UNIPAR – UNIVERDIDADE PARANAENSE

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CONSTRUÇÃO DE UMA FERRAMENTA PARA AUXÍLIO DIDÁTICO AO ENSINO DE LINGUAGENS FORMAIS E AUTÔMATOS

Autora: Gisele Fernandes
Orientador: Prof. Yandre Maldonado e Gomes da Costa

Trabalho de Conclusão de Curso apresentado como parte das exigências para obtenção do título de BACHAREL em Ciência da Computação, pela Universidade Paranaense.

PARANAVAÍ
Estado do Paraná
Novembro de 2002

Ficha catalográfica preparada pela Seção de Catalogação Classificação da Biblioteca da UNIPAR

005
F324c
Ex.1

Fernandes, Gisele
Construção de uma ferramenta para auxilio didático ao ensino de linguagens formais e autômatos. Orientação de Yandre Maldonado e Gomes da Costa; Bancas de Roni Francis Shigeta e Claudia Moser. ---- Paranavaí: UNIPAR, Campus – Paranavaí, 2002. 62p.
Contém tabelas.

1. Informática 2. Linguagens Formais e Autômatos 3. Gramática regular Gramática livre de contexto. I. Costa, Yandre Maldonado e Gomes da. II. Shigeta, Roni Francis. II Moser, Claudia. IV.Titulo V. Série.

CDD 005

Índice para catalogo Sistemático

1. Informática: Linguagens Formais – Autômatos 2. Gramática regular. 3. Gramática livre de contexto.

UNIPAR UNIVERSIDADE PARANAENSE

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CONSTRUÇÃO DE UMA FERRAMENTA PARA AUXÍLIO DIDÁTICO AO ENSINO DE LINGUAGENS FORMAIS E AUTÔMATOS

Autora: Gisele Fernandes
Orientador: Prof. Yandre Maldonado e Gomes da Costa

TITULAÇÃO: Bacharel em Ciência da Computação

APROVADO em 26 de novembro de 2002.

Prof. Yandre Maldonado e G. da Costa (Orientador)	Roni Francis Shigeta
Prof. Claudia Moser	

"Que Deus me dê a serenidade para aceitar as coisas que não posso mudar, coragem para mudar as que posso e sabedoria para distinguir entre elas"

Abrahan Lincoln

AGRADECIMENTOS

Agradeço a minha família por todo o apoio e compreensão, a quem divido os méritos dessa vitória.

Aos meus amigos Ana Maria, Ricardo Francelin, João Carlos Catarin e Helga, pelo companheirismo ao longo desses anos.

Ao meu Orientador Yandre Maldonado e Gomes da Costa por toda a dedicação e incentivo para a realização deste trabalho.

LISTA DE FIGURAS

LISTA DE SIGLAS

AFD	–	Autômato Finito Determinístico
AFND	–	Autômato Finito Não Determinístico
AP	–	Autômato com Pilha
FNC	–	Forma Normal de Chomsky
GLC	–	Gramática Livre de Contexto
GLUD	–	Gramática Linear Unitária à Direita
GR	–	Gramática Regular
LFA	–	Linguagens Formais e Autômatos
LLC	–	Linguagem Livre de Contexto
LR	–	Linguagem Regular
MT	–	Máquina de Turing
SIMFOR	–	Simulador de Modelos Formais

ÍNDICE

RESUMO

A Teoria da Computação caracteriza-se como uma importante área de estudo na Ciência da Computação, proporcionando um grande conhecimento teórico e o desenvolvimento de um raciocínio lógico e formal, necessários para a fundamentação da ciência da computação. Desse modo, o desenvolvimento de ferramentas integradas em um único ambiente computacional, permitindo um fácil entendimento e associação dos conceitos teóricos e da dinâmica dos modelos formais, pode auxiliar significativamente no estudo de Linguagens Formais e Autômatos. O ambiente computacional desenvolvido neste trabalho, proporciona ao usuário a construção de modelos formais como: Autômato Finito Determinístico, Autômato Finito Não-determinístico, Autômato com Pilha, Máquina de Turing, Gramática Regular e Gramática Livre de Contexto. Desta forma, será possível a construção tanto de modelos formais reconhecedores (autômatos e máquinas) quanto de modelos formais geradores (gramáticas) de sentenças. Outro fato importante de se observar é a possibilidade de se definir, neste ambiente, modelos formais relativos a diferentes classes de linguagens. Podendo, depois de construir um modelo formal, submeter cadeias de símbolos para a realização de testes sintáticos através do ambiente aqui proposto.

INTRODUÇÃO

A Teoria de Linguagens Formais e Autômatos é um tópico abrangente que se insere no estudo da Teoria da Computação. Especificamente, concentra-se no estudo de modelos formais que permitam a descrição de linguagens.

A Teoria da Computação caracteriza-se como uma importante área de estudo na Ciência da Computação, pois nela estuda-se tanto fundamentos que descrevem o computador como um modelo matemático (máquinas universais e computabilidade), quanto modelos formais que permitem a definição de linguagens.

Pode-se verificar a importância da Teoria da Computação na descrição e representação de linguagens formais no processo de desenvolvimento de tradutores de linguagens computacionais, como os compiladores e interpretadores. Estes tradutores se encarregam de analisar a linguagem fonte utilizada pelo seu usuário e verificar se as construções estabelecidas pelo mesmo respeitam as suas regras sintáticas. Para que isto seja possível, esta linguagem fonte deve ser representada por algum modelo formal.

Na Teoria da Computação, os formalismos podem ser classificados em dois grupos: o grupo dos formalismos reconhecedores, onde se encontram as máquinas (como os autômatos); e o grupo dos formalismos geradores, onde se encontram as gramáticas. Os formalismos geradores são modelos formais que procuram gerar (produzir) as sentenças de uma linguagem formal, enquanto que os formalismos reconhecedores percorrem uma determinada seqüência de símbolos a fim de verificar se a mesma é reconhecida ou não pela linguagem formal.

Através de estudos sobre as Linguagens Formais e Autômatos, este trabalho tem por objetivo o desenvolvimento de ferramentas integradas em um único ambiente computacional, para uma melhor associação entre os conceitos teóricos e a dinâmica dos mesmos quando em operação.

Na prática do ensino relacionado a estes conceitos, verifica-se uma grande dificuldade por parte dos aprendizes em assimilá-los, sobretudo pelo fato de que estes muitas vezes não conseguem associar tais conceitos às situações práticas onde podem ser aplicados. Adicionalmente, o forte caráter matemático que sustenta os formalismos supracitados, acaba impondo uma resistência a mais aos aprendizes.

Considerando estes fatos, julga-se oportuna a construção de um conjunto de ferramentas computacionais que possam contribuir para que os aprendizes se aproximem aos elementos formais estudados podendo participar direta e rapidamente da elaboração destes e verificando através de simulação instantânea o processamento dinâmico realizado pelos mesmos.

Para o desenvolvimento de tal trabalho foi desenvolvida inicialmente uma pesquisa a fim de identificar quais elementos formais seriam contemplados pelas ferramentas a serem desenvolvidas. Isto feito, foi dado início ao projeto e construção do ambiente computacional que contem tais ferramentas. Depois de construído este ambiente, ele foi submetido a intensas baterias de testes para a identificação e correção de possíveis falhas, além da verificação da usabilidade do mesmo, que deve ser maximizada principalmente por se tratar de um ambiente de auxílio didático.

A ferramenta permite a construção de cinco tipos de formalismos reconhecedores: Autômato Finito Determinístico (AFD), Autômato Finito Não Deterministico (AFND), Autômato com Pilha (AP) e Maquina de Turing (MT). Dois tipos de Formalismos Geradores: Gramática Regular (GR) e Gramática Livre de Contexto (LLC).

De uma forma geral, é importante observar que as quatro classes presentes na Hierarquia de Chomsky são contempladas por esta ferramenta. Além da construção destes formalismos, a ferramenta também possibilita a verificação sintática de cadeias e também oferece as transformações de AFND para AFD, GR para AFD, GR para AFND, AFD para GR e GLC para Forma Normal de Chomsky.

O Capitulo 1 apresenta os conceitos básicos no estudo de Linguagens Formais e Autômatos, como definições de: Alfabeto, Cadeia, Linguagem, Formalismos Geradores (Gramáticas, Derivação, Linguagem Gerada), Formalismos Reconhecedores (Autômato, Autômato com Pilha, Maquina de Turing)

O Capitulo 2 mostra a classificação da linguagens formais de acordo com a hierarquia de Chomsky, onde estão divididas em quatro classes distintas, são elas: Linguagens Regulares (ou tipo 3), Linguagens Livres de Contexto (ou tipo 2), Linguagens Sensíveis ao Contexto (ou tipo 1) e Linguagens Enumeráveis Recursivamente (ou tipo 0).

O Ambiente Computacional, o SIMFOR, é apresentado no Capítulo 3, onde é abordado a sua construção e os itens que o compõem. Neste capítulo, também são descritos modelos formais capazes de definir Linguagens Regulares, tais como: o

Autômato Finito Determinístico (AFD) - para modelar sistemas de entradas e saídas discretas e que internamente possuem vários estados que mudam à medida que a entrada é processada; o Autômato Finito Não-Determinístico (AFND) – semelhante ao AFD, mas podendo existir múltiplos caminhos para processar um determinada cadeia - e a Gramática Regular (GR) – permite definir linguagens através de regras de substituição. Os Modelos Formais capazes de definir Linguagens Livres de Contexto, como: o Autômato com Pilha (AP), cuja estrutura é semelhante à do Autômato Finito, mas possui uma memória auxiliar tipo pilha, a qual pode ser lida ou gravada e que acrescenta poder computacional ao formalismo; e a Gramática Livre de Contexto (GLC), onde as regras de produção são definidas de forma mais livre que na Gramática Regular. Os Modelos Formais capazes de definir Linguagens Enumeráveis Recursivamente e Linguagens Sensíveis ao Contexto como: Máquina de Turing.

CAPITULO 1 – CONCEITOS BÁSICOS

"O Dicionário Aurélio define linguagem como 'o uso da palavra articulada ou escrita como meio de expressão e comunicação entre pessoas'. Entretanto, esta definição não é suficientemente precisa para permitir o desenvolvimento matemático de uma teoria sobre linguagens." (MENEZES, 2000, p.21)

1.1 ALFABETO

Alfabeto é um conjunto finito de símbolos. Um conjunto vazio também é considerado um alfabeto. Os símbolos de um alfabeto não precisam ter um único caracter, nem precisam ter todos o mesmo número de caracteres. A única restrição que se faz, é que o tamanho (número de caracteres) de cada letra seja finito.

Exemplos de Alfabeto: $\Sigma_1=\{a, b, c, d, e, f\}$

$\Sigma_2=\{1, 2, 3, x, y, \&,\#\}$

$\Sigma_3=\{aa,bc, dd\}$

1.2 PALAVRA, CADEIA DE CARACTERES OU SENTENÇA

Uma seqüência finita de símbolos justapostos é uma palavra (cadeia ou sentença). Por exemplo, se a, b e c são símbolos de um mesmo alfabeto, então abcab é um exemplo de palavra formada sobre este alfabeto.

"A *palavra vazia*, representada pelo símbolo λ, é uma palavra sem símbolo. Se Σ representam um alfabeto, então Σ* denota o conjunto de todas as palavras possíveis sobre Σ. Analogamente, Σ^+ representa o conjunto de todas as palavras sobre Σ executando-se a palavra vazia, ou seja, $\Sigma^+ = \Sigma^* - \{\lambda\}$." (MENEZES, 2000, p. 21)

Exemplos de Cadeia: abc é a cadeia sobre Σ_1;

1#32 é cadeia sobre Σ_2;

1.3 LINGUAGEM

Uma Linguagem (formal) é um conjunto de palavras sobre um alfabeto.

O conjunto vazio e o conjunto formado pela palavra vazia são exemplos de linguagens sobre Σ.

Exemplo de linguagem: L={a, b, aa, ab, ba, bb} é uma linguagem do alfabeto{a, b}.

1.4 CONCATENAÇÃO SUCESSIVA

Para a descrição de linguagens ao longo deste trabalho, será realizada a notação da Concatenação Sucessiva.

A Concatenação Sucessiva de uma palavra (com ela mesma), representada na forma de um expoente w^n onde w é uma palavra e n indica o número de concatenações sucessivas, é definida indutivamente a partir da concatenação binária, como segue:

a) Caso 1: $w \neq \lambda$

$w^0 = \lambda$

$w^n = w^{n-1}w$, para $n > 0$

b) Caso 2: $w = \lambda$

$w^n = \lambda$, para $n > 0$

w^n é indefinida para $n = 0$

Nota-se que a concatenação sucessiva é indefinida para λ^0.

Exemplo: Seja w uma palavra e a um símbolo. Então:

$w^3 = www$

$a^5 = aaaaa$

$a^n = aaa...a$ (o símbolo a repetido n vezes)

1.4 FORMALISMOS GERADORES

Através de um Formalismo Gerador, as sentenças de uma linguagem podem ser sistematicamente geradas. As gramáticas são Geradores, onde o formalismo é obtido 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 as suas sentenças. As regras ou leis de formação são chamadas de produções.

1.4.1 Gramáticas

Uma Gramática é capaz de construir (gerar) conjuntos de cadeias de uma determinada linguagem. Facilitam muito a definição das características sintáticas das linguagens e permitem definir, formalmente e sistematicamente, uma representação finita para linguagens infinitas.

Sendo assim, uma Gramática possui um conjunto finito de variáveis (também chamada de não-terminais) e de símbolos terminais, que pertencem ao alfabeto da linguagem.

As regras de produção definem a geração das palavras da linguagem, e tem a forma $\alpha \rightarrow \beta$. Uma seqüência de regras de produção com o mesmo símbolo do lado esquerdo pode ser abreviada como uma única produção :

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid ... \mid \beta_n$$

Uma regra de produção típica estabelece como uma determinada configuração de variáveis e terminais pode ser rescrita, gerando uma nova configuração.

Formalmente, uma gramática é definida como uma quádrupla G=(V,T,P,S), onde:

V é um conjunto finito de não-terminais, ou variáveis;

T é um conjunto de terminais, ou alfabeto, sendo disjunto de V;

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 $(V \cup T)^*$;

S é um símbolo não-terminal inicial, sendo $S \in V$.

1.4.2 Derivação

A Derivação e uma substituição efetuada de acordo com as regras de produção da gramática. Representa-se esta operação pelo símbolo $\Rightarrow$ com domínio em $(V \cup T)+$ e contra-domínio em $(V \cup T)^*$.

1.4.3 Linguagem Gerada

Uma Linguagem gerada pela gramática G = (V, T, P, S) é denotada por L(G), onde é formada por todas as palavras de símbolos terminais deriváveis a partir do símbolo inicial S.

Exemplo de Gramática Derivação e Linguagem Gerada:

A gramática G=(V, T, P, S) onde:

V = {S, A, B}

T= {a, b}

P = {S$\rightarrow$ bA, A$\rightarrow$ aB, B$\rightarrow$ b}

A derivação da cadeia bab é como segue:

S$\Rightarrow$ bA $\Rightarrow$ baB $\Rightarrow$ bab

A linguagem gerada pela gramática:

L(G) = {bab}

1.5 FORMALISMOS RECONHECEDORES

Um Reconhecedor 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.

1.5.1 Autômato

Um autômato funciona como um reconhecedor de uma determinada linguagem, é um modelo matemático cujo comportamento pode ser descrito com uma seqüência simples, discreta e linear de acontecimentos (estímulos/resposta) no tempo. Em um dado momento, o sistema encontra-se numa determinada configuração interna, à qual dá-se o nome de estado. Através de estímulos, usualmente chamados de símbolos de entrada é determinado o estado posterior.

Pode-se representar um autômato através de um diagrama de transição de estados, como mostra a Figura 1, onde um circulo representa um estado rotulado com os nomes. A seta ligando dois estados, também rotulada com um elemento do alfabeto, mostra cada transição e indica que a leitura deste símbolo do alfabeto da cadeia de entrada, leva o autômato do estado do qual parte a seta para o estado indicado por ela.

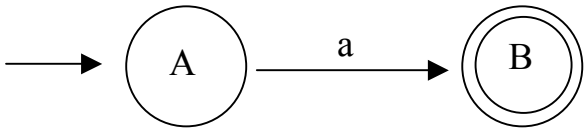


FIGURA 1 – Diagrama de transição de estados para AFD

No diagrama de transições, o estado inicial é representado pelo estado A e indicado por uma seta, sem um estado de origem, como mostra a Figura 2:

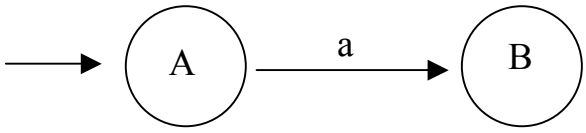


FIGURA 2: Indicação do Estado Inicial em Diagrama de transições

Os estados finais são marcados com círculos duplos ou um circulo de contorno mais forte, demonstrados abaixo pela Figura 3:.

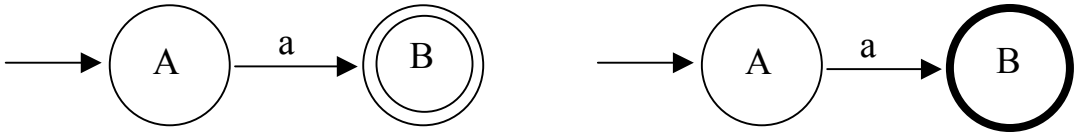


FIGURA 3: Estados Finais em Diagramas de Transições

A função de transição é representada de acordo com a Figura 4, em um diagrama de transição de estados.

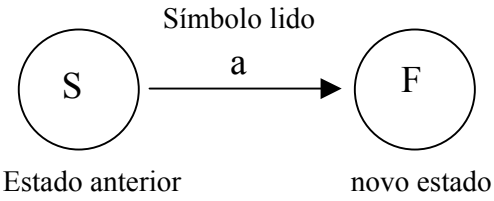


FIGURA 4 - Representação da função de transição de AFD

1.5.2 Autômato com Pilha

O Autômato com Pilha adiciona poder aos Autômatos Finitos através da pilha, ou seja, possui uma memória auxiliar que armazena símbolos durante a análise de cadeias.

A Pilha não possui tamanho fixo nem máximo, onde a base da pilha é fixa e define o seu início, o topo é variável e defini a posição do ultimo símbolo gravado, sendo o primeiro a ser lido.

Um Autômato com pilha não deterministico, permite que possa-se escolher diferentes possibilidades de movimentos a partir de uma determinada configuração do AP. Se qualquer uma delas levar a uma condição de aceitação, então a cadeia analisada faz parte da linguagem do AP. “A facilidade do não determinismo é importante e necessária, pois aumenta o poder computacional dos Autômatos com Pilha” (MENEZES, 2000, p104).

1.5.3 Máquina de Turing

A Máquina de Turing foi proposta por Alan Turing em 1963. Lembra, em muito, os computadores atuais, embora tenha sido proposta anos antes do primeiro computador digital. Pode ser considerada como um modelo formal de procedimento efetivo, algoritmo ou função computável.

CAPÍTULO 2 - HIERARQUIA DE CHOMSKY

Linguagem é um conjunto de cadeias de símbolos de tamanho finito. Partindo dessa definição, um número infinito de linguagens diferentes, podem ser geradas.

NETO[1] (1987), citado por PICOLI (2000, p.8), 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 facilitara 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" .

A classificação das linguagens formais de acordo com a hierarquia de Chomsky, demonstrada na Figura 5, é feita em quatro classes distintas:

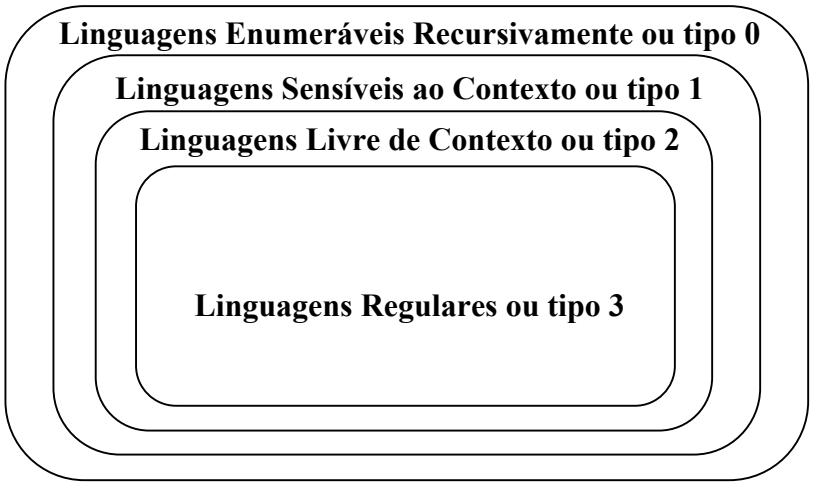


FIGURA 5 - Hierarquia de Chomsky

Pode-se descrever uma relação de continência entre estas classes de linguagens onde a classe tipo 3 é um subconjunto da classe tipo 2, a classe tipo 2 é um subconjunto da classe tipo 1, e a classe tipo 1 é um subconjunto da classe tipo 0. Os modelos formais utilizados para a representação de linguagens são divididos de forma semelhante, sendo que para cada classe de linguagens é estabelecido um modelo formal com poder suficiente para representar as linguagens presentes na classe.

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

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

Embora um modelo para a representação de uma linguagem do tipo 0 seja suficiente para representar qualquer outra, é importante observar que quanto mais abrangente for a classe de linguagens maior será a complexidade do formalismo necessário para representá-la. Adicionalmente, é importante lembrar que a classe de linguagens livres de contexto é de fundamental importância para a informática, pois ela compreende um amplo universo de linguagens capaz de tratar adequadamente os principais problemas sintáticos típicos das linguagens computacionais (MENEZES, 2002, p.85) além do fato de que os formalismos que podem representa-las têm complexidade significativamente menor do que os formalismos necessários para representar as linguagens das classes mais abrangentes (0 e 1). Sendo assim, este trabalho concentrará grande parte do seus estudos em formalismos relacionados às Linguagens Livres de Contexto.

2.1 LINGUAGENS REGULARES

As Linguagens Regulares (ou tipo 3) são representadas por formalismos de pouca complexidade e fácil implementação. Segundo ZANIN (2002) “As linguagens regulares constituem um conjunto de linguagens decidíveis bastante simples e com propriedades bem definidas e compreendidas”.

Os formalismos capazes de representá-las podem ser reconhecedores, como o Autômato Finito Determinístico ou Não Determinístico, ou podem ser geradores como a Gramática Regular.

2.2 LINGUAGENS LIVRES DE CONTEXTO

As Linguagens Livres de Contexto (ou tipo 2) são muito mais flexíveis que as linguagens regulares e muito mais importantes em computação, já que as linguagens de programação são exemplos particulares destas linguagens. Pode-se defini-las de duas maneiras distintas: usando gramáticas ou autômatos.

Usando gramáticas formais podemos dizer que uma linguagem é livre de contexto se pode ser gerada por uma gramática que é livre de contexto. Estas gramáticas são caracterizadas pelo fato de que do lado esquerdo de cada regra há apenas uma variável. Assim, onde quer que uma dada variável X apareça no curso de uma derivação, ela pode ser substituída pelo lado direito de uma regra que tenha X como lado esquerdo. Em outras palavras, a substituição de X não depende de quem são seus vizinhos (o contexto).

Usando autômatos, podemos dizer que uma linguagem é livre de contexto se existe um autômato com pilha não determinístico que a aceita. Estes autômatos são caracterizados pelo fato de terem uma memória infinita (a pilha) cujo acesso é muito restrito: em cada momento o autômato só tem acesso ao último item posto na pilha.

De uma maneira geral, as linguagens de programação são exemplos de linguagens livre de contexto determinísticas, que são aquelas aceitas por autômatos de pilha determinísticos.

MENEZES (2000, p.85) refere-se às Linguagens Livres de Contexto, afirmando que:

> "Seu estudo é de fundamental importância na informática pois:
> - compreende um universo mais amplo de linguagens (comparativamente com regulares) tratando, adequadamente, questões como parênteses balanceados, construções bloco-estruturadas, entre outras, típicas de linguagens de programação como Pascal, C, Algol, etc.;
> - os algoritmos reconhecedores e geradores que implementam as Linguagens Livres de Contexto são relativamente simples e possuem uma boa eficiência;
> - exemplos típicos de aplicações dos conceitos e resultados referentes às Linguagens Livres de Contexto são analisadores sintáticos, tradutores de linguagens e processadores de texto em geral."

2.3 LINGUAGENS ENUMERÁVEIS RECURSIVAMENTE

As Linguagens Enumeráveis Recursivamente representam a classe mais genérica de Linguagens reconhecíveis mecanicamente, ou seja, das linguagens que podem ser definidas e reconhecidas por um algoritmo computacional.

Esta classe de linguagens pode ser definida pelas Gramáticas Irrestritas. São assim denominadas, porque não possuem qualquer restrição no lado esquerdo ou direito de suas regras de produção. Sendo que a regra de produção da forma $\alpha \rightarrow \beta$, onde $\alpha \in (V \cup T)^+$ e $\beta \in (V \cup T)^*$.

O mecanismo de reconhecimento para esta classe de Linguagens é a Máquina de Turing.

2.4 LINGUAGENS SENSÍVEIS AO CONTEXTO

A classe das Linguagens Sensíveis ao Contexto engloba as Linguagens Livres de Contexto, mas é um subconjunto das Linguagens Enumeráveis Recursivamente.

As Linguagens Sensíveis ao Contexto são geradas pelas Gramáticas Sensíveis ao Contexto, que são similares as Gramáticas Irrestritas, tendo como restrição nas regras de produção: $\alpha \rightarrow \beta$ tal que $|\beta| \geq |\alpha|$, exceto quando $\beta = \lambda$ onde $\beta \in (V \cup T)^*$ e $\alpha \in (V \cup T)^+$.

Toda Gramática Sensível ao Contexto é também uma Gramática Irrestrita, mas nem toda Gramática Irrestrita é Sensível ao Contexto.

CAPÍTULO 3 – O AMBIENTE COMPUTACIONAL - SIMFOR

O Simulador de Modelos Formais (SIMFOR), 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 SIMFOR está dividido nos seguintes módulos: Autômato Finito Deterministico, Autômato Finito Não Deterministico (Com transformação para AFD) , Gramática Regular (Com tranformação 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.

A tela principal pode ser verificada através da Figura 6:

FIGURA 6: Tela principal do SIMFOR

3.1 MODELOS FORMAIS PARA DEFINIÇÃO DE LINGUAGENS REGULARES

3.1.1 AUTÔMATO FINITO DETERMINÍSTICO

Um Autômato Finito Determinístico (AFD) permite que uma linguagem regular seja definida de maneira formal através de um modelo matemático, com entradas e saídas discretas. Assume uma única seqüência de estados que é seguida ao fazer o reconhecimento de uma seqüência de símbolos de um dado alfabeto Σ.

Segundo DELAMARO (2002), "um AFD A é um dispositivo que define formalmente uma linguagem L(A) sobre um dado alfabeto Σ. Mais precisamente, dada uma cadeia x $\in$ Σ*, o autômato A deve ser capaz de responder se x pertence ou não à linguagem L(A)".

Desse modo, o AFD é modelo reconhecedor de Linguagens Regulares.

Formalmente defini-se um AFD, como uma quíntupla M=<Σ,S,S_0,δ,F>, onde:

Σ é o alfabeto de entrada;

S é um conjunto finito não vazio de estados;

S_0 é o estado inicial, $S_0 \in S$;

δ é a função transição de estados, definida δ: S x Σ $\rightarrow S$;

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

Σ é o alfabeto com o qual a cadeia de entrada é definida. O Conjunto *S* contem os estados que o autômato pode assumir e um dos estados (S_0) é escolhido com o qual será o estado ativo no inicio do processamento de uma cadeia. Um estado muda para outro através da função δ, sendo definida em δ: S x Σ $\rightarrow S$, ou seja, δ (s,a) = s' leva um par < s , a > em que s é um estado e a é uma letra do alfabeto para um estado s'. O autômato, estando no estado s, ao processar a letra a da cadeia, irá passar para o estado s'. O conjunto *F* indica quais estados de *S* são estados finais. E para que uma cadeia x seja aceita, ela deve levar o autômato do estado S_0 até um dos estados de *F* ao termino do processamento.

Depois de estabelecidos todos os elementos da quíntupla, a análise de cadeias em AFD se dá da seguinte forma:

A cabeça de leitura é posicionada no elemento da cadeia mais à esquerda (no primeiro símbolo da cadeia) e colocamos S_0 como estado ativo.

A análise começa:

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

2. O *próximo estado* é obtido a partir do estado corrente e do símbolo corrente, usando a função de transição do autômato. Se não for encontrado nenhum estado, o autômato pára e a cadeia é rejeitada.

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

4. O *próximo Estado* torna-se no *estado Ativo*, e retorna-se ao passo (1).

5. Se após processar o último símbolo da cadeia:

- o AFD assume um estado final, então o autômato pára e a cadeia é aceita;
- o AFD assume um estado não-final, então o autômato pára e a cadeia é rejeitada.

Exemplos:

EXEMPLO 1: Um AFD $A = <\{0,1\},\{S_0, S_1\}, S_0, \delta, \{S1\}>$ pode definir a Linguagem $L(A)=\{\{0,1\}^+ \mid$ quantidade de 1 é impar$\}$, onde δ é verificada abaixo, ou através do diagrama de estados da Figura 7, eu pela Tabela de Transições :

Função de Transição: $F(S_0,0)=S_0$

$F(S_0,1)=S_1$

$F(S_1,0)=S_1$

$F(S_1,1)=S_0$

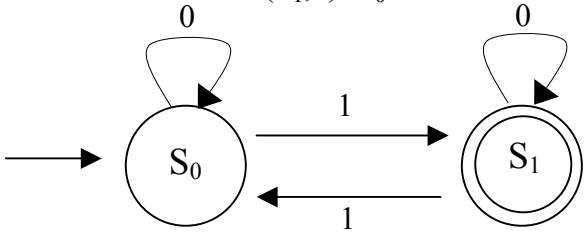


FIGURA 7 - Diagrama de transição de estados

Tabela de transição:

δ	**0**	**1**
S_0	S_0	S_1
S_1	S_1	S_0

EXEMPLO 2: Para reconhecer a Linguagem L = {λ} podemos definir A=<{*a*,*b*}, {S_1, S_2}, S_1, δ, {S_1}>, onde δ é dada pelo diagrama da Figura 8.

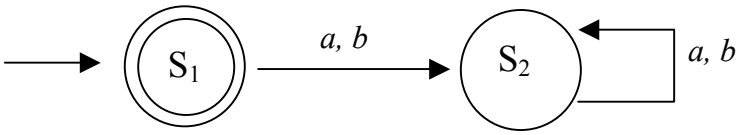


FIGURA 8 - Diagrama de Estados da Linguagem L_2 = { λ }

Nesse caso, o estado inicial S_1 é também o único estado final. O AFD só assume esse estado quando nenhuma letra foi consumida. Qualquer letra irá levar o AFD ao estado S_2. Assim L (A) = L

EXEMPLO 3: Considerando a Linguagem *L* (A) = { b $(ab)^n$ b | n ≥ 0 }, reconhecida pelo AFD A = <{a,b}, {S_0, S_1, S_2, S_3}, S_0, δ, {S_3}>, onde δ é dada pelo diagrama da Figura 9.

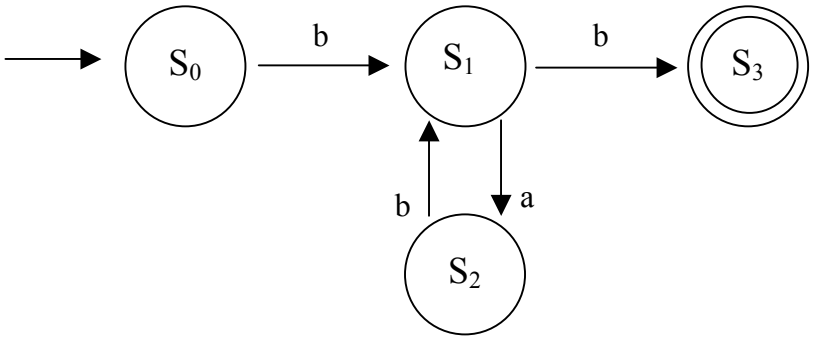


FIGURA 9 - AFD para a Linguagem *L* (A_3) = { b $(ab)^n$ b | n ≥ 0 }

Tabela de transição:

δ	**a**	**b**
$\mathbf{S_0}$	–	S_1
$\mathbf{S_1}$	S_2	S_3
$\mathbf{S_2}$	–	S_1
$\mathbf{S_3}$	–	–

3.1.1.1 IMPLEMENTAÇÃO DE AFD

A Ferramenta do AFD, disponibiliza a analise de cadeias definidas pelo usuário. Divide-se em três partes: a inserção dos símbolos (alfabeto, estado); definição do Estado Inicial, do Estado Final e das Regras de Transição; e análise de cadeias.

O algoritmo abaixo demonstra o processo de verificação de uma cadeia:

Inicio

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.

Utilizando-se do Exemplo 3, pode-se visualizar a Ferramenta AFD através da Figura 10, durante a análise da cadeia *babb*:

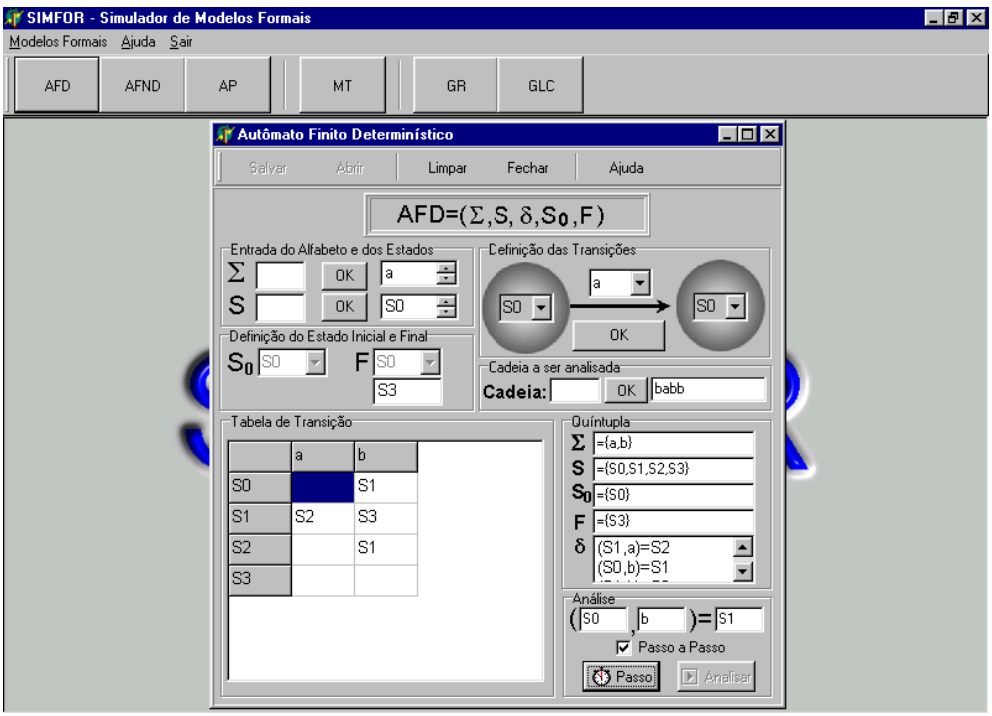


FIGURA 10: Tela do AFD

3.1.2 AUTÔMATO FINITO NÃO DETERMINÍSTICO

O Autômato Finito Não Determinístico (AFND), assemelha-se com o AFD, no que se refere ao modelo matemático. Mas se difere em permitir múltiplos caminhos para processar uma determinada cadeia, ao contrário do AFD.

Um AFD pode simular um AFND, pois nem sempre a facilidade de não determinismo, aumenta o poder de reconhecimento de linguagens de uma classe de autômatos. Mas segundo MENEZES (2000, p.39), "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".

Um Autômato Finito Determinístico é uma quíntupla $M=<\Sigma, S, S_0, \delta, F>$, onde:

Σ é o alfabeto de entrada;

S é um conjunto finito não vazio de estados;

S_0 é um conjunto não vazio de estados iniciais, $S_0 \subseteq S$;

δ é a função transição de estados, definida $\delta: S \times \Sigma \rightarrow \rho(S)$;

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

Σ, S e F são caracterizados exatamente como nos AFD's. Nos AFND's podem haver diversos (mas pelo menos um) estados iniciais, fazendo de S_0 um subconjunto de S. Uma diferença significativa do AFND, é que pode ter mais do que um estado "ativo" ou corrente ao mesmo tempo. A função δ é definida $\delta: S \times \Sigma \rightarrow \rho(S)$, onde $\rho(S)$ é o conjunto de todos os subconjuntos de S.

A função de transição de um AFND, é representada em um diagrama de transição de estados, como ilustrado na Figura 11 abaixo, supondo que:

Função: (S,a) = {q1,q2, ...,qn}

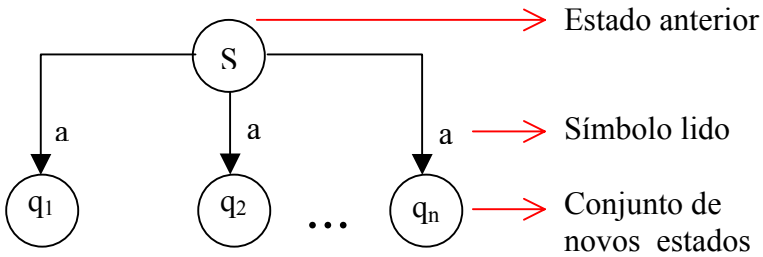


FIGURA 11 - Representação de uma função de transição de AFND

Nos AFND's, ao processar uma dada letra, como demonstra a Figura 12, todas as transições rotuladas com aquela letra, a partir de todos os estados ativos serão efetuados.

A Figura 12a representa esta afirmação, onde inicialmente S_0 e S_1 estão ativos. Quando a letra a é processada, os estados S_2, S_3 e S_4 são ativados, como mostra a Figura 12b.

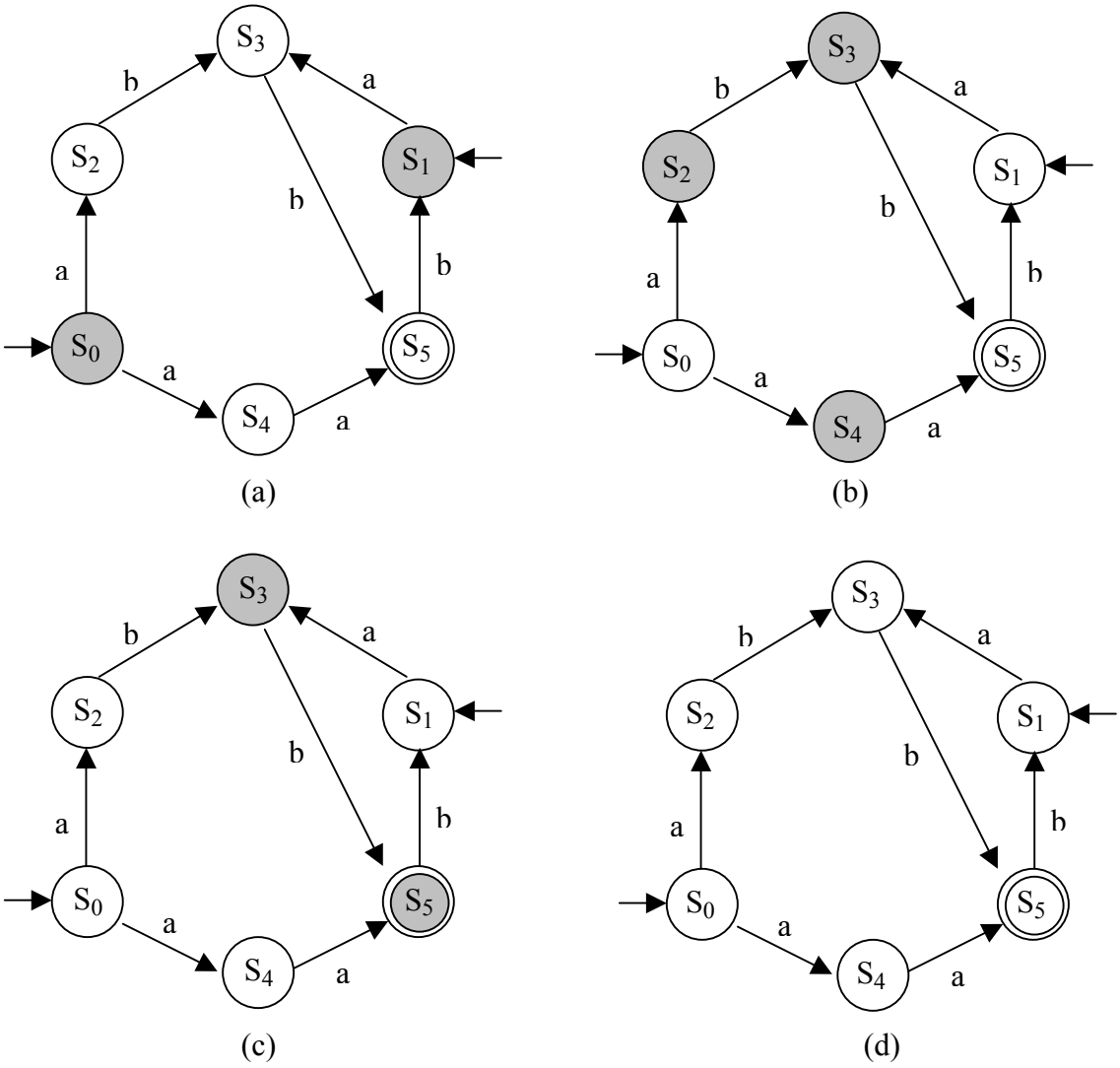


FIGURA 12 - Comportamento de um AFND em Diagramas de Estado

A Figura 12c mostra, ao processar a letra *a*, S_5 é ativado e S_3 continua ativo. Ao processar outro *a* nenhum estado é ativado, como na Figura 12d. Visto que $\delta(S_3, a)= \delta(S_5, a)=0$, pois não há arco saindo dos estados S_3 e S_5, rotulado com a letra *a*, isso faz com que o processamento da cadeia de entrada termine, o que indica que tal cadeia não seja reconhecida pelo AFND (pois não se chegaria a nenhum estado final (*F*)).

Pode-se também, descrever a análise de cadeias em AFD da seguinte forma:

A cabeça de leitura é posicionada no elemento da cadeia mais à esquerda (no primeiro símbolo da cadeia) e ativa-se o conjunto de S_0's .

A análise começa:

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

2. Os *próximos estados ativos* são obtido a partir do estado corrente (ativo) e do símbolo corrente, usando a função de transição do autômato. Se não for encontrado nenhum estado, o autômato pára e a cadeia é rejeitada.

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

4. Retorna-se ao passo 1.

5. Se após processar o último símbolo da cadeia:

- O AFND possui um estado final ativo, então o autômato pára e a cadeia é aceita;
- O AFND não possui um estado final ativo, então o autômato pára e a cadeia é rejeitada.

Exemplos:

EXEMPLO 4: Considerando a linguagem $L(A)=\{x \in \{a,b\}^* \mid aaa$ é subpalavra de $x\}$, sendo definida pelo AFND $A=(\{a,b\}, \{S_0, S_1, S_2, S_f\}, \delta, \{S_0\}, \{S_f\})$, onde δ pode ser verificado abaixo e através do Diagrama de estados apresentados pela Figura 13, e pala Tabela de Transição:

Função de Transição: $F(S_0,a)=\{S_0, S_1\}$

$F(S_0,b)=\{S_0\}$

$F(S_0,a)=\{S_1\}$

$F(S_1,a)=\{S_2\}$

$F(S_2,a)=\{S_f\}$

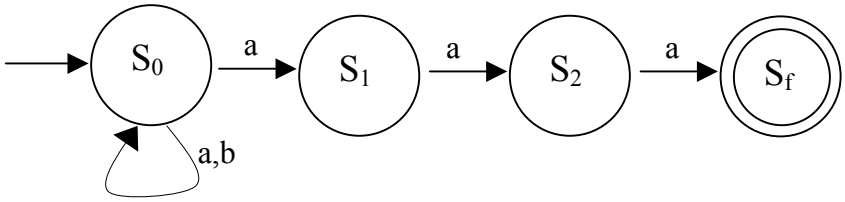


FIGURA 13 - Diagrama de Estados para L (A) = { x ∈ {a,b}* | aaa é subpalavra de x}

Tabela de transição:

δ	**a**	**b**
S0	S_0, S_1	S_0
S1	S_2	–
S2	S_f	–
Sf	–	–

EXEMPLO 5: A linguagem L (A) ={w ∈{0,1}* | w contem 1011} é definida pelo AFND A = <{0,1}, { S_0, S_1, S_2, S_3, S_4}, { S_0}, δ, { S_4}>, onde δ pode ser dada pelo diagrama da Figura 14.

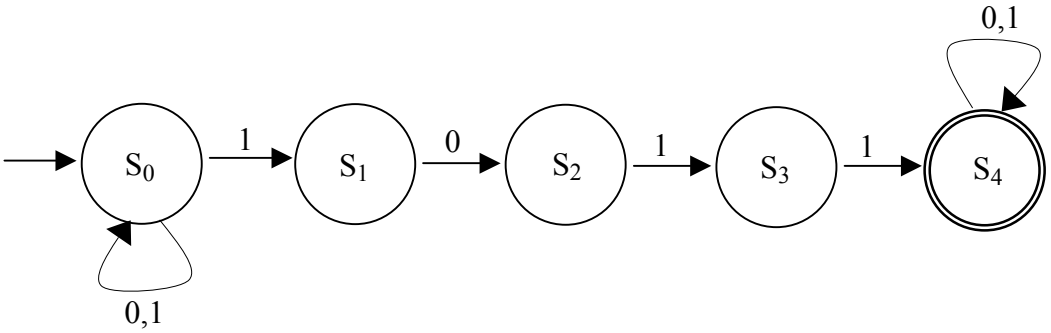


FIGURA 14 – AFND para *L*(A)= {w ∈{0,1}* | w contem 1011}

Tabela de transição:

δ	**0**	**1**
S_0	S_0	S_0, S_1
S_1	S_2	_
S_2	_	S_3
S_3	_	S_4
S_4	S_4	S_4

EXEMPLO 6: A linguagem L(A) = $\{a^m b^n \mid m, n \geq 0 \ \wedge m + n$ é par$\}$ é definida pelo AFND A= <{a,b}, { S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8 }, {S_0}, δ, {S_3, S_6, S_7}>, onde δ pode ser dada pelo diagrama da Figura 15.

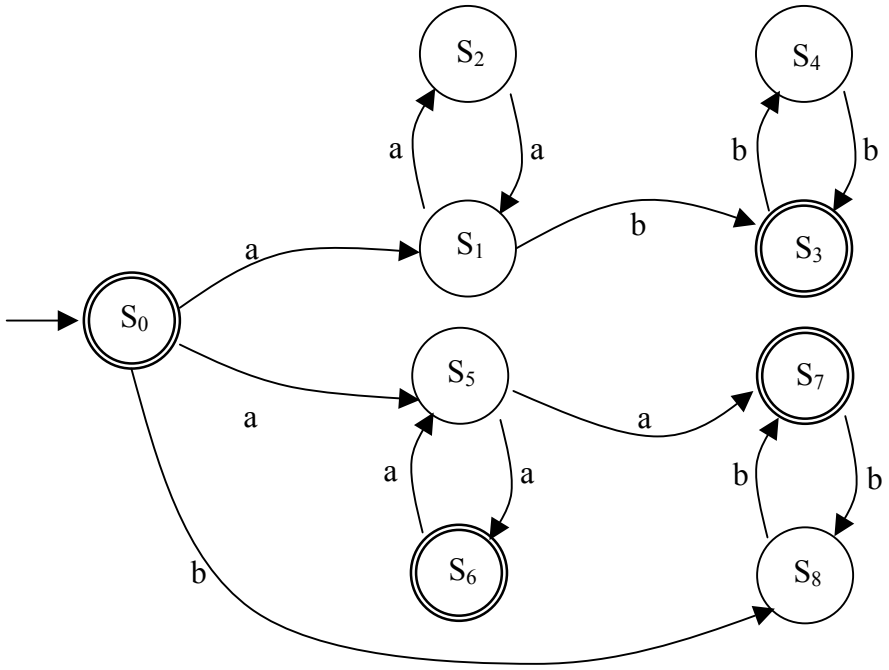


FIGURA 15 - AFD para L(A) = $\{a^m b^n \mid m, n \geq 0 \wedge m + n$ é par$\}$

Tabela de Transição:

δ	a	b
S_0	S_1, S_5	S_8
S_1	S_2	S_3
S_2	S_1	–
S_3	–	S_4
S_4	–	S_3
S_5	S_6, S_7	–
S_6	S_5	–
S_7	–	S_8
S_8	–	S_7

3.1.2.1 TRANSFORMAR AFND EM AFD

Um AFD é um caso especial de AFND, sendo que a diferença entre ambos está no fato do AFD utilizar uma função de transição de cada vez, no reconhecimento de cadeias, enquanto AFND utiliza um conjunto de transições. Pode-se afirmar que todo AFD é um AFND, mas nem todo AFND é AFD.

É possível encontrar um AFD equivalente a um AFND, ou seja, que reconheça a mesma linguagem.

Para iniciar uma transformação de AFND em AFD, e preciso eliminar as transições λ. É importante notar que $|S'| = 2^{|S|}$, ou seja, o tamanho de S' cresce exponencialmente em relação ao tamanho do AFND original. Na maioria das vezes, muitos dos estados S' são inacessíveis e podem ser desconsiderados na obtenção de A', ou seja, considerando apenas os estados alcançáveis a partir do S_0'.

Através dos exemplos a seguir, verifica-se esta transformação de AFND em AFD.

Exemplos:

EXEMPLO 7: Um AFND A=<{a, b}, {S_0, S_1}, { S_0, S_1}, δ, {S_1}> cuja função de transição é dada pelo diagrama da Figura 16a.

O conjunto de estados de A' é dado por ρ ({S_0, S_1}) = {{}, {S_0}, {S_1}, {S_0, S_1}}. O estado inicial é o subconjunto formado por todos os estados iniciais de A, ou seja S_0' = {S_0, S_1}. O conjunto de estados finais, formado por todos os subconjuntos de S que possuem pelo menos um estado final de A, ou seja, F'= {{S_1}, {S_0, S_1}}. E a função δ' é determinada por:

δ' ({}, a) = {}

δ' ({S_0}, a) = δ (S_0, a) = {S_0, S_1}

δ' ({S_1}, a) = δ (S_1, a) = {S_0}

δ' ({S_0, S_1}, a) = δ (S_0, a) ∪ δ (S_1, a) = {S_0, a}

δ' ({}, b) = {}

δ' ({S_0}, b) = δ (S_0, b) = { S_1}

δ' ({S_1}, b) = δ (S_1, b) = {}

δ' ({S_0, S_1}, b) = δ (S_0, b) ∪ δ (S_1, b) = { S_1}

Obtendo-se o diagrama da Figura 18b:

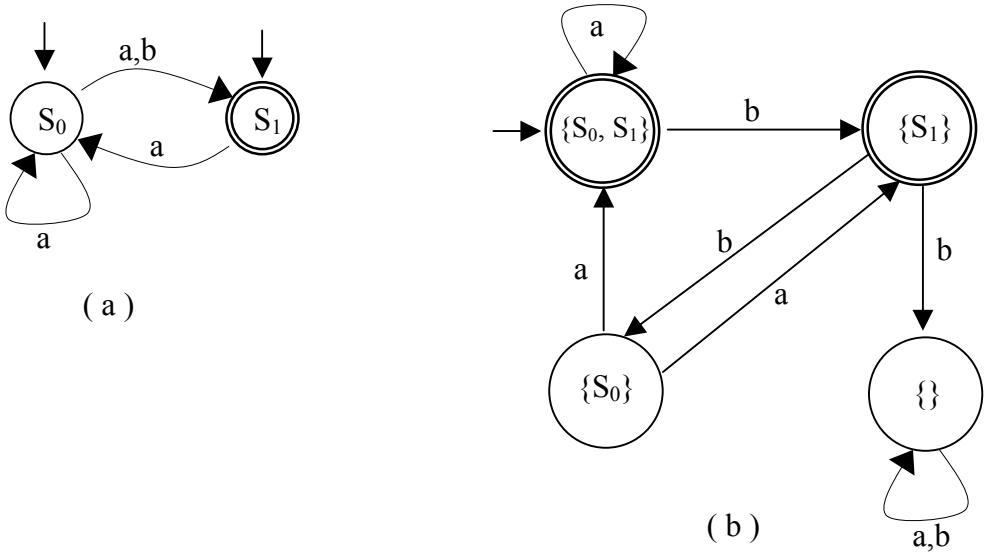


FIGURA 16: AFND com AFD equivalente

EXEMPLO 8: Um AFND A=<{a, b}, {S_0, S_1, S_2, S_3,}, {S_0}, δ, {S_3}> cuja função de transição é dada pelo diagrama da Figura 17, tem seu AFD equivalente demonstrado na Figura 18:

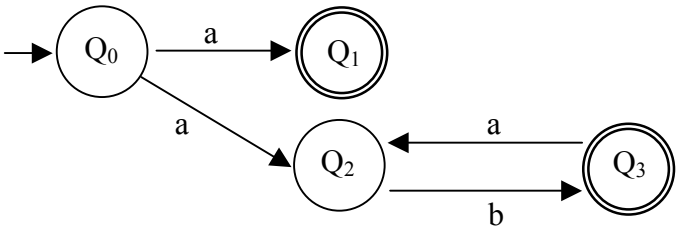


FIGURA 17: AFND para L(A) = {a $(ab)^n$ | n ≥ 0}

A partir deste AFND pode-se obter o seguinte AFD:

δ' ({S_0}, a) = {S_0, S_2}

δ' ({S_0}, b) = {}

δ' ({S_1, S_2}, a) = {}

δ' ({S_2, S_2}, b) = {S_3}

δ' ({}, a) = δ ({}, b) = {}

δ' ({S_3}, a) = {S_2}

δ' ({S_3}, b) = {}

δ' ({S_2}, a) = {}

δ' ({S_2}, b) = {S_3}

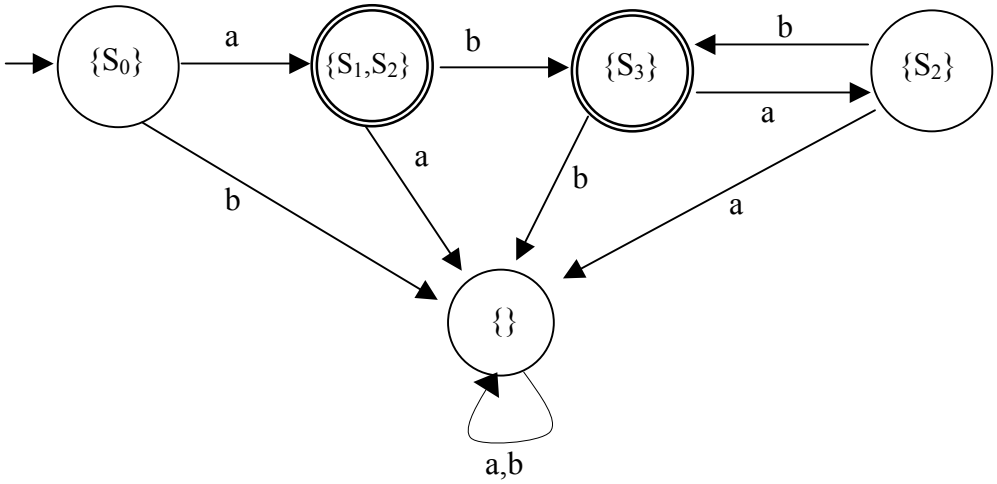


FIGURA 18: AFD equivalete ao AFND

3.1.2.2 IMPLEMENTAÇÃO DE AFND

A Ferramenta do AFND, é semelhante a Ferramenta do AFD, mas possibilita que vários caminhos sejam percorridos durante a analise de cadeias, ao invés de possuir apenas um Estado Ativo, como no AFD, possibilita que um conjunto de estados sejam acionais. Divide-se, assim como o AFD, em três partes: a inserção dos símbolos (alfabeto, estado); definição do Estado Inicial, do Estado Final e das Regras de Transição; análise de cadeias.

O processo de verificação sintática de uma cadeia é apresentado abaixo de forma algorítmica:

```
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.
```

Utilizando-se do Exemplo 5, pode-se visualizar a Ferramenta AFD através da Figura 19, durante a análise da cadeia 101101:

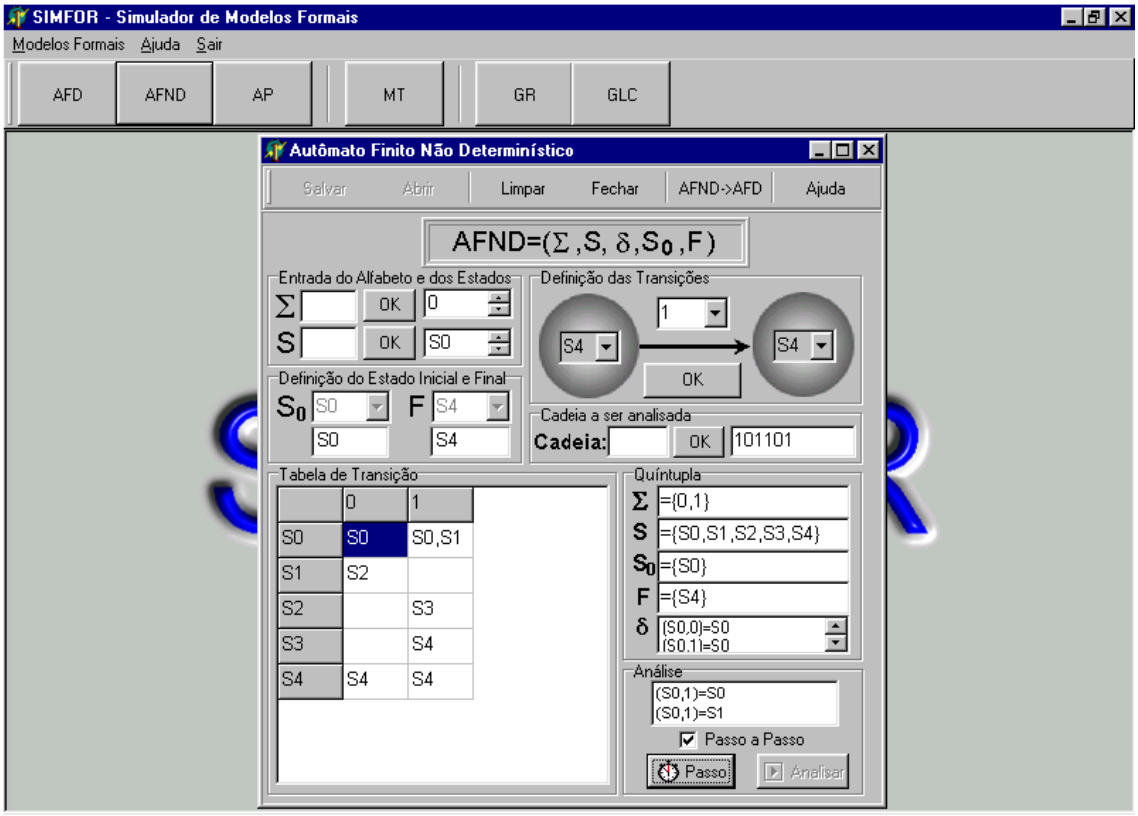


FIGURA 19: Tela do AFND

No SIMFOR encontra-se também disponível uma ferramenta que possibilita a transformação de AFND em AFD, sendo que, ao acionar a transformação, novos estados são criados, assim como o estado inicial, estado final e regras de transição, através da combinação dos estados de AFND.

Ao final da transformação é apresentado o AFD, construído a partir de um AFND.

Esta transformação pode ser verificada através do algoritmo apresentado a seguir:

```
Inicio
    Estados_AFD ← ρ(estados_AFND);
    Para x variar do Símbolo inicial do alfabeto até o símbolo finald o alfabeto
        Faça  Para y variar do Estado_AFND inicial até Estado_AFND final
                  Faça  Se existe δ_AFND (x,y);
                            Então  k ← Estado_AFD ⊃ y;
                                   δ_AFD ← (x, k );
    r ←1;
    Repita
        s ← 1;
        Repita
            Se Estado Inicial_AFND[s] ⊂ Estado_AFD[r]
                Então  ACEITA;
                Senão REJEITA;
            s ← s+1;
        Até REJEITA ou s > numero Estados Iniciais_AFND;
    Se ACEITA
            Então Estado Inicial_AFD ← Estado_AFD[r];
    Até ACEITA ou r > numero Estados_AFD;
    Elimina_Estado_inalcançável(Estados_AFD);
    n ← numero de Estados Finais_AFD;
    r ←1;
    Repita
        s ← 1;
        Repita
            Se Estado Final_AFND[s] ⊂ Estado_AFD[r]
                Então  ACEITA;
                Senão REJEITA;
            s ← s+1;
        Até ACEITA ou s > numero Estados finais_AFND;
```

Se ACEITA

Então { Estado Final_AFD$_{[n]}$ ← Estado_AFD$_{[r]}$;
n ←n+1;

Até ACEITA ou r > numero Estados_AFD;

Fim.

* Elimina_Estado_inalcançável: função que elimina todo os estados de uma AFD que não podem ser alcançados a partir do estado inicial.

A Figura 20 mostra esta transformação, onde está sendo aplicado o exemplo 5.

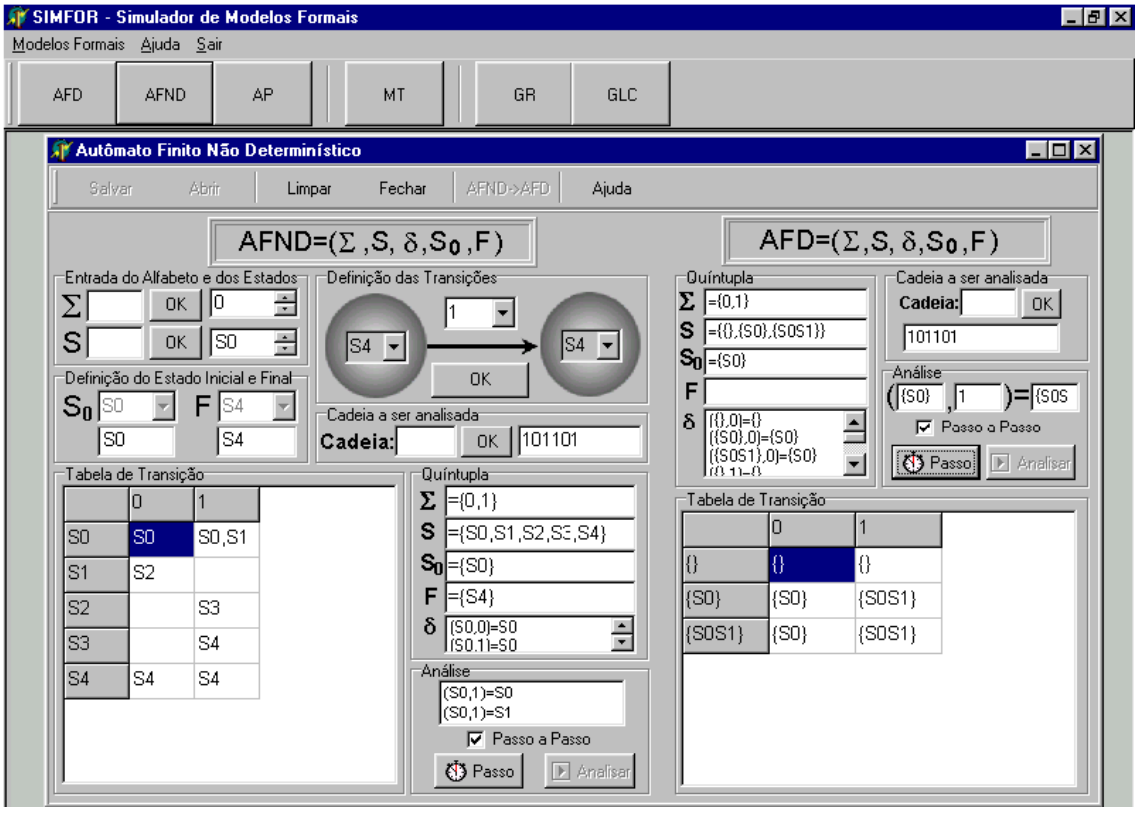


FIGURA 20: Tela do AFND com transformação em AFD

3.1.3 GRAMÁTICA REGULAR

Uma gramática é um mecanismo que nos permite definir formalmente uma linguagem. Através dela pode-se gerar todas as sentenças da linguagem definida por ela.

"Um ponto a se notar é que, contrariamente aos Autômatos Finitos, as Gramáticas possuem um caráter 'gerador' de cadeias. Por isso é comum dizer-se que um Autômato Finito 'reconhece' uma linguagem e uma Gramática 'gera' uma linguagem". (DELAMARO, 2002)

As Gramáticas Regulares são gramáticas mais restritas e mais próximas das formas de produção correspondentes às transições de autômatos finitos.

Define-se uma Gramática Regular por uma quádrupla G = < V, T, *P*, *S* >, onde:

V é um conjunto não vazio de símbolos não terminais;

T é o conjunto de símbolos terminais ($T \cap V = \varnothing$);

P é um conjunto de produções da forma $\alpha \rightarrow \beta$

S é o símbolo inicial ($S \in V$);

Dentre as restrições existentes na Gramática Regular, a utilizada neste trabalho é a Gramática Linear Unitária a Direita (GLUD), onde em uma Gramática G= (V,T,P,S), sejam A e B elementos de V e w uma palavra de T*, as regras de produção são da forma:

$A \rightarrow wB$ ou $A \rightarrow w$ onde $| w | \leq 1$ ou $w = \lambda$

As cadeias que pertencem à linguagem definida por uma gramática são compostos exclusivamente por símbolos terminais, ou seja, do conjunto T. Desse modo, para se chegar a uma cadeia da linguagem aplica-se produções que substituam todos os símbolos não terminais, até que se obtenha uma cadeia que possua apenas terminais, ou seja faz-se a Derivação.

Uma regra $\alpha \rightarrow \beta$ indica que α pode ser substituído por β sempre que α aparecer. Enquanto houver símbolo não-terminal na cadeia em derivação, esta derivação não terá terminado.

O símbolo inicial é o símbolo através do qual deve iniciar o processo de derivação de uma sentença, onde os elementos de T são terminais, podendo ser representados por letras minúsculas (a, b, c, d...) e os elementos de V são os não-terminais, representados por letras maiúsculas (A, B, C, D ...). as cadeias mistas, isto é, aquelas que contem símbolos de T e símbolos de V (cadeias pertencentes à $(T \cup V)^{+}$) , podendo ser representados por letras gregas (α, β, γ, δ...).

Exemplos:

EXEMPLO 9: A linguagem L={ab} é gerada através da gramática G = ({S, A, B}, {a, b}, P, S), com as regras de derivação:

P = { $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$}

Derivação:

- $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$

EXEMPLO 10: A linguagem L(G = $\{a\ (ba)^n \mid n \geq 0\}$ é gerada pelas G=({S, A, B}, {a, b}, P, S) onde P possui as seguintes regras de produção:

$S \rightarrow aA$

$A \rightarrow bB \mid \lambda$

$B \rightarrow aA$

Derivação:

- $S \Rightarrow aA \Rightarrow a$
- $S \Rightarrow aA \Rightarrow abB \Rightarrow abaA \Rightarrow aba$
- $S \Rightarrow aA \Rightarrow abB \Rightarrow abaA \Rightarrow ababB \Rightarrow ababaA \Rightarrow abab$
- ...

EXEMPLO 11: A Linguagem L (G) = $\{a^n b \mid n \geq 0\}$, pode ser definida através da gramática G = ({S}, {a, b}, P, S) com as regras de derivação:

$P = S \rightarrow aS$

$S \rightarrow b$

Derivação:

- $S \Rightarrow aS \Rightarrow ab$
- $S \Rightarrow b$
- $S \Rightarrow aS \Rightarrow aaS \Rightarrow aab$
- $S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaaS \Rightarrow aaaab$
- ...

EXEMPLO 12: A gramática G = ({S, A}, {a, b, c}, P, S), permite a definição da linguagem L (G) = $\{a^n bc \mid n = 0\}$, através das regras de derivação:

$P = S \rightarrow aS \mid bA$

$A \rightarrow c$

A cadeia terminal gerada:

- $S \Rightarrow aS \Rightarrow abA \Rightarrow abc$

3.1.3.1 IMPLEMENTAÇÃO DA GR

A Ferramenta para a definição de Gramáticas Regulares possibilita a construção de gramáticas regulares diretamente ou através da construção de Autômatos Finitos (Deterministicos e Não Determinísticos) e vice-versa, para a analise de cadeias.

Para construir uma GR a partir de um AFND, é preciso transformá-lo em um AFD, pois em uma GR há apenas um estado inicial, ao contrário do AFND, onde pode haver mais de um estado inicial.

O algoritmo abaixo demonstra o processo de transformação de uma GR para um AFD.

n ← numero de estados finais;

Inicio

Para x variar do Símbolo inicial do alfabeto até o símbolo finald o alfabeto

Faça para y variar do Estado_GR inicial até Estado_GR final

Faça Se existe δ_GR (x,y);

Então Se $alfabeto_{[x]}$ = '&'

Então $Estado\_final_{[n]} \leftarrow estado\_GR_{[y]}$;

n ← x+1;

Senão δ_GR ← (x, k);

Senão REJEITA;

Estado_Inicial_AFD←Estado_Inicial_GR;

Fim.

Através da Figura 21, pode-se verificar a construção de uma GR, apresentada no Exemplo 10 e também sua transformação em AFD, analisando a cadeia aba:

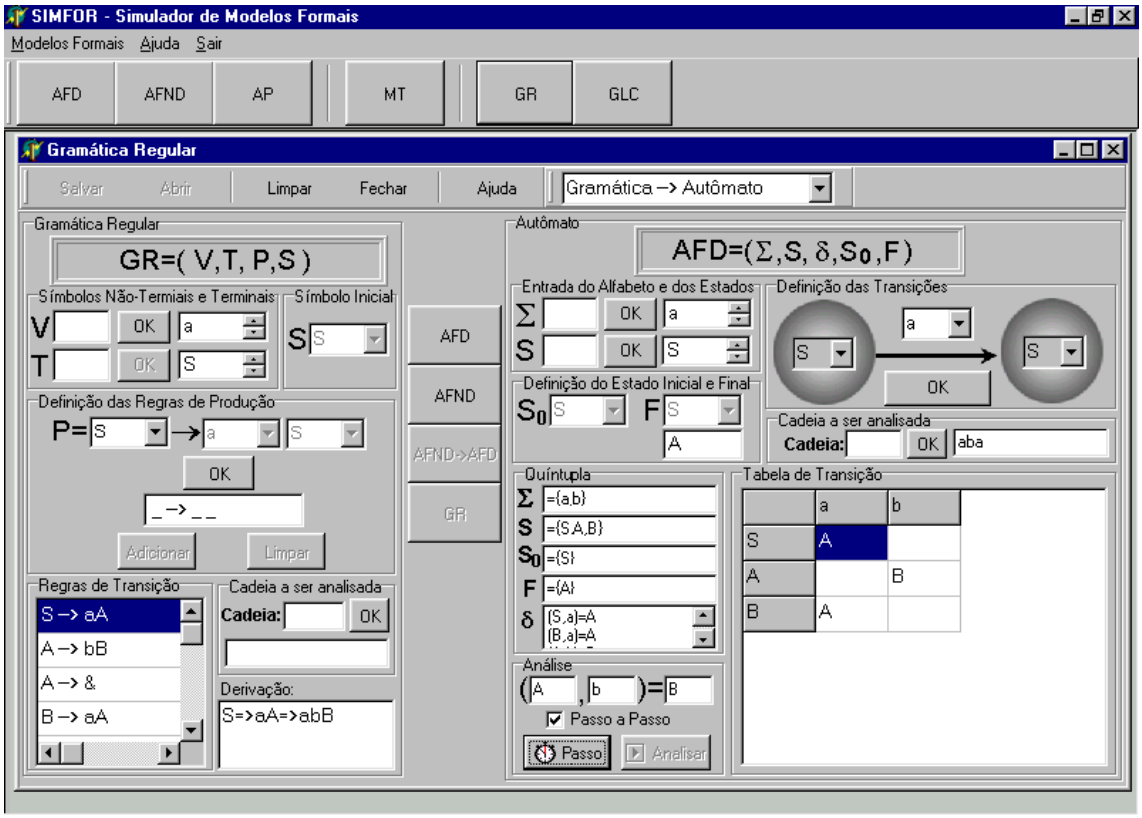


FIGURA 21: Tela da GR

3.2 MODELOS FORMAIS PARA DEFINIÇÃO DE LINGUAGENS LIVRES DE CONTEXTO

3.2.1 AUTOMATO COM PILHA

Os Autômatos Finitos, tem poder suficiente para reconhecer as Linguagens Regulares. Entretanto, são mecanismos limitados para reconhecer linguagens mais complexas como as Linguagens Livres de Contexto.

Estendendo o conceito dos Autômatos Finitos, obtem-se um novo tipo de mecanismo reconhecedor que possa reconhecer as cadeias pertencentes a uma Linguagem Livre de Contexto qualquer. O problema com os Autômatos Finitos é a memória reduzida. Um AF em cada instante só sabe o estado atual, o próximo símbolo a ser consumido e o próximo estado quando o símbolo for consumido. Para as LLC's isto é insuficiente. Então, acrescentando-se ao AF uma memória adicional estruturada em forma de Pilha obtêm-se uma nova categoria de Autômatos chamados Autômatos com Pilha.

Um Autômato com Pilha não determinístico (AP) é uma sextupla P = $<\sum, \Gamma, S, S_0, \delta, B>$ onde:

$\sum$ é o alfabeto de entrada do *AP*;

Γ é o alfabeto da pilha;

S é o conjunto finito não vazio de estados do AP;

S_0 é o estado inicial, $S_0 \in S$;

δ é a função de transição de estados, δ: *S* x $(\Sigma \cup \{\lambda\})$x $\Gamma \rightarrow$ conjunto de subconjuntos finitos de S x Γ *;

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

Cada transição terá como entrada o estado atual, o próximo símbolo na entrada e o símbolo no topo da pilha, gerando um próximo estado e uma operação na pilha.

Assim δ pode ser dada por uma tripla $<s, x>/\alpha$, onde *s* é um estado corrente, *x* é o elemento da cadeia que falta ser processada e α é o conteúdo da pilha,

como o topo no início de α. O Autômato com Pilha, anda ou move-se de uma configuração para outra através da aplicação de uma função de transição.

Na análise da cadeia, verifica-se o conteúdo do topo da pilha, o AP pode ainda retirar o símbolo do topo e colocar em seu lugar uma cadeia $\alpha \in \Gamma^*$. Se $\alpha = A$, $A \in \Gamma$, então o símbolo do topo é substituído por A e a cabeça de leitura/escrita da pilha continua posicionada no mesmo lugar. Se $\alpha = A_1 A_2 .. A_n$, $n > 1$ então o símbolo do topo da pilha é retirado, sendo A_n colocado no seu lugar, A_{n-1} na posição seguinte, e assim por diante. A cabeça é deslocada para a posição ocupada por A_1 que é então o novo topo da pilha. Se $\alpha = \lambda$ então o símbolo do topo da pilha é retirado, fazendo a pilha decrescer e o topo ficar numa posição imediatamente inferior à anterior.

Assim como nos AF's, o funcionamento dos AP's é determinado pela função de transição. A diferença aqui é que δ é função do estado corrente, da letra corrente na fita de entrada e do símbolo no topo da pilha. Além disso, ela determina não só o próximo estado que o AP assume, mas também como o topo da pilha dever ser substituído. O AP inicia sua operação num estado denotado S_0 e com um único símbolo na pilha, denotado por B.

Os AP's não possuem estados finais como os AF's. Assim, um cadeia x é aceita se, ao chegar ao final da cadeia de entrada, a pilha estiver vazia, independentemente do estado em que o AP se encontra;

Um AP pode ser representado por um diagrama de transição de estados semelhante ao utilizado para AF's, como mostra a Figura 22, onde cada estado é representado por um circulo e cada possível movimento, representado por um arco (ou reta) direcionado entre dois estados. Cada arco é rotulado com uma letra do alfabeto e uma letra que deve estar no topo da pilha para que o movimento possa ocorrer e com o valor da cadeia a ser colocado no topo da pilha, em substituição à letra lá existente.

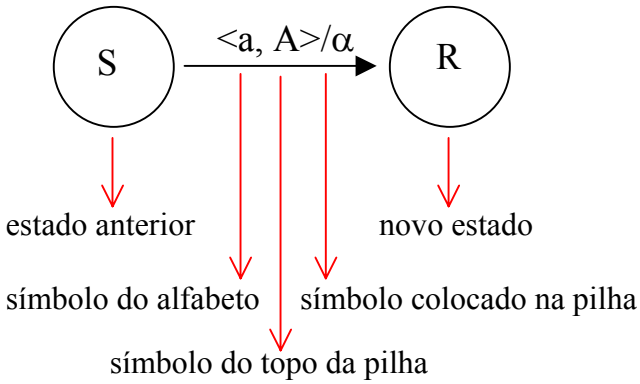


FIGURA 22 - Representação do diagrama de transição de estados

Pode-se também, descrever a análise de cadeias em AP da seguinte forma:

A cabeça de leitura é posicionada no elemento da cadeia mais à esquerda (no primeiro símbolo da cadeia) e colocamos S_0 como estado ativo e B na base da pilha.

A análise começa:

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

2. O *próximo estado* e a *próximo topo da pilha,* são obtidos a partir do estado corrente, do símbolo corrente e do símbolo da base da pilha, usando a função de transição do autômato. Substitui-se o símbolo do topo da pilha pelo o *próximo topo da pilha* encontrando, podendo causar um empilhamento ou desempilhamento, se o símbolo encontrado for λ. Se não for encontrado nenhum estado, o autômato pára e a cadeia é rejeitada.

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

4. O *próximo Estado* e torna-se no *estado Ativo* e o *próximo topo da pilha* torna-se o símbolo do topo da pilha, e retorna-se ao passo (1).

5. Se após processar o último símbolo da cadeia:

- nenhuma Pilha do AP estiver vazia, o Autômato pára e a cadeia é aceita;
- nenhuma Pilha do AP não estiver vazia , então o autômato pára e a cadeia é rejeitada.

Exemplos:

EXEMPLO 1: Para a LLC = $\{a^n b^n \mid n \geq 0\}$, em que P = <{a, b}, {*A*, *B*}, $\{S_0, S_1\}$, S_0, δ, *B* > onde δ é apresentada abaixo e através da Figura 23:

$\delta (S_0, a, B) = \{<S_0, A>\}$

$\delta (S_0, a, A) = \{<S_0, AA>\}$

$\delta (S_0, b, A) = \{<S_1, \lambda>\}$

$\delta (S_1, b, A) = \{<S_1, \lambda>\}$

$\delta (S_0, \lambda, B) = \{<S_1, \lambda>\}$

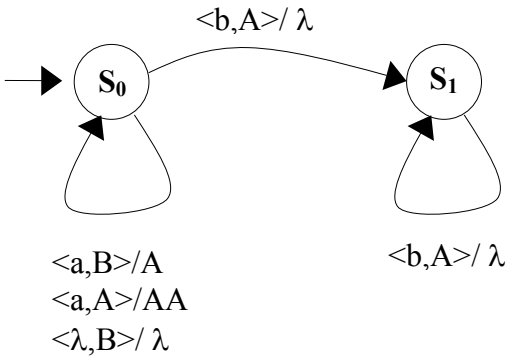
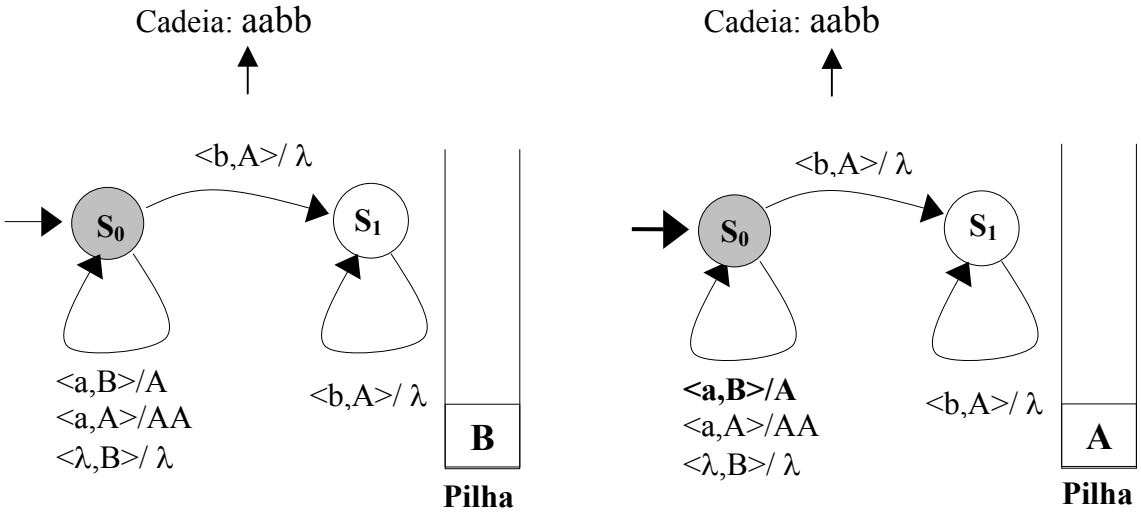


FIGURA 23 - Diagrama de estados de AP para LLC = $\{a^n b^n \mid n \geq 0\}$

A Figura 24 mostra o processamento da cadeia aabb no AP citado acima:



(a) (b)

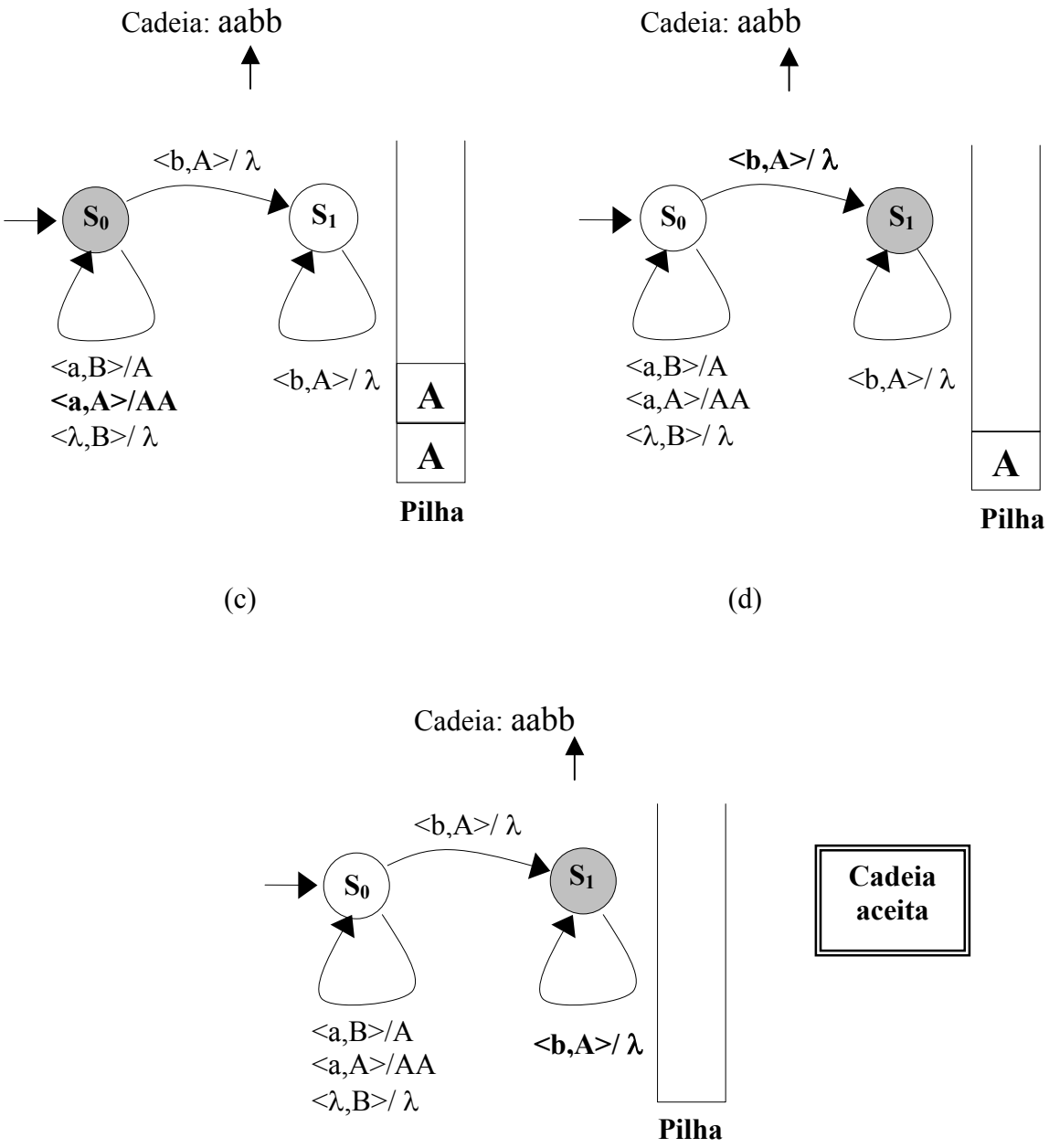


FIGURA 24 - Processamento de uma cadeia de AP

Nos AP's não determinísticos, podem existir diversos movimentos possíveis. Se algum deles leva o AP a < s, λ, λ > então a cadeia é aceito. A cadeia só é rejeitada se nenhuma seqüência de movimentos leva a < s, λ, λ >.

EXEMPLO 2: Para representar a linguagem $L(P_1) = \{a^n b^i c^m \mid i = n + m\}$, constrói-se o AP $P = < \{a, b, c\}, \{B, N, I\}, \{S_0, S_1, S_2, S_3\}, S_0, \delta, B >$ com δ representada na Figura 25. No estado S_0 são lidos todos os *a*'s para cada um deles inclui-se um *N* na pilha. Ao encontrar-se um *b* passa-se para S_1 ou S_2. Se algum *a* foi lido (existem *N*'s na pilha) passa-se para S_1 onde um *N* é desempilhado para cada *b* lido. Se nenhum *a* foi lido então passa-se direto de S_0 para S_2 onde são colocados *I*'s na pilha, um para cada *b* lido na entrada. Em S_3 um *I* é retirado para cada *c* lido, fazendo com que o número de *a*'s mais o número de *c*'s seja igual ao número de *b*'s para que a cadeia seja aceito.

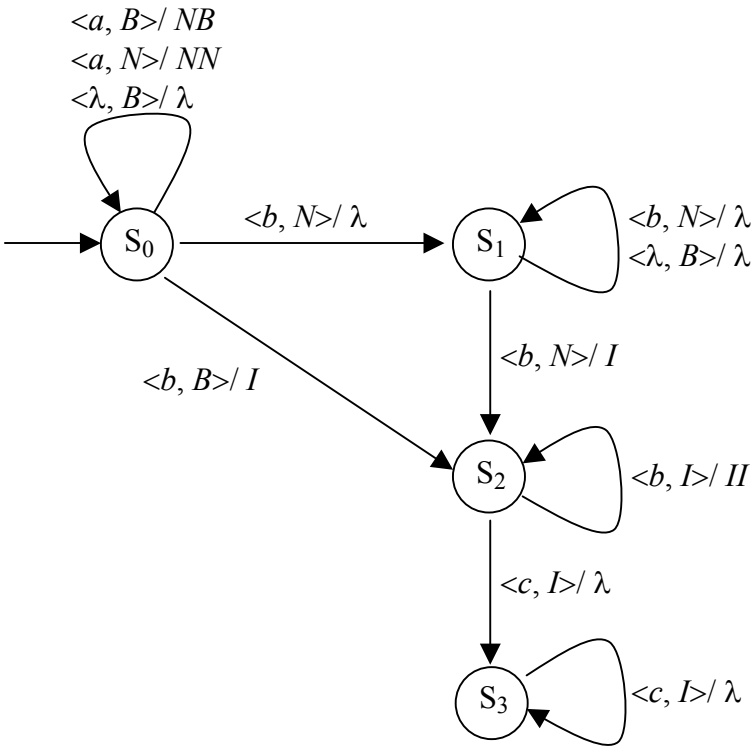
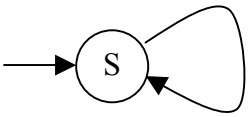


FIGURA 25 – AP que reconhece $L(P)=\{a^n b^i c^m \mid i = n + m\}$

EXEMPLO 3: Considerando a linguagem L = {x ∈ {a, b}* | $|x|_a = |x|_b$}, com AP que a define P = <{*a*, *b*}, {*A*, *B*}, {*S*, R}, *S*, *δ*, *B* > onde δ é apresentada abaixo e pela Figura 26:

δ (S, a, C) = { <S, AC>}
δ (S, b, C) = { <S, BC>}
δ (S, λ, C) = { <S, λ >}
δ (S, a, A) = { <S, AA>}
δ (S, b, A) = { <S, λ >}
δ (S, a, B) = { <S, λ >}
δ (S, b, B) = { <S, BB>}



< a, C >/ AC
< b, C >/ BC
< λ, C >/ λ
< a, A >/ AA
< b, A >/ λ
< a, B >/ λ
< b, B >/ BB

FIGURA 26 – AP que reconhece L = {x ∈ {a, b}* | $|x|_a = |x|_b$}

3.2.1.1 IMPLEMENTAÇÃO DO AP

A Ferramenta do AP, disponibiliza a analise de cadeias definidas pelo usuário para Linguagens Livres de Contexto. Divide-se em três etapas: a inserção dos símbolos (alfabeto, estado), definição do Estado Inicial, do Estado Final e das Regras de Transição e analise de cadeias.

Utilizando-se do Exemplo 1, pode-se visualizar a Ferramenta AFD através da Figura 27, durante a análise da cadeia aaabbb:

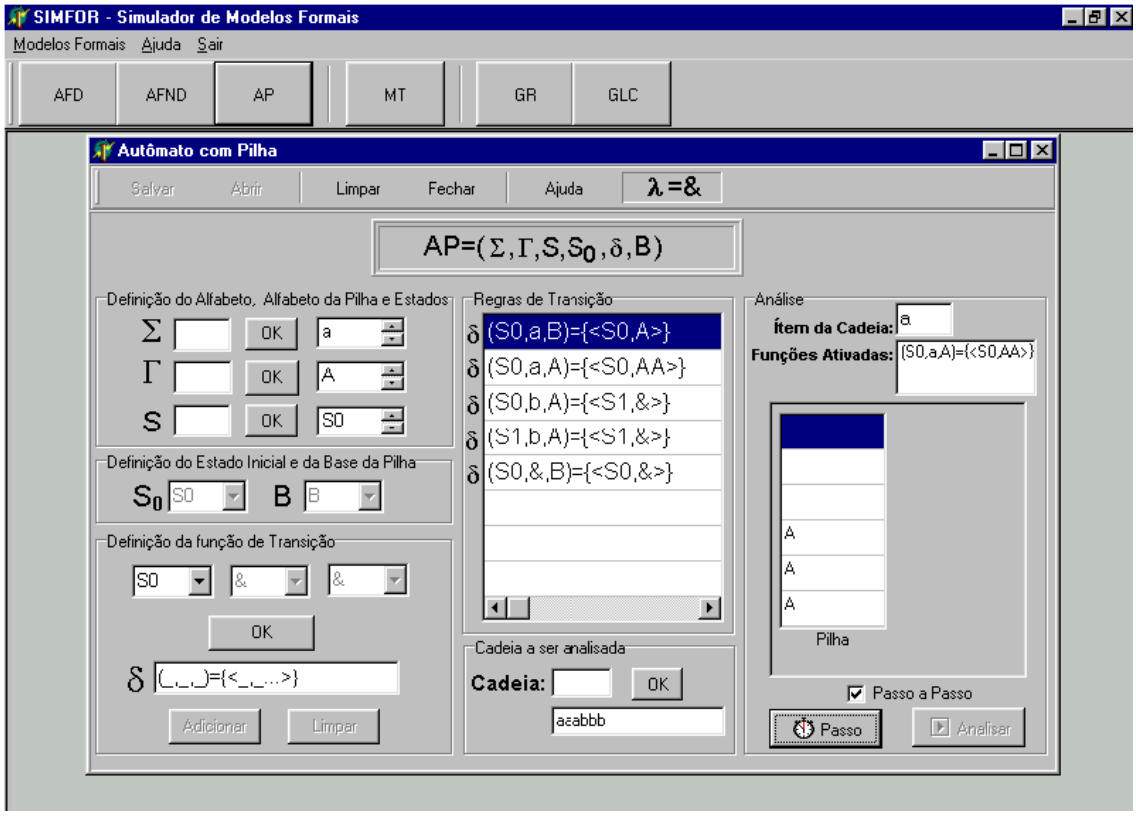


FIGURA 27: Tela do AP

3.2.2 GRAMÁTICA LIVRE DE CONTEXTO

Uma Gramática Livre de Contexto (GLC) é uma quádrupla GLC = < V, T, *P*, *S* >, onde:

V é um conjunto não vazio de símbolos não terminais;

T é o conjunto de símbolos terminais ($T \cap V = \varnothing$);

P é um conjunto finito de pares, do tipo $A \rightarrow \alpha$ onde $\alpha \in (V \cup T)^*$

S é o símbolo inicial ($S \in V$);

MENEZES (2000, 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 $A \rightarrow \alpha$. 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."

Observando a definição das GLC's fica claro que as Gramáticas Regulares atendem a esta definição, ou seja, as GLC's são uma classe mais ampla de gramáticas, pois toda GR é uma GLC, mas nem toda GLC é uma GR. As GR's constituem um subconjunto de GLC's.

Exemplos:

EXEMPLO 4: A linguagem $L = \{a^n b^n \mid n \geq 0\}$ é definida através da Gramática G = ({S}, {a, b}, P, S), onde P possui as seguintes regras de produção:

$P = \{ S \rightarrow aSb \mid S \rightarrow \lambda \}$

A palavra (sentença) aabb pode ser gerada pela seguinte seqüência de derivação:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

Este exemplo permite estabelecer analogia entre $a^n b^n$ e linguagens que possuem duplo balanceamento como:

a) Linguagens bloco-estruturadas do tipo $BEGIN^n END^n$;

b) Linguagens com parênteses balanceados na forma $(^n)^n$.

EXEMPLO 5: A linguagem L gerada pela Gramática Livre de Contexto abaixo é composta de expressões aritméticas contendo colchetes balanceados, dois operadores e um operando:

G = ({E}, {+, *, [,], x}, P, E), onde:

P = {E $\rightarrow$ E+E | E*E | [E] | x}

A expressão [x + x] * x pode ser gerada pela seqüência de derivação:

E $\Rightarrow$ E*E $\Rightarrow$ [E]*E $\Rightarrow$ [E+E]*E $\Rightarrow$ [x+E]*E $\Rightarrow$ [x+x]*x

EXEMPLO 6: A Gramática G = ({S, Z}, {a, b}, P, Z), com as seguintes regras de produção:

P = { Z $\rightarrow$ λ | S

S $\rightarrow$ aSb | ab}

Gera os *cadeia* λ , ab, aabb, aaabbb:

- Z $\Rightarrow$ λ
- Z $\Rightarrow$ S $\Rightarrow$ aSb $\Rightarrow$ ab
- Z $\Rightarrow$ S $\Rightarrow$ aSb $\Rightarrow$ aaSbb $\Rightarrow$ aabb
- Z $\Rightarrow$ S $\Rightarrow$ aSb $\Rightarrow$ aaSbb $\Rightarrow$ aaaSbbb $\Rightarrow$ aaabbb

3.2.2.1 FORMA NORMAL DE CHOMSKY

Segundo MENEZES (2000, p.97), "as Formas Normais estabelecem restrições rígidas na definição das produções, sem reduzir o poder de geração das Gramáticas Livres de Contexto".

A Forma Normal de Chomsky condiciona uma Gramática Livre de Contexto a estar da forma:

$$A \rightarrow BC \quad \text{ou} \quad A \rightarrow a$$

onde A, B, C são símbolos não terminais e *a* é um símbolo terminal.

Para transformar uma GLC, cuja linguagem gerada não possua a palavra vazia, em uma gramática na Forma Normal de Chomsky é necessário:

1. excluir as produções vazias (como $A \rightarrow \lambda$) e produções da forma $A \rightarrow B$, a gramática resultante desta etapa seria: $G_1 = (V_1, T_1, P_1, S)$;

2. garantir que o lado direito das produções de comprimento maior ou igual a dois seja composto somente por variáveis, substituindo um símbolo não terminal *a* por uma variável intermediária X, incluindo-se assim, uma nova produção como $Xa \rightarrow a$. A gramática resultante desta etapa seria: $G_2 = (V_2, T_1, P_2, S)$, onde V_2 E P_2 são construídos como mostra o algoritmo abaixo (MENEZES, 2000, p. 99):

C_a: variável intermediária;

$V_2 = V_1$;

$P_2 = P_1$;

Para toda $A \rightarrow X_1 X_{2...} X_n \in P_2$ tal que $n \geq 2$

Faça se para $r \in \{1, ..., n\}$, X_r é um símbolo terminal

então (suponha $X_r = a$)

$V_2 = V_2 \cup \{C_a\}$;

Substitui a pela variável C_a em $A \rightarrow X_1 X_{2...} X_n \in P_2$;

$P_2 = P_2 \cup \{C_a \rightarrow a\}$;

3. garantir que as produções em que o lado direito de comprimento maior do que um, seja composto somente por duas variáveis, utilizando-se de variáveis intermediárias. A gramática resultante desta etapa é: $G_3 = (V_3, T_1, P_3, S)$, onde V_3 e P_3 são construídos como mostra o algoritmo abaixo (MENEZES, 2000, p. 99):

$V_3 = V_2$;

$P_3 = P_2$;

Para toda $A \rightarrow B_1 B_2 ... B_n \in P_3$ tal que $n \geq 3$

Faça $P_3 = P_3 - \{ A \rightarrow B_1 B_2 ... B_n \}$;

$V_3 = V_3 \cup \{ D_1 D_2 ... D_n \}$;

$P_3 = P_3 \cup \{ A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2, ..., D_{n-3} \rightarrow B_{n-2} D_{n-2}, D_{n-2} \rightarrow B_{n-1} B_n \}$;

Exemplo:

EXEMPLO 7: Considerando a gramática G que gera expressões aritméticas;

$G_1 = (\{E\}, \{+, *, [,], x\}, P, E)$, onde:

$P_1 = \{E \rightarrow E+E \mid E*E \mid [E] \mid x\}$

- transformar o lado direito das produções de comprimento ≥ 2:

$E \rightarrow E\ X+E \mid EC*E \mid C[EC]$

$C+ \rightarrow +$

$C* \rightarrow *$

$C[\rightarrow [$

$C] \rightarrow]$

- apenas duas variáveis do lado direito:

$E \rightarrow E\ X+E \mid EX*E \mid X[EC]$

$E \rightarrow EY1 \mid EY2 \mid X[Y3$

$Y1 \rightarrow X+E$

$Y2 \rightarrow X*E$

$Y3 \rightarrow EX]$

Obtêm-se assim, a gramática na Forma Normal de Chomsky:

G2 = ({E, X+, X*, X[, X], Y1, Y2, Y3}, {+, *, [,], x}, P_2, E), onde:

P_2 = {E $\rightarrow$ EY1 | EY2 | X[Y3 | x,

Y1 $\rightarrow$ X+E, Y2 $\rightarrow$ X*E, Y3 $\rightarrow$ EX],

C+ $\rightarrow$ +, C* $\rightarrow$ *, C[$\rightarrow$ [, C] $\rightarrow$] }

3.2.2.2 ALGORITMO DE COCKE-YOUNGER-KASAMI

O Algoritmo de Cocke-Younger-Kasami é construído sobre uma gramática na Forma Normal de Chomsky e utilizado para análise de cadeias em GLC.

Baseia-se na construção de uma tabela triangular de derivação, sobre a cadeia a ser analisada.

Inicia-se com as variáveis que geram diretamente terminais, como A→ a. Em seguida, as produções que geram duas variáveis, como A→ BC.

Quando não for encontrada nenhuma produção a célula fica vazia e se o símbolo inicial da gramática estiver no vértice da tabela, então a cadeia é aceita.

Exemplos:

EXEMPLO 8: Considerando a Gramática G_1 = ({S, A}, {a, b}, P, S), onde:

P = { S → AA | AS | b, A → SA| AS | a }

A tabela triangular de derivação para palavra de entrada abaab é ilustrada na Figura 28:

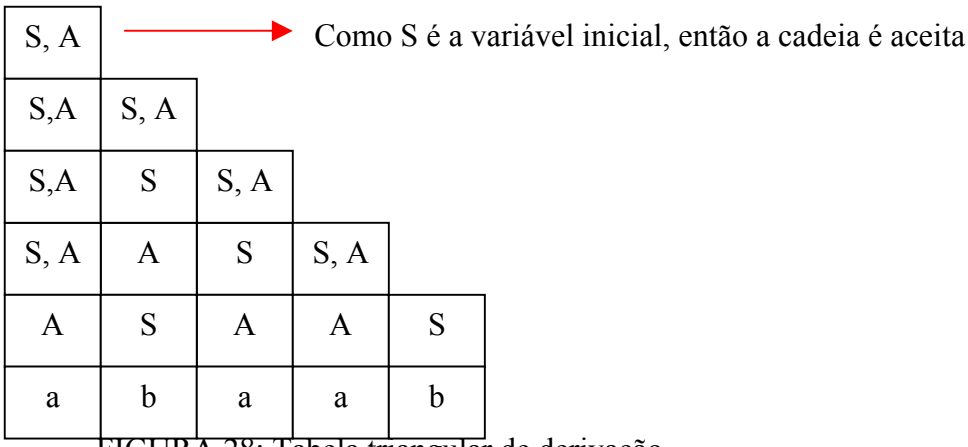


FIGURA 28: Tabela triangular de derivação

EXEMPLO 9: Considerando a G = ({S, R }, {a, b}, P, S}, onde:

$$P = \{S \rightarrow RN \mid RR,\ R \rightarrow a \mid b \mid NS,\ N \rightarrow c \mid NR\ \}$$

A tabela triangular de derivação para palavra de entrada aabaa é ilustrada na Figura 29:

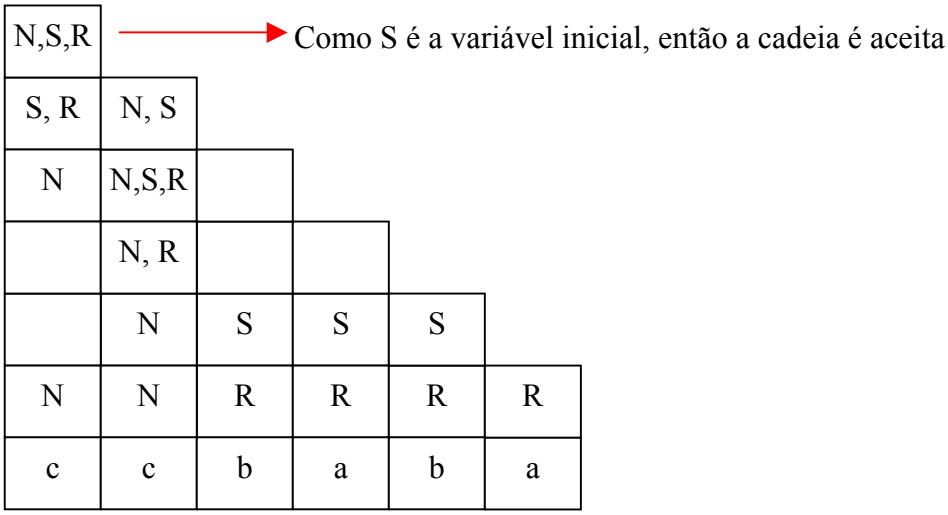


FIGURA 29: Tabela triangular de derivação

3.2.2.3 IMPLEMENTAÇÃO DA GLC

A Ferramenta do GLC possibilita a construção de Gramáticas Livres de Contexto, transformando-as na Forma Normal de Chomsky para a análise de cadeias através do algoritmo de Cocke-Younger-Kasami.

Depois de definida uma GLC, cadeias podem ser submetidas a testes sintáticos para Linguagens Livres de Contexto. Para isto, a gramática é transformada para Forma Normal de Chomsky e à ela aplicada o algoritmo de Cocke-Younger-Kasami. Em seguida é feita a análise da cadeia inserida.

O algoritmo de Cocke-Younger-Kasami (MENEZES, 2000, p.123), pode ser verificado abaixo:

* Variáveis quer geram diretamente ($A \rightarrow a$);

para r variando de 1 até n
faça $V_{r1} = \{A \mid A \rightarrow a_r \in P\}$

* Produções que gera duas variáveis ($A \rightarrow BC$);

para s variando de 2 até n
faça para r variando de 1 até n - s + 1
faça $V_{rs} = \varnothing$
para k variando de 1 até s-1
faça $V_{rs} = V_{rs} \cup \{A \mid A \rightarrow BC \in P, B \in V_{rk} \text{ e } C \in V_{(r+k)\,(s-k)}\}$

Utilizando-se do Exemplo 1, pode-se visualizar a Ferramenta GLC através da Figura 30, durante a análise da cadeia x + x:

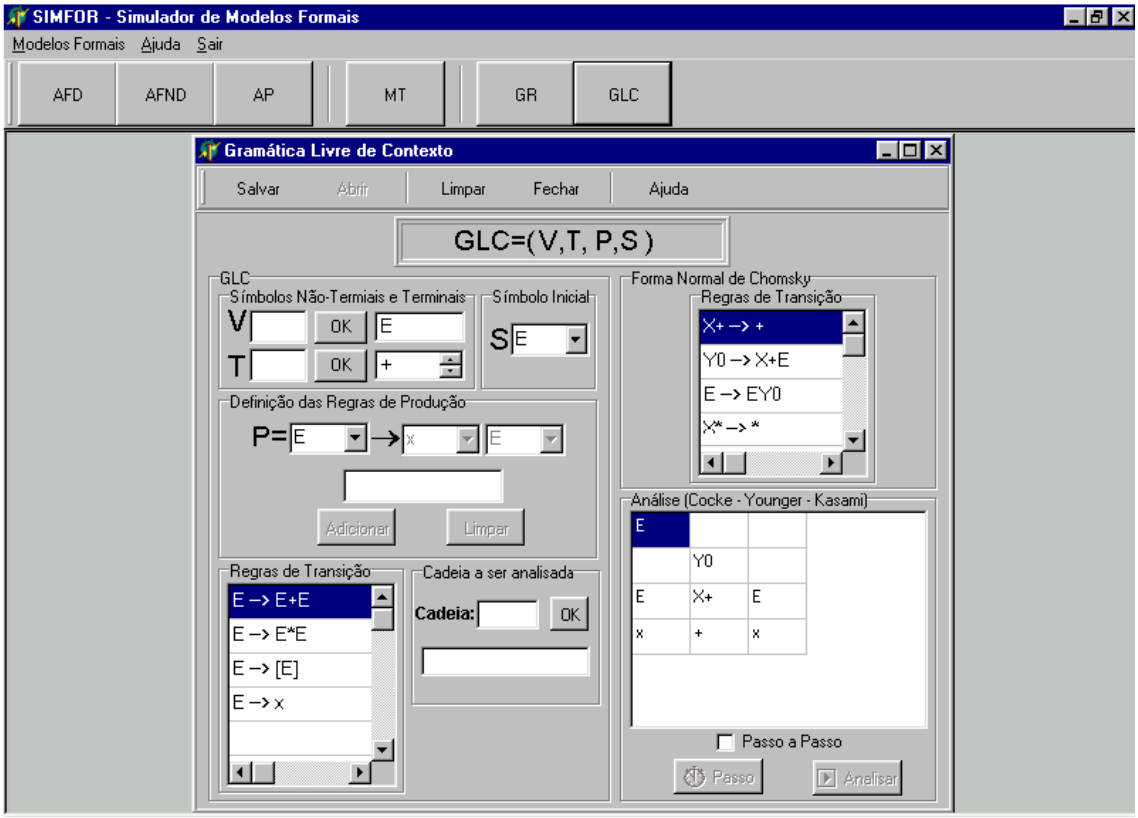


FIGURA 30: Tela do GLC

3.3 MODELOS FORMAIS PARA DEFINIÇÃO DE LINGUAGENS ENUMERÁVEIS RECURSIVAMENTE E LINGUAGENS SENSÍVEIS AO CONTEXTO

3.3.1 MÁQUINA DE TURING

A Máquina de Turing é um modelo matemático que consegue representar qualquer Algoritmo Computacional.

Segundo DIVERIO e MENEZES (2000, p.83), "a Máquina de Turing (...) é universalmente conhecida e aceita como formalização de algoritmo. Trata-se de um mecanismo simples que formaliza a idéia de uma pessoa que realiza cálculos".

Uma Máquina de Turing é uma 8-upla, $MT=(\Sigma, S, \delta, S_0, F, V, \beta, ¤)$, onde:

Σ alfabeto de símbolos de entrada;

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

δ função de transição (parcial):

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

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 e D. A função de transição pode ser representada, por um Diagrama de estados, como mostra a Figura 31:

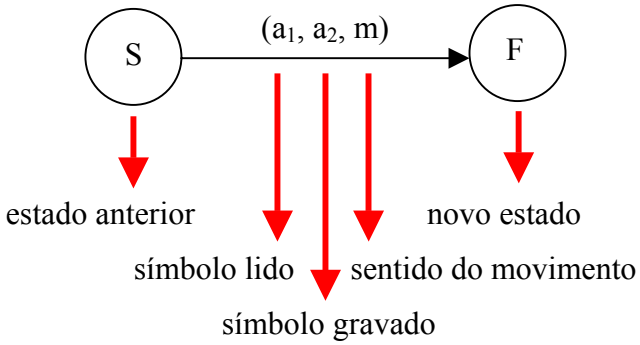


FIGURA 31 - Representação do diagrama de transição de estados da MT

Exemplo:

EXEMPLO 1: Para representar a linguagem L (M) = $\{a^n\ b^n \mid n \geq 0\}$, constrói-se a MT M = < ({a, b}, {S_0, S_1, S_2, S_3, S_4}, δ, S_0, {S_4}, {A, B}, β, ¤) > com δ representada na Figura 32.

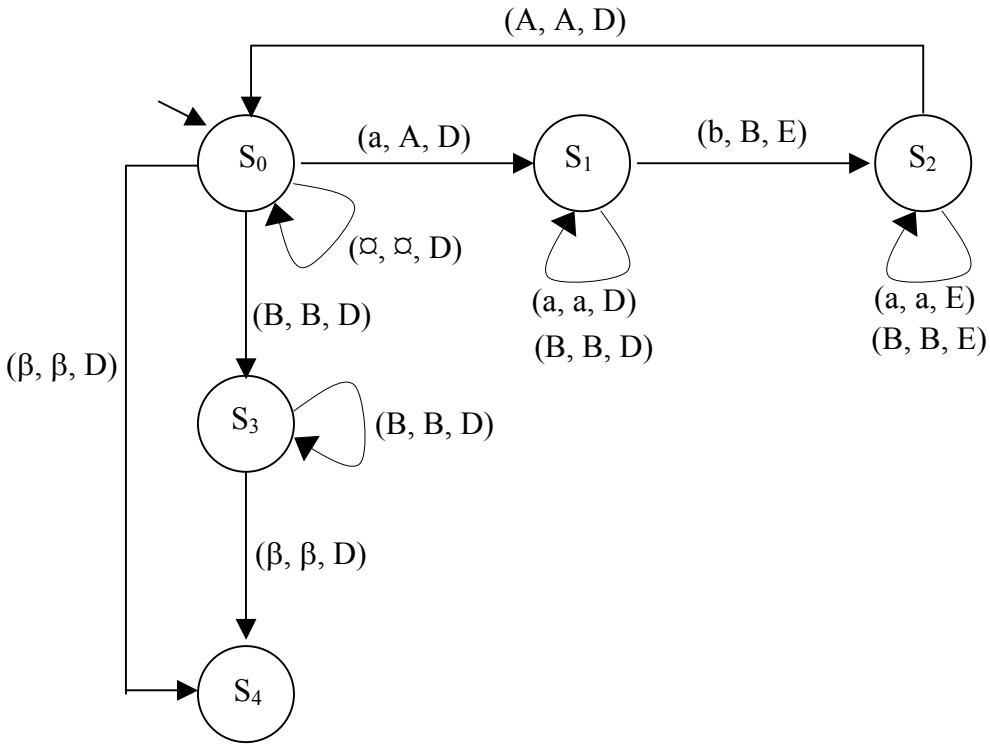


FIGURA 32: Maquina de Turing para L (M) = $\{a^n\ b^n \mid n \geq 0\}$

Tabela de transição:

δ	¤	a	b	A	B	β
S_0	(S_0, ¤, D)	(S_1, A, D)			(S_3, B, D)	(S_4, β, D)
S_1		(S_1, a, D)	(S_2, B, E)		(S_1, B, D)	
S_2		(S_2, a, E)		(S_0, A, D)	(S_2, B, E)	
S_3					(S_3, B, D)	(S_4, β, D)
S_4						

A análise da cadeia aabb é feita como mostra a Figura 33:

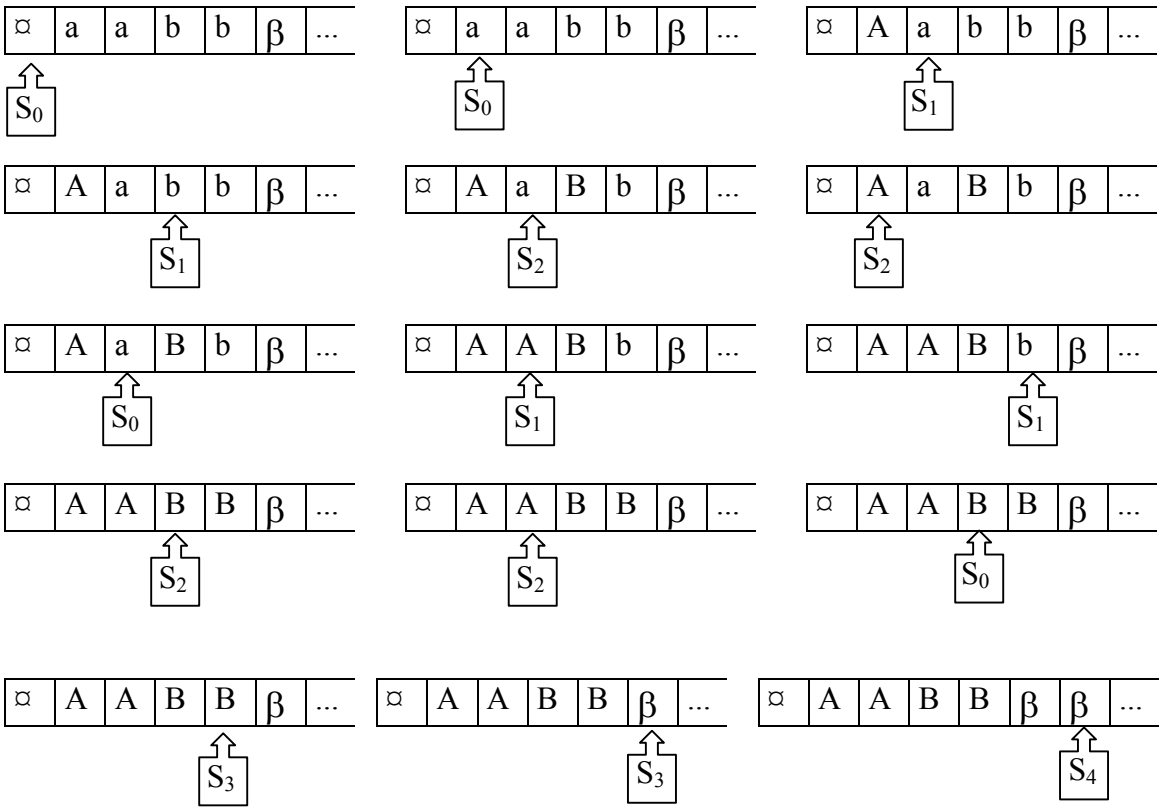


FIGURA 33 – Seqüência do processamento de uma MT

EXEMPLO 2: Para representar a linguagem L = {w | w tem o mesmo número de símbolos a e b}, constrói-se a MT M = < ({a, b}, {S_0, S_1, S_2, S_3, S_4}, δ, S_0, {S_4}, {A, B}, β, ¤) > com δ representada na Figura 34 e na tabela de transição abaixo.

Tabela de Transição:

δ	**¤**	**a**	**b**	**A**	**B**	**β**
S_0	(S_0, ¤,D)	(S_1,A,D)	(S_3,B,D)	(S_0,A,D)	(S_0,B,D)	(S_4,β,D)
S_1		(S_1,a,D)	(S_2,B,E)	(S_1,A,D)	(S_1,B,D)	
S_2	(S_0, ¤,D)	(S_2,a,E)	(S_2,b,E)	(S_2,A,E)	(S_2,B,E)	
S_3		(S_2,A,E)	(S_3,b,D)	(S_3,A,D)	(S_3,B,D)	
S_4						

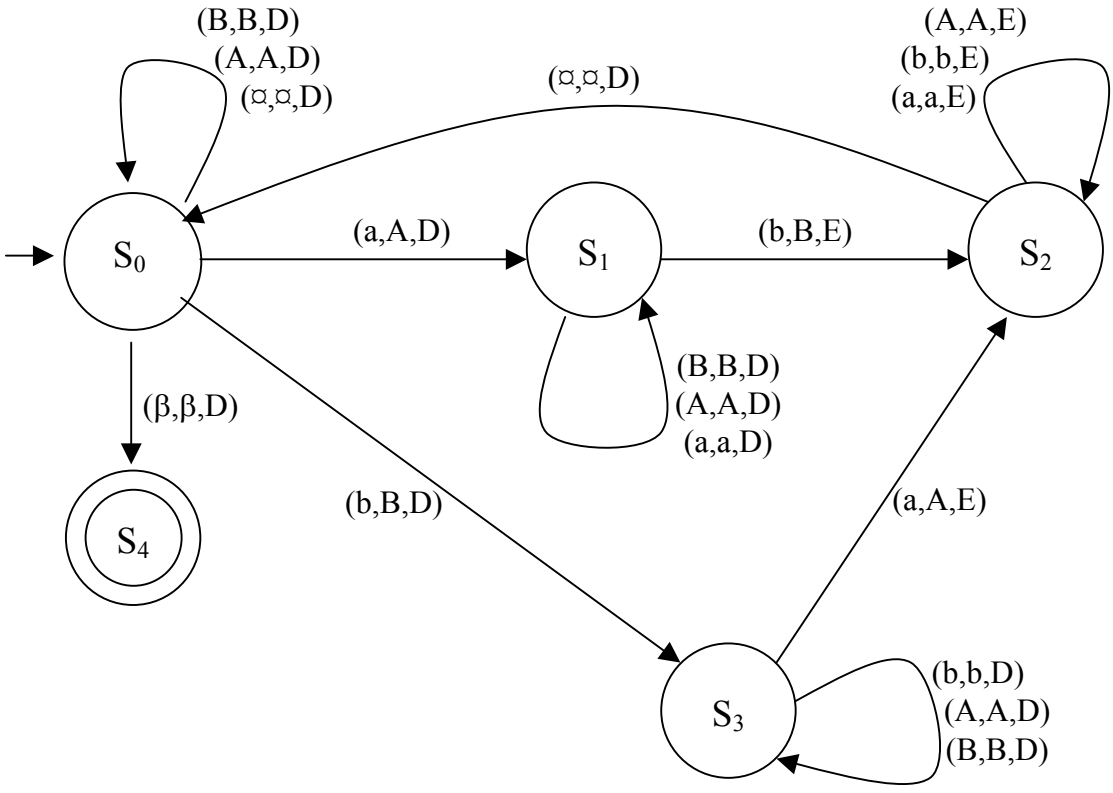


FIGURA 34 – Diagrama de Estados para Máquina de Turing

3.2.3.1 IMPLEMENTAÇÃO DA MÁQUINA DE TURING

A Ferramenta da MT possui uma fita, que é finita à esquerda e infinita à direita , sendo dividida em células, cada uma armazenando um símbolo. Os símbolos podem pertencer ao alfabeto, ao alfabeto auxiliar ou ainda ser "branco" (β) ou "marcador de início da fita" (¤).

Após a inserção dos símbolos (alfabeto, alfabeto auxiliar e estado) é definido o Estado Inicial e os Estados Finais e são construídas as transições. Em seguida é feita a análise da cadeia inserida.

Inicialmente, a cadeia a ser reconhecida , ocupa as células mais à esquerda após o marcador de início da fita, ficando as demais com o símbolo β.

Através da Figura 35, pode ser verificado a análise da cadeia aabb, utilizando-se do exemplo1.

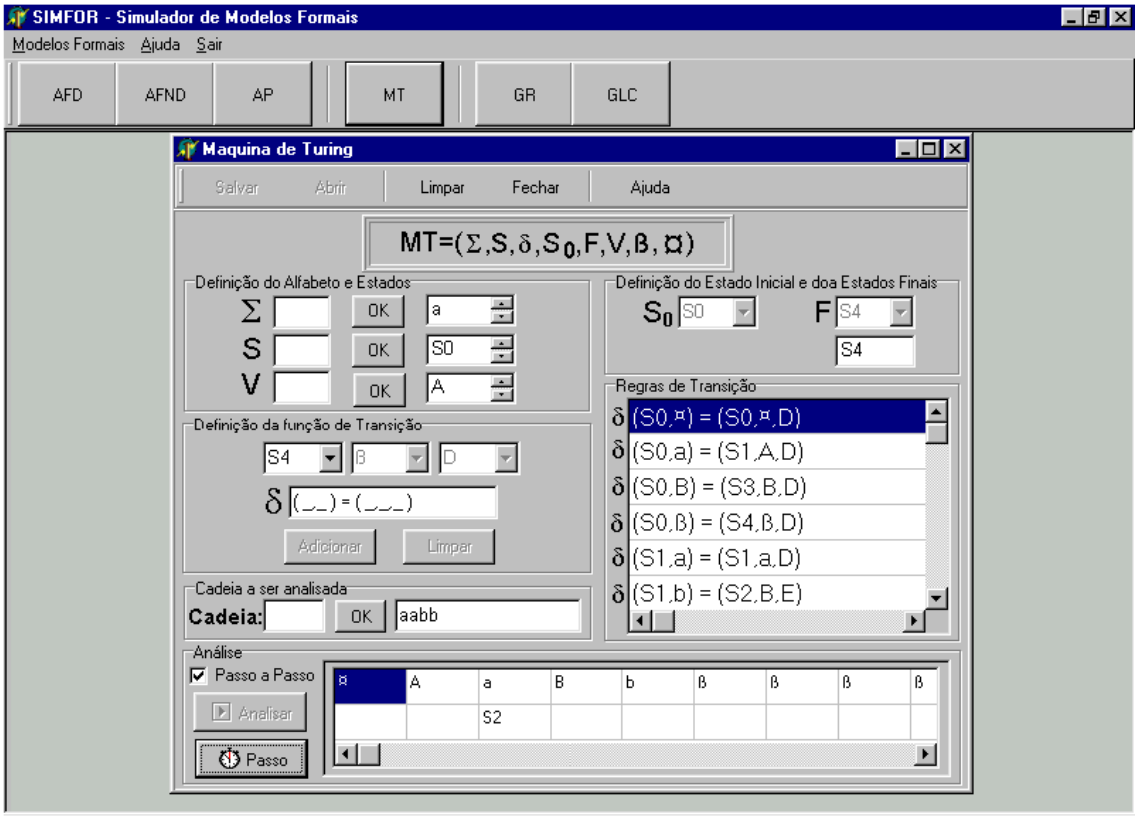


FIGURA 35: Tela da MT

CONCLUSÃO

O estudo da teoria de Linguagens Formais e Autômatos fornece conceitos e princípios que ajudam a compreender a ciência da computação. A construção de modelos abstratos de computadores e de computação, incorporam características importantes que são essenciais a muitas construções complexas, proporcionando um entendimento e associação dos problemas abstratos com os problemas da ciência da computação.

O SIMFOR – Simulador de Modelos Formais – integra vários modelos formais, proporcionando um aprendizado mais dinâmico dos conceitos de geração e de reconhecimento de linguagens, observando-se a funcionalidade dos Autômatos (Máquinas) e das Gramáticas.

O SIMFOR caracteriza-se como uma ferramenta que pode trazer grandes benefícios didáticos no ensino de Linguagens Formais e Autômatos, pois a grande facilidade na realização de testes sintáticos (teste de cadeias) poderá auxiliar os alunos (usuários) na compreensão dos conceitos teóricos, pois as restrições na construção de Gramáticas e Autômatos, presentes na maioria das ferramentas pesquisadas, foram eliminadas nesta ferramenta.

BIBLIOGRAFIA

CENTRO UNIVERSITÁRIO DO ESPÍRITO SANTO. **Compiladores**. Disponível em: <http://www.hipernet.ufsc.br/foruns/Convidados/Computacao_-_UNESC/Compiladores2.html> acesso em: 09 junho 2002.

DELAMARO, Marcio E. **Linguagens formais e autômatos**. Disponível em: <http://www.yandre.cjb.net> acesso em: 2 abril 2002.

DIVÉRIO, T. A.; MENEZES, P. B.**Teoria da computação: máquinas universais e computabilidade.** 2 ed. Porto Alegre: Ed. Sagra Luzato, 2000.

KATSUKI, Lucia Kumato. **Teoria da computação**. Disponível em: <http://wwwp.fc.unesp.br/~lucia/Cap1Introducao.doc> acesso em; 10 de junho de 2002.

MENEZES, Paulo Fernando Blauth. **Linguagens formais e autômatos**. 3. ed. Porto Alegre: Ed. Sagra Luzzato, 2000.

PICOLI, Michelle. **Desenvolvimento de um reconhecedor de linguagens livres de contexto**. Paranavaí, 2000. 76f. Monografia (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Universidade Paranaense – UNIPAR.

RIBEIRO, Carlos H. C. **Uma aplicação da teoria de autômatos**. Disponível em: <http://www.comp.ita.cta.br/~osvandre/atividades/ct200/criptautomata/automatocomsaida/osautomatos.htm> acesso em: 05 de junho de 2002.

DIAS, Silvano B. **Um gerador de analisadores léxicos para linguagens gráficas**. Santa Maria, 1995. 30f. Monografia (Graduação em Informática) – Curso de Informática, Universidade Federal de Santa Maria.

SIMÕES, Marco A. C. **Notas de aula sobre LFA**. Disponível em: <http://www.unit.br/marcoss/Disciplinas/SB1/Apostila_Ling_Formais/Apostila%20Linguagens%20Formais.htm#T2> acesso em: 01 junho 2002.

UNIVERSIDADE FEDERAL DO PARANÁ. Sistema de Bibliotecas. **Normas para Apresentação de Documentos Científicos**. Curitiba: Ed. da UFPR, 2000.

ZANIN, Fabio. **Linguagens Formais**. Universidade Regional Integrada do Alto Uruguai e das Missões – URI, Campus Erechim. Disponível em: <http://www.lia.ufc.br/adriano/Linguagens%20Formais.pdf> acesso em: 09 junho 2002.