ANIBAL SANTOS JUKEMURA

GINUX ABSTRACT MACHINE

PROPOSTA DE UM SIMULADOR DE OPERAÇÕES SOBRE AUTÔMATOS EM AMBIENTE LINUX PARA AUXÍLIO AO ESTUDO ACADÊMICO DE LINGUAGENS FORMAIS E MÁQUINAS ABSTRATAS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Pós-graduação *Lato Sensu* Administração em Redes Linux para a obtenção do título de especialista.

Orientador
Prof. MSc Joaquim Quinteiro Uchôa

LAVRAS
MINAS GERAIS – BRASIL
2004

ANIBAL SANTOS JUKEMURA

GINUX ABSTRACT MACHINE

PROPOSTA DE UM SIMULADOR DE OPERAÇÕES SOBRE AUTÔMATOS EM AMBIENTE LINUX PARA AUXÍLIO AO ESTUDO ACADÊMICO DE LINGUAGENS FORMAIS E MÁQUINAS ABSTRATAS

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Pós-graduação *Lato Sensu* Administração em Redes Linux para a obtenção do título de especialista.

Aprovada em 26 de abril de 2004

_____
Prof. Samuel Pereira Dias

_____
Prof. MSc. Rudini Menezes Sampaio

_____
Prof. MSc. Joaquim Quinteiro Uchôa
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL
2004

Agradecimentos

Ao Anderson Pereira Ataídes pelo auxílio na compreensão da programação em linux e do uso do GTK como conjunto de bibliotecas principais de desenvolvimento. Com o seu apoio e suporte iniciais o projeto pôde ser inicializado em tempo adequado.

À Juliana Imília Pinheiro de Oliveira Souza pelo apoio e incentivo recebidos para a viabilização deste trabalho.

Resumo

Um Autômato Finito é um modelo matemático de um sistema que descreve entradas e saídas. O estado do sistema sumariza a informação obtida em entradas informadas, que é necessária para determinar o comportamento do sistema em entradas subseqüentes. Na Ciência da Computação, a teoria de Autômatos Finitos é bastante útil para a modelagem desses Autômatos ou máquinas abstratas. O estudo de sistemas de estados finitos e máquinas abstratas resume-se no fato de que esses sistemas são aplicáveis em diversos ambientes computacionais e matemáticos, como importante ferramenta para modelagem e solução de problemas. Neste trabalho, será desenvolvida uma ferramenta para a simulação e modelagem das máquinas de estados finitos usadas no aprendizado e compreensão do complexo de sistemas de estados finitos. A ferramenta proposta dará condições de simular as máquinas abstratas em suas diversas classificações, bem como também realizar as principais operações e conversões definidas para essas classes.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

Na Ciência da Computação, a teoria de Autômatos Finitos é bastante útil para a modelagem de sistemas de estados finitos. Compiladores, analisadores léxicos e editores de textos por exemplo, fazem amplo uso desses sistemas.

Sob esse aspecto, a mais importante razão para o estudo de sistemas de estados finitos e máquinas abstratas resume-se na naturalidade de conceitos que envolve toda sua teoria, simplesmente pelo fato de que essas surgem em diversas aplicações computacionais e matemáticas, como importante ferramenta para modelagem e solução de problemas.

A *Ginux Abstract Machine* (GAM), produto deste trabalho, surge como uma importante ferramenta para a simulação e modelagem das máquinas de estados finitos. Essa ferramenta pode ser usada no aprendizado e compreensão do uso de sistemas de estados finitos em sua aplicabilidade funcional.

A *Ginux Abstract Machine* fornece condições de simular as máquinas abstratas em suas diversas classificações, bem como também realiza as principais operações e conversões definidas para essas classes. O principal propósito da GAM é o de auxiliar o meio acadêmico na compreensão da teoria que envolve os Autômatos e Linguagens Formais.

A ferramenta foi desenvolvida sob licença GPL, em ambiente gráfico, para o sistema operacional GNU/Linux. Possui seu código fonte e executáveis disponíveis e acessíveis ao meio acadêmico.

O texto da monografia está dividido em oito capítulos. O Capítulo 2 apresenta uma breve explicação de sistemas de estados finitos. Esse capítulo resume-se na definição das máquinas abstratas e nos conceitos principais de sua teoria.

Um outro aspecto importante é o da operação de minimização de máquinas, no que se refere à quantidade de estados envolvidos na operação. Toda essa teoria é apresentada no Capítulo 3.

O Capítulo 4 descreve as máquinas que apresentam uma estrutura auxiliar de pilha para análise de linguagens mais complexas. O Capítulo 5 descreve a Máquina de Turing como uma importante ferramenta de análise de Linguagens Formais. Já no Capítulo 6 encontra-se uma breve descrição da *Ginux Abstract Machine* (GAM), destacando-se os benefícios que essa ferramenta poderá trazer no estudo das máquinas abstratas. O capítulo também contém uma breve explicação do ambiente de desenvolvimento usado e quais as ferramentas que foram necessárias para implementar a GAM.

O Capítulo 7 apresenta a ferramenta de simulação de máquinas abstratas - GAM – em si. E, finalmente, o Capítulo 8 descreve as considerações finais sobre a ferramenta, ilustrando-se as principais vantagens e limitações dessa versão do projeto.

Ressalta-se ainda que grande parte teórica do texto foi baseada em [MENEZES].

2. Autômatos ou Máquinas Abstratas (*Abstract Machines*)

2.1. Sistemas de Estados Finitos

Um ***Sistema de Estados Finitos*** é um modelo matemático de sistema com entradas e saídas discretas. Pode assumir um número ***finito*** e ***pré-definido*** de estados. Cada estado resume somente as informações do passado necessárias para determinar as ações para a próxima entrada.

Um forte motivacional para o estudo de Sistemas de Estados Finitos é o fato de poderem ser associados a diversos tipos de sistemas naturais e construídos. Um exemplo clássico e de simples entendimento é um elevador. Trata-se de um sistema que não memoriza as requisições anteriores. Cada "estado" sumariza as informações "andar corrente" e "direção de movimento". As entradas para o sistema são requisições pendentes.

Analisadores Léxicos e Processadores de Texto (ou algumas ferramentas de Processadores de Texto) também são exemplos de sistemas de estados finitos, onde cada estado, basicamente, memoriza a estrutura do prefixo da palavra em análise.

Entretanto, nem todos os Sistemas de Estados Finitos são adequados para serem estudados por esta abordagem. Um contra-exemplo é o cérebro humano. Existem evidências de que um neurônio pode ser representado por um número finito de *bits*. O cérebro é composto por cerca de 2^{35} células. Portanto, a princípio, é possível representá-lo por um número finito de estados. Entretanto, o elevado número de combinações de células (e, conseqüentemente, de estados) determina uma abordagem pouco eficiente.

Outro contra-exemplo é o computador. Os estados determinados pelos processadores e memórias podem ser representados como um sistema de estados finitos. Entretanto, o estudo adequado da noção de computabilidade exige uma memória sem limite pré-definido.

Nas próximas páginas, é também apresentado um outro formalismo de Autômato, a Máquina de Turing, mais adequado ao estudo da computabilidade. É importante observar, entretanto, que o estudo da computabilidade e solucionabilidade de problemas não é um dos objetivos desta publicação.

2.2. Autômato Finito (Determinístico)

Um Autômato Finito Determinístico ou simplesmente Autômato Finito pode ser visto como uma máquina composta, basicamente, de três partes:

a) **Fita**. Dispositivo de entrada que contém a informação a ser processada;

b) **Unidade de Controle**. Reflete o estado corrente da máquina. Possui uma unidade de leitura (cabeça da fita) a qual acessa uma célula da fita de cada vez e movimenta-se exclusivamente para a direita;

c) **Programa ou Função de Transição**. Função que comanda as leituras e define o estado da máquina.

A fita é finita (à esquerda e à direita), sendo dividida em células, onde cada uma armazena um símbolo. Os símbolos pertencem a um alfabeto de entrada. Não é possível gravar sobre a fita (e não existe memória auxiliar). Inicialmente, a palavra a ser processada (ou seja, a informação de entrada para a máquina) ocupa toda a fita.

A unidade de controle possui um número finito e predefinido de estados. A unidade de leitura lê o símbolo de uma célula de cada vez. Após a leitura, a cabeça da fita move-se uma célula para a direita. Inicialmente, a cabeça está posicionada na célula mais à esquerda da fita, como ilustrado na Figura 1.

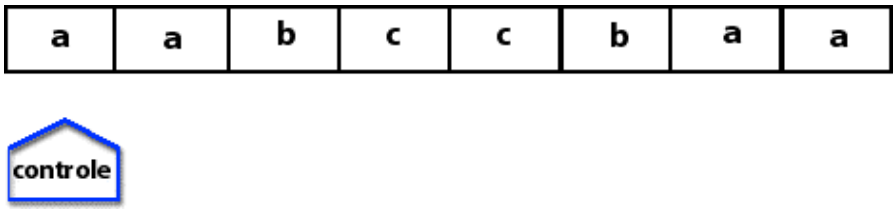

Figura 1 : Autômato finito como uma máquina com controle finito

O programa é uma função parcial que, dependendo do estado corrente e do símbolo lido, determina o novo estado do Autômato. Deve-se reparar que o Autômato Finito não possui memória de trabalho. Portanto, para armazenar as informações passadas necessárias ao processamento, deve-se usar o conceito de estado.

2.2.1. Definição: Autômato Finito Determinístico (AFD)

Em [MENEZES], um Autômato Finito Determinístico (AFD) ou simplesmente Autômato Finito (AF) M é uma 5-upla:

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

onde:

Σ = alfabeto de símbolos de entrada;

Q = conjunto de estados possíveis do Autômato o qual é finito;

δ = função programa ou função de transição:

$$\delta: Q \times \Sigma \rightarrow Q$$

a qual é uma função parcial;

q_0 = estado inicial tal que q_0 é elemento de Q;

F = conjunto de estados finais tal que F está contido em Q.

A função programa pode ser representada como um grafo finito direto, como ilustrado na Figura 2.

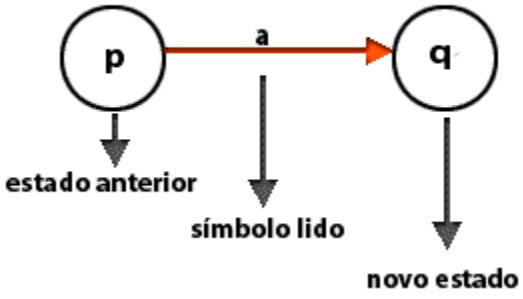

Figura 2: Representação da função programa como um grafo

Nesse caso, os estados iniciais e finais são representados como ilustrado na Figura 3.

Figura 3: Estado inicial (esq.) e final (dir. e em negrito) como vértices de grafos

O processamento de um Autômato Finito M, para uma palavra de entrada w, consiste na sucessiva aplicação da função programa para cada símbolo de w (da esquerda para a direita) até ocorrer uma situação de parada.

Obeserva-se o Exemplo 1 para um Autômato Finito:

Exemplo 1: Considere a linguagem:

$L_1 = \{ w \mid w \text{ possui aa ou bb como sub palavra } \}$

O Autômato Finito:

$$M_1 = (\{ a, b\}, \{q_0, q_1, q_2, q_f\}, \delta_1, q_0, \{ q_f \})$$

onde δ_1 (lê-se "delta um") é como se segue, representada na forma de uma tabela, reconhece a linguagem L_1.

δ_1	**a**	**b**
$\mathbf{q_0}$	q_1	q_2
$\mathbf{q_1}$	q_f	q_2
$\mathbf{q_2}$	q_1	q_f
$\mathbf{q_f}$	q_f	q_f

Tabela 1 - Representação da Função Programa

O Autômato pode ser representado pelo grafo ilustrado na Figura 4. O algoritmo apresentado usa os estados q_1 e q_2 para "memorizar" o símbolo anterior. Assim, a seguinte idéia é válida: "q_1 representa o fato de que o símbolo anterior é **a** e q_2 representa o fato de que o símbolo anterior é **b**". Após identificar dois **a** ou dois **b** consecutivos, o Autômato assume o estado q_f (final) e varre o sufixo da palavra de entrada, sem qualquer controle lógico, somente para terminar o processamento.

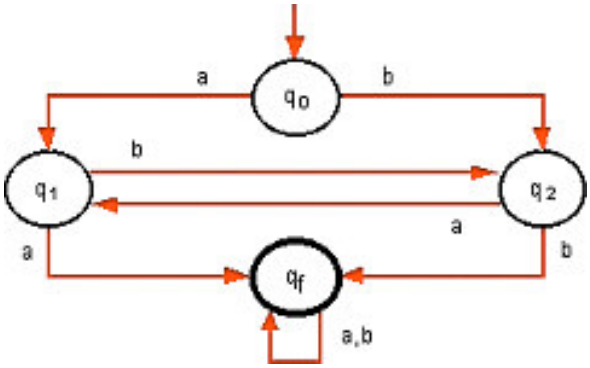

Figura 4: Exemplo de um Autômato Finito Determinístico

Um Autômato Finito sempre pára ao processar qualquer entrada pois, como toda palavra é finita e como um novo símbolo da entrada é lido a cada aplicação da função programa, não existe a possibilidade de ciclo (*loop*) infinito.

A parada de um processamento pode ser de duas maneiras: aceitando ou rejeitando uma entrada **w**. As situações são as seguintes:

a) Após processar o último símbolo da fita, o Autômato Finito assume um estado final: o Autômato pára e a entrada w é aceita;

b) Após processar o último símbolo da fita, o Autômato Finito assume um estado não-final: o Autômato pára e a entrada w é rejeitada;

c) A função programa é indefinida para o argumento (estado corrente e símbolo lido): a máquina pára e a palavra de entrada w é rejeitada.

O funcionamento de um AFD pode ser claramente compreendido, segundo o Teorema 1 indicado e provado em [SUDKAMP]:

Teorema 1 – Seja $M = (\Sigma, Q, \delta, Q_0, F)$ um AFD e seja $\omega \in \Sigma^*$. Então ω determina um único caminho p_ω no diagrama de estados de M e $\underline{\delta}(q_0, \omega) = q_\omega$.

Para complementar o estudo de AFDs, deve-se compreender a definição de Função Programa Estendida. Seja $M = (\Sigma, Q, \delta, q_0, F)$ um Autômato Finito Determinístico. A Função Programa Estendida denotada por:

$$\underline{\delta}: Q \times \Sigma^* \rightarrow Q$$

é a função programa : $\delta: Q \times \Sigma \rightarrow Q$ estendida para palavras e é indutivamente definida como segue:

$$\underline{\delta}(q, \varepsilon) = q$$

$$\underline{\delta}(q, aw) = \underline{\delta}(\delta(q, a), w)$$

Exemplo 2: Seja o Autômato Finito $M_1 = (\{a, b\}, \{q_0, q_1, q_2, q_f\}, \delta_1, q_0, \{q_f\})$ representado anteriormente. Então, a função programa estendida aplicada à palavra **abaa** a partir do estado inicial q_0 é como segue:

$\underline{\delta}(q_0, abaa) =$	função estendida sobre abaa
$\underline{\delta}(\delta(q_0, a), baa) =$	processa **a**baa
$\underline{\delta}(q_1, baa) =$	função estendida sobre baa
$\underline{\delta}(\delta(q_1, b), aa) =$	processa **b**aa
$\underline{\delta}(q_2, aa) =$	função estendida sobre aa
$\underline{\delta}(\delta(q_2, a), a) =$	processa **a**a
$\underline{\delta}(q_1, a) =$	função estendida sobre a
$\underline{\delta}(\delta(q_1, a), \varepsilon) =$	processa aba**a**
$\underline{\delta}(q_f, \varepsilon) = q_f$	função estendida sobre ε: fim da indução

e, portanto, a palavra é aceita $q_f \in F_1$.

A linguagem aceita por um Autômato Finito M = (Σ, Q, δ, q_0, F), denotada por **ACEITA (M)** ou **L(M)**, é o conjunto de todas as palavras pertencentes a Σ* aceitas por M, ou seja:

$$\mathbf{ACEITA(M) = \{ w \mid \delta (q_0, w) \in F \}}$$

Analogamente, **REJEITA(M)** é o conjunto de todas as palavras pertencentes a Σ* rejeitadas por **M**. As seguintes afirmações são verdadeiras:

- ACEITA(M) ∩ REJEITA(M) = ∅
- ACEITA(M) ∪ REJEITA(M) = Σ*
- o complemento de ACEITA (M) é REJEITA (M)
- o complemento de REJEITA (M) é ACEITA (M)

Cita-se ainda que, dois Autômatos Finitos M_1 e M_2 são ditos *Autômatos Finitos Equivalentes* se, e somente se:

$$ACEITA(M_1) = ACEITA(M_2)$$

2.3. Autômato Finito Não-Determinístico (AFND)

O não-determinismo é uma importante generalização dos modelos de máquinas, sendo de fundamental importância no estudo da Teoria da Computação e da Teoria das Linguagens Formais. Nem sempre a facilidade de não-determinismo aumenta o poder de reconhecimento de linguagens de uma classe de Autômatos. Por exemplo, conforme é mostrado adiante, qualquer Autômato Finito Não-Determinístico pode ser simulado por um Autômato Finito Determinístico.

A facilidade de *não-determinismo* para Autômatos Finitos é interpretada como segue: a função programa, ao processar uma entrada composta pelo estado corrente e símbolo lido, tem como resultado um conjunto de novos estados.

Visto como uma máquina composta por fita, unidade de controle e programa, pode-se afirmar que um Autômato Não-Determinístico assume um conjunto de estados alternativos, como se houvesse uma multiplicação da unidade de controle.

Para cada alternativa, ocorre um processamento independente e sem compartilhamento de recursos. Assim, o processamento de um caminho não influi no estado, símbolo lido e posição da cabeça dos demais caminhos alternativos.

2.3.1. Definição: Autômato Finito Não-Determinístico

Um Autômato Finito Não-Determinístico (AFND) M, como descrito em [MENEZES], é uma 5-upla:

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

Onde:

Σ= alfabeto de símbolos de entrada;

Q= conjunto de estados possíveis do Autômato o qual é finito;

δ = função programa ou função de transição:

$$\delta: Q \times \Sigma \rightarrow 2^Q \ (2^Q = \text{Conjuntos das Partes de Q})$$

a qual é uma função parcial;

q_0= estado inicial tal que q_0 é elemento de Q;

F= conjunto de estados finais tal que F está contido em Q.

Excetuando-se pela função programa, as componentes, Q, q_0 e F são como na definição do AFD. A função programa pode ser representada como um grafo finito direto, como ilustrado na Figura 5, supondo que:

$$\delta(q, a) = \{ p_1, p_2, ..., pn \}$$

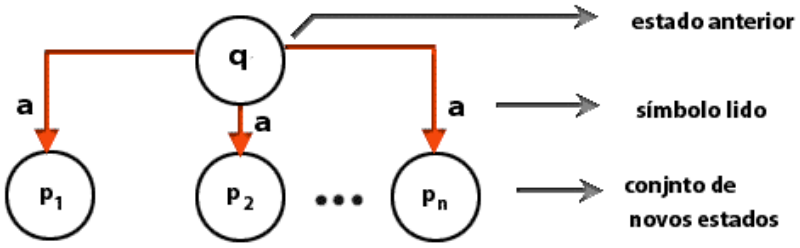

Figura 5 : Função programa não-determinística como um grafo

O processamento de um Autômato Finito Não-Determinístico M para um conjunto de estados, ao ler um símbolo, é a união dos resultados da função programa aplicada a cada estado alternativo.

Como ocorre com os AFDs, é essencial que se compreenda a definição de Função Programa Estendida, como segue:

Seja M = (Σ, Q, δ, q_0, F) um Autômato Finito Não-Determinístico. A *Função Programa Estendida* de M denotada por:

$$\underline{\delta}: 2^Q \times \Sigma^* \rightarrow 2^Q$$

é a função programa $\delta: Q \times \Sigma \rightarrow 2^Q$ estendida para palavras e é indutivamente definida como:

$$\underline{\delta}(P, \varepsilon) = P$$

$$\underline{\delta}(P, aw) = \underline{\delta}\left(\cup_{q \in P} \delta(q, a), w\right)$$

Para um dado conjunto de estados $\{q_1, q_2, ..., q_n\}$ e para um dado símbolo a:

$$\underline{\delta}(\{q_1, q_2, ..., q_n\}, a) = \delta(q_1, a) \cup \delta(q_2, a) \cup ... \cup \delta(q_n, a)$$

A linguagem aceita por um Autômato Finito Não-Determinístico M = (Σ, Q, δ, q_0, F), denotada por **ACEITA (M)** ou **L(M)**, é o conjunto de todas as

palavras pertencentes a Σ* tais que existe pelo menos um caminho alternativo que aceita a palavra, ou seja:

ACEITA(M) = {w| existe $q \in \delta(q_0, w)$ tal que $q \in F$}

Analogamente, **REJEITA (M)** é o conjunto de todas as palavras pertencentes a Σ* rejeitadas por todos os caminhos alternativos de M (a partir de q_0).

Cita-se o Exemplo 3 que mostra um Autômato Finito Não-Determinístico usado para representar a linguagem L_2 = {w | w possui aaa como sufixo}.

Exemplo 3: O Autômato Finito Não-Determinístico:

$$M_2 = (\{a, b\}, \{q_0, q_1, q_2, q_f\}, \delta_2, q_0, \{q_f\})$$

ilustrado na Figura 6, é tal que ACEITA(M_2) = L_2.

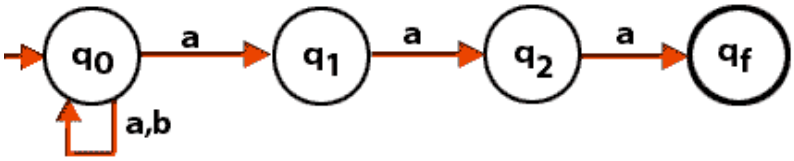

Figura 6: Exemplo de AFND

Embora a facilidade de não-determinismo seja, aparentemente, um significativo acréscimo ao Autômato Finito, na realidade não aumenta seu poder computacional. Assim, para cada AFND, é possível construir um AFD equivalente que realiza o mesmo processamento. O contrário também é verdadeiro. A prova do Teorema 2, descrito a seguir, pode ser visto em [HOPCROFT]:

Teorema 2 – Seja L um conjunto aceito por um Autômato Finito não-determinístico. Então existe um Autômato Finito determinístico que aceita L.

Outros teoremas que complementam esse estudo podem ser encontrados em detalhes em [AHO], [BARR], [DEGANO] e [SERNADAS].

2.4. Autômato Finito com Movimentos Vazios

Movimentos vazios constituem uma generalização dos modelos de máquinas não-determinísticas.

Um movimento vazio é uma transição sem leitura de símbolo algum da fita. Pode ser interpretado como um não-determinismo interno ao Autômato o qual é encapsulado, ou seja, excetuando-se por uma eventual mudança de estados, nada mais pode ser observado sobre um movimento vazio.

Uma das vantagens dos Autômatos Finitos com Movimentos Vazios no estudo das Linguagens Formais é o fato de facilitar algumas construções e demonstrações relacionadas com os Autômatos.

No caso dos Autômatos Finitos, a facilidade de movimentos vazios não aumenta o poder de reconhecimento de linguagens.

Conforme é mostrado adiante, qualquer Autômato Finito com Movimentos Vazios pode ser simulado por um Autômato Finito Não-Determinístico.

2.4.1. Definição: Autômato Finito com Movimentos Vazios

De acordo com [MENEZES], um Autômato Finito Não-Determinístico e com Movimentos Vazios (AFNDε) ou simplesmente Autômato Finito com Movimentos Vazios (AFε) M é uma 5-upla:

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

onde:

Σ= alfabeto de símbolos de entrada;

Q= conjunto de estados possíveis do Autômato o qual é finito;

δ = função programa ou função de transição:

$$\delta: Q \times \{\Sigma \cup \{\varepsilon\}\} \rightarrow 2^Q$$

a qual é uma função parcial;

q_0= estado inicial tal que q_0 é elemento de Q;

F= conjunto de estados finais tal que F está contido em Q.

Portanto, excetuando-se pela função programa, as componentes $\sum$, Q, F e q_0 são como na definição de AF.

Um movimento vazio (ou transição vazia) é representado pela aplicação da função programa, em um dado estado q ao símbolo especial ε, ou seja, $\delta(q, \varepsilon)$. A função programa com movimentos vazios pode ser interpretada como um grafo finito direto.

O processamento de um AFε é análogo ao de um AFND. Adicionalmente, o processamento de uma transição para uma entrada vazia também é não-determinística. Assim, um AFε ao processar uma entrada vazia assume simultaneamente os estados destino e origem.

A origem de um movimento vazio sempre é um caminho alternativo.

Exemplo 4: Considere a linguagem:

L_3 = {w | qualquer símbolo "a" antecede qualquer símbolo "b"}

A máquina $M_3 = (\{a, b\}, \{q_0, q_f\}, \delta_3, q_0, \{q_f\})$, é tal que ACEITA($M_3$) = L_3, como pode ser verificado na Figura 7.

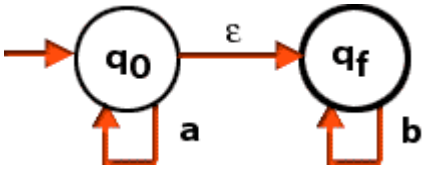

Figura 7: Exemplo de AFε

Para essa classe de Autômatos, é interessante ressaltar a importância das definições de Função Fecho Vazio e Função Fecho Vazio Estendida.

Assim sendo, seja M = (Σ , Q, δ, q_0, F) um Autômato Finito com Movimentos Vazios.

a)A *Função Fecho Vazio* denotada por:

FECHO-ε: Q é indutivamente definida como segue:

FECHO-ε(q) = {q}, se δ (q, ε) é indefinido;

FECHO-ε(q) = {q} $\cup$ δ(q, ε) $\cup$ ($\cup_{p \in \delta(q, \varepsilon)}$ FECHO-ε(p)), caso contrário.

b)A *Função Fecho Vazio Estendida* denotada por:

$$\underline{\text{FECHO-}\varepsilon} : 2^Q \rightarrow 2^Q$$

<u>FECHO-ε</u>: é a função fecho vazio estendida para conjunto de estados e é tal que:

$$\underline{\text{FECHO-}\varepsilon}(P) = \cup_{q \in P} \text{FECHO-}\varepsilon(q)$$

Observa-se o seguinte Autômato Finito com Movimentos Vazios

$$M_4 = (\{a, b\}, \{q_0, q_f\}, \delta_4, q_0, \{q_f\}).$$

Então:

FECHO-ε (q_0) = {q_0, q_f}

FECHO-ε (q_f) = {q_f}

FECHO-ε ({q_0, q_f}) = {q_0, q_f}

Seja M = ($\sum$, Q, δ, q_0, F) um Autômato Finito com Movimentos Vazios. A *Função Programa Estendida* de M denotada por:

$$\underline{\delta} : 2^Q \times \sum^* \rightarrow 2^Q$$

é a função programa : δ: Q x $\{\Sigma \cup \{\varepsilon\}\} \rightarrow 2^Q$ estendida para um conjunto finito de estados e para uma palavra e é indutivamente definida como segue:

$\delta(P, \varepsilon)$ = FECHO-ε(P)

$\delta(P, wa)$ = FECHO-ε(R) onde R = $\{r \mid r \in \delta(s, a)$ e $s \in \delta(P, w)\}$

A linguagem aceita por um *Autômato Finito com Movimentos Vazios* M = ($\sum$, Q, δ, q_0, F), denotada por ACEITA (M) ou L(M), é o conjunto de todas as palavras pertencentes a $\sum^*$ tais que existe pelo menos um caminho alternativo que aceita a palavra, ou seja:

ACEITA(M) = $\{w \mid$ existe $q \in \delta(q_0, w)$ tal que $q \in F\}$

Analogamente, REJEITA (M) é o conjunto de todas as palavras de $\sum^*$ rejeitadas por todos os caminhos alternativos de M (a partir de q_0).

Considere a linguagem:

L = {w | w possui como sufixo **a** ou **bb** ou **ccc**}

O Autômato Finito com Movimentos Vazios:

M = ({a, b, c}, $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}$, δ, q_0, $\{q_f\}$)

ilustrado na Figura 8, é tal que ACEITA(M) = L.

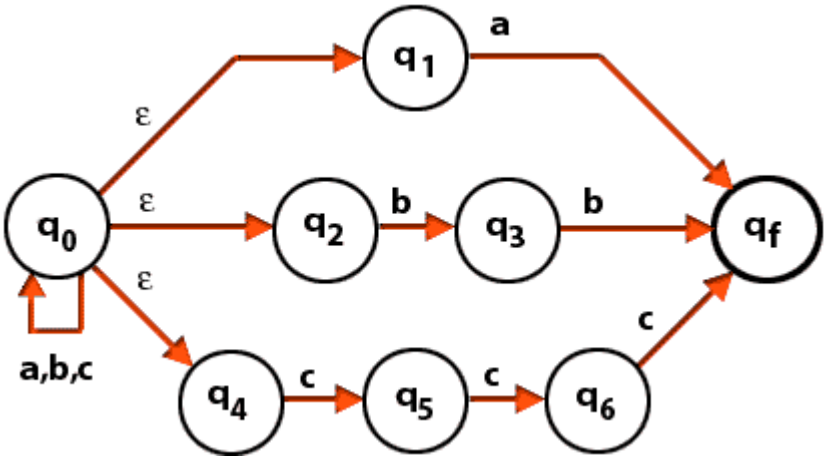

Figura 8: Autômato Finito com Movimentos Vazios

Relativamente ao fecho vazio, tem-se que, por exemplo, FECHO-ε ($\{q_0\}$) = $\{q_0, q_1, q_2, q_4\}$. Exemplificando a função estendida, tem-se que:

$\underline{\delta}(\{q_0\}, abb)$ = FECHO-ε($\{r \mid r \in \delta(s, b)$ e $s \in \delta(\{q_0\}, ab)\}$), onde:

$\underline{\delta}(\{q_0\}, ab)$ = FECHO-ε($\{r \mid r \in \delta(s, b)$ e $s \in \delta(\{q_0\}, a)\}$), onde:

$\underline{\delta}(\{q_0\}, a)$ = FECHO-ε($\{r \mid r \in \delta(s, a)$ e $s \in \delta(\{q_0\}, \varepsilon)\}$)

Como $\underline{\delta}(\{q_0\}, \varepsilon)\}$ = FECHO-ε($\{q_0\}$) = $\{q_0, q_1, q_2, q_4\}$ tem-se que:

$\underline{\delta}(\{q_0\}, a) = \{q_0, q_1, q_2, q_4, q_f\}$

$\underline{\delta}(\{q_0\}, ab) = \{q_0, q_1, q_2, q_3, q_4\}$

$\underline{\delta}(\{q_0\}, abb) = \{q_0, q_1, q_2, q_3, q_4, q_f\}$

Como pode ser visto em [HOPCROFT], para cada AFε, é possível construir um AFND equivalente que realiza o mesmo processamento. O Teorema 3 descreve esse fato:

Teorema 3 – Se L é aceito por um AFND com ε-movimentos, então L é aceita por um AFND sem ε-movimentos.

Para realizar a conversão de um AFε em um AFD, eliminando-se o os movimentos vazios e o não determinismo intermediário, basta utilizar-se do Algoritmo 1, visto em [SUDKAMP]:

Algoritmo 1 – Construção de Uma Máquina abstrata Determinística (AFD) a partir de um AF_ε.

Entrada: um AF_ε $M = (\sum, Q, \delta, Q_0, F)$
função transição *t* de M

1.Inicialize Q' com {ε-fechamento(q_0)}
2.repita
2.1.se não existe algum vértice $X \in Q'$ e um símbolo $a \in \sum$ com nenhuma aresta deixando X e nomeado por *a* **então**
2.1.1.seja $Y = \cup\, t\,(q_i,a)$
$q_i \in X$
2.1.2.**se** $Y \notin Q'$ **então** faça $Q' := Q' \cup \{Y\}$
*2.1.3.***adicione** um arco de X para Y nomeado *a*
senão feito:= TRUE
até feito
3. O conjunto de estados aceitos da AFD é $F' = \{X \in Q' \mid X$ contém um elemento $q_i \in F\}$

3.Minimização de um Autômato Finito

O objetivo da minimização é gerar um Autômato Finito equivalente com o menor número de estados possível. A definição de Autômato Mínimo é universalmente aceita e adotada na maioria das soluções práticas. Entretanto, em algumas aplicações especiais, a minimização do número de estados não implica necessariamente no menor custo de implementação.

Um exemplo típico seria o desenho de circuitos eletrônicos, quando pode ser desejável introduzir estados intermediários em determinadas posições (do circuito), de forma a melhorar a eficiência, ou simplesmente facilitar as ligações físicas. Portanto, o algoritmo de minimização nestes casos deve ser modificado, prevendo as variáveis específicas da aplicação.

O Autômato Finito Mínimo é único. Assim, dois Autômatos distintos que aceitam a mesma linguagem ao serem minimizados geram o mesmo Autômato Mínimo, diferenciando-se, eventualmente, na identificação dos estados. Nesse caso, os conjuntos dos estados são isomorfos.

Basicamente, o algoritmo de minimização unifica os estados equivalentes. Dois estados q e p são ditos equivalentes se, e somente se, para qualquer palavra w pertencente **a***, (q,w) e (p,w) resultam simultaneamente em estados finais, ou não-finais. Ou seja, o processamento de uma entrada qualquer a partir de estados equivalentes gera, em qualquer caso, o mesmo resultado aceita/rejeita.

Na definição que segue, lembre-se que, para um dado conjunto A, #A denota o cardinal de A.

Um *Autômato Mínimo* de uma Linguagem Regular L é um Autômato Finito Determinístico $M = (\sum , Q, \delta, q_0, F)$ tal que ACEITA(M) = L e que, para

qualquer outro Autômato Finito Determinístico M' = ($\sum$, Q', δ', q_0', F') tal que ACEITA(M') = L, tem-se que #Q' ≥ #Q.

Um Autômato Finito a ser minimizado deve satisfazer aos seguintes pré-requisitos:

a) Deve ser determinístico;

b) Não pode ter estados inacessíveis (não-atingíveis a partir do estado inicial);

c) A função programa deve ser total (a partir de qualquer estado são previstas transições para todos os símbolos do alfabeto).

Como citado em [HOPCROFT], o Teorema 4 possui, dentre outras conseqüências, a implicação de que existe um AFD mínimo único para todo o conjunto regular.

Teorema 4 – (O Teorema de Myhill-Nerode) – As três seguintes afirmações são equivalentes:

1) O conjunto $L \subseteq \Sigma^*$ é aceito por um Autômato Finito.

2) L é a união de alguma das classes de equivalência de uma certa relação de equivalência invariante de índice finito.

3) Seja R_l uma relação de equivalência definida por: xR_ly, se e somente se, para todo z em Σ^*, xz está em L exatamente quando yz está em L. Então R_l é de índice finito.

Em decorrência disso, como descrito em [SUDKAMP], o Algoritmo 2 é usado para se determinar o AFD mínimo e único que aceita uma linguagem regular L.

Algoritmo 2 – Determinação de estados equivalentes de um AFD

entrada: AFD $M = (\Sigma, Q, \delta, Q_0, F)$

nicialização)

para todos os pares de estados q_i e q_j , i<j **faça**

1.1.D[i,j] := 1

1.2.S[I,j] := 0

fim para

2.**para** todos os pares i,j, i<j, **se** um dos q_i ou q_j é um estado final e o outro é não-final **então** faça D[i,j]:=1

3.para todo o par i,j, i<j, com D[i,j] = 0 **faça**

3.1.**se**, para algum $a \in \Sigma$, $\delta (q_i,a) = q_m$ e $\delta (q_j,a) = q_n$ e D[m,n] =1 ou D[n,m] =1, **então** DIST (i,j)

3.2.**senão** para cada $a \in \Sigma$ **faça**: Caso $\delta (q_i,a) = q_m$ e $\delta (q_j,a) = q_n$

se m<n e [i,j] ≠ [m,n], **então adicione** [i,j] a S[m,n]

senão se m>n **e** [i, j] ≠ [m,n], **então adicione** [i, j] a S[n, m]

fim para

Dist (i, j);

Inicio

D[I, j] := 1

para todo [m, n] ∈ S [i, j], DIST(m,n)

fim.

O Algoritmo 2 faz uso do Teorema 5, que pode ser encontrado em [SUDKAMP], para se determinar o conjunto de estados D [i,j].

Teorema 5 – Os estados q_i e q_j são distintos se, e somente se, D[i, j] = 1 de acordo com o algoritmo 2.

A aplicação do algoritmo de minimização sobre um AFD que obedece aos pré-requisitos de aplicação do algoritmo, como o AFD da Figura 9, resulta em seu AFD mínimo e único ilustrado na Figura 10.

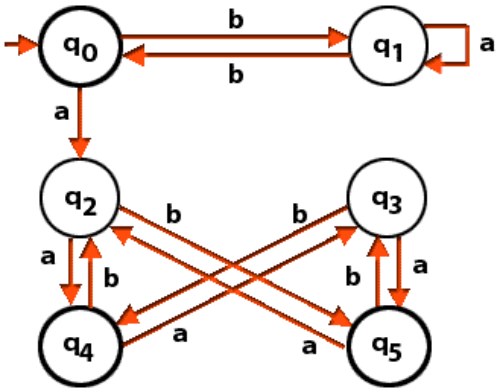

Figura 9: AFD antes da minimização

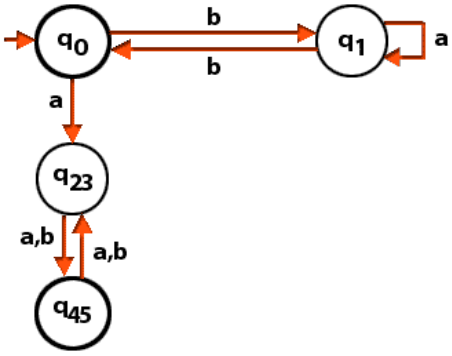

Figura 10: AFD Minimizado

4. Autômato com Pilha (AP ou APD – Autômato *PushDown*)

Analogamente às Linguagens Regulares, a Classe das Linguagens Livres do Contexto pode ser associada a um formalismo do tipo Autômato, denominado Autômato com Pilha.

O Autômato com Pilha é análogo ao Autômato Finito, incluindo uma pilha como memória auxiliar e a facilidade de não-determinismo. A pilha é independente da fita de entrada e não possui limite máximo de tamanho ("infinita"). Estruturalmente, sua principal característica é que o último símbolo gravado é o primeiro a ser lido, como ilustrado na Figura 11. A *base* de uma pilha é fixa e define o seu início. O *topo* é variável e define a posição do último símbolo gravado.

A facilidade de não-determinismo é importante e necessária, pois aumenta o poder computacional dos Autômatos com Pilha, permitindo reconhecer exatamente a Classe das Linguagens Livres do Contexto. Por exemplo, o reconhecimento da linguagem:

$L=\{ww^r \mid w \text{ é palavra sobre } \{a, b\}\}$

só é possível por um Autômato com Pilha Não-Determinístico.

Embora o poder computacional do Autômato com Pilha seja muito superior ao do Autômato Finito, ainda é restrito, não sendo possível reconhecer linguagens simples, como, por exemplo:

$L=\{ww \mid w \text{ é palavra sobre } \{a, b\}\}$ ou $\{a^n b^n c^n \; n \geq 0\}$

Um resultado interessante mostrado adiante é que qualquer Linguagem Livre do Contexto pode ser reconhecida por um Autômato com Pilha com

somente um estado (ou três estados, dependendo da definição). Isso significa que a estrutura de pilha é suficiente como única memória, não sendo necessário usar os estados para "memorizar" informações passadas. Ou seja, a estrutura de estados no Autômato com Pilha poderia ser excluída sem reduzir o poder computacional.

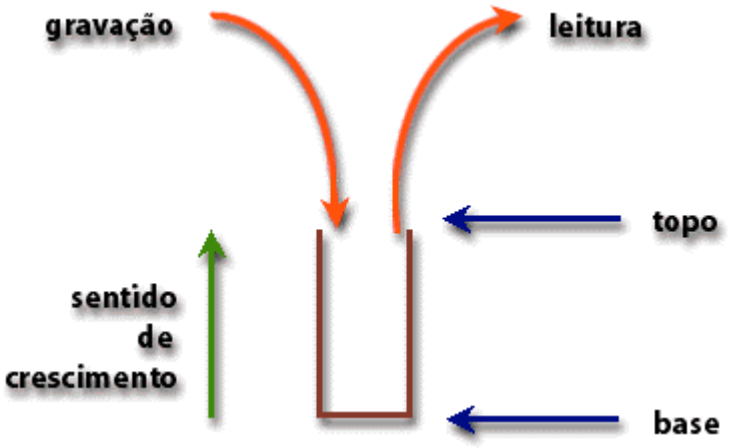

Figura 11: A pilha de um APD

O modelo Autômato com Pilha possui duas definições universalmente aceitas que diferem no critério de parada do Autômato, como segue:

- o valor inicial da pilha é vazio e o Autômato pára aceitando ao atingir um estado final;
- a pilha contém, inicialmente, um símbolo especial denominado símbolo inicial da pilha. Não existem estados finais e o Autômato pára aceitando quando a pilha estiver vazia.

As duas definições são equivalentes (possuem o mesmo poder computacional), sendo fácil modificar um Autômato com Pilha para satisfazer a outra definição. Nesta publicação, é adotada a com estados finais.

Um Autômato com Pilha ou Autômato com Pilha Não-Determinístico é composto, basicamente, por quatro partes, como segue:

a) ***Fita***. Análoga à do Autômato Finito;

b) ***Pilha***. Memória auxiliar que pode ser usada livremente para leitura e gravação;

c) ***Unidade de Controle***. Reflete o estado corrente da máquina. Possui uma cabeça de fita e uma cabeça de pilha;

d) ***Programa ou Função de Transição***. Comanda a leitura da fita, leitura e gravação da pilha e define o estado da máquina.

A pilha é dividida em células, armazenando, cada uma, um símbolo do alfabeto auxiliar (pode ser igual ao alfabeto de entrada). Em uma estrutura do tipo pilha, a leitura ou gravação é sempre na mesma extremidade, denominada topo.

A pilha não possui tamanho fixo e nem máximo, sendo seu tamanho corrente igual ao tamanho da palavra armazenada. Seu valor inicial é vazio.

A unidade de controle possui um número finito e predefinido de estados. Possui uma cabeça de fita e uma cabeça de pilha, como segue:

a)***Cabeça da Fita***. Unidade de leitura a qual acessa uma célula da fita de cada vez e movimenta-se exclusivamente para a direita. É possível testar se a entrada foi completamente lida;

b)***Cabeça da Pilha***. Unidade de leitura e gravação a qual move para a esquerda (ou "para cima") ao gravar e para a direita (ou "para baixo") ao ler um símbolo. Acessa um símbolo de cada vez, estando sempre posicionada no topo. A leitura exclui o símbolo lido.

c)É possível testar se a pilha está vazia. Em uma mesma operação de gravação, é possível armazenar uma palavra composta por mais de um símbolo. Neste caso, o símbolo do topo é o mais à esquerda da palavra gravada.

O programa é uma função parcial que, dependendo do estado corrente, símbolo lido da fita e símbolo lido da pilha, determina o novo estado e a palavra a ser gravada (na pilha). Possui a facilidade de movimento vazio (análoga à do Autômato Finito), permitindo mudar de estado sem ler da fita.

4.1. Definição de Autômato com Pilha

Segundo [MENEZES], um Autômato *PushDown* (APD), Autômato com Pilha Não-Determinístico (APN) ou simplesmente Autômato com Pilha (AP) M é uma 6-upla:

$$M = (\sum, Q, \delta, q_0, F, V)$$

onde:

$\sum$ = *alfabeto de símbolos de entrada*;

Q = *conjunto de estados* possíveis do Autômato o qual é finito;

δ = *função programa* ou de *função de transição*:

$$\delta: Q \times (\sum \cup \{ \varepsilon, ? \}) \times (V \cup \{ \varepsilon , ? \}) \rightarrow 2^{Q \times V^*}$$

a qual é uma função parcial;

q_0 = *estado inicial* do Autômato tal que q0 é elemento de Q;

F = *conjunto de estados finais* tal que F está contido em Q;

V= *alfabeto auxiliar* ou *alfabeto da pilha*.

As seguintes características da função programa devem ser consideradas:

- a função pode não ser total, ou seja, indefinida para alguns argumentos do conjunto de partida; a omissão do parâmetro de leitura, representada por "?", indica o teste de pilha vazia ou toda palavra de entrada lida;

- símbolo na leitura indica a facilidade de movimento vazio da fita ou da pilha (o Autômato não lê nem move a cabeça). Note-se que, para o movimento ser considerado não-determinístico, é suficiente que o movimento seja vazio na fita;
- símbolo na gravação indica que nenhuma gravação é realizada na pilha (e não move a cabeça).

Por exemplo, $\delta(p, ?, \varepsilon) = \{(q, \varepsilon)\}$ indica que no estado **p** se a entrada foi completamente lida, não lê da pilha, assume o estado **q** e não grava na pilha. A função programa pode ser representada como um grafo direto, como ilustrado na Figura 12.

O processamento de um Autômato com Pilha, para uma palavra de entrada **w**, consiste na sucessiva aplicação da função programa para cada símbolo de **w** (da esquerda para a direita) até ocorrer uma condição de parada.

No conjunto de exemplos para essa classe de Autômatos, é possível encontrar uma máquina que nunca atinja uma condição de parada. Nesse caso, fica processando indefinidamente (ciclo ou *loop* infinito). Um exemplo simples de ciclo infinito é um programa que empilha e desempilha um mesmo símbolo indefinidamente, sem ler da fita.

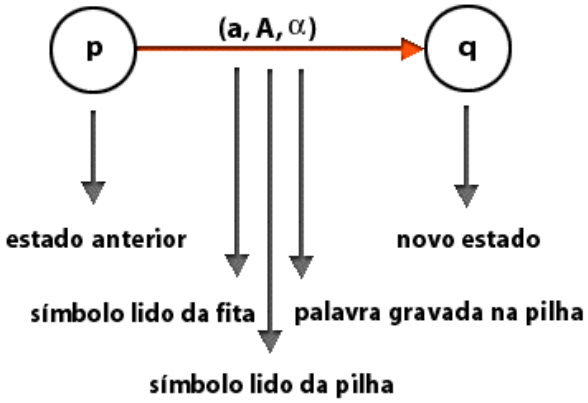

Figura 12: APD

Um Autômato com Pilha pode parar aceitando ou rejeitando a entrada ou ficar em *loop* infinito, como segue:

a) Um dos caminhos alternativos assume um estado final: o Autômato pára e a palavra é aceita;

b) Todos os caminhos alternativos rejeitam a entrada: o Autômato pára e a palavra é rejeitada;

c) Pelo menos um caminho alternativo está em *loop* infinito e os demais rejeitam (ou também estão em *loop* infinito): o Autômato está em *loop* infinito.

As seguintes notações são adotadas (suponha que **M** é um Autômato com Pilha):

a) **ACEITA (M)** ou **L(M)**: conjunto de todas as palavras de $\sum^*$ aceitas por **M**;

b) **REJEITA (M)**: conjunto de todas as palavras de $\boldsymbol{\sum^*}$ rejeitadas por **M**;

c) **LOOP (M)**: conjunto de todas as palavras de $\boldsymbol{\sum^*}$ para as quais **M** fica processando indefinidamente.

As seguintes afirmações são verdadeiras:

- $ACEITA(M) \cap REJEITA(M) \cap LOOP(M) = \varnothing$
- $ACEITA(M) \cup REJEITA(M) \cup LOOP(M) = \sum^*$
- o complemento de:
  - ACEITA(M) é $REJEITA(M) \cup LOOP(M)$
  - REJEITA(M) é $ACEITA(M) \cup LOOP(M)$
  - LOOP(M) é $ACEITA(M) \cup REJEITA(M)$

Considere a seguinte linguagem:

$$L = \{a^n b^n \mid n \geq 0\}$$

O Autômato com Pilha M = ({a, b}, {q_0, q_1, q_f}, δ, q_0, {q_f}, {B}), onde δ é como abaixo, é tal que ACEITA(M) = L:

$$\delta (q_0, a, \varepsilon) = \{(q_0, B)\}$$

$$\delta (q_0, b, B) = \{(q_1, \varepsilon)\}$$

$$\delta (q_0, ?, ?) = \{(q_f, \varepsilon)\}$$

$$\delta (q_1, b, B) = \{(q_1, \varepsilon)\}$$

$$\delta (q_1, ?, ?) = \{(q_f, \varepsilon)\}$$

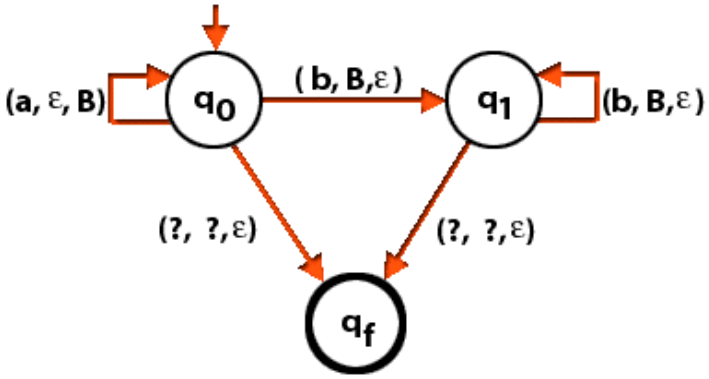

Figura 13: Exemplo de um APD

O Autômato pode ser representado pelo grafo ilustrado na Figura 13. Note-se que o Autômato é determinístico. No estado q_0, para cada símbolo a lido da fita é armazenado um símbolo B na pilha. No estado q_1, é realizado um batimento, verificando se para cada símbolo b da fita, existe um correspondente B na pilha. O algoritmo somente aceita se ao terminar de ler toda a palavra de entrada a pilha estiver vazia.

5. Máquina de Turing

A Máquina de Turing para ser considerada como um modelo formal de procedimento efetivo, algoritmo ou função computável deve satisfazer às seguintes propriedades, entre outras:

a) A descrição do algoritmo deve ser finita;

b) Deve consistir de passos:

- discretos;
- executáveis mecanicamente;
- em um tempo finito.

O modelo proposto por Alan Turing em 1936, conhecido como Máquina de Turing, consiste basicamente de 3 partes:

a) **Fita**. Usada simultaneamente como dispositivo de entrada, saída e memória de trabalho;

b) **Unidade de Controle**. Reflete o estado corrente da máquina. Possui uma unidade de leitura e gravação (cabeça da fita) a qual acessa uma célula da fita de cada vez e movimenta-se para a esquerda ou direita;

c) **Programa ou Função de Transição**. Função que comanda as leituras e gravações, o sentido de movimento da cabeça e define o estado da máquina.

A fita é finita à esquerda e infinita à direita, sendo dividida em células, onde cada uma armazena um símbolo. Os símbolos podem pertencer ao alfabeto de entrada, ao alfabeto auxiliar ou ainda, ser "branco" ou "marcador de início de fita".

Inicialmente, a palavra a ser processada (ou seja, a informação de entrada para a máquina) ocupa as células mais à esquerda, após o marcador de início de fita, ficando as demais com "branco", como ilustrado na Figura 14,

onde β e ☸ representam "branco" e "marcador de início de fita", respectivamente.

A unidade de controle possui um número finito e predefinido de estados. A **cabeça da fita** lê o símbolo de uma célula de cada vez e grava um novo símbolo. Após a leitura/gravação, a cabeça move uma célula para a direita ou esquerda. O símbolo gravado e o sentido do movimento são definidos pelo programa.

O programa é uma função que, dependendo do estado corrente da máquina e do símbolo lido, determina o símbolo a ser gravado, o sentido do movimento da cabeça e o novo estado.

5.1. Definição

Como descrito em [MENEZES], uma Máquina de Turing é uma 8-upla:

$$M = (\sum, Q, \delta, q_0, F, V, \beta, ☸)$$

onde:

$\sum$ = alfabeto de símbolos de entrada;

Q = conjunto de estados possíveis da máquina o qual é finito;

δ = programa ou função de transição:

$$\delta : Q \times (\sum \cup V \cup \{ \beta , ☸\}) \rightarrow Q \times (\sum \cup V \cup \{\beta , ☸\}) \times \{E, D\}$$

a qual é uma função parcial;

q_0 = estado inicial da máquina tal que q_0 é elemento de Q;

F = conjunto de estados finais tal que F está contido em Q;

V = alfabeto auxiliar (pode ser vazio);

β = símbolo especial branco;

☸= símbolo ou marcador de início da fita.

O símbolo de início de fita sempre ocorre exatamente uma vez e na célula mais à esquerda da fita, auxiliando na identificação de que a cabeça da fita se encontra na célula mais à esquerda da fita.

A função programa, considera o estado corrente e o símbolo lido da fita para determinar o novo estado, o símbolo a ser gravado e sentido de movimento da cabeça onde esquerda e direita são representados por **E** e **D**, respectivamente. A função programa pode ser interpretada como um grafo finito direto, como ilustrado na Figura 14. Neste caso, os estados inicial e finais são representados como nos Autômatos Finitos.

O processamento de uma Máquina de Turing M = ($\sum$, Q, δ, q_0, F, V,β,☸), para uma palavra de entrada w, consiste na sucessiva aplicação da função programa a partir do estado inicial q_0 e da cabeça posicionada na célula mais à esquerda da fita até ocorrer uma condição de parada. O processamento de **M** para a entrada **w**, pode parar ou ficar em loop infinito. A parada pode ser de duas maneiras: aceitando ou rejeitando a entrada **w**. As condições de parada são as seguintes:

a) A máquina assume um estado final: a máquina pára e a palavra de entrada é aceita;
b) A função programa é indefinida para o argumento (símbolo lido e estado corrente): a máquina pára e a palavra de entrada é rejeitada;
c) O argumento corrente da função programa define um movimento à esquerda e a cabeça da fita já se encontra na célula mais à esquerda: a máquina pára e a palavra de entrada é rejeitada.

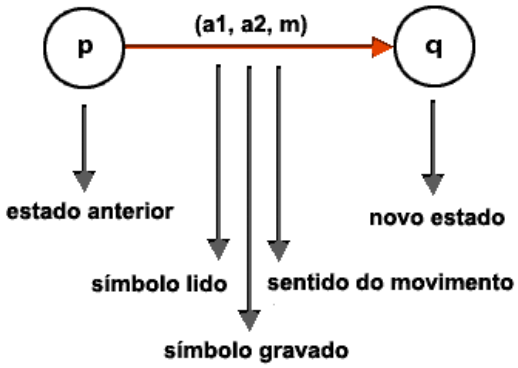

Figura 14: Representação da função programa como um grafo

Para definir formalmente o comportamento de uma Máquina de Turing, é necessário estender a definição da função programa usando como argumento um estado e uma palavra.

As seguintes notações são adotadas, as quais são análogas às usadas para Autômatos com Pilha (suponha que **M** é uma Máquina de Turing):

a)**ACEITA(M)** ou **L(M)**: conjunto de todas as palavras de Σ^* aceitas por **M**;

b)**REJEITA(M)**: conjunto de todas as palavras de Σ^* rejeitadas por **M**;

c)**LOOP(M)**: conjunto de todas as palavras de Σ^* para as quais **M** fica processando indefinidamente.

Da mesma forma, as seguintes afirmações são verdadeiras:

- $ACEITA(M) \cap REJEITA(M) \cap LOOP(M) = \varnothing$
- $ACEITA(M) \cup REJEITA(M) \cup LOOP(M) = \Sigma^*$
- O complemento de:
  - ACEITA(M) é $REJEITA(M) \cup LOOP(M)$
  - REJEITA(M) é $ACEITA(M) \cup LOOP(M)$

– LOOP(M) é ACEITA(M) ∪ REJEITA(M)

Considere a linguagem:

$$L = \{a^n b^n \mid n \geq 0\}$$

A Máquina de Turing:

$$M = (\{a, b\}, \{q_0, q_1, q_2, q_3, q_4\}, \delta, q_0, \{q_4\}, \{A, B\}, \beta, ❁)$$

ilustrada na Figura 15, é tal que:

ACEITA(M) = L

REJEITA(M) = ∑* - L

e, portanto, LOOP(M_1) = ∅.

O algoritmo apresentado reconhece o primeiro símbolo **a**, o qual é marcado como **A**, e movimenta a cabeça da fita à direita, procurando o **b** correspondente, o qual é marcado como **B**. Esse ciclo é repetido sucessivamente até identificar para cada **a** o seu correspondente **b**.

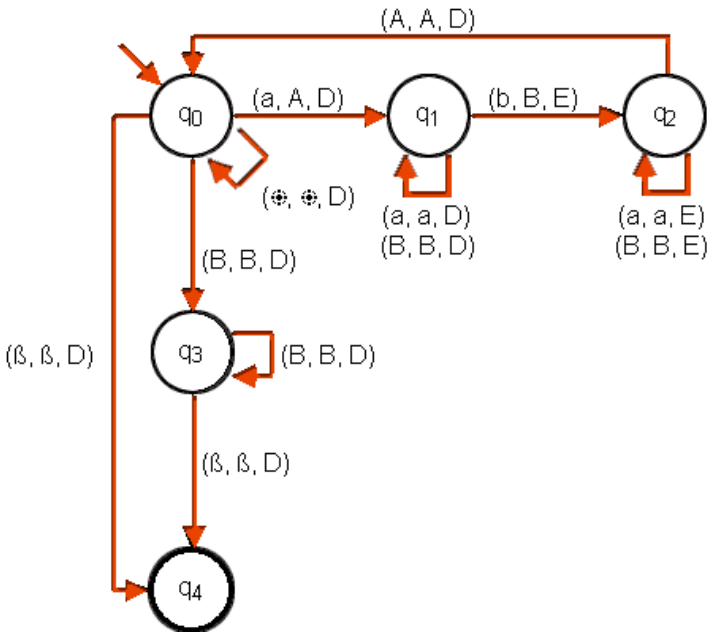

Figura 15: Máquina de Turing

Adicionalmente, o algoritmo garante que qualquer outra palavra que não esteja na forma “ab” é rejeitada. Note-se que o símbolo de início de fita não tem influência na solução proposta.

6. Desenvolvimento da GAM

Os conjuntos finitos de processos, sumarizados como sistemas de estados finitos, acabam surgindo de uma forma natural em diversas áreas da computação. Esse conjunto de máquinas abstratas firmam-se como uma classe estruturalmente rica e potencialmente aplicável em diversas áreas da computação. Máquinas de venda de latas de refrigerantes, máquinas de venda de jornais e mecanismos de controle de elevadores são bons exemplos de aplicabilidade de sistemas de estados finitos.

Sob esse aspecto, a teoria de Autômatos Finitos apresenta-se como um conjunto de ferramentas fundamentais que auxiliam o desenvolvimento desses e de outros muitos sistemas. Durante o estudo de máquinas abstratas, fica-se evidente que existe uma necessidade emergente de se possuir um ambiente capaz de apresentar recursos que auxiliem a compreensão da teoria que envolve o reconhecimento das linguagens reconhecidas pelas máquinas.

A *Ginux Abstract Machine* apresenta-se essencialmente como uma ferramenta para uso didático. Sob licença GPL, a GAM deverá ser utilizada no aprendizado e estudo dos Autômatos.

A ferramenta para o estudos de máquinas abstratas está configurada para desempenhar basicamente, as seguintes funções:

- Construção e reconhecimento de AFDs (Autômatos Finitos Determinísticos)
- Construção e reconhecimento de AFNDs (Autômatos Finitos Não Determinísticos)
- Conversões da classe não-determinística para a determinística
- Minimização de AFDs

- Construção de APDs (Autômatos *PushDown*)
- Construção de MTs (Máquinas de Turing)

O ambiente gráfico utilizado foi o Gnome 2.2.0.2-4, em um Sistema Operacional GNU/Linux Red Hat 9.0.

Foram usados pacotes fontes no padrão "tar.gz" e construídos segundo às regras de compilação padrão, como pode ser visto no documento INSTALL contido em todos os pacotes.

Foi necessária a instalação da seguinte lista em seqüência de pacotes de desenvolvimento para deixar o pacote de desenvolvimento Glademm funcional: libsigc++-1.2.2 , pkgconfig-0.15.0, glib-2.2.1, atk-1.2.0, tiff-v3.5.7, jpeg-6b, libpng-1.2.5, gtk+-2.2.0, gtkmm-2.1.0, libxml2-2.5.7, intltool-0.26, libxslt-1.0.9, docbookx412, scrollkeeper-0.3.12, glademm-2.0.0, glade-2.0.0.

Todos os pacotes podem ser encontrados em [WELCOME], e versões superiores eventualmente podem ser testadas e usadas.

6.1. Uma Breve apresentação do Glade e seus componentes

Para desenvolver a ferramenta de estudo de máquinas abstratas, foi utilizado essencialmente o conjunto de quatro softwares-ferramentas interdependentes:

- Gtkmm-2.1.0
- Glademm-2.0.0
- GTK+
- Glade-2.0.0

Os dois primeiros pacotes são responsáveis por gerar código em C++ para a futura compilação que será feita pelo GCC e G++ versão 3.x.

O pacote *front-end* glade é o grande responsável pela interface de todo o projeto. Glade é um construtor de interfaces de usuários *free* para o uso do GTK+ e GNOME. A versão utilizada está sob a licença *GNU General Public License* (GPL).

O GTK+ é uma ferramenta multiplataforma para a criação da interface de usuário. A ferramenta é composta por um conjunto largo de componentes (*widgets*), sendo adequada desde o desenvolvimento de projetos pequenos até ambientes completos de aplicações mais sofisticadas.

O GTK+ está também sob a licença GPL. Entretanto, termos da licença para o GTK+, que fazem parte da GNU LGPL, permitem que o GTK+ seja usado por todos desenvolvedores, incluindo aqueles que se utilizam da ferramenta para gerar *softwares* proprietários, sem que haja cobrança de *royalties* ou complementos de licença de uso.

O GTK+ baseado em três bibliotecas de desenvolvimento principais:

- GLIB : biblioteca do núcleo de baixo nível que forma a base do GTK+ e do GNOME. Ela fornece as estruturas de dados à linguagem principal C, suporte à portabilidade e interfaces para as funcionalidades em tempo de execução, tais como carga dinâmica, *threads*, eventos em *loop*, e sistema de objetos (no caso de C++, por exemplo).

- Pango: biblioteca usada na renderização de textos e na produção do *layout* do projeto.

- ATK: fornece um conjunto de interfaces para a acessibilidade ao projeto em si. Assim sendo, uma aplicação ou kit de ferramentas podem ser usados em conjunto com leitores de tela, magnificadores ou dispositivos alternativos de entrada.

O GTK+ foi desenvolvido para suportar diferentes linguagens de programação. A principal delas é a linguagem C. Graças ao pacote GTKmm, a linguagem de programação C++, usada no desenvolvimento da GAM, também é suportada.

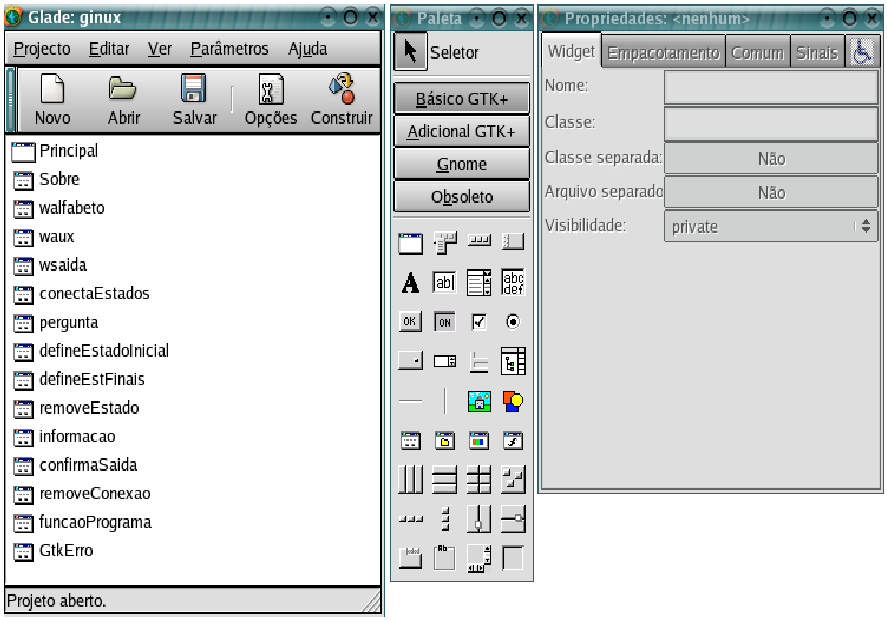

Figura 16 - Ambiente Glade

A Figura 16 apresenta o ambiente Glade, basicamente composto por três janelas padrão: a principal, a barra de ferramentas e a janela de propriedades de *widgets*.

A Figura 17 apresenta a janela principal do Glade. Ela lista todas as janelas principais e janelas diálogos que fazem parte do projeto em desenvolvimento. Ao dar um clique duplo em um dos itens, a janela correspondente é carregada na tela.

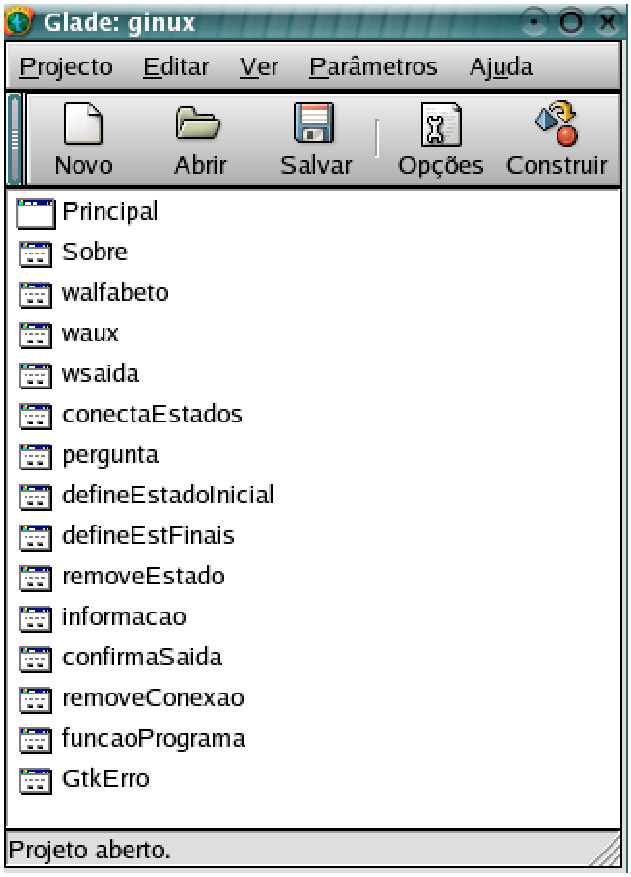

Figura 17 - Glade: Janela Principal

O menu e a barra de ferramentas habilita o desenvolvedor a criar novos projetos, carregá-los e salvá-los.

Uma opção importante é a de gerar o código fonte de forma automática na linguagem especificada através da opção “Construir”. O menu e a barra de ferramentas podem ser vistos na Figura 18.

Figura 18 - Barra de Ferramentas do Glade

A segunda e mais importante janela do Glade é a paleta de widgets (Figura 19). Nelas estão ícones que representam todos os componentes suportados pelo Glade para o desenvolvimento de projetos.

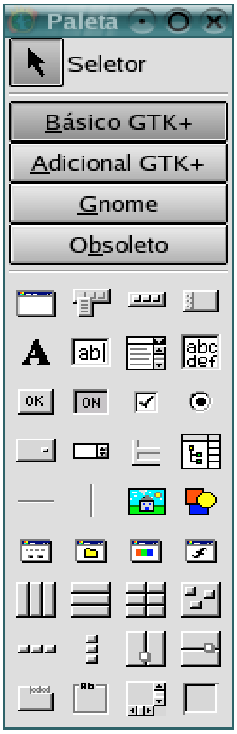

Figura 19 - Glade: Paleta de Widgets (componentes)

Para adicionar janelas e diálogos, por exemplo, basta simplesmente selecionar o ícone correspondente na paleta, determinando-se somente a posição inicial na tela, como descreve a Figura 20.

Figura 20 - Exemplos de como adicionar componentes no Glade

A última janela importante do Glade é a de edição de propriedades. Essa janela permite ao desenvolvedor alterar as propriedades dos componentes (*widgets*) do projeto, tais como o texto, tamanho, largura e posição.

A página de sinais fornecem a condição de se controlar os manipuladores de sinais para os componentes. Assim sendo, pode-se especificar a função que será invocada ao se utilizar um *widget*, como por exemplo, ao se apertar um botão.

A ferramenta Glade gera basicamente dois tipos de arquivos separados em duas classes:

- A primeira classe encapsula o código responsável por gerar toda a interface do projeto. O produto resultante dessa etapa são arquivos com nomenclatura "_glade", com extensões .hh e .cc (ex.: teste_glade.hh e teste_glade.cc). Normalmente somente o Glade modifica tais arquivos.
- A segunda classe apresenta arquivos que serão editáveis pelo desenvolvedor. Os arquivos resultantes apresentam nomes equivalentes aos nomes dos arquivos gerados na primeira classe e apresentam também extensões .hh e .cc.

Uma característica interessante do Glade é que ao gerar todo o código da interface, a ferramenta prepara a estrutura de diretório contendo todo o conteúdo do projeto, de tal forma que o aplicativo *make* se encarrega de compilar todo o projeto através de um *Makefile* gerado de forma automática.

Informações complementares sobre o Glade, GTKmm e seus componentes podem sem encontradas em [CHAPLIN] e [CUMMING].

Ressalta-se também que, ao longo do desenvolvimento da ferramenta GAM, foram usadas as seguintes ferramentas de apoio: gimp 1.2.3, make,

autoconf, dia 0.90, pkgconfig-0.15.0, atk-1.2.0, gtk+-2.2.0, gtkmm-2.1.0, intltool-0.26, glademm-2.0.0, glade-2.0.0, checkinstall, vim.

6.2. Modelagem da Ginux Abstract Machine

A Figura 21 ilustra a organização da ferramenta GAM. A partir da classe Principal, chamam-se janelas de diálogo que representam as demais classes e suas funcionalidades.

A classe Quadro faz parte da classe Principal como uma classe componente usada na composição gráfica das máquinas.

É na classe principal que se encontram todas as funções da GAM. Assim como pode ser percebido também na Figura 21, todas as demais classes resguardam funções específicas que são inicializadas a partir dessa classe.

A Figura 22 representa a complementação da classe Principal, que, a partir dela, chamam-se as classes pergunta, salvaProjeto e carregaProjeto.

Destacam-se ainda as classes salvaProjeto e carregaProjeto, visto que são derivadas da classe gtkmm/fileselection.h. A partir de qualquer uma das classes, chama-se a classe informação, contendo a um texto informando sobre a conclusão da operação de arquivo.

A Figura 23 representa a classe ComboEstado, cujo componente de interface foi construido manualmente. A ComboEstado será usada como componente das classes removeEstado, removeConexao, conectaEstados, conectaEstadosAPD e conectaEstadosMT.

Ressalta-se que na classe ComboEstado, os estados recém-criados de uma máquina abstrata são determinados dinamicamente.

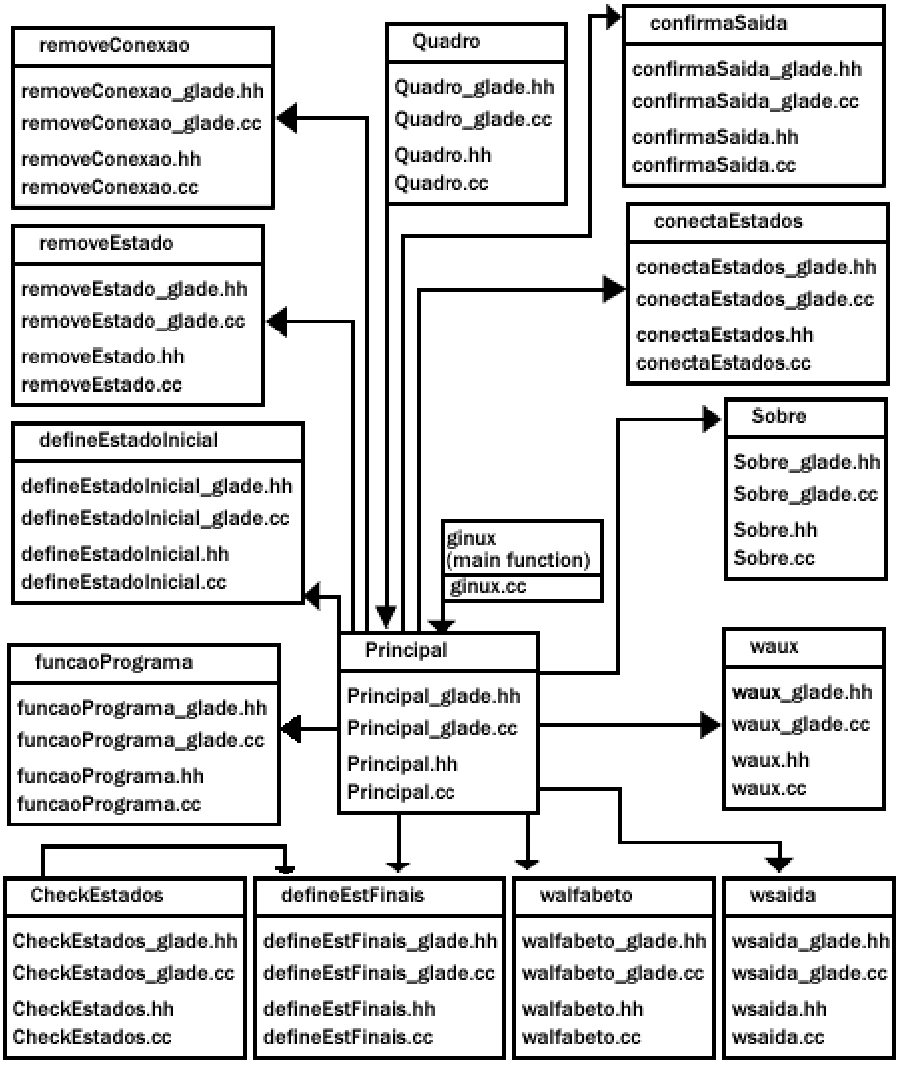

Figura 21 - Modelagem de Arquivos e de Classes da GAM

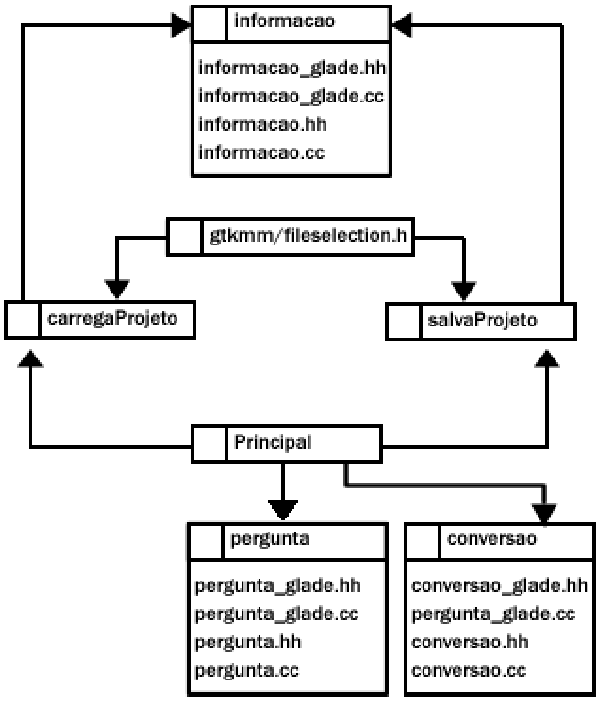

Figura 22 - Modelagem de Arquivos e de Classes da GAM

A classe simboloAF irá ser usada pelas classes conectaEstados e conectaEstadosAPD para se determinar quais os símbolos poderão ser usados pelas conexões do Autômato.

Similarmente, as classes simboloAPD e simboloMT determinam os símbolos auxiliares (para uso da pilha) e símbolos de saída (para a Máquina de Turing) respectivamente.

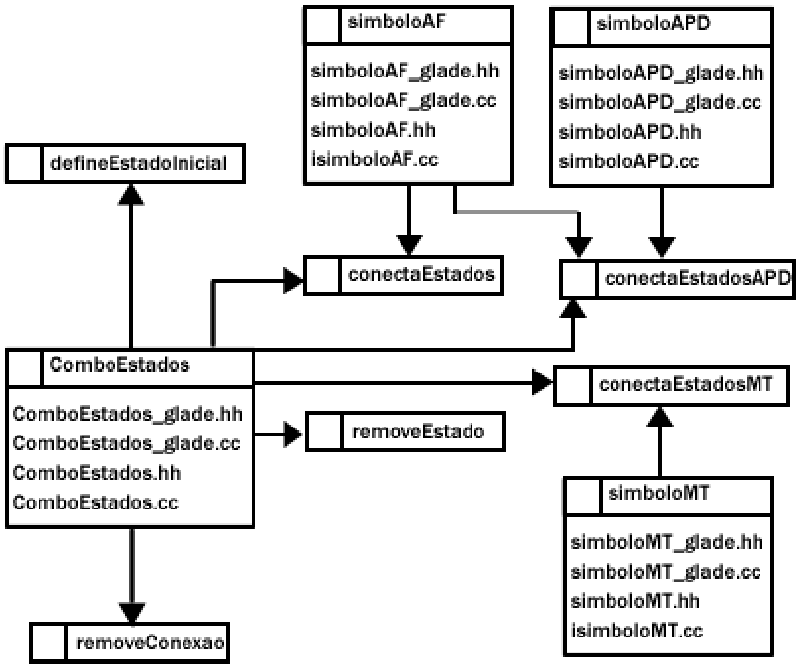

Figura 23 - Modelagem de Arquivos e de Classes da GAM

A Figura 24 contém a informação de que a classe GtkErro, que guarda todas as mensagens de operações inválidas do projeto, será usada por todas as classes identificadas na figura.

Ressalta-se ainda que a classe constantes.hh contém todas as constantes usadas no projeto.

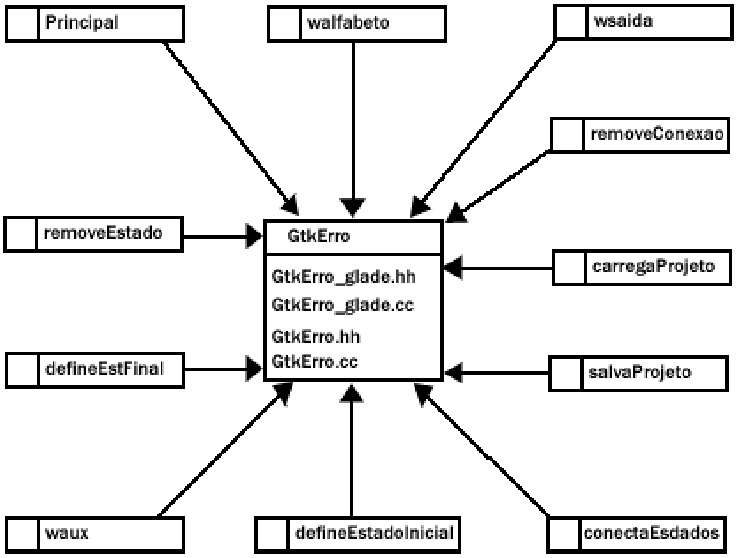

Figura 24 - Modelagem de Arquivos e de Classes da GAM

A Figura 25 identifica a classe automato como a classe lógica essencial do programa. Todas as demais classes a ela relacionadas, fazem uso de seus objetos e métodos públicos.

É também importante observar que as classes Principal e funcaoPrograma fazem uso do *header file* funcoes.hh que contém todas as operações necessárias para manipular os Autômatos em atividades específicas, tais como calcular a função programa e copiar Autômatos, por exemplo.

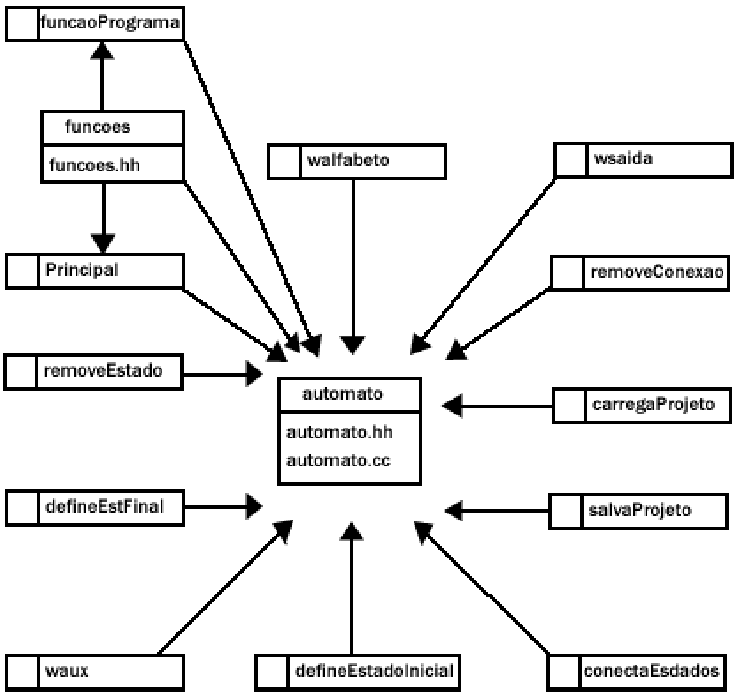

Figura 25 - Modelagem de Arquivos e de Classes da GAM

7. Uso da Ginux Abstract Machine (GAM) Versão 1.1.0-3c

A ferramenta Ginux Abstract Machine é bastante simples e apresenta-se intuitivamente fácil de se manipular.

Essencialmente, a ferramenta deverá ser usada em ambientes de estudo das linguagens formais e das máquinas abstratas de uma forma geral. Portanto, funcionalmente indica-se o uso da ferramenta em laboratórios de estudo da Teoria da Computação e de construção Autômatos.

O Objetivo principal da ferramenta é servir como ambiente de estudo de linguagens formais e Autômatos.

7.1. Descrição

7.1.1. Janela Principal

Ao abrir a ferramenta, o usuário irá encontrar a janela principal do projeto, que está dividida em três blocos principais: o menu principal, a janela de desenhos, a barra de análise de entradas e a barra de ferramentas.

a)O menu principal está subdividido em quatro itens, como ilustra a Figura 26:

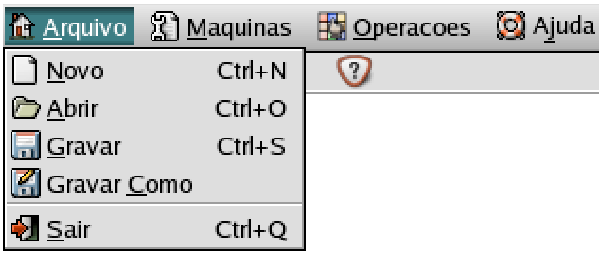

Figura 26 - GAM: A janela Principal

– **Menu Máquinas**: a Figura 27 apresenta as opções de simulação das máquinas abstratas principais que são os AFDs, AFNDs, APDs e MTs.

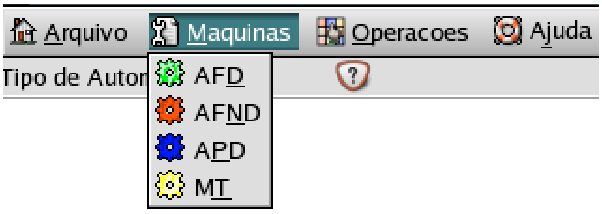

Figura 27 - GAM: Menu Principal (Máquinas)

– **Menu Arquivo**: a Figura 28 apresenta as opções de criar novo projeto, carregar projeto, salvar projeto, salvar projeto como e sair da ferramenta.

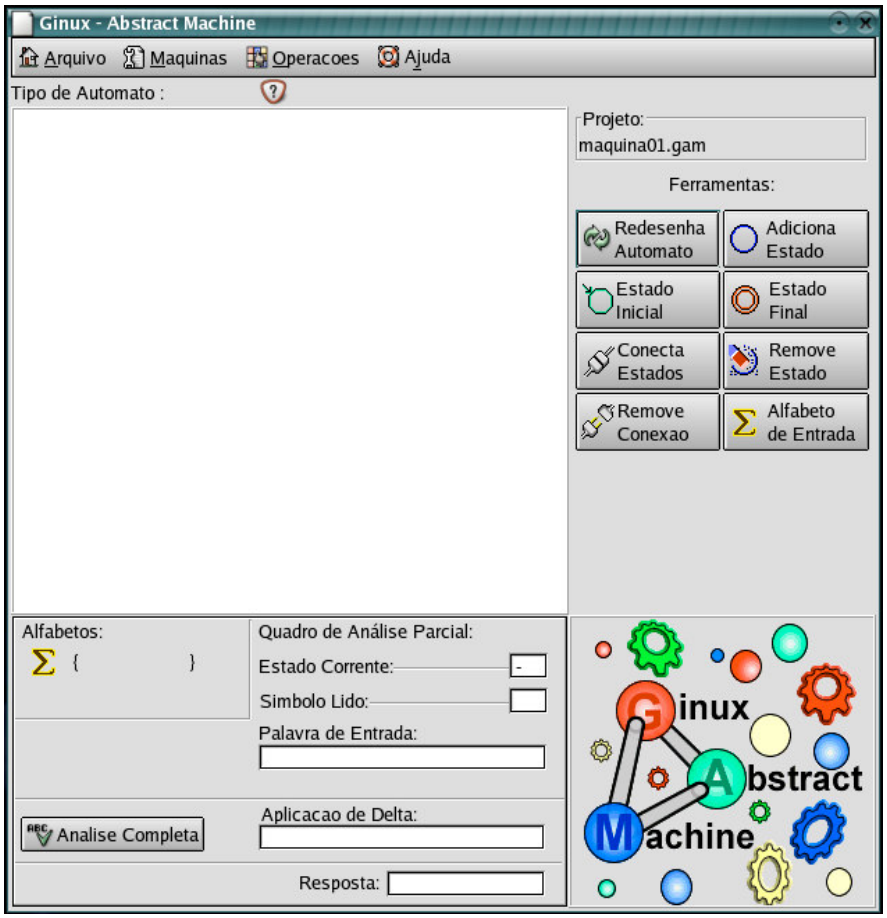

Figura 28 - GAM: Menu Principal (Arquivo)

– **Menu Operações**: a Figura 29 apresenta as funções de minimização de AFDs, conversão de AFNDs para AFDs, visualização da função programa e restauração das máquinas (volta das operações para a máquina inicial).

Figura 29 - GAM: Menu Principal (Operações)

- **Menu Ajuda**: a Figura 30 ilustra o menu de ajuda com informações iniciais sobre a Ferramenta. A Figura 31 ilustra a janela de diálogo contendo informações sobre a versão da GAM.

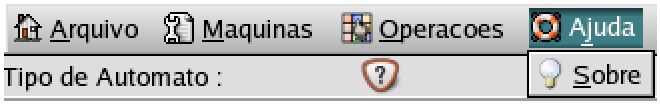

Figura 30 - GAM: Menu Principal (Ajuda)

b) A janela de desenhos é uma área dinâmica que representará as máquinas abstratas em sua forma diagramática.

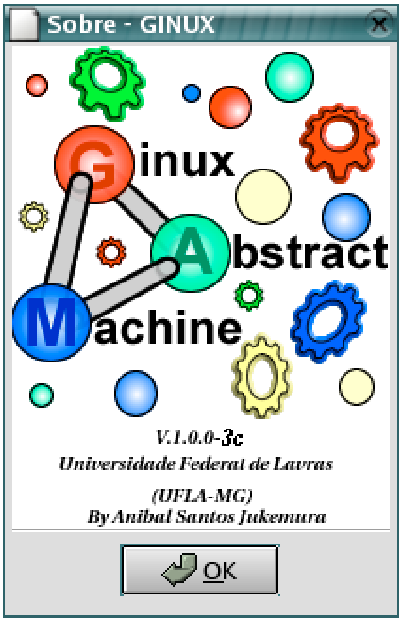

Figura 31 - Tela de Informações sobre a GAM

b)A barra de análise de entradas se subdivide essencialmente em dois grupos:

– Análise passo-a-passo: nesse quadro de análise, como ilustra a Figura 32, o usuário poderá percorrer a palavra de entrada um símbolo por vez, visualizando o funcionamento do Autômato pausadamente.

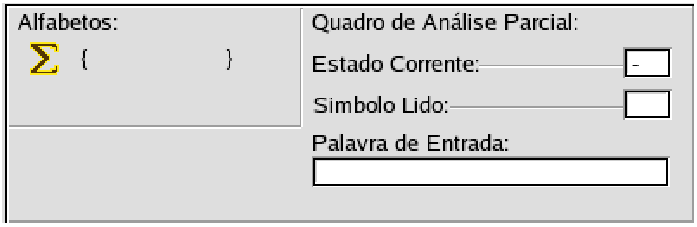

Figura 32 - Quadro de Análise Passo a Passo

– Análise: nesse quadro, ao clicar no botão de análise, o usuário terá como resposta, se a palavra de entrada foi aceita ou não. Acidionalmente, o usuário receberá uma apresentação do resultado da aplicação da função programa, seja ela do modelo determinístico ou não. A Figura 33 ilustra o quadro de análise rápida (completa):

Figura 33 - Quadro de Análise Rápida (Completa)

d) Barra de ferramentas: essa é a parte essencial do projeto, visto que todas as operações de construção da máquina estão nela definidas. As operações implementadas na barra de ferramentas são:

– Redesenha Autômato (Figura 34): para evitar problemas de "sumiço do desenho" (*refreshing*) o usuário pode restaurar a área de desenho sempre que desejar, bastando usar essa opção.

Figura 34 - Botão Redesenha Autômato

– Adiciona Estado (Figura 35): a versão da GAM permite ao usuário inserir até 10 (dez) estados. Apesar da opção permitir a inserção

desses elementos, essa versão ainda não fornece a liberdade ao usuário de movimentar os estados.

–

Figura 35 - Botão de Adição Estado

– Estado Inicial (Figura 36): permite ao usuário indicar qual estado será usado como estado inicial.

–

Figura 36 - Botão de definição de Estado Inicial

– Estado Final (Figura 37): permite que o usuário possa definir quais estados serão finais ou não.

Figura 37 - Botão de definição de Estados Finais

– Conecta Estados (Figura 38): permite a conexão dos estados previamente inseridos, seguindo as restrições de cada classe de Autômatos.

Figura 38 - Botão de Conexão de Estados

– Remove Estados (Figura 39): permite a remoção dos estados não desejados, juntamente com as conexões relacionadas com o mesmo.

–

Figura 39 - Botão de Remoção de Estados

– Remove Conexão (Figura 40): permite ao usuário remover os estados desnecessários.

Figura 40 - Botão de Remoção de Conexões

– Alfabeto de entrada (Figura 41): define os símbolos do Autômato que serão usados na construção da função programa de cada máquina.

–

Figura 41 - Botão de Definição dos Símbolos de Entrada

– Alfabeto de Saída (Figura 42): para a classe das MTs (Máquinas de Turing), essa opção define o alfabeto de saída após a aplicação de um movimento.

Figura 42 - Botão de Definição de Alfabeto de Saída para as Mts

- Alfabeto Auxiliar (Figura 43): para a classe dos APDs, essa opção permite a definição dos símbolos que serão usados na pilha auxiliar.
-

Figura 43 - Botão para Definição de Alfabeto Auxiliar para as APDs

7.1.2. Janelas complementares:

1-Abrir Projeto:

Essa janela, ilustrada na Figura 44, é um componente do próprio GNOME. Com essa opção, pode-se abrir projetos da GAM que foram salvos anteriormente. A extensão de arquivo válida é ".gam".

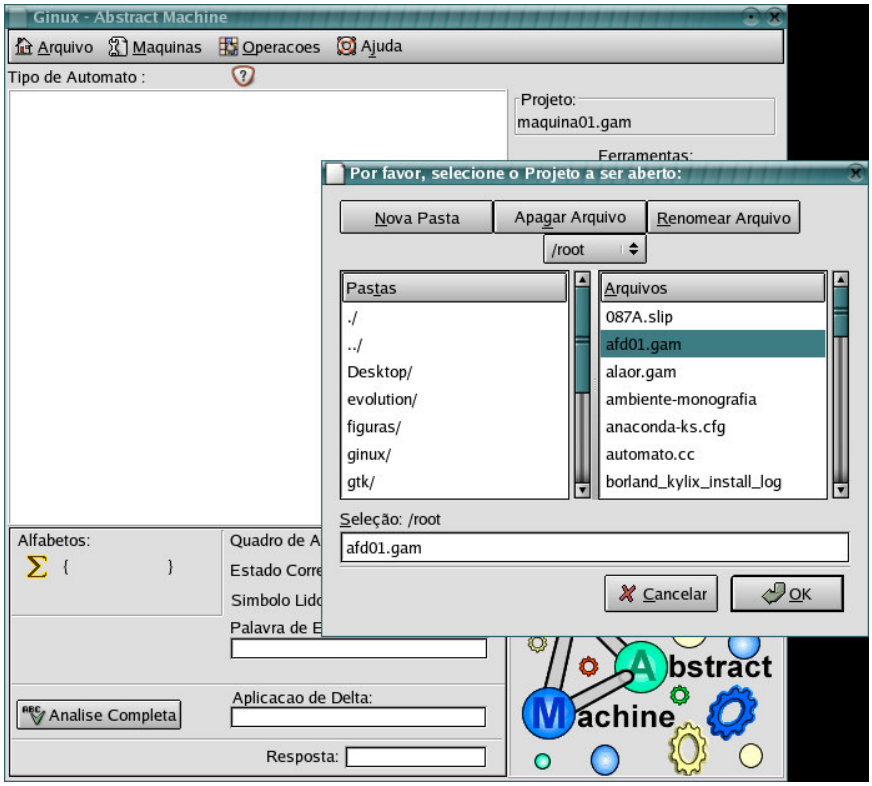

Figura 44 - Tela de Abertura de Projetos

2-Salvar Projeto:

A janela representada na Figura 45 é um componente do próprio GNOME. Com essa opção, pode-se salvar projetos da GAM. A extensão de arquivo válida é ".gam".

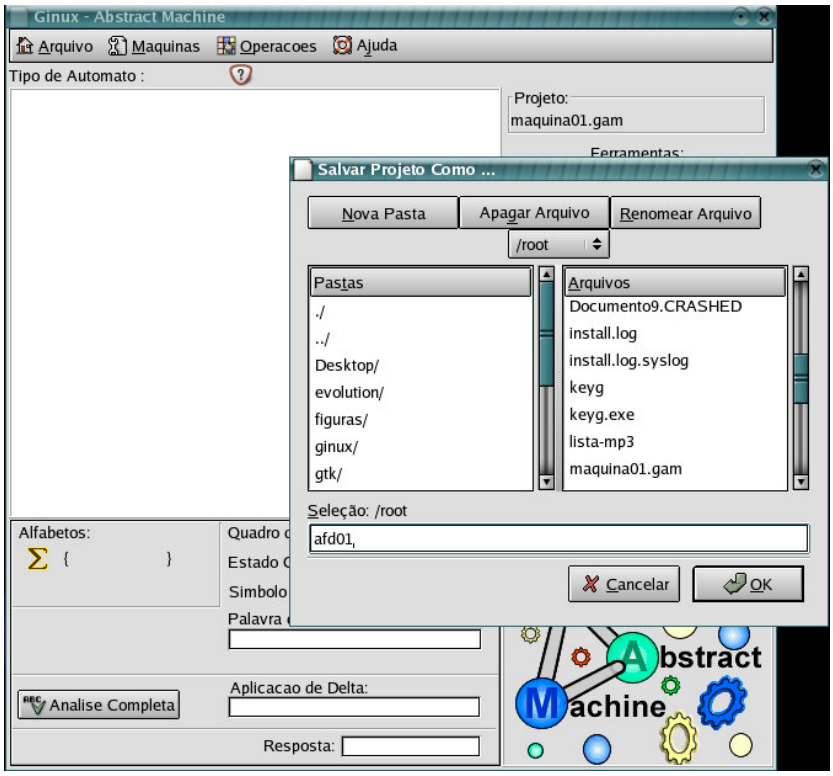

Figura 45 - Janela de Gravação de Projeto

3-Estado Inicial:

Através dessa janela, como mostra a Figura 46, seleciona-se qual estado previamente criado será o estado inicial do Autômato. Esse estado é único e possui a funcionalidade de alternar a condição de ser inicial entre todos os estados criados, com um simples clique.

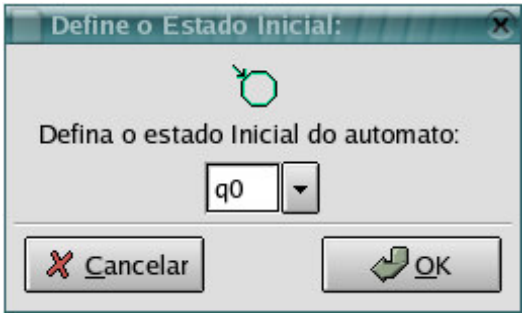

Figura 46 - Janela de Definição de Estado Inicial do Autômato

4-Estado Final:

Essa janela apresenta a funcionalidade de se alternar entre a condição de "final" e "não-final" dos estados do Autômato, como mostra a Figura 47.

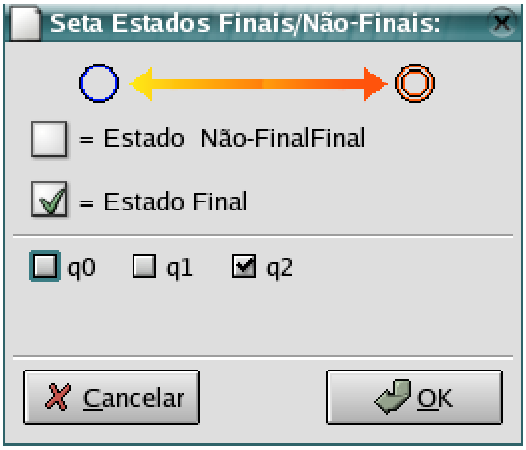

Figura 47 - Define Estados Finais / Não-Finais

5-Conecta Estados:

Essa opção fornece uma interface com o usuário para conectar os estados indicando os símbolos da operação. Como ilustra a Figura 48, basta selecionar o estado inicial e final da conexão e definir o símbolo de conexão. Essa opção é válida para os AFDs e AFNDs.

Figura 48 - Janela de Conexão de Estados

6-Remove Estados:

A janela da Figura 49 oferece a condição de remoção dos estados indesejados no Autômato. Vale ressaltar que, além do estado em questão, a ferramenta elimina todas as conexões relacionadas e renomeia os estados restantes.

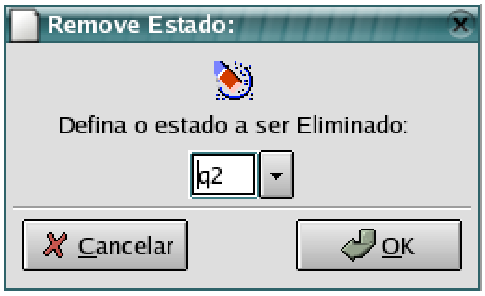

Figura 49 - Opção de Eliminação de Estados

7-Alfabeto de Entrada:

Essa opção fornece uma interface para o usuário definir quais são os símbolos de entrada do Autômato em questão, através da caixa de entrada ilustrada na Figura 50. Essa versão da ferramenta limita-se em três símbolos de entrada.

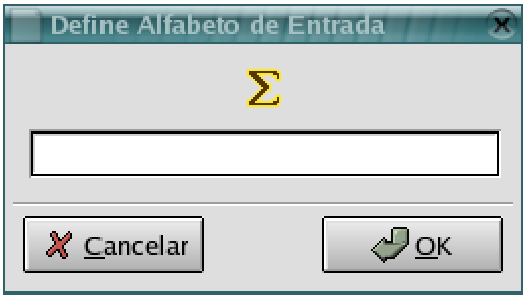

Figura 50 - Janela de Definição dos Símbolos de Entrada

8-Alfabeto de Saída:

Essa opção fornece uma interface para o usuário definir quais são os símbolos de saída da máquina de Turing. Essa versão da ferramenta limita-se em três símbolos de saída. A Figura 51 ilustra a opção:

Figura 51 – Janela de Definição de Símbolos de Saída

9-Alfabeto Auxiliar:

Essa opção, vista na Figura 52, fornece uma interface para o usuário definir quais são os símbolos auxiliares da pilha do Autômato *PushDown*. Essa versão da ferramenta limita-se em três símbolos auxiliares.

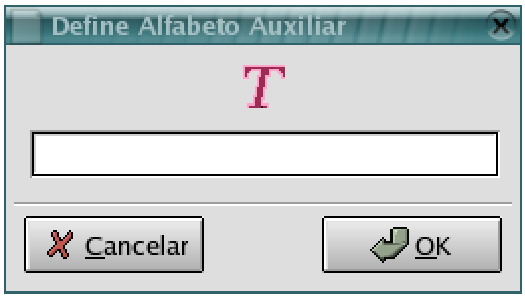

Figura 52 - Janela de Definição de Símbolos Auxiliares (pilha)

10-Remove Conexões:

A opção ilustrada na Figura 53 permite a remoção das conexões entre estados. Se uma conexão é removida, todos os símbolos a ela relacionados também serão eliminados da máquina.

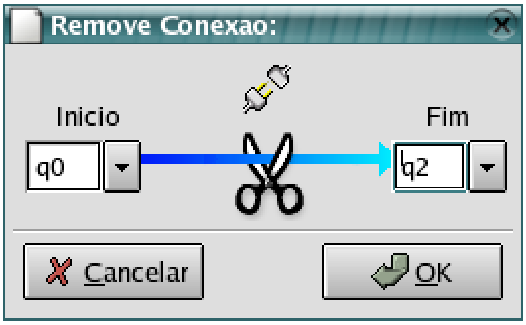

Figura 53 - Remoção de Conexões entre Estados

7.2. – O Funcionamento da GAM

Basicamente, a versão 1.1.0-3c da *Ginux Abstract Machine* fornece as seguintes funções:

- Abrir projetos salvos;
- Salvar projetos confeccionados;
- Criar novos projetos;
- Definir tipos de máquinas (AFD, AFND, APD e MT);
- Inserir estados;
- Conectar estados;
- Remover estados;
- Remover conexões;
- Definir alfabeto de entrada;
- Definir alfabeto de saída;
- Definir alfabeto auxiliar;

– Análise detalhada de uma entrada para os modelos determinísticos de Autômatos;
– Análise rápida (completa), com aplicação da função programa às entradas válidas ou inválidas para cada máquina;
– Confecção automática da função programa de um Autômato;

Dentre as operações, destacam-se:

– Conversão de AFNDs para AFDs;
– Minimização de AFDs;
– Restauração da máquina original após a aplicação de uma das operações acima citadas.

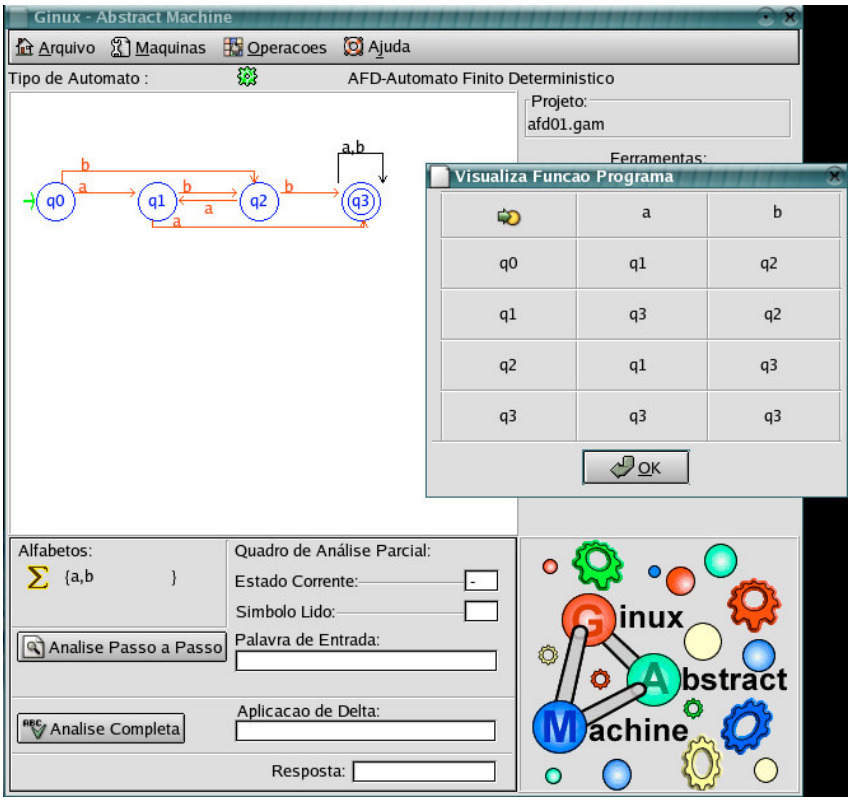

Figura 54 - Simulação de um AFD (destaque para função programa)

As Figuras 54 e 55 demonstram a GAM em uma simulação de um AFD:

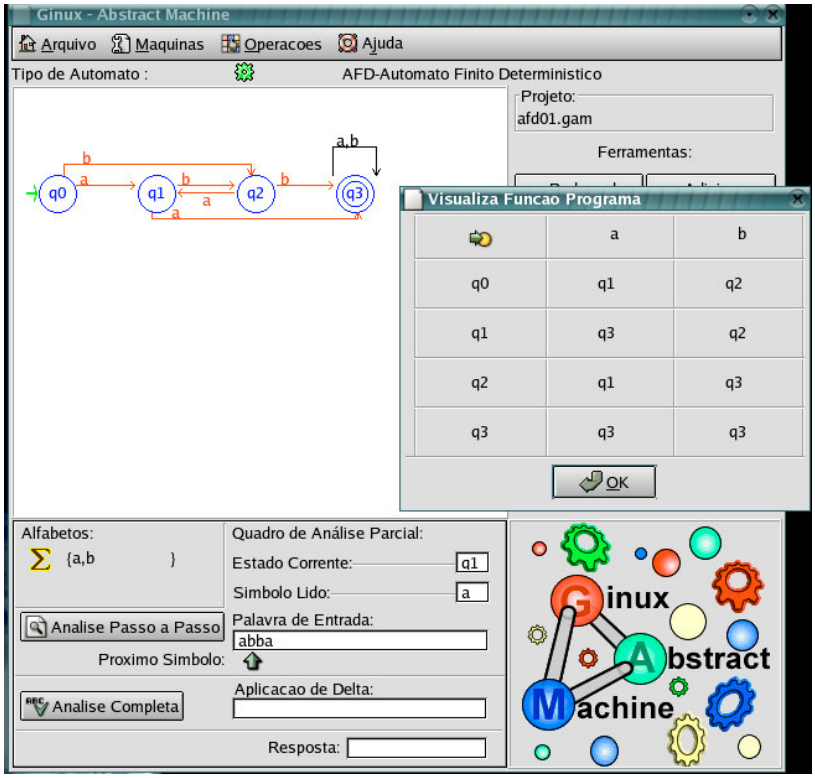

Figura 55 - Simulação da análise passo-a-passo de uma entrada em um AFD

A Figura 56 demonstra o funcionamento de um AFND e a aplicação da função programa (delta):

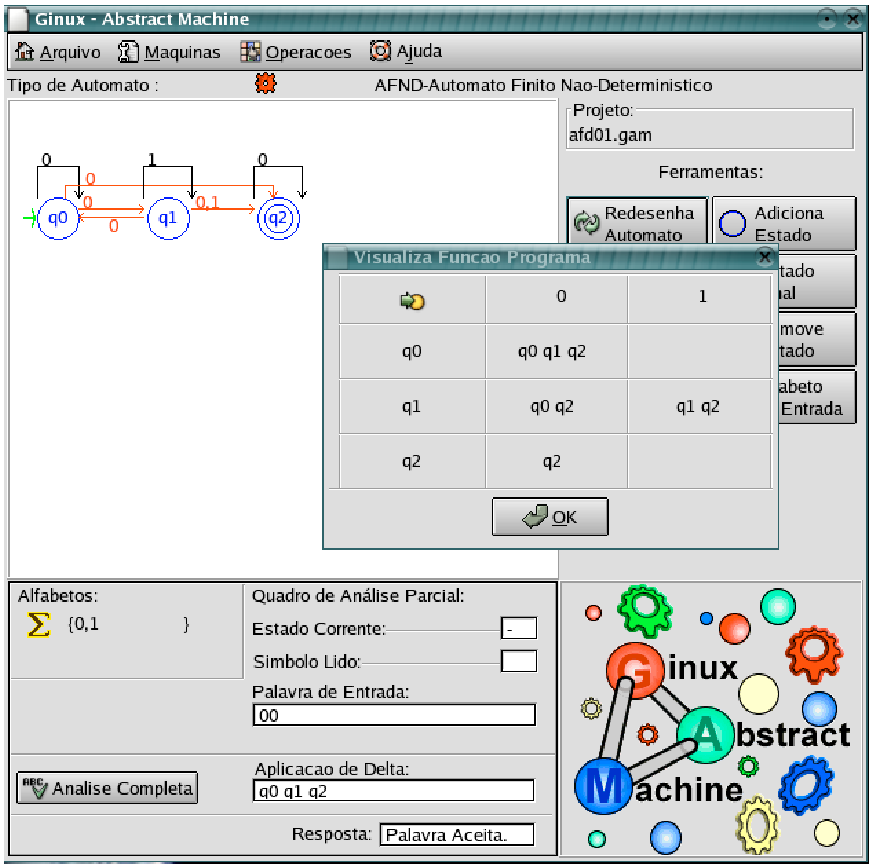

Figura 56 - Simulação de um AFND

A GAM, como citado anteriormente, apresenta a função de conversão entre um AFND a um AFD. As Figuras 57 e 58 ilustram essa funcionalidade:

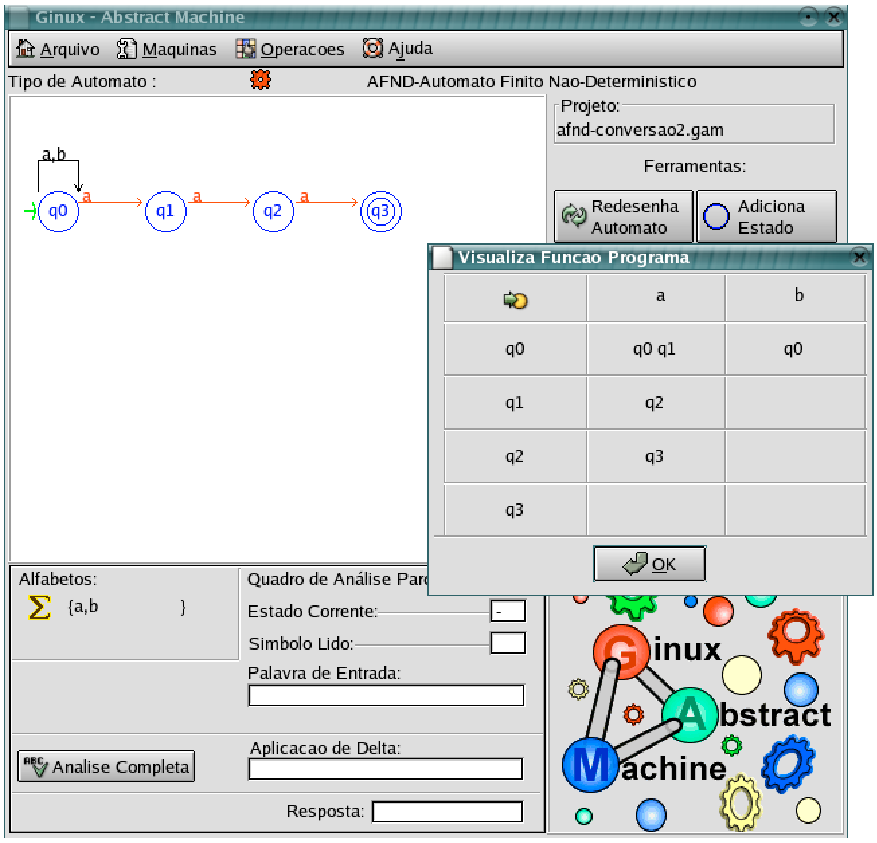

Figura 57 - AFND antes da conversão

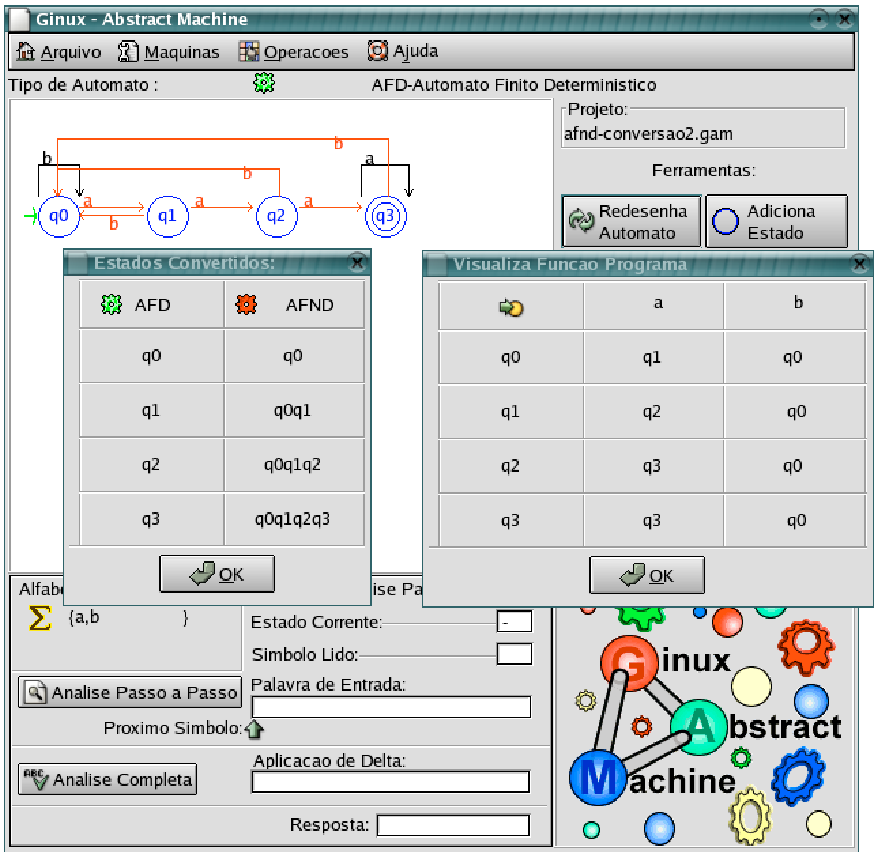

Figura 58 - AFD gerado a partir do AFND da Figura 57

A GAM apresenta a função de Minimização de AFDs. As Figuras 59 e 60 ilustram essa funcionalidade:

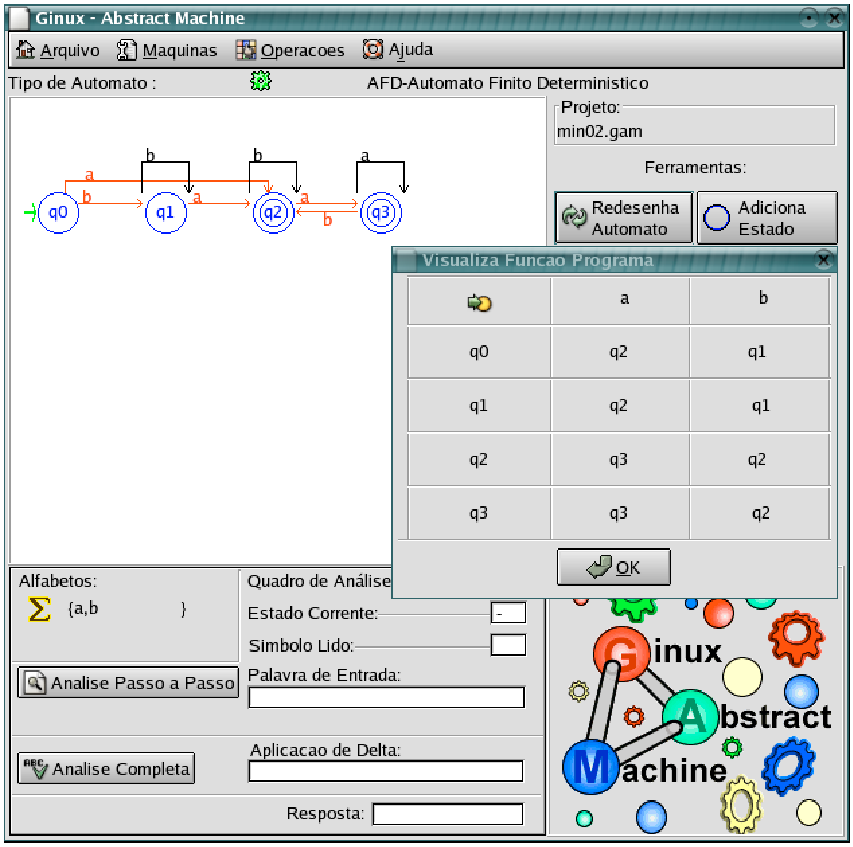

Figura 59 - AFD pronto para ser minimizado

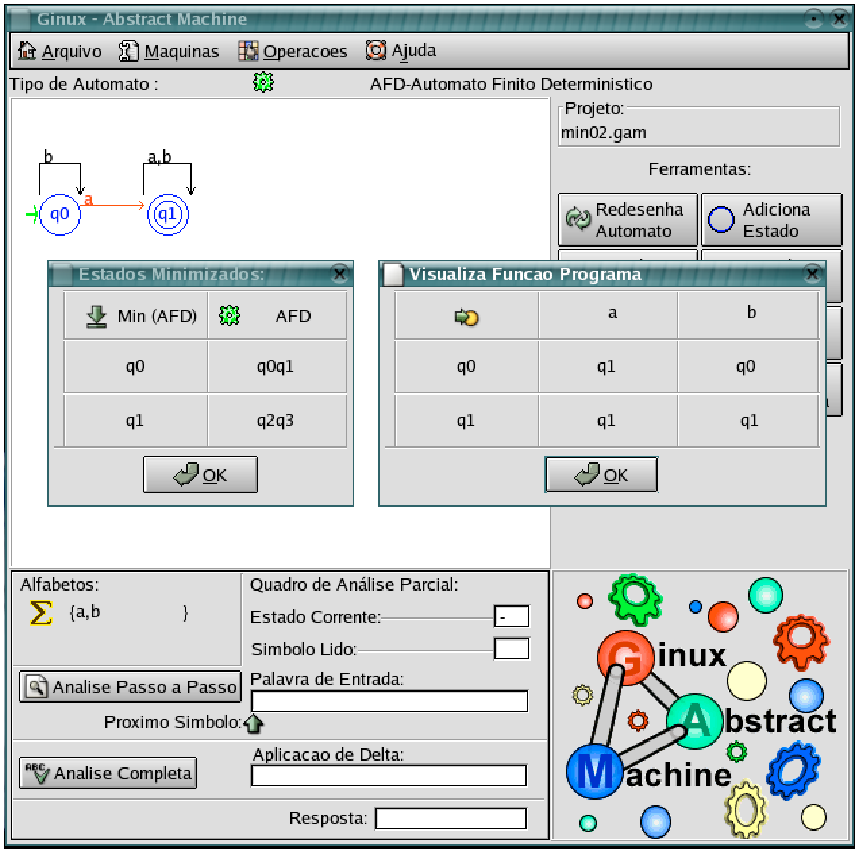

Figura 60 - AFD minimizado a partir do AFD da Figura 59

A Figura 61 ilustra a GAM simulando um Autômato *PushDown*.

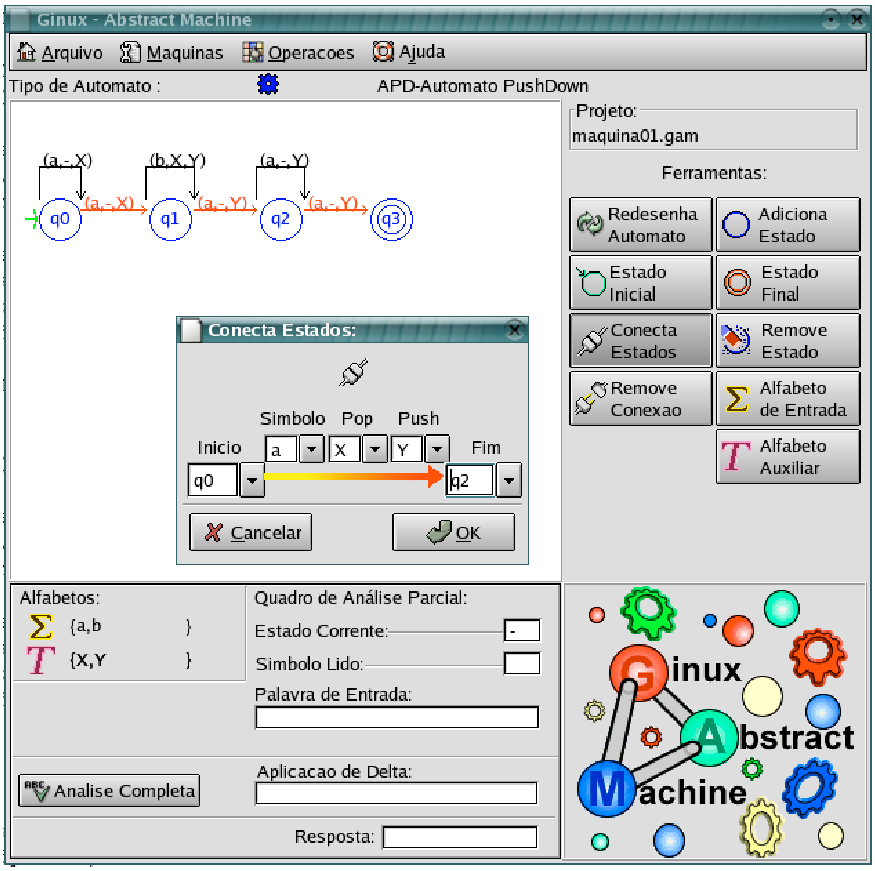

Figura 61 – GAM: Simulação de um APD

A Figura 62 ilustra a GAM simulando uma Máquina de Turing de fita

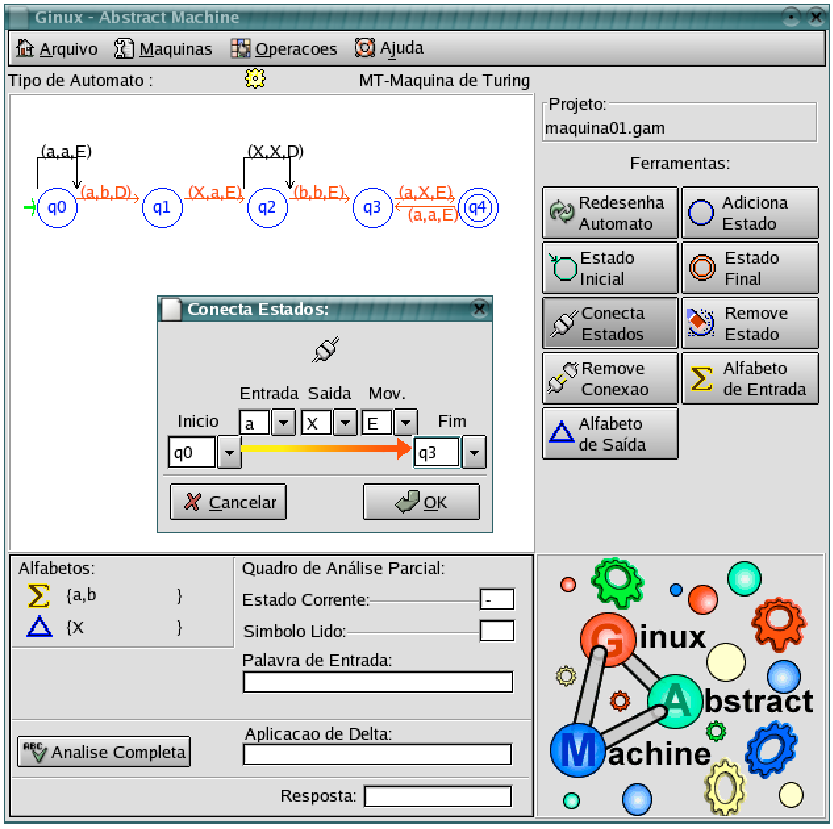

Figura 62 - GAM: Simulação de uma MT com Fita Única

8. Considerações Finais

A GAM foi implementada com as funções de simulação de AFDs e AFNDs, com opções de análise passo a passo e análise total para *strings* de entrada com até vinte caracteres. Com o auxílio da interface que expõe a função programa, fica evidente a facilidade de estudar essas classes de uma forma muito clara e prática.

A principal vantagem da ferramenta encontra-se nas funções de simulação de AFD e AFND, juntamente com as operações de conversão de AFNDs para AFDs e minimização de AFDs, pois fornecem uma excelente opção prática à exposição teórica do estudo desses itens.

O desenvolvimento da ferramenta centrou-se primariamente no estudo e aprendizado da linguagem GTKmm, bem como também na utilização de todos os grupos de ferramentas necessárias para a conclusão do projeto.

Com base nisso, e no tempo gasto na implantação do projeto, a primeira versão da GAM foi desenvolvida com algumas limitações:

- Os símbolos para os alfabetos de entrada, saída e auxiliar foram limitados a uma quantidade máxima de três.
- quadro de desenhos é estático, portanto não fornece a liberdade ao usuário de manipular com liberdade os componentes das máquinas.
- A ferramenta fornece uma interface para criação de autômatos com até dez estados.
- Os símbolos das transições, em algumas situações, ficam superpostos, ocasionando dificuldades na leitura. Essa situação, porém, é contornada com o uso da interface para expor a função programa que mapeia todas as transições.

–

– As classes APD e MT especificamente são simuladas e expostas graficamente. Porém, para essa versão da GAM, ainda não conseguem ler e interpretar strings de entrada e simular suas funcionalidades em particular.

Cita-se também que o projeto foi desenvolvido sob licença GPL. Com a sua continuidade, em conjunto com a divulgação à comunidade livre, espera-se o surgimento de muitas contribuições para o desenvolvimento das próximas versões.

Adicionalmente, propõe-se de imediato, a implantação das seguintes operações em uma nova versão:

- Análise de cadeia de entrada para os modelos de APDs e MTs.
- Simulação passo a passo da pilha auxiliar para os APDs.
- Simulação da fita de entrada para as MTs.

Fica evidente também que algumas otimizações podem ser projetadas em futuras novas versões da ferramenta, como por exemplo:

- Alteração da classe de desenho gráfico das máquinas abstratas para que suportem mais estados e também possa fornecer liberdade de manipulação de componentes pelo usuário.
- Implementação dos Autômatos Finitos Não-Determinísticos com movimentos vazios. Em decorrência disso a implementação do ε-fecho dos estados para esse tipo de Autômato.

O projeto como um todo apresenta-se adequado como ferramenta acadêmica para compreensão da teoria que envolve os Autômatos e Linguagens Formais, satisfazendo, portanto, seu principal propósito.

Referências bibliográficas

AHO, A. V., *Currents in the Theory of Computing*, Prentice Hall, 1973

BARR, M. e Wells, C., *Category Theory for Computing Science*, Prentice Hall, 1990

DEGANO, J. G., *Theory of Computation, Formal Languages, Automata and Complexity*, Benjamin/Cummings, California, E.U.A., 1989

HOPCROFT, John E., *Introduction to automata theory, languages and computation*. USA: Addison-Wesley Publishing Company, Inc. 1979. P.1-170

MENEZES, Paulo F. Blauth. *Linguagens Formais e Autômatos*. Porto Alegre: Ed. Sagra Luzzatto, 2000. 160p

SUDKAMP, Thomas A., *Languages and Machines*. USA: Addison-Wesley Publishing Company, Inc. Segunda Edição. P.155-295

SERNADAS, C., *Introdução à Teoria da Computação*, Editora Presença, Portugal, 1992

CUMMING Murray, *GTKmm – The C++ Interface to GTK+*. [on-line]. Disponível na internet via http://www.gtkmm.org. arquivo capturado em maio de 2001.

CHAPLIN Damon, *GLADE GTK+ - User Interface Builder*. [on-line]. Disponível na internet via http://glade.gnome.org/. Arquivo capturado em maio de 2001.

WELCOME Uriah, *SourceForge(tm)*. [on-line]. Disponível na internet via http://sourceforge.net. Arquivo capturado em dezembro de 2002.

Apêndice A - Lista de Símbolos e Abreviaturas

☸	Símbolo ou marcador de início da fita
β	Símbolo especial branco
δ	Função Programa (delta)
Σ	Alfabeto de entrada para AFD, AFND, AFε, APD e MT
2^Q	Conjunto das partes de um conjunto Q

A

AF	Autômato Finito
AFε	Autômato Finito com Movimentos Vazios
AFNDε	Autômato Finito Não Determinístico com Movimentos Vazios
AFD	Autômato Finito Determinístico
AFND	Autômato Finito Não-Determinístico
AP	Autômato *PushDown* ou Autômato com Pilha
APD	Autômato *PushDown* ou Autômato com Pilha
APN	Autômato com Pilha Não-Determinístico
ATK	Biblioteca que fornece um conjunto de interfaces para acessibilidade

F

FECHOε	Função Fecho Vazio
FECHOε	Função Fecho Vazio Estendida

G

G++	GNU C++ *Compiler*
GAM	*Ginux Abstract Machine*
GCC	GNU C *Compiler*
Glade	Ambiente construtor de interfaces de usuário para GTK+
Glademm	Extensão para glade e glade-2 para criação de fontes em C++ para programas baseados em gtk-- e gtkmm
GLIB	Biblioteca de utilitários de propósito geral
Gnome	Ambiente de desktop para GNU/Linux e Unix
GNU	Acrônimo para "*GNU's Not UNIX*"
GPL	*General Public License*
GTK	Gimp *Toolkit*
GTK+	Biblioteca para criação de interfaces gráficas de usuário

GTKmm	Interface C++ para a interface gráfica GTK+

L

LGPL	Library General Public License

M

MT	Máquina de Turing