FAFIMAN – FUNDAÇÃO FACULDADE DE FILOSOFIA, CIÊNCIAS E LETRAS DE MANDAGUARI
DEPARTAMENTO DE INFORMÁTICA

RAFAEL CASSOLATO DE MENESES

CONSTRUÇÃO DE UMA FERRAMENTA MULTIPLATAFORMA PARA CRIAÇÃO E TESTE DE MODELOS FORMAIS

MANDAGUARI
Estado do Paraná
2007

RAFAEL CASSOLATO DE MENESES

CONSTRUÇÃO DE UMA FERRAMENTA MULTIPLATAFORMA PARA CRIAÇÃO E TESTE DE MODELOS FORMAIS

Trabalho de Conclusão como requisito para obtenção do grau de bacharel, no Curso de Ciência da Computação da Fundação Faculdade de Filosofia, Ciência e Letras de Mandaguari, orientado pelo Prof. Ms. Yandre Maldonado e Gomes da Costa.

MANDAGUARI
Estado do Paraná
2007

RAFAEL CASSOLATO DE MENESES

CONSTRUÇÃO DE UMA FERRAMENTA MULTIPLATAFORMA PARA CRIAÇÃO E TESTE DE MODELOS FORMAIS

Trabalho de Conclusão como requisito para obtenção do grau de bacharel, no Curso de Ciência da Computação da Fundação Faculdade de Filosofia, Ciência e Letras de Mandaguari, pela seguinte banca examinadora:

Aprovado em 1 de Dezembro de 2007.

_____

Prof. Ms. Yandre Maldonado e Gomes da Costa - FAFIMAN

-Assinatura-

_____

Prof. Ms. Flávio Uber - FAFIMAN

-Assinatura-

_____

Prof. Ms. Heloíse Manica Paris Teixeira - FAFIMAN

-Assinatura-

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado uma vida maravilhosa e uma família lida, também aos meus amigos e colegas de sala. Ao professor Yandre e todos os professores da FAFIMAN, vou ficar com muitas saudades de todos.

Também ao pessoal da nossa república, pois entre brigas e discussões, passamos por todo esse tempo de faculdade juntos.

"*A felicidade não está em quantas bocas você beija, mas em quantas pessoas você ama.*"

Antônio Henrique(Antôoooonioooo)

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AFD	–	Autômato Finito Determinístico
AFMV	–	Autômato Finito Com Movimentos Vazios
AFND	–	Autômato Finito Não Determinístico
AP	–	Autômato com Pilha
ER	–	Expressões Regulares
GLC	–	Gramática Livre de Contexto
JFLAP	–	*Java Formal Language and Automata Package*
LER	–	Linguagens Enumeráveis Recursivamente
LLC	–	Linguagens Livres de Contexto
LR	–	Linguagens Regulares
LSC	–	Linguagens Sensíveis ao Contexto
MT	–	Máquina de Turing
SCTMF	–	Sistema para Criação e Teste de Modelos Formais
SIMFOR	–	Simulador de Modelos Formais

ÍNDICE

RESUMO

Este trabalho apresenta o projeto e a implementação parcial do SCTMF (Sistema para a Criação de Teste de Modelos Formais), um ambiente de software multiplataforma para a criação de modelos formais estudados em Teoria da Computação e realização de testes que permitam que o usuário acompanhe a dinâmica de funcionamento dos mesmos. São abordados também detalhes inerentes a modernas técnicas de Engenharia de Software empregadas no desenvolvimento desta ferramenta.

O diferencial desta ferramenta em relação a outras já desenvolvidas, é o fato de ela ser uma ferramenta multiplataforma e com uma arquitetura com estilo "plugável", isto é, os modelos formais são desenvolvidos separadamente e são incorporados a essa arquitetura.

O sistema proporciona aos usuários a criação e teste dos seguintes modelos formais: Autômato Finito Determinístico, Autômato Finito Não Determinístico, Autômato Finito com Movimentos Vazios, Expressões Regulares, formalismos estes pertencentes às linguagem regulares; Autômato com Pilha, Gramática Livre de Contexto, pertencentes às linguagens livres de contexto e finalmente Máquina de Turing, modelo formal mais genérico da teoria da computação, muito utilizado nos estudos de computabilidade. Além da opção de teste de cadeias de símbolos, outro fato importante e que merece destaque nessa ferramenta é a funcionalidade de abrir e salvar os modelos formais criados.

INTRODUÇÃO

Este trabalho apresenta o desenvolvimento de um sistema para proporcionar a criação e a realização de testes de tais modelos formais estudados na disciplina de Teoria da Computação. Modelos estudados que permitem a representação de linguagens formais, bastante úteis no processo de desenvolvimento de tradutores de linguagens computacionais, como os compiladores e interpretadores.

É grande a dificuldade dos alunos que estudam estes modelos formais em assimilar os conteúdos, já que os mesmos, além de muito abstratos, possuem um forte caráter matemático. Acredita-se que o desenvolvimento de um sistema que permita a criação e realização de testes sintáticos com estes modelos, possa favorecer o ensino de tais conceitos.

A ferramenta aqui proposta tem por objetivo melhorar a compreensão do funcionamento dos modelos formais. Para seu desenvolvimento foi utilizado o paradigma da programação orientada a objetos e o uso de alguns padrões de projeto (*design patterns*), possibilitando assim uma maior flexibilidade na arquitetura da aplicação, a qual também proporciona ao sistema a funcionalidade de abrir e salvar os modelos formais já criados.

Para desenvolvimento da ferramenta, foi utilizada a linguagem JAVA na sua versão 6.0, e para distribuição da aplicação, foi utilizada a tecnologia *Java Web Start*, mantendo assim para os usuários da ferramenta, sempre a versão mais atualizada para a utilização.

O que diferencia esse sistema dos outros já desenvolvidos, como a JFLAP(2007) e o SIMFOR(2002), além da característica da ferramenta ser multiplataforma e ser disponibilizada através do ambiente *Java Web Start*, pode-se dar ênfase à arquitetura da aplicação, arquitetura na qual possibilita que outros desenvolvedores também criem seus modelos formais e acoplem a aplicação.

O Presente trabalho está organizado da seguinte forma:

- No capítulo 1, são apresentados os conceitos básicos das linguagens formais e as suas definições, possui um texto simples para as definições desses conceitos.
- No capítulo 2, são apresentadas as classes de linguagem que compõem a hierarquia de Chomsky e também alguns de seus modelos formais que compõem suas linguagens. A hierarquia de Chomsky é composta pelas Linguagens Regulares ou Tipo 3, pelas Linguagens Livre de Contexto ou Tipo 2, também pelas Linguagens Sensíveis ao Contexto ou Tipo 1, e finalmente pelas linguagens Enumeráveis Recursivamente ou Tipo 0. Para as linguagens regulares são apresentados os seguintes modelos formais: Autômato Finito

Determinístico; Autômato Finito Não Determinístico; Autômato Finito Com Movimentos Vazios; Expressões Regulares. Para as Linguagens livres de Contexto são apresentados o Autômato com Pilha e a Gramática Livre de Contexto. E finalmente para as Linguagens Enumeráveis Recursivamente, a Máquina de Turing.

- No Capítulo 3, é apresentado o desenvolvimento e a modelagem da ferramenta, no primeiro subitem é apresentado um resumo sobre os padrões de projeto (*desing patterns*), no segundo é apresentada a aplicabilidade desses padrões e finalmente é descrita a arquitetura da ferramenta no último subitem desse capítulo.
- No Capítulo 4, é apresentada a ferramenta e seus módulos, juntamente com algumas imagens das telas dos módulos implementados.
- No Capítulo 5, é apresentada uma sessão de trabalhos correlatos e também é apresentada uma tabela comparativa entre o SCTMC e esses trabalhos.

CAPÍTULO 1 – CONCEITOS DE LINGUAGENS FORMAIS

Neste capitulo são apresentados os conceitos básicos das linguagens formais e as suas definições, possuindo uma curta definição de tais conceitos.

MENEZES (1998, p.1) afirma

> ***"Teoria das Linguagens Formais foi originariamente desenvolvida na década de 1950 com o objetivo de desenvolver teorias relacionadas com as linguagens naturais. Entretanto, logo foi verificado que esta teoria era importante para o estudo de linguagens artificiais e, em especial, para as linguagens originárias na Ciência da Computação."***

1.1 SÍMBOLO

MENEZES (1998, p.21) afirma

> ***"Um símbolo ou caractere é uma entidade abstrata básica a qual não é definida formalmente, ou seja, um símbolo é todo elemento pertencente a um alfabeto. Geralmente, descreve-se um símbolo através de letras ou dígitos numéricos."***

1.2 ALFABETO

Um alfabeto é um conjunto finito e não vazio de símbolos (ou letra) de tamanhos finitos. Dado um alfabeto Σ, *a é um símbolo (ou letra) de* Σ *se a* $\in$ Σ.

Por exemplo, os seguintes conjuntos são alfabetos:

- {1,2}
- {aa,bb}
- {x,y,z}

Embora possamos chamar de letra cada um desses símbolos, eles não precisam ter necessariamente um único caractere, nem o mesmo número de caracteres. A única restrição existente é que o número de caracteres que compõem o símbolo seja finito.

1.3 CADEIA (PALAVRA) e FECHAMENTO

Uma cadeia é uma seqüência de símbolos formada a partir de um único alfabeto.

A cadeia poder se nula (ou vazia). Esta cadeia é um caso especial, geralmente denotada por ε, tem tamanho igual a zero e pode ser definida a partir de qualquer alfabeto. Pode-se dizer também que uma sentença vazia é uma palavra sem símbolo.

Exemplo: dado o alfabeto Σ= {a, b} tem-se que *a*, *aa*, *b*, *bb*, *ab*, *ba*, *bbb* são cadeias que podem ser formadas a partir deste alfabeto.

Se Σ representa um alfabeto, então Σ^* (Fechamento completo) denota o conjunto de todas as palavras possíveis formadas sobre Σ. Analogamente, Σ^+ (Fechamento positivo) representa o conjunto de todas as palavras sobre Σ excetuando-se a palavra vazia, ou seja, $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

1.4 LINGUAGEM

Uma linguagem é um conjunto de cadeias formadas a partir de um mesmo alfabeto, ou seja, dado o alfabeto Σ, *o conjunto de palavras L é uma linguagem sobre Σ, se* $L \subseteq \Sigma^*$.

Exemplo:

Dado o alfabeto Σ={a, b}, L={a, b, aa, ab, ba, bb} é uma linguagem formada sobre este alfabeto;

1.5 FORMALISMOS GERADORES (Denotacionais)

Através de um formalismo gerador ou denotador, as sentenças de uma linguagem podem ser sistematicamente geradas. As gramáticas são modelos geradores, onde a linguagem é obtida pela geração das cadeias através da aplicação das leis de formação.

De modo geral, gramáticas são conjuntos de regras que especificam a sintaxe da linguagem impondo uma estrutura às suas sentenças. As regras ou leis de formação são chamadas de produções. Pode-se citar como exemplos de formalismo gerador, as gramáticas e as expressões regulares.

1.6 FORMALISTAMOS RECONHECEDORES (Axiomáticos)

Um formalismo reconhecedor ou axioma verifica se uma determinada sentença pertence ou não a uma determinada linguagem.

Ao contrário das Gramáticas, que geram sentenças de uma linguagem, os reconhecedores são dispositivos de aceitação de cadeias.

Como exemplo de formalismo reconhecedor, pode-se citar os autômatos.

1.7 HIRARQUIA DE CHOMSKY

NETO (1987), afirma que:

> "Noam Chomsky, em 1959, desenvolveu a Hierarquia de Chomsky através de modelos matemáticos para as gramáticas juntamente com seus estudos das linguagens naturais, quando concluiu que facilitaria se houvesse uma classificação na qual dividiria-se as linguagens em quatro classes diferentes. Conseqüentemente a separação das linguagens em classes diferentes levaria a uma separação das gramáticas e reconhecedores em classes correspondentes, onde cada uma é caracterizada pela imposição de diferentes restrições nas produções das gramáticas que as definem" .

Noam Chomsky definiu as classes listadas na figura 1 como (potenciais) modelos para linguagens naturais.

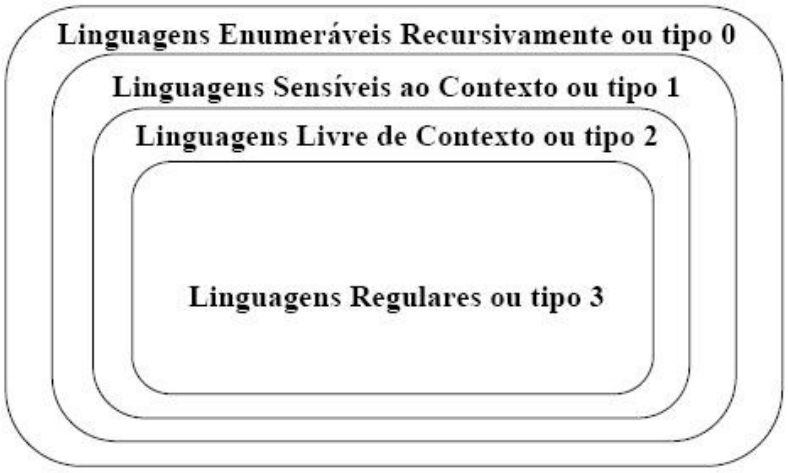


Figura 1 – Hierarquia de Chomsky.

Um formalismo com poder para representar uma linguagem enumerável recursivamente, poderia representar qualquer linguagem presente em qualquer uma das outras categorias.

Pode-se observar que todas as outras categorias estão contidas nas Linguagens Enumeráveis Recursivamente, ou seja, $\text{tipo}_3 \subset \text{tipo}_2 \subset \text{tipo}_1 \subset \textit{tipo}_0$.

CAPÍTULO 2 – MODELOS PARA A DEFINIÇÃO DE LINGUAGENS

Neste capítulo são apresentadas as linguagens que compõem a hierarquia de Chomsky, demonstrada na figura 1, e também alguns de seus modelos formais cujo compõe as suas linguagens. Esta hierarquia é composta pelas Linguagens Regulares ou Tipo 3, pelas Linguagens Livre de Contexto ou Tipo 2, também pelas Linguagens Sensíveis ao Contexto ou Tipo 1, e finalmente pelas linguagens Enumeráveis Recursivamente ou Tipo 0.

2.1 Linguagens Regulares ou Tipo 3.

MENEZES (1998, p.32) afirma

> ***"De acordo com a Hierarquia de Chomsky, trata-se da classe de linguagens mais simples, sendo possível desenvolver algoritmos de reconhecimento ou de geração (Expressão Regular) de pouca complexidade, grande eficiência e fácil implementação."***

Entretanto, por ser uma classe relativamente simples, é restrita e limitada.

Para mostrar que uma determinada linguagem é Regular, é suficiente representá-la usando o AFD, AFND, Expressão Regular ou Gramática Regular.

2.1.1 Autômato Finito Determinístico (AFD)

Um AFD é Finito, pelo fato de o seu número de estados envolvidos no sistema ser finito. Além disso, é dito Determinístico pelo fato de que para dada uma cadeia $x \in L(A)$, só existe uma única seqüência de estados no autômato finito determinístico para processá-la. Este modelo formal pertence ao conjunto dos formalismos reconhecedores, ou seja, se dada uma cadeia x, ele verifica se essa cadeia pertence à linguagem L(Y).

Para uma cadeia ser aceita, ela deve:

Na definição matemática, pode-se dizer que um AFD é uma quintupla $<\Sigma, S, S_0, \delta, F>$, onde:

Σ é o alfabeto de entrada;

S é um conjunto finito não vazio de estados;

$\mathbf{S_0}$ é o estado inicial, $S_0 \in S$;

δ é a função de transição de estados, definida $\boldsymbol{\delta: S \times \Sigma \rightarrow S}$;

F é o conjunto de estados finais, F⊆ S.

MENEZES (1998, p.33) afirma que:

> (...) O AFD pode ser visto com uma máquina composta, basicamente distribuída em três partes: Fita, a Unidade de Controle e o Programa ou Função de Transição.
>
> a) Fita: Dispositivo de entrada quem contém a informação a ser processada;
>
> b) Unidade de Controle: Reflete o estado corrente da máquina, possui uma unidade de leitura 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 análise de cadeias do Autômato Finito Determinístico é executada da seguinte forma:

1. A cabeça de leitura é posicionada no primeiro símbolo da cadeia, e S_0 é ativado.
2. O símbolo que se encontra apontado pela cabeça de leitura é lido(*símbolo corrente*);
3. O próximo estado é obtido através do estado corrente e do símbolo corrente, usando a função de transição do autômato, se não for encontrado nenhum estado, o processamento para e a cadeia é rejeitada;
4. A cabeça de leitura move-se um elemento para a direita.
5. O próximo estado torna-se o estado ativo, e o fluxo retorna ao passo 2.
6. Após o ultimo símbolo da cadeia que está sendo analisada, o AFD assume o estado corrente como estado final e verifica se esse estado pertence ao conjunto de estados finais do AFD. Caso pertença, a seqüência é aceita caso não a seqüência é rejeitada.

O algoritmo do processamento de uma cadeia no AFD é apresentado abaixo:

Início

Estado Atual ← Estado Inicial;
Para I variar do Símbolo inicial da fita até o símbolo final Faça
Se Existe δ (Estado Atual, I) Então
Estado Atual ← δ (Estado Atual, I);
Senão REJEITA;
Se Estado Atual é estado final Então

ACEITA;

Senão REJEITA;

Fim.

2.1.2 Autômato Finito Não Determinístico (AFND)

MENEZES (1998, p.39) afirma:

> "O não-determinismo é uma importante generalização dos modelos de máquinas. Mas nem sempre essa facilidade aumenta o poder de reconhecimento de linguagens de uma classe de autômatos, pois qualquer AFND pode ser simulado por um AFD."

O AFND tem como uma das suas características principais, a funcionalidade de ter mais de um estado inicial, ou seja, pode ter vários estados ativos correntes durante o processamento de reconhecimento de uma cadeia, ele assume um conjunto de estados alternativos, como um conjunto de estados alternativos, como se houvesse uma multiplicação da unidade de controle, uma para cada alternativa, processando independentemente, sem compartilhar recursos com os demais estados. Também adota-se como uma das suas principais características, a possibilidade de cadastrar um mesmo estado de origem e um símbolo para vários estados destino, ou seja, as transições δ: S1 x a $\rightarrow$ S2; e δ: S1 x a $\rightarrow$ S3 podem ser cadastradas, diferentemente do AFD, o qual não permitiria.

Pode-se definir matematicamente um AFND como uma quíntupla representada por $<\Sigma, S, S_0, \delta, F>$, onde:

Σ é o alfabeto de entrada;

S é um conjunto finito não vazio de estados;

S_0 é o conjunto de estados iniciais, $S_0 \subseteq S$;

δ é a função de transição de estados, definida **δ: S x $\Sigma \rightarrow 2^Q$**;

F é o conjunto de estados finais, $F \subseteq S$

Pela definição matemática, pode-se notar que o estado inicial pode estar contido ou ser igual à quantidade de estados do AFND.

A análise de cadeias do Autômato Finito Não Determinístico é executada da seguinte forma:

1. A cabeça de leitura é posicionada no primeiro símbolo da cadeia e são ativados os S_0.s;

2. O símbolo que se encontra apontado pela cabeça de leitura é lido (*símbolo corrente*);

3. Os próximos estados ativos são obtidos através do estado corrente e do símbolo corrente, usando a função de transição do autômato, se não for encontrado nenhum estado, o processamento para e a cadeia é rejeitada;

4. A cabeça de leitura move-se um elemento para a direita.

5. O fluxo retorna ao passo 2.

6. Após o ultimo símbolo da cadeia que esta sendo analisada, o AFND assume o estado corrente como estado final e verifica se esse estado pertence aos estados finais dos AFND. Caso sim, a seqüência é aceita caso não a seqüência é rejeitada.

O algoritmo do processamento de uma cadeia no AFND é apresentado :

```
Inicio
    ACEITA ← Falso;
    Estados Ativos ← Estados Iniciais;
    Para i variar do Símbolo inicial da fita até o símbolo final Faça
        Para r variar do Estado Ativo inicial até o Estado Ativo final Faça
            Se existe δ (r, i) Então
                Estado Ativo_Aux ← δ (r, i);
        Esvazia (Estados Ativos);
        Para s variar do Estado Ativo_Aux inicial até Estado Ativo_Aux final
        Faça
            Estado Ativo [s] ← Estado Ativo_Aux [s]
        r ← 1;
    Repita
        Se Estado Ativos [r] é Estado Final Então
            ACEITA← Verdadeiro;
        r ← r +1;
    Até ACEITA ou r > numero de estados ativos
    Se ACEITA Então
        Mensagem ('Cadeia Aceita');
    Senão Mensagem ('Cadeia Rejeitada');
```

Fim.

2.1.3 Autômato Finito com Movimentos Vazios (AFMV)

MENEZES (1998, p.44) afirma :

> "Movimentos vazios constituem uma generalização dos modelos de maquinas não-determinística. Um *movimento vazio* é uma transição sem leitura de símbolo algum da fita. Pode ser interpretado com um *não-determinismo interno* ao autômato o qual é encapsulado, ou seja, excetuado-se por uma eventual mudança de estados, nada mais por 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"

Em comparação com o AFD o AFMV tem somente um estado inicial, mas por ter o caráter não-determinístico do AFND, em suas funções de transição é permitido que além de ser permitido que durante o processamento de teste da seqüência, estejam ativos mais de um estado, também é permitido que um estado ative outro sem a necessidade de existir um símbolo na fita, desde que exista uma transição que permita esse fato.

Pode-se definir matematicamente um AFMV como uma quíntupla representada por $<\Sigma, S, S_0, \delta, F>$, onde:

Σ é o alfabeto de entrada;

S é um conjunto finito não vazio de estados;

$\mathbf{S_0}$ é o conjunto de estados iniciais, $S_0 \in S$;

δ é a função de transição de estados, definida $\boldsymbol{\delta}$**: S x (Σ ∪ {λ}) → 2^Q**;

F é o conjunto de estados finais, $F \subseteq S$

2.1.4 Expressões Regulares (ER)

MENEZES (1998, p.50) afirma:

> "As ER's são consideradas adequadas para a comunicação homem x homem e principalmente a comunicação homem x máquina Uma Expressão Regular é definida a partir de conjuntos básicos e operações de concatenação e união, trata-se de um formalismo denotacional, considerado gerador."

Com as Expressões Regulares, podemos representar qualquer linguagem regular.

Uma ER sobre um alfabeto $\sum$, é definida como:

$\varnothing$ é uma ER e denota a linguagem vazia;

ε é uma ER e denota a linguagem contendo exclusivamente a palavra vazia, ou seja, $\{\varepsilon\}$;

Qualquer símbolo x pertencente a Σ é uma ER e denota a linguagem contendo a palavra x, ou seja, $\{x\}$;

Se r e s são ER´s e denotam as linguagens R e S, respectivamente, então:

(r+s) é ER e denota a linguagem $R \cup S$;

(rs) é ER e denota a linguagem $RS=\{uv | u \in R \text{ e } v \in S\}$;

(r*) é ER e denota a linguagem R*;

Na tabela baixo são representados alguns exemplos de Expressões Regulares:

Expressão Regular	**Linguagem Representada**
ba*	Todas as cadeias que iniciam por b, seguido por zero ou mais a
a*ba*ba*	Todas as combinações de a's contendo exatamente dois b's
(a+b)*(aa+bb)	Todas as cadeias que terminam com aa ou bb.

TABELA 1: Exemplos de Expressões Regulares

2.2 Linguagens Livres de Contexto ou Tipo 2.

A classe das Linguagens Livres de Contexto ou Tipo 2 contém propriamente a classe das Linguagens Regulares, tem um formalismo gerador (axiomático) e um reconhecedor (ou operacional).

Os dois modelos formais que podem descrever LLCs são: a Gramática Livre de Contexto cujo tem caráter gerador, e o Autômato com Pilha, têm caráter reconhecedor.

Seu estudo é de fundamental importância, pois além de compreender um universo mais amplo de linguagens quando comparada com as linguagens regulares, os algoritmos reconhecedores e geradores são relativamente simples e possuem uma boa eficiência.

Pode-se verificar exemplos típicos do uso das LLCs em analisadores sintáticos, tradutores de linguagens e processadores de texto em geral.

2.2.1 Autômato com Pilha (AP)

Pode-se dizer que o AP é um autômato cuja estrutura básica é análoga a do Autômato Finito, adicionando uma memória auxiliar tipo pilha (na qual pode ser lida e gravada) e a facilidade do não determinismo.

A pilha qual é adicionada não possui limite de tamanho máximo, ou seja, é de tamanho infinito, onde a pilha contém inicialmente um, um símbolo especial denominado símbolo inicial da pilha. Não existem estados finais e o autômato pára aceitando quando a pilha estiver vazia .

A configuração de um AP é dada por uma tripla <s, x, α> onde s é o estado corrente, x é a cadeia da fita que falta ser processada e α é o conteúdo da pilha, com o topo no início de α, onde AP anda ou move-se de uma configuração para outra através da aplicação de uma função de transição.

Um autômato com Pilha é basicamente composto por cinco partes:

1. *Fita:* análoga ao AFD e ao AFND;
2. *Pilha*: é uma memória auxiliar que pode ser usada livremente para leitura e gravação;
3. *Unidade de Controle*: reflete o estado corrente da máquina, possui uma cabeça de fita e uma cabeça de pilha;
4. *Programa ou Função de Transição*: comanda a leitura da fita, leitura e gravação da pilha e define o estado da máquina.

Pode-se definir matematicamente um AP como uma sêxtupla $<\Sigma,\Gamma,S,S_0,\delta,B>$, onde:

$\mathbf{\Sigma}$ é o alfabeto de entrada do AP;

$\mathbf{\Gamma}$ é o alfabeto da pilha;

S é um conjunto finito não vazio de estados;

$\mathbf{S_0}$ é o estado inicial, onde $S_0 \in S$;

$\mathbf{\delta}$ é a função de transição de estados, definida

$$\mathbf{\delta: S \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \text{conjunto de subconjuntos finitos de } S \times \Gamma;}$$

B é o símbolo da base da pilha, $B \in \Gamma$.

2.2.2 Gramática Livre de Contexto (GLC)

MENEZES (1998, p.86) afirma que:

> ***O nome 'Livre de Contexto' se deve ao fato de representar a mais geral classe de linguagens cuja produção é da forma A → α. Ou seja, em uma derivação, a variação, a variável A deriva α sem depender ('Livre') de qualquer análise dos símbolos que antecedem ou sucedem A ('Contexto') na palavra que está sendo derivada"***

A GLC é definida como uma gramática onde as regras de produção são definidas de forma mais livre que na gramática regular. Definindo-se a gramática G, como G = (V, T, P, S), com restrição de qualquer regra de produção de P é da forma $A \rightarrow \alpha$, onde A é uma variável de V e α é uma palavra de $(V \cup T)^*$, portanto uma GLC é uma gramática onde o lado esquerdo das produções contém exatamente uma variável ou símbolo não terminal.

É através das gramáticas que se pode gerar todas as sentenças da linguagem a qual ela define.

A GLC é definida matematicamente por uma quádrupla ordenada G = (V, T, P, S), onde:

V é um conjunto finito de símbolos não-terminais (ou variáveis); Os elementos de V são símbolos auxiliarem que são criados para permitir a definição das regras da linguagem.

T é um conjunto finito de símbolos terminais disjunto de V; Os símbolos de T são aqueles que aparecem no programa de uma linguagem de programação, é o alfabeto em cima do qual a linguagem é definida.

P é um conjunto finito de pares, denominados regras de produção tal que a primeira componente é palavra de $(V \cup T)^+$ e a segunda componente é palavra de $(V \cup T)^*$; Uma regra de produção (α, β) é representada por $\alpha \rightarrow \beta$.

S é um elemento de V, denominado símbolo inicial (ou símbolo de partida).

É importante ressaltar no processo de derivação que enquanto houver símbolo não-terminal na cadeia em derivação, esta derivação não terá terminado.

2.3 Linguagens Sensíveis ao Contexto ou Tipo 1.

As Linguagens Sensíveis ao Contexto estão contidas propriamente nas Linguagens Enumeráveis Recursivamente, esta pode ser desenvolvida a partir de formalismos geradores,

no caso da Gramática Sensível ao Contexto, e formalismos reconhecedores como a Máquina de Turing com Fita Limitada.

Na Gramática Sensível ao Contexto, o termo "sensível ao contexto" deriva do fato de que o lado esquerdo das produções da gramática pode ser uma palavra de símbolos não terminais e terminais, definindo um "contexto de derivação";

Já na Máquina de Turing a com Fita Limitada, a fita de entrada é de tamanho finito. Um detalhe que merece destaque é o fato de que não é conhecido se a facilidade de não-determinismo aumenta ou não o poder computacional desta, então a sua definição incluir a facilidade do não-determinismo.

Uma gramática sensível ao contexto G é uma gramática onde G = (V, T, P, S) com a restrição que qualquer regra de produção de P é da forma $\alpha \rightarrow \beta$, onde:

1. α é uma palavra de $(V \cup T)^+$;
2. β é uma palavra de $(V \cup T)^*$;
3. $|\alpha| \leq |\beta|$, executando-se, eventualmente, para $\beta=\varepsilon$.

A cada etapa da derivação, o tamanho da palavra derivada não pode diminuir, excetuando-se para gerar a palavra vazia, se esta pertencer à linguagem.

2.4 Linguagens Enumeráveis Recursivamente ou Tipo 0.

As Linguagens Enumeráveis Recursivamente são definidas a partir das Máquinas de Turing.

MENEZES (1998, p.132) define que:

> "Segundo a hipótese de Church, a Máquina de Turing é o mais geral dispositivo de computação, então as a Classe das Linguagens Enumeráveis Recursivamente representa o conjunto de todas as linguagens que podem ser reconhecidas mecanicamente e em um tempo finito."

2.5 Máquina de Turing (MT)

DELAMARO (1998, pg.157) define:

> "A Máquina de Turing é um autômato que permite que em uma classe mais ampla de linguagens sejam definidas, as linguagens de tipo 1 e 0.

> Além de serem reconhecedora de cadeias, pode ser também utilizada como modelo de máquinas para computação de funções, assim, uma Máquina de Turing pode ser vista com uma simples "máquina de computação" servindo por isso modelo para determinar quais funções são computáveis e quais não. Assim as Máquinas de Turing podem também ser utilizadas como base para as medidas da complexidade de algoritmos.
>
> A MT para ser considerada como um modelo formal de procedimento efetivo deve ter uma descrição de algoritmo finita e deve consistir de passos discretos, executáveis mecanicamente em um tempo finito".

A definição de uma Máquina de Turing é:

Uma Máquina de Turing é uma óctupla, MT = (Σ, S, δ, S_0, F, V, β, ¤), onde:

Σ é o alfabeto de símbolos de entrada;

S é conjunto de estados possíveis da máquina o qual é finito;

δ representa as funções de transição (parcial), onde:

$$\delta: Q \times (\Sigma \cup V \cup \{\beta, ¤\}) \rightarrow Q \times (\Sigma \cup V \cup \{\beta, ¤\}) \times \{E, D\};$$

S_0 estado inicial da máquina, $S_0 \subseteq Q$;

F conjunto de estados finais, $F \subseteq Q$;

V alfabeto auxiliar (pode ser vazio);

β símbolo especial "branco";

¤ símbolo ou marcador de início da fita;

O símbolo de início da fita encontra-se mais à esquerda da fita, para informar a cabeça da fita se encontrada na célula mais à esquerda da fita. A função de transição possui o estado corrente e o símbolo lido da fita, levando ao novo estado, ao símbolo a ser gravado e o sentido em que a cabeça da fita se moverá, sendo representados por E (movimento para a esquerda) e D (movimento para a direita).

CAPÍTULO 3 – MODELAGEM DA FERRAMENTA - SCTMF

3.1 PADRÕES DE PROJETO (*DESIGN PATTERNS*)

Definir da melhor forma padrões de projeto (*design patterns*) tem sido alvo de debate constante desde a década de 80, quando surgiu o movimento de padrões de projeto de software. A definição original do arquiteto e professor Christopher Alexander*(Oxford University Press*, 1979*)* diz: "Cada padrão é uma regra de três partes, que expressa a relação dentre um contexto, um problema e uma solução" .

Para estudar um padrão precisamos estudar as suas partes: o problema, a solução e o contexto onde é aplicável. No caso de padrões de software orientado a objetos, tais problemas costumam ser a criação de objetos, estruturação de classes, modos de acesso a dados, formas de troca de mensagens e outros que os desenvolvedores enfrentam de maneiras semelhantes em diversos sistemas.

Os padrões costumam ser encontrados em catálogos voltados para os contextos e o tipo de problema aos quais se aplicam. O livro *Design Patterns* fornece um catálogo geral de padrões de projeto orientado a objetos.

3.1.1 APLICABILIDADE DOS PADRÕES DE PROJETO

Os padrões geralmente fazem parte dos mecanismos centrais de uma aplicação, e estão diretamente ligados a seus requisitos não-funcionais como performance, escalabilidade, manutenção e segurança. Podemos citar como um exemplo, o padrão *Bridge*[GoF] o qual isola a interface de uma classe da sua implementação, permitindo que as duas evoluam independentemente, aumentando assim a manutenibilidade.

Deve-se também considerar os impactos da solução adotada. Ao escolher um padrão que aumenta a flexibilidade do sistema, pode-se deixa-lo mais complexo ou mais lento.

Pode-se citar que o esforço para se aplicar uma solução é proporcional 'a sua generalidade. Os catálogos de padrões de projeto trazem uma lista de conseqüências comuns da aplicação de cada padrão, baseando-se nas experiências dos autores, o que torna a decisão de adotá-los ou não mais segura.

3.2 ARQUITETURA DA FERRAMENTA

Para o desenvolvimento da arquitetura da ferramenta, foram utilizados três *design patterns*: *Singleton*, *AbstractFactory* e *Factory*.

O *Singleton* garante que apenas uma instancia da classe na aplicação, e fornece um único ponto de acesso a essa instância. *O AbstractFactory* cria famílias de objetos relacionados ou dependentes, sem especificar suas classes concretas. Finalmente, o *Factory* é usando quando é necessário criar um objeto, mas não se sabe qual classe deve ser instanciada até a hora da execução. É necessário separar a criação do objeto da escolha de qual classe instanciar.

Para o desenvolvimento da arquitetura do sistema foi proposta a seguinte idéia: criar uma arquitetura na qual outros modelos formais fossem desenvolvidos posteriormente e acoplados a essa, causando o mínimo de impacto possível na arquitetura.

Para isso, foi criada uma classe genérica onde todas as classes de negócio, ou seja, os modelos formais, que fossem implementados, fossem filhos dessa classe, classe essa com o nome de ModeloFormal. E para o desenvolvimento da interface da implementação do modelo formal, fosse filho da classe ModeloFormalGUI.

Para os testes de cadeia, uma classe específica foi criada, ou seja, a cada modelo formal adicionado a arquitetura, um método que recebesse uma instância do modelo formal a ser utilizado e a seqüência a ser testada como parâmetro, e que retornasse um valor *booleano*, como: **public boolean testarAFD(AFD afd, String sequencia) {...}**.

Também foi incorporada a arquitetura as funcionalidades de abrir e salvar os modelos formais criados. Para essa incorporação basicamente foram criadas as classes que abrem e salvam os modelos formais, as interfaces respectivas dessas classes e as classes que tratam as possíveis exceções que podem ocorrer durante o processo.

A figura 2 demonstra o diagrama de classes, que retrata como está implementada esta funcionalidade:

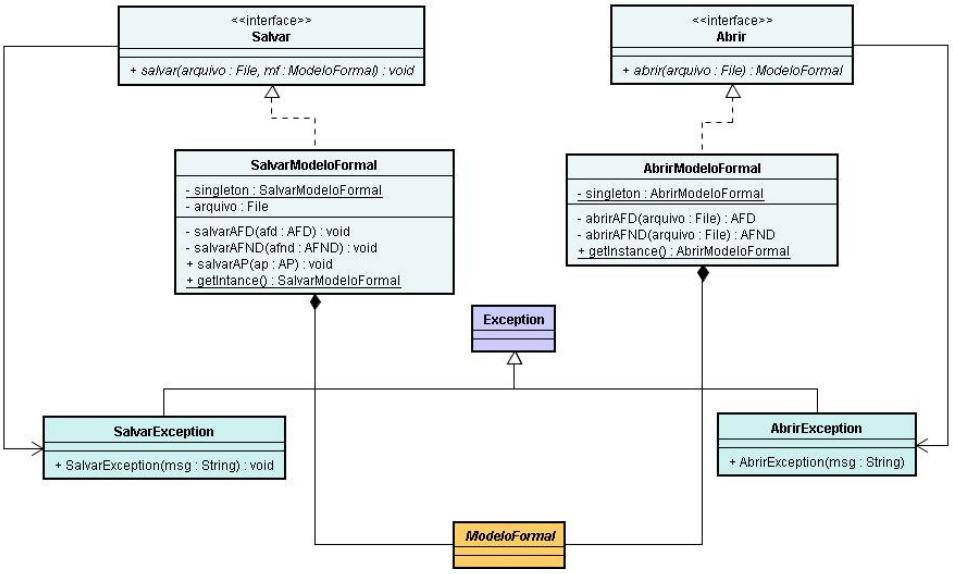


Figura 2 – Diagrama de Classes da funcionalidade de Abrir e Salvar os Modelos Formais.

CAPÍTULO 4 – A FERRAMENTA MULTIPLATAFORMA - SCTMF

O Sistema para Criação e Teste de Modelos Formais (SCTMF), possui diversos módulos implementados para auxílio ao ensino dos modelos formais.

O sistema foi desenvolvido na linguagem JAVA, na sua versão 6.0(Mustang), e utiliza também a tecnologia *Java Web Start* para a sua distribuição. Para a execução deste, os usuários necessitarão ter instalado o ambiente de execução JAVA (JRE 6.0), ambiente este que se encontra disponível para *download* no endereço: http://java.sun.com/downloads.

É importante ressaltar que por a linguagem JAVA ser multiplataforma, o sistema não depende de sistema operacional para ser executado.

Por também ser utilizada a tecnologia *Java Web Start* para distribuição da ferramenta, se tem a garantia de que a versão utilizada da ferramenta será sempre a mais atual disponibilizada.

O SCTMF está dividido nos seguintes módulos: o módulo das linguagens regulares, constituído por: Autômato Finito Determinístico (AFD), Autômato Finito Não Determinístico (AFND) e Expressões Regulares (ER). Já o módulo das Linguagens Livres de Contexto é constituído delo Autômato com Pilha (AP) e a Gramática Livre de Contexto (GLC) e finalmente o módulo das linguagens enumeradas recursivamente, constituído pela Máquina de Turing (MT).

O diagrama do escopo do sistema pode ser demonstrado na figura 3:

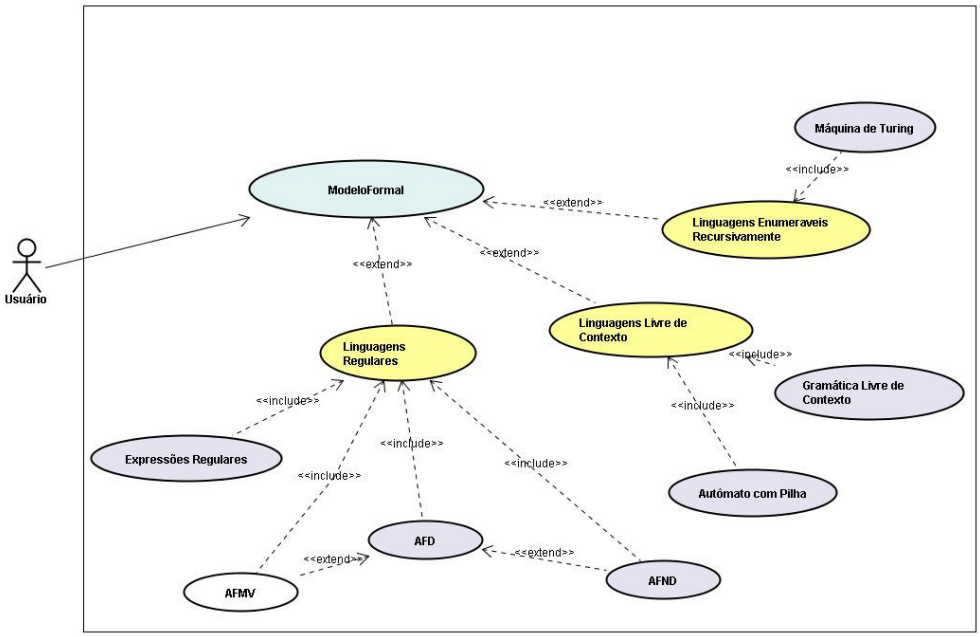


Figura 3 – Diagrama do Escopo do Sistema.

A tela principal do sistema é exibida na figura 4:

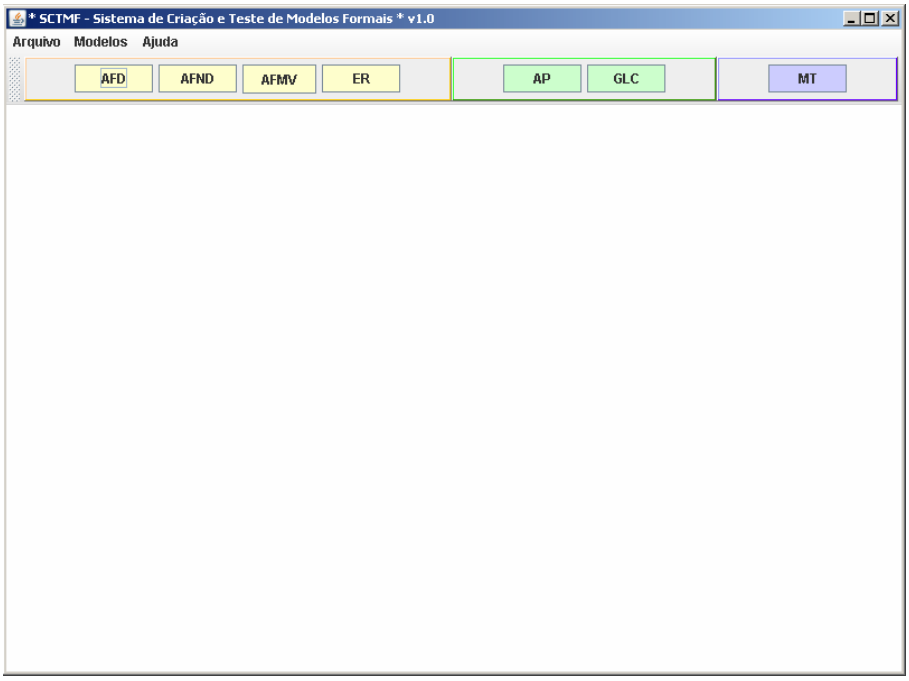
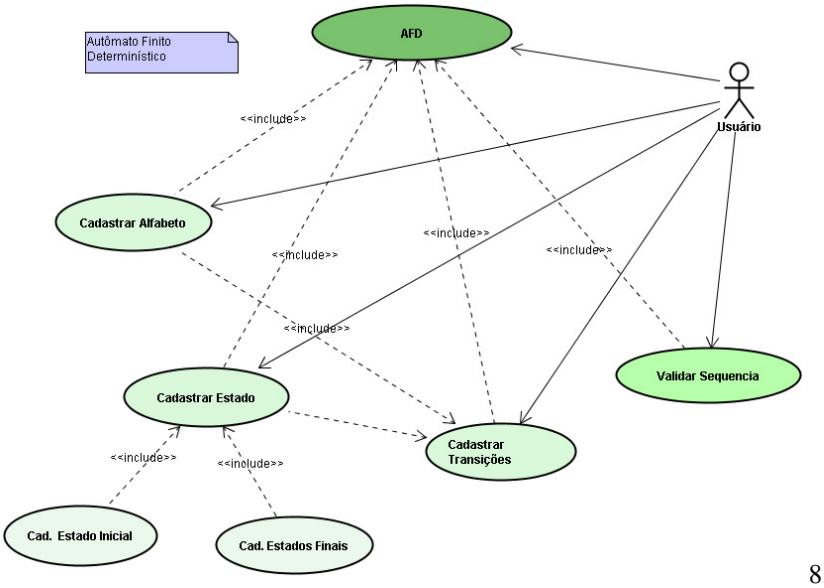


Figura 4 – Tela Principal do Sistema.

4.1 MÓDULO DAS LINGUAGENS REGULARES

4.1.1 MÓDULO DO AFD

A figura 5 ilustra o caso de uso do módulo para a criação de AFDs.



8

Figura 5 – Diagrama de Caso de Uso do AFD.

O módulo do AFD é divido em quatro fases:

- Cadastramento dos Símbolos do Alfabeto da Linguagem: Nessa são cadastrados os símbolos do alfabeto da linguagem que será montada;

A figural 6 ilustra a tela de cadastramento do alfabeto.

Figura 6 – Tela do Cadastramento do Alfabeto – AFD.

- Cadastramento dos Estados: Nessa fase são cadastrados os estados que essa linguagem possui, também é definido o estado inicial e o(s) estado(s) final (is) do autômato;

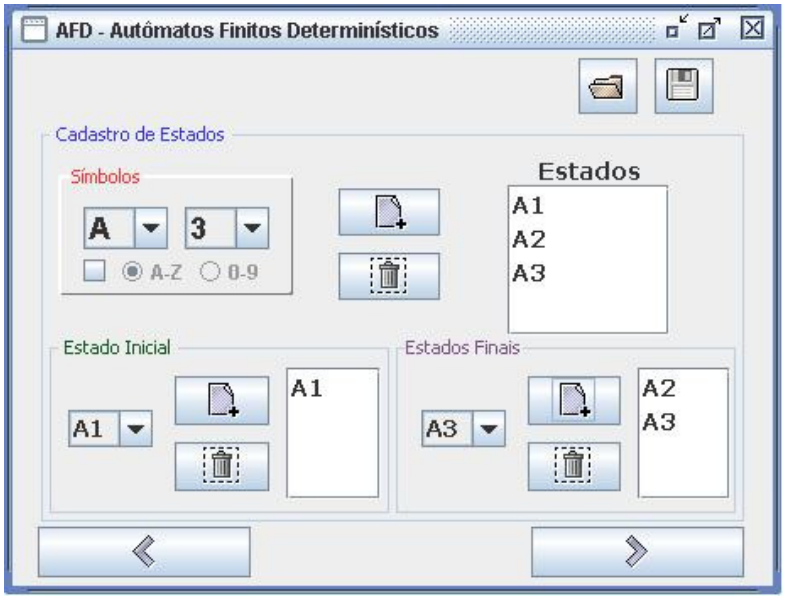


Figura 7 – Tela do Cadastramento dos Estados – AFD.

- Cadastramento das Funções de Transição: fase onde são cadastradas as funções de transição da linguagem. As funções de transição determinam o caráter sintático da linguagem definida pelo modelo, elas levam um par formado por um estado e o símbolo corrente da cadeia que está sendo processada à um novo estado. A figura 8 mostra a interface através da qual se pode cadastrar as funções de transição do AFD que está sendo criado;

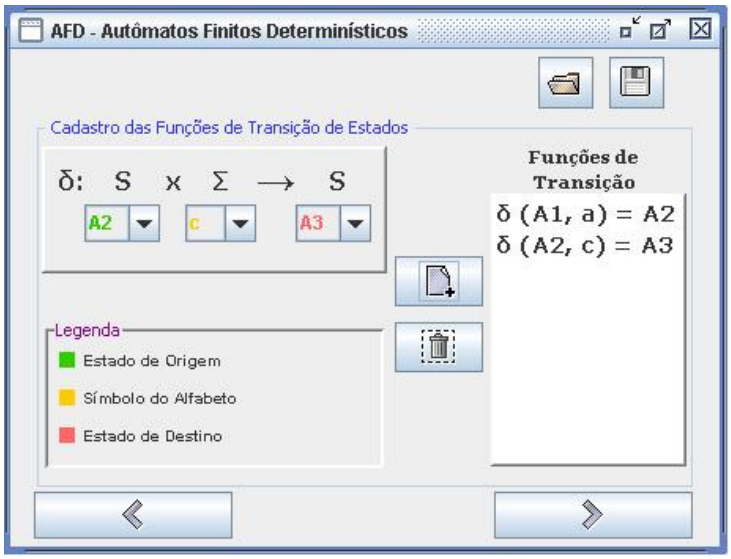


Figura 8 – Tela do Cadastramento das Funções de Transição – AFD.

- Validação das seqüências: nesta fase, autômato construído nas fases anteriores é exibido no lado esquerdo da tela, e a seqüência a ser testada é inserida em um campo a direta da tela. Quando o usuário clicar no botão Testar, o algoritmo do AFD é aplicado para realizar o processamento da seqüência de símbolos (cadeia) fornecida pelo usuário e exibe uma mensagem informando se a seqüência é aceita ou rejeitada.

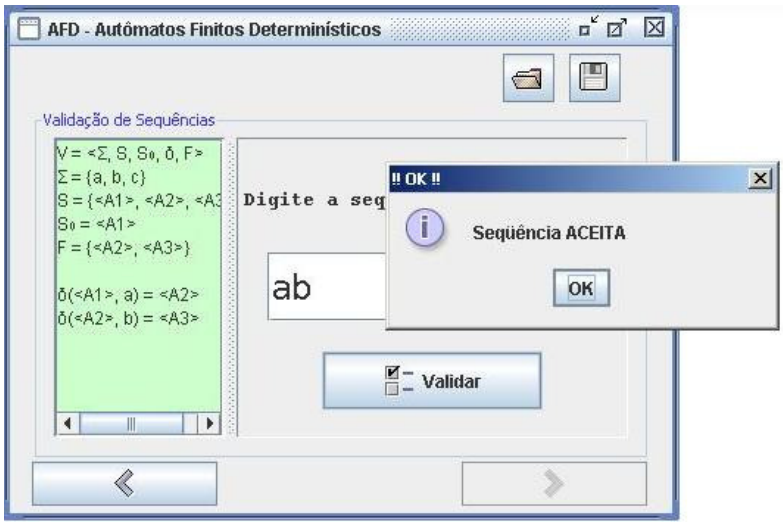


Figura 9 – Tela de Validação de Seqüências – AFD.

4.1.2 MÓDULO DO AFND

As diferenças existentes entre os modelos AFD e AFND, presentes nas definições matemáticas, naturalmente provocam diferenças nos módulos implementados que permitem a criação de instâncias destes dois modelos. Neste sentido, no módulo para criação de AFNDs, na tela de Cadastramento de Estados, será permitido o cadastramento de um ou mais estados iniciais, diferentemente do AFD, no qual é permitido cadastrar apenas um estado inicial, também deve-se salientar a diferença existente no cadastramento das funções de transição, nesse modulo é permitido que sejam cadastrados vários símbolos que ativam um mesmo estado, ou seja, de um estado de origem, podem existir um ou mais símbolos da cadeia que ativam o mesmo estado destino, dando assim o caráter de não-determinismo ao modelo formal.

Figura 10 – Tela do Cadastramento de Estados – AFND.

4.1.3 MÓDULO DO AFMV

Apesar deste módulo parecer muito com o módulo apresentado anteriormente, duas diferenças o destacam do modelo anterior, a primeira é a possibilidade de ser cadastrado apenas um estado inicial, como no AFD, a segunda é no cadastramento das funções de transição, o símbolo Lâmbida(λ) foi acrescentado para permitir o movimento vazio, ou seja, um estado poder ativar o outro, sem necessitar conter um símbolo da cadeia que está sendo analisada, dando assim, dando assim a característica do movimento vazio

Demonstrada na figura 11, a tela de cadastramento de funções de transição do AFMV.

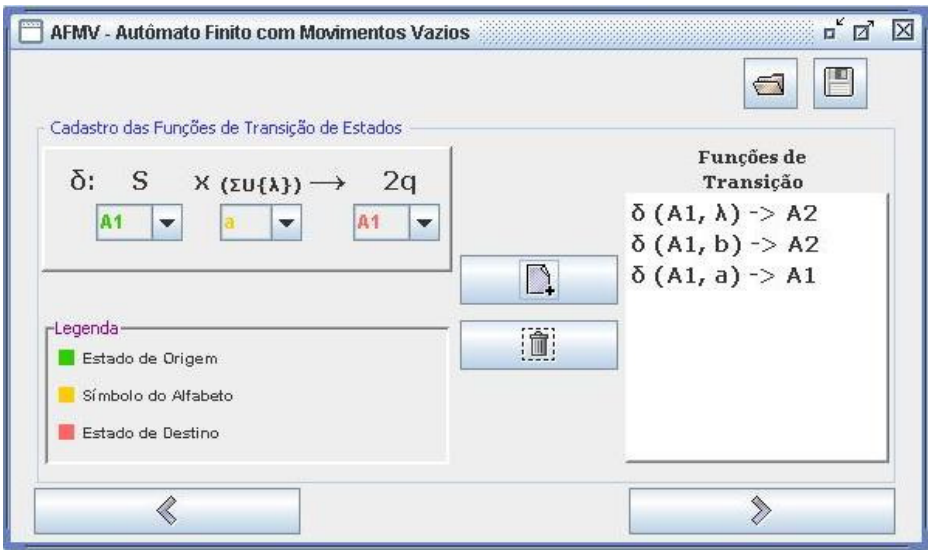


Figura 11 – Tela do Cadastramento das Funções de Transição – AFMV.

4.1.4 MÓDULO DAS EXPRESSÕES REGULARES

Este módulo é composto de somente uma única tela, nela é possível cadastrar os símbolos que farão parte do alfabeto, e também é possível inserir os parênteses para uma eventual quebra de prioridade entre operações, o símbolo da concatenação e o símbolo da concatenação sucessiva.

Para o processo de validação das seqüências no módulo das Expressões Regulares, foram usados dois processos: o cálculo da precedência dos operadores e a regressão da ER em um AFMV, possibilitando assim, a validação da seqüência.

GUIMARÃES afirma :

> ***• "Uma expressão matemática pode usar vários pares de parênteses agrupados. Ex.: ((A + B) * (C – D)). Sendo assim, os pares de parênteses devem ser agrupados corretamente. Ou seja, o número de parênteses de abertura deve ser igual ao número de parênteses de fechamento e um parêntese de fechamento deve ser precedido pelo seu respectivo parêntese de abertura.***
> ***• Em uma expressão matemática devemos considerar a precedência dos operadores. As operações de multiplicação e de divisão têm prioridade sobre as operações de adição e subtração. Onde aparecerem operadores de mesma prioridade, os cálculos serão efetuados na ordem que aparecem na expressão, da esquerda para direita; exceto na potenciação que é feita da direita para esquerda.***
> ***• Os parênteses causam a mudança de prioridade dos operadores numa expressão. Por exemplo, A * B + C produz resultado diferente de A * (B + C)."***

Baseando-se na afirmação, o cálculo das expressões foram adaptados para serem utilizados no calculo da precedência e avaliação dos operadores na transformação da ER em um AFMV.

Pode-se considerar três formas para representar uma expressão: infixa, pré-fixa e pós-fixa, como demonstrada na tabela 2:

TIPO	*DESCRIÇÃO*	*EXEMPLO*
Infixa	Operador está entre os operandos	**(A + B)**
Pré-fixa	Operador precede os operandos	**(+ AB)**
Pós-fixa	Operador após os operandos	**(AB +)**

TABELA 2: Formas de representar uma expressão aritmética.

GUIMARÃES afirma :

> • ***"... As formas pré-fixa e pós-fixa são conhecidas como notação polonesa e notação polonesa reversa, respectivamente, em que a última tem mostrado ser a mais eficiente para construção de algoritmos....***
> • ***.... Para a realização da conversão infixa em pós-fixa, basta considerar a precedência das operações. Isso pode ser enfatizado pelo uso de parênteses colocados de forma explícita. Devemos lembrar que as operações de precedência mais alta devem ser convertidas primeiro. Em seguida, a parte convertida é encarada como um único operando".***

Para o cálculo da precedência de operadores na ferramenta, foi utilizada a forma pré-fixa, então primeiramente foi necessário transformar a expressão cuja estava na forma infixa (definida pelo usuário) para a forma pré-fixa.

Depois de ser transformada a expressão para a forma pré-fixa, a expressão é percorrida e para cada operando encontrado, este é transformado em um AFMV, quando um operador é encontrado, é enviado para um método utilitário no qual faz a operação com um ou dois AFMVs afim de produzir um único, e assim por diante, até o final da expressão.

Ao final da expressão é obtido um único AFMV, no qual a seqüência que foi enviada para análise é enviada para a validação agora com um AFMV.

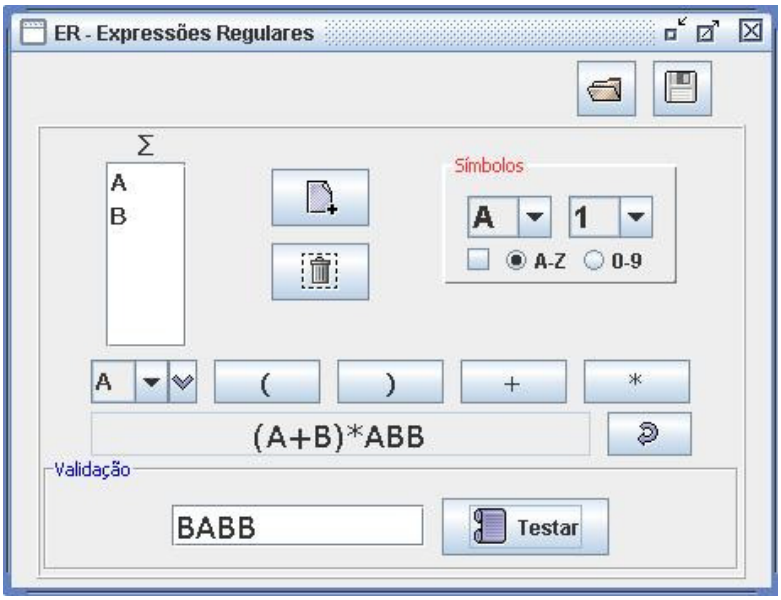


Figura 12 – Tela do Cadastramento e Teste das ERs.

4.2 MÓDULOS DAS LINGUAGENS LIVRES DE CONTEXTO

4.2.1 MÓDULO DO AP

É apresentado na figura 13, o caso de uso do AP.

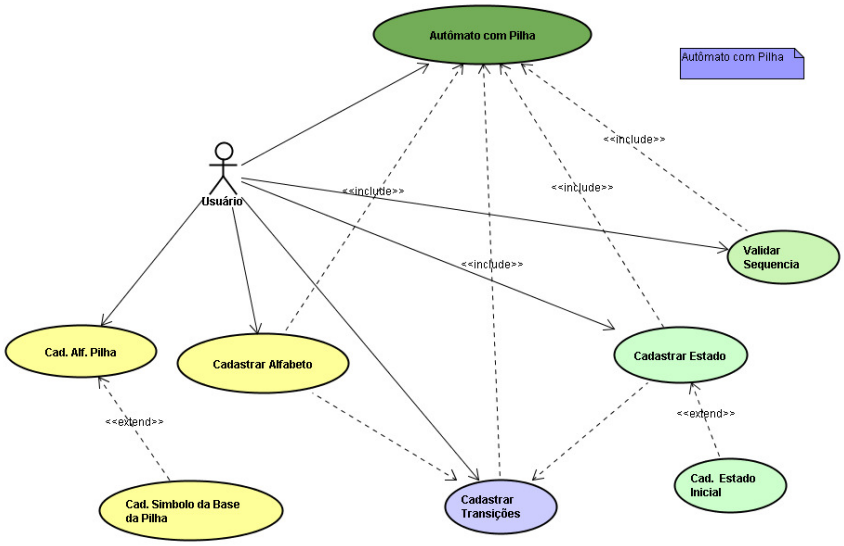


Figura 13 – Diagrama de Caso de Uso do AP.

Este módulo também como o AFD, o AFND e o AFMV é dividido em 4 fases:

- Cadastramento dos Alfabetos: Nessa parte são cadastrados os símbolos do alfabeto da linguagem, e os símbolos do alfabeto da pilha.

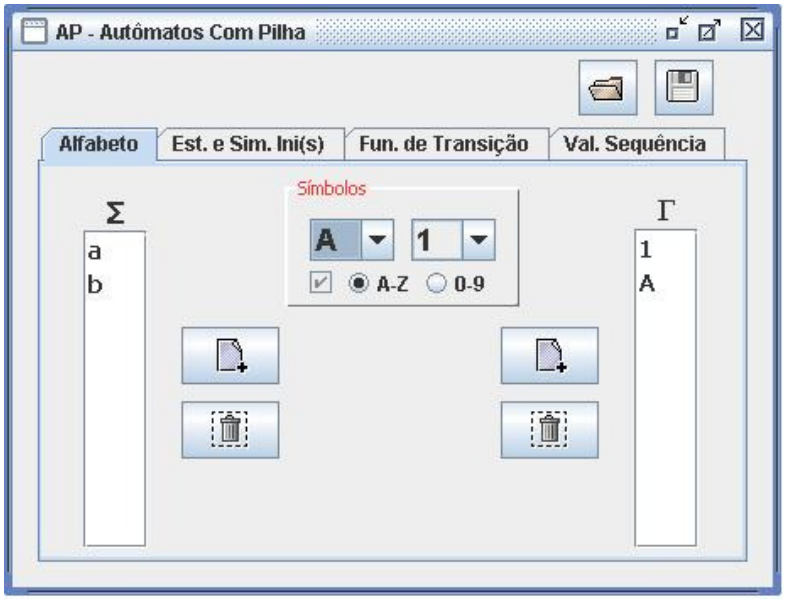


Figura 14 – Tela do Cadastramento dos Alfabetos – AP.

- Cadastramento dos Estados: Nesse módulo como no AFD e no AFND são cadastrados os estados do autômato. Uma notável diferença entre o AP e o AFD e AFND, é que, na definição formal de AP utilizada para a implementação desta ferramenta, o modelo não possui estados finais, ou seja, o que vai determinar se a cadeia vai ser aceita ou rejeitada, é se a pilha estará vazia no final do processamento da cadeia.

Figura 15 – Tela do Cadastramento dos Estados e Símbolo da Base da Pilha – AP.

- Cadastramento das Funções de transição: no AP a transição é constituída de cinco elementos. O estado de origem, o símbolo do alfabeto, o símbolo do topo da pilha, o estado destino e a entrada da pilha. A figura 14 demonstra a tela do cadastramento de funções de transição no AP.

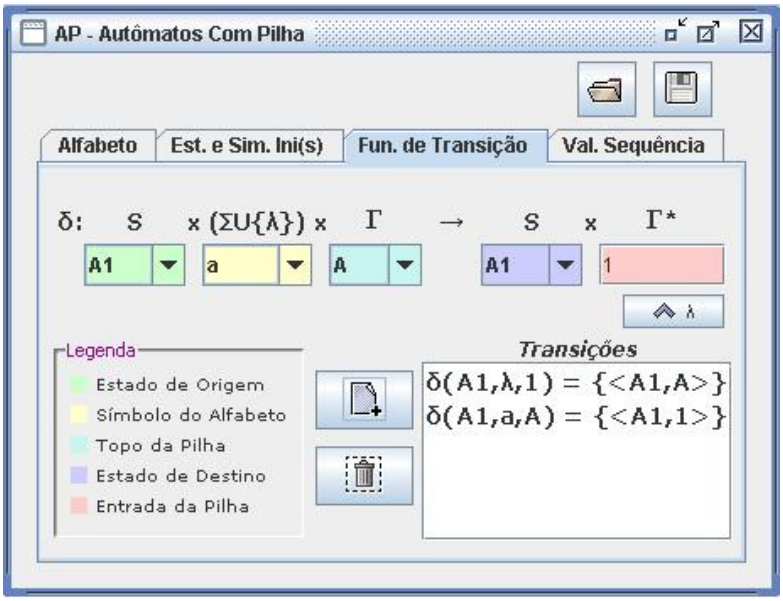


Figura 16 – Tela do Cadastramento das Funções de Transição – AP.

4.2.2 MÓDULO DA GRAMÁTICA LIVRE DE CONTEXTO

Para esse módulo foi adotado o uso do algoritmo de Cocke-Younger-Kasami, um dos algoritmos mais conhecidos de análise ascendente de cadeias para linguagens livres de contexto, o qual pode ser usado em compiladores para o desenvolvimento da análise sintática dos códigos fontes. Na figura 17 é demonstrado o cadastramento dos alfabetos da gramática, ao lado esquerdo é apresentado o alfabeto dos símbolos não-terminais, e do lado direito, o alfabeto dos símbolos terminais.

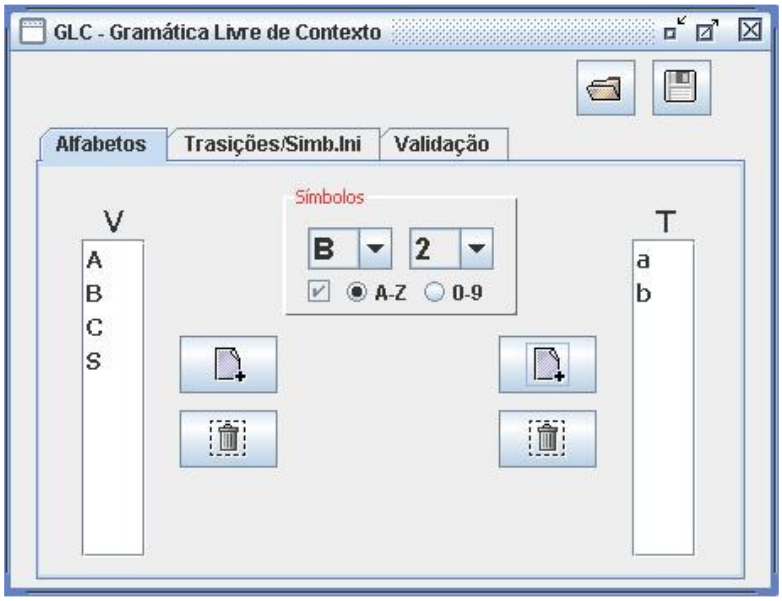


Figura 17 – Tela do Cadastramento dos Alfabetos – GLC

O próximo passo é ser feito o cadastramento das regras de produção e definição do símbolo inicial da gramática, o sistema exige que o lado direito da expressão esteja na Forma Normal de Chomsky(FNC), restrição dada pelo algoritmo de CYK. A figura 18 demonstra o passo descrito anteriormente.

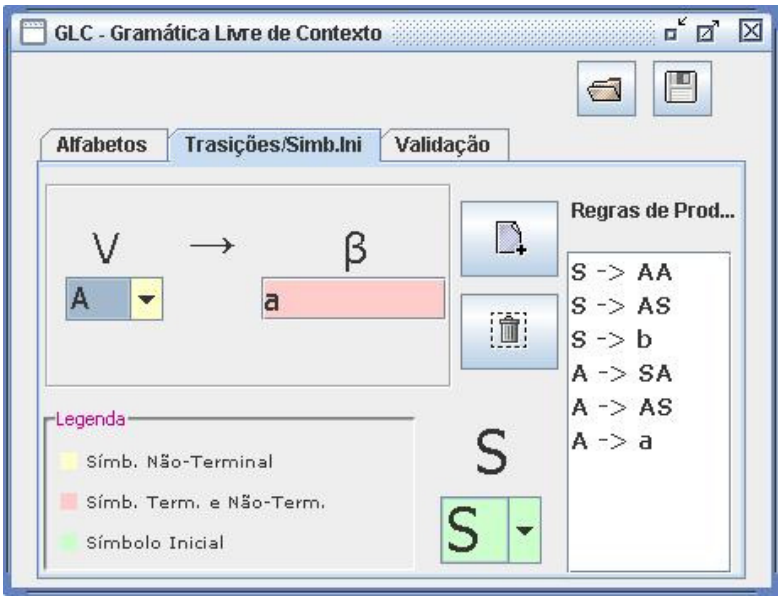


Figura 18 – Tela das Transições/Símbolo Inicial – GLC

Finalmente depois de efetuado os passos anteriores, a tela apresentada na figura 19 demonstra a o teste da seqüência definida na linguagem anteriormente criada.

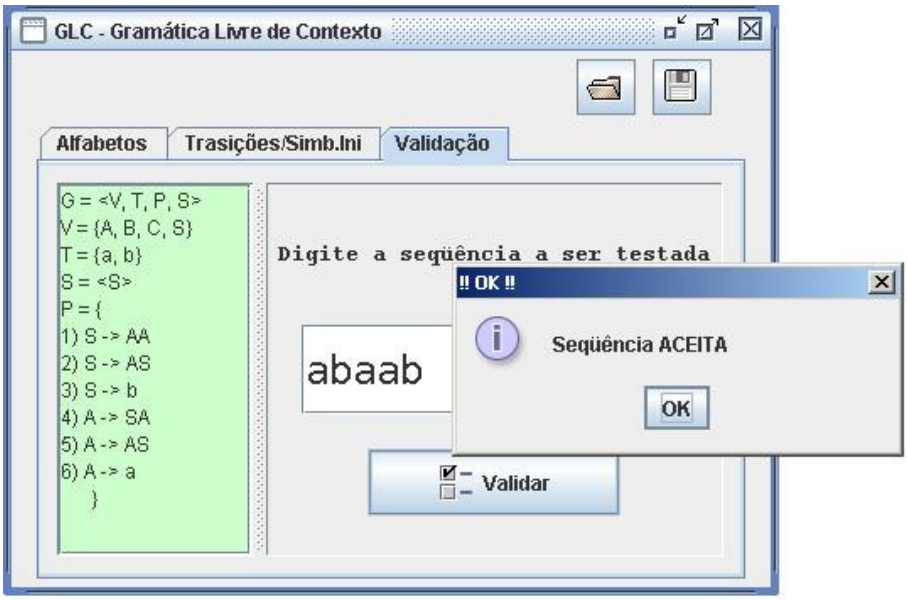


Figura 19 – Validação da Seqüência – GLC

4.3 MÓDULO DAS LINGUAGENS ENUMERÁVEIS RECURSIVAMENTE

4.3.1 MÓDULO DA MÁQUINA DE TURING

Neste módulo pode-se dar um destaque especial para o uso da fita de entrada e das funções de transição, pois através das funções de transição pode-se alterar os valores da fita de entrada, de forma que a fita não seja somente de entrada, mas também de saída.

Na tela de cadastramento de Alfabetos são cadastrados 2 alfabetos, o alfabeto de símbolos de entrada representado a esquerda da figura 20 e o alfabeto auxiliar apresentado a direita de mesma.

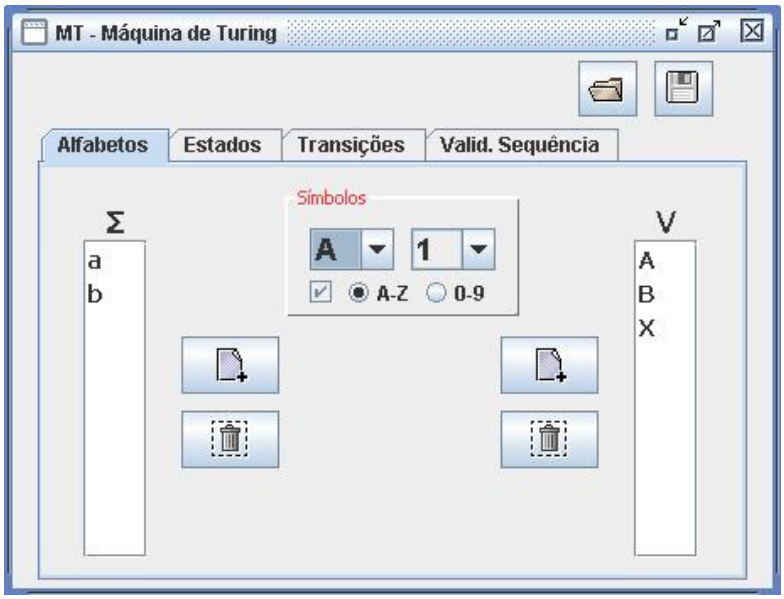


Figura 20 – Cadastramento dos Alfabetos – MT

Na tela de cadastramento de Estados são cadastrados os estados, e definidos o estado inicial e os estados finais.

Na tela de cadastramentos de funções de transição são cadastradas as transições, as quais são compostas no lado direito (Estado de Origem, Símbolo a ser lido) e no lado esquerdo(Estado de Destino, Símbolo a ser Gravado e o Sentido do Movimento, o qual é a direita ou a esquerda), na figura 21 é apresentada a tela.

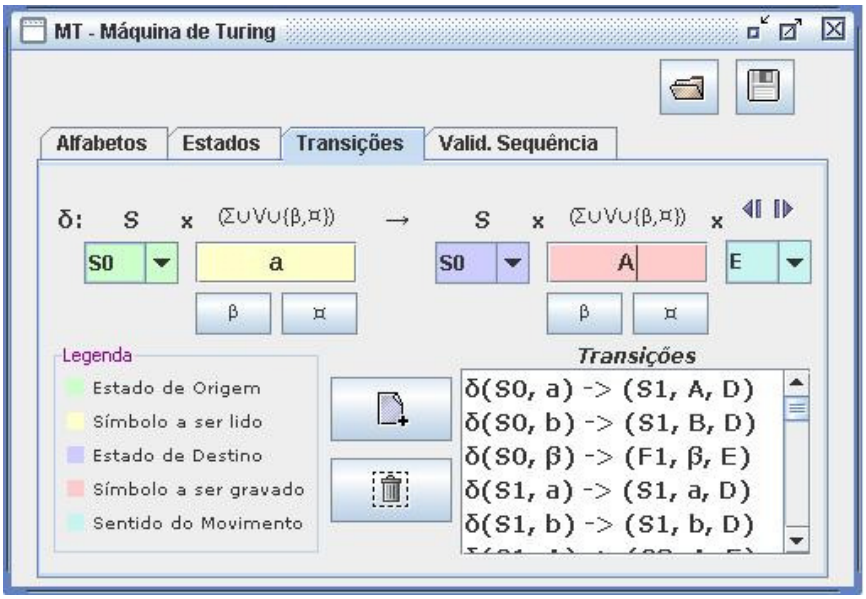


Figura 21 – Cadastramento das Transições – MT

Na figura 22 é apresentada a tela de validação de seqüências, no qual é apresentado o modelo formal juntamente com suas transições.

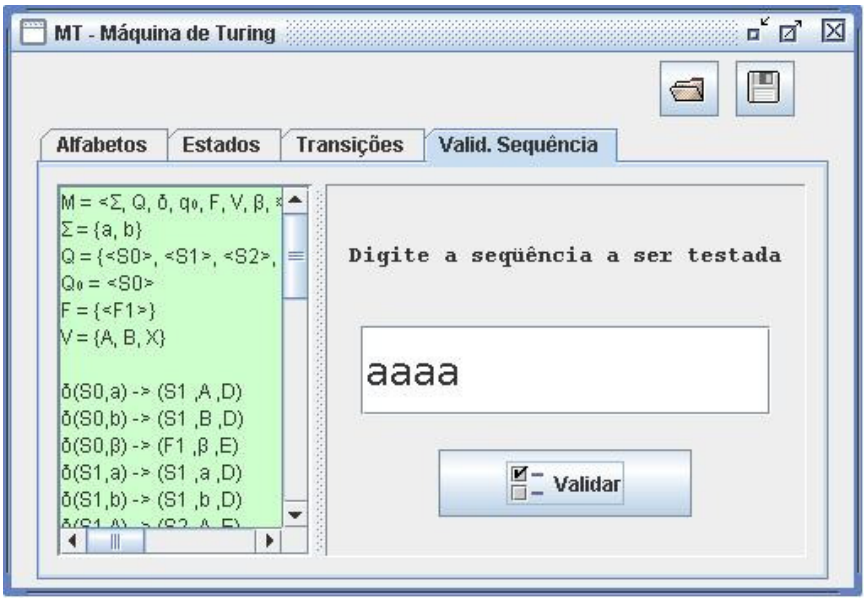


Figura 22 – Validação de Seqüência – MT

Ao final do processamento de validação é apresentada a fita da máquina, e em seguida é exibido se a seqüência foi aceita ou rejeitada. A figura 23 apresenta a fita da máquina construída.

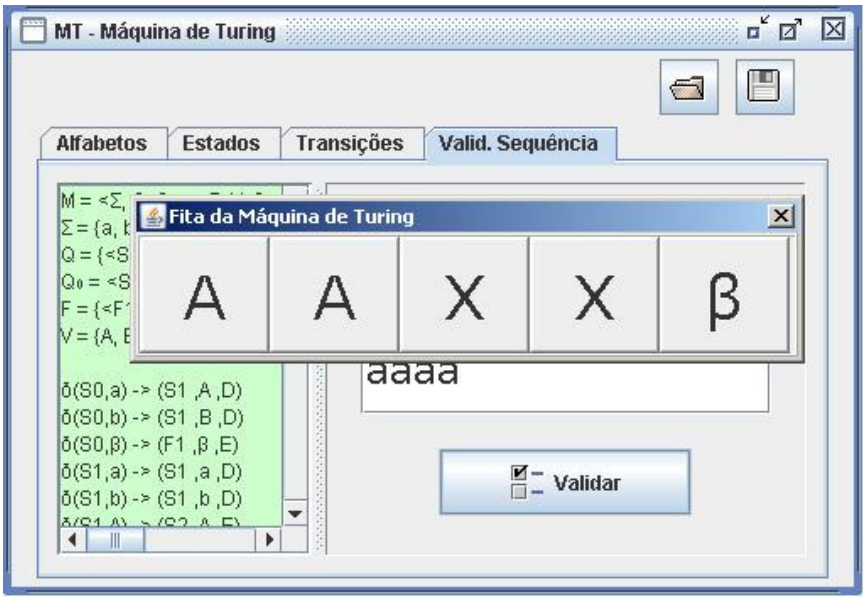


Figura 23 – Fita da Máquina de Turing processada.

CAPÍTULO 5 – TRABALHOS CORRELATOS

Neste capítulo são apresentados alguns trabalhos desenvolvidos com objetivos parecidos aos da ferramenta SCTMF, também é apresentada uma tabela comparativa entre SCTMF e estas ferramentas.

5.1 AMBIENTES DIDÁTICOS JÁ DESENVOLVIDOS

Em trabalhos como (Jukemura, 2005), pode-se verificar que o desenvolvimento de ambientes didáticos para o ensino dos modelos formais vem sendo objeto de estudo de alguns pesquisadores.

A tabela 1 exibe um comparativo das características do SCTMC e 2 ferramentas similares a sua funcionalidade, o JFLAP e o SIMFOR:

5.1.1 JFLAP

O JFLAP (*Java Formal Language and Automata Package*) é uma ferramenta visual usada para criar e simular diversos tipos de autômatos, e converter diferentes representações de linguagens. Foi desenvolvida pela Prof. Dr. Susan H. Rodger, professora do departamento de ciência da computação da Duke University, Durham, NC 27708-0129; Atualmente a ferramenta encontra-se no seu *release* 6.1(18/05/07) com mais de 87.000 acessos ao site da ferramenta.

Seu principal objetivo é facilitar o aprendizado de teoria de linguagens, através de uma interface simples e intuitiva. Ele pode ser usado tanto como aparato de auxílio às aulas, como ferramenta de estudo e pesquisa, facilitando tanto a criação de autômatos, quanto a verificação das seqüências testadas na ferramenta.

O passo inicial para a utilização do JFLAP é a criação de autômatos. O usuário utiliza a interface visual para criar um grafo representando os estados, o diagrama de transições e os seus rótulos. Em seguida, pode-se definir a palavra de entrada e então visualiza-se a execução de cada passo do autômato, verificando-se o seu projeto é coerente. Isso pode ser feito com três escolhas de execução, um modo rápido, que indica a resposta imediata, um modo passo a passo, que mostra os estados percorridos, e o modo múltiplo, que mostra o teste de diversas palavras, da mesma maneira que o modo rápido.

5.1.2 SIMFOR

O SIMFOR (Simulador de Modelos Formais), descrito em (FERNANDES, G. 2002), possui um conjunto de ferramentas computacionais para apoio ao ensino de Linguagens Formais e Autômatos, foi desenvolvido em Borland Delphi 5.0, para Sistema Operacional Windows.

O sistema está dividido nos seguintes módulos: Autômato Finito Determinístico, Autômato Finito Não Determinístico (Com transformação para AFD), Gramática Regular (Com transformação para AFD e AFND), Autômato com pilha, Gramática Livre de Contexto (Com transformação para a Forma Normal de Chomsky e análise através de algoritmo de Cocke-Younger-Kasami) e Máquina de Turing.

O que diferencia esse sistema dos outros já desenvolvidos, como a JFLAP(2007) e o SIMFOR(2002), além da característica da ferramenta ser multiplataforma e ser disponibilizada através do ambiente *Java Web Start*, pode-se dar ênfase à arquitetura da aplicação, arquitetura na qual possibilita que outros desenvolvedores também criem seus modelos formais e acoplem a aplicação.

Características	JFLAP	SIMFOR	SCTMF
S.O.	Todos	Windows	Todos
Execução	Local/Applet	Local	Java Web Start
Linguagem de Desenvolvimento	Java 1.4	Borland Delphi 5	Java 6.0
Idioma	Inglês	Português	Português
Usuários	Alguns países do continente africano / UFMG / PUC-RS / UnB / USP	UEM	UEM / FAFIMAN
Licença	Desconhecida	Desconhecida	Apache 2.0
Modelos Formais	AFs / Máquina de Mearly / Máquina de Moore/ AP / MT / Gramáticas / *L-System* / ER / LLC.	AFD / AFND / AP / MT / GR / GLC.	AFD / AFND / AFMV/ ER / AP / GLC / MT.

TABELA 3: Comparativo entre o SCTMF, SIMFOR e JFLAP

CONCLUSÃO

Uma das principais vantagens da ferramenta encontra-se na sua arquitetura ‘plugável’, característica essa que deixa em aberto a implementação de outras funcionalidades para a ferramenta, além da sua característica multiplataforma, a qual facilita a utilização pelos alunos na medida em que confere alto grau de portabilidade à ferramenta.

A implementação dessa arquitetura foi favorecida pelo uso da programação orientada a objetos e a utilização de padrões de projeto (*Singleton, Factory e AbstractFactory*), pois o uso do polimorfismo (uma das principais características da Orientação a Objetos) foi de alta importância para a parte “plugável”da arquitetura da ferramenta.

Adicionalmente do que se foi proposto para este trabalho, foram implemendas a funcionalidade de abrir e salvar os modelos formais para todos os modelos propostos. Também foi acrescentado ao sistema, o modelo formal do Autômato Finito com Movimentos Vazios, dando assim um complemento a mais para a ferramenta.

Em trabalhos futuros pretende-se implementar o módulo de debug dos modelos formais, ou seja, poderão ser observados pelos alunos os estados, funções de transição, e situação de depuração do modelo formal, uma espécie de *debug* para a ferramenta. Essa funcionalidade não foi implementada na versão 1.0, pois não estava no escopo do que foi proposto e também a consumação de tempo para o desenvolvimento da funcionalidade é elevada, pois será necessário uma quantidades muito alta de horas de estudo, pesquisa e desenvolvimento, o que não viabilizava um bom resultado dentro dos prazos para o desenvolvimento desse trabalho. Entretanto para a versão 2.0 será implementado esse módulo, dando assim um maior clareza para os usuários.

Também a construção de uma interface interativa, na qual o aluno poderá ver com maior clareza o modelo formal finalizam os módulos que serão implementados futuramente.

Outra implementação futura será para o módulo da GLC, atualmente são cadastradas somente funções de transição que estejam na FNC (Forma Normal de Chomsky), para a versão posterior da ferramenta não será necessário esse critério para a FNC, a ferramenta automaticamente converterá a função de transição para a FNC.

BIBLIOGRAFIA

CARROL, J. e LONG, D. *Theory of Finite Automata With a Introduction to Formal Languages, Prentice-Hall* , 1989.

DELAMARO, Márcio Eduardo. *Linguagens. Formais e Autômatos*. UEM, 1998.

FERNANDES, G. *Construção de uma Ferramenta para Auxílio didático no Ensino de Linguagens formais e Autômatos*. Paranavaí, 2002. 62p. Monografia (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Universidade Paranaense – UNIPAR.

[GoF] GAMMA, Erich; HELM, Richart; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object Oriented Software*(Addison-Wesley, 1994)

GUIMARÃES, Renato. *Resolvendo expressões com a classe Stack*. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/vbnet/visualbasic_Stack.mspx>. Acesso em: 15 nov. 2007.

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. *Introdução à Teoria de Autômatos, Linguagens e Computação*, Ed.: Campus, Rio de Janeiro, 2002.

JFLAP: *JFLAP is a package of graphical tools which can be used as an aid in learning the basic concepts of Formal Languages and Automata Theory.* Disponível em: <http://www.jflap.org/>. Acesso em: 26 out. 2007.

JUKEMURA, A. S.; NASCIMENTO, H. A. D.; UCHOA, J. Q. in *GAM – Um simulador para auxiliar o ensino de linguagens formais e autômatos* in Anais do 25º Congresso da Sociedade Brasileira de Computação, São Leopoldo, 2005.

MENEZES, P. B. in *Linguagens Formais e Autômatos*, Ed.: Sagra-Luzzatto, Porto Alegre, 1998.

NETO, José João. *Introdução à Compiladores. Livros técnicos e Científicos*, 1987.

REZENDE, Pedro A. D. *Autômatos e computabilidade - UNB*. Disponível em: <http://www.cic.unb.br/~pedro/prog_ac.html>. Acesso em: Acesso em: 04 nov. 2007.

SUDKAMP, T. A. Languages and Machines: An introduction to Theory of Computer Science, Second Edtion, Addison Wesley, 1997.

VIEIRA, Newton J. *Fundamentos da Teoria da Computação - UFMG.*
Disponível em: <http://homepages.dcc.ufmg.br/~nvieira/cursos/ftc/a07s1/lExtra.pdf>.
Acesso em: Acesso em: 02 nov. 2007.