UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MARCO AURÉLIO GAIDZINSKI

AMBIENTE DE CRIAÇÃO E MANIPULAÇÃO DE AUTÔMATOS FINITOS NA FORMA GRÁFICA OU TABULAR PARA O RECONHECIMENTO DE SENTENÇAS

CRICIÚMA, JULHO DE 2007.

MARCO AURÉLIO GAIDZINSKI

AMBIENTE DE CRIAÇÃO E MANIPULAÇÃO DE AUTÔMATOS FINITOS NA FORMA GRÁFICA OU TABULAR PARA O RECONHECIMENTO DE SENTENÇAS

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

Orientadora: Profª. M.Sc. Christine Vieira Scarpato

CRICIÚMA, JULHO DE 2007.

Para meu filho,

Rafael Ronsoni Gaidzinski

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter permitido o alcance de mais essa conquista.

A minha orientadora, Profª. M.Sc. Christine Vieira Scarpato, por acreditar em meu projeto, em minha capacidade para colocá-lo em execução e essencialmente por solucionar todas as minhas duvidas sempre com muita dedicação e paciência.

Aos meus pais, Altair Gaidzinski e Leudite Spilere Gaidzinski que nunca mediram esforços para me dar uma boa educação e que sempre estiveram presentes com apoio e incentivo, mostrando os verdadeiros valores da vida.

Aos meus amigos e colegas do Curso de Ciência da Computação pelos momentos de alegria compartilhados juntos. A todos que de alguma forma contribuíram para realização deste projeto.

E principalmente, a minha namorada, Valéria de Bona Tiscoski pelo apoio dedicado, e por estar sempre ao meu lado, compartilhando anseios, idéias e conquistas.

RESUMO

Este trabalho apresenta o desenvolvimento de um software que tem como finalidade permitir a construção de Autômatos Finitos Determinísticos e Não Determinísticos tanto em sua forma gráfica quanto na forma tabular. Para isto foi feito um estudo sobre Autômatos Finitos Determinísticos e Não Determinísticos, suas linguagens, suas equivalências e as suas duas formas de representação (Tabular e Gráfica). O presente trabalho ainda aborda as técnicas para implementação de autômatos finitos que permitem o reconhecimento de sentenças. O trabalho traz a descrição da ferramenta desenvolvida e a metodologia aplicada para seu desenvolvimento, bem como o algoritmo que permite o reconhecimento de sentenças tanto para um Autômato Finito Determinístico (AFD) como também para um Autômato Finito Não Determinístico (AFND) sem a necessidade de transformações ou minimização dos mesmos.

Palavras-chave: Autômato Finito Determinístico, Autômato Finito Não Determinístico e Reconhecedor de Sentenças.

ABSTRACT

This work shows the software development that has as purpose to build a deterministic and non-deterministic finite automates through state transition diagrams as well as state transition tables. A study about this issue was made in order to achieve this task. It was focused on their program languages, their equivalences and their representation states (diagrams and tables). Furthermore, this software also allows recognizing sentences by the technical way of finite automata implementation. The applied methodology of the designed toll and the algorithm to recognize a sentence of deterministic finte automates (AFD) and non-deterministic finite automates (AFND) without transformations or minimization necessities were described.

Word-key: Deterministic Finite Automata, Non-Deterministic Finite Automata and Sentence Recognizer.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AF	Autômato Finito
AFs	Autômatos Finitos
AFD	Autômato Finito Determinístico
AFND	Automato Finito Não-Determinístico

SUMÁRIO

1 INTRODUÇÃO

Autômatos finitos (AFs) são utilizados nas disciplinas de Compiladores e Linguagens Formais e também estão ligados a Teoria da Computação. Em Compiladores são utilizados para elaborar o analisador léxico, enquanto que para Linguagem Formais e Teoria da Computação sua finalidade é de reconhecer se uma determinada sentença pertence ou não a linguagem em uso, atuando desta forma como um reconhecedor.

Entende-se como reconhecedor de sentença, um dispositivo que recebe um valor de entrada, e responde sim caso o valor pertença à linguagem e não caso contrário. Este reconhecimento é realizado com a representação dos AFs na sua forma gráfica (diagrama de transição) ou na sua forma tabular (tabela de transição de estados), por meio de desenho em papel, ou seja, sem o uso de uma ferramenta computacional durante o ciclo acadêmico.

Esta situação torna-se difícil quando se quer representar um Autômato Finito (AF) com um grande número de estados ou testar várias sentenças para o mesmo autômato, ou até mesmo alterá-lo, acrescentando ou eliminando estados. No papel esta situação requer muitas vezes que se reescreva todo o autômato novamente.

Desse modo, ou seja, diante desta dificuldade surge o grupo de pesquisa de linguagens formais do curso de Ciência da Computação da UNESC que visa o desenvolvimento do projeto de uma *shell* de AFs, denominada AFLAB, com o objetivo de num futuro próximo disponibilizá-la gratuitamente ao meio acadêmico.

Conseqüentemente, este é um dos módulos em desenvolvimento para a *shell* AFLAB. Este módulo será dividido em várias partes, sendo que a pesquisa aqui

proposta compreende a criação de um reconhecedor de sentenças por meio de autômatos finitos na forma tabular ou na forma gráfica.

1.1 OBJETIVO GERAL

Desenvolver um reconhecedor de sentenças baseado em autômatos finitos, que permita a construção dos autômatos finitos em sua forma tabular ou gráfica.

1.2 OBJETIVOS ESPECÍFICOS

Esta pesquisa possui como objetivos específicos:

a) compreender Autômatos Finitos Determinísticos e Autômatos Finitos Não Determinísticos;
b) transformar autômatos finitos em diagrama de transição para a forma tabular ou na forma tabular para o diagrama de transição;
c) implementar interfaces gráficas no ambiente Borland® Delphi™;
d) disponibilizar um reconhecedor de linguagens de forma gratuita.

1.3 JUSTIFICATIVA

Esta pesquisa se refere a um ambiente para a criação e manipulação de AFs em função da sua aplicabilidade em diferentes meios, como por exemplo, nos corretores ortográficos, mecanismos de controle de elevadores, entre outros. Além disso, constituem-se no tipo mais simples de reconhecedores de linguagens. Podendo ser utilizado como reconhecedor de padrões em processamento de textos e também para a

criação do analisador léxico em um compilador. Segundo Scarpato[1] os AFs apresentam algumas vantagens como: a simulação de um AF requer uma quantidade finita de memória; existe uma diversidade de teoremas e algoritmos para se construir e simplificar autômatos finitos e a simulação de um autômato finito apresenta uma boa velocidade de processamento devido ao pequeno número de operações efetuadas para processar um símbolo de entrada.

O autômato finito que será utilizado neste projeto poderá ser visualizado, criado ou manipulado por meio do diagrama de transição ou da tabela de transição de estados, proporcionando ao usuário mais funcionalidades no uso da *shell* AFLAB.

Apesar dos AFs serem amplamente utilizados e analisados na disciplina de Linguagens Formais, Compiladores e Teoria da computação, é desconhecida a existência de ferramentas com ambiente gráfico que permitam a construção de AFs através do diagrama de transição (forma gráfica) disponível no meio acadêmico nos dias de hoje, quando se vivencia a era da informação. Além disso, a Ciência da Computação é tradicionalmente ensinada com o uso de poucos softwares específicos para a área, realizando a maioria de suas tarefas sem o uso do ambiente computacional.

Com isso, a pesquisa compreende o desenvolvimento do módulo reconhecedor de sentenças da Shell AFLAB, que poderá ser utilizada no meio acadêmico do curso de ciência da computação da UNESC, nas disciplinas de linguagens formais e compiladores.

Nesse módulo da *shell* AFLAB o reconhecimento de uma sentença não irá preocupar-se com a transformação de AFND para um AFD, ou até mesmo em uma possível minimização.

---

[1] SCARPATO, Christine Vieira. **Linguagens Formais e Compiladores.** Apostila da disciplina de Linguagens Formais e Compiladores da UNESC, 2004.

Assim, a Shell AFLAB irá percorrer todos os caminhos possíveis de um autômato finito, ignorando o fato de este por sua vez ser um não determinístico, para finalmente diagnosticar se a sentença pertence ou não a linguagem em uso.

1.4 ESTRUTURA DO TRABALHO

O capítulo 2 faz uma introdução sobre a máquina de estado finito, explicando o que é um Autômato Finito. Descreve o conceito de um AFD e suas linguagens aceitas, assim como as características de um AFND com suas linguagens aceitas. Este capítulo ainda mostra a equivalência entre um AFD e um AFND e seus relacionamentos. O capítulo 3 abrange os tipos de representação para AFs. O capítulo 4 traz as técnicas de implementação capazes de realizar o reconhecimento de sentenças por meio de AFs.

Alguns trabalhos correlatos são apresentados no capítulo 5. No capítulo 6 é apresento o módulo da *shell* AFLAB que realiza o reconhecimento de sentenças por meio de AFs. Neste capítulo é feito uma descrição das principais características e funcionalidades da ferramenta, bem como a metodologia aplicada para a realização da mesma. No capítulo 7 são apresentadas as considerações finais para o trabalho e algumas sugestões para trabalhos futuros.

2 MÁQUINA DE ESTADO FINITO

Uma máquina de estado finito é um modelo que retrata as características de um computador digital atual, onde, se sabe que as informações internas são armazenadas de forma binária. Em um dado momento o computador contém determinadas informações armazenadas em sua memória interna definidas por um padrão de dígitos binários, que será intitulado de estado do computador nesse momento. Como se sabe que um computador possui uma quantidade finita de memória, fica evidente que ele possui um número finito de estados para poder assumir. Seu *clock* interno ou relógio interno irá sincronizar suas ações. Em cada ciclo do relógio, a entrada pode ser lida o que pode provocar alguma mudança nas posições de memória, logo, muda-se o estado da máquina para um novo estado. O que cada novo estado representa depende da entrada e de seu estado anterior, portanto se estes dois fatores forem conhecidos cada alteração será previsível e não casual. Desta forma, após uma seqüência de ciclos do relógio, a máquina irá produzir uma série de saídas em resposta ao conjunto de entradas (GERSTING, 1995).

Portanto as máquinas de estados finitos apresentam características de um computador, onde percebe-se as seguintes propriedades no seu comportamento:

a) as operações da máquina são sincronizadas por ciclos discretos do relógio;
b) a máquina atua de forma determinística, ou seja, para a seqüência de entrada existe uma resposta completamente previsível;
c) a máquina fornece respostas mediante as entradas;
d) como existe um número finito de estados, somente este número finito poderá se alcançado pela máquina. A máquina está em qualquer

momento num desses estados. O estado que ela vai estar a seguir será o resultado da função aplicada no estado atual e da entrada atual. O estado atual, contudo, depende dos estados e entradas anteriores e assim sucessivamente até se chegar novamente a configuração inicial. Conseqüentemente, o estado da máquina serve como uma espécie de memória das entradas precedentes;

e) a máquina tem a capacidade de produzir saídas. Cada saída é resultado de uma função aplicada em cada estado atual da máquina.

As máquinas de estados finitos são máquinas abstratas que capturam os elementos essenciais de alguma máquina real. Compreendendo desde máquinas de vender jornais, estendendo-se por relógios digitais, elevadores e chegando até programas de computador, onde executam procedimentos de editores de texto e de compiladores (VIEIRA, 2006).

Embora existam máquinas abstratas mais importantes do que as de estado finito, as máquinas de estado finito são apropriadas, tanto na sua parte teórica quanto na parte prática para a construção de um amplo espectro de máquinas como as mecânicas, eletrônicas, de software, entre outros (VIERA, 2006).

2.1 AUTÔMATOS FINITOS (AF)

A teoria dos autômatos estuda os dispositivos de computação abstratos, ou máquinas. Em 1930 antes da existência dos computadores, Alan Turing pesquisou uma máquina abstrata com características similares aos computadores atuais, pelo menos no que se refere a sua capacidade de calcular. Seu objetivo era expor com clareza o limite entre o que uma máquina de computação poderia ou não fazer. Os resultados adquiridos

com estas pesquisas estão aplicados nas *máquinas de Turing* abstratas e também nos micro computadores atuais (HOPCROFT; ULLMAN; MOTWANI, 2002).

Entre 1940 e 1950 modelos de máquinas mais simples denominados nos dias de hoje como autômatos finitos foram estudados por diversos pesquisadores. Esses autômatos que inicialmente foram propostos para modelar a função do cérebro se mostraram muito úteis em várias outras finalidades, como softwares que projetam e verificam o comportamento de circuitos digitais; analisadores léxicos de compiladores; softwares que fazem varredura em corpos de texto, a fim de encontrar ocorrência de palavras entre outros propósitos. Já no final os anos 50, Avram Noam Chomsky (professor de lingüística) deu início ao estudo de gramáticas regulares, que apesar do relacionamento estreito com autômatos essas gramáticas servem como base para algumas partes dos compiladores, como no processo de análise sintática, análise semântica e geração de código (HOPCROFT; ULLMAN; MOTWANI, 2002).

Stephen Cook no ano de 1969 aprofundou os estudos de Turing sobre o que podia ou não ser calculado por um computador, assim, Cook separou os problemas que são resolvidos de forma eficiente por uma máquina, daqueles problemas que também podem ser resolvidos, mas que necessitam de muito tempo para gerar um resultado, tornando-se inviáveis (conhecidos como problemas intratáveis). Cook ainda declara que mesmo com uma melhoria exponencial na velocidade de computação, adquiridos com o avanço que a arquitetura vem obtendo, é improvável que se tenha um impacto significativo sobre nossa habilidade para resolver estes problemas intratáveis (HOPCROFT; ULLMAN; MOTWANI, 2002).

Todos estes estudos estão diretamente relacionados com o que os cientistas da computação praticam atualmente, onde alguns conceitos, como autômatos finitos e certos tipos de gramáticas formais são utilizados no projeto e na construção de

importantes componentes de software, como compiladores, sistemas de varredura entre outros (HOPCROFT; ULLMAN; MOTWANI, 2002).

Autômatos finitos podem ser entendidos como reconhecedores de linguagens regulares ou expressões regulares. Um reconhecedor de linguagem é um sistema que recebe como entrada uma cadeia de caracteres *x* e retorna sim caso a cadeia pertença à linguagem em estudo ou não caso contrário (MENEZES, 2005).

Em outras palavras um autômato finito ou um sistema de estados finitos também pode ser visto como um modelo matemático de um sistema, controlando suas entradas e saídas discretas, e ainda podendo assumir número finito e pré-definido de estados. Cada estado mantém somente as informações do passado que são necessárias para determinar as ações para a próxima entrada. Assim um fator motivacional para o estudo de estados finitos é o relacionamento destes com diversos tipos de sistemas naturais ou construídos (MENEZES, 2005).

O fato de se ter apenas um estado inicial para cada autômato não enfraquece seu poder computacional (VIEIRA, 2006).

Como exemplo de um sistema baseado em AFs, tem-se o mecanismo de controle de elevadores, onde, o sistema não lembra de todas as requisições anteriores, mas armazena informações como a do andar atual, juntamente com a direção a ser seguida (para cima ou para baixo) e uma relação de requisições pendentes. Outros sistemas como corretores ortográficos (ferramentas para processamento de texto), também utilizam à idéia de autômatos finitos. Aqui cada estado, essencialmente, memoriza a estrutura do prefixo da palavra em análise (MENEZES, 2005).

Porém, nem todos os sistemas de estado finito são apropriados para serem analisados por esta abordagem (autômatos finitos). Um exemplo que pode ser tratado como uma exceção é o próprio cérebro humano, onde existem provas de que um

neurônio pode ser modelado por um número finito de bits. Deste modo o cérebro que é composto por cerca de 2^{35} células, pode ser representado por um número finito de estados. Contudo, o altíssimo número de combinações de células implica em uma abordagem precária em termos práticos. E o ato de implementar um sistema como este ocasionaria em um fenômeno conhecido como explosão de estados. Outro exemplo é o computador, onde os estados determinados pelos processadores e memórias podem ser simulados por um sistema de estados finitos. Contudo, o estudo apropriado da computabilidade exige memória sem limite predefinido, ocasionando também o fenômeno de explosão de estados quando simulados por autômatos finitos (MENEZES, 2005).

A construção da maioria dos sistemas é composicional, o que significa que são construídos a partir de sistemas conhecidos, e deste modo continuamente até chegar ao nível mais simples. Entre as formas de composição três se destacam:

a) *seqüencial*: A realização do próximo componente depende da finalização do componente anterior;
b) *concorrente*: São componentes independentes, onde a ordem em que são executados não importa, podendo ocorrer até ao mesmo tempo;
c) *não-determinística*: O próximo componente a ser executado será resultado de uma escolha entre diversos componentes alternativos, ou seja, ao contrário do determinismo onde o próximo componente a ser executado dado as mesmas condições será sempre o mesmo. O não-determinismo ainda pode ser tratado como interno, quando o sistema escolhe o próximo componente a ser executado, ou externo, quando a escolha do próximo componente é externa ao sistema (não depende do sistema).

Um AF ou sistema de estados finitos possui um número finito e predefinido de estados que constituem um modelo computacional do tipo seqüencial, bastante comum nos vários estudos da computação e informática, destacando-se nas áreas de Linguagens Formais, Compiladores, Semântica Formal e Modelos para Concorrência (MENEZES, 2005).

Segundo Menezes (2005) um autômato finito pode ser visto como uma máquina composta, basicamente, de três partes:

a) *fita*: dispositivo de entrada que contem 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 corrente da máquina.

A fita está dividida em células e é finita tanto a esquerda quanto a direita. Cada célula irá armazenar um símbolo, sendo que estes pertencem a um alfabeto de entrada. O sistema não permite gravar sobre a fita, mesmo porque não se tem memória auxiliar. Inicialmente a palavra a ser analisada irá determinar o tamanho da fita, visto que a mesma irá ocupar toda a sua extensão (MENEZES, 2005).

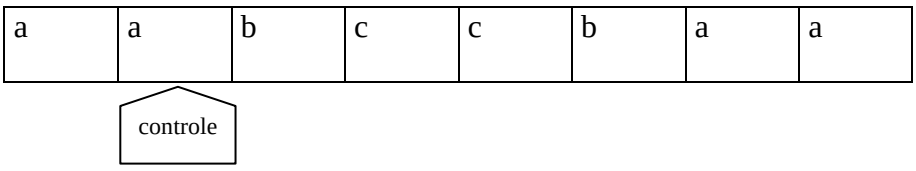

Figura 1. Autômato finito como uma máquina com controle finito

A unidade de controle do autômato armazena o número finito e predefinido de estados. Inicialmente, a cabeça da fita que representa a unidade de controle irá estar

posicionada na célula mais a esquerda da fita, e à medida que a leitura é realizada, a cabeça irá se deslocar uma célula para a direita. A unidade de leitura lê um símbolo de entrada de cada vez (MENEZES, 2005).

O programa é uma função que irá mapear o estado atual, e dependendo do estado corrente e do símbolo lido, determinará o novo estado do autômato (MENEZES, 2005).

De forma resumida, um AF é representado por um controle finito, que tem acesso à fita que contem a seqüência a ser analisada. O autômato percorre a fita da esquerda para a direita, lendo um símbolo de cada vez. Estando o controle em um estado, a leitura do símbolo de entrada é realizada, permitindo uma transição de estado, que faz com que o controle passe para outro estado e avance para o próximo símbolo de entrada. Se isto se repetir com sucesso até atingir o final da fita, e o autômato se encontrar em um estado final, a seqüência será reconhecida, caso contrário, se ocorrer alguma falha como transição indefinida ou finalizar em um estado não final o reconhecimento para a seqüência não acontecerá.

Vale lembrar que um AF não possui memória de trabalho, desta forma, para que ocorra o armazenamento das informações necessárias ao processamento, deve-se usar o conceito de estado (MENEZES, 2005).

O conjunto de estados de um autômato é sempre finito e cada elemento está representado por nomes distintos (por exemplo, q1, q2,...., n). Onde normalmente estes nomes são mnemônicos, utilizando os mesmos conceitos aplicados às variáveis de programação. É evidente que em aplicações distintas a nomenclatura deve assumir valores que facilitem a identificação, eliminando nomes arbitrários (VIEIRA, 2006).

2.1.1 Autômato Finito Determinístico (AFD)

Segundo Menezes (2005) um autômato finito determinístico é formalmente definido por uma 5-upla M= (Σ, Q, *t*, q0, F), onde:

Σ = alfabeto dos símbolos de entrada;

Q = conjunto dos estados possíveis no Autômato o qual é finito;

t = função programa ou função de transição ou simplesmente programa, onde formalmente *t*: Q x Σ → Q;

q0 = estado inicial do Autômato e que pertence ao conjunto Q;

F = conjunto dos estados finais do Autômato pertencente ao conjunto Q.

A função de transição de estados irá tomar como argumentos um estado juntamente com um símbolo de entrada e retornar um estado. Esta função será habitualmente denotada por *t*. Logo, na representação da função *t* (A,a) = B, se *A* é um estado e *a* é um símbolo de entrada, então o estado *B* será o estado resultante ou estado mapeado pela função (HOPCROFT; ULLMAN; MOTWANI, 2002).

Como introdução clara de um AFD, deve-se entender por aquele que se encontra em um único estado após a leitura da seqüência de entrada. Onde o termo *determinismo* identifica a existência de somente um estado ao qual o autômato pode transitar a partir do estado atual (HOPCROFT; ULLMAN; MOTWANI, 2002).

2.1.1.1 Linguagem em Autômato Finito Determinístico

Uma linguagem aceita ou linguagem reconhecida por um autômato finito determinístico é o conjunto de todas as palavras pertencentes ao conjunto dos símbolos de entrada aceitas pelo autômato, incluído o vazio, partindo de seu estado inicial.

Semelhantemente, a linguagem rejeitada por um autômato finito determinístico é o conjunto de todas as palavras pertencentes ao conjunto dos símbolos de entrada mais o vazio e que são rejeitadas pelo autômato, a partir de seu estado inicial.

Conseqüentemente, cada AF definido sobre o alfabeto do conjunto dos símbolos de entrada, leva a uma participação do conjunto de todas as palavras em duas classes análogas, ou seja, as que aceitam ou as que rejeitam o autômato. Sendo necessário, quando um dos dois conjuntos for vazio, a participação induzida com apenas um conjunto, que coincide com o conjunto dos símbolos de entrada incluindo o vazio (MENEZES, 2005).

Deste modo seja P = (Σ, Q, *t*, q0, F) um autômato finito determinístico, e a linguagem aceita por P denotada por ACEITA(P) e a rejeitada denotada por REJEITADA(P) e ainda supondo que o conjunto dos símbolos de entrada incluindo o vazio seja o universo, se pode fazer as seguintes afirmações:

ACEITA(P) ∩ REJEITA(P) = ø

ACEITA(P) U REJEITA(P) = Σ^{*}

~ACEITA(P) = REJEITA(P)

~REJEITA(P) = ACEITA(P)

Dois autômatos finitos P_1 e P_2 são considerados equivalentes somente se a linguagem aceita por P_1 for igual à linguagem aceita por P_2 (MENEZES, 2005).

Uma linguagem regular ou linguagem do tipo três existe quando se tem pelo menos um autômato finito determinístico que aceite esta linguagem (MENEZES, 2005).

Logo, com um autômato finito determinístico P = (Σ, Q, *t*, q0, F) e sua linguagem denotada por L(P) e definida por:

$L(P) = \{h|t\ (q_o, h)$ está em F$\}$

Isto implica que a linguagem de P é o conjunto de *strings h* que fazem com que o estado inicial q_o chegue até um dos estados de aceitação. Com este reconhecimento se pode dizer que L(P) é uma linguagem regular (HOPCROFT; ULLMAN; MOTWANI, 2002).

2.1.2 Autômato Finito Não-Determinístico (AFND)

Segundo Menezes (2005) um autômato finito não determinístico também é uma 5-upla M=(Σ, Q, *t*, q0, F), onde:

Σ = alfabeto dos símbolos de entrada;

Q = conjunto dos estados possíveis no Autômato o qual é finito;

t = função programa ou função de transição ou simplesmente programa, onde formalmente *t*: Q x Σ → 2^Q, indicando que a aplicação da função poderá resultar em mais de uma saída;

q0 = estado inicial do Autômato e que pertence ao conjunto Q;

F = conjunto dos estados finais do Autômato pertencente ao conjunto Q.

Para cada AFND, é possível construir um AFD equivalente que realiza o mesmo processamento. O contrário também é verdadeiro (MENEZES, 2005).

Um AFND é aquele em que pode existir mais de uma transição saindo de um estado para o mesmo símbolo de entrada. Quando uma determinada entrada, a partir de certo estado apontar para mais de um estado, o autômato deixa de ser determinístico. Em outras palavras, no estado atual o símbolo de entrada não determina mais qual será o próximo estado (AHO; SETHI; ULLMAN, 1995).

Como um AFND tem o poder de estar em mais de um estado ao mesmo tempo, essa capacidade é explorada na sua utilização durante a busca de certas seqüências de caracteres em longos *strings* de texto. Assim torna-se útil para adivinhar que se está no começo de um desses *strings* e usar uma seqüência de estados apenas para conferir se a *string* surgiu, caractere por caractere (HOPCROFT; ULLMAN; MOTWANI, 2002).

Naturalmente por existir um número finito de estados, só pode haver um número finito de escolhas a tomar a partir de um estado e de um símbolo de entrada.

Uma facilidade apresentada pelo não determinismo para autômatos finitos é divulgada no programa, que é uma função parcial, onde, dependendo do estado corrente e do símbolo lido, determina um conjunto de estados do autômato (MENEZES, 2005).

Uma semântica utilizada para explicar o não-determinismo é apresentada por Menezes (2005, p. 55):

> Um Autômato Finito é visto como uma máquina composta por fita, unidade de controle e programa, pode-se afirmar que um Autômato Finito Não-determinístico assume um conjunto de estados alternativos, como se houvesse uma multiplicação da unidade de controle, uma para cada alternativa, processando independentemente, sem compartilhar recursos com as demais. Assim, o processamento de um caminho não influi no estado, símbolo lido e posição da cabeça dos demais caminhos alternativos.

Porém não será esta a semântica utilizada, se assumirá que a máquina escolherá um dos caminhos arbitrários a percorrer. Caso a sentença não seja reconhecida logo no primeiro caminho, outros caminhos serão percorridos até que todas

as possibilidades tenham sido esgotadas ou a sentença tenha sido reconhecida. Em contrário a sentença não será reconhecida.

Menezes (2005) diz ainda, 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.

2.1.2.1 Linguagens em Autômato Finito Não Determinístico

Um AFND aceita uma *string y* se for possível ter uma opção de escolha do próximo estado, enquanto os caracteres da *string y* são lidos, saindo do estado inicial e chegando a algum estado de aceitação. A ocorrência de uma escolha com os mesmos símbolos de entrada (*string y*) pode fazer com que se chegue a um estado de não aceitação ou até mesmo de não levar a nenhum estado absoluto (estados mortos), mas isso não impede *y* de ser aceito pelo AFND (HOPCROFT; ULLMAN; MOTWANI, 2002).

Portanto, uma linguagem aceita por um AFND é o conjunto de *strings y* em um conjunto finito de símbolos de entrada (incluindo o vazio), tais que o resultado da função de transição aplicada a partir de seu estado inicial junto com este conjunto de *strings*, resulte em pelo menos um estado de aceitação (HOPCROFT; ULLMAN; MOTWANI, 2002).

2.1.3 Equivalência entre AFD e AFND

O conjunto dos autômatos finitos determinísticos é equivalente ao conjunto dos autômatos finitos não determinísticos. Prova disso é que a partir de um AFND qualquer é possível construir um AFD capaz de realizar as mesmas operações, ou seja, em outras palavras, o AFD irá simular o AFND. A idéia principal do algoritmo que implementa tal transformação é a construção de estados no AFD que simulem as diversas combinações de estados alternativos presentes no AFND. O processo inverso, de construir um AFND a partir de um AFD é a aplicação da forma reversa desta transformação (MENEZES, 2005).

Entretanto, nos piores casos de transformação o menor AFD pode conter 2^n estados, enquanto o menor AFND para a mesma linguagem tem apenas *n* estados. O que pode provocar uma explosão de estados quando a transformação for realizada num AFND com número elevado de estados. Mas felizmente na prática um AFD tem quase tantos estados quanto um AFND equivalente, apesar de ter com freqüência mais transições (HOPCROFT; ULLMAN; MOTWANI, 2002).

2.1.4 Determinismo X Não Determinismo

Na maioria das vezes é mais fácil desenvolver um autômato finito não-determinístico do que um determinístico. A conclusão determinística não é vulgar e acaba resultando num autômato com um número relativamente grande de estados. Dessa forma, na maioria dos casos é preferível construir inicialmente uma solução não-determinística e sobre esta solução aplicar o algoritmo para transformação de um AFND em um AFD (MENEZES, 2005).

3 REPRESENTAÇÃO DE AUTÔMATOS FINITOS

Entre as formas de representação para autômatos finitos, o diagrama de estados e a tabela de transição de estados são empregadas com maior freqüência durante o ciclo acadêmico e na implementação de algoritmos para reconhecedores de linguagens.

3.1 DIAGRAMA DE ESTADOS

Uma das maneiras de se representar um autômato finito é por meio do diagrama de estados ou diagrama de transição. Trata-se de um grafo direcionado e rotulado, que segundo Menezes (2005) apresenta as seguintes características:

a) estados são nodos representados graficamente por círculos e devidamente identificados por um símbolo pertencente ao conjunto dos estados possíveis de um autômato;

b) transições são representadas graficamente por arestas (veja figura 2), responsáveis pela ligação entre os nodos. Cada transição deve possuir um rótulo com caractere pertencente ao alfabeto dos símbolos de entrada;

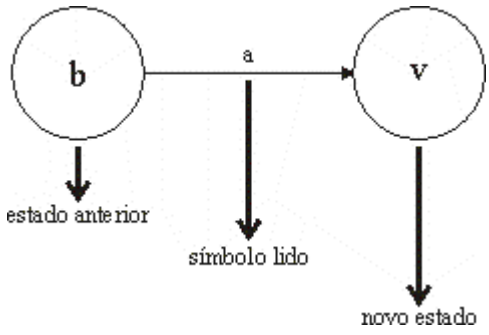

Figura 2. Diagrama (AFD): transição

c) a representação dos estados finais e iniciais ocorre de forma distinta dos demais nodos, como ilustra a figura 3, onde o inicial é identificado por uma seta que não possui origem em nenhum nó e os finais são representados por círculos duplos;

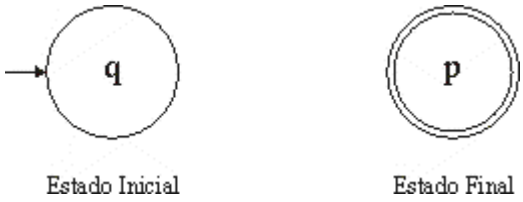

Figura 3. Diagrama (AFD): estado inicial (esquerda) e final (direita)

d) transições paralelas são aquelas que apresentam o mesmo nodo de origem e destino e podem ser representadas graficamente de duas formas conforme ilustra a figura 4.

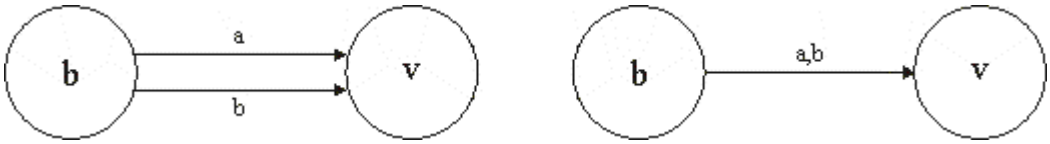

Figura 4. Diagrama (AFD): representações alternativas para transições paralelas

Esse tipo de representação ajuda a identificar a presença de estados inacessíveis ou mortos por meio de uma simples visualização.

3.2 TABELA DE TRANSIÇÃO DE ESTADOS

Uma outra maneira de representar um autômato finito é com a utilização da tabela de transição de estados ou fórmula tabular. Nesta tabela as linhas representam os estados, sendo que o estado inicial recebe uma seta e os finais recebem o caractere

asterisco, enquanto que as colunas representam o conjunto do alfabeto dos símbolos de entrada. O cruzamento das linhas com as colunas da tabela irá indicar as possíveis transições de estado (HOPCROFT; ULLMAN; MOTWANI, 2002).

Para exemplificar esse tipo de representação, tem-se um autômato finito determinístico representado por meio de diagrama de estados, conforme está ilustrado na figura 5, onde o mesmo é representado na seqüência em sua forma tabular (Tabela1).

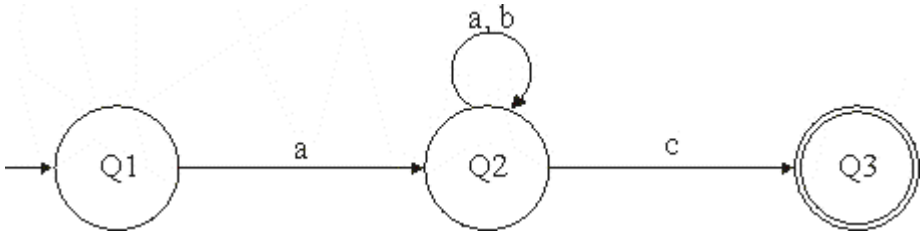

Figura 5. Diagrama de um Autômato Finito Determinístico

Tabela 1. Representação na forma tabular ou tabela de transição de estados

Alfabeto / Estado	**A**	**b**	**c**
→Q1	Q2	-	-
Q2	Q2	Q2	Q3
*** Q3**	-	-	-

Na tabela, as células representadas com o caractere “-“ indicam transições que não são mostradas no diagrama de estados do autômato finito, ou seja, transições para estados de erro (inexistentes) ou transições impossíveis (MENEZES, 2005).

Portanto a forma tabular é uma alternativa muito aplicada para representar uma função programa, presente na maioria dos programas que fazem o reconhecimento de linguagens.

4 ALGORITMO PARA IMPLEMENTAÇÃO DE AUTÔMATOS FINITOS

Existem muitas maneiras de demonstrar um AFD ou um AFND em código. Os métodos que serão apresentados neste trabalho possuem características presentes na maior parte de sistemas que implementam autômatos.

Entretanto deve-se observar que a escolha entre qual tipo de implementação deverá ser empregada, dependerá unicamente da aplicação. E na prática qualquer aplicação deve ser adaptada segundo seus objetivos, onde o que mais ajuda é uma boa filosofia de implementação.

Uma forma de se implementar um autômato finito é conhecida como implementação específica. Nesse tipo de algoritmo, cada estado do autômato finito é representado através de um conjunto de instruções, ou seja, consiste em programar cada estado de um autômato finito. Note que a implementação de um autômato com grande número de estados gera uma complexidade do código, principalmente quando se tem o crescimento do número de estados diferentes ao longo de caminhos arbitrários. Na seqüência, é exemplificado este tipo de implementação para o autômato finito apresentado pela tabela de transição de estados conforme ilustra a Tabela 2[2]:

Tabela 2. Tabela para demonstração da implementação especifica

	TAB	A	b	c	<> a, b, c
(q0)	1	1	2	4	4
(q1)	2	4	2	3	4
(q2)	3	1	4	3	4
(qerro)	4	4	4	4	4

início

[2] FURTADO, Olinto José Varela. **Linguagens Formais e Compiladores**. Apostila de Linguagens Formais e Compiladores da UFSC, 2002. Disponível em: HTTP//WWW.inf.ufsc.br/~olinto/ acessado em: 22 junho 2006.

q0: leia CAR
se CAR = ‘a’
então vá para q0
senão se CAR ≠ ‘b’
então vá para qerro
fim se
fim se

q1: leia CAR
se CAR = ‘b’
então vá para q1
senão se CAR ≠ ’c’
então vá para qerro
fim se
fim se

q2: leia CAR
se CAR = ‘$’
então escreva ‘Seqüência Reconhecida’ pare
senão se CAR = ‘c’
então vá para q2
senão se CAR = ‘a’
então vá para q0
fim se
fim se
fim se

q erro: enquanto CAR ≠ ‘$’ faça
leia CAR
fim enquanto
escreva ‘Seqüência não Reconhecida’

fim[3]

[3] FURTADO, Olinto José Varela. **Linguagens Formais e Compiladores.** Apostila da disciplina de Linguagens Formais e Compiladores da UFSC, 2002. Disponível em: http://www.inf.ufsc.br/~olinto/ acessado em: 22 Junho 2006.

Uma outra forma utilizada para de implementação de autômatos finitos é conhecida por implementação geral. Neste modelo, será preciso além da tabela de transições de estados um vetor para controle dos estados finais. Portanto, aqui ocorre um procedimento genérico no algoritmo, relacionando a tabela de transição em função do conjunto de entrada a ser analisado. Conforme demonstra a Tabela 3 onde se tem uma tabela verdade (indexada pelo estado), que será utilizada para verificar se o estado em que à máquina se encontra é final ou não, determinando deste modo se a seqüência será reconhecida ou não reconhecida[4].

Tabela 3. Tabela para demonstração da implementação geral

	TAB	a	b	C	<> a,b,c	VE	F
(q0)	1	1	2	4	4	1	0
(q1)	2	4	2	3	4	2	0
(q2)	3	1	4	3	4	3	1
(qerro)	4	4	4	4	4	4	0

```
início
    (* inicialização de TAB e VEF *)
    leia EST, CAR  (* estado inicial, próximo caractere *)
    enquanto CAR ≠ '$' faça
        EST:= TAB[EST, CAR]
        leia CAR
    fim enquanto
    se VEF [EST] = 1
            então escreva 'seqüência reconhecida'
            senão escreva 'seqüência não reconhecida'
    fim se
fim
```

[4] FURTADO, Olinto José Varela. **Linguagens Formais e Compiladores.** Apostila da disciplina de Linguagens Formais e Compiladores da UFSC, 2002. Disponível em: http://www.inf.ufsc.br/~olinto/ acessado em: 22 Junho 2006.

Outros métodos de implementação são apresentados por Louden (2004), onde inicialmente ele apresenta um AFD em sua forma gráfica (veja figura 6) e o algoritmo que irá simular esse autômato. Ressaltando que este método é considerado pelo autor como o código mais fácil.

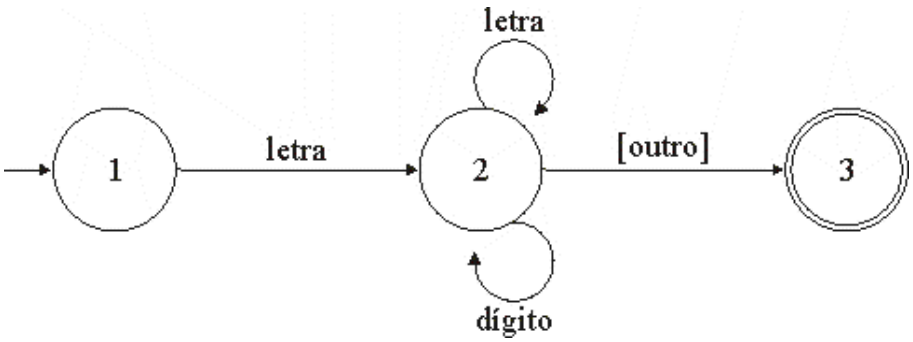

Figura 6. Diagrama de um AFD para demonstração do código de Louden

```
{inicio – estado 1}
se próximo caractere for letra então
    avance entrada;
    {estado 2}
    enquanto próximo caractere for letra ou dígito faça
        avance entrada; {permaneça no estado 2}
    fim enquanto;
    {passa para o estado 3 sem avançar entrada}
    aceitação
senão
    {tratamento para erro ou outros casos}
fim se;
```

Código como este tem sido utilizado para escrever sistemas de varredura pequenos, entretanto esse método apresenta dois problemas. O primeiro é que lê *ad hoc*, ou seja, cada autômato finito determinístico necessita ser tratado ligeiramente diferente, o que torna difícil construir um algoritmo para demonstrar todos os AFDs em código dessa forma. O segundo problema é a complexidade que o código irá adquirir

com o crescimento do número de estados diferentes ao longo de caminhos arbitrários (LOUDEN, 2004).

Um método de implementação consideravelmente melhor faz uso de uma variável para manter o estado corrente e escreve as transições como declarações dentro de estruturas do tipo *case*, onde o primeiro caso testa o estado atual e a segunda condição testa o caractere de entrada, dado o estado (LOUDEN, 2004). Exemplificando, o AFD presente na figura 6 pode ser traduzido no seguinte código:

```
estado := 1; {início}
enquanto estado = 1 ou 2 faça
    caso estado de
        1: caso caractere de entrada de
            letra: avance entrada;
                estado := 2;
            senão estado : = ...{erro ou indefinido};
            fim caso;
        2: caso caractere de entrada de
        letra, dígito: avance entrada;
                estado:= 2;
        senão estado := 3;
        fim caso;
    fim caso;
fim enquanto;
se estado = 3 então sentença reconhecida senão erro;
```

O código descrito acima reflete diretamente o AFD, as transições correspondem a atribuições de novos valores (estados) à variável *estado* a ao avanço de entrada (LOUDEN, 2004).

Outro tipo de algoritmo, que não codifica o AFD de forma direta no código e que permite expressar o AFD como uma estrutura de dados e criar um código genérico que capture as ações da estrutura de dados, é a utilização de uma matriz bidimensional

indexada por estados e caracteres do conjunto de símbolos de entrada, e que expressa os valores da função de transição representados em uma tabela de transição conforme ilustra a Tabela 4 (LOUDEN, 2004).

Tabela 4. Tabela de transição utilizada para o algoritmo genérico de AFD

	Alfabeto dos símbolos de entrada (Σ)
Conjunto de Estados (Q)	Estados que representam transições $t\ (\ Q, \Sigma)$

"Nessa tabela, as células em branco representam transições que não são mostradas no diagrama do AFD" (LOUDEN, 2004, p.63). O estado inicial será o primeiro estado listado. Porém, a tabela não indica quais são os estados finais ou estados de aceitação. Assim, a informação de quais estados são de aceitação deverá ser armazenada na mesma estrutura ou em uma estrutura de dados separada, a fim de permitir o reconhecimento de uma sentença (LOUDEN, 2004).

Como exemplo, temos a Tabela 5 com a estrutura que irá armazenar os estados de aceitação, representado novamente o AFD da figura 6.

Tabela 5. Tabela com controle de aceitação de estados

caractere de entrada / Estado	Letra	dígito	outro	*Aceitação*
1	2			não
2	2	2	3	não
3				sim

Dessa forma, com uma estrutura como a da tabela acima, pode-se escrever um código de forma a implementar qualquer AFD. Portanto o esquema de código a

seguir assume que uma matriz de transições *t* armazena todas as transições, indexadas por estados e caracteres de entrada, e que os estados de aceitação são dados na matriz booleana *Aceita* indexada por estados. A matriz booleana *Avance,* indexada também por estados e caracteres de entrada, armazena as transições que avançam a entrada (LOUDEN, 2004).

estado := 1;
CAR := *próximo caractere de entrada*;
enquanto não *Aceita*[*estado*] **e não** *erro*(*estado*) **faça**
    novoestado := *t*[*estado*, *CAR*];
    se *Avance*[*estado*, *CAR*] **então** *CAR* := *próximo caractere de entrada*;
    estado := *novo estado*;
fim enquanto;
se Aceita[estado] **então** *sentença reconhecida* **senão** *erro*;

5 TRABALHOS CORRELATOS

Autômatos finitos vêm sendo aplicados em várias pesquisas do meio acadêmico, especialmente para o reconhecimento de sentenças em linguagens. A seguir serão apresentadas algumas destas pesquisas.

5.1 AGASTUDIO

Na Universidade do Extremo Sul Catarinense (UNESC) tem-se um trabalho de conclusão de curso onde um reconhecedor de linguagens por meio de autômatos finitos, aplica as técnicas da máquina de *Mealy* e máquina de *Moore* para a geração de animações. Na ferramenta o autômato finito é tratado em sua forma tabular e não existe

um estado final. A linguagem Java foi usada para implementação deste software, permitindo a sua utilização em diferentes plataformas (STACHOWOSKI, 2005).

5.2 LANGUAGE EMULATOR

Na Universidade Federal de Minas Gerais (UFMG) tem-se a implementação de um software denominado Language Emulator como trabalho no curso de pós-graduação. O sistema foi desenvolvido para ajudar estudantes de graduação, com o intuito de facilitar o aprendizado das idéias relacionadas à teoria dos autômatos. Nesse sistema, o reconhecimento de uma sentença ocorre por meio de autômatos finitos determinísticos e não determinísticos representados em sua forma tabular, assim como, implementa a máquina de Morre, máquina de Mealy e autômatos finitos não determinísticos com transições lambda. O sistema desenvolvido em linguagem Java (ambiente multi–plataforma) oferece a possibilidade de ser executado em vários sistemas operacionais (VIEIRA; VIEIRA, 2002).

5.3 AFCHECKER

Na Universidade Católica de Pelotas (UCPEL) foi desenvolvido como trabalho na disciplina de linguagens formais e autômatos, um reconhecedor de linguagens por autômatos finitos dos tipos determinísticos, não determinísticos e não determinístico com movimento vazio. A ferramenta foi desenvolvida com uso da linguagem C++ e os autômatos são tratados internamente como grafos, utilizando estruturas com pilhas e listas encadeadas para realizar o reconhecimento da linguagem em estudo (BASTOS; SALVADOR, 2002).

5.4 EDITAB

O EDITAB ou Editor de Autômatos Finitos por Tabela foi desenvolvido na Universidade Federal de Santa Catarina (UFSC). Esse software faz o reconhecimento de sentenças por meio de um algoritmo que implementa o autômato finito em sua forma tabular, ou seja, utilizando uma matriz bidimensional indexada por estados e conjunto dos símbolos de entrada. Essa ferramenta foi desenvolvida para ambiente MS-DOS, limitando-se no aspecto gráfico, e na ausência de um dispositivo apontador o que facilitaria seu uso. O software não apresenta uma função que permita a exportação da tabela gerada para outras linguagens de programação, permitindo apenas sua visualização ou impressão (FURTADO; DELUCCA; KREIS, 1992).

5.5 EDIGRAF

Na Universidade Federal de Santa Catarina foi desenvolvido um Editor Gráfico de Autômatos Finitos denominado EDIGRAF com a finalidade de facilitar no aprendizado dos acadêmicos do curso ciência da computação na disciplina de introdução a compiladores. A ferramenta atua como um reconhecedor de linguagens. O software desenvolvido para ambiente MS-DOS, apresenta uma interface gráfica limitada. A ausência de um dispositivo apontador num programa como este (editor gráfico) torna-se uma desvantagem considerável (FURTADO; DELUCCA; KREIS, 1991).

6 AFLAB – AUTÔMATOS FINITOS PARA O RECONHECIMENTO DE SENTENÇAS.

Este módulo da *Shell* AFLAB é um modelo de software que tem como propósito permitir a criação de autômatos finitos tanto na sua forma tabular como também em sua forma gráfica, para posteriormente realizar o reconhecimento de sentenças.

O sistema foi desenvolvido em linguagem *object pascal*, utilizando para isso a plataforma Borland® Delphi™ Enterprise versão 7.0. Este ambiente de programação foi escolhido pelo fato de já ser utilizado no meio acadêmico, permitindo dessa forma que seu código fonte seja estudado ou até mesmo modificado conforme a necessidade dos acadêmicos, e por possuir a característica de gerar programas bastante leves, não consumindo assim muitos recursos do computador.

A ferramenta tende a auxiliar os acadêmicos nas disciplinas de linguagens formais, compiladores e teoria da computação, pois permite a construção de autômatos finitos determinístico e não determinísticos para o reconhecimento de sentenças.

Sua interface oferece ao usuário a possibilidade de construir o AF em sua forma gráfica ou tabular de forma similar a aplicada nestas disciplinas durante o ciclo acadêmico. Outra opção fornecida pela ferramenta é a de carregar ou salvar os autômatos finitos construídos em arquivos com extensão do tipo INI (*initialization Files*).

6.1 CRIANDO UM AUTÔMATO

Durante a inicialização do aplicativo o usuário pode escolher entre montar o AF em sua forma gráfica ou tabular para iniciar um projeto, para isso é necessário selecionar a aba que corresponde a forma desejada no formulário inicial da ferramenta, conforme ilustrado na figura 7.

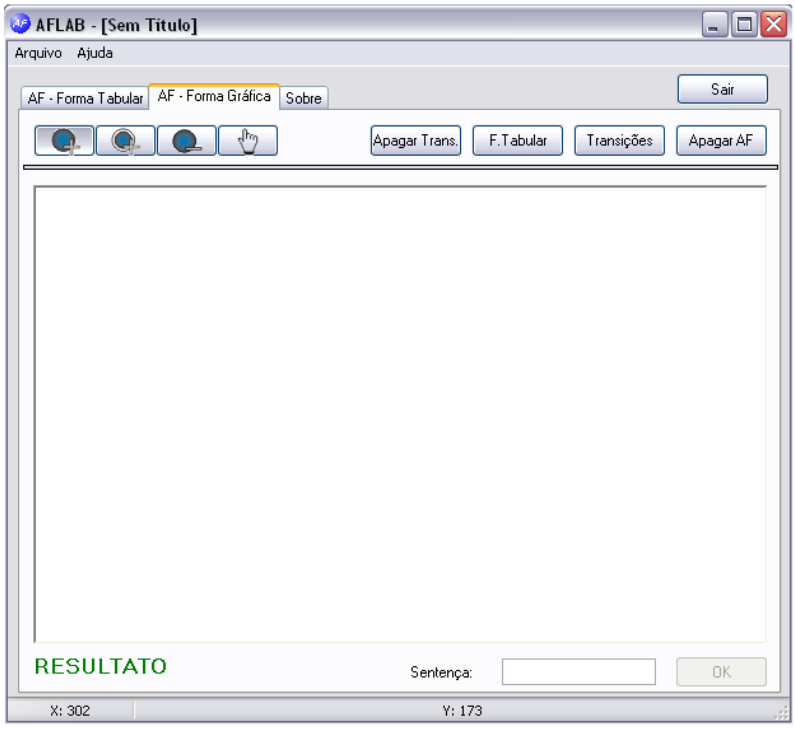

Figura 7. Visualização do Formulário inicial

Feita a escolha da forma para construção do AF, o usuário deverá entrar com os estados, os símbolos de entrada e as transições para o autômato em estudo. Caso a opção de representação escolhida tenha sido a tabela de transições de estados (aba AF - Forma Tabular), o usuário deverá entrar com a nomenclatura dos Estados, Alfabeto, Estado Inicial e Estados Finais nos locais respectivamente apropriados. Na seqüência o botão **transições** deverá ser acionado para liberar a tabela de transições de estados, onde as produções presentes no AF deverão ser devidamente inseridas. Ao

finalizar as entradas, o AF estará construído, pronto para testar as sentenças que deverão ser inseridas no campo de edição denominado **sentença**. Como exemplo tem-se um AF M = ({a,b},{Q0, Q1, Q2}, {*t* (Q0, a) = Q1, *t* (Q0, a) = Q2, *t* (Q0, b) = Q1, *t* (Q1, a) = Q2, *t* (Q1, a) = Q0, *t* (Q1, b) = Q2}, Q0, Q2) construído em sua forma tabular presente na figura 8.

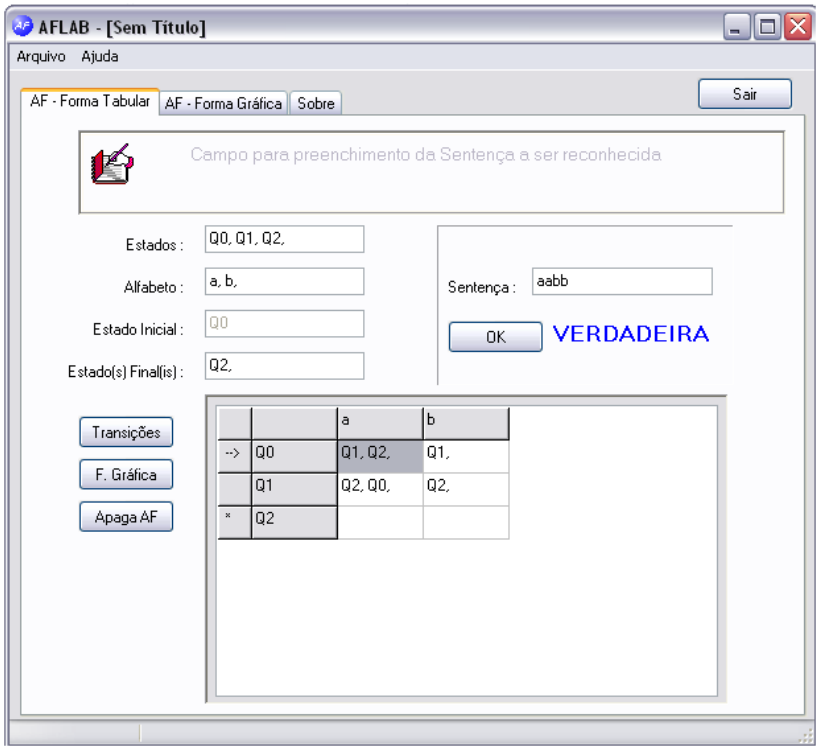

Figura 8. Visualização do Autômato Finito na forma Tabular

Já se o usuário optar por construir o AF em sua forma de Diagrama de Estados, a inclusão dos estados irá ocorrer com o acionamento do botão esquerdo do dispositivo apontador sobre a área destinada a sua construção, ou seja, na aba **AF - Forma Gráfica** da ferramenta. O conjunto de botões localizado na parte superior esquerda desta aba será responsável em determinar se o estado a ser criado (com o evento *click*) será final ou não, conforme o botão que estiver acionado (adicionar estado ou adicionar estado final). Sendo que por *default* o primeiro estado a ser criado será o estado inicial do AF. As transições serão incluídas com o acionamento do botão

Transições, liberando dessa forma o formulário que permite a inclusão de transições apresentado na figura 9.

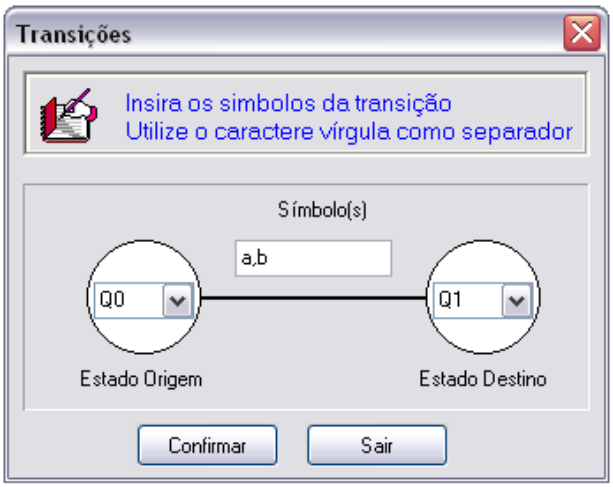

Figura 9. Formulário para Inclusão das Transições no Modo Gráfico

Assim o AF estará construído pronto para o reconhecimento de sentenças. A figura 10 mostra o mesmo AF construído na figura 8 agora modelado em forma de Diagrama de Estados (Forma Gráfica).

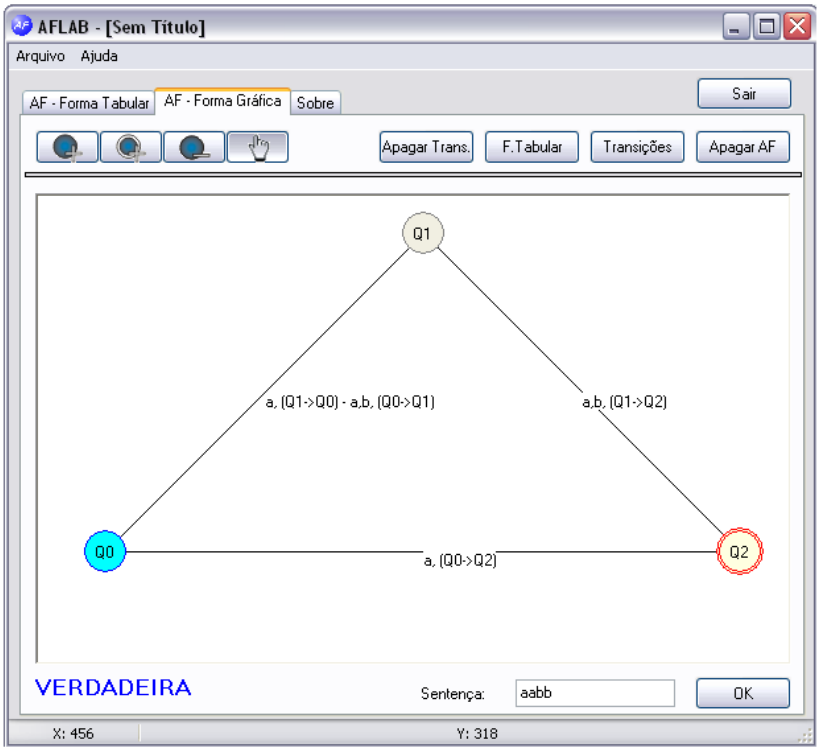

Figura 10. Visualização do Autômato Finito na Forma Gráfica

6.2 DESENVOLVIMENTO

A ferramenta utiliza uma estrutura única para o armazenamento de autômatos finitos, permitindo que este por sua vez seja um autômato finito determinístico ou não determinístico, sem criar restrições. Outra característica é o fato de que esta mesma estrutura irá armazenar todos os AF construídos, independente do usuário ter optado por construí-lo em sua forma gráfica ou na sua forma tabular, permitindo assim que um AF construído em sua forma gráfica seja visualizado em sua forma tabular e o contrário também será aceito.

Desse modo para o armazenamento do AF a estrutura da ferramenta foi dividida basicamente em três classes. A primeira denominada *Testados* é responsável pelo armazenamento do nome (rótulo) de cada estado presente no AF, além disso, esta classe contém os demais atributos de um estado, como se o estado é inicial, final e as coordenadas responsáveis por marcar a posição de tela para cada estado construído em forma gráfica.

Uma segunda classe denominada *Talfabeto* irá armazenar apenas os símbolos de entrada pertencentes ao AF.

Na terceira e última classe chamada de *Tconexao*, tem-se as informações referentes a cada transição pertencente ao AF. Nessa classe três atributos são utilizados, como o estado origem (estado de onde a transição parte), o símbolo de entrada que deve pertencer ao conjunto de alfabetos do AF e que é representado por um único caractere, e o estado destino (estado para o qual a transição aponta). Ressaltando que tanto o estado origem como o estado destino devem pertencer ao conjunto de estados do AF. O código mostrado na figura 11 mostra a parte do algoritmo onde as classes são declaradas juntamente com seus métodos.

```
type
  Testados = class
  nome_estado :string;
  estado_final: Boolean;
  Estado_inicial: Boolean;
  PosX, PosY: integer;
  procedure  NovoEstado(Rotulo,  Efinais,  Einicial:  string  ;  X,  Y:
   integer);
  procedure ApagaEstados;
  procedure DefineEstadoFinal(EstadosFinais: string);
  procedure DefineEstadoInicial(EstadoInicial: string);
  function ContaEstados: integer;
  function VerificaEstadoRepetido(Nome: string): Boolean;
  procedure RemoveEstado(Posicao: integer);
  procedure AtualizaEstadoFinal(Finais: string);
end;
  Talfabeto = class
  letra_alfabeto :string;
  procedure NovoAlfabeto(Rotulo: string);
  procedure ApagaAlfabeto;
  function VerificaAlfabetoRepetido(Simbolo: string): Boolean;
end;
  Tconexao = class
  estado_origem, estado_destino, Caractere :string;
  procedure NovaConexao(Origem, Destino, Rotulo: string);
  procedure ApagaConexoes;
  procedure MostraConexoes(num_conexoes: integer);
  procedure RemoveConexoes(NomeEstado: string);
  procedure ExcluiConexao(PosVet: integer);
  function TestaSentenca(EstadoInicial, Sentenca: string): Boolean;
  function TestaEstadoFinal(Estado: string): Boolean;
  function BuscaProximoEstado(Estado, Alfabeto: string): string;
  function ContaConexoes: integer;
  function   VerificaConexoesRepetidas(EOrigem,   EDestino,   Simbolo:
string): Boolean;
end;
```

Figura 11. Estrutura de Armazenamento

Para o armazenamento dos dados, o tipo de variável utilizada nesta aplicação é a de vetores dinâmicos, permitindo dessa forma que o usuário final não fique restrito quanto a um número fixo de estados, símbolos de entrada ou até mesmo de transições, durante a construção do AF. O trecho de código presente na figura 12 mostra a declaração dos tipos empregados na ferramenta.

```
var
 Estados: array of Testados;
 Alfabeto: array of Talfabeto;
 Conexao: array of Tconexao;
```

Figura 12. Tipos Declarados

Na fundamentação teórica deste projeto encontram-se alguns algoritmos que poderiam ser aplicados nesta ferramenta para realizar o reconhecimento de sentenças, porém nenhum desses algoritmos permite o reconhecimento de sentenças em AFD ou em AFND simultaneamente. Assim nessa parte do projeto surgiu um questionamento, como realizar o reconhecimento de sentenças em AFD e AFND sem realizar a transformação de um AFND em AFD? Visto que todos os algoritmos estudados durante a elaboração da fundamentação teórica só permitem o reconhecimento de sentenças para AFD. Dessa forma durante a implementação da ferramenta foram feitos outros levantamentos bibliográficos e identificou-se a existência de uma técnica que possibilita o reconhecimento de sentenças tanto para um AFD como para um AFND sem a necessidade da transformação do AFND em AFD, técnica essa abordada no livro de Vieira (2006).

Apesar de não disponibilizar o algoritmo, o autor descreve um modelo de estrutura capaz de realizar a tarefa de forma simples e eficiente como é demonstrado a seguir.

Nesse modelo de reconhecimento, uma estrutura similar a uma árvore é construída, onde a base da árvore surge a partir da leitura do primeiro caractere da sentença em conjunto com o estado inicial, gerando a partir dessa combinação um conjunto de estados (novos ramos da árvore). Na seqüência é feita a leitura do próximo caractere que será combinado com cada um dos estados pertencente a esse novo conjunto, gerando um novo conjunto de estados, e assim sucessivamente até a realização da leitura do último caractere da sentença.

Ao final, para identificar se essa sentença pertence ou não a linguagem em estudo, basta verificar se no ultimo conjunto de estados gerados se tem pelo menos um estado final, nesse caso a sentença será verdadeira, caso contrário será falsa.

A Figura 13 demonstra como a modelagem da árvore ocorre para o reconhecimento da sentença “aabba” no AF construído em sua forma tabular para a Tabela 6, mostrando dessa forma o conjunto final de estados possíveis para este símbolo de entrada.

Tabela 6. AF para demonstrar o reconhecimento de sentença na *shell*

Alfabeto / **Estado**	**a**	**b**
→Q0	Q1	Q2
Q1	Q2	Q0
*** Q2**		Q2, Q1

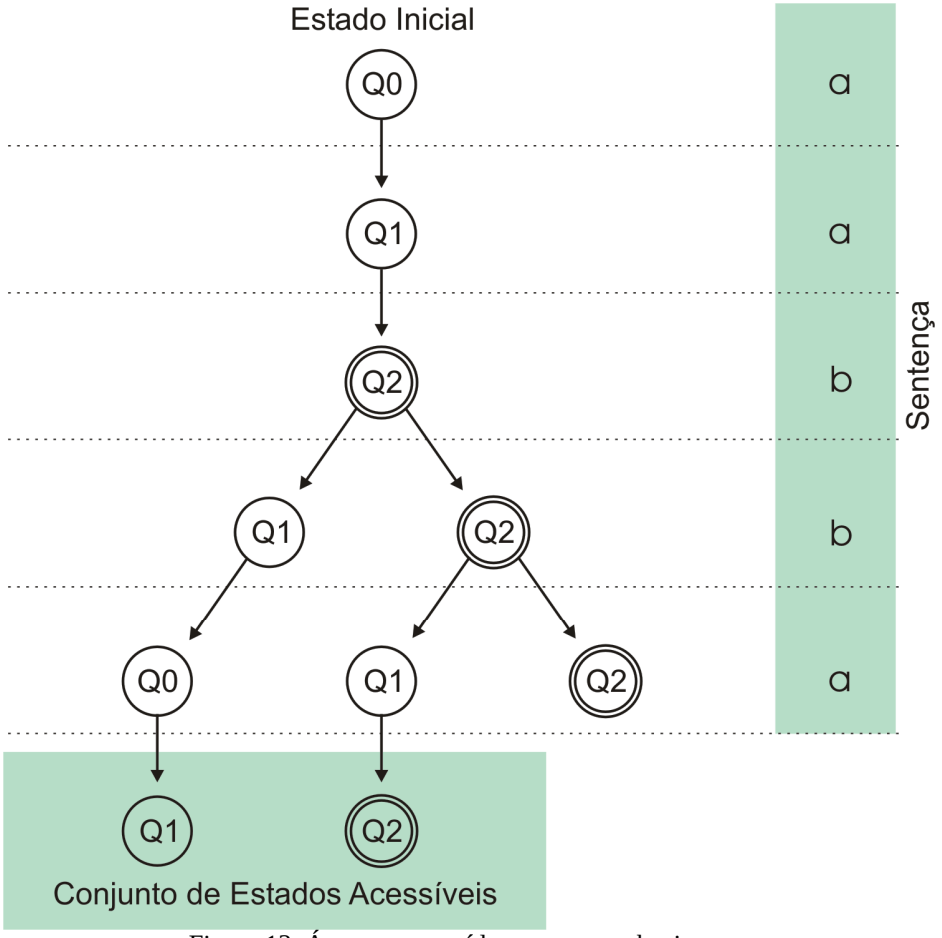

Figura 13. Árvore construída para o reconhecimento

Seguindo esse raciocínio o algoritmo responsável pelo reconhecimento de sentenças foi dividido basicamente em três funções denominadas *TestaSentenca, TestaEstadoFinal e BuscaProximoEstado*.

A função *TestaSentenca* em conjunto com a função *BuscaProximoEstado* são responsáveis pela montagem da árvore propriamente dita, ou seja, durante as suas execuções elas partem do estado inicial e fazem a leitura do primeiro símbolo da sentença gerando um conjunto de estados que por sua vez irá conter os estados possíveis de serem alcançados mediante a transição em questão. A partir desse momento, com o primeiro conjunto de estados acessíveis definido, é feita a leitura do próximo símbolo de entrada e combinado com cada elemento pertencente a este conjunto, gerando um novo conjunto de estados acessíveis. Dessa forma enquanto a função *TestaSentença* realiza a ação de percorre a sentença até chegar ao seu final a função *BuscaProximoEstado* irá simplesmente retornar o conjunto de estados acessíveis a partir da combinação entre um símbolo de entrada e o conjunto de estados que lhe são passados como parâmetros.

Ao final com a leitura de todos os símbolos de entrada tem-se a geração do último conjunto de estados acessíveis. Então a função *TestaEstadoFinal* será chamada passando como parâmetros este último conjunto de estados gerado pela função *BuscaProximoEstado*. Assim, na função *TestaEstadoFinal* ocorre apenas uma leitura de todos os estados que lhe foram passados como parâmetro verificando se existe algum estado final, caso isso ocorra ela irá retornar verdade e no contrário ela irá retornar falso definindo dessa forma se a sentença pertence ou não a linguagem em questão. Nas Figuras 14, 15,16 são demonstrados os trechos do algoritmo onde estas três funções estão declaradas.

```
function   Tconexao.TestaSentenca(EstadoInicial,   Sentenca:   string):
Boolean;
var i, c, cont: integer;
S, S2, S3: string;
begin
  Result := False;
  cont := 0;
  if (length(Conexao) <= 0) then
  begin
     Exit;
  end;
  S2 := Trim(EstadoInicial) + ',';
  S3 := '';
  while Length(Sentenca) >  0 do
  begin
     S := Trim(Copy(Sentenca, 1, 1));
     Delete(Sentenca, 1, 1);
     c := Pos(',', S2);
     if ((EstadoInicial + ',')= S2) and (cont = 0) then
     begin
        cont := 1;
        S2 := BuscaProximoEstado(Copy(S2, 1, c -1), S);
        if Length(Sentenca) = 0 then
        begin
          Result := TestaEstadoFinal(Trim(S2));
          Exit;
        end;
        S := Trim(copy(Sentenca, 1, 1));
        //Delete(Sentenca, 1, 1);
     end;
     c := Pos(',', S2);
     while c > 0 do //Estados
     begin
      if BuscaProximoEstado(Copy(S2, 1, c - 1), S) <> '' then
        S3 :=  Trim(S3) + BuscaProximoEstado(Copy(S2, 1, c - 1), S);
      delete(S2, 1, c);
      c := Pos(',', S2);
     end;
     S2 := Trim(S3);
     S3 := '';
  end;
  Result := TestaEstadoFinal(Trim(S2));
end;
```

Figura 14. Função Testa Sentença

```
function Tconexao.TestaEstadoFinal(Estado: string): Boolean;
var i, c: integer;
S: string;
begin
  c := Pos(',', Estado);
  while c > 0 do
  begin
    S := Trim(Copy(Estado, 1, c - 1));
    Delete(Estado, 1, c);
    for i := 0 to NEstados - 1 do
    begin
      if (S = Trim(Estados[i].nome_estado)) and
(Estados[i].estado_final) then
      begin
        Result := True;
        Exit;
      end
      else
        Result := False;
    end;
    c := Pos(',', Estado);
  end;
end;
```

Figura 15. Função Testa Estado Final

```
function Tconexao.BuscaProximoEstado(Estado, Alfabeto: string):
string;
var i: integer;
S: string;
begin
  S := '';
  for i := 0 to NConexoes - 1 do
  begin
      if (Conexao[i].estado_origem = Estado) and
(Conexao[i].Caractere = Alfabeto) then
        S := Trim(S) + Conexao[i].estado_destino + ',';
  end;
  Result := S;
end;
```

Figura 16. Função Busca Próximo Estado

Na primeira versão do software tinha-se a idéia de facilitar as entradas do usuário, quando o mesmo optasse pela construção de um AF em seu modo tabela de transição de estados (forma tabular). Para isso, imaginava-se que a utilização do componente *combobox* no lugar de campos de edição no preenchimento dos campos

destinados as entradas: *estado inicial, estados finais e transições (tabela),* possibilitariam um melhor entendimento ao usuário e ainda impediria entrada de dados inválidos para estes respectivos campos. Porém, após a implementação, constatou-se que esse modelo de algoritmo apresentava dois defeitos bastante significativos. Primeiro quanto mais estados o AF possuísse, mais linhas o *combobox* apresentaria, uma vez que o sistema deve dar a possibilidade de montar tanto um AFD como um AFND. Lembrando que a geração dos estados possíveis de serem alcançados em AFND é realizado matematicamente com uso da fórmula da combinação simples.

Exemplificando em um AF com 15 estados, para que se possa montar um conjunto com todas as combinações de estados possíveis é necessário realizar a seguinte operação matemática:

$$C_{15}^{1} = \frac{15!}{1! \cdot (15-1)!} + C_{15}^{2} = \frac{15!}{2! \cdot (15-2)!} + \dots + C_{15}^{15} = \frac{15!}{15! \cdot (15-15)!}$$

O que resultará em 32.767 combinações (para apenas 15 estados). Assim o número de linhas presentes no *combobox* facilmente chegaria a números exorbitantes, o que acarretaria em problemas para o usuário durante a escolha dos valores correspondentes para o preenchimento do campo em questão. Outro problema relacionado a este método, seria o tempo de processamento gasto para gerar estas combinações, sem desconsiderar a possibilidade de ocorre *overflow* (estouro de memória) caso o AF construído possuir um grande número de estados.

Dessa forma esse método foi ignorado e um tratamento para validar as entradas do usuário foi empregado na tentativa de evitar erros durante sua execução. Nesse tratamento a validação das entradas para os campos **Estado Inicial** e **Estados Finais** ocorre mediante o preenchimento do campo denominado **Estados**. Assim, com o campo de edição **Estados** devidamente preenchido os estados do AF estarão definidos,

formando o que será chamado de conjunto de estados do AF. Finalizando a validação das entradas, um algoritmo irá tratar os valores empregados para os demais campos (estado inicial e estados finais), só permitindo a entrada valores que pertençam ao conjunto de estados do AF desenvolvido.

Para o ambiente de representação do AF em sua forma gráfica, foi definido que o primeiro estado a ser criado será o estado inicial, e este poderá ser inicial e final simultaneamente desde que o botão estado final esteja ativado. A partir desse momento os demais estados inseridos poderão ser estados normais ou estados finais mediante ao tipo de botão (Estado ou Estado Final) que estiver ativado no momento da inclusão.

Como todos os estados são representados graficamente pela forma geométrica de um circulo, a utilização de cores em seus preenchimentos ajuda o usuário a identificar o tipo de estado construído. Portanto o estado inicial irá receber a cor azul, um estado normal será cinza e um estado final além de ser representado por um duplo circulo recebe a cor vermelha. Quando o estado inicial for também um estado final este será representado por um duplo circulo de cor azul.

Durante a construção de um AF em seu modo de representação gráfica as transições são representadas por meio de um sinal em forma de flecha que indica uma direção (seta), porém a utilização dessa seta para as transições do AF foram ignoradas por duas razões básicas:

A primeira é a necessidade de existir os rótulos indicativos para cada transição não seria dispensada, uma vez que dois estados ligados em ambos os sentidos requerem este tratamento para que o usuário possa identificar cada transição.

Uma segunda razão é o fato de que durante a manipulação do AF é permitido mover cada estado dentro da área destinada para a construção do autômato, assim durante a realização desta ação cada estado e cada aresta (transição) serão

redesenhados na tela e a inclusão de setas tornaria ainda maior o número de operações a serem processadas, o que provocaria em AFs com grande número de transições em um retardo na resposta do dispositivo apontador para o usuário. Por esse motivo as setas não estão presentes na ferramenta.

A solução adotada para este problema foi a representação de cada transição com a utilização de uma linha interligada aos estados conectados devidamente rotulada. Sendo que o rótulo irá informar ao usuário os símbolos de entrada para a transição em questão assim como o sentido da transição, ou seja, o estado de onde ela parte (Estado Origem) e o estado para onde ela aponta (estado Destino).

O sistema, tanto na forma tabular quanto na forma gráfica, ainda apresenta um controle sobre os dados inseridos no campo de edição denominado **Sentença**, onde o algoritmo construído irá validar as entradas mediante o conjunto dos símbolos do alfabeto definidos anteriormente durante a construção do AF. Desse modo somente valores (símbolos de entrada) definidos no AF serão reconhecidos pelo sistema.

Com o sistema desenvolvido, tinha-se a necessidade de verificar sua eficiência e também de tentar identificar seus possíveis erros durante sua execução. Para isso um questionário foi elaborado (Apêndice A) e aplicado junto aos acadêmicos da disciplina de Linguagens Formais na 5ª Fase do curso de Ciência da Computação da UNESC (Universidade do Extremo Sul Catarinense), onde os mesmos tiveram seu primeiro contato com a ferramenta na tentativa de identificar possíveis falhas.

Após uma análise mais detalhada sobre o resultado do questionário aplicado, foi ressaltado pelos entrevistados a necessidade de algumas modificações para a *Shell* AFLAB, como as relacionadas abaixo:

a) após definir um estado inicial, pelo modo de apresentação na forma de tabela de transição de estados, este por sua vez não deveria mais ser apagado a menos que o usuário resolva criar um novo AF;
b) o Autômato Finito criado em forma tabular deve ser o mesmo que o criado na forma gráfica e vice-versa;
c) ao pressionar o botão para apagar o AF construído, uma mensagem de confirmação da exclusão deveria ser exibida ao usuário;
d) permitir a exclusão de uma transição especifica do AF durante a sua construção em modo de diagrama de estados, uma vez que a ferramenta só elimina transições para a forma gráfica mediante a eliminação de um estado

6.3 METODOLOGIA

Neste projeto foi desenvolvida uma interface para facilitar a criação de AFs utilizados para o reconhecimento de sentenças. Para tornar possível o desenvolvimento deste *software* (objetivo geral), e para atingir o objetivo especifico proposto para este trabalho, foi adotada uma metodologia que se iniciou com um levantamento bibliográfico. Na seqüência um estudo mais aprofundado sobre autômatos finitos foi realizado, na busca de algoritmos que permitam o reconhecimento de sentenças por meio de AF, e também na busca da melhor estrutura para o armazenamento do AF, visto que este poderá ser visualizado em duas formas de representação distintas (Tabular e Gráfica).

O próximo passo da metodologia indispensável para facilitar o entendimento das funcionalidades da ferramenta, bem como a seqüência em que as

operações devem ser executadas pelo sistema, foi a modelagem do protótipo por meio de diagrama de modelo de UML. A figura 17 mostra o diagrama de Casos de Uso (Use Case). Neste tipo de diagrama é realizada uma especificação das ações que cada entidade pode realizar dentro do sistema.

Figura 17 Diagrama de Casos de Uso (Use Case)

O diagrama de atividades demonstrado na figura 18, permite identificar como o sistema irá realizar a leitura de uma sentença na busca de seu reconhecimento mediante o AF em estudo.

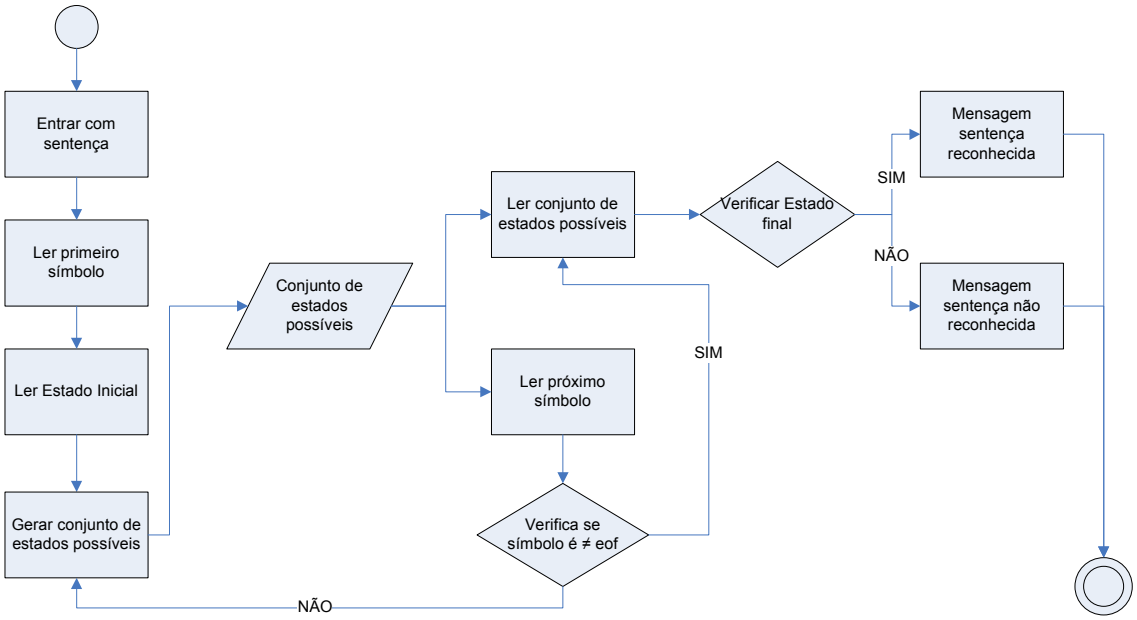

Figura 18. Diagrama de Atividades

Um último diagrama elaborado foi o diagrama de classes representado na figura 19, onde a estrutura de como os dados referentes ao AF serão armazenados no programa é demonstrada.

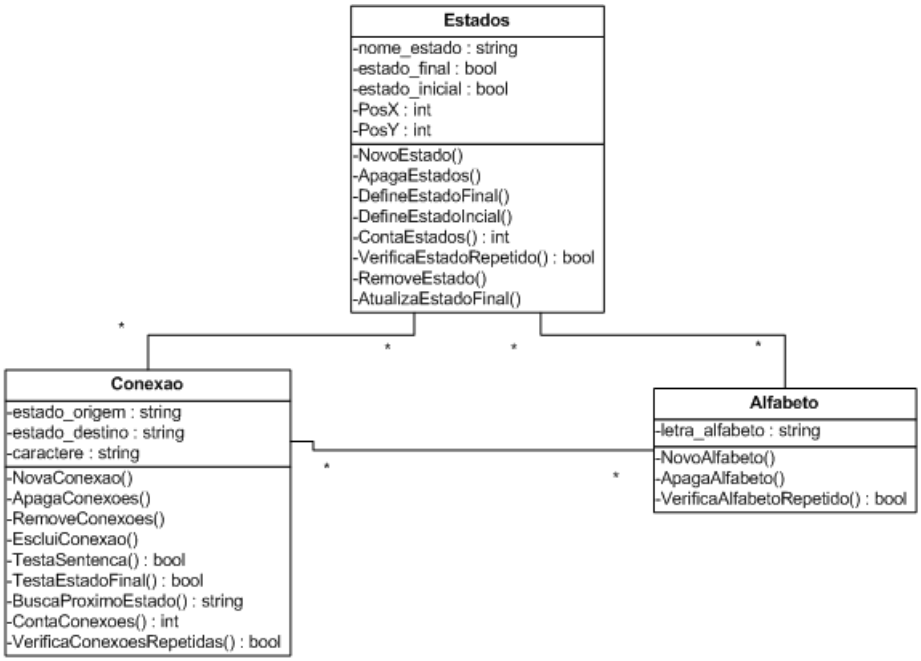

Figura 19. Diagrama de Classes

Na seqüência se deu início a fase de implementação da ferramenta. Nessa fase foi necessário estudar os recursos disponibilizados pela linguagem object pascal, na tentativa de buscar a melhor estrutura para armazenamento e manipulação dos dados. Durante a construção da ferramenta foi aplicado os recursos oferecidos pela técnica de Programação Orientada a Objetos, recursos estes facilitados com a utilização da plataforma Borland® Delphi™ Enterprise.

A opção de utilizar uma única estrutura de armazenamento dos dados do AF surgiu com a necessidade de permitir ao usuário a escolha do tipo de representação para o AF construído. Assim, o usuário irá poder optar por construir o AF em sua forma gráfica e posteriormente querer visualizado em sua forma tabular ou vice-versa e ainda irá poder modificar qualquer AF já construído, no modo de visualização que achar mais apropriado.

Já em sua etapa final, o projeto foi colocado em teste na tentativa de localizar possíveis problemas por meio de um questionário (Apêndice A) aplicado junto aos acadêmicos da disciplina de Linguagens Formais na 5ª fase do curso de Ciência da Computação da UNESC. E finalmente após os testes algumas correções foram realizadas, assim como algumas melhorias na tentativa de transformar o software em um produto confiável.

6.4 RESULTADOS OBTIDOS

Com a realização dos testes finais foi possível constatar que o sistema desenvolvido conseguiu atingir os propósitos apresentados para este trabalho, ou seja, conseguiu cumprir com as metas estabelecidas dentro do objetivo geral e especifico para o projeto. Dessa forma este módulo da *Shell* permite a interação do usuário com

autômatos finitos em suas diferentes formas de representação, proporcionando um feedback imediato para o reconhecimento de sentenças.

7 CONCLUSÃO

A necessidade de softwares direcionados para áreas específicas é uma realidade, não só para propósitos comerciais como também para os fins educacionais na era em que se vivência. Partindo desse principio este software foi elaborado, desenvolvido e a partir desse momento estará disponível para que usuários das mais diversas áreas possam utilizá-lo em seus estudos sobre autômatos finitos.

Como o objetivo geral deste trabalho foi o desenvolvimento do módulo da *Shell* AFLAB responsável pelo reconhecimento de sentenças por meio de autômatos finitos, houve a necessidade de se estudar detalhadamente as técnicas dos algoritmos que realizam tal reconhecimento. Já na fase de implementação um estudo dos recursos oferecidos pela plataforma Borland® Delphi™ Enterprise foi realizado, na tentativa de se encontrar a melhor solução na implementação de cada funcionalidade sugerida para esta ferramenta.

Porém durante o desenvolvimento deste projeto algumas dificuldades foram encontradas. Em um primeiro momento houve a necessidade de compreender a melhor maneira de se aplicar conteúdo estudado para transformá-lo neste documento. Já durante a fase de implementação a maior dificuldade foi a construção do algoritmo capaz de realizar o reconhecimento tanto para AFD como também para AFND sem a necessidade de transformações. Um obstáculo a ser superado, não menos complexo do que os outros enfrentados durante o desenvolvimento deste projeto foi a escassez de material publicado para esta área da Ciência da Computação.

Dessa forma, com todos os estudos realizados foi possível atingir o objetivo proposto para este trabalho, desenvolvendo a ferramenta que permite o reconhecimento

de sentenças tanto para um AFND como também para AFD sem que ocorra a transformação de um AFND em AFD, ou sejam feitas minimizações.

O uso desse módulo da *Shell* AFLAB durante os testes com os acadêmicos teve um nível elevado de aprovação. Para os estudantes que participaram do teste, o fato de poderem verificar se os trabalhos realizados no papel estavam corretos foi de grande valia. O teste realizado ainda permitiu a inclusão de alguns recursos como os relatados no capítulo 6, na tentativa de facilitar a interação com o usuário.

Uma das finalidades do projeto é a de despertar o interesse dos acadêmicos para esta área da Ciência da Computação. Portanto, como trabalhos futuros sugere-se a expansão da *Shell* AFLAB apresentada neste trabalho com a inclusão de recursos como permitir a transformação de um AFND em um AFD, minimização de estados, autômatos finitos com saídas, geração de gramáticas regulares a partir da AFs e a geração de AFs a partir de uma expressão regular ou gramática regular.

REFERÊNCIAS

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

BASTOS, Eduardo; SALVADOR, Otávio. **AFCHECKER**. Trabalho na disciplina de Linguagens Formais. Local – UCPEL, 2002. Disponível em http://afchecker.codigolivre.org.br/licenca.php. Acessado em 20 junho 2006.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Tradução
de Fábio Freitas da Silva. Rio de Janeiro: Campus, 2000.

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

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

FURTADO, Olinto J. Varela; DE LUCCA, José Eduardo; KREISS, Rodrigo. **EDITAB**: editor de autômatos finitos por tabela. Trabalho na disciplina de Introdução a Compiladores. Local – UFSC, 1992. Disponível em http://www.inf.ufsc.br/~olinto/publica.htm. Acessado em 20 junho 2006.

FURTADO, Olinto J. Varela; DE LUCCA, José Eduardo; KREISS, Rodrigo. **EDIGRAF**: editor gráfico de autômatos finitos. Trabalho na disciplina de Introdução a Compiladores. Local – UFSC, 1991. Disponível em http://www.inf.ufsc.br/~olinto/publica.htm. Acessado em 20 junho 2006.

GERSTING, Judith L. **Fundamentos matemáticos para a ciência da computação**. 3.ed Rio de Janeiro: LTC, 1995.

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

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

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

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. **Implementação de linguagens de programação:** compiladores. Porto Alegre: Sagra Luzzatto, 2000.

STACHOVOSKI, Lislaine. **AGASTUDIO**: ambiente de criação e manipulação de animações baseadas em autômatos finitos. Trabalho de Conclusão de Curso em Ciência da Computação. Local – UNESC, 2005. Disponível em http://dcc.unesc.net/sulcomp/05/artigos.htm. Acessado em 22 junho 2005.

VIEIRA, Luiz Filipe Menezes; VIEIRA, Marcos Augusto Menezes. **LANGUAGE EMULATOR.** Trabalho no Curso de Pós Graduação. Local – UFMG, 2002. Disponível em http://homepages.dcc.ufmg.br/~lfvieira/ftc.html. Acessado em 20 maio 2006.

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

APÊNDICE A - QUESTIONÁRIO AFLAB

1 - Como você avalia o processo de Instalação da ferramenta?
( ) Ótimo ( ) Bom ( ) Regular ( ) Péssimo

2 - Qual a facilidade em construir um autômato finito com uso da ferramenta no modo forma tabular?
( ) Ótima ( ) Boa ( ) Regular ( ) Péssima

3 - E para construir um autômato finito no modo forma gráfica?
( ) Ótima ( ) Boa ( ) Regular ( ) Péssima

4 - Você conseguiu identificar com clareza, como deveria ser a entrada dos componentes do autômato (estados, símbolo de entrada, estado inicial, estado final e transições) no modo forma tabular?

( ) Sim ( ) Não

5 - Caso a resposta tenha sido **não** descreva onde você encontrou maior dificuldade:
_____
_____
_____
_____

6 - Você conseguiu identificar com clareza, como deveria ser a entrada dos componentes do autômato (estados, símbolo de entrada, estado inicial, estado final e transições) no modo forma gráfica?

( ) Sim ( ) Não

7 - Caso a resposta tenha sido **não** descreva onde você encontrou maior dificuldade:
_____
_____
_____
_____

8 - Você conseguiu identificar com clareza, como deveria ser a entrada da sentença a ser analisada tanto no modo forma tabular quanto no modo forma gráfica?

( ) Sim ( ) Não

9 - Caso a resposta tenha sido **não** descreva onde você encontrou maior dificuldade:
_____
_____

_____
_____

10 - De modo geral, o processo de montar um autômato finito e fazer o reconhecimento de uma sentença com uso da Shell AFLAB, pode ser avaliada como:

( ) Ótima ( ) Boa ( ) Regular ( ) Péssima

11 - Em sue opinião, o uso da ferramenta durante o ciclo acadêmico pode ser avaliado como:

( ) Ótimo ( ) Bom ( ) Regular ( ) Péssimo

12 - A ajuda presente na ferramenta pode ser considerada?

( ) Ótima ( ) Boa ( ) Regular ( ) Péssima

13 - Faça as suas considerações em relação à ferramenta :

_____
_____
_____
_____
_____