UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
FACULDADE DE CIÊNCIAS DA COMPUTAÇÃO

SIMMoL – SIMULADOR ITERATIVO DE MODELOS PARA LINGUAGENS

Clemilton Mackson Fagundes Baracho

Natal /RN
2007

Clemilton Mackson Fagundes Baracho

SIMMoL- SIMULADOR INTERATIVO DE MODELOS PARA LINGUAGENS

Relatório de Graduação apresentado ao curso de Ciências da Computação na Universidade Federal do Rio Grande do Norte, como exigência para obtenção do título de bacharel em Ciências da Computação com habilitação em Sistema de Informação sob a orientação do Professor Doutor Benjamin René Callejas Bedregal

Natal
2007

Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Ciências da Computação

Aprovado em:
__/__/07

Banca Examinadora

_____
Prof. Dr. Benjamin René Callejas Bedregal
Orientador

_____
Examinador 1

_____
Examinador 2

_____
Suplente

Natal
2007

"*Não me desencorajo, porque cada tentativa errada descartada é outro passo à frente.*"
(Thomas Edison)

AGRADECIMENTOS

Agradeço à Deus, pois sem a fé que tenho nele nada seria possível.
Agradeço a minha mãe, que é a razão da minha existência.

SUMÁRIO

ILUSTRAÇÕES

RESUMO

As linguagens formais têm um papel fundamental na ciência da computação, e por isso mereceram e sempre merecerão atenção especial dos estudantes, professores e cientistas da área da computação. O alto poder de processamento e a vasta memória dos atuais computadores dão o suporte necessário para manipular os diversos modelos e conceitos ligados às linguagens formais e este fato traz consigo a possibilidade de criar softwares que permitam o processamento de algoritmos usados para representar esses modelos, que, unido à necessidade dos estudantes de usarem ferramentas de caráter educativo para auxiliá-los no aprendizado de disciplinas semelhantes à Teoria da Computação tornaram eminente a construção de softwares com esta finalidade. Muitos acadêmicos de vários níveis já produziram ferramentas com intuito de auxiliar no aprendizado de conceitos ligados à teoria da Computação, entretanto, algumas não englobam interface gráfica com grande iteração com usuário e outras não tratam todos os tipos de modelos desta área. Fatos como estes sempre motivam a produção de softwares de caráter educacional, aplicados ao estudo das disciplinas semelhantes à Teoria da Computação.

INTRODUÇÃO

A ciência da computação está fixada sobre bases sólidas, que é o motivo para o avanço e progresso de outras áreas, como engenharias, ciências médicas, e todos os ramos relacionados com informações e cálculos. O estudo sobre as linguagens formais, autômatos determinísticos e não determinísticos é com certeza co-responsável pela estrutura dessas bases. As linguagens formais possuem uma formalidade imutável, o que não acontece com linguagens de cunho falado, como português e inglês, uma vez que estas adquirem novas palavras e palavras já existentes adquirem significados novos ou duplo-significado.

As linguagens formais têm um papel fundamental na computação, uma vez que linguagens de programação são linguagens formais.

As linguagens formais são constituídas de um alfabeto e uma gramática. O alfabeto forma cadeias de símbolos aceitas pela linguagem. Gramática é um conjunto de regras de aceitação para essas cadeias. Uma gramática possui um conjunto finito de objetos chamados variáveis, um conjunto de objetos chamados símbolos e outro chamado “símbolo terminais”, um símbolo especial chamado de variável de inicio e um conjunto finito de produções. As regras de produção definem como construir as cadeias que constituem a linguagem associada à gramática.

Existem mecanismos que testam estas cadeias, dos quais podemos citar autômatos e expressões regulares.

Autômato é um modelo de um computador digital. Um autômato possui um mecanismo para ler entradas, e apartir dessas entradas gerar uma saída

que será interpretada. Desta forma autômatos podem ser usados como testadores ou reconhecedores de linguagens formais.

Expressão regular é um modelo matemático também usado para reconhecer linguagens formais de uma certa classe, a classe das linguagens regulares.

Existem varias formas de representar a mesma linguagem Regular (autômatos finitos, expressões regulares, gramáticas regulares) e como tais modelos podem representar a mesma linguagem regular, então eles podem ser convertidos um no outro através de algoritmos de conversão já existentes, por exemplo, autômatos não determinísticos podem ser convertidos em autômatos determinísticos. Muitos desses algoritmos são relativamente difíceis de usar, o que muitas vezes resultam em erros de conversão de uma forma para outra. Felizmente há muito já é possível construir programas aplicativos que executem estes algoritmos de forma a auxiliar o entendimento destes.

Por exemplo, podemos citar o SAGEMoLiC[1], que é um moderno software com interface gráfica e de iteração com o usuário. Contudo alguns softwares não englobam todas as formas, ou algoritmos de conversão, por isso é sempre motivante a criação de novos aplicativos, sobretudo de caráter educacional e científico, que de alguma maneira possa ajudar o entendimento destes algoritmos e formas.

O desenvolvimento e apresentação de uma ferramenta chamada “Sistema Interativo de Manipulação de Modelos para Linguagens” (SIMMoL),

1Disponível em http://www.mat.unb.br/~ayala/tcgroup/SAGEMoLiC. SAGEMoLiC é um acrônimo para Sistema de Animação Gráfica de Teoremas de Equivalência entre Modelos e Linguagens Computação

que permita a construção e visualização de representações de linguagens regulares, como autômatos, gramáticas, expressões regulares, bem como o uso de algoritmos de conversões entre estas representações é o intuito deste trabalho.

SIMMoL é um applet (software que pode ser manipulado através de um navegador, como Internet Explorer, Mozilla ou Ópera) para Internet, de acesso gratuito desenvolvido em Java.

O SIMMoL está na primeira versão e nesta fase ele se propõe a representar e manipular os diversos modelos de representação para linguagens regulares. Assim esta versão do SIMMoL não tratará de outras classes de linguagens além das linguagens regurales. Linguagem regular é o tipo de linguagem mais simples de linguagens formais e é possível descrevê-las através de expressões regulares, reconhecedores finitos determinísticos e gramáticas regulares. O SIMMoL provê mecanismos para criação e manipulação de modelos para esse tipo de linguagem, como expressões regulares, autômatos determinísticos e gramáticas regulares. A proposta do SIMMoL não é tornar obsoleto os aplicativos semelhantes já existentes, e sim, acrescentar mais opções ao conjunto de softwares dedicados a área acadêmica e sobretudo auxiliar estudantes a entender e manipular os diversos conceitos e formalismos abordados nas disciplinas que abordam o assunto de Teoria da Computação, uma vez que este conjunto ainda é bastante reduzido se comparado ao conjunto de softwares existente para a área de redes de computadores por exemplo.

1.BACKGROUND

As linguagens regulares podem ser representadas por diferentes modelos matemáticos (autômatos, expressões regulares e gramáticas regulares), os quais são equivalentes, pois podem ser convertidos um nos outros modelando a mesma linguagem. Existe, por "trás" destes modelos uma série de conceitos, propriedades e regras que serão discutidos de forma breve a seguir.

1.1. CONCEITOS MATEMÁTICOS BÁSICOS

1.1.1. CONJUNTOS

A teoria dos conjuntos tem um papel relevante para a teoria das linguagens formais, pois linguagens formais são conjuntos, portanto, veremos algumas operações sobre conjuntos e suas propriedades algébricas. Sejam S1 e S2 dois conjuntos quaisquer.

- União: $S_1 \cup S_2 = \{x/x \in S_1 \text{ ou } x \in S_2\}$
- Intersecção: $S_1 \cap S_2 = \{x/x \in S_1 \text{ e } x \in S_2\}$
- Diferença: $S_1 - S_2 = \{x/x \in S_1 \text{ e } x \notin S_2\}$
- Complementação: $\overline{S} = \{x/x \notin S\}$

Propriedades:

- $S \cup \varnothing = S - \varnothing = S$,
- $S \cap \varnothing = \varnothing$,
- $\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2}$,

- $\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2}$

Um conjunto S_1 diz-se um subconjunto de S, denotado por $S_1 \subseteq S$, se todo elemento de S_1 também é um elemento de S.

Se S_1 e S_2 não tem elemento em comum, isto é, $S_1 \cap S_2 = \varnothing$, então os conjuntos são disjuntos.

Um conjunto diz-se finito se contém um número finito de elementos, caso contrário ele se diz infinito. O tamanho de um conjunto finito é o numero de elementos que ele contem, denotado por |S|.

1.1.2. FUNÇÕES E RELAÇÕES

Uma função é uma regra que associa a elementos de um conjunto um único elemento de outro conjunto. Se f denota uma função, então o primeiro conjunto é chamado domínio de f e o segundo conjunto de contradomínio de f.

Uma função pode ser representada por um conjunto de pares

$$\{(x_1,y_1),(x_2,y_2),\ldots\}$$

onde x_i é um elemento no domínio da função e y_i é o valor correspondente no contradomínio. Definido-se função como um conjunto de pares ordenados, cada x_i pode ocorrer no máximo uma vez como primeiro elemento do par. Se isto não for exigido tal conjunto é chamado uma relação.

Um tipo de relação especialmente importante são relações de equivalência, que generalizam o conceito de igualdade. Para indicar que um par (x,y) é equivalente , escrevemos $x \equiv y$.

Uma relação binária sobre um conjunto, denotada por ≡, é considerada

uma equivalência se ela satisfaz 3 regras:

Reflexividade: $x \equiv x$ para todo x,

Simetria: se $x \equiv y$ então $y \equiv x$ e

Transitividade: se $x \equiv y$ e $y \equiv z$ então $z \equiv x$.

1.1.3. GRAFOS

Um grafo é um construto construído de dois conjuntos finitos, um conjunto $V = \{v_1, v_2, \ldots, v_n\}$ de vértices e o conjunto $E = \{c_1, c_2, \ldots, c_n\}$ de arestas. Cada aresta é um par de vértices de V, por exemplo, $e_1 = \{v_j, v_k\}$;

Uma maneira conveniente de se visualizar grafos é através de diagramas nos quais os vértices são representados como círculos e as arestas como linhas com setas conectando os vértices. O grafo com vértices $\{v_1, v_2\}$ e aresta $\{(v_1, v_2)\}$ é desenhado na figura 1.1.

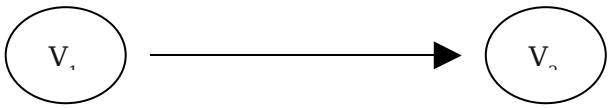

figura 1.1- diagrama de um grafo

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

caminho. Este rótulo é a seqüência de rótulos das arestas encontradas quando se percorre a trilha.

1.2. LINGUAGEM FORMAL

Num sentido amplo, uma linguagem é um subconjunto de cadeias formadas de forma criteriosa apartir de um alfabeto pertencente a esta linguagem. As regras de formação para essas cadeias são conhecidas como gramática, assim como a língua portuguesa é um conjunto de palavras e sentenças, com a ressalva que na teoria das linguagens formais não há distinção entre palavras e sentenças, sendo representadas apenas por cadeias.

Entende-se por Teoria das Linguagens Formais e dos Autômatos, o estudo de modelos matemáticos que possibilitam a especificação e reconhecimento de linguagens, suas classificações, estruturas, propriedades, características e inter-relacionamentos.

Segundo wikipedia <http: //www.wikipedia.org/wiki/linguagemfomal> pode-se dizer que linguagens formais são mecanismos formais para representação e especificação de linguagens, baseados na Teoria da Computação. As representações podem ser feitas por reconhecedores e geradores. Os reconhecedores são dispositivos formais que servem para verificar se uma sentença pertence ou não à determinada linguagem. São os autômatos: autômatos finitos, autômatos de pilha e máquinas de Turing, entre outros. Os sistemas geradores são dispositivos formais que permitem a geração sistemática de todas as sentenças de uma linguagem. Os principais sistemas geradores disponíveis são as gramáticas, onde se destacam as gramáticas

sensíveis ao contexto. Então, linguagens formais podem ser representadas de maneira finita e precisa através de sistemas com sustentação matemática.

Como linguagens são conjuntos, então a união, intersecção e diferença de duas linguagens são imediatamente definidas. O complemento de linguagem é com respeito a Σ^* (conjunto de todas as cadeias com símbolos do alfabeto incluindo a cadeia vazia), isto é, o complemento de L é:

$$\overline{L} = \Sigma^* - L.$$

A concatenação de duas linguagens L_1 e L_2, denotado por L_1L_2, é o conjunto de todas as cadeias obtidas em concatenando-se qualquer elemento de L1 com qualquer elemento de L2.

1.3. GRAMÁTICAS

Gramáticas são mecanismos usados para construir linguagens. Uma gramática nos diz se uma sentença em particular é bem formada ou não, então gramática é um conjunto de regras que rege a construção de sentenças. Uma gramática é definida como a quadrupla G={V,T,S,P} onde:

- V é um conjunto finito de objetos chamados de variáveis,
- T é um conjunto finito de objetos chamados de símbolos terminais,
- S é um símbolo especial que pertence ao conjunto de variáveis chamado de variável de inicio e
- P é um conjunto finito de produções. As produções definem como uma cadeia é transfomada em outra, e dessa forma elas definem uma linguagem associada à gramática.

Por uma gramática, linguagens podem ser classificadas em diversos tipos como linguagens regulares, linguagens livres de contextos e linguagens irrestritas entre outras, lembrando que apenas as linguagens regulares são exploradas nesse trabalho.

O principal na gramática são as regras de produção. Elas especificam como a gramática transforma uma cadeia em outra, e através disso, elas definem uma linguagem associada com a gramática.

Em Teoria da Computação as Gramáticas regulares, também conhecidas como Tipo 3 da hierarquia de Chomsky, é uma restrição sobre a forma das produções. Elas são de grande importância no estudo dos compiladores por possuírem propriedades adequadas para a obtenção de reconhecedores simples.

1.4. AUTÔMATOS FINITOS

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

1.4.1. AUTÔMATOS FINITOS DETERMINÍSTICOS

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

- Q é um conjunto finito de estados internos,
- Σ é um conjunto de símbolos chamado alfabeto de entrada,
- δ é uma função chamada de função de transição,
- q_0 é o estado inicial, $q_0 \in Q$ e,
- F é o conjunto de estados finais. Para visualizar e representar autômatos usa-se grafos de transições nos quais os vértices representam estados e as arestas representam transições.

Os rótulos dos vértices são os nomes dos estados enquanto os rótulos das arestas são os valores correntes dos símbolos de entrada. A figura 1.2 mostra um autômato finito determinístico. Os autômatos determinísticos possuem papel relevante nesse trabalho, uma vez que uma linguagem é regular se e somente se existe um autômato finito determinístico para reconhecer todas as suas cadeias conforme [Acióly, Bedregal & Lyra, 2002], portanto uma linguagem é o conjunto de todas as cadeias reconhecidas por um autômato finito determinístico. Podemos então dizer apartir desse momento que um autômato finito determinístico (afd) é um reconhecedor finito determinístico. Desta forma, se $M = \{Q,\Sigma, \delta,q_0,F\}$ é um reconhecedor finito determinístico, então a ele está associado um grafo de transição G tendo exatamente |Q| vértices, cada um rotulado com um $q_i \subset Q$ diferente. Para cada regra de transição $\delta(q_i,a)=q_j$, o grafo tem uma aresta (q_i,q_j) rotulada por a.

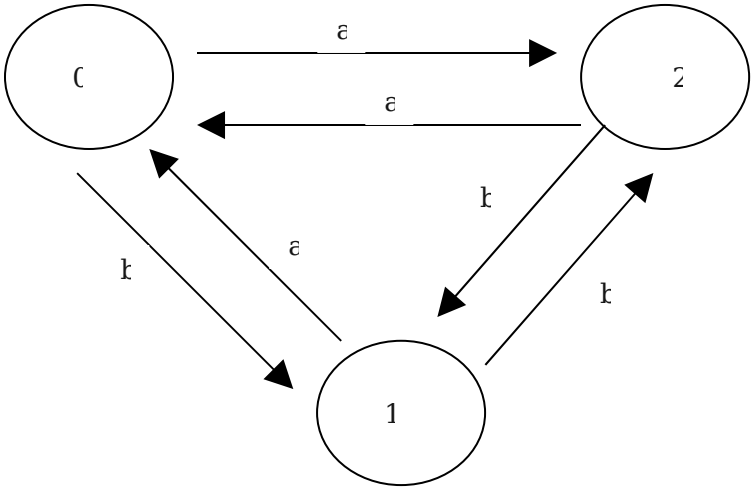

Figura 1.2. Exemplo de autômato determinístico.

Existe também uma outra família de autômatos que merecem atenção especial nesse trabalho, são os autômatos finitos não determinísticos que têm por característica principal a possibilidade de se atingir um conjunto de estados apartir de um determinado estado consumindo um determinado símbolo de entrada, ou sem consumir nenhum símbolo, daí a expressão “não determinístico”, pois não se pode prevê o próximo estado a ser alcançado apartir de um símbolo em um determinado momento. Autômatos finitos determinísticos não possuem esta característica, pois, para apartir de cada estado, em si consumindo algum símbolo de entrada atingir-se-á algum estado do autômato.

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

1. $\delta^*(q,\lambda) = q$

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

A linguagem aceita por um afd $M = \{Q,\Sigma, \delta,q_0,F\}$ é o conjunto de todas as cadeias sobre Σ aceita por M conforme disponível em [Acióly, Bedregal & Lyra, 2002], de maneira formal temos:

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

Teorema 2.1- Seja $M = \{Q,\Sigma, \delta,q_0,F\}$ um afd, e seja G seu grafo associado. Então para todo $q_i,q_j \in Q$ e $w \in \Sigma^+$, $\delta^*(q_i,w)=q_j$ se e somente se existe em G um caminho com rotulo w de q_i a q_j.

Demonstração: Ver [Acióly, Bedregal & Lyra, 2002].

1.4.2. AUTÔMATOS FINITOS NÃO DETERMINÍSTICOS

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

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

Existem duas diferenças principais entre essa definição e a de autômato finito determinístico. Aqui a imagem de δ é o conjunto das partes de Q. Então seu valor não é um único elemento de Q, mas um subconjunto dele. Este

subconjunto define o conjunto de todos os estados possíveis que podem ser alcançados pela transição.

Analogamente a autômatos finitos determinísticos, os autômatos finitos não determinísticos podem ser representados por grafos de transição. Também da mesma forma que rfd´s, autômatos finitos não determinísticos também reconhecem linguagens, porém as linguagens regulares somente podem ser reconhecidas por afd´s, daí a importância de se estabelecer uma equivalência entre afd´s e afn´s. Esta equivalência é verdadeira e segundo [Acióly, Bedregal & Lyra, 2002], dois reconhecedores são equivalentes se eles reconhecem a mesma linguagem. O procedimento de transformação de equivalência é mostrado em [Acióly, Bedregal & Lyra, 2002].

1.5. EXPRESSÕES REGULARES

Expressão regular é uma notação usada também para representar linguagens regulares. Essa notação envolve uma combinação de cadeias de símbolos de algum alfabeto, parênteses, e operadores de união, concatenação e fecho de cadeias, para os quais usamos os símbolos “+”, “.” e “*” respectivamente. O teorema de Kleene expressa que se R é uma expressão regular sobre um alfabeto qualquer e L é uma linguagem regular, então existe um autômato finito que reconhece L.

Devemos observar cada operação efetuada sobre os símbolos do alfabeto, que significa agir sobre partes da expressão, conforme os casos seguintes:

-Caso 1. Suponha que R = (rs). Sendo L_r e L_s linguagens correspondentes

a r e s, L correspondente a R. Então suponha que os autômatos $M_r = (Q_r, \Sigma_r, q_r, \delta_r, A_r)$ e $M_s = \{Q_s, \Sigma_s, q_s, \delta_s, A_s\}$ reconhecem L_r e L_s, respectivamente. Desejamos construir $M = \{Q, \Sigma, q_0, \delta, A\}$ para reconhecer $L_r L_s$ uma vez que $L = L_r L_s$. O processo para tal é simples, aceitamos o estado inicial de M como sendo q_r (estado inicial de r), aceitamos os elementos de A_s (estados de s), e criamos transições lambda de cada elemento de A_r (estados finais de r) para q_s (estado inicial de s). Assim, cadeias pertencentes à linguagem $L_r L_s$ iniciam em q_r, seguem caminho até os estados finais de r e através de transições lambda saltam para q_s e seguem até os estados finais de s que serão os estados finais de M e os estados finais de M_r e M_s não mais serão estados finais.

-Caso 2. Suponha que R = (r+s). Como antes, as linguagens correspondentes a L_r e L_s, são reconhecidas por autômatos $M_r = (Q_r, \Sigma_r, q_r, \delta_r, A_r)$ e e $M_s = \{Q_s, \Sigma_s, q_s, \delta_s, A_s\}$. $M = \{Q, \Sigma, q_0, \delta, A\}$ é construído criando-se um novo estado inicial q0 que não está presente em Q_r ou Q_s, Q será a união entre Q_r, Q_s e q_0, A será a união entre A_r e A_s, cria-se transições lambda entre q_0 e q_r e q_0 e q_s e as demais transições são as mesmas para cada um dos autômatos r e s.

-Caso 3. Suponha agora R = (r*). Temos $M_r = (Q_r, \Sigma_r, q_r, \delta_r, A_r)$ reconhecendo L_r. Constrói-se $M = \{Q, \Sigma, q_0, \delta, A\}$ apartir de um novo estado que será seu estado inicial e final criando-se transições lambda entre ele e o estado inicial de r que seguirá com suas transições normalmente, após isso se constroem transições lambda de cada estado final de r para q_0.

2.SOFTWARES EXISTENTES

Muitos aplicativos com a finalidade de auxiliar estudantes no processo de aprendizado na disciplina de Teoria da Computação já foram produzidos. Em geral estes softwares são construídos por grupos de trabalhos. É conveniente usar uma linguagem de programação de alta portabilidade para produzir estes softwares, por isso a linguagem de programação Java é escolhida na maioria dos casos recentes, não somente devido a sua portabilidade, mas também pela possibilidade de uso destes sistemas através de uma rede de computadores, em contrapartida este último caso requer um software instalado na máquina cliente. Existem softwares dos mais diversos níveis de interação com o usuário, e de abrangência no assunto e de modo resumido analisaremos os seguintes:

2.1. LANGUAGE EMULATOR

O Language Emulator [2] é um aplicativo em Java para desktop, isto é, não é visualizado através de um navegador e, portanto não pode ser operador através de uma rede. Quanto a linguagens regulares ele provê operações sobre autômatos finitos determinísticos, autômatos finitos não determinísticos, gramáticas lineares à direita e expressões regulares. A figura 2.1 mostra a tela inicial do Language Emulator. Ele é um aplicativo de boa interação com o usuário que permite que sejam criados modelos representativos de uma linguagem regular através de entradas de dados escritas em campos de texto. Toda a interface (de entrada e de saída) é feita através de campos de texto, não sendo possível, por exemplo, a visualização de autômatos como grafos de

2 Disponível em http://homepages.dcc.ufmg.br/~lfvieira/languageemulator.jar

transição.

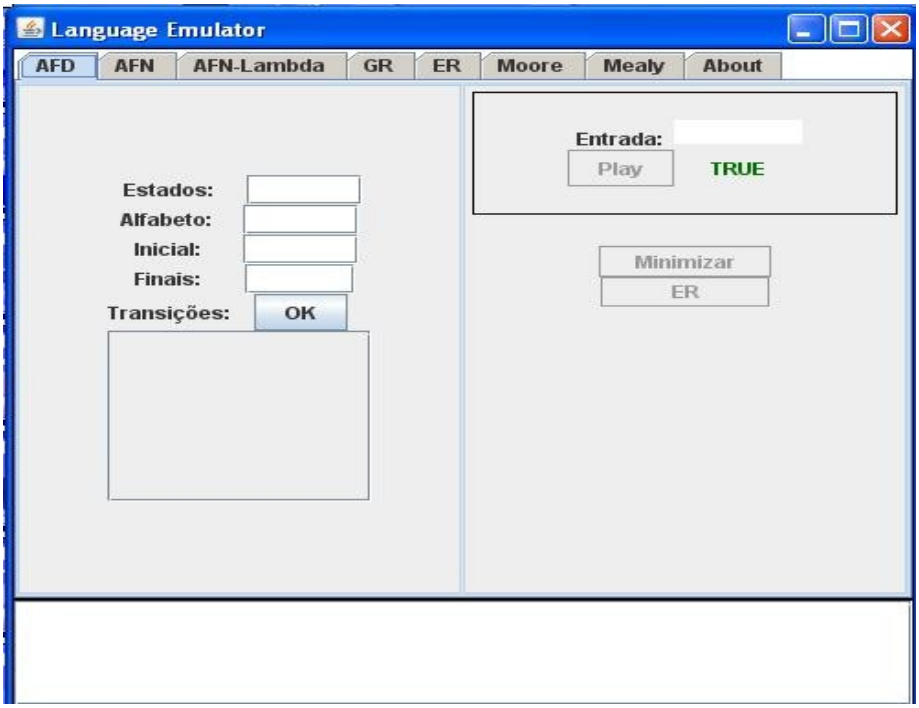

figura 2.1. Tela inicial do Language Emulator.

No Language Emulator, para se construir um autômato, determinístico ou não determinístico, deve-se primeiramente estabelecer os estados, o alfabeto, o estado inicial e o estado final, todos em campos diferentes, e só depois criar as transições. Ele provê também um campo de texto para determinar se uma determinada cadeia de entrada pertence à linguagem reconhecida pelo autômato gerado. A figura 2.3 mostra a criação do autômato finito determinístico mostrado na figura 2.2, (que por sua vez foi gerado no SIMMoL) e a verificação da cadeia aab para a linguagem gerada por este afd. E apartir da construção desse autômato pode-se minimizar o número de estados e gerar uma expressão linear à direita correspondente, porém em meus testes tais operações por motivo desconhecido não foram concretizadas.

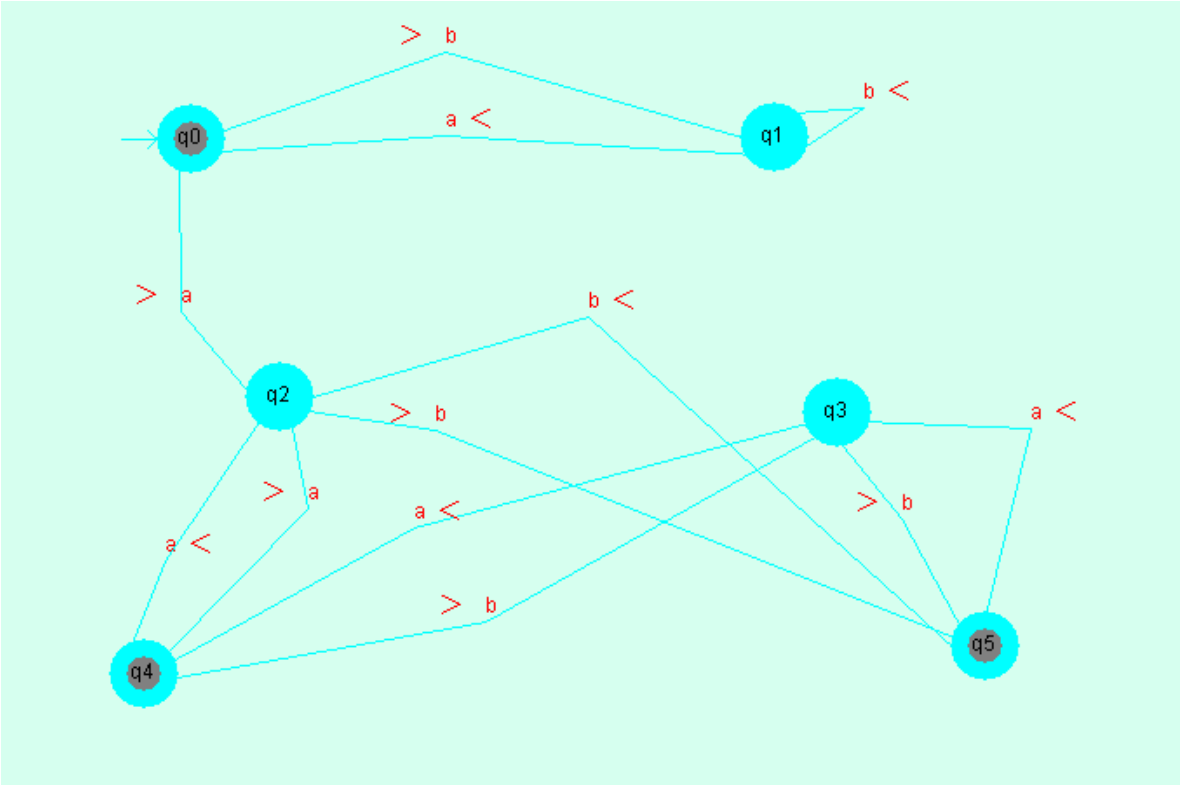

figura 2.2. Autômato gerado no SIMMoL.

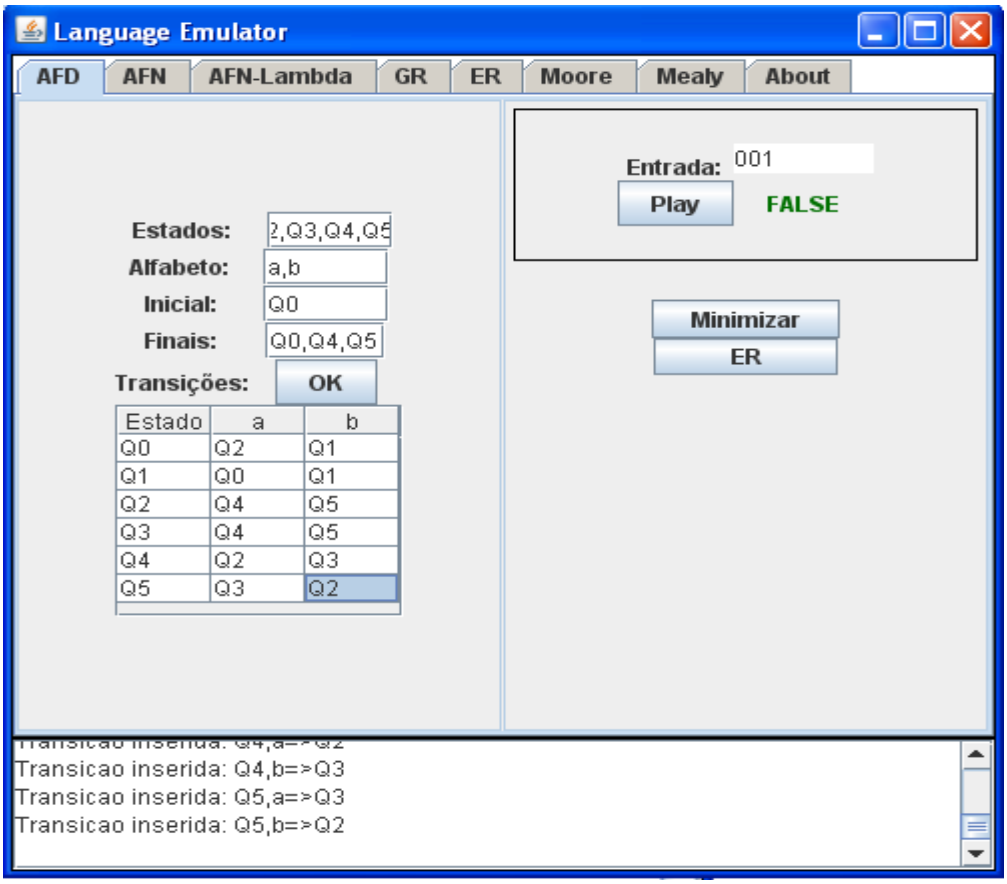

figura 2.3.Autômato igual ao autômato da figura 2.2 gerado no Language Emulator.

O Language Emulator possui apenas uma média interatividade e “peca” no fato de não permitir a construção ou exibição de autômatos na forma de grafos.

Outras operações também podem ser realizadas, vejamos algumas:

1. Transformação de gramáticas regulares em afn;
2. Transformação de um afn em um afd;
3. Conversão de expressão regular em afn;
4. Conversão de afd em expressão regular;
5. Minimização de um afd.

2.2. SAGEMoLiC

O SAGEMoLiC é um applet Java para Internet e foi projetado pelo grupo de teoria da computação da Universidade de Brasília com o intuito de simular os diversos modelos de representação para linguagens regulares e linguagens livres de contexto. O SAGEMoLiC é um software bastante interativo e bem completo no que se propõe, porém possui dificuldades ao ilustrar autômatos na tela, uma vez que o mostra de forma aleatória provocando uma desorganização. A figura 2.4 mostra a tela do SAGEMoLiC. Esta ferramenta possui poder para operar sobre autômato (determinístico, não determinísticos e autômatos a pilha), expressões regulares, gramáticas lineares e gramáticas livres de contexto, sendo também de fácil uso. Uma vez que se define o alfabeto pode-se criar autômatos clicando em estados, selecionando transições e símbolos tudo em um painel de construção.

As operações que podem ser realizadas sobre um autômato são:

1. Transformar um autômato com lambda-transições em um autômato sem lambda-transições

2. Transformar um autômato finito não determinístico e sem lambda-transições em um autômato finito determinístico

3. Minimizar um autômato finito determinístico

4. Transformar um autômato em uma gramática linear à direita

5. Transformar um autômato em uma expressão regular

6. Criar um autômato a partir de uma gramática linear à direita

7. Criar um autômato a partir de uma expressão regular

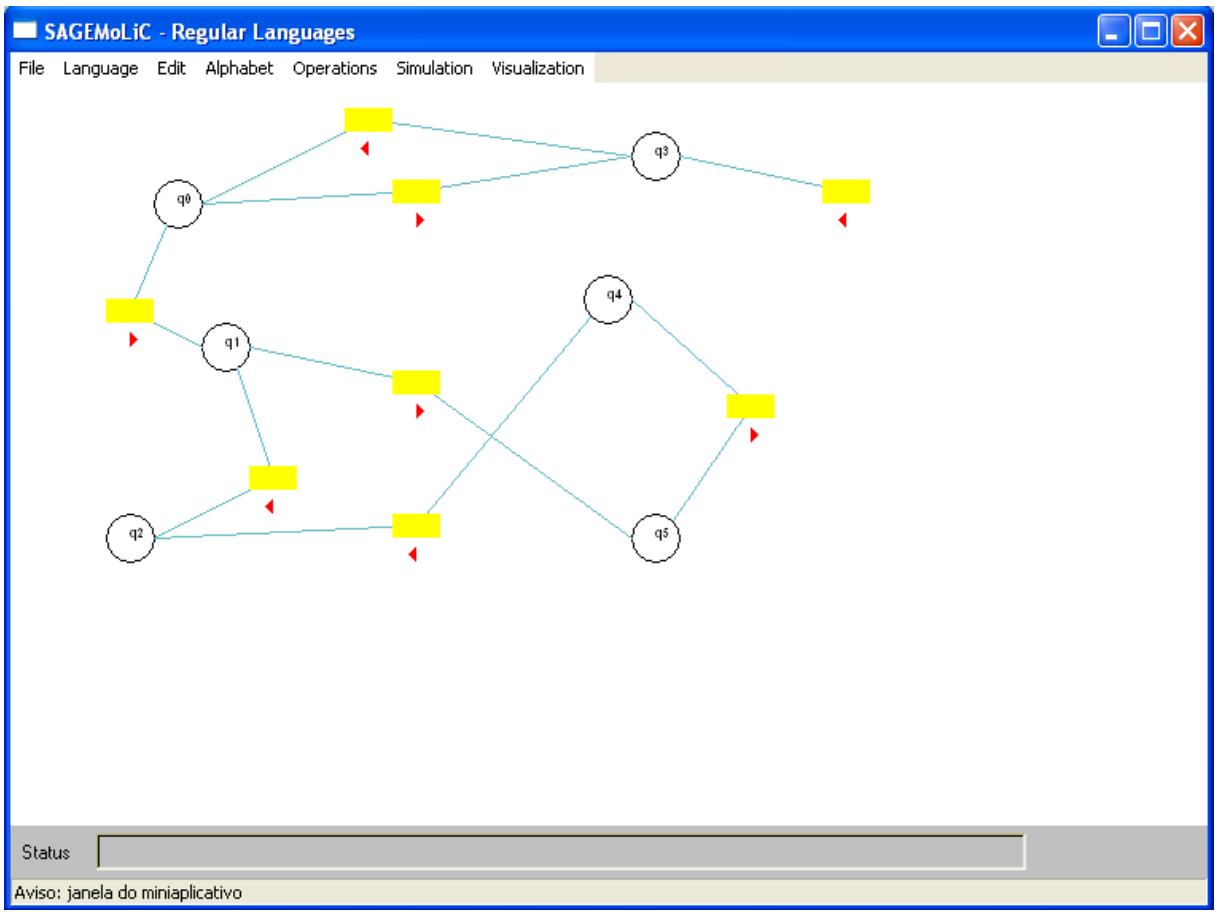

figura 2.4-tela do SAGEMOLIC

2.3 JFLAP

O JFLAP[3] é um simulador para uma gama de modelos representativos para linguagens incluindo autômatos finitos, autômatos a pilha, gramáticas, expressões regulares, máquinas de Turing, lema da bomba entre outros modelos, ele foi desenvolvido para ser usado em rede usando Java, e é de fácil uso, a produção de autômatos é simples de ser feita bastando clicar nos estados e transições e definir os símbolos sem se preocupar com o alfabeto, pois este será montado em tempo de construção do autômato. Uma vez formado o autômato é possível realizar operações de diversos tipos sobre ele, como minimização, transformação em afd, expressões regulares. A construção de linguagens através de gramáticas também é simples e apartir disso pode-se criar as linguagens como se queira.

3Disponível em http://www.jflap.org

3. SIMMoL

O SIMMoL[4] é um software de visualização e manipulação de modelos de linguagens regulares com interface gráfica e cujas funções estão no escopo de linguagens regulares, sendo, portanto impróprio para operação com linguagens não regulares, como linguagens livres de contexto, por exemplo. O SIMMoL possui uma série de funcionalidades direcionadas para autômatos, expressões regulares e gramáticas lineares a direita, verifica se um determinado autômato finito é determinístico e verifica se uma cadeia é reconhecida por um autômato finito determinístico. A figura 3.1 representa a tela principal da ferramenta.

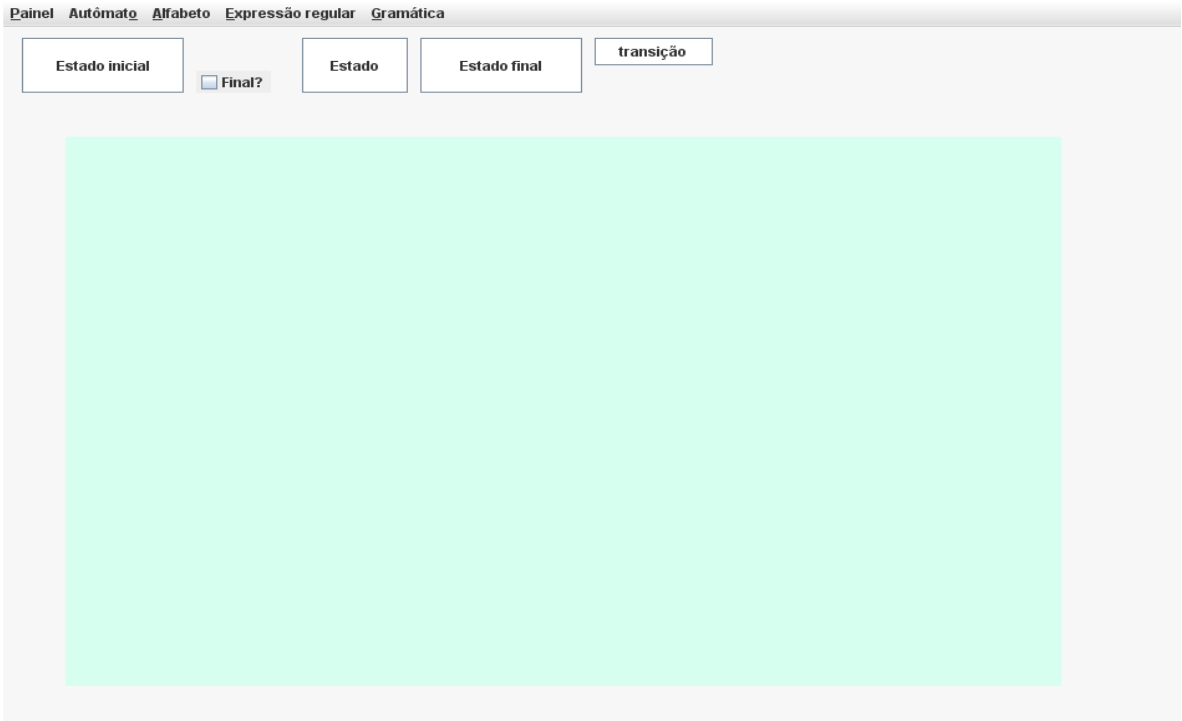

figura 3.1-tela principal do SIMMoL.

4SIMMoL é um acrônimo Sistema Interativo de Manipulação de Modelos para Linguagens

A tela principal apresenta um menu para que o usuário escolha qual modelo deve ser operado, porém já é apresentado um painel com botões e uma área para construção de autômatos. Também consta submenus para operação com outros modelos.

O menu painel possui as opções de realizar nova simulação, limpando assim a área de construção identificada na cor verde, e também a opção de encerrar a aplicação.

Uma vez iniciada a aplicação já está pronta para operar autômato sendo necessário primeiramente construir o autômato clicando nos botões; estado inicial, estado, estado final e transição e em seguida clicando no painel sendo este local do clicado o local em que será desenhado o estado ou transição, no caso de transição deve-se escolher o símbolo a ser usado na transição definido previamente no alfabeto.

Manipular gramáticas regulares e expressões regulares também é possível usando o menu indicado para cada operação.

3.1. CONSTRUINDO E OPERANDO AUTÔMATOS

A figura 3.2 mostra um autômato construído e pronto para ser operado.

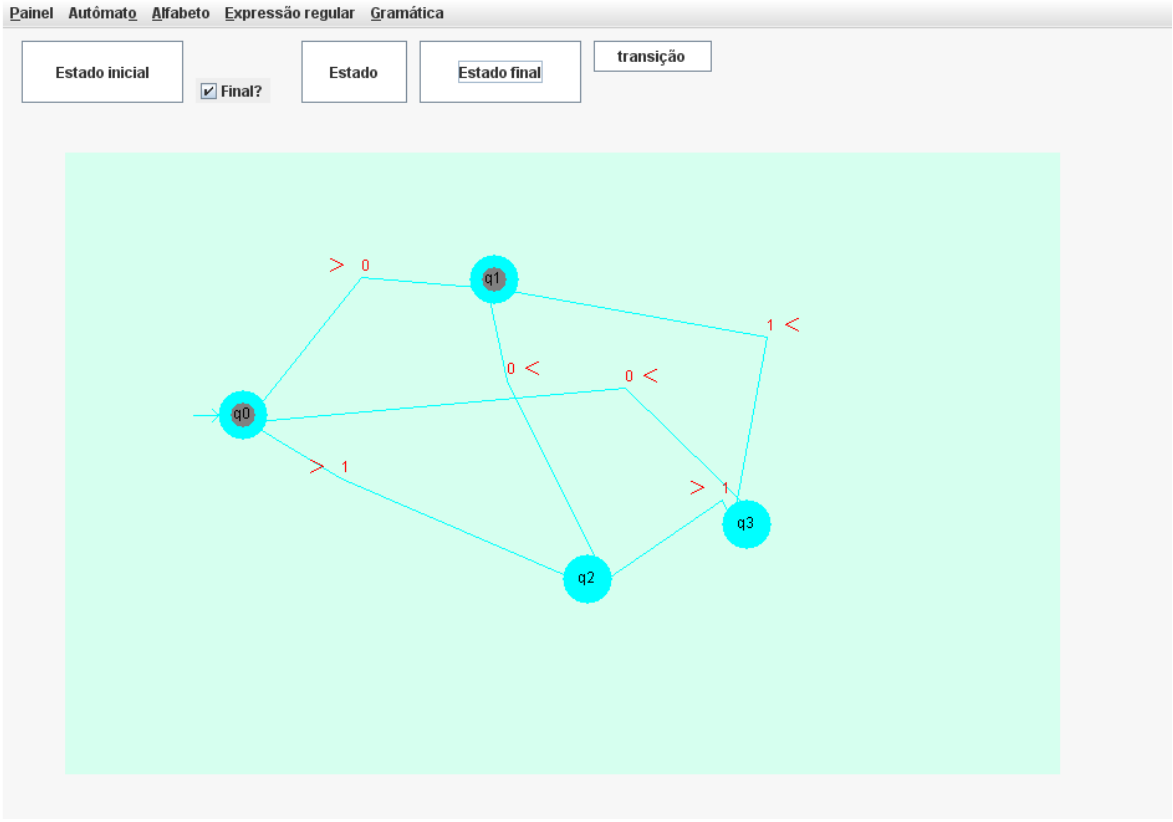

figura 3.2- Autômato construído usando o SIMMoL.

Para a melhor visualização do autômato é possível arrastar estados e transições usando um click do mouse e arrastando. O botão Estado Inicial cria um estado inicial uma vez pressionando-o e clicando na área de construção, sendo possível estabelecer que este estado é também um estado final marcando previamente a opção final. O mesmo procedimento é usado para criar estados e estados finais usando os botões: Estado e Estado final. Para adicionar uma transição basta escolher o símbolo, clicar no botão transição, clicar no estado de inicio da transição e arrastar o mouse até o estado final da transição. A transição possui uma seta de orientação > ou < indicando que o

sentido da transição é do estado mais à esquerda ou do estado mais a direita respectivamente. Uma vez construído, o autômato pode ser operado de forma a transformá-lo em um autômato determinísticos, em uma gramática ou ele pode ser minimizado, sendo que para isso este autômato deverá ser transformado em afd antes. Usando a opção do menu autômato, submenu operar autômato e item Transformar em afd o autômato em questão será transformado em um autômato determinístico finito, conforme mostrado na figura 3.3.

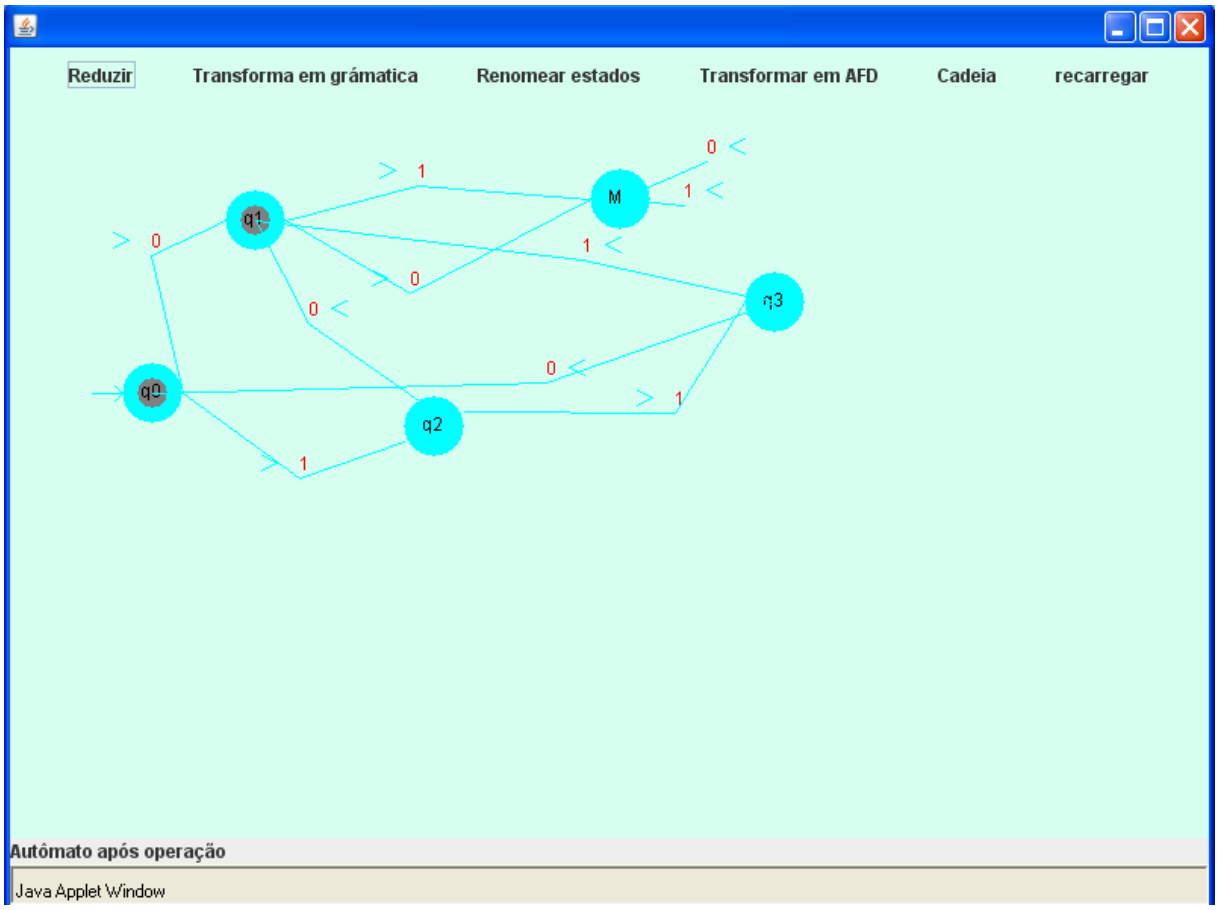

figura 3.3- autômato finito determinístico equivalente ao autômato da figura 3.2.

Uma vez transformado o autômato em questão será apresentado em outra tela e nesta poderá ser operado novamente, de forma a transformá-lo em gramática, reduzi-lo a um numero mínimo de estado, re-nomear os seus

estados, transformá-lo em autômato determinístico finito (mas este não é o caso uma vez que ele já é autômato determinístico) e se este já for afd será mostrado uma mensagem de confirmação para prosseguir ou não na operação, ou recarregar a área de amostra. Esta última opção se aplica ao autômato que propicie um difícil entendimento, então, pode-se recarregar a área ou clicar e arrastar estado e transições. A figura 3.4 mostra este autômato com novos rótulos para seus estados, e a figura 3.5 mostra este autômato com um número mínimo de estados.

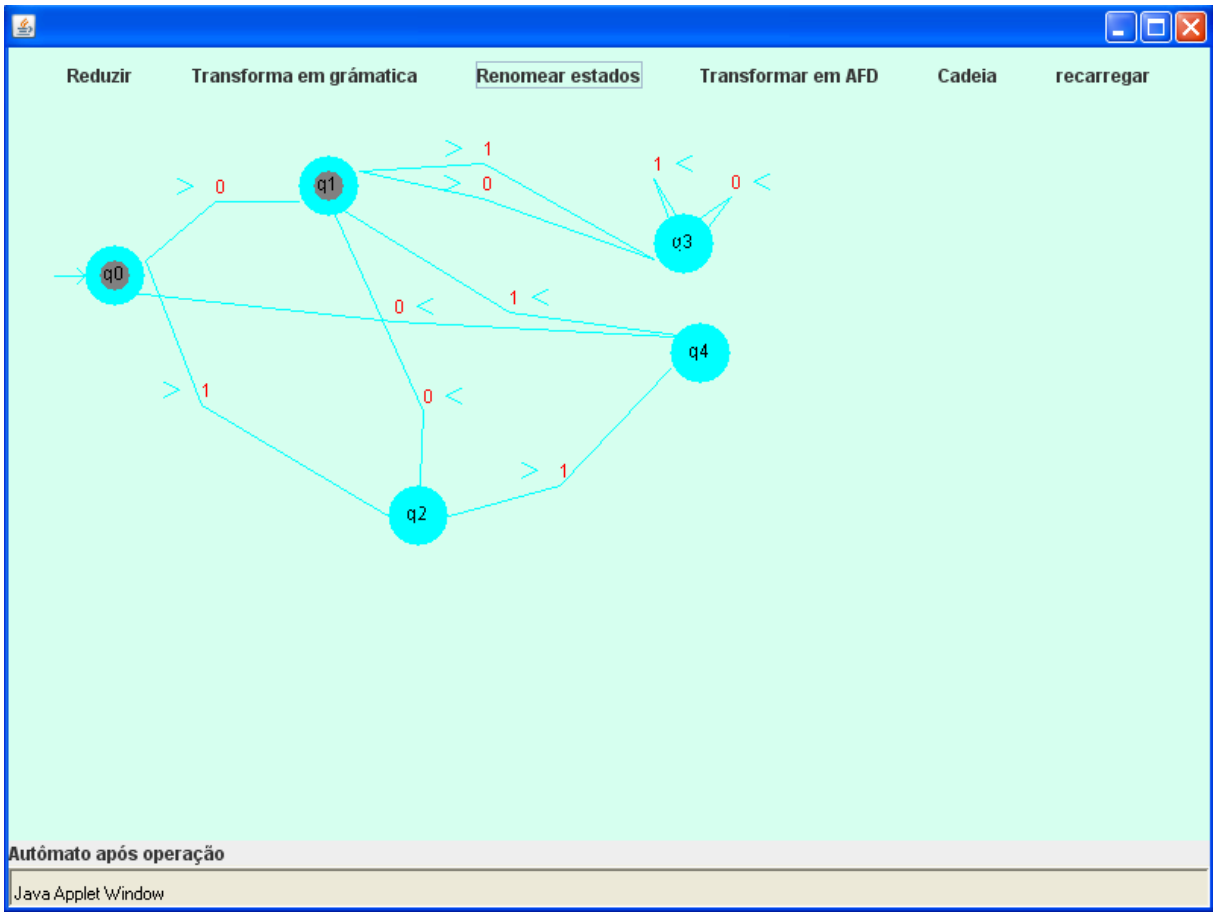

figura 3.4. Re-nomeando de rótulos para um autômato usando o SIMMoL

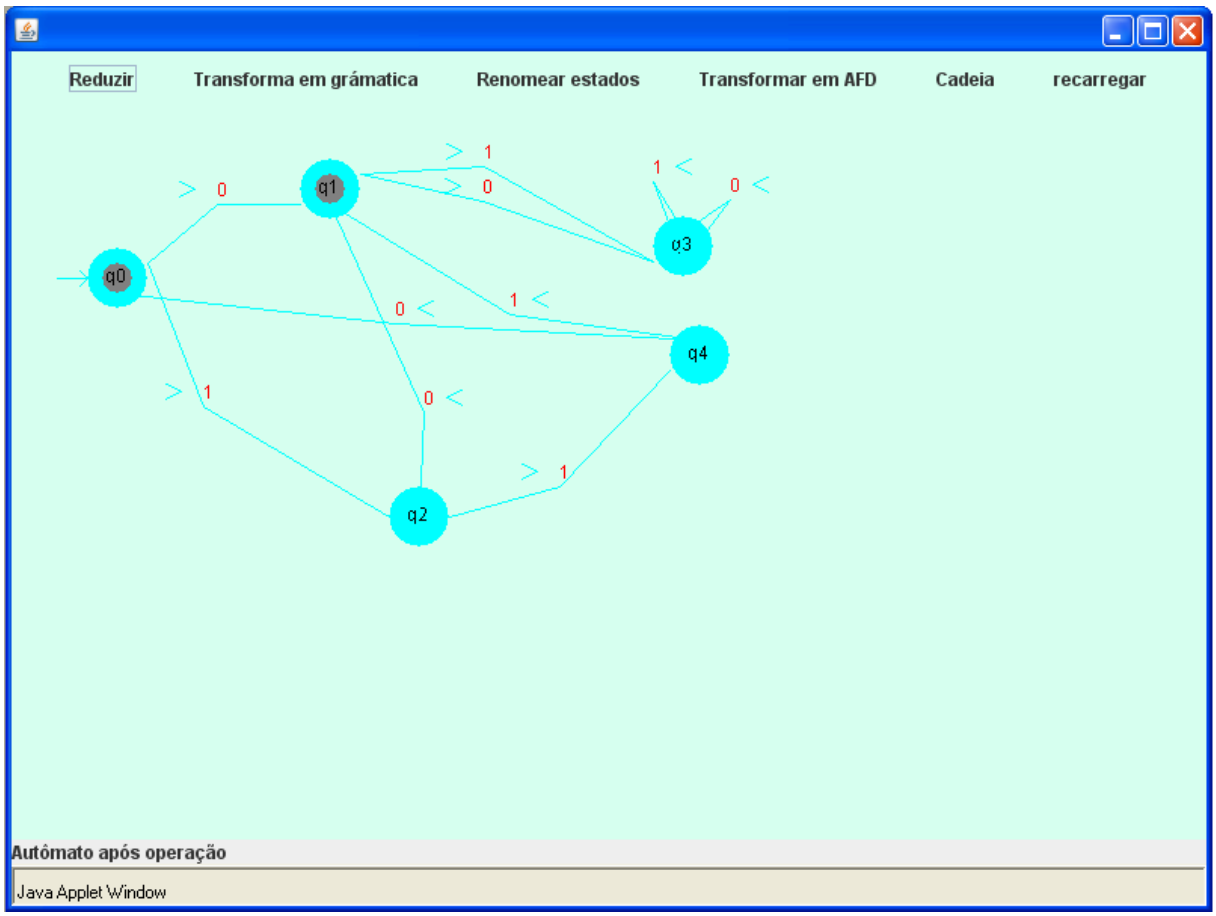

figura 3.5. Autômato com numero mínimo de estados

A qualquer momento pode-se gerar uma gramática linear à direita com mostrado na figura 3.6 ou determinar se uma cadeia é reconhecida por esse autômato bastando para isso clicar no botão cadeia e digitar a cadeia a ser procurada, conforme figura 3.7 e será retornado true em caso verdadeiro e false caso falso.

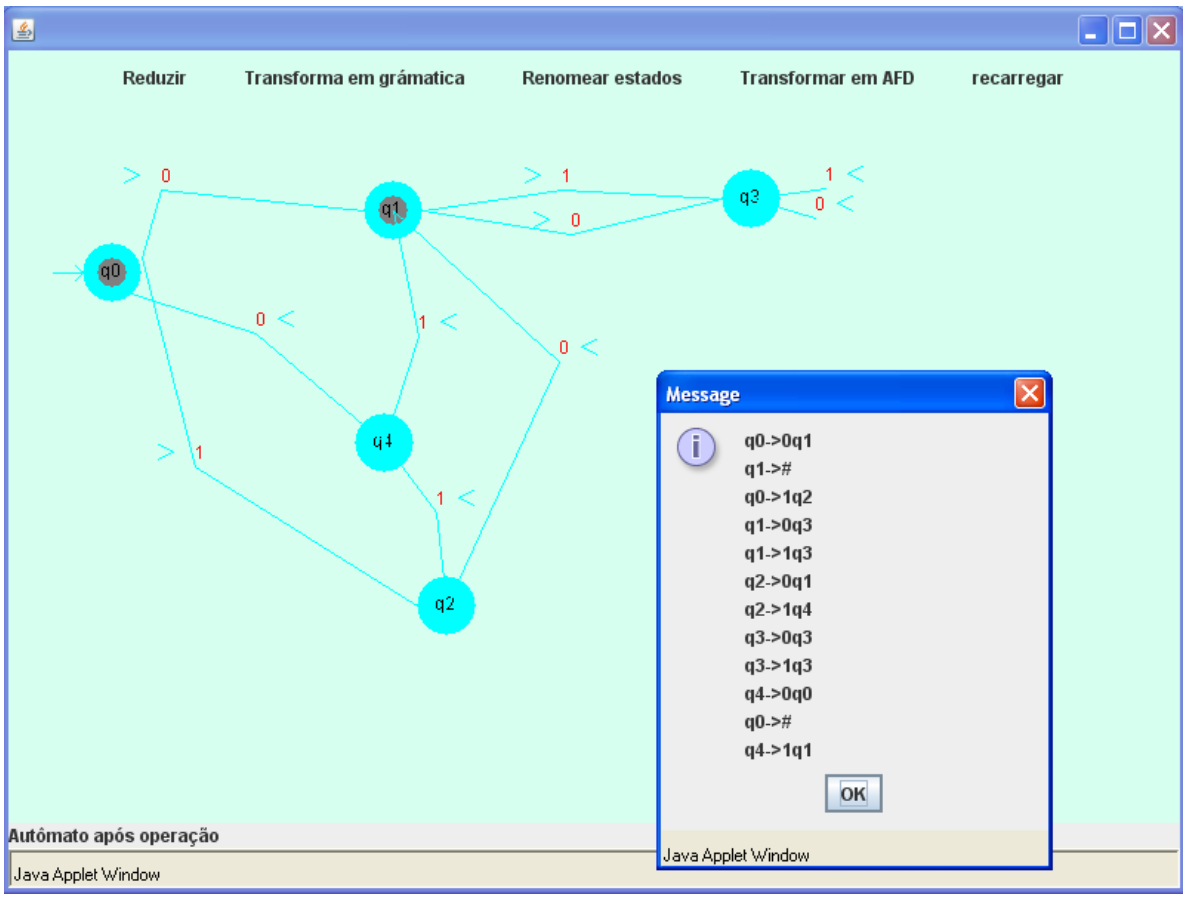

figura 3.6. Gramática linear à direita para o autômato.

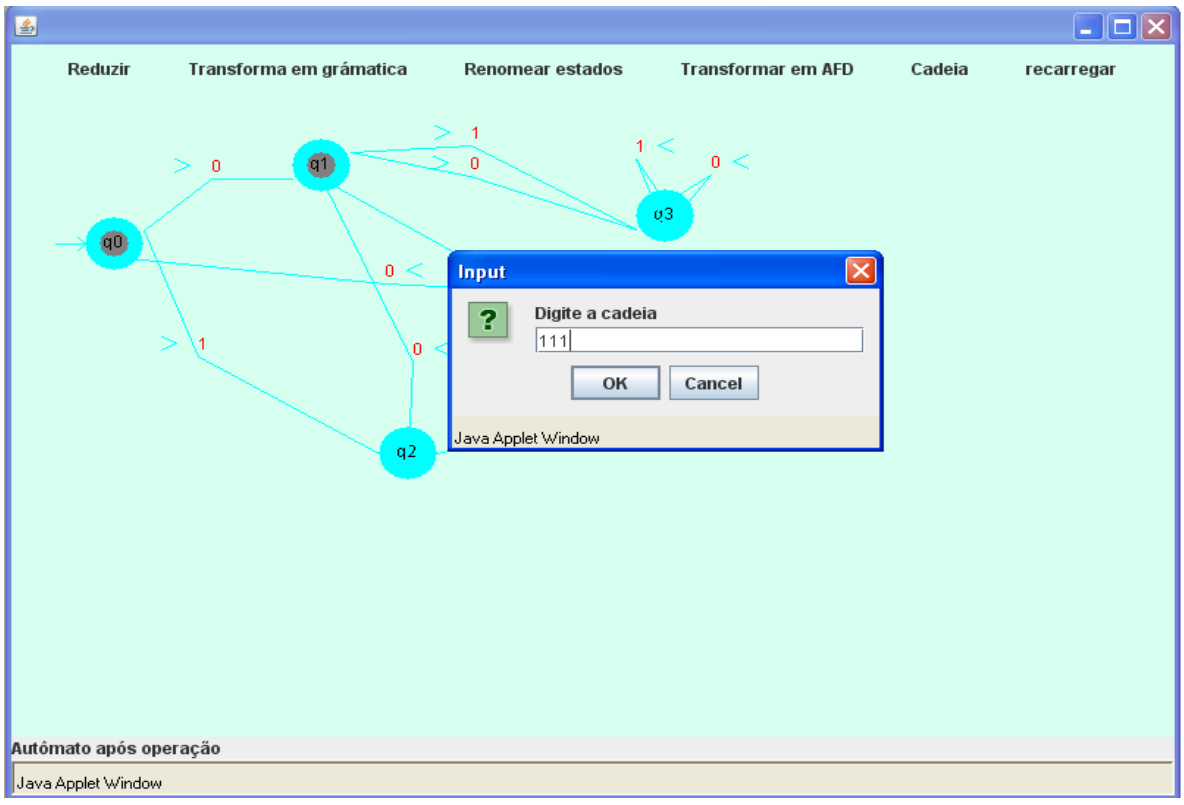

figura 3.7- Reconhecendo cadeia com SIMMoL

3.2. CONSTRUINDO E OPERANDO GRAMÁTICAS

Para construir gramática deve-se usar o menu Gramática e em uma área de construção de gramática construir as produções e adicioná-la à gramática conforme mostrado na figura 3.8. Uma vez criada a gramática pode-se transformá-la em autômato clicando em transformar, e este autômato será mostrado em um painel igual ao painel mostrado na figura 3.9, onde poder-se-á realizar operações neste autômato.

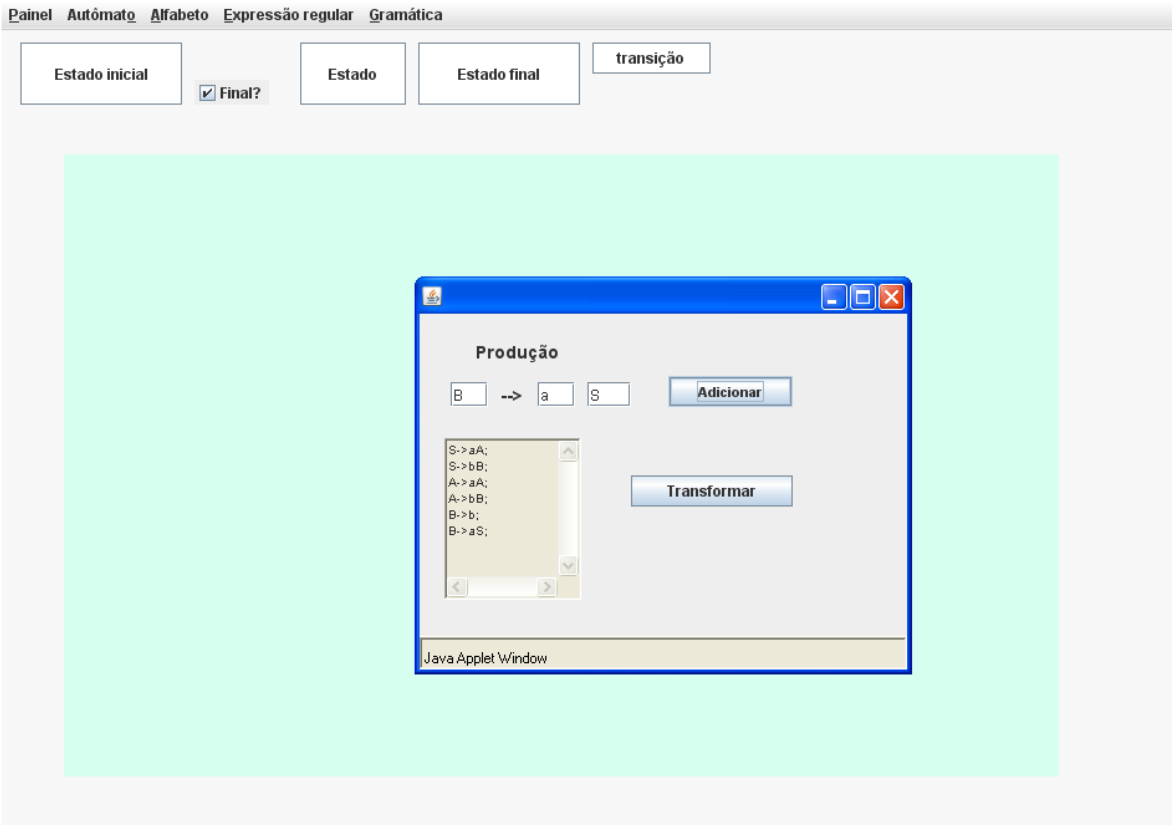

figura 3.8. Área de construção de uma gramática.

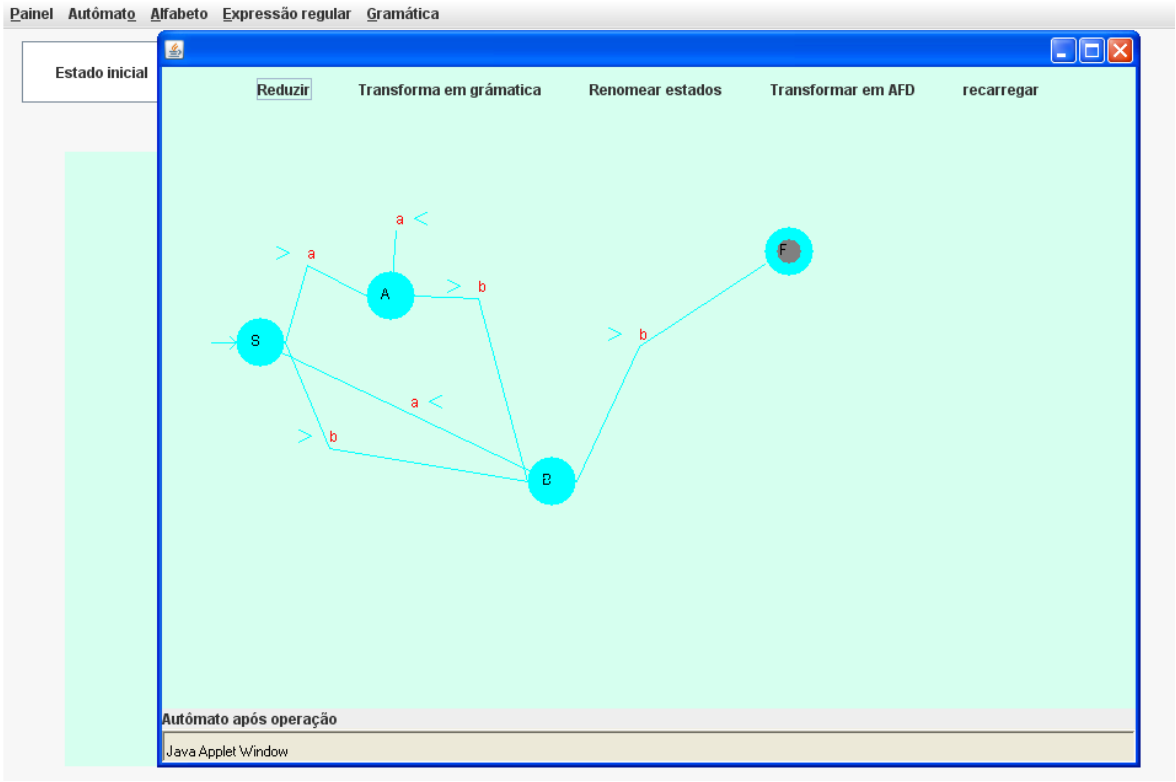

figura 3.9. Gramática da figura 3.8 convertida em autômato.

3.3. CONSTRUINDO E OPERANDO EXPRESSÕES REGULARES

O SIMMoL permite também operações sobre expressões regulares, para isso, basta usar o menu Expressão regular e o item operar expressão e construir a expressão usando símbolos de sua escolha (não necessariamente os que foram definidos no alfabeto), os operadores '+' para união, '.' Para concatenação e '*' para fecho e autômato correspondente será mostrado em um painel. As figuras 3.10 e 3.11 mostram a construção da expressão (a+b).c e um autômato equivalente respectivamente.

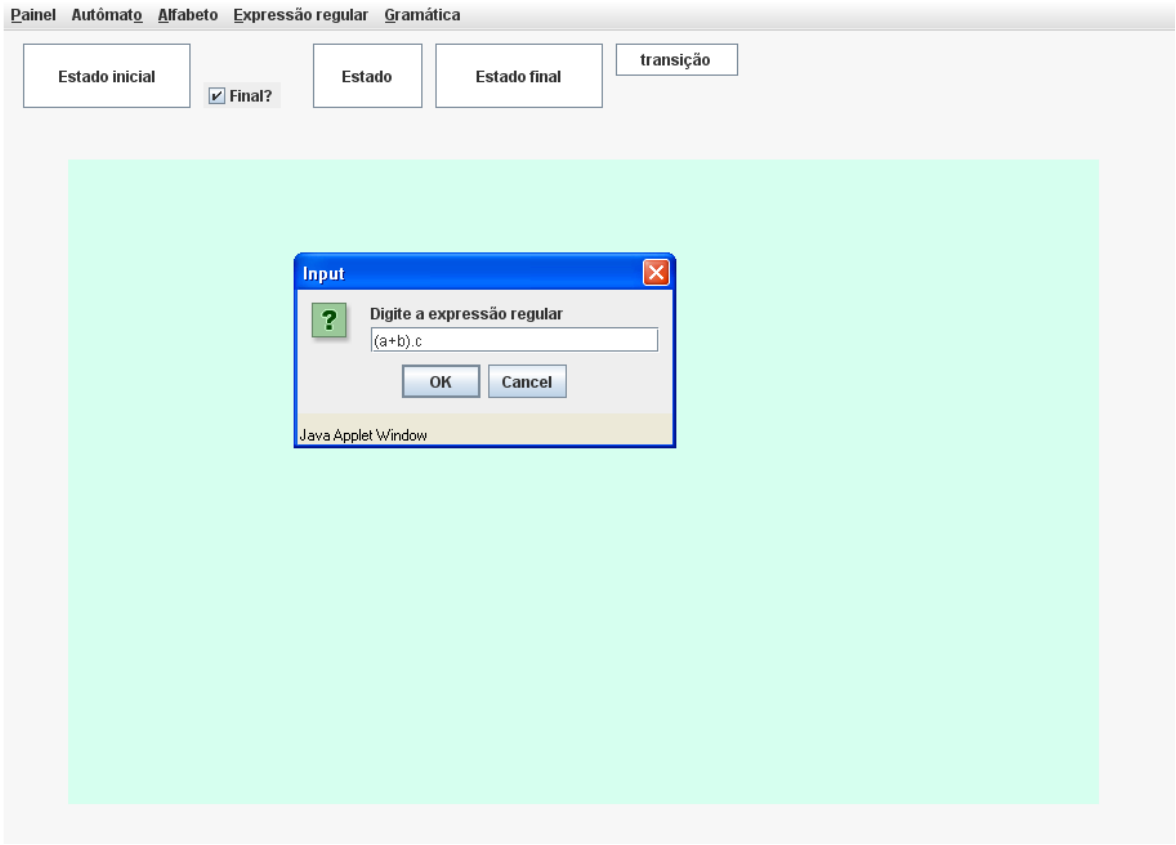

figura 3.10. Construção de expressões regulares

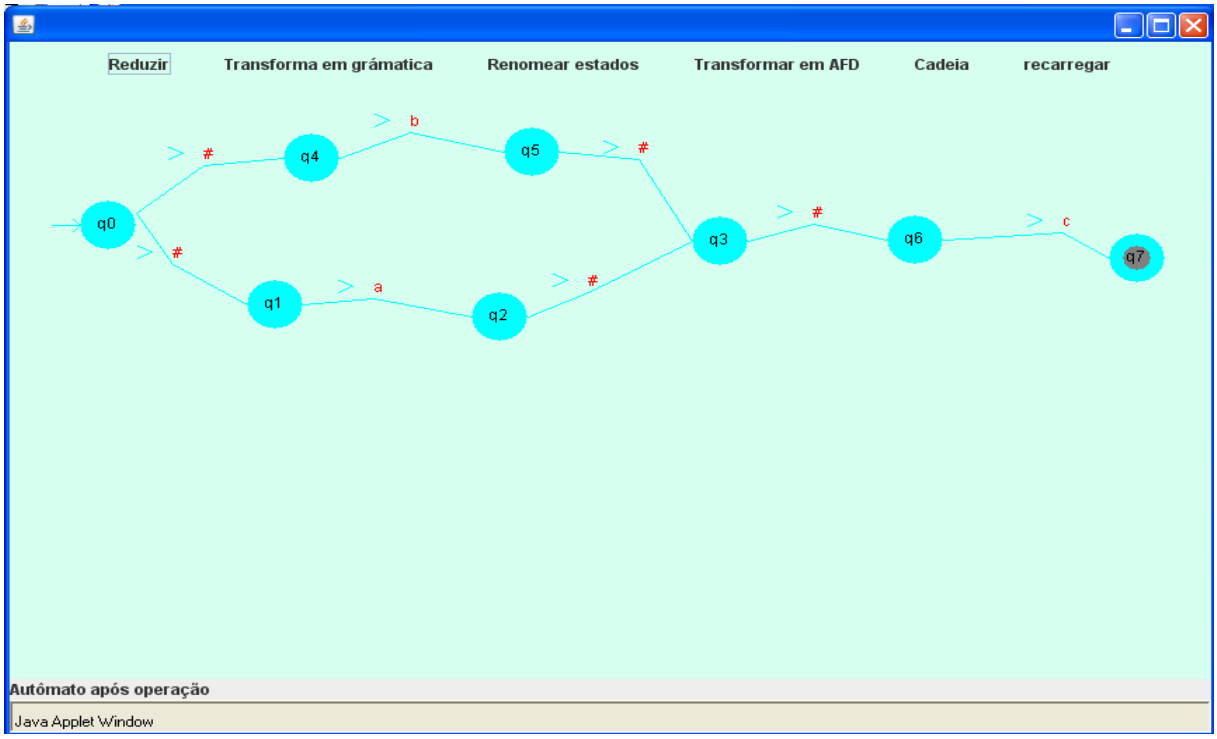

figura 3.11. Autômato equivalente à expressão (a+b).c.

4.COMPARAÇÂO ENTRE OS SOFTWARES

A título de comparação entre as ferramentas acima se pode dizer que todas são razoavelmente boas naquilo em que se propõem ou em pontos específicos, porém analisando como softwares devemos observar alguns aspectos primordiais dos softwares. Os aspectos escolhidos foram usabilidade, conformidade, portabilidade e robustez e para isso foi feito teste para a criação e manipulação de autômatos e expressões regulares de baixa e media complexidade e sempre observando os aspectos citados. Usabilidade é a característica que objetos em geral têm de ser facilmente manipulados, conformidade é o aspecto que os projetos possuem de corresponderem ao esperado em termos de resultados, portabilidade é a característica que possuem os sistemas de computação de serem implantados em diferentes ambientes e robustez é a possibilidade que um sistema possui de sobressair a situações adversas.

Testes foram realizados com as ferramentas visando a construção de autômatos e expressões regulares. Os testes procederam da seguinte forma:

1. Construção da expressão regular: (0+1).1.1.0.(0+1)* e em seguida convertê-la em um afn, converter este afn em seu afd e depois minimizá-lo;
2. Construção do AFN abaixo e em seguida transformá-lo em AFD e minimizá-lo.

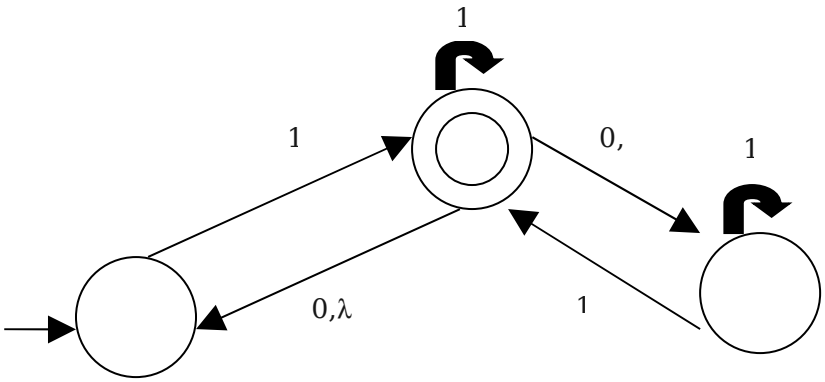

figura 3.12. Autômato usado nos testes

Boa parte das ferramentas foi bem sucedida em grande parte do teste. Analisando racionalmente o JFLAP é graficamente mais "inteligente", o que por outro lado consome maior processamento da máquina utilizada, uma vez que as operações são processadas localmente. O Language Emulator não conseguiu operar a expressão regular nem o autômato por motivos desconhecido. O SIMMoL, SAGEMoLiC por sua vez, assim como o JFLAP, executaram todos os testes com sucesso.

Um quadro comparativo abaixo ilustra os quesitos citados acima e com respeito aos testes realizados.

	Usabilidade	Conformidade	Portabilidade	Robustez
L. Emulator	Baixa	Baixa	Alta	Alta
SAGEMoLiC	Média	Alta	Alta	Alta
JFLAP	Alta	Alta	Alta	Baixa
SIMMoL	Média	Média	Alta	Alta

O SIMMoL possui uma série de desvantagens se comparado ao SAGEMoLiC, JFLAP e o Language Emulator, sobretudo no que diz respeito à abrangência conceitual, uma vez que estes manipulam

modelos não abordados no momento pelo SIMMoL, como Máquinas de Turing e simulações para o lema da bomba (no caso do JFLAP), porém concentrando-se no aspecto de interface, o SIMMoL está mais conforme que o Language Emulator e o SAGEMoLiC e no caráter robustez o JFLAP demonstrou-se inadequado para uso em máquina com baixo processamento.

CONCLUSÂO

Há muitas ferramentas que se propõem a facilitar o estudo ou a manipulação de modelos de linguagens formais, e de certa forma as ferramentas existentes realizam de maneira satisfatória grande parte dos conceitos relacionados às linguagens formais, porém as ferramentas utilizadas ainda não conseguem apresentar grafos em molduras de maneira a se adaptar inteligentemente ao formato de autômatos. O SIMMoL foi proposto inicialmente para manipular formalidades de linguagens regulares, e de certa forma corresponde satisfatoriamente esta condição, porém há diversos outros conceitos no que diz respeito a linguagens formais que ainda podem ser explorados por ele, como verificação de igualdade entre modelos, testes para linguagens regulares (verificar se uma linguagem é regular), verificação de cadeias, construir um algoritmo para minimização "intuitiva" de expressões regulares entre outras operações e ainda abordar outras classes de linguagens como linguagens livres do contexto, linguagens sensíveis ao contexto e linguagens recursivamente enumeráveis. Tais abrangências podem ser incorporadas ao SIMMoL em trabalhos futuros.

Bibliografia

[HU79] Hopcroft and Ullman. Introduction to automato theory, Languages and Computation. Reading, Mass.: Addison-Wesley, 1979.

[DD01] Deitel e Deitel. Java: Como Programar, Porto Alegre, RS.: Bookman, 2001.

[ABL02] Acióly, Bedregal e Lyra: Introdução à Teoria das Linguagens Formais, dos Autômatos e da Computabilidade.: Natal, RN.: Edições UNP, 2002.

[AFPS02] Ayala-Rincón,Fonseca,Poubel and Siqueira: A framework to visualize equivalences between computational models of regular languages.:Information processing letters, vol. 84, número 1,
p: 5-16,2002.