB

Aline Teixeira[1], Christine Vieira Scarpato[1]

[1]Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC)
Criciúma – SC – Brasil

linyteixeira@yahoo.com.br, cvi@unesc.net

Abstract. *AFLAB is a software that allows the deterministic and non-deterministic finite automata construction on two different representation forms: diagrams and tables. It also allows recognizing sentences through these automata. This paper presents the addition of two new features on AFLAB: the transformation of a regular expression into a finite automaton and the conversion of a regular grammar into a finite automaton. However, to achieve the goals of the new modules, a study about finite automata, regular grammar, regular expressions and algorithms to do the transformation between formalisms was made. Both added modules are described here, as well as the implementation details about them and their features. In general, the new modules work perfectly, coming up to expectations and keeping the AFLAB characteristics, with explanatory texts and a simple and easy to use interface.*

Resumo. *O AFLAB é um software que permite a construção de autômatos finitos determinísticos e não determinísticos em sua forma gráfica ou tabular, permitindo ainda o reconhecimento de sentenças por meio destes autômatos. Este trabalho apresenta uma continuação do AFLAB e consta da adição de dois módulos nesta ferramenta: um para transformação de expressões regulares em autômato finito e o outro para obtenção de autômato finito a partir de gramáticas regulares. Contudo, para que estes novos módulos pudessem ser implementados, foi necessário primeiramente fazer um estudo acerca de autômatos finitos, gramáticas regulares, expressões regulares, bem como sobre os algoritmos presentes na literatura que realizam a transformação desejada entre os formalismos. Este trabalho traz também a descrição dos módulos adicionados, detalhes sobre a implementação e demonstra suas funcionalidades. De uma forma geral, os novos módulos funcionam perfeitamente, atendendo as expectativas e mantendo as características iniciais do AFLAB, com textos explicativos e interface simples e fácil de utilizar.*

1 Introdução

Linguagem formal é um conjunto de palavras sobre o alfabeto de uma linguagem (MENEZES, 2005). As linguagens formais têm utilidade na área da matemática e em outras que a utilizam: engenharia, física, química e computação.

Uma linguagem pode ser reconhecida por um autômato finito, que é uma máquina de estados finitos com dois resultados possíveis: aceitação ou rejeição da entrada. Uma linguagem aceita por um autômato finito é chamada Linguagem Regular.

Expressão regular é um formalismo denotacional que define a linguagem regular por meio de uma expressão, a partir de conjuntos básicos e operações de concatenação, união e repetição. A gramática regular, por sua vez, utiliza regras para defini-la, sendo que por meio dessas regras se consegue gerar todas e apenas as palavras da linguagem.

A partir de uma expressão regular é possível construir o autômato reconhecedor da linguagem que ela denota. Da mesma forma, partindo de uma gramática regular é possível obter o autômato finito que reconhece a linguagem por ela gerada.

Na maioria das vezes, esses conceitos são abordados em disciplinas sem auxílio de ferramentas de software, limitando professor e alunos ao uso de papel e caneta e resultando no estudo de autômatos finitos e linguagens regulares não muito complexos. Observou-se então, que seria interessante ter um software que pudesse ser utilizado para manipulação desses autômatos e linguagens. Partindo desta necessidade, foi iniciado o desenvolvimento do AFLAB, que auxilia no ensino de autômatos finitos.

Considerando que a primeira versão do AFLAB não contempla o tratamento de gramáticas e expressões regulares, o objeto de estudo desta pesquisa é a implementação de dois módulos na *shell:* um para obter autômatos finitos a partir de expressões regulares e outro para gerar autômatos finitos partindo de gramáticas regulares.

2 Máquinas de Estados

Segundo Santos (2004, p. 11) “máquinas de estados são estruturas lógicas compostas por um conjunto de estados e um conjunto de regras de transição entre os estados”. Essas máquinas podem auxiliar no controle de processos, descrevendo as situações em que eles se encontram a cada momento. Tais situações são representadas pelos estados da máquina, enquanto as condições necessárias para o processo sair de uma situação e partir para a próxima, dependem das regras de transição.

O funcionamento destas máquinas depende de dados de entrada, que combinados com as regras de transição, geram dados de saída como resultados. Para iniciar o processamento, deve ser definido um estado inicial e o(s) estado(s) final(is). A combinação de um dado de entrada com o estado corrente pode gerar ou não uma transição, dependendo das regras. No término de sua execução, a máquina aceita a entrada a que foi submetida se o estado corrente for um estado final, caso contrário, ela a rejeita (SANTOS, 2004).

2.1 Autômato Finito

Autômato finito (AF) é uma máquina de estados com um número finito e predefinido de estados (MENEZES, 2005). Ele sempre terá uma condição de parada, pois qualquer palavra que ele vir a processar, será finita, não existindo então uma possibilidade de ciclo infinito (*loop*). A parada de seu processamento acontece depois da aceitação ou rejeição de uma entrada (MENEZES, 2005).

Se ao final do processamento de um AF, a linguagem for aceita por ele, então ela é chamada de linguagem regular. Por outro lado, se não houver algum AF que a reconheça, então ela não pertence a essa classe de linguagens (VIEIRA, 2006).

Uma forma de representação de um AF é o diagrama de estados ou diagrama de transição. Segundo Menezes (2005) esta representação é da seguinte forma:

a) os círculos (nodos) representam os estados do autômato;
b) as arestas entre dois nodos são as transições entre eles;
c) uma flecha vinda de fora do diagrama indica o estado inicial;

d) um círculo duplo ou de maior espessura caracteriza um estado final.

A Figura 1 mostra a representação de um AF na forma de diagrama.

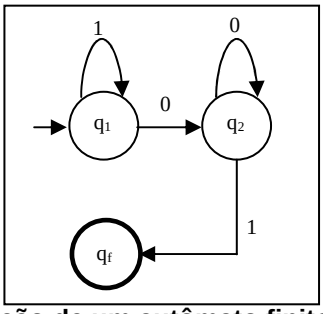


Figura 1. Representação de um autômato finito na forma de um diagrama

Uma outra forma de representar um AF é por meio de sua tabela de transições, que é uma representação tabular das transições do autômato (HOPCROFT; ULLMAN; MOTWANI, 2002).

A tabela de transições de um AF possui as seguintes características:

a) a primeira linha contém os símbolos do alfabeto de entrada do autômato;

b) a primeira coluna contém os estados;

c) o cruzamento das linhas com as colunas representam as transições;

d) o estado inicial é marcado com uma seta;

e) os estados finais são marcados com um asterisco.

Estas características podem ser melhor observadas na Tabela 1, que mostra o AF da Figura 1 representado por meio de sua tabela de transições.

Tabela 1. Representação de um autômato finito por meio de sua tabela de transições

δ	0	1
→**q_1**	q_2	q_1
q_2	q_2	q_3
***q_3**	-	-

2.1.1 Autômatos Finitos Determinísticos

Segundo Vieira (2006) um autômato finito determinístico (AFD) é definido como uma quíntupla: $M_D = (E, \Sigma, \delta, q_0, F)$, onde:

a) E: conjunto finito de estados;

b) Σ: alfabeto ou símbolos de entrada;

c) q_0: estado inicial, pertencente a E;

d) F: conjunto de estados finais, também pertencentes a E;

e) δ: função de transição ou função programa: $\delta : E \times \Sigma \rightarrow E$. Essa função demonstra que o resultado da combinação entre um estado pertencente a E e um símbolo de entrada pertencente a Σ, resulta em um estado novo, ou não, de E. Exemplo: $\delta (q_0, a) = q_1$.

Segundo Vieira (2006) se a função de transição do AF mapeia a combinação do estado corrente com um símbolo de entrada para um, e apenas um novo estado (podendo ser o atual), então esse autômato é dito determinístico. Desta forma, para aceitar uma determinada palavra de entrada, é possível apenas seguir uma única seqüência de estados partindo-se do estado inicial do autômato.

2.1.2 Autômatos Finitos Não Determinísticos

Assim como o AFD, o autômato finito não determinístico (AFND) é definido por uma quíntupla (HOPCROFT; ULLMAN; MOTWANI, 2002): $M_N = (E, \Sigma, \delta, q_0, F)$, onde:

a) E: conjunto finito de estados;

b) Σ: conjunto finito de símbolos de entrada;

c) δ: função de transição ou função programa: $\delta : E \times \Sigma \rightarrow 2^E$;

d) q_0: estado inicial, pertencente a E;

e) F: conjunto de estados finitos, também pertencentes a E.

O não determinismo de um AFND é dado pela sua função de transição, que retornará um conjunto (vazio ou não) de estados para cada combinação de estado corrente com símbolo de entrada, ao contrário do AFD, que indica apenas um próximo estado possível, para cada símbolo lido (HOPCROFT; ULLMAN; MOTWANI, 2002).

Devido ao não determinismo, diferentes seqüências de estados podem ser seguidas para que se chegue à aceitação (VIEIRA, 2006).

2.1.3 Autômato Finito Com Movimentos Vazios

Os autômatos finitos com movimentos vazios (AFNDε) possuem transições em que nenhum símbolo da fita é lido (MENEZES 2005).

O AFNDε também é definido como uma quíntupla: $M_\varepsilon = (E, \Sigma, \delta, q_0, F)$, onde todos os componentes são iguais ao de um AFND, exceto a sua função de transição, que agora passa a receber como argumento um estado de E e um elemento de $\Sigma \cup \varepsilon$, ou seja, um símbolo de entrada ou um movimento vazio, representado pelo símbolo ε (HOPCROFT; ULLMAN; MOTWANI, 2002).

Quando um movimento vazio é encontrado, é como se ele fosse invisível, pois nada acrescenta na palavra formada ao longo do processamento (HOPCROFT; ULLMAN; MOTWANI, 2002). Este tipo de transição é utilizada porque permite maior facilidade na construção e demonstração dos autômatos finitos (MENEZES 2005).

3 Gramáticas

As gramáticas são formalismos utilizados para definir linguagens. Elas são conjuntos finitos de regras que aplicadas sucessivamente, geram palavras, sendo que o conjunto de todas essas palavras define a linguagem (MENEZES, 2005; VIEIRA, 2006).

Segundo Menezes (2005) uma gramática pode ser chamada também de Gramática de Chomsky, sendo uma quádrupla ordenada: G = (V, T, P, S), onde:

a) V: conjunto de símbolos não terminais;

b) T: conjunto de símbolos terminais (alfabeto);

c) P: regras de produção;

d) S: símbolo inicial (é um símbolo não terminal).

3.1 Gramáticas Regulares

Assim como um AF é um tipo de reconhecedor de uma linguagem regular, uma gramática regular especifica esta linguagem por meio de um gerador, ou seja, ela mostra como gerar todas e apenas as palavras de uma linguagem regular (VIEIRA, 2006).

As restrições das regras de produção desse tipo de gramática permitem definir exatamente a classe das linguagens regulares. Tais restrições são representadas pelos quatro diferentes tipos de gramáticas lineares:

a) linear à direita, ex: $S \rightarrow aA$, $A \rightarrow baA$, $B \rightarrow \varepsilon$

b) linear à esquerda, ex: $S \rightarrow Sba$, $S \rightarrow a$

c) linear unitária à direita, ex: $S \rightarrow aA$, $A \rightarrow bB$, $A \rightarrow \varepsilon$, $B \rightarrow aA$

d) linear unitária à esquerda, ex: $S \rightarrow Aa$, $S \rightarrow a$, $A \rightarrow Sb$

Diz-se então que uma gramática é regular se suas regras possuem o formato de uma das gramáticas lineares, ou seja, se ela é uma gramática linear (MENEZES, 2005).

3.2 Transformação de Gramáticas Regulares em Autômatos Finitos

Uma linguagem regular é gerada por alguma gramática regular e para mostrar que uma linguagem é regular, basta que se construa seu AF reconhecedor (MENEZES, 2005). Ou seja, se G é uma gramática regular, então existe um AF M tal que a linguagem aceita por M é a mesma linguagem gerada por G (L(M) = L(G)).

Sendo assim, é possível construir um AF *M* que reconheça uma determinada linguagem regular denotada por uma gramática *G*: ACEITA (M) = GERA (G).

4 Expressões Regulares

Expressões regulares são expressões propriamente ditas que definem uma linguagem, permitindo representá-la por meio de uma descrição algébrica. Elas podem definir exatamente a mesma linguagem que os autômatos aceitam e que as gramáticas regulares geram: as linguagens regulares (CRESPO, 2001; HOPCROFT; ULLMAN; MOTWANI, 2002).

Segundo Louden (2004) expressões regulares são montadas com os caracteres de um alfabeto. Porém, há dois símbolos adicionais que devem ser considerados para situações especiais: o epsilon (ε) para denotar a cadeia vazia, que é uma cadeia composta por zero (0) caracteres, e o Ø para representar o conjunto vazio, que não contém cadeia de caracteres.

As álgebras de todos os tipos permitem construir expressões com constantes ou variáveis, alguns operadores e símbolos como parênteses para agrupamento e definição de prioridades. Assim como nessas álgebras, as expressões regulares seguem esse padrão para denotar linguagens, sendo que as três operações básicas por elas utilizadas são: união (+), concatenação e repetição (*) (também chamada fechamento) (HOPCROFT; ULLMAN; MOTWANI, 2002; LOUDEN, 2004).

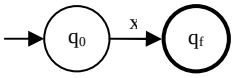
O fecho transitivo ($^+$), cujo funcionamento é semelhante a repetição sem aceitar cadeia vazia, e os parênteses, que mudam a precedência das operações, são permitidos para facilitar a definição das expressões regulares (CRESPO, 2001; LOUDEN, 2004).

4.1 Transformação de Expressões Regulares em Autômatos Finitos

Uma linguagem é dita regular somente se for possível construir um AF capaz de reconhecê-la e as expressões regulares definem a classe de linguagens regulares. Assim sendo, partindo de uma expressão regular se pode construir um AF capaz de reconhecer a linguagem denotada por ela (MENEZES, 2005): ACEITA (M) = GERA (r).

Caso uma expressão regular *r* não possua operadores, fica simples definir seu AF correspondente, como pode ser observado na Tabela 2.

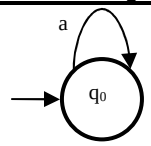
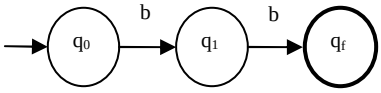
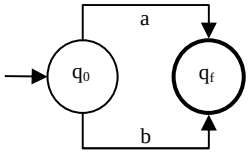
Tabela 2. Autômatos finitos correspondentes às expressões regulares com zero operadores

ER	*AF Correspondente*
r = ∅	
r = ε	
r = x	

Fonte: MENEZES, P. (2005)

A obtenção de expressões regulares com um ou mais operadores, é realizada por meio da combinação das operações das expressões regulares nos autômatos finitos, como demonstrado na Tabela 3.

Tabela 3. Autômatos finitos correspondentes às expressões regulares com um ou mais operadores

ER	*AF Correspondente*	*Operação*
$r = a^*$		**Repetição**
r = bb		**Concatenação**
r = a + b		**União**

Fonte: MENEZES, P. (2005)

5 A Ferramenta AFLAB

O AFLAB é um ambiente de criação e manipulação de autômatos finitos que iniciou com os módulos de reconhecimento de sentenças por meio de autômatos finitos determinísticos ou não determinísticos definidos na forma gráfica ou tabular. O sistema foi proposto por Gaidzinski (2007) e desenvolvido em *object pascal*, na plataforma Borland® Delphi™ Enterprise, versão 7.0.

Internamente, a ferramenta possui uma estrutura única para o armazenamento dos autômatos, permitindo dessa forma, que o AF construído na forma tabular possa ser

visualizado na forma gráfica e vice versa. Essa estrutura foi dividida basicamente em três classes (GAIDZINSKI, 2007):

a) *Testados*: armazena atributos dos estados;

b) *Talfabeto*: armazena os símbolos de entrada do AF;

c) *Tconexao*: armazena as transições do autômato.

No módulo de Forma Tabular é necessário informar os estados do AF, símbolos de entrada, estado inicial e estados finais. Depois do preenchimento dessas informações, prossegue-se informando as transições entre os estados e os respectivos símbolos de entrada. Finalmente a sentença a ser testada pode ser informada no campo correspondente, que trará como resposta a palavra “VERDADEIRA” se tal sentença for reconhecida pelo autômato informado ou “FALSA” caso contrário. O módulo de Forma Tabular pode ser observado na Figura 2.

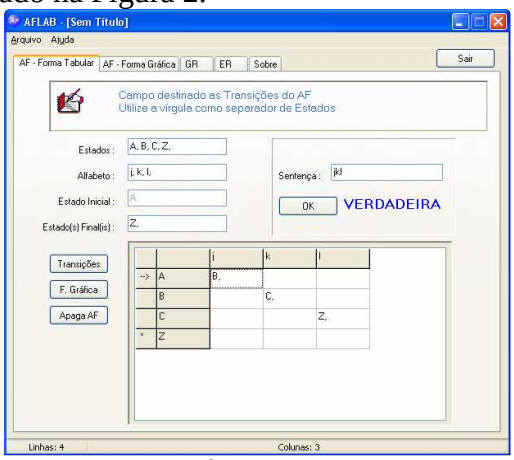


Figura 2. AFLAB – Autômato Finito na Forma Tabular
Fonte: GAIDZINSKI, M. (2007)

O módulo de Forma Gráfica do AFLAB permite que o AF seja desenhado na tela, utilizando-se as opções de adicionar estados, adicionar estados finais, remover e mover estados. Os símbolos de entrada são adicionados com as transições. Após informar o autômato, pode-se testar a sentença da mesma forma que no ambiente tabular. O módulo de Forma Gráfica pode ser observado na Figura 3.

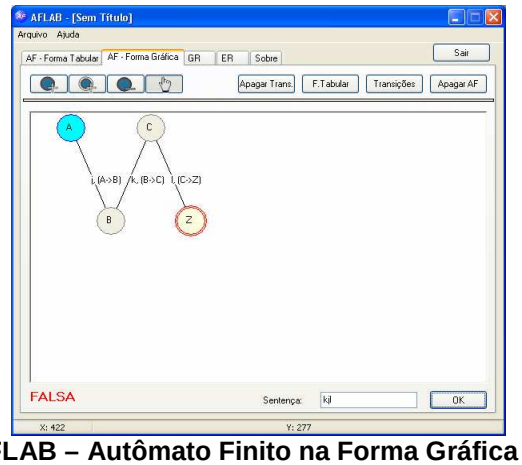


Figura 3. AFLAB – Autômato Finito na Forma Gráfica
Fonte: GAIDZINSKI, M. (2007)

Após o início do desenvolvimento da ferramenta AFLAB, percebeu-se que ela poderia ser expandida, pois existem outros assuntos relacionados a autômatos finitos do

que simplesmente o reconhecimento de sentenças. Por este motivo foi dado início ao desenvolvimento dos módulos de Gramáticas Regulares e Expressões Regulares.

5.1 Módulo de Gramáticas Regulares

Na interface do módulo foi criada a estrutura da gramática para que o usuário possa informá-la, sendo o primeiro campo para a definição dos símbolos não terminais, o segundo para os terminais e o terceiro para o símbolo inicial. As regras de produção devem ser definidas em uma String Grid, sendo uma linha para cada produção da gramática, conforme pode ser observado na Figura 4.

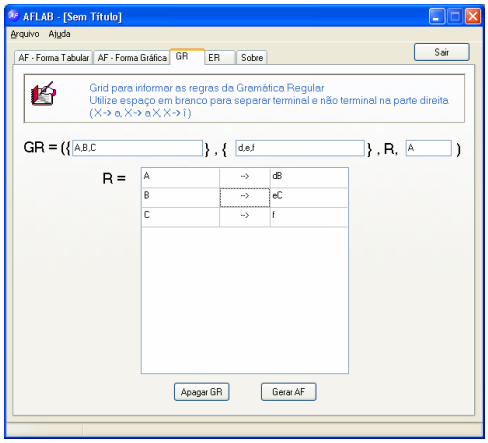


Figura 4. AFLAB – Módulo de gramáticas regulares

O usuário deve informar primeiramente os não terminais e terminais, seguidos do símbolo inicial. É obrigatório informar primeiramente estes três campos antes de seguir para o preenchimento das regras da gramática.

Após o término da definição da gramática, o usuário poderá solicitar a criação do AF. Após sua criação, o programa mostrará o módulo “AF – Forma Gráfica” para que o autômato possa ser visualizado graficamente, podendo ser selecionada também a guia “AF – Forma Tabular” possibilitando a visualização na forma tabular.

O AF gerado a partir da gramática regular da Figura 4 pode ser visualizado na sua forma tabular na Figura 5 e na sua forma gráfica na Figura 6.

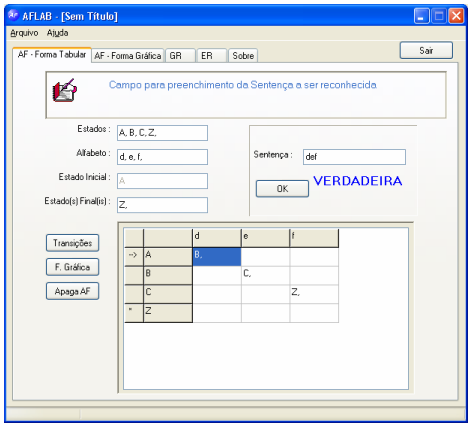


Figura 5. Forma Tabular

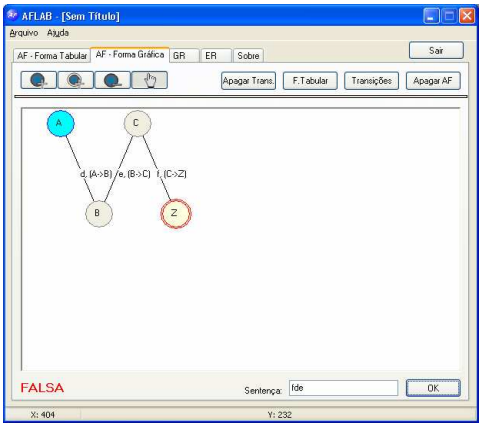


Figura 6. Forma Gráfica

5.2 Módulo de Expressões Regulares

A tela para informar a expressão regular na ferramenta AFLAB é composta apenas por um campo para que a expressão seja informada e um botão no qual o usuário solicita a

geração do AF reconhecedor da linguagem definida pela expressão. O módulo de Expressões Regulares pode ser observado na Figura 7.

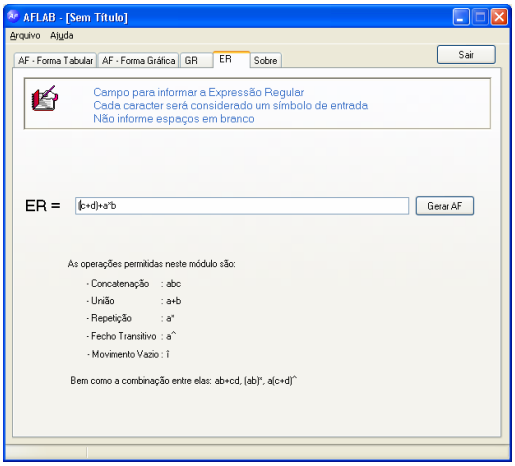


Figura 7. AFLAB – Módulo de Expressões Regulares

As operações permitidas neste módulo são:

a) concatenação: não utiliza nenhum caractere adicional, apenas a sucessiva utilização das letras e / ou números desejados. Exemplos: abc, 123, ab12;

b) união: é caracterizada pela combinação dos caracteres da expressão regular com o símbolo + , lido como ‘ou’ e significa que o AF percorrerá um caminho ou outro. Exemplos: a+b, ab+cd, 12+ab;

c) repetição: caracteriza-se pela presença do símbolo * e significa a ausência ou repetição de uma determinada parte da expressão regular. Exemplos: ab*, a(ab)*, 0(1+2)*;

d) fecho transitivo: nesta ferramenta será representado pelo símbolo ^ devido a impossibilidade de representar o $^{+}$. Sua função é semelhante a repetição, porém garante a ocorrência da parte estipulada pelo menos uma vez. Exemplos: ab^, a(ab)^, 0(1+2)^.

Após informar a expressão regular, o usuário pode solicitar sua transformação em AF e assim como no módulo de Gramáticas Regulares, após a criação, o foco irá para o módulo “AF – Forma Gráfica”, para que o usuário possa conferir o resultado graficamente, podendo ele alternar também pém para o módulo “AF – Forma Tabular”.

O resultado da transformação da expressão demonstrada na Figura 7 pode ser visto na Figura 8, que mostra o autômato gerado em sua forma tabular e na Figura 9, que mostra o mesmo autômato em sua forma gráfica.

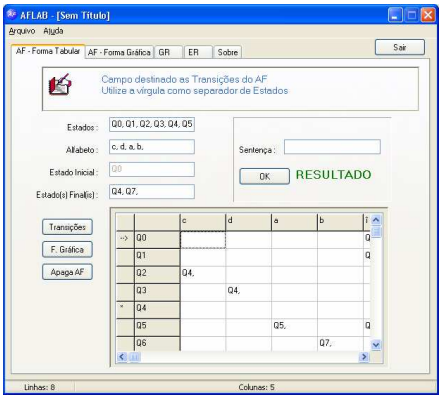


Figura 8. Forma Tabular

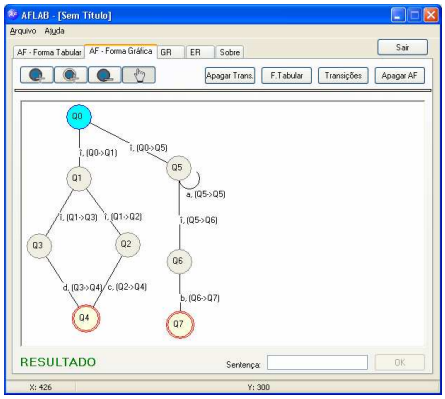


Figura 9. Forma Gráfica

Os movimentos vazios (î) encontrados nos autômatos apresentados visam facilitar a construção e proporcionar uma visualização mais simplificada dos mesmos, porém estes autômatos não são capazes de reconhecer sentenças. Isso acontece devido ao fato de que a primeira versão do AFLAB não contemplava movimentos vazios e eles não foram tratados na *procedure* que realiza o reconhecimento de sentenças.

6 Conclusão

Ainda é notável nos dias atuais a falta de ferramentas em determinadas áreas de conhecimento. Por esta razão, vários softwares para fins educacionais têm surgido em instituições para tentar suprir esta necessidade e aos poucos ir mudando este cenário.

O AFLAB partiu do princípio de se criar um software que suprisse a necessidade da manipulação de autômatos finitos e aplicação prática dos conceitos relacionados a eles. Após a conclusão de sua primeira versão, que abrangeu autômatos finitos na forma gráfica e tabular e o reconhecimento de sentenças por meio destes autômatos, outras necessidades foram levantadas e novas funcionalidades passaram a ser implementadas.

Os novos módulos desenvolvidos no AFLAB que transformam uma gramática regular ou expressão regular em um AF reconhecedor da linguagem gerada por estes formalismos enriqueceram ainda mais a ferramenta, contribuindo para que ela ficasse mais completa. Além disso, há também uma contribuição no que diz respeito aos softwares existentes nesta área específica, pois não são muitos os que realizam estas transformações de forma satisfatória.

Após o término de todo o estudo acerca do tema e da implementação dos novos módulos, testes foram realizados e pôde-se perceber então que os objetivos deste trabalho foram alcançados. A partir de agora o AFLAB conta com dois novos módulos que funcionam perfeitamente e devem atender as expectativas dos usuários.

7 Referências

CRESPO, Rui Gustavo. **Processadores de linguagens**: da concepção à implementação. Lisboa: IST Press, 2001.

GAIDZINSKI, Marco Aurélio. **Ambiente de criação e manipulação de autômatos finitos na forma gráfica ou tabular para o reconhecimento de sentenças**. 2007. 61 f. Trabalho de Conclusão de Curso (Bacharelado) – Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2007.

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

LOUDEN, Kenneth C. **Compiladores:** princípios e práticas. São Paulo: Pioneira Thomson Learning, 2004. Tradução de Flávio Soares Correia da Silva.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos**. 5. ed. Porto Alegre: Sagra Luzzatto, 2005.

SANTOS, Gilliard Lopes dos. Máquinas de Estados Hierárquicas em Jogos Eletrônicos. In: _____. **Máquinas de Estados em Jogos Eletrônicos**. Rio de Janeiro, 2004. Cap 2. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/4711_3.PDF?NrOcoSis=11468&CdLinPrg=pt>. Acesso em: 30 ago. 2007.

VIEIRA, Newton José. **Introdução aos Fundamentos da Computação**: Linguagens e Máquinas. São Paulo: Pioneira Thomson Learning, 2006.