Universidade Federal do Pará
Departamento de Informática
Curso de Bacharelado em Ciência da Computação

Linguagens Formais

Profª Carla Alessandra Lima Reis

Belém, Junho de 2002.

Linguagens Formais

1 CONCEITOS BÁSICOS 3

1.1 Alfabeto e Linguagens 3
1.2 Gramáticas 4
1.3 Tipos de Gramáticas 7

2 EXPRESSÕES REGULARES (ER) 10

3 AUTÔMATO FINITO (AF) 11

3.1 Autômato Finito Determinístico (AFD) 11
3.2 Autômato Finito Não-Determinístico (AFND) 14
3.3 Autômato Finito Não-Determinístico com movimentos ε (AFND-ε) 16
3.4 Relação entre Autômatos Finitos e Gramáticas Regulares 19
3.5 Equivalência entre Autômatos Finitos e Expressões Regulares 20
3.6 Minização de AFD completo 24

4 GRAMÁTICAS LIVRES DE CONTEXTO (GLC) 27

4.1 Extensão da definição de GLC 27
4.2 Árvores de Derivação para GLC 27
4.3 Derivação mais à esquerda e mais à direita 29
4.4 Gramática Ambígua 30
4.5 Transformações em G.L.C 30
4.5.1 Eliminação de Símbolos Inúteis 31
4.5.2 Transformação de uma GLC qualquer para uma GLC ε-Livre 32
4.5.3 Remoção de Produções Simples (unitárias) 34
4.5.4 Fatoração de GLC 35
4.5.5 Eliminação de Recursão à Esquerda (e a Direita) 36
4.6 Forma Normal de Chomsky (CNF) 38
4.7 Forma Normal de Greibach (GNF) 39
4.8 Outros Tipos de GLC 41

5 AUTÔMATO DE PILHA 42

5.1 PDA e Linguagens Livres de Contexto 47

6 MÁQUINA DE TURING 50

6.1 Modelo Básico da Máquina de Turing 50
6.2 Definição de Máquina de Turing 51
6.3 Relação entre Máquina de Turing e Gramática Irrestrita 53

BIBLIOGRAFIA 54

1 Conceitos Básicos

Uma linguagem é um meio de comunicação, formada por um conjunto de palavras e de regras gramaticais que permitem combinar as palavras em sentenças sintaticamente corretas.

Uma linguagem é dita formal quando pode ser representada através de um sistema com sustentação matemática.

A Linguística Formal compreende a representação da sintaxe (estrutura) e da semântica (significado) das sentenças de uma linguagem. Nesta disciplina será abordada somente a estrutura sintática das linguagens.

1.1 Alfabeto e Linguagens

Alfabeto, ou vocabulário, é um conjunto finito de símbolos. Exemplo: dígitos, letras, letras gregas, etc.

Sentença, ou palavra, definida sobre um alfabeto é qualquer sequência finita de símbolos do alfabeto. Exemplo:

alfabeto V = { 0,1}
sentenças válidas = 001 010011

O tamanho, ou comprimento, de uma sentença é dado pelo número de símbolos que compõem a sentença. Exemplo:

alfabeto V = { a, b, c }
sentença w = ab
tamanho |w| = 2

Sentença vazia é a sentença que não contém símbolos, possuindo tamanho 0. É representada por **ε**.

Seja V um alfabeto, representamos por:

V^* = conjunto de todas as sentenças compostas de símbolos de V incluindo a sentença vazia.

$V^+ = V^* - (\varepsilon)$.

Exemplos:

V = { 0,1}
V^* = { ε, 0, 1, 00, 01, 10, 11, ...}
V^+ = { 0, 1, 00, 01, 10, 11, ...}

Linguagem é qualquer conjunto de sentenças sobre um alfabeto, ou seja uma linguagem L sobre um alfabeto V é um subconjunto de V*. Conforme o número de sentenças que possuem, as linguagens se classificam em:

- finitas
- vazias
- infinitas

Como representar uma linguagem ?

1. listando as palavras (só para linguagem finita)
2. através de uma descrição algébrica
   Exemplo: (a^n b^n c^n | n >= 1)
3. definindo um algoritmo que determina se uma sentença pertence ou não à linguagem: Reconhecimento
   Exemplo: **autômato finito**
4. definindo um mecanismo que gera sistematicamente as sentenças da linguagem em alguma ordem: Geração
   Exemplo: **gramática**

1.2 Gramáticas

Uma gramática serve para definir qual o subconjunto de sentenças que faz parte de uma determinada linguagem.

É um dispositivo formal para especificar uma linguagem potencialmente infinita de uma forma finita.

Existem diversas formas de representar a sintaxe das linguagens: BNF, diagramas de sintaxe, notação formal de gramática.

Exemplo: **Backus Naur Form** (BNF)

<programa> ::= <decl>; <bloco>.
<bloco> ::= BEGIN <comando> END
<comando> ::= <if> <atrib> <goto> ...

Elementos:

não-terminais ou variáveis = <programa> <bloco>
símbolos terminais = BEGIN ; END
produções = <comando> ::= <if>
símbolo inicial = <programa>

Notação Formal de Gramática

Uma gramática G é definida por uma quádrupla

$$G = (N, T, P, S) \quad \text{onde}$$

N - conjunto finito de não-terminais
T - conjunto finito de terminais
P - conjunto finito de regras de produção
S - símbolo inicial da gramática

Observações:

$N \cap T = \varnothing$
$V = N \cup T$
$S \in N$
$P = \{ \alpha \rightarrow \beta \mid \alpha \in V^+ \text{ e } \beta \in V^* \}$

Exemplo:

$G = (N, T, P, I)$
$N = \{ I, D \}$
$T = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$
$P = (I \rightarrow D , D \rightarrow 0 , D \rightarrow 1 , D \rightarrow 2 , D \rightarrow 3 , D \rightarrow 4 , D \rightarrow 5 ,$
$D \rightarrow 6 , D \rightarrow 7 , D \rightarrow 8 , D \rightarrow 9)$

Convenções

Para facilitar a compreensão das expressões, são adotadas sempre que possível as seguintes convenções:

$N = \{ A, B, C, \ldots T \}$	Letras maiúsculas do início do alfabeto para símbolos não-terminais
$T = \{ a, b, c, \ldots t \}$	Letras minúsculas do início do alfabeto para símbolos terminais
$T^* = \{ u, v, \ldots z \}$	Letras minúsculas do fim do alfabeto para sequências de terminais
$(N \cup T)^* = \{ \alpha, \beta, \gamma, \ldots \}$	Letras gragas para sequências de variáveis e terminais
$N \cup T = \{ U, V, \ldots Z \}$	Letras maiúsculas do fim do alfabeto para um símbolo terminal ou não-terminal

Derivação

Derivação é uma operação de substituição efetuada de acordo com as regras de produção da gramática. Representa-se esta operação pelo símbolo $\Rightarrow$

Logo: $\gamma\ \alpha\ \delta \Rightarrow \gamma\ \beta\ \delta$
se $\alpha \rightarrow \beta \in P$ e
γ e $\delta \in V^*$

Uma sequência de derivações, de zero ou mais passos, é representada por $\overset{*}{=>}$. Isto é:

$\alpha 1 \overset{*}{=>} \alpha_m$ se
$\alpha 1 \Rightarrow \alpha 2$, $\alpha 2 \Rightarrow \alpha 3$, ..., $\alpha_{m-1} \Rightarrow \alpha_m$,
$\alpha 1, \alpha 2, ..., \alpha_m \in V^*$

Para garantir que pelo menos uma derivação seja efetuada, a notação utilizada é $\overset{+}{=>}$. Isto é:

$\alpha 1 \overset{+}{=>} \alpha_m$ se
$\alpha 1 \Rightarrow \alpha 2$, $\alpha 2 \overset{*}{=>} \alpha_m$,
$\alpha 1, \alpha 2, ..., \alpha_m \in V^*$

Sentença e Forma Sentencial

Uma sentença de uma linguagem é uma sequência formada de terminais, obtida a partir do símbolo inicial da gramática desta linguagem, através de derivações sucessivas.

$S \overset{+}{=>} u$ {u= sequência de terminais}

Uma forma sentencial é uma sequência qualquer, composta de símbolos terminais e de não-terminais, obtida a partir do símbolo inicial da gramática através de derivações sucessivas.

$S \overset{*}{=>} \alpha$ {α=sequência de variáveis e terminais}

Linguagem

A linguagem gerada por uma gramática G = (N, T, P, S) é o conjunto de todas as sentenças que podem ser geradas a partir do símbolo inicial desta gramática, através de derivações sucessivas.

$L(G) = \{ w \mid w \in T^* \text{ e } S \overset{+}{=>} w \}$

Exemplo:

G = (N, T, P, S)
G = ({S}, {0,1}, P, S)
P = { $S \rightarrow 0S1$

$S \rightarrow 01$ }

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0^3S1^3 \Rightarrow .... 0^{n-1}S1^{n-1} \Rightarrow 0^n1^n$

$L(G) = \{ 0^n 1^n \mid n >= 1 \}$

1.3 Tipos de Gramáticas

Segundo a hierarquia de Chomsky, as gramáticas podem ser de quatro tipos. Para G = (N, T, P, S) e V = N ∪ T, temos:

Tipo 0 - **Gramáticas Irrestritas (GI)**

São definidas pelas seguintes regras de produção:

$$P = \{ \alpha \rightarrow \beta \mid \alpha \in V^+ , \beta \in V^* \}$$

Ou seja, do lado esquerdo da produção pode haver uma sequência de quaisquer símbolos, desde que, entre eles, haja um não-terminal. Do lado direito da produção pode haver qualquer sequência de símbolos, inclusive a sentença vazia.

Tipo 1 - **Gramáticas Sensíveis ao Contexto (GSC)**

Para toda produção

$\alpha \rightarrow \beta \in P$

$|\alpha| <= |\beta|$

Ou seja, o comprimento da sentença do lado esquerdo deve ser menor ou igual ao comprimento da sentença do lado direito da produção. Do lado direito não é aceito a sentença vazia.

Ex: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 B \alpha_2$

Tipo 2 - **Gramáticas Livres de Contexto (GLC)**

Quando as regras de produção são todas na seguinte forma:

$$P = \{ \alpha \rightarrow \beta \mid \alpha \in N \text{ e } \beta \neq \varepsilon \}$$

Ou seja, do lado esquerdo da produção deve, sempre, ocorrer um e apenas um não-terminal. A sentença vazia também não é aceita do lado direito da produção.

Ex: $X \rightarrow abcX$ {não interessa o contexto em que X se encontra}

Tipo 3 - **Gramáticas Regulares (GR)**

Toda produção é da forma:

$A \rightarrow aB$ ou

$A \rightarrow a$
Ou seja:
$P = \{ A \rightarrow aX \mid A \in N, a \in T, X \in N \cup \{\varepsilon\} \}$

Ou seja, do lado esquerdo da produção deve, sempre, ocorrer um e apenas um não-terminal e do lado direito podem ocorrer ou somente um terminal, ou um terminal seguido de um não-terminal.

Observação:

Exemplo de Gramática Livre de Contexto:

$G = (N, T, P, S)$
$N = \{ S, A, B \}$
$T = \{ a, b \}$
$P = \{$ $S \rightarrow aB$ $\quad A \rightarrow bAA$
$S \rightarrow bA$ $\quad B \rightarrow b$
$A \rightarrow a$ $\quad B \rightarrow bS$
$A \rightarrow aS$ $\quad B \rightarrow aBB \}$

L(G) é o conjunto de todas as palavras em T que tem o mesmo número de a's e b's.

Linguagens geradas por gramáticas

Conforme o tipo da gramática que dá origem a uma linguagem, estas se classificam em:

LSC - Linguagem Sensível ao Contexto
LLC - Linguagem Livre de Contexto
LR - Linguagem Regular

A sentença vazia

Vamos estender as definições das gramáticas Tipo 1, 2 e 3 para permitir produções do tipo $S \rightarrow \varepsilon$, sendo S o símbolo inicial. Entretanto, isto só será possível se S não aparecer do lado direito de qualquer produção.

Desde modo, a regra $S \rightarrow \varepsilon$ somente será utilizada para dar origem à sentença vazia.

A gramática que possuir o símbolo inicial aparecendo do lado direito de regras de produção, deverá ser transformada em outra equivalente que obedeça a esta restrição.

Observação: gramáticas equivalentes devem gerar a mesma linguagem.

Quando for necessário transformar a gramática, deverá ser incluído um novo símbolo não-terminal (S') que passará a ser o novo símbolo inicial. Em seguida, deverão ser incluídas em P todas as regras que tinham o símbolo S do lado esquerdo, substituindo este por S', mais a regra $S' \rightarrow \varepsilon$.

Exemplo: Modificar a gramática abaixo de modo a incluir a sentença vazia.

$G = \{ \{ S, A \}, \{a\}, P, S \}$
P: $S \rightarrow aA$
$A \rightarrow aA \mid a$

Resposta:

$G = \{ \{ S', S, A \}, \{a\}, P', S' \}$
P': $S' \rightarrow S \mid \varepsilon$
$S \rightarrow aA$
$A \rightarrow aA \mid a$

2 Expressões Regulares (ER)

Toda linguagem regular pode ser descrita por uma expressão simples, denominada **Expressão Regular** (ER). Trata-se de um formalismo gerador, pois expressa como construir (gerar) as palavras da linguagem. Uma ER é definida recursivamente a partir de conjuntos (linguagens) básicas e operação de concatenação e união.

Linguagens Regulares	gramáticas	geradores de linguagens
	expressões regulares	
	autômatos finitos	reconhecedores de linguagens

Regras de formação de expressões regulares

Dado um alfabeto

1) os símbolos do alfabeto são expressões regulares;
2) se R1 e R2 são ER, então (R1 ∪ R2) é uma ER;
3) se R1 e R2 são ER, então (R1 R2) é uma ER;
4) se R1 é uma ER, então $(R1)^*$ é uma ER;

(R1 R2) representa concatenação de linguagens;
$(R1)^*$ representa a linguagem formada pela concatenação de zero ou mais palavras de R1.
(R1 | R2) representa união de linguagens

Observação:

$p^+ = pp^*$

Exemplos:

Expressão Regular	**Linguagem Gerada**
aa	contém somente a palavra **aa**
ba*	todas as palavras que iniciam por b, seguido de zero ou mais a´s
(a) *	{ ε, a, aa, aaa, ... }
(a | b)*	{ ε, a, b, aa, ab, bb, ba, aaa, ... }
(a (a | b))*	{ ε, aa, ab, aaaa, abaa, aaba, ... }
(a (a | b)*)	{ a, aa, ab, aaa, aba, aab, ... }
Identificadores l (l | d) *	{ l, ll, ld, lll, ... }
Inteiro com sinal (+ | -) d (d)*	

3 Autômato Finito (AF)

Um autômato finito é um modelo matemático (máquina abstrata) de um sistema, com entradas e saídas discretas. O sistema pode estar em qualquer uma de um número finito de configurações internas (ou *estados*). O estado de um sistema resume as informações anteriores até o momento atual do reconhecimento necessárias para determinar o seu comportamento face ao resto da sequência que está analisando.

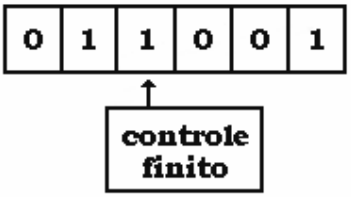
Um bom exemplo de um autômato finito é o mecanismo de controle de um elevador. Ele não lembra todas as requisições anteriores mas somente o andar atual, o direção de movimento (para cima ou para baixo), e a relação de requisições pendentes.

3.1 Autômato Finito Determinístico (AFD)

Um AFD é definido formalmente por uma 5-tupla $(K, \Sigma, \delta, q_0, F)$, onde:

K	é um conjunto finito, não vazio, de estados
Σ	é um alfabeto finito de entrada
q_0	é o estado inicial e $q_0 \in K$
F	é o conjunto de estados finais e $F \subseteq K$
δ	é a função de transição de estados, mapeando $K \times \Sigma$ em K. Isto é, $\delta(q, a)$ é um estado para cada estado q e símbolo de entrada a.

Um autômato finito é representado através de um controle finito, que tem acesso a uma fita onde está a sequência a ser analisada. O autômato percorre esta fita da esquerda para a direita, lendo um símbolo de cada vez. Estando o controle em um estado, apontando para um determinado símbolo na entrada, ocorre uma transição de estado, que faz com que o controle passe para outro estado e avance para o próximo símbolo na entrada. Se isto se repetir com sucesso até o final da fita de entrada e, no final, o autômato estiver em um estado final, a sequência foi reconhecida. Se, ao contrário, ocorrer alguma falha (transição impossível) o reconhecimento para e a sequência não é reconhecida.

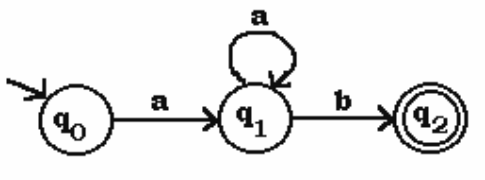


δ (q, a) = p para q e p $\in$ K e a $\in \Sigma$, significa que, estando M no estado q e lendo o símbolo de entrada a, o cabeçote de leitura é movido uma célula para a direita e M vai para o estado p.

Diagrama de Transição

Uma outra maneira de representar um autômato finito são os diagramas de transição. Trata-se de um grafo direcionado e rotulado. Os vértices representam os estados, sendo representados por círculos. Os estados finais são representados por 2 círculos concêntricos e o estado inicial é indicado por uma seta. As arestas representam as transições entre 2 estados, sendo o rótulo o símbolo reconhecido.

Exemplo:



$\delta(q_0, a) = q_1$
$\delta(q_1, a) = q_1$
$\delta(q_1, b) = q_2$

$M = ((q_0, q_1, q_2), \{ a, b \}, \delta, q_0, \{q_2\})$

$T (M) = \{ a^n b \mid n >= 1 \}$

Tabela de Transição de Estados

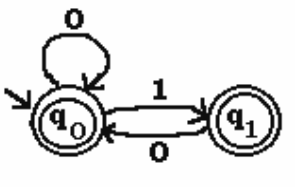
Uma terceira maneira de representar um autômato finito é através de uma tabela, indicando na vertical os estados e na horizontal os símbolos do alfabeto. O estado inicial é indicado com uma seta e os finais com asteriscos. O conteúdo da tabela indica as transições de estado possíveis.

Exemplo:

		a	b
$\rightarrow$	q_0	q_1	-
	q_1	q_1	
q_2			
*	q_2	-	-

Exemplo:

Autômato finito que reconhece sentenças em { 0, 1 }, as quais não contém dois ou mais 1's consecutivos.



$M = (K, \Sigma, \delta, q_0, F)$
$K = (q_0, q_1)$
$\Sigma = (0, 1)$
$F = (q_0, q_1)$

$\delta (q_0, 0) = q_0$
$\delta (q_0, 1) = q_1$
$\delta (q_1, 0) = q_0$

	0	1
$*\rightarrow q_0$	q_0	q_1
$* \quad q_1$	q_0	-

Extensão da função δ para o domínio K x Σ*

$\xi (q, \varepsilon) = q$
$\xi (q, xa) = \delta (\xi (q, x), a)$
para $x \in \Sigma^*$ e $a \in \Sigma$

$\xi (q, x) = p$ significa que M, iniciando no estado q, com a sequência x na fita de entrada, entrará no estado p, após o cabeçote mover-se para a direita tantas células quanto for o comprimento de x.

Uma sentença x é aceita por M se $\delta (q_0, x) = p$ para algum p em F.

Linguagem aceita por M

O conjunto de todos os x aceitos por M é:

$T(M) = \{ x \mid \delta (q_0, x) = p \wedge p \in F \}$

Qualquer conjunto de sentenças aceito por um autômato finito é dito ser REGULAR.

3.2 Autômato Finito Não-Determinístico (AFND)

Um AFND é um sistema (K, ∑, δ, q_0, F), onde:

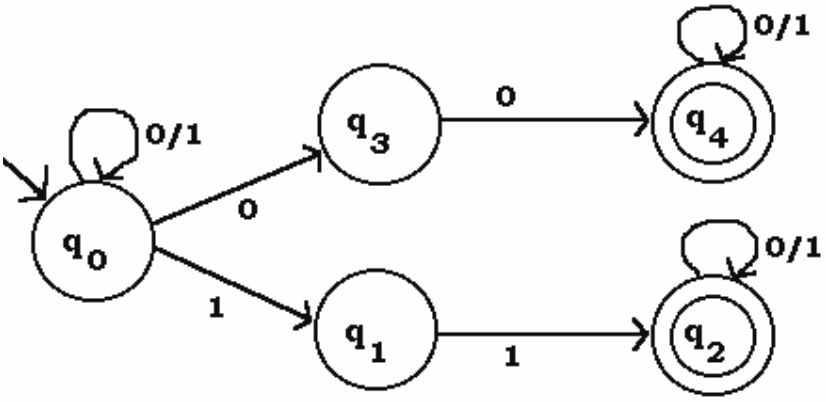
K é um conjunto finito, não vazio, de estados
∑ é um alfabeto finito de entrada
q_0 é o estado inicial e $q_0 \in K$
F é o conjunto de estados finais e $F \in K$
δ é a função de transição de estados, mapeando K x ∑ em 2^K. Isto é, δ(q, a) é um subconjunto de K para cada estado *q* e símbolo de entrada *a*.

Observação: 2^K é o conjunto de todos os subconjuntos de K

Diferença importante entre determinísticos e não-determinísticos:
δ(q, a) é um conjunto de estados (possivelmente vazio) ao invés de um único estado.

δ(q, a) = { p_1, p_2, ..., p_k }, significa que, M estando no estado q, olhando a na fita de entrada, move uma célula para a direita e escolhe qualquer um de p_1, p_2, ..., p_k como próximo estado.

Exemplo:
AFND que aceita o conjunto de todas as sentenças que contém dois 0's ou dois 1's consecutivos.



M = ({ q_0, q_1, q_2, q_3, q_4 }, { 0, 1 }, δ, q_0, { q_2, q_4 })

$\delta(q_0, 0) = \{ q_0, q_3 \}$	$\delta(q_0, 1) = \{ q_0, q_1 \}$
$\delta(q_1, 0) = \varnothing$	$\delta(q_1, 1) = \{ q_2 \}$
$\delta(q_2, 0) = \{ q_2 \}$	$\delta(q_2, 1) = \{ q_2 \}$
$\delta(q_3, 0) = \{ q_4 \}$	$\delta(q_3, 1) = \varnothing$
$\delta(q_4, 0) = \{ q_4 \}$	$\delta(q_4, 1) = \{ q_4 \}$

Equivalência entre DFA's e NFA's

Teorema: Se L é um conjunto aceito por um autômato finito não-determinístico, então existe um autômato finito determinístico que aceita L.

Seja M = (K, Σ, δ, q_0, F) um AFND que aceita L. Definimos um AFD

M' = (K', Σ', δ', q_0', F') tal que:

- os estados de M' constituem todos os subconjuntos do conjunto de estados de M: K' = 2^K.

- F' é o conjunto de todos os estados de K' contendo um estado de F.

- um elemento de K' é denotado por [q_1 q_2 ... q_i] onde q_1 q_2 ... q_i $\in$ K.

- q_0' = [q_0]

- definimos δ' ([q_1 q_2 ... q_i], a) = [p_1 p_2 ... p_j]
se e somente se δ ({ q_1, q_2, ... q_i }) = { p_1, p_2, ... p_j} =
= δ (q_1, a) $\cup$ δ (q_2, a) $\cup$ ... $\cup$ δ (qi, a)

Exemplo

Dado o AFND M = ({ q_0, q_1 }, { 0, 1 }, δ, q_0, { q_1 }), onde:

δ (q_0, 0) = { q_0, q_1 }
δ (q_0, 1) = { q_1 }
δ (q_1, 0) = $\varnothing$
δ (q_1, 1) = { q_0, q_1 }

Construir um AFD M' que reconheça a mesma linguagem que M.

M' = (K', { 0, 1 }, δ', [q_0], F')
K' = { [q_0], [q_1], [q_0 q_1], $\varnothing$)
F' = { [q_1], [q_0 q_1])

δ (q_0, 0) = { q_0, q_1 }	δ' ([q_0], 0) = [q_0, q_1]
δ (q_0, 1) = { q_1 }	δ' ([q_0], 1) = [q_1]
δ (q_1, 0) = $\varnothing$	δ' ([q_1], 0) = $\varnothing$
δ (q_1, 1) = { q_0, q_1 }	δ' ([q_1], 1) = [q_0, q_1]

δ' ([q_0, q_1], 0) = [q_0, q_1] desde que
δ ({q_0, q_1 }, 0) = δ (q_0, 0) $\cup$ δ (q_1, 0) = { q_0, q_1 } $\cup$ $\varnothing$ = { q_0, q_1 }

e δ' ($[q_0, q_1]$, 1) = [q_0, q_1] desde que
δ ($\{q_0, q_1\}$, 1) = δ (q_0, 1) $\cup$ δ (q_1, 1) = $\{ q_1 \} \cup \{ q_0, q_1 \} = \{ q_0, q_1 \}$

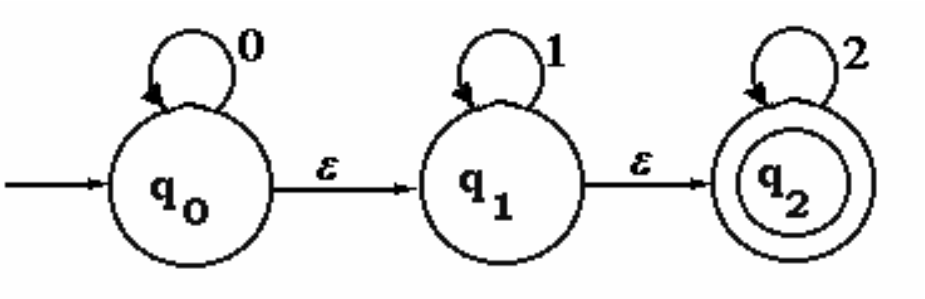
δ ($\varnothing$, 0) = δ' ($\varnothing$, 0) = $\varnothing$

δ	0	1
$\rightarrow q_0$	$[q_0, q_1]$	q_1
$*q_1$	$\varnothing$	$[q_0,q_1]$

δ'	0	1
$\rightarrow[q_0]$	$[q_0,q_1]$	$[q_1]$
$*[q_1]$	$\varnothing$	$[q_0,q_1]$
$*[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_1]$

3.3 Autômato Finito Não-Determinístico com movimentos ε (AFND-ε)

Pode-se estender o modelo do AFND para permitir transições com o ε como símbolo de entrada. Como exemplo tomemos o diagrama de transição abaixo que aceita a linguagem consistindo de qualquer número (incluindo zero) de 0's, seguido de um número qualquer de 1's que são seguidos igualmente de por um número qualquer de 2's.



Formalmente, nós definimos AFND-ε como sendo uma quíntupla (K, Σ, δ, q_0, F), onde:

K é um conjunto finito, não vazio, de estados
Σ é um alfabeto finito de entrada
q_0 é o estado inicial e $q_0 \in K$
F é o conjunto de estados finais e $F \in K$
δ é a função de transição de estados, mapeando K x ($\Sigma \cup \{\varepsilon\}$). A intenção é que δ (q, a) consiste de todos os estados *p* tal que existe uma transição *a* de *q* para *p*, onde *a* é ε ou um símbolo de Σ.

Exemplo:

A tabela de transição para o AFND-ε acima é a seguinte

δ	0	1	2	ε
q_0	$\{q_0\}$	∅	∅	$\{q_1\}$
q_1	∅	$\{q_1\}$	∅	$\{q_2\}$
q_2	∅	∅	$\{q_2\}$	∅

Nós devemos estender a função de transição δ para a função ξ que mapeia K x Σ^* em 2^K. O resultado desta função será todos os estados *p* tal que pode-se ir de *q* para *p* através de um caminho *ω*, talvez incluindo-se arcos ε. Na construção de ξ será importante saber o conjunto de estados atingíveis a partir de um determinado estado *q* usando-se somente ε. Nós utilizamos ε-CLOSURE(q) para assinalar o conjunto de todos os estados *p*, tal que existe um caminho de *q* para *p* nomeado ε.

Exemplo:

No exemplo acima o ε-CLOSURE(q_0) = $\{q_0, q_1, q_2\}$.

Nós podemos considerar que o ε-CLOSURE(P), onde *P* é um conjunto de estados, seja $\cup_{q \text{ em } P}$ ε-CLOSURE(q). Agora, nós definimos ξ como segue:

1) ξ(q, ε) = ε-CLOSURE(q).
2) Para w em Σ^* e a em Σ, ξ(q, wa) = ε-CLOSURE(P), onde P = {p | para algum *r* em ξ(q, w), *p* está em δ(r, a)}

é conveniente estender δ e ξ para conjuntos dos estados de

3) $\delta(R, a) = \cup_{q \text{ em } R} \delta(q, a)$, e
4) $\xi(R, w) = \cup_{q \text{ em } R} \xi(q, w)$

para os conjuntos de estados R. Note que nesse caso, ξ(q, a) não é necessariamente igual a δ(q, a), pois ξ(q, a) inclui todos os estados alcançáveis a partir de *q* por um caminho *a* (incluindo arcos ε), enquanto que δ(q, a) inclui somente os estados alcançáveis a partir de q através de arcos *a*. Igualmente, ξ(q, ε) não é necessariamente igual a δ(q, ε).

Exemplo:

Considere novamente o exemplo dado anteriormente.
$\xi(q_0, \varepsilon)$ = ε-CLOSURE(q_0) = $\{q_0, q_1, q_2\}$

Então

$\xi(q_0, 0)$ = ε-CLOSURE($\delta(\xi(q_0, \varepsilon), 0)$
ε-CLOSURE($\delta(\{q_0, q_1, q_2\}, 0)$
ε-CLOSURE($\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$)
ε-CLOSURE($\{q_0\} \cup \varnothing \cup \varnothing$)
ε-CLOSURE($\{q_0\}$) = $\{q_0, q_1, q_2\}$

e

$\xi(q_0, 1)$ = ε-CLOSURE($\delta(\xi(q_0, \varepsilon), 1)$
ε-CLOSURE($\delta(\{q_0, q_1, q_2\}, 0)$
ε-CLOSURE($\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$)
ε-CLOSURE($\varnothing \cup \{q_1\} \cup \varnothing$)
ε-CLOSURE($\{q_1\}$) = $\{q_1, q_2\}$

Equivalência entre AFND's e AFND-ε's

Teorema: Se L é aceita por um AFND-ε, então L é aceita por um AFND.

Prova: Considere que M = (K, ∑, δ, q_0, F) seja um AFND-ε. Construa M = (K, ∑, δ', q_0, F') onde

F' = F ∪ $\{q_0\}$ se ε-CLOSURE($\{q_0\}$) possui um estado de F, ou
F nos outros casos,

e δ'(q,a) é ξ(q, a) para $q \in K$ e $a \in \sum$. Note que M' não possue transições ε.

Base: |x| = 1. Então x é um símbolo a, e $\delta'(q_0,a) = \xi(q_0, a)$ por definição de δ'.

Indução: |x| > 1. Considere $x = wa$ para símbolo $a \in \sum$. Então
$\delta'(q_0,wa) = \delta'(\delta'(q_0,w), a)$

Pela hipótese indutiva, $\delta'(q_0,w) = \xi(q_0, w)$. Considere $\xi(q_0, w) = P$. É preciso mostrar que $\delta'(P,a) = \xi(q_0, wa)$. Porém
$\delta'(P,a) = \cup_{q \text{ em } P} \delta'(q, a) = \cup_{q \text{ em } P} \xi(q, a)$.

Então quando $P = \xi(q_0,w)$ temos
$\cup_{q \text{ em } P} \xi(q, a) = \xi(q, wa)$

pela regra (2) definição de ξ. Então
$\delta'(q_0,wa) = \xi(q_0,wa)$.

Para completar a prova devemos mostrar que $\delta'(q_0,x)$ contem um estado de F' se e somente se $\xi(q_0, x)$ contem um estado de F. Se $x = \varepsilon$, então a condição é imediata a partir da definição de F'. Isso é, $\delta'(q_0,x) = \{q_0\}$, e q_0 é colocado em F' quando $\xi(q_0, x)$, que é o ε-CLOSURE(q_0), contem um

estado ∈ F. Se $x \neq \varepsilon$, então $x = wa$ para algum símbolo a. Se $\xi(q_0, x)$ contem um estado de F, então $\delta'(q_0,x)$ contem o mesmo estado de F'. Do mesmo jeito, se $\delta'(q_0,x)$ contem um estado de F' diferente de q_0, então $\xi(q_0, x)$ contem um estado de F. Se $\delta'(q_0,x)$ contem q_0, e q_0 não pertence a F, então como $\xi(q_0, x) = \varepsilon\text{-CLOSURE}(\delta(\xi(q_0,w),a))$, o estado em ε-CLOSURE(q_0) e em F precisa estar em $\xi(q_0, x)$.

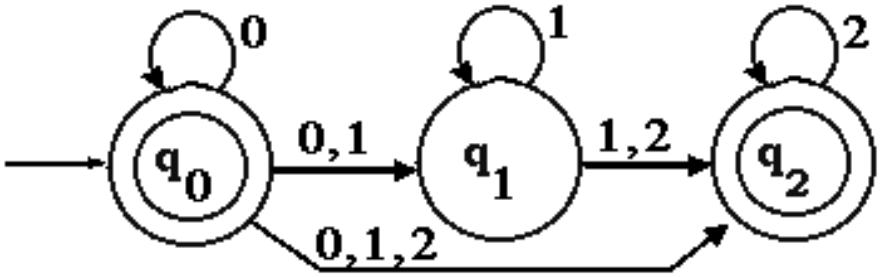
Exemplo:

Considere o AFND-ε dado anteriormente, vamos transformá-lo em AFND.

F' = $\{q_0, q_1, q_2\}$

	δ'	0	1	2
*→	q_0	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
	q_1	∅	$\{q_1, q_2\}$	$\{q_2\}$
*	q_2	∅	∅	$\{q_2\}$

O diagrama seria assim:



3.4 Relação entre Autômatos Finitos e Gramáticas Regulares

Teorema:

Se G = (N, T, P, S) é uma gramática do tipo regular então existe um autômato finito M = (K, T, δ, S, F) tal que T(M) = L(G).

- M é um AFND (ou AFND-ε)
- os estados de M são as variáveis de G, mais um estado adicional a, A ∉ N

  K = N ∪ {A}
- o estado inicial de M é S
- se P tem produção S → ε então F = {S,A}, caso contrário F = {A}
- transições de M:
  1. δ(B,a) = A para cada B → a ∈ P
  2. δ(B,a) = C para cada B → aC ∈ P
  3. δ(A,a) = ∅ para todo a ∈ T

Exemplo:

G = ({S,B}, {0,1}, P, S)

P: $S \rightarrow 0B$
$B \rightarrow 0B \mid 1S \mid 0$

Construir um autômato finito que aceite L(G):

M = ({S,B,A}, {0,1}, δ, S, {A}) é um AFND (ou AFND-ε)

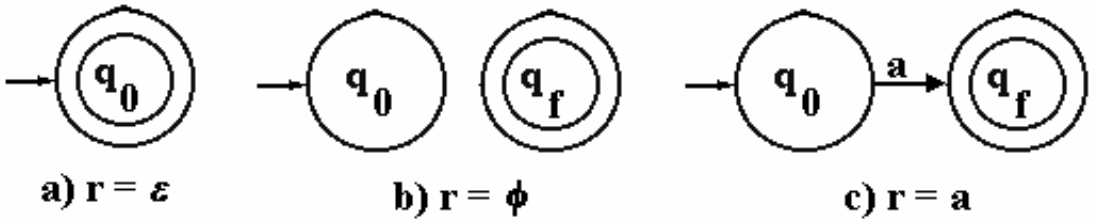
δ(S,0) = {B}	$S \rightarrow 0B$
δ(S,1) = ∅	
δ(B,0) = {B,A}	$B \rightarrow 0B \mid 0$
δ(B,1) = {S}	$B \rightarrow 1S$

3.5 Equivalência entre Autômatos Finitos e Expressões Regulares

Teorema:
Considere que *r* é uma expressão regular. Então existe um AFND-ε que aceita L(r).

Prova:
Será mostrado por indução no número de operadores da ER que existe um AFND-ε, tendo um estado final que não possui saída a partir dele, tal que L(M) = L(r).

Base: (Sem operadores)
A ER r precisa ser ε, ∅, ou *a* para algum $a \in \Sigma$. Os AFND's abaixo claramente satisfazem as condições.



Indução: (um ou mais operadores)
Assuma que o teorema é verdadeiro para ER com menos que *i* operadores, *i* >= 1. Considere *r* tendo *i* operadores, Existe 3 casos dependendo da forma de *r*.

CASO 1: $r = r_1 + r_2$. r_1 e r_2 precisam ter menos que i operadores. Então existe AFND-ε's $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ e $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ com $L(M_1) = L(r_1)$ e $L(M_2) = L(r_2)$. Já que nós podemos renomear os estados de

um AF ao nosso prazer, nós assumimos que Q_1 e Q_2 são desarticulados. Considere q_0 como sendo um novo estado e f_0 um novo estado final. Construa

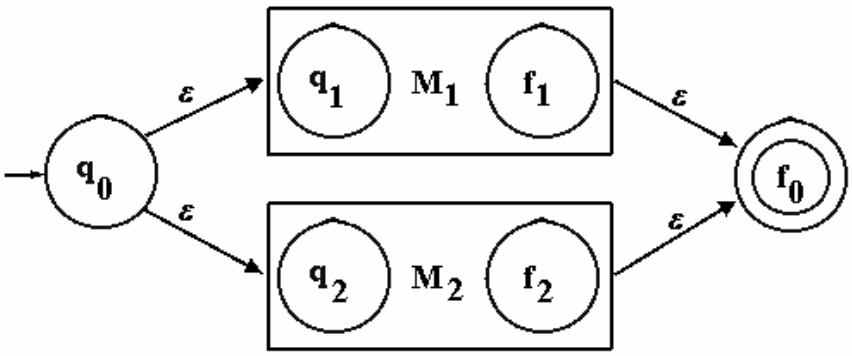
$$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \ \Sigma_1 \cup \Sigma_2, \delta, q_0, \{ f_0\})$$

onde δ é definido por

i) $\delta(q_0, \varepsilon) = \{q_1, q_2\}$,
ii) $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{ f_1\}$ e $a \in \Sigma_1 \cup \{\varepsilon\}$,
iii) $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2 - \{ f_2\}$ e $a \in \Sigma_2 \cup \{\varepsilon\}$,
iv) $\delta(f_1, \varepsilon) = \delta(f_2, \varepsilon) = \{f_0\}$.

Lembre-se que pela hipótese indutiva que não existe transições fora de f_1 e f_2 em M_1 ou M_2. Então todos os movimentos de M_1 e M_2 estão presentes em M.

A construção de M é mostrada na figura abaixo. Qualquer caminho no diagrama de transição de M a partir de q_0 para f_0 precisa começar indo para q_1 ou q_2 com ε. Se o caminho vai para q_1, então ele seguirá qualquer caminho em M_1 para f_1 e então irá para f_0 com ε. Similarmente, podemos concluir o mesmo com caminhos através de M_2. Estes são os únicos caminhos possíveis de q_0 para f_0. Segue-se imediatamente que existe um caminho x em M a partir de q_0 para f_0 se e somente se existe um caminho x em M_1 a partir de q_1 para f_1 ou caminho x em M_2 a partir de q_2 para f_2. Então $L(M) = L(M_1) \cup L(M_2)$ como desejado.



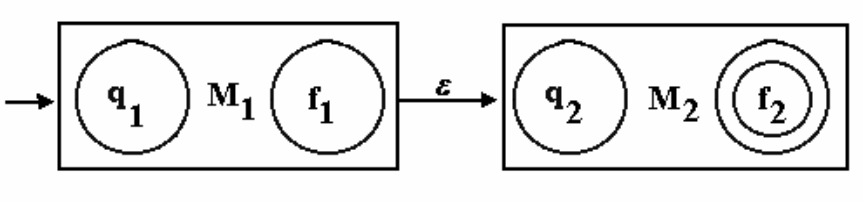
CASO 2: $r = r_1 r_2$. Considere M_1 e M_2 sendo como em CASO 1 e construa

$$M = (Q_1 \cup Q_2, \ \Sigma_1 \cup \Sigma_2, \delta, \{q_0\}, \{ f_2\})$$

onde δ é dado por

i) $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{ f_1\}$ e $a \in \Sigma_1 \cup \{\varepsilon\}$,
ii) $\delta(f_1, \varepsilon) = \{q_2\}$,
iii) $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2$ e $a \in \Sigma_2 \cup \{\varepsilon\}$.

A construção de M é mostrada na figura abaixo. Cada caminho em M a partir de q_1 para f_2 é um caminho x de q_1 para f_1, seguido pela ligação de f_1 para q_2 por ε seguido por um caminho y de q_2 para f_2. Então L(M) = { xy | x $\in$L(M_1) e y $\in$ L(M_2)} e L(M) = L(M_1) L(M_2) como desejado.



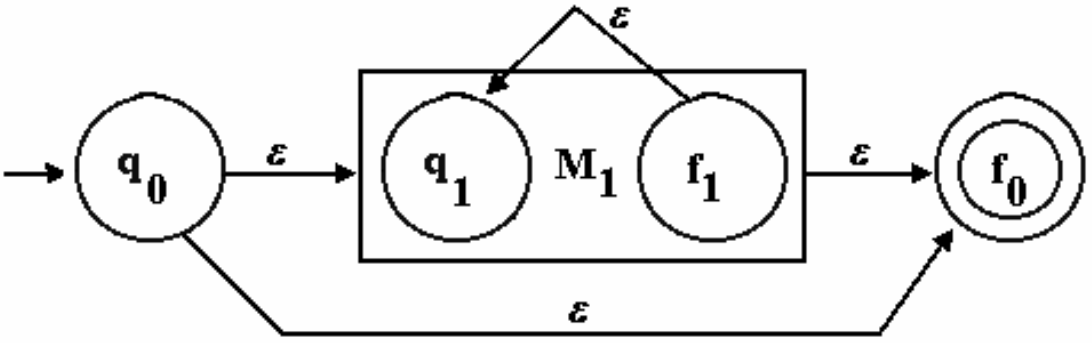
CASO 3: r = r_1^*. Considere M_1 sendo como em CASO 1 e L(M_1) = r_1, construa

$$M = (Q_1 \cup \{q_0, f_0\}, \ \Sigma_1, \delta, q_0, \{ f_0\})$$

onde δ é dado por

i) $\delta(q_0, \varepsilon) = \delta_1(f_1, \varepsilon) = \{q_1, f_0\}$,
ii) $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 - \{ f_1\}$ e $a \in \Sigma_1 \cup \{\varepsilon\}$.

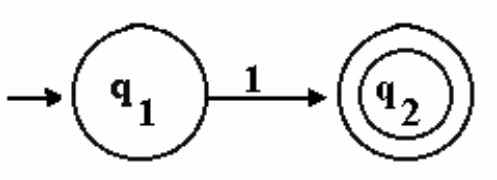
A construção de M é mostrada na figura abaixo. Cada caminho a partir de q_0 para f_0 consiste de um caminho de q_0 para f_0 com ε ou um caminho de q_0 para f_1 com ε seguido por algum número (possivelmente zero) de caminhos de q_1 para f_1 e então novamente para q_1 com ε, cada caminho determinado por uma sequência de símbolos $\in$ L(M_1), seguido por um caminho de q_1 para f_1 com com uma sequência de símbolos $\in$ L(M_1) e finalmente para f_0 com ε. Então existe um caminho x em M a partir de q_0 para f_0 se e somente se nós podemos escrever x = x_1 x_2 ... x_j para algum j >= 0 (o caso j = 0 é x = ε) tal que cada x_i está em L(M_1). Então L(M) = L(M_1)* como desejado.



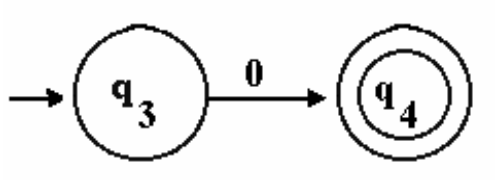
Exemplo:

Construir um AFND-ε para expressão regula **0 1* | 1**.

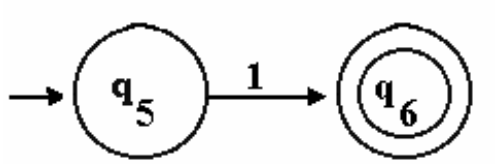
Podemos reescrever essa expressão com parenteses para ficar mais claro a precedência. Ela ficaria assim: (0(1*)) | 1, então essa expressão é na forma $r_1 + r_2$, onde $r_1 = 01^*$ e $r_2 = 1$. O autômato para r_2 é fácil, sendo representado abaixo.



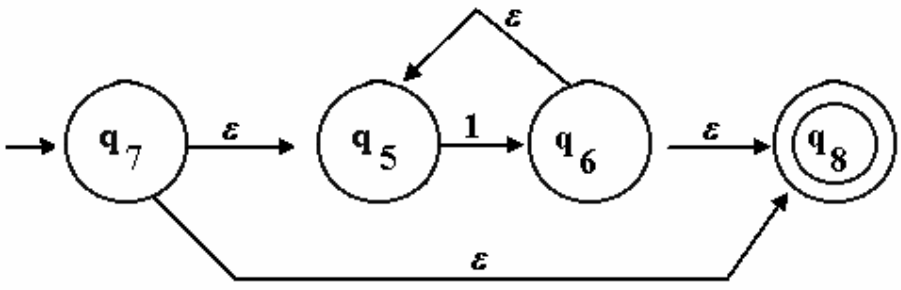
Nós podemos expressar r_1 como $r_3\ r_4$, onde $r_3 = 0$ e $r_4 = 1^*$. O autômato para r_3 seria também facilmente construido:



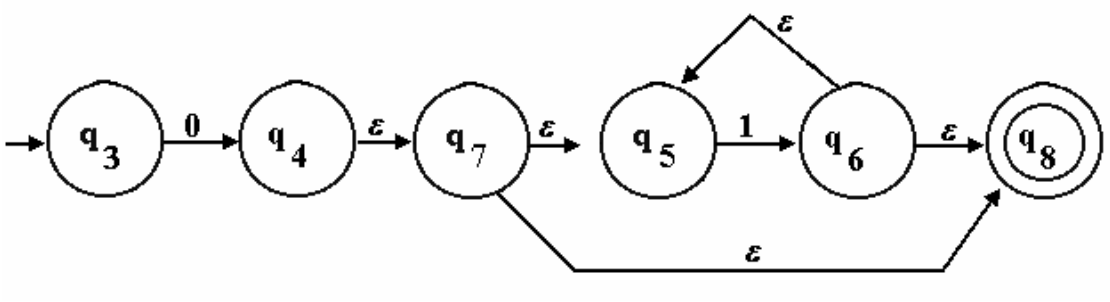
Podemos notar que r_4 é r_5^* onde $r_5 = 1$. Um AF para r_5 seria



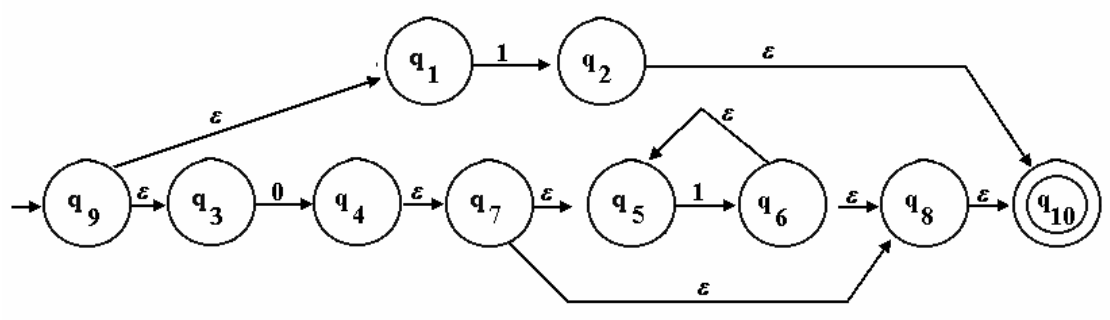
Para construir um AF para r_5^* usa-se o CASO 3 visto anteriormente. O AF resultante é mostrado abaixo.



Então, para $r_1 = r_3\ r_4$ usa-se o CASO 2 visto anteriormente. o AF resultante seria o seguinte:



E finalmente, utiliza-se o CASO 1 para completar o AF para $r = r_1 + r_2$. O AF final é mostrado abaixo.



3.6 Minização de AFD completo

Um autômato finito M = (K, T, δ, q_0, F) é mínimo quando:

1. não possuir estados inacessíveis;
2. não possuir estados mortos;
3. não possuir estados equivalentes.

Construção do AF mínimo:

1. retirar de K os estados inacessíveis, ou seja, todo estado $q \in K$ para o qual não exista nenhuma sentença ω tal que, a partir de q_0, q seja alcançado durante o reconhecimento;

2. retirar de K os estados mortos, ou seja, todo estado que não seja estado final a partir do qual nenhum estado final poderá ser alcançado durante um reconhecimento;

3. construir todas as possíveis classes de equivalência de estados;

4. construir M' = (K', T, δ', q_0', F') conforme segue:
   - K' é o conjunto das classes de equivalência obtidas;
   - q_0' é a classe de equivalência que contém q_0;
   - F' é o conjunto das classes de equivalência que contém pelo menos um estado de F;
   - δ' é o conjunto de transições tais que δ'([p], a) = [q] para toda transição δ(p_1, a) = q_1, onde p_1 e q_1 são elementos de [p] e [q] respectivamente.

Construção das Classes de Equivalência:

Um conjunto de estados q_1, q_2, ..., q_j está em uma mesma classe de equivalência se todas as transições possíveis a partir de cada um destes estados levam o autômato aos estados

q_i, q_{i+1}, ..., q_n,

estando estes últimos todos em uma mesma classe de equivalência.

Para construir as classes de equivalência procede-se da seguinte maneira:

1. criar um estado ∅ para representar as transições indefinidas;

2. dividir K em duas classes de equivalência iniciais, uma contendo os estados finais e a outra todos os demais estados de K;

3. dividir sucessivamente as classes obtidas, de acordo com o conceito de classe de equivalência, até que nenhuma nova classe possa ser obtida.

Exemplo:

Construa o AF mínimo que reconheça a mesma da linguagem do AF da tabela abaixo:

	δ	a	b
→	q_0	q_1	q_5
	q_1	-	q_2
*	q_2	q_3	q_2
*	q_3	q_3	q_3
	q_4	q_4	q_1
	q_5	q_5	q_5

a) estado inacessível : q4
b) estado morto : q5

c) classes de equivalência :

	δ‘	a	b
→	q_0	q_1	0
	q_1	0	q_2
*	q_2	q_3	q_2
*	q_3	q_3	q_3
	∅	∅	∅

d) Autômato Finito sem Classes Equivalentes :

	δ“	a	b
*	[0]	[0]	[0]
→	[1]	[3]	[2]
	[2]	[2]	[2]
	[3]	[2]	[0]

e) Autômato Finito Mínimo :

	δ‘“	a	b
*	[0]	0	0
→	[1]	3	-
	[3]	-	0

4 Gramáticas Livres de Contexto (GLC)

As Gramáticas Livres de Contexto tem grande importância dentro do estudo das Linguagens Formais pois através delas pode ser descrita a maior parte das construções sintáticas das linguagens de programação.

4.1 Extensão da definição de GLC

Vamos estender a definição de GLC para permitir quaisquer produções da forma:

$$A \rightarrow \varepsilon$$

Estas produções, cujo lado direito contém somente a sentença vazia, são chamadas de ε-produções. Deste modo, uma GLC é definida formalmente por regras de produção dadas por:

$$P = \{ A \rightarrow \beta \mid A \in N, \beta \in (N \cup T)^* \}$$

GLC ε-livre

Uma GLC é ε-livre quando não possui ε-produções ou quando possui uma única ε-produção, $S \rightarrow \varepsilon$, onde S é o símbolo inicial da gramática e S não aparece do lado direito de nenhuma regra de produção.

4.2 Árvores de Derivação para GLC

As árvores de derivação são representações gráficas para as derivações nas GLC. Através destas, temos representada explicitamente a estrutura hierárquica que está implícita na linguagem.

Formalmente, consideremos que G = (N, T, P, S) seja uma GLC. Uma árvore é uma *árvore de derivação* para G se:

1. todo nodo tem um rótulo que é um símbolo de $N \cup T \cup \{ \varepsilon\}$;

2. o rótulo da raiz é S;

3. se um nodo A tem um ou mais descendentes, então A é um elemento de N;

4. se A1, A2, ..., An são descendentes diretos de A, da esquerda para a direita, então A → A1 A2 ... An é uma produção de P;

5. se D é a única subárvore da raiz e tem rótulo ε, então a regra S → ε ∈ P.

Exemplo:

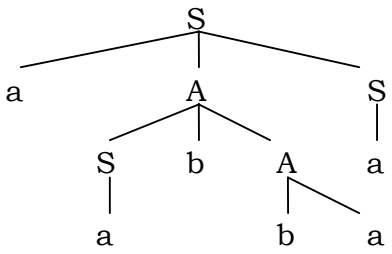
Considere a gramática G = ({S, A}, {a, b}, P, S), onde P consiste de:

S → aAS | a
A → SbA | SS | ba

a derivação da sentença *aabbaa* é dada por:

S → aAS → aSbAS → aabAS → aabbaS → aabbaa

A árvore de derivação correspondente a essa sentença seria:



Profundidade da Árvore de Derivação

É o comprimento do maior caminho entre a raiz e um nodo terminal. No exemplo anterior, a árvore de derivação da sentença tem profundidade 3.

Limite de uma Árvore de Derivação

É a sequência formada pela concatenação, da esquerda para a direita, das folhas da árvore de derivação.

4.3 Derivação mais à esquerda e mais à direita

Uma árvore de derivação ignora variações na ordem em que os símbolos foram substituídos na derivação. Por exemplo, na gramática de expressão aritmética abaixo:

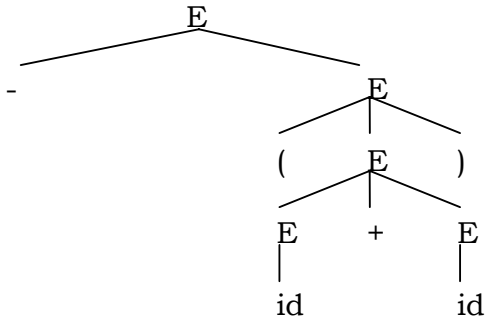
G = ({E}, {+, *. (,), -, id}, P, E)

onde
P = $E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow - E$
$E \rightarrow id$

a sentença "- (id * id)" pode ser derivada de dois modos diferentes:

$E \rightarrow - E \rightarrow - (E) \rightarrow - (E + E) \rightarrow - (id + E) \rightarrow - (id + id)$
ou
$E \rightarrow - E \rightarrow - (E) \rightarrow - (E + E) \rightarrow - (E + id) \rightarrow - (id + id)$

As derivações acima correspondem à mesma árvore de derivação:



Uma derivação é chamada de **mais à esquerda** quando o símbolo substituído for o não-terminal **mais à esquerda** da forma sentencial. Na derivação mais à direita, o símbolo substituído é o não-terminal mais à direita.

Nas duas derivações da sentença "- (id + id)" mostradas acima, a primeira é mais à esquerda e a segunda mais à direita.

Exemplo:

Para a mesma gramática anterior, obtenha as derivações mais à esquerda e mais à direita da sentença

id + id * id

Solução:

Derivação mais à esquerda

$E \rightarrow E + E \rightarrow id + E \rightarrow id + E * E \rightarrow id + id * E \rightarrow id + id * id$

Derivação mais à direita

$E \rightarrow E + E \rightarrow E + E * E \rightarrow E + E * id \rightarrow E + id * id \rightarrow id + id * id$

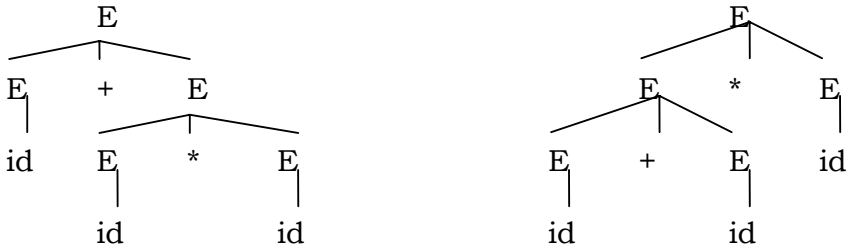
4.4 Gramática Ambígua

Uma GLC é ambígua quando, para alguma sentença da linguagem gerada, existe mais de uma árvore de derivação.

Exemplo:

A gramática de expressão aritmética apresentada antes é ambígua. Isto pode ser visto através de duas árvores de derivação diferentes para a sentença vista:

id + id * id



As duas derivações analisadas no exemplo anterior estão representadas na primeira árvore de derivação.

Linguagens Inerentemente Ambígua

É uma linguagem para a qual todas as GLC que a geram são ambíguas.

Exemplo de linguagem inerentemente ambígua:

$L = \{ a^n b^n c^m d^m \mid n >= 1, m >= 1\} \cup \{ a^n b^m c^m d^n \mid n >= 1, m >= 1\}$

4.5 Transformações em G.L.C

A seguir serão vistas algumas transformações que podem ser efetuadas em GLC's com o objetivo de torná-las mais simples ou de prepará-las para posteriores aplicações. É importante notar que,

qualquer que seja a transformação efetuada, a linguagem gerada deverá ser sempre a mesma.

4.5.1 Eliminação de Símbolos Inúteis

Em uma GLC, um símbolo (terminal ou não-terminal) é inútil se ele não aparece na derivação de nenhuma sentença. Ou seja, um símbolo é inútil se ele é estéril (isto é, não gera nenhuma sequência de terminais pertencente a uma sentença) ou inalcançável (isto é, não aparece em nenhuma forma sentencial da gramática).

Determinação do conjunto de símbolos férteis

Pode ser efetuada através do seguinte algoritmo:

a) Construir o conjunto $N_0 = \varnothing$ e fazer i = 1

b) Repetir
$N_i = N_{i-1} \cup \{ A \mid A \rightarrow \alpha \in P \text{ e } \alpha \in (N_{i-1} \cup T)^* \}$
$i = i + 1$
até que $N_i = N_{i-1}$

c) N_i é o conjunto de símbolos férteis.

Se o símbolo inicial não fizer parte do conjunto de símbolos férteis, a linguagem gerada pela gramática é **vazia**.

Exemplo:
Retirar os símbolos estéreis da gramática:

G = ({S,A,B,C,D}, {a,b,c,d}, P, S)
P: $S \rightarrow a\ A$
$A \rightarrow a \mid b\ B$
$B \rightarrow b \mid d\ D$
$C \rightarrow cC \mid c$
$D \rightarrow d\ D$

Solução:
$N_0 = \varnothing$
$N_1 = \{A,B,C\}$
$N_2 = \{S,A,B,C\}$
$N_3 = \{S,A,B,C\} = N_2$
Conjunto de símbolos férteis: {S,A,B,C}
Gramática simplificada:
G' = ({S,A,B,C}, {a,b,c}, P', S)
P': $S \rightarrow a\ A$
$A \rightarrow a \mid b\ B$

$B \rightarrow b$
$C \rightarrow cC \mid c$

Determinação do conjunto de símbolos alcançáveis

Pode ser efetuada através do seguinte algoritmo:

a) Construir o conjunto $V_0 = \{S\}$ (S = símbolo inicial) e fazer i = 1

b) Repetir
$V_i = \{ X \mid$ existe algum $A \rightarrow \alpha X \beta$ e $A \in V_{i-1}$ e $\alpha, \beta \in (N \cup T)^*\} \cup V_{i-1}$
$i = i + 1$
até que $V_i = V_{i-1}$

c) V_i é o conjunto de símbolos alcançáveis.

Exemplo: Simplificar a gramática G' do exemplo anterior, retirando os símbolos inalcançáveis.

Solução:
$V_0 = \{S\}$
$V_1 = \{S, a, A\}$
$V_2 = \{S, a, A, b, B\}$
$V_3 = \{S, a, A, b, B\} = V_2$

Conjunto de símbolos alcançáveis: {S, a, A, b, B}
Gramática simplificada:
G' = ({S,A,B}, {a,b}, P", S)
P": $S \rightarrow a\ A$
$A \rightarrow a \mid b\ B$
$B \rightarrow b$

OBS: Note que aparentemente os dois passos são independentes e podem ser aplicados em qualquer ordem. Isto não é verdadeiro. Isto ocorre porque a eliminação de símbolos inativos pode ocasionar o surgimento de símbolos inatingíveis. O contrário não é verdadeiro.

4.5.2 Transformação de uma GLC qualquer para uma GLC ε-Livre

Esta transformação sempre é possível e pode ser efetuada pelo seguinte algoritmo:

a) Reunir em um conjunto os não-terminais que derivam direta ou indiretamente a sentença vazia:
$Ne = \{A \mid A \in N \text{ e } A \rightarrow^{+} \varepsilon\}$

b) Construir o conjunto de regras P' como segue:

b1) incluir em P' todas as regras de P, com exceção daquelas da forma $A \rightarrow \varepsilon$

b2) para cada ocorrência de um símbolo Ne do lado direito de alguma regra de P, incluir em P' mais uma regra, substituindo este símbolo por ε. Isto é, para regra de P do tipo
$A \rightarrow \alpha B \beta$, $B \in Ne$ e $\alpha, \beta \in V^*$
incluir em P' a regra
$A \rightarrow \alpha\ \beta$

c) Se $S \in Ne$, adicionar a P' as regras $S' \rightarrow S$ e $S' \rightarrow \varepsilon$, sendo que N' ficará igual a $N \cup S'$. Caso contrário trocar os nomes de S por S' e N por N'.

d) A nova gramática será definida por:
G' = (N', T, P', S')

Exemplo:

Transformar as GLC abaixo, definidas pelo respectivo conjunto de regras de produção P, para GLC ε-Livres.

a) G = ({S, B}, {a,b}, P, S)
P: $S \rightarrow a\ B$
$B \rightarrow b\ B \mid \varepsilon$
Solução:
Ne = {B}
P ': $S \rightarrow a\ B \mid a$
$B \rightarrow b\ B \mid b$

b) G = ({S, D, C}, {b,c,d,e}, P, S)
P: $S \rightarrow b\ D\ C\ e$
$D \rightarrow d\ D \mid \varepsilon$
$C \rightarrow c\ C \mid \varepsilon$
Solução:
Ne = {D, C}
P ': $S \rightarrow b\ D\ C\ e \mid b\ C\ e \mid b\ D\ e \mid b\ e$
$D \rightarrow d\ D \mid d$
$C \rightarrow c\ C \mid c$

c) G = ({S}, {a}, P, S)
P: $S \rightarrow a\ S \mid \varepsilon$
Solução:
Ne = {S}
P ': $S' \rightarrow S \mid \varepsilon$
$S \rightarrow a\ S \mid a$

4.5.3 Remoção de Produções Simples (unitárias)

Produções simples são produções da forma $A \rightarrow B$ onde A e $B \in N$. Estas produções podem ser removidas de uma GLC através do seguinte algoritmo:

a) Transformar a GLC em uma GLC ε-livre, se necessário

b) Para todo não-terminal de N, construir um conjunto com os não-terminais que ele pode derivar, em um ou mais passos. Isto é, para todo $A \in N$, construir $N_A = \{ B \mid A \ ^*\!\rightarrow B \}$

c) Construir P' como segue:
se $B \rightarrow \alpha \in P$ e não é uma produção simples, adicione a P' as produções:

$$A \rightarrow \alpha \text{ para todo } A \mid B \in N_A$$

d) A GLC equivalente, sem produções simples, será definida por:
G' = (N, T, P', S)

Exemplo:
Transformar as GLC abaixo em gramáticas equivalentes que não apresentem produções simples.

a) G = ({S, A}, {a,b}, P, S)
P: $S \rightarrow b\ S \mid A$
$A \rightarrow a\ A \mid a$
Solução:
Ns = {A}
$N_A = \{\}$
P ': $S \rightarrow b\ S \mid a\ A \mid a$
$A \rightarrow a\ A \mid a$

b) G = ({S, A, B}, {a,b,c}, P, S)
P: $S \rightarrow a\ S\ b \mid A$
$A \rightarrow a\ A \mid B$
$B \rightarrow b\ B\ c \mid b\ c$
Solução:
Ns = {A, B}
$N_A = \{B\}$
$N_B = \{\}$
P ': $S \rightarrow a\ S\ b \mid a\ A \mid b\ B\ c \mid b\ c$
$A \rightarrow a\ A \mid b\ B\ c \mid b\ c$
$B \rightarrow b\ B\ c \mid b\ c$

4.5.4 Fatoração de GLC

Uma GLC está fatorada se ela é determinística, isto é, não possui produções cujo ladodireito inicie com o mesmo conjunto de símbolos ou com símbolos que gerem sequências que iniciem com o mesmo conjunto de símbolos. Por exemplo, a gramática fatorada **não** deverá apresentar as seguintes regras:

$$A \rightarrow a\ B \mid a\ C$$

pois as duas iniciam com o mesmo terminal **a**. Outro exemplo de gramática não-fatorada é o seguinte:

$$S \rightarrow A \mid B$$
$$A \rightarrow a\ c$$
$$B \rightarrow a\ b$$

Para fatorar uma GLC devemos alterar as produções envolvidas no não-determinismo da seguinte maneira:

a) as produções que apresentam não-determinismo direto, da forma

$$A \rightarrow \alpha\ \beta \mid \alpha\ \delta$$

serão substituídas por

$$A \rightarrow \alpha\ A'$$
$$A' \rightarrow \beta \mid \delta$$

sendo A' um novo não-terminal

b) O não-determinismo indireto é retirado fazendo, nas regras de produção, as derivações necessárias para torná-lo um determinismo direto, resolvido posteriormente como no item anterior.

Exemplo:

Fatorar as GLC abaixo

a) G = ({S, A, B}, {a,b}, P, S)

P: $S \rightarrow a\ A \mid a\ B$
$A \rightarrow a\ A \mid a$
$B \rightarrow b$

Solução:

P ': $S \rightarrow a\ S'$
$S' \rightarrow A \mid B$
$A \rightarrow a\ A'$
$A' \rightarrow A \mid \varepsilon$
$B \rightarrow b$

b) G = ({S, A}, {a,b}, P, S)
P: S → A b | a b | b a A
A → a a b | b
Solução:
P ‘: S → a a b b | b b | a b | b a A
A → a a b | b
P”: S → a S’ | b S”
S’ → a b b | b
S” → b | a A
A → a a b | b

4.5.5 Eliminação de Recursão à Esquerda (e a Direita)

Uma gramática G = (N, T, P, S) tem recursão à esquerda se existe $A \in N$ tal que $A \overset{+}{\rightarrow} A\,\alpha$, $\alpha \in (N \cup T)^*$

Uma gramática G = (N, T, P, S) tem recursão à direita se existe $A \in N$ tal que $A \overset{+}{\rightarrow} \alpha\,A$, $\alpha \in (N \cup T)^*$

A recursão é dita direta se a derivação acima for em um passo, isto é:

- G tem recursão direta à esquerda se existe produção $A \rightarrow A\,\alpha \in P$

- G tem recursão direta à direita se existe produção $A \rightarrow \alpha\,A \in P$

Para eliminar a recursão direta à esquerda usa-se o seguinte algoritmo:

Seja G = (N, T, P, S) uma GLC sem produções ε e sem produções do tipo $A \rightarrow A$.

a) Para eliminar recursão direta à esquerda envolvendo um símbolo não-terminal A dividimos inicialmente o conjunto das produções de P do tipo $A \rightarrow \alpha$ em subconjuntos:

C1 = conjunto das produções $A \rightarrow \alpha$ que apresentam recursão direta à esquerda, ou seja, $A \rightarrow A\,\alpha \in P$ onde $\alpha\ (N \cup T)^*$.

C2 = conjunto das produções $A \rightarrow \beta$ que não apresentam recursão direta à esquerda, ou seja, $A \rightarrow \beta \in P \mid \beta \neq A\,\alpha$ para qualquer α.

b) Cria-se um novo não-terminal B

c) Substitua as produções $A \rightarrow \alpha$ da gramática original de acordo com os seguintes passos:

c.1) para cada produção $A \rightarrow \beta_i \in C2$ criar a produção

$$A \rightarrow \beta_i B$$

c.2) para cada produção $A \rightarrow A\alpha_i \in C1$ criar as produções

$$B \rightarrow \alpha_i B$$
$$B \rightarrow \varepsilon$$

OBS: A recursão direta à esquerda foi transferida para recursão à direita. Se a gramática original tiver produções $A \rightarrow A$ poderão surgir produções ε após a execução do algoritmo. Além disso, podem aparecer produções simples após a execução do algortimo.

O algoritmo para eliminação de recursão direta à direita é análogo.

Para eliminar a recursão geral à esquerda usa-se o seguinte algoritmo:

Método: Parte-se de uma gramática G sem produções ε

a) Estabeleça uma **ordenação** qualquer no conjunto de **não-terminais** da gramática original G, isto é, faça $N = \{ A_1, A_2, A_3, \ldots, A_n\}$, sendo que $A_1 < A_2 < A_3 < \ldots < A_n$

b) Gere uma GLC G' = (N', T, P', S) equivalente em que para toda regra do tipo $A_i \rightarrow A_j \alpha$ onde $\alpha \in (N \cup T)^*$, deve ocorrer $A_i < A_j$.

c) $k = 0$
enquanto existir k tal que existe produção $A_k \rightarrow A_j \alpha$ tal que $k >= j$, isto é, existe produção fora do padrão $A_i \rightarrow A_j \alpha$, $i < j$
faça:
- procure o **menor** inteiro k tal que exista produção $A_k \rightarrow A_j \alpha$ tal que $k >= j$
  OBS: Isto significa que para este k
  - existe produção "incorreta" $A_k \rightarrow A_j \alpha$ onde $k >= j$
  - para todo $i < k$, todas as produções $A_i \rightarrow A_j \alpha$ estão "corretas", isto é, $i < j$
- enquanto houver produções do tipo $A_k \rightarrow A_j \alpha$, onde $k > j$ faça:
  - eliminar a produção $A_k \rightarrow A_j \alpha$
  - para cada produção $A_j \rightarrow \beta$ existentes criar a produção $A_k \rightarrow \beta \alpha$
- se houver recursão direta à esquerda com o símbolo A_k, isto é, existe a produção $A_k \rightarrow A_k \alpha$, elimine-a pelo algortimo dado anteriormente.

Exemplo:

Elimine as recursões à esquerda da GLC abaixo

G = (N, T, P, S)
P: $S \rightarrow A\ a$
$A \rightarrow S\ b \mid c\ A \mid a$

Solução:

P': $S \rightarrow A\ a$
$A \rightarrow A\ a\ b \mid c\ A \mid a$
P": $S \rightarrow A\ a$
$A \rightarrow c\ A\ A' \mid a\ A'$
$A' \rightarrow a\ b\ A' \mid \varepsilon$

4.6 Forma Normal de Chomsky (CNF)

Uma GLC está na Forma Normal de Chomsky se ela é ε-livre e apresenta todas as produções da forma

$A \rightarrow B\ C$ ou $A \rightarrow a$

onde

$A, B, C \in N$ e $a \in T$.

Prova-se que toda Linguagem Livre de Contexto ε-livre pode ser gerada por uma GLC na Forma Normal de Chomsky.

Uma GLC ε-livre G = (N, T, P, S) pode ser colocada na CNF através do seguinte algoritmo:

a) obter G' = (N', T, P', S) a partir de G, removendo de G as suas produções simples, de modo que L(G') = L(G)

b) nas regras de G' cujo lado direito apresenta mais de um termo, substituir cada terminal (por exemplo, $a \in T$) por um novo não-terminal (A_a), incluindo para cada um destes novos não-terminais uma nova regra, $A_a \rightarrow a$, resultando G" = (N", T, P", S)

c) substituir cada regra do tipo
$A \rightarrow B_1\ B_2\ \ldots\ B_m$, $m >= 3$
onde A, B_1, B_2, ..., B_m são não-terminais, pelo conjunto de regras
$A \rightarrow B_1\ B'_1$
$B'_1 \rightarrow B_2\ B'_2$
...
$B'_{m-2} \rightarrow B_{m-1}\ B_m$
onde B'_1, B'_2, ..., B_{m-2} são novos não-terminais

d) a gramática gerada está na CNF e é dada por
G"' = (N"', T, P"', S)

Exemplo:

Dada a GLC abaixo, ache a gramática equivalente na CNF

G = ({S, A, B}, {a, b}, P, S)

P: $S \rightarrow A \mid A\ B\ A$
$A \rightarrow a\ A \mid a$
$B \rightarrow b\ B \mid b$

Solução

a) P': $S \rightarrow a\ A \mid a \mid A\ B\ A$
$A \rightarrow a\ A \mid a$
$B \rightarrow b\ B \mid b$

b) P'': $S \rightarrow A_a\ A \mid a \mid A\ B\ A$
$A \rightarrow A_a\ A \mid a$
$B \rightarrow A_b\ B \mid b$
$A_a \rightarrow a$
$A_b \rightarrow b$

c) P''': $S \rightarrow A_a\ A \mid a \mid A\ B'$
$B' \rightarrow B\ A$
$A \rightarrow A_a\ A \mid a$
$B \rightarrow A_b\ B \mid b$
$A_a \rightarrow a$
$A_b \rightarrow b$

4.7 Forma Normal de Greibach (GNF)

Uma GLC está na Forma Normal de Greibach se ela é ε-livre e apresenta todas as produções na forma:

$$A \rightarrow a\ \alpha$$

onde $A \in N$, $a \in T$ e $\alpha \in N^*$.

Prova-se que toda a Linguagem Livre de Contexto ε-livre pode ser gerada por uma GLC na Forma Normal de Greibach.

Para achar a gramática equivalente a G = (N, T, P, S), na GNF, deve-se seguir os seguintes passos:

a) achar G' = (N', T, P', S) tal que L(G') = L(G) e que G' esteja na CNF

b) ordenar os não-terminais de G' em uma ordem quaisquer - por exemplo: $N' = \{ A_1, A_2, \ldots, A_m \}$

c) modificar as regras de P' de modo a que, se $A_i \rightarrow A_j \gamma$ é uma regra de P', então j > i

d) a gramática obtida do passo anterior, G", apresentará todas as regras de A_m com o lado direito iniciando por um terminal; através de substituições sucessivas dos primeiros termos das regras A_i anteriores, coloca-se estas também nessa forma

e) se no item *c* tiverem sido incluídos novos não-terminais B_i (para retirar recursões à esquerda), fazer também para as regras correspondentes a estes, as devidas substituições dos primeiros termos (que serão sempre terminais ou A_i)

f) a gramática final, G"', está na GNF

Exemplo:

Coloque a GLC abaixo na GNF

G = ({S,A}, {a,b}, P, S)

P: $S \rightarrow A\ S \mid a$
$A \rightarrow S\ A \mid b$

Solução

a) G já está na CNF

b) Renomear os não-terminais (ordem):
$S = A_1$ e $A = A_2$
Logo
P': $A_1 \rightarrow A_2\ A_1 \mid a$
$A_2 \rightarrow A_1\ A_2 \mid b$

c) a única regra a modificar é
$A_2 \rightarrow A_1\ A_2$
substituindo A_1 nessa regra pelas suas regras temos
$A_2 \rightarrow A_2\ A_1\ A_2 \mid a\ A_2 \mid b$
para retirar a recursividade à esquerda da 1ª regra, introduzimos um novo não-terminal B_2 resultando para G":
P": $A_1 \rightarrow A_2\ A_1 \mid a$
$A_2 \rightarrow a\ A_2\ B_2 \mid b\ B_2 \mid a\ A_2 \mid b$
$B_2 \rightarrow A_1\ A_2\ B_2 \mid A_1\ A_2$

d) P"': $A_2 \rightarrow a\ A_2\ B_2 \mid b\ B_2 \mid a\ A_2 \mid b$
$A_1 \rightarrow a\ A_2\ B_2\ A_1 \mid b\ B_2\ A_1 \mid a\ A_2\ A_1 \mid b\ A_1 \mid a$
$B_2 \rightarrow a\ A_2\ B_2\ A_1\ A_2\ B_2 \mid b\ B_2\ A_1\ A_2\ B_2 \mid$
$a\ A_2\ A_1\ A_2\ B_2 \mid b\ A_1\ A_2\ B_2 \mid a\ A_2\ B_2 \mid$
$a\ A_2\ B_2\ A_1\ A_2 \mid b\ B_2\ A_1\ A_2 \mid a\ A_2\ A_1\ A_2 \mid$

b A_1 A_2 | a A_2

4.8 Outros Tipos de GLC

a) **Gramática reduzida**

É uma GLC que satisfaz às seguintes condições:

1 - $L(G) \neq \varnothing$ e
2 - se $A \rightarrow \alpha \in P$ então $A \neq \alpha$ e G não possui símbolos inúteis

b) **Gramática sem ciclos**

É uma GLC que não possui derivação da forma:

$A \rightarrow^{+} A$ para $A \in N$

c) **Gramática Própria**

É uma GLC que

1 - não possui ciclos
2 - é ε-livre
3 - não possui símbolos inúteis

d) **Gramática de Operadores**

É uma GLC que não possui produções da forma:

$A \rightarrow \ldots B\ C \ldots$ onde $A, B, C \in N$

ou seja, é uma GLC na qual não aparecem dois não-terminais juntos em nenhuma regra de produção.

e) Gramática Linear

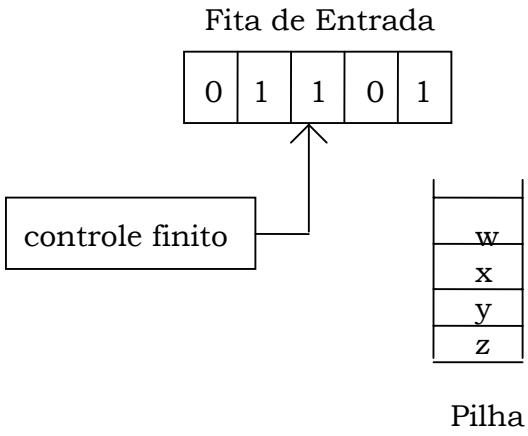
É uma GLC na qual todas as produções se apresentam na forma:

$A \rightarrow x\ B\ w$ ou $A \rightarrow x$ onde $A, B \in N$ e $x, w \in T^*$

5 Autômato de Pilha

O Autômato de Pilha é a máquina abstrata que reconhece as Linguagens Livres de Contexto (LLC). É chamado usualmente de *Pushdown Automata* (PDA). Consiste de um controle finito que tem acesso a uma fita de entrada (onde está a sequência a ser analisada) e a uma pilha. O controle percorre a fita de entrada da esquerda para a direita, lendo um símbolo de cada vez. Na pilha, o controle pode empilhar símbolos, além de retirar ou substituir o símbolo do topo.

Uma característica do PDA é ser não-determinístico.



Pilha

Movimento do Autômato de Pilha

O movimento do PDA é determinado pelo símbolo apontado na entrada, pelo símbolo no topo da pilha e pelo estado do controle finito.

Existem dois tipos de movimentos:

1) um símbolo da fita é lido e o cabeçote avança para o próximo símbolo;

2) *ε-move* - movimento vazio, no qual o cabeçote não se move, não importando qual é o símbolo que está sendo apontado na fita de entrada.

Linguagem Aceita por um PDA

É o conjunto de sentenças que, quando o autômato chegar no final de seu reconhecimento:

a) resultam na pilha vazia ou

b) levam o autômato a estado final.

Estes dois tipos de reconhecimento, por pilha vazia ou por estado final, são equivalentes.

No reconhecimento através de pilha vazia, o conjunto de estados finais do autômato é irrelevante, podendo ser vazio.

Uma linguagem é aceita por um autômato de pilha se e somente se for uma Linguagem Livre de Contexto.

Definição de Autômato de Pilha

Um autômato de pilha M é um sistema definido por:

$M = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ onde

K - conjunto finito de estados
Σ - alfabeto finito de entrada
Γ - alfabeto finito da pilha
q_0 - estado inicial, sendo que $q_0 \in K$
Z_0 - símbolo inicial na pilha, sendo que $Z_0 \in \Gamma$
F - conjunto finito de estados finais, sendo que $F \subseteq K$
δ - mapeamentos

δ: $K \times (\Sigma \cup \varepsilon) \times \Gamma \rightarrow$ subconjuntos de $K \times \Gamma^*$

Tipos de mapeamentos:

1. $\delta (q, a, z) = ((p_1, \xi_1), (p_2, \xi_2), ..., (p_m, \xi_m))$

onde:
$q, p_1, p_2, ..., p_m \in K$ e
$a \in \Sigma, Z \in \Gamma, \xi_1 \in \Gamma^*$
significando que o autômato de pilha **M**, estando no estado **q**, vendo o símbolo **a** na entrada e sendo **Z** o símbolo que está no topo da pilha, pode, para qualquer **i**, entrar no estado $\mathbf{p_i}$, substituir **Z** por $\boldsymbol{\xi_i}$ e avançar o cabeçote de leitura da fita de entrada para o próximo símbolo.

2. $\delta (q, \varepsilon, z) = ((p_1, \xi_1), (p_2, \xi_2), ..., (p_m, \xi_m))$

significando que o autômato de pilha **M**, estando no estado **q**, independente do símbolo de entrada e com **Z** no topo da pilha, pode passar para um estado $\mathbf{p_i}$ e substituir **Z** por $\boldsymbol{\xi_i}$, sem avançar o cabeçote de leitura da entrada.

Costuma-se representar a pilha na horizontal, sendo o símbolo do topo da pilha aquele que está mais à esquerda de ξ_i.

Dependendo do comprimento de ξ, temos as seguintes possibilidades de atuação do autômato de pilha:

$\|\xi\| > 1$	troca o símbolo do topo e empilha outros
$\|\xi\| = 1$	troca o símbolo do topo
$\|\xi\| = 0$	desempilha ($\xi = \varepsilon$).

Autômato de Pilha Determinístico

Um PDA é determinístico se:

1. para cada $q \in K$ e $Z \in \Gamma$, sempre que $\delta (q, \varepsilon m Z) \neq \varnothing$ então $(1, a, Z) = \varnothing$ para todo $a \in \sum$

2. para qualquer $q \in K$ e $Z \in \Gamma$ e $a \in \sum \cup \varepsilon$, $\delta (q, \varepsilon m Z)$ nunca contém mais de um elemento.

A condição 1 evita a possibilidade de uma escolha entre um movimento independente da entrada (ε-move) e um movimento envolvendo um símbolo de entrada, enquanto que a condição 2 previne uma escolha de movimento para qualquer (q, a, Z) ou (q, ε, Z).

Exemplo:
um PDA que aceita, por pilha vazia, a linguagem

$$(w C w^r \mid w \in \{0,1\}^*)$$

onde w^r é a sequência reversa de w, pode ser definido da seguinte maneira:

$M = (\{q_1, q_2\}, \{0,1,C\}, \{Z_0,Z,U\}, \delta, q_1, Z_0, 0)$

$\delta (q_1, 0, Z_0) = \{ (q_1, ZZ_0) \}$
$\delta (q_1, 0, Z) = \{ (q_1, ZZ) \}$
$\delta (q_1, 0, U) = \{ (q_1, ZU) \}$
$\delta (q_1, C, Z_0) = \{ (q_2, Z_0) \}$
$\delta (q_1, C, Z) = \{ (q_2, Z) \}$
$\delta (q_1, C, U) = \{ (q_2, U) \}$
$\delta (q_2, 0, Z) = \{ (q_2, \varepsilon) \}$
$\delta (q_2, \varepsilon, Z_0) = \{ (q_2, \varepsilon) \}$
$\delta (q_1, 1, Z_0) = \{ (q_1, UZ_0) \}$
$\delta (q_1, 1, Z) = \{ (q_1, UZ) \}$
$\delta (q_1, 1, U) = \{ (q_1, UU) \}$
$\delta (q_2, 1, U) = \{ (q_2, \varepsilon) \}$

Configuração de um Autômato de Pilha

Chama-se de configuração de um PDA a um par (q, ξ) onde q ∈ K e ξ ∈ Γ*. Significa que o autômato está no estado q, com ξ na pilha.

A linguagem de uma configuração para outra é representada por:

$$a: (q, Z\xi) \underset{M}{\vdash} (p, \beta\xi)$$

onde

$a \in \sum \cup \varepsilon$
$\xi, \beta \in \Gamma^*$
$Z \in \Gamma$
$(p, \beta) \in \delta (q, a, Z)$

De acordo com as regras do PDA, o símbolo de entrada **a** pode levar **M** a passar de uma configuração (q, Zξ) para outra (p, βξ).

Do mesmo modo, podemos representar:

$$a_1 a_2 \ldots a_n : (q_1, \xi_1) \underset{M}{\vdash}^* (q_{n+1}, \xi_{n+1})$$

se

$a_1, a_2, \ldots, a_n \in \sum \cup \varepsilon$
$q_1, q_2, \ldots, q_{n+1} \in K$
$\xi_1, \xi_2, \ldots, \xi_{n+1} \in \Gamma^*$

e

$$a_i : (q_i, \xi_i) \underset{M}{\vdash} (q_{i+1}, \xi_{i+1}) \text{ para todo } 1 <= i <= n$$

Por convenção:

$$\varepsilon : (q, \xi) \underset{M}{\vdash}^* (q, \xi)$$

Descrição instantânea de um PDA

A descrição instantânea de um PDA é dada por seu estado, o conteúdo de sua pilha e pelo restante da sequência de entrada a analisar. Assim, a descrição instantânea

$$(q_1, bc, AB)$$

significa que o autômato está no estado q_1, falta reconhecer bc e a pilha contém AB (com o símbolo A no topo).

Exemplo:

usando a descrição instantânea, o reconhecimento da sentença

0 1 C 1 0

através do PDA do exemplo anterior seria representado por

$$
\begin{aligned}
(q_1, 01C10, Z_0) &\vdash (q_1, 1C10, ZZ_0) \\
&\vdash (q_1, C10, UZZ_0) \\
&\vdash (q_2, 10, UZZ_0) \\
&\vdash (q_2, 0, ZZ_0) \\
&\vdash (q_2, \varepsilon, Z_0) \\
&\vdash (q_2, \varepsilon, \varepsilon)
\end{aligned}
$$

Linguagem aceita por um PDA

Utilizando a notação vista de configuração de uma autômato de pilha, podemos representar a linguagem aceita por ele de duas maneiras, conforme o modo em que é efetuado o reconhecimento das sentenças:

$T(M)$ = linguagem aceita por estado final
= $\{w \mid w : (q_0, Z_0) \overset{*}{\underset{M}{\vdash}} (q, \xi)$ para $\xi \in \Gamma^*$ e $q \in F\}$

$T(M)$ = linguagem aceita por pilha vazia
= $\{w \mid w : (q_0, Z_0) \overset{*}{\underset{M}{\vdash}} (q, \varepsilon)$ para $q \in F\}$

Representação Gráfica do Autômato de Pilha

O PDA também pode ser representado por um grafo, similar aquele utilizado no autômato finito. Os rótulos dos arcos deverão indicar, além do símbolo apontado na fita de entrada, o símbolo que está no topo da pilha e o conjunto de símbolos que deve substituir aquele que está no topo da pilha durante a transição. Isto é, o rótulo deve ser da forma

$(a, W) \rightarrow Z$ onde

$a \in \Sigma$ é o símbolo de entrada
$W \in \Gamma$ é o símbolo no topo da pilha
$Z \in \Gamma^*$ é a sequência que irá substituir o topo da pilha

Exemplo:

L(M) = (an bn | n >= 0)

As transições representadas no grafo deste exemplo são as seguintes:

δ (q_0, a, Z_0) = { (q_1, XZ_0) }
δ (q_1, a, Z_0) = { (q_1, XX) }
δ (q_1, b, Z_0) = { (q_2, ε) }
δ (q_2, b, Z_0) = { (q_2, ε) }
δ (q_2, ε, Z_0) = { (q_3, ε) }

5.1 PDA e Linguagens Livres de Contexto

Teorema: Se L é uma Linguagem Livre de Contexto (LLC), então existe um autômato de pilha M tal que L = N (M).

Algoritmo para construir M

Seja G = (N, T, P, S) uma GLC na GNF. Assumimos que $\varepsilon \notin$ L(G). O PDA M que reconhece esta linguagem é dado por:

M = ({q_1}, T, N, δ, q1, S, $\varnothing$)

O mapeamento é construído a partir das regras de P, sendo que para regra $A \rightarrow a\,\xi \in P$ corresponde $\delta(q_1, a, A) \supset (q_1, \xi)$

Exemplo:

Construir um PDA M para reconhecer a linguagem gerada pela gramática abaixo:

G = ({S,A}, {a,b}, {S $\rightarrow$ aAA, A $\rightarrow$ bS | aS | a}, S)

Solução:

M = ({q_1}, {a,b}, {S,A}, δ, q_1, S, $\varnothing$)

$\delta(q_1, a, S)$ = { (q_1, AA) }
$\delta(q_1, b, A)$ = { (q_1, S) }
$\delta(q_1, a, A)$ = { (q_1, S), (q_1, ε) }

Teorema: Se L é uma linguagem reconhecida por algum autômato de pilha, então L é uma Linguagem Livre de Contexto, isto é, pode ser gerada por uma gramática G livre de contexto.

Algoritmo para construir G

Seja M = (K, ∑, Γ, δ, q_0, Z_0, ∅) um PDA. A gramática que gera a linguagem reconhecida por este autômato é dada por:

G = (N, ∑, P, S)

onde

N - conjunto de objetos da forma [q, A, p] onde q, p ∈ K e A ∈ Γ, mais o símbolo S

P - obtido da seguinte forma:

1. $S \rightarrow [q_0, Z_0, q]$ para cada q ∈ K
2. $[q, A, p] \rightarrow a\ [q_1, B_1, q_2]\ [q_2, B_2, q_3]\ \ldots\ [q_m, B_m, q_{m+1}]$ para cada q, q_1, q_2, ..., q_{m+1} ∈ K, onde $p = q_{m+1}$, a ∈ ∑ U ε, A, B_1, B_2, ..., B_m ∈ Γ, tal que $\delta(q, a, A) \supset (q1, B_1B_2 \ldots B_m)$. Se m = 0, então q1 = p, $\delta(q, a, A) \supset (p, \varepsilon)$ e a produção é $[q, A, p] \rightarrow a$.

Exemplo:

Seja M = ({q0, q1}, {0,1}, {X, Z0}, δ, q_0, Z_0, ∅) onde

$(q_0, 0, Z_0)$	=	$\{ (q_0, XZ_0) \}$
$(q_0, 0, X)$	=	$\{ (q_0, XX) \}$
$(q_0, 1, X)$	=	$\{ (q_1, \varepsilon) \}$
$(q_1, 1, X)$	=	$\{ (q_1, \varepsilon) \}$
(q_1, ε, X)	=	$\{ (q_1, \varepsilon) \}$
(q_1, ε, Z_0)	=	$\{ (q_1, \varepsilon) \}$

Construir uma GLC G = (N, T, P, S) tal que L(G) = T(M)

Solução:

N = { S, $[q_0, X, q_0]$, $[q_0, X, q_1]$, $[q_1, X, q_0]$, $[q_1, X, q_1]$, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$ }

T = { 0, 1}

Na construção das regras de produção inicia-se com as produções de S, acrescentando a P só as produções dos não-terminais que aparecem do lado direito de algum regra:

P:	S	→	$[q_0, Z_0, q_0]\ [q_0, Z_0, q_1]$
	$[q_0, Z_0, q_0]$	→	$0\ [q_0, X, q_0]\ [q_0, Z_0, q_0]$
		|	$0\ [q_0, X, q_1]\ [q_1, Z_0, q_0]$

$[q_0, Z_0, q_1] \rightarrow 0\ [q_0, X, q_0]\ [q_0, X, q_1]$
$\quad\quad\quad\quad\quad\quad\ \ |\ \ 0\ [q_0, X, q_1]\ [q_1, X, q_1]$
$[q_0, X, q_0] \rightarrow 0\ [q_0, X, q_0]\ [q_0, X, q_0]$
$\quad\quad\quad\quad\quad\quad\ \ |\ \ 0\ [q_0, X, q_1]\ [q_1, X, q_0]$
$[q_0, X, q_1] \rightarrow 0\ [q_0, X, q_0]\ [q_0, X, q_1]$
$\quad\quad\quad\quad\quad\quad\ \ |\ \ 0\ [q_0, X, q_1]\ [q_1, X, q_1]$
$\quad\quad\quad\quad\quad\quad\ \ |\ \ 1$
$[q_1, Z_0, q_1] \rightarrow \varepsilon$
$[q_1, X, q_1] \rightarrow 1 \mid \varepsilon$

Como não existem produções para os não-terminais $[q_0, X, q_0]$ e $[q_1, Z_0, q_0]$, as regras que os apresentam devem ser removidas. Simplificando, chegamos ao seguinte conjunto de regras P:

P: $S \rightarrow [q_0, Z_0, q_1]$
$[q_0, Z_0, q_1] \rightarrow 0\ [q_0, X, q_1]\ [q_1, Z_0, q_1]$
$[q_0, X, q_1] \rightarrow 0\ [q_0, X, q_1]\ [q_1, X, q_1]$
$[q_1, Z_0, q_1] \rightarrow \varepsilon$
$[q_0, X, q_1] \rightarrow 1$
$[q_1, X, q_1] \rightarrow 1 \mid \varepsilon$

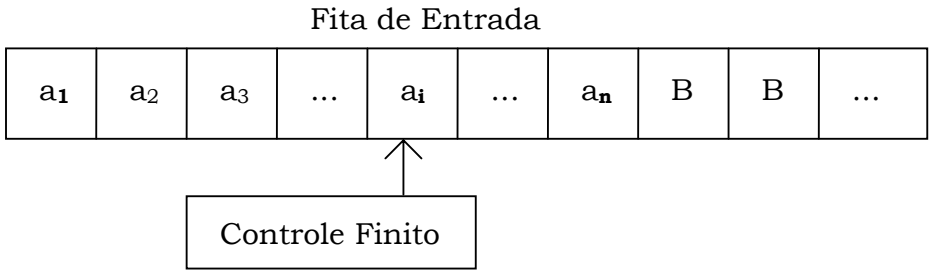
6 Máquina de Turing

As linguagens chamadas de linguagens irrestritas, podem ser reconhecidas por Máquinas de Turing.

As máquinas de Turing foram propostas como modelos matemáticas para representar procedimentos. Qualquer cômputo que possa ser descrito por uma máquina de Turing, poderá ser realizado mecanicamente. De outra parte, qualquer cômputo que possa ser realizado por um computador digital, poderá ser descrito por uma máquina de Turing.

Tese de Church - qualquer processo que possa ser chamado de procedimento, pode ser realizado por uma máquina de Turing.

6.1 Modelo Básico da Máquina de Turing



A máquina de Turing possui uma fita de entrada, onde é colocada a sentença a ser reconhecida, posicionada nas células mais à esquerda (a_1 ... a_n). Esta fita é limitada à esquerda, mas é infinita à direita. O restante das células da fita de entrada é preenchido com caracteres *brancos* (B).

O movimento da máquina de Turing depende do símbolo lido e do estado do controle finito. Dependendo destes elementos, a máquina pode:

- trocar de estado
- gravar um símbolo diferente de branco sobre o símbolo lido na fita, substituindo o que estava gravado anteriormente
- mover o cabeçote de leitura uma célula para a esquerda ou para a direita sobre a fita de entrada.

6.2 Definição de Máquina de Turing

Uma máquina de Turing (T) é definida por

$T = (K, \Sigma, \Gamma, \delta, q_0, F)$, onde:

K = conjunto finito de estados

Γ = conjunto finito de símbolos da fita, incluindo o símbolo *branco*

Σ = conjunto de símbolos de entrada, que é um subconjunto de Γ, não incluindo o branco

δ = função de movimento, que pode ser indefinida para alguns argumentos, sendo definida por

$\delta : K \times \Gamma \rightarrow K \times (\Gamma - B) \times (L, R)$

onde L = movimento do cabeçote para a esquerda e
R = para a direita

q_0 = estado inicial, $q_0 \in K$

F = conjunto finito de estados finais, $F \subseteq K$

Configuração da Máquina de Turing

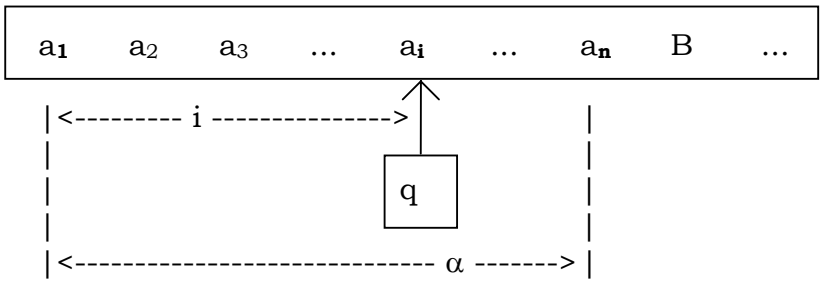
A configuração de uma máquina de Turing é representada por

(q, α, i) onde

q = estado atual da máquina, $q \in K$

α = porção não branca da fita, $\alpha \in (\Gamma - B)^*$

i = distância do cabeçote ao limite esquerdo da fita.



Descrição Instantânea da Máquina de Turing

É definida por $\alpha_1 q \alpha_2$ onde

q = estado corrente da máquina, $q \in K$

$\alpha_1 \alpha_2$ = sequência de caracteres em Γ *, conteúdo da fita de entrada à esquerda do cabeçote, ou até o último símbolo diferente de B.

Assume-se que o cabeçote esteja apontando para o símbolo mais à esquerda de α_2. Se $\alpha = \varepsilon$, o cabeçote está apontando para B. O símbolo B pode ocorrer em $\alpha_1 \alpha_2$.

Movimentos da Máquina de Turing

Seja (q, A_1 A_2 ... A_n, i) uma configuração da máquina de Turing T, onde 1 <= i <= n+1.

Se 1 <= i <= n e δ(q, A_i) = (p, A, R) então
(q, A_1 A_2 ... A_n, i) $\vdash$ (p, A_1 A_2 ... A_{i-1} A A_{i+1} ... A_n, i+1)

Se 2 <= i <= n e δ(q, A_i) = (p, A, L) então
(q, A_1 A_2 ... A_n, i) $\vdash$ (p, A_1 A_2 ... A_{i-1} A A_{i+1} ... A_n, i-1)

Se i = n+1 e δ(q, B) = (p, A, R) então
(q, A_1 A_2 ... A_n, i+1) $\vdash$ (p, A_1 A_2 ... A_n A, i+2)

Se i = n+1 e δ(q, B) = (p, A, L) então
(q, A_1 A_2 ... A_n, i+1) $\vdash$ (p, A_1 A_2 ... A_n, n)

Linguagem aceita pela Máquina de Turing

É o conjunto de palavras em Σ* que causa a máquina de Turing a entrar num estado final, tendo partido do estado inicial, com o cabeçote de leitura sobre o primeiro símbolo de estrada.

Assim, a linguagem aceita por T = (K, Σ, Γ, δ, q_0, F) é

L(T) = { w | w $\in \Sigma$ e (q_0, w, 1) $\vdash$* (q, α, i) para algum q $\in$ F, $\alpha \in \Gamma$ * e um inteiro i }

Parada da Máquina de Turing

Se uma máquina de Turing T aceita a linguagem L, então T pára para as palavras de L. Entretanto, para palavras que não pertencem à linguagem L e que, portanto, não são aceitas por T, é possível que T não pare.

Exemplo: Definir uma máquina de Turing que reconhece a LLC abaixo

$$L = \{ 0n\ 1n \mid n > 0\ 1 \}$$

e representar, através de configurações sucessivas, o reconhecimento da sentença 000111.

Solução:

$T = (K, \Sigma, \Gamma, \delta, q_0, F)$
$K = \{ q_0, q_1, q_2, q_3, q_4, q_5 \}$
$\Sigma = \{ 0, 1 \}$
$\Gamma = \{ 0, 1, B, X, Y \}$
$F = \{ q_5 \}$
$\delta = \{ \delta(q_0, 0) = (q_1, X, R)$
$\delta(q_1, 0) = (q_1, 0, R)$
$\delta(q_1, 1) = (q_2, Y, L)$
$\delta(q_2, Y) = (q_2, Y, L)$
$\delta(q_2, X) = (q_3, X, R)$
$\delta(q_2, 0) = (q_4, 0, L)$
$\delta(q_4, 0) = (q_4, 0, L)$
$\delta(q_4, X) = (q_0, X, R)$
$\delta(q_3, Y) = (q_3, Y, R)$
$\delta(q_3, B) = (q_5, Y, R)$
$\delta(q_1, Y) = (q_1, Y, R)$

$(q_0, 000111, 1) \vdash (q_1, X00111, 2) \vdash (q_1, X00111, 3) \vdash$

$(q_1, X00111, 4) \vdash (q_2, X00Y11, 3) \vdash (q_4, X00Y11, 2) \vdash$

$(q_4, X00Y11, 1) \vdash (q_0, X00Y11, 2) \vdash (q_1, X00Y11, 3) \vdash$

$(q_3, XXXYYY, 6) \vdash (q_3, XXXYYY, 7) \vdash (q_5, XXXYYY, 8)$

6.3 Relação entre Máquina de Turing e Gramática Irrestrita

Teorema: Se a linguagem L é gerada por uma gramática irrestrita, então L pode ser reconhecida por uma Máquina de Turing

Teorema: Se uma linguagem L é reconhecida por uma Máquina de Turing, então L é gerada por uma gramática irrestrita.

Bibliografia

HOPCROFT, J.E. & ULLMAN, J.D. *Formal Languages and Their Relation to Automata*. Addison-Welley Publishing Company, Reading, Massachussets, 1969.

HOPCROFT, J.E. & ULLMAN, J.D. *Introduction to Automata Theory, Languages, and Computation*. Addison-Welley Publishing Company, Reading, Massachussets, 1979.

MENEZES, PAULO FERNANDO BLAUTH. *Linguagens Formais e Autômatos.* Editora Sagra-Luzzatto, Porto Alegre, 1997.

PRICE, ANA MARIA DE ALENCAR & EDELWEISS, NINA. *Introdução As Linguagens Formais*, PORTO ALEGRE, 1989. 60 P.

RAYWARD-SMITH, V.J. *A First Course in Formal Language Theory*. Blackwell Scientific Publications, Oxford, 1983.