

UNIVERSIDADE PRESBITERIANA MACKENZIE

NOTAS DE AULA

**Elementos de
LINGUAGENS FORMAIS E AUTÔMATOS**

([©]Distribuição restrita: para uso próprio dos alunos.)

Pedro Paulo Balbi de Oliveira

pedrob@mackenzie.br
professor.mackenzie.br/pedrob

27 de fevereiro de 2024

UNIDADE 1

REVISÃO DE CONJUNTOS

1.1 DEFINIÇÕES GERAIS

Def. 1.1: CONJUNTO - uma coleção qualquer de elementos.

$a \in A$ [a pertence a A]

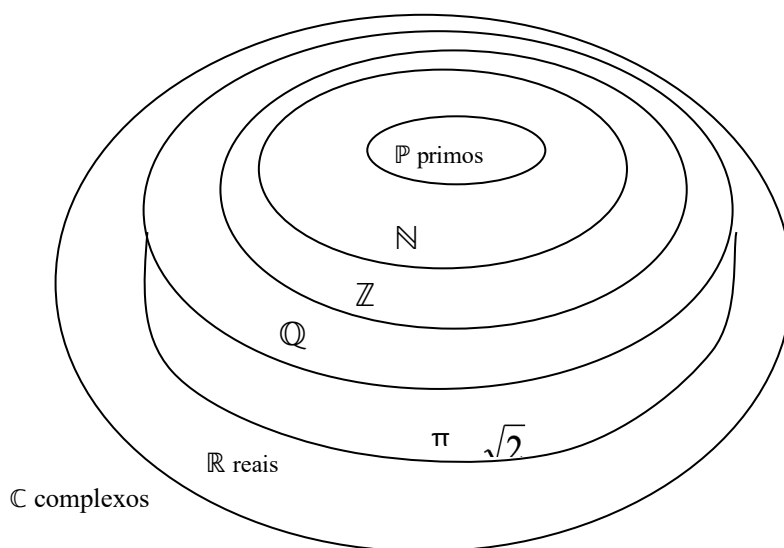
$a \notin A$ [a não pertence a A]

$A = \{a_1, a_2, \dots, a_n\} \Leftrightarrow a_1 \in A, a_2 \in A, \dots, a_n \in A$

Ex.: $\mathbb{N} = \{1, 2, 3, \dots\}$ NATURAIS

$\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ INTEIROS

\mathbb{Q} = números da forma i/j , $i, j \in \mathbb{Z}, j \neq 0$ RACIONAIS



Def. 1.2: CARDINALIDADE: Número de elementos do conjunto ($\#A$)

Def. 1.3: CONJUNTO VAZIO: Conjunto que não tem elementos (\emptyset)
 $\emptyset \equiv \{ \}$

Obs. 1: $\#\emptyset = 0$

Obs. 2: Especificação de conjuntos

$$\begin{aligned} x = \{-3, -2, -1, 0, 1, 2, 3\} &= \{x \mid x \in \mathbb{Z}, -3 \leq x \leq 3\} \\ &= \{x \in \mathbb{Z} \mid -3 \leq x \leq 3\} \end{aligned}$$

Obs. 3: $\#\mathbb{R} \rightarrow \infty$

Def. 1.4: IGUALDADE

$$A = B \Leftrightarrow \forall x \in A \Leftrightarrow \forall x \in B$$

Def. 1.5: SUBCONJUNTO

$A \subseteq B$, se $\forall a \in A \Rightarrow a \in B$
[caso contrário: $A \not\subseteq B$]



Obs. 1: $A \subset B$ A está contido em B
 $B \supset A$ B contém A

Obs. 2: SUBCONJUNTO PRÓPRIO: $A \subset B$ (A é subconjunto de B, mas $A \neq B$)

Obs. 3: $A = B \Leftrightarrow A \subset B$ e $B \subset A$

Obs. 4: $\forall A \Rightarrow \begin{cases} \emptyset \subset A \\ A \subset A \end{cases}$

Def. 1.6: CONJUNTO POTÊNCIA (Power Set)

Conjunto de todos os subconjuntos de um conjunto A. Representação: 2^A .

Ex.: $A = \{1, 2, 3\}$
 $2^A = \{ \emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\} \}$

Obs.: $A = \{a_1, a_2, \dots, a_n\}$
 $\#2^A = 2^n$

Def. 1.7: COMPLEMENTO

$\bar{A} = \{u \mid u \notin A\}$
 OBS: $u \in U$ (conjunto universo)

Def. 1.8: DIFERENÇA RELATIVA

$$A - B = \{a \mid a \in A \wedge a \notin B\} = A \setminus B$$

Def. 1.9: UNIÃO

$$A \cup B = \{u \mid u \in A \vee u \in B\}$$

Def. 1.10: INTERSECÇÃO

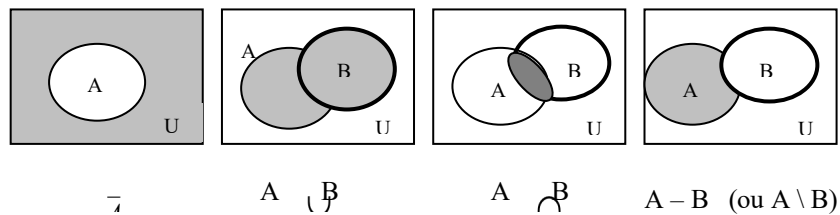
$$A \cap B = \{u \mid u \in A \wedge u \in B\}$$

Def. 1.11: CONJUNTO DISJUNTOS

$$A \cap B = \emptyset \quad (A \text{ e } B \text{ são disjuntos})$$

Def. 1.12: DIAGRAMA DE VENN

U = conjunto universo



OBS: Leis de Morgan

$$\overline{A \cup B} = \bar{A} \cap \bar{B}$$

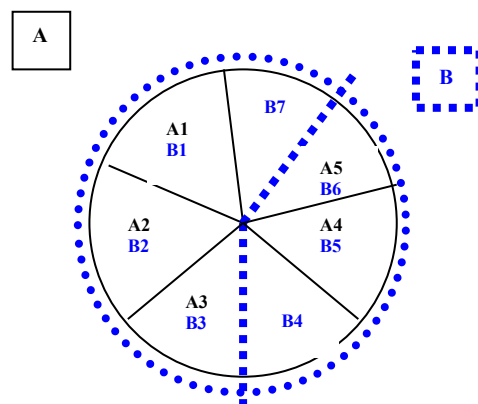
$$\overline{A \cap B} = \bar{A} \cup \bar{B}$$

Def. 1.13: PARTIÇÃO

$\pi = \{A_i\} \mid i \in K$, K é um conjunto de índices arbitrários

π é uma partição de A se todo elemento de A está exatamente em dos A_i .

Obs.: Os A_i são os blocos da partição



$$\pi_A = \{A_1, A_2, A_3, A_4, A_5\}$$

$$\pi_B = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7\}$$

$$\pi_B \text{ refina } \pi_A \Leftrightarrow \text{ todos } B_j \subset A_i$$

Def. 1.14: R – TUPLA

Sequência ordenada escrita na forma (a_1, a_2, \dots, a_r)

O i -ésimo termo \Rightarrow i -ésima coordenada

Se $r = 2 \Rightarrow$ par ordenado

Def. 1.15: PRODUTO CARTESIANO

A_1, A_2, \dots, A_r : conjunto qualquer

Produto cartesiano: todas as r -tuplas (a_1, a_2, \dots, a_r) tal que $a_1 \in A_1, a_2 \in A_2, \dots, a_r \in A_r$

Representação: $A_1 \times A_2 \times \dots \times A_r$

Se os A_i são idênticos: $A_1 \times A_2 \times \dots \times A_r = A^r$

Ex.: $A = \{0, 1\}$ e $B = \{1, 2, 3\}$

$$A \times B = \{ (0,1), (0,2), (0,3), (1,1), (1,2), (1,3) \}$$

$$B \times A = \{ (1,0), (1,1), (2,0), (2,1), (3,0), (3,1) \}$$

$$A \times A = A^2 = \{ (0,0), (0,1), (1,1), (1,0) \}$$

1.2 RELAÇÕES

Def. 1.16: RELAÇÃO

\forall subconjunto do produto cartesiano $A_1 \times A_2 \times \dots \times A_r$

RELAÇÃO BINÁRIA

Quando $r=2$, tem-se os subconjuntos de $A_1 \times A_2 = \{ (x,y) \mid x \in A_1 \text{ e } y \in A_2 \}$

Notação:

- R é uma relação de A em $B \Rightarrow (a,b) \in R$
- a é relacionado com $b \Rightarrow a R b$
- se $(x, y) \in R$: $x R y$ ou $y = R(x)$

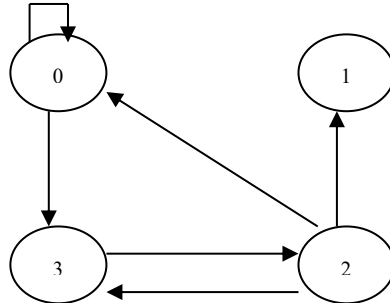
Ex.: $A = \{2, 3, 4\}$ e $B = \{2, 3, 4, 5, 6\}$

Seja R de A em B : $a R b \Leftrightarrow a$ divide b , ou seja $\text{resto}(b/a) = 0$.

$$R = \{ (2, 2), (2,4), (2,6), (3,3), (3,6), (4,4) \}$$

Obs.: Representação de uma relação binária como grafo:

$$R = \{ (0,0), (0,3), (3,2), (2,3), (2,0), (2,1) \}$$



Def. 1.17: DOMÍNIO

$\{a \in A \mid a R b \text{ existe, para algum } b\}$

Def. 1.18: CONTRADOMÍNIO

$\{b \in B \mid a R b \text{ existe, para algum } a\}$

Ex.: $R = \{ (2,2), (2,4), (2,6), (3,3), (3,6), (4,4) \}$

Domínio = $\{2, 3, 4\}$ [elementos de A usados em R]

Contradomínio = $\{2, 3, 4, 6\}$ [elementos de B usados em R]

Def. 1.19: RELAÇÃO INVERSA

\check{R} : se R é de A em B, então \check{R} é de B em A, tal que $b \check{R} a \Leftrightarrow a R b$

Ou seja: os pares de \check{R} são os pares de R em ordem inversa

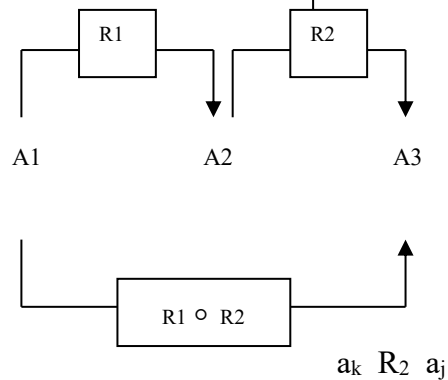
Ex.: A $\{2, 3, 4\}$

B $\{2, 3, 4, 5, 6\}$ seja $\check{R} = b \check{R} a \Leftrightarrow a \text{ divide } b$

$R = \{(2,2), (4,2), (6,2), (3,3), (6,3), (4,4)\}$

Def. 1.20: COMPOSIÇÃO DE RELAÇÕES

$$\left. \begin{array}{l} R_1 \text{ de } A_1 \text{ em } A_2 \\ R_2 \text{ de } A_2 \text{ em } A_3 \end{array} \right\} \begin{array}{l} R_1 \circ R_2 \text{ de } A_1 \text{ em } A_3, \text{ tal que} \\ a_i (R_1 \circ R_2) a_j \text{ [ou } R_1 \circ R_2 (a_i, a_j)] \\ \text{sse } a_k \in A_2 \Rightarrow \begin{cases} a_i R_1 a_k \\ a_k R_2 a_j \end{cases} \end{array}$$



Ex.: $A_1 = \{1, 2, 3, 4\}$
 $A_2 = \{2, 3, 4\}$
 $A_3 = \{1, 2, 3\}$

$$R_1 = \{ (a_i, a_k) \mid a_i + a_k = 6 \} = \{ (2,4), (3,3), (4,2) \}$$

$$R_2 = \{ (a_k, a_j) \mid a_k - a_j = 1 \} = \{ (3,2), (4,3), (2,1) \}$$

$$R_1 \circ R_2 = \{ (2,3), (3,2), (4,1) \} \Rightarrow \{ (a_i, a_j) \mid a_i + a_j = 5 \}$$

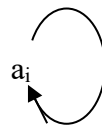
$$\text{Pois: } a_i + a_k = 6 \Rightarrow a_i + (1 + a_j) = 6 \\ a_i + a_j = 5$$

$$R_1 \circ R_2 = \begin{array}{c} \begin{array}{c} \overbrace{\begin{array}{ccccc} & & R_1 & & \\ & A_2 & 2 & 3 & 4 \end{array}} \\ A_1 \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{array} \quad \begin{array}{c} \overbrace{\begin{array}{ccccc} & & R_2 & & \\ & A_3 & 1 & 2 & 3 \end{array}} \\ A_2 \\ \begin{array}{c} 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \\ = \begin{array}{c} \overbrace{\begin{array}{ccccc} & & R_1 \circ R_2 & & \\ & A_2 & 1 & 2 & 3 \end{array}} \\ A_1 \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \end{array} \end{array}$$

Def. 1.21: RELAÇÃO REFLEXIVA

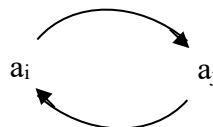
$$\forall a_i \in A, a_i R a_i$$

Ex : Divide.

**Def. 1.22: RELAÇÃO SIMÉTRICA**

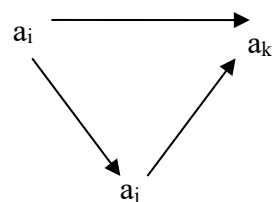
$$\forall a_i, a_j \in A, a_i R a_j \Rightarrow a_j R a_i$$

Ex : Ser irmão de.

**Def. 1.23: RELAÇÃO TRANSITIVA**

$$\forall a_i, a_j, a_k \in A, a_i R a_j \wedge a_j R a_k \Rightarrow a_i R a_k$$

Ex : Maior do que.

**Def. 1.24: RELAÇÃO DE ORDEM TOTAL**

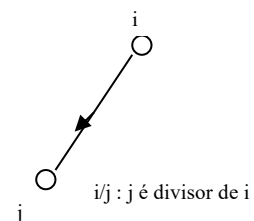
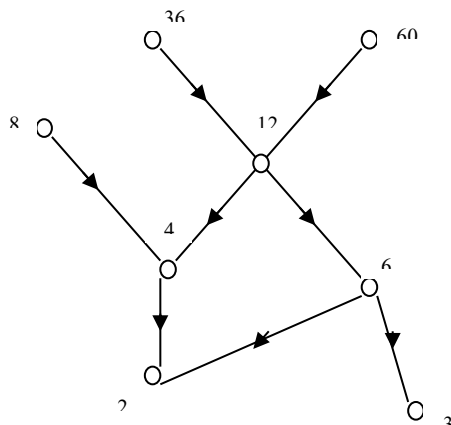
$$\text{Relação tal que } \forall a_i, a_j \in A \Rightarrow a_i R a_j \vee a_j R a_i$$

Quando isso não acontece \Rightarrow RELAÇÃO DE ORDEM PARCIAL

Ex. 1: \mathbb{R} é totalmente ordenado pela relação \leq .

Ex. 2: A relação / definida sobre um conjunto de inteiros, onde $i/j \Leftrightarrow j$ um divisor de i , é uma R.O.P.

$$A = \{2, 3, 4, 6, 8, 12, 36, 60\}$$



1.3 RELAÇÕES DE EQUIVALÊNCIA

Def. 1.25: Uma relação binária R sobre S é uma RELAÇÃO DE EQUIVALÊNCIA se:

1. R é reflexiva
2. R é simétrica
3. R é transitiva.

Exemplos:

1) Seja a relação binária R_i definida sobre o conjunto \mathcal{H} dos seres humanos, tal que

$$i R_i j \Leftrightarrow i \text{ é irmão de } j.$$

R_i não é uma relação de equivalência. Por que?

2) Seja R de A em B : $a R b \Leftrightarrow a$ divide b .

R não é uma relação de equivalência. Por que?

TEOREMA 1.1:

Se R é uma relação de equivalência sobre S então é possível dividir S em K subconjuntos distintos ($K > 1$), chamados de CLASSES DE EQUIVALÊNCIA, tais que:

$$a R b \Leftrightarrow a \text{ e } b \text{ pertencem ao mesmo conjunto.}$$

Em outras palavras: uma relação de equivalência R sobre S induz uma partição de S .

Exemplo:

Seja R_s definida sobre o conjunto \mathcal{H} dos seres humanos, tal que:

$$i R_s j \Leftrightarrow i \text{ é do mesmo sexo que } j$$

R_s é uma relação de equivalência?

Se sim, represente a partição Π_s induzida por R_s no conjunto \mathcal{H} .

Def. 1.26: O índice de uma relação de equivalência R , representado por $i(R)$, é o número de classes de equivalência de R .

Def. 1.27: Sejam R_1 e R_2 relações de equivalência:

$$R_1 \text{ refina } R_2 \text{ se } R_1 \subseteq R_2 \quad (\text{isto é, } x R_1 y \Rightarrow x R_2 y)$$

Obs.: se R_1 refina R_2 , então: $i(R_1) \geq i(R_2)$

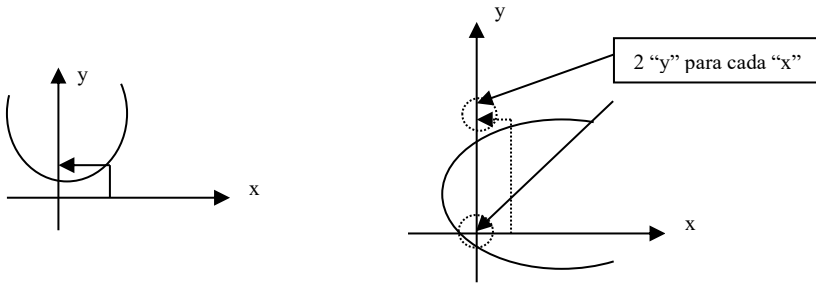
1.4 FUNÇÕES

Def. 1. 25: FUNÇÃO

É uma regra que designa para cada $a_i \in A$, um único $b_i \in B$
 $f: A \rightarrow B$ (Domínio \rightarrow Contradomínio)

Notação: $(x,y) \in f$
 $y = f(x)$
 $x \mapsto y$

Ex.: $f: \mathbb{R} \rightarrow \mathbb{R}$, $y = f(x) = x + 1$
 Se $x \geq 0 \Rightarrow y \geq 1$
 (Imagem)



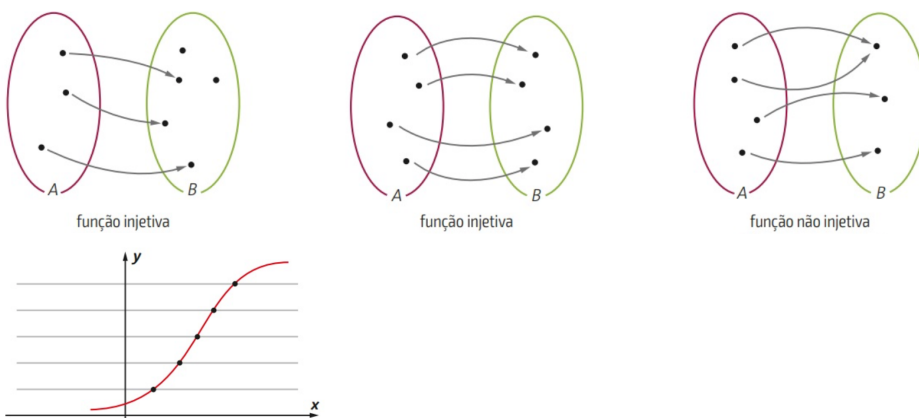
Função de r variáveis:

$f(a_1, a_2, \dots, a_r)$
 $f: A_1 \times A_2 \times \dots \times A_r \rightarrow B$

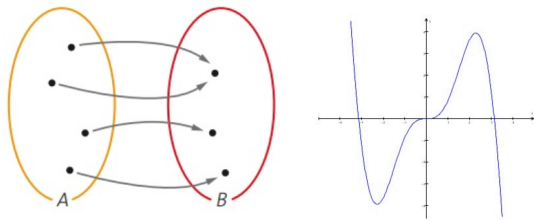
Com $r=2$: $z = x + y$, $x \in \mathbb{R}, y \in \mathbb{R}$
 (função do tipo $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ ou $\mathbb{R}^2 \rightarrow \mathbb{R}$)

Def. 1.26: INJEÇÃO

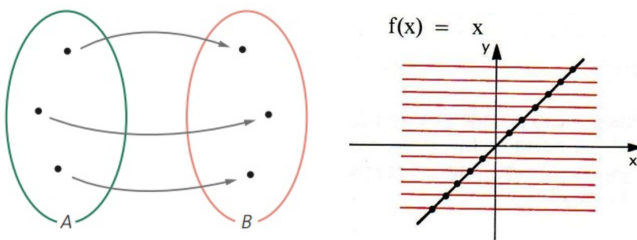
Para quaisquer 2 elementos de A correspondem 2 elementos distintos em B



Def. 1.27: SOBREJEÇÃO
Esgota B



Def. 1.28: BIJEÇÃO
Acontece injeção e sobrejeção.

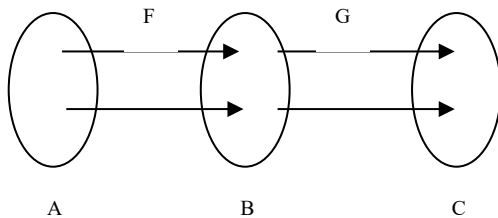


OBS: Somente a bijeção aceita inversa
 f^{-1} é inversa de f .

Ex.: $y = ax + b \Rightarrow$ Inversa: $x = (y - b)/a$

(Fonte dos diagramas: www.todoestudo.com.br)

Def. 1.29: COMPOSIÇÃO DE FUNÇÕES



$$h = f \circ g = g(f)$$

$$\left. \begin{array}{l} f(x) = \text{sen}(x) \\ g(x') = e^{-x'} \end{array} \right\} h(x) = e^{-\text{sen}(x)}$$

UNIDADE 2

LINGUAGENS, GRAMÁTICAS E RECONHECEDORES

2.1 ALFABETOS E LINGUAGENS

Def. 2.1: Um **alfabeto** (ou **vocabulário**) Σ é um conjunto finito, não vazio de símbolos.

Def. 2.2: Uma **palavra** (ou **cadeia**, ou **string**) sobre um alfabeto Σ é uma seqüência finita de símbolos de Σ .

$$x \text{ é uma cadeia} \Rightarrow x = a_1 a_2 \dots a_n, \quad n \geq 0 \\ a_i \in \Sigma, \quad i = 1, 2, \dots, n$$

Quando $n = 0 \Rightarrow \epsilon$ (cadeia vazia)

Def. 2.3: Comprimento de uma cadeia x sobre um alfabeto Σ (representado por $|x|$) é o número de ocorrências de símbolos de Σ em x .

$$\text{Ex.: } \Sigma = \{0, 1\} \Rightarrow \begin{cases} |011| = 3 & (\text{n.º de elementos na cadeia}) \\ |\epsilon| = 0 \end{cases}$$

Def. 2.4: Sejam x e y duas cadeias sobre Σ a concatenação de x e y é definida como a cadeia xy .

$$\text{Ex.: } \Sigma = \{0, 1\}, \quad x = 010, \quad y = 10 \Rightarrow \begin{cases} xy = 01010 \\ yx = 10010 \end{cases}$$

OBS.1: A cadeia vazia é o elemento neutro da operação de concatenação $\Rightarrow \epsilon\epsilon x = x\epsilon = \epsilon x = x$

OBS.2: Concatenação sucessiva: $(10)^n = \underbrace{101010\dots 10}_n$
 n ocorrências concatenadas da cadeia 10

Def. 2.5: Sejam X e Y conjuntos de cadeias sobre Σ . O produto de X e Y é definido por:

$$XY = \{xy \mid x \in X, y \in Y\}$$

$$\text{Notação: } X^0 = \{\epsilon\} \\ X^{i+1} = X^i X, \quad i \geq 0$$

Operações com um conjunto X de cadeias:

Fechamento de Kleene (*Kleene's closure*): X^*

Fechamento Positivo (*positive closure*): X^+

$X^+ = \cup X^i, i \geq 1 \quad \therefore$ todas as concatenações possíveis

$X^* = \cup X^i, i \geq 0 \quad \therefore$ todas as concatenações possíveis, mais a cadeia vazia.

Exemplo: $\Sigma = \{0, 1\}$

$\Sigma^* = \{ \epsilon, 0, 1, 01, 10, 00, 11, \dots \}$

$\Sigma^+ = \{ 0, 1, 01, 10, 00, 11, \dots \}$

Exercício:

Sejam $\Sigma = \{ 0, 1 \}$, $N = \{ A, B \}$ e $V = \Sigma \cup N$. Para cada expressão abaixo, dizer se ela é falsa ou verdadeira:

$0 \in V^*$	$AA \in V^*$	$\#V^* \rightarrow \infty$	$AA \in V^+$
$\epsilon \in V^*$	$B1\epsilon 0B \in V^*$	$B1\epsilon 0B \in V^+$	$01 \in V^+NV^+$
$A \in V^*NV^*$	$\epsilon \in V^*NV^*$	$01 \in V^*NV^*$	$0A1 \in V^+NV^+$
$AA \in V^*NV^*$	$AA \in V^+NV^+$	$01A \in V^+NV^+$	$AAA \in V^+NV^+$

Def. 2.6: Uma linguagem L é um conjunto qualquer de cadeias sobre um alfabeto Σ , ou seja: $L \subset \Sigma^*$.

Como representar L?

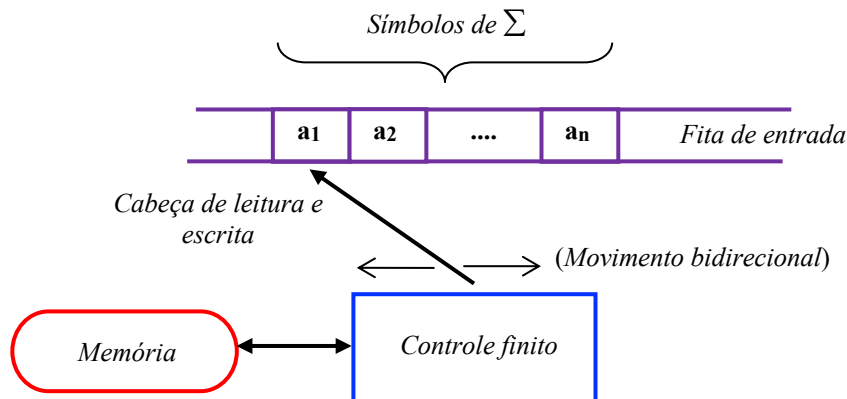
- Se L é finito, basta listar todas as cadeias.
- Se L é infinito, existem 2 sistemas principais para representação de linguagens:
 1. O sistema gerador: **Gramática**
 2. O sistema reconhecedor: **Autômato**

2.2 RECONHECEDORES E GRAMÁTICAS

Um reconhecedor de L é a representação de um procedimento que, quando apresentado a uma cadeia qualquer:

- **pára** e responde **sim**, após um número finito de passos, caso a cadeia pertença a L ; ou
- **pára** e responde **não**, caso a cadeia não pertença a L ; ou
- **não pára**, caso a cadeia não pertença a L .

Simbolicamente:



Uma configuração do reconhecimento é uma descrição:

- (a) do estado do controle finito
- (b) do conteúdo da fita de entrada e da posição da cabeça
- (c) do conteúdo da memória.

Um reconhecedor aceita (ou reconhece) uma cadeia w se:

1. partindo de uma configuração inicial;
2. faz uma sequência finita de movimentos; e
3. termina em uma configuração final.

A linguagem aceita por um reconhecedor R é:

$$L(R) = \{w \in \Sigma^* \mid R \text{ aceita } w\}$$

Def. 2.7: Uma gramática é uma 4-upla $G = (\Sigma, N, S, P)$ onde:

Σ é um conjunto finito não vazio de símbolos, chamados de terminais,

N é um conjunto finito não vazio de símbolos, chamados de não-terminais, com $\Sigma \cap N = \emptyset$

$S \in N$, é o símbolo (não terminal) inicial

P é um conjunto de regras (de produção) da forma $\alpha \rightarrow \beta$, onde:

$$\begin{aligned}\alpha &\in (N \cup \Sigma)^* N (N \cup \Sigma)^* \\ \beta &\in (N \cup \Sigma)^*\end{aligned}$$

Ex.: $G = (\Sigma, N, S, P)$ onde:

$N = \{A, S\}$

S : símbolo (não-terminal) inicial

$\Sigma = \{a, b\}$

$P = \{ S \rightarrow ab ,$

$S \rightarrow aASb ,$

$S \rightarrow bSb ,$

$AS \rightarrow bSb ,$

$A \rightarrow \epsilon ,$

$aASAb \rightarrow aa \}$

Notação:

Letra maiúscula: não-terminais

Letra minúscula: terminais

Exemplo de cadeias geradas por essa gramática:

1. $S \rightarrow ab$

2. $S \rightarrow aASb \rightarrow abSbb \rightarrow ababbb$

Ou seja, a linguagem gerada $L(G) = \{ ab , ababbb , \dots \}$

OBS:

- A linguagem gerada por G pode ser obtida pela construção da Árvore de Derivação correspondente.
- Se L é gerada por G , qualquer gramática G' obtida a partir de G passa a gerar L' , possivelmente diferente de L .
- Ao se projetar uma gramática que gere L , deve-se tomar o cuidado de permitir a geração de todas e apenas as cadeias de L .

Exercício:

Construa uma gramática para cada uma das 3 linguagens abaixo. Lembrem-se de que a gramática tem de ser capaz de gerar todas as cadeias da linguagem, nem mais e nem menos.

$$L_1 = \{ 01^n \mid n \geq 1 \} = \{ 01, 011, 0111, \dots \}$$

$$L_2 = \{ 0^n 1^n \mid n \geq 1 \} = \{ 01, 0011, 000111, \dots \}$$

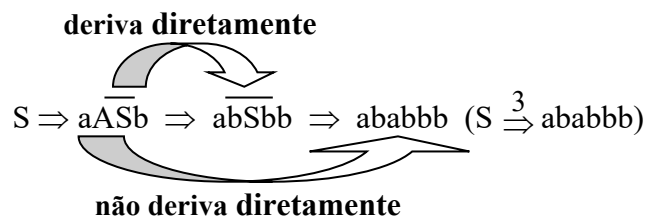
$$L_3 = \{ \text{Números inteiros decimais, com ou sem sinal (sem iniciar com zero)} \}$$

Def. 2.8: Sejam: uma gramática $G = (\Sigma, N, S, P)$, $V = N \cup \Sigma$, $\alpha' \in V^*$, e $\beta' \in V^*$.

α' deriva diretamente β' ($\alpha' \Rightarrow \beta'$) se existem $\alpha_1, \alpha_2, \alpha, \beta \in V^*$ tais que:

$$\begin{aligned}\alpha' &= \alpha_1 \alpha \alpha_2 , \\ \beta' &= \alpha_1 \beta \alpha_2 , \text{ e} \\ \alpha &\rightarrow \beta \in P\end{aligned}$$

Por exemplo, segundo G definido anteriormente:



Notação: \xRightarrow{n} deriva em n passos
 $\xRightarrow{*}$ deriva em zero ou mais passos

Def. 2.9: Sejam $G = (\Sigma, N, S, P)$ e $V = N \cup \Sigma$. O conjunto de Formas Sentenciais de G é definido por:

$$S(G) = \{ \alpha \in V^* \mid S \xRightarrow{*} \alpha \}$$

Def. 2.10: Sejam $G = (\Sigma, N, S, P)$ e $V = N \cup \Sigma$. A Linguagem Gerada (ou Representada) por G é o conjunto:

$$L(G) = \{ w \in \Sigma^* \mid S \xRightarrow{n} w \} = \Sigma^* \cap S(G)$$

Ou seja: são todas as cadeias terminais, deriváveis a partir de S .

OBS:

As formas sentenciais de uma gramática são a própria árvore de derivação associada à gramática, enquanto a linguagem gerada constitui o conjunto das folhas da árvore.

OBS: No contexto usual de linguagens de programação,

1. Σ (símbolos) \equiv palavras reservadas + variáveis definidas + símbolos numéricos + operadores + delimitadores...
2. cadeia \equiv programa sintaticamente correto
3. L (linguagem) \equiv conjunto de programas sintaticamente corretos
4. G (gramática) \equiv estrutura sintática

2.3 HIERARQUIA DE CHOMSKY

A gramática, como foi definida anteriormente, é chamada de gramática **Irrestrita** ou **Tipo-0** (ou ainda: Recursivamente Enumerável, gramática Semi-Thue; ou gramática de Estrutura de Frase).

Def. 2.11: $G = (\Sigma, N, S, P)$ com $V = N \cup \Sigma$ é gramática **Sensível ao Contexto** ou **Tipo-1**, se toda produção de P é da forma:

$$1) \alpha \underset{\text{A}}{\text{A}} \gamma \rightarrow \alpha \underset{\text{B}}{\text{B}} \gamma \quad \left\{ \begin{array}{l} A \in N \\ \alpha, \gamma \in V^* \text{ , } V = N \cup \Sigma \\ \beta \in V^+ \end{array} \right.$$

- 2) $S \rightarrow \epsilon$, sendo que se esta regra de produção ocorrer, S não pode aparecer no lado direito de qualquer outra produção, se isto puder ocasionar redução do tamanho da cadeia da direita (por exemplo, uma regra de produção como $S \rightarrow 1S$ não geraria problema).

Exemplo 1: Algumas regras de produção da primeira gramática apresentada não estão expressas na forma canônica do tipo sensível ao contexto:

$$\begin{aligned} G &= (\{a, b\}, \{A, S\}, S, P) \\ P &= \{ \\ &\quad S \rightarrow ab, \\ &\quad S \rightarrow aASb, \\ &\quad S \rightarrow bSb, \\ &\quad AS \rightarrow bSb, \\ &\quad A \rightarrow \epsilon, \\ &\quad aASAb \rightarrow aa \\ &\quad \} \end{aligned}$$

Def. 2.12: $G = (\Sigma, N, S, P)$ é gramática **Livre de Contexto** ou **Tipo-2** se toda produção de P é da forma:

$$A \rightarrow \beta, \quad \begin{array}{l} A \in N \\ \beta \in V^* \end{array}$$

Def. 2.13: $G = (\Sigma, N, S, P)$ é uma gramática **Regular** ou **Tipo-3**, se toda produção de P é da forma linear à direita (*right-linear*):

$$\begin{array}{ll} 1) & A \rightarrow wB \\ 2) & A \rightarrow w \end{array} \quad \left\{ \begin{array}{l} A, B \in N \\ w \in \Sigma^* \end{array} \right.$$

Ou da forma linear à esquerda (*left-linear*):

$$\begin{array}{ll} 3) & A \rightarrow Bw \\ 4) & A \rightarrow w \end{array} \quad \left\{ \begin{array}{l} A, B \in N \\ w \in \Sigma^* \end{array} \right.$$

Exemplo 2: $L_1 = \{ 01^n, n \geq 1 \}$ é regular, mas G_1 não evidencia o fato.

$\mathbf{G_1 = (\{0, 1\}, \{I, A\}, I, P_1)}$ $P_1 = \{$ $I \rightarrow 0A1 ,$ $A \rightarrow 1A ,$ $A \rightarrow \varepsilon$ $\}$	$\mathbf{G_2 = (\{0, 1\}, \{I\}, I, P_2)}$ $P_2 = \{$ $I \rightarrow 01 ,$ $I \rightarrow II$ $\}$
--	--

Def. 2.14: Uma linguagem L é do **Tipo-x**, se existe uma gramática **Tipo-x**, G , tal que $L = L(G)$ (isto é, a linguagem L é gerada pela gramática G).

OBS:

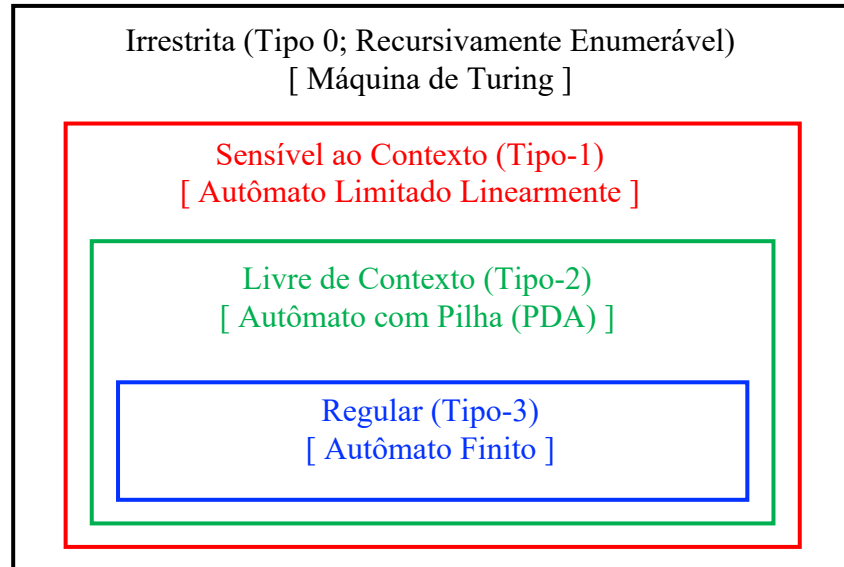
Tipos 2 e 3: Só existe 1 não-terminal do lado esquerdo (e nenhum terminal).

Tipo-1: Pode haver mais de 1 não-terminal do lado esquerdo, mas só 1 é transformado em cada regra de produção.

Tipo-0: Qualquer quantidade de não-terminais do lado esquerdo das produções,

Hierarquia de Chomsky:

Relacionamento entre Gramáticas (ou Linguagens) e seus reconhecedores correspondentes.



OBS2: Uma visão mais sintética, segundo Wolfram:

Tipo-0: Produções arbitrárias.

M.T.: Memória arbitrariamente grande.

Tipo-1: Produções da forma $\alpha \rightarrow \beta$ tal que $|\alpha| \leq |\beta|$, $\alpha \in (N \cup \Sigma)^+$, $\beta \in (N \cup \Sigma)^*$.

A.L.L.: Memória proporcional ao comprimento da cadeia de entrada.

Tipo-2: Produções da forma $A \rightarrow \alpha$ tal que $A \in N$, $\alpha \in (N \cup \Sigma)^*$, e $|\alpha|$ finito.

P.D.A.: Memória em pilha, com uma quantidade fixa de elementos disponíveis em um dado tempo.

Tipo-3: Produções da forma (unitária) $A \rightarrow sB$ ou $A \rightarrow s$ tal que $A, B \in N$, $s \in (\Sigma \cup \epsilon)$.

A.F.: Memória apenas de constantes ou de quantidades indeterminadas.

UNIDADE 3

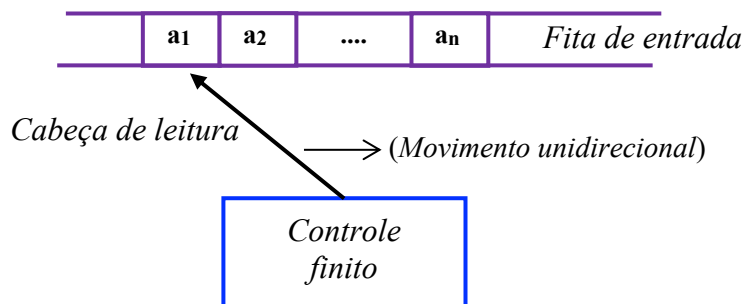
AUTÔMATOS FINITOS E LINGUAGENS REGULARES

3.1 AUTÔMATOS FINITOS

Def 3.1 : Um autômato finito é uma **5-upla** $A = (Q, \Sigma, \delta, q_0, F)$ onde:

Q	Conjunto finito não vazio (Estados)
Σ	Alfabeto (de entrada); $\Sigma \cap Q = \emptyset$
$q_0 \in Q$	Estado inicial
$F \subseteq Q$	Conjunto de estados finais
$\delta: Q \times \Sigma \rightarrow Q$	Função de transição de estados

Simbolicamente:



$\delta(q, a) = q' \Leftrightarrow$ O autômato estando no estado q e lendo o símbolo a na fita de entrada, move a cabeça leitora uma posição para a direita e vai para o estado q' .

Função δ' estendida: $\delta': Q \times \Sigma^* \rightarrow Q$

$\delta'(q, \epsilon) = q$ transição que não altera estado

$\delta'(q, xa) = \delta(\delta(q, x), a) \quad ; \quad \begin{matrix} x \in \Sigma^* \\ a \in \Sigma \end{matrix}$

$\delta'(q, x) = q' \Leftrightarrow$ O autômato estando no estado q e lendo o símbolo mais à esquerda da cadeia x , vai estar no estado q' após todos os símbolos de x terem sido lidos.

Def. 3.2:

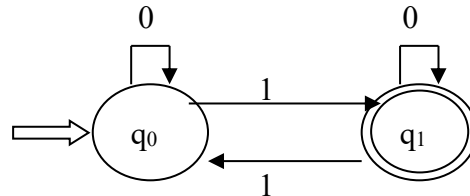
Sejam A um autômato finito e $x \in \Sigma^*$, x é aceita por A se $\delta'(q_0, x) \in F$.

Caso contrário, diz-se que a cadeia é rejeitada (ou, não aceita).

Def. 3.3:

Seja A um autômato finito. A linguagem reconhecida por A é $L(A) = \{ x \in \Sigma^* \mid \delta'(q_0, x) \in F \}$.

Exemplo de um Autômato Finito (AF):



(Diagrama de Transição de Estados)

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

$A = (\{q_0, q_1\}, \{0,1\}, \delta, q_0, \{q_1\})$

Tabela de Transição de Estados

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

Exemplos de cadeias aceitas por A: 1, 01, 01101, ...

Exemplos de cadeias rejeitadas por A: 0, 101, 101101, ...

$L(A) = \{ x \in \Sigma^* \mid x \text{ tem quantidade ímpar de } 1\text{'s} \}$

Exercícios:

Para cada uma das linguagens a seguir, todas definidas sobre o alfabeto $\Sigma = \{ a, b \}$, construa um autômato finito que a reconheça. Lembrem-se de que:

1. O autômato criado tem de ser capaz de reconhecer todas as cadeias da linguagem, nem mais e nem menos.
2. As transições de estado só podem estar associadas aos símbolos do alfabeto (portanto, não se pode ter transição via cadeia vazia).
3. De cada estado tem de sair uma (e apenas uma) transição de estado para cada um dos símbolos do alfabeto.

$$L_1 = \Sigma^+$$

$$L_2 = \{ \epsilon \}$$

$$L_3 = \Sigma^*$$

$$L_4 = \{ \}$$

$L_5 = \{ \text{Todas as cadeias em que tanto os } a\text{'s quanto os } b\text{'s ocorrem em quantidades pares} \}$

Def. 3.4:

Seja R , relação de equivalência sobre Σ^* . R é uma relação de equivalência à direita (ou, relação invariante à direita) se: $xRy \Rightarrow (\forall z \in \Sigma^*) xzRyz$

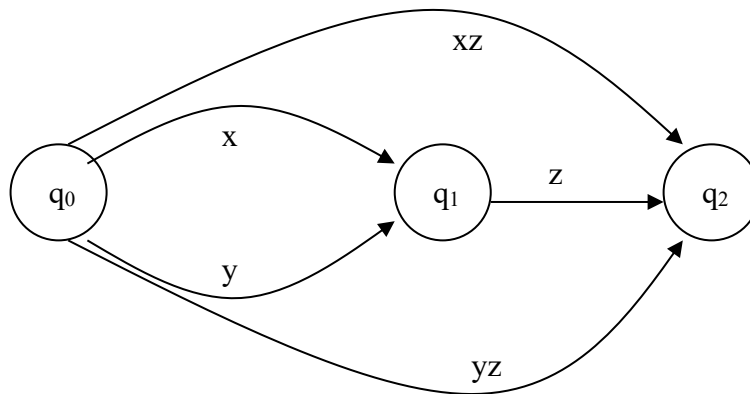
Ex: Seja R sobre Σ^* definida por:

$$xRy \Leftrightarrow \delta(q_0, x) = \delta(q_0, y)$$

$$xRy \Rightarrow \delta(q_0, x) = \delta(q_0, y) = q'$$

$$\text{Mas: } \delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(q', z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz) \\ \Rightarrow xzRyz$$

Intuitivamente: R é a resposta do autômato à cadeia.

**Def 3.5:**

Sejam R , relação de equivalência sobre Σ^* , e $L \subseteq \Sigma^*$.

R refina L se $xRy \Rightarrow (x \in L \Leftrightarrow y \in L)$. Ou seja: se R refina L , L é a união de algumas (ou todas) as classes de equivalência de R .

Def 3.6:

Seja $L \subseteq \Sigma^*$. A relação R_L é definida por:

$$xR_L y \Leftrightarrow (\forall z \in \Sigma^*) (xy \in L \Leftrightarrow yz \in L)$$

Teorema 3.1: (1) R_L é uma relação de congruência à direita.

(2) R_L refina L .

(3) Se R é uma relação de congruência à direita que refina $L \Rightarrow R \subseteq R_L$.

Ou seja: R_L é a menor (i.e., menor número de classes de equivalência) relação de congruência à direita, que refina L .

Teorema 3.2: Teorema de Myhill-Nerode

Seja $L \subseteq \Sigma^*$; são equivalentes:

1. L é regular.
2. Existe uma relação de congruência à direita R sobre Σ^* , que refina L , e que tem índice finito.
3. $i(R_L)$ é finito.

OBS.: Este teorema é muito útil para provar que certas linguagens **não são** regulares.

3.2 MINIMIZAÇÃO DE AUTÔMATOS FINITOS

OBS.: Computabilidade \times Performance:

- Computabilidade: Ser ou não ser capaz de computar.
- Performance: Qual a facilidade de computar (gasto de memória; tempo de execução; esforço de codificação; facilidade de manutenção; etc).

A. F. Mínimo: Mesma computabilidade, porém melhor performance.

Def 3.7:

Seja $A = (Q, \Sigma, \delta, q, F)$. Um estado $q \in Q$ é acessível se $\exists x \in \Sigma^*$ tal que $q = \delta(q_0, x)$.

Def 3.8:

O autômato conexo associado a $A = (Q, \Sigma, \delta, q, F)$ é definido por :

$A^c = (Q', \Sigma, \delta', q_0, F')$ onde:

$$Q' = \{ q \in Q \mid q \text{ é acessível} \}$$

$$F' = F \cap Q'$$

$$\delta' = \delta \cap (Q' \times \Sigma^* \times Q')$$

Def 3.9:

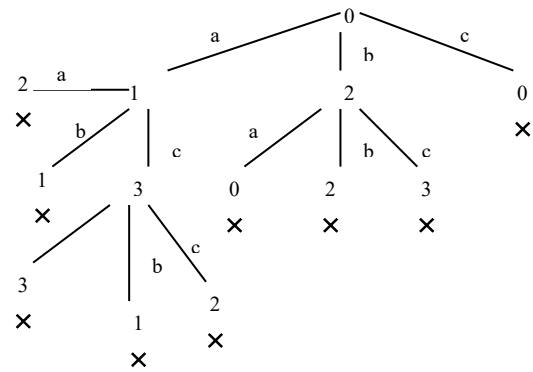
Dois autômatos A_1 e A_2 são equivalentes, se $L(A_1) = L(A_2)$.

Teorema 3.3:

Para todo autômato finito A , A e A^c são equivalentes. A determinação de estados acessíveis pode ser mecanizada, organizando-se a função δ em forma de árvore e verificando-se os estados que ocorrem.

Ex: Seja A, onde $\Sigma = \{a, b, c\}$; $Q = \{0, 1, 2, 3, 4, 5\}$; $q_0 = 0$; $F = \{3, 4\}$, e δ dado por

δ	a	b	c
0	1	2	0
1	2	1	3
2	0	2	3
3	3	1	2
4	5	1	2
5	0	4	5



Conclusão: $\begin{cases} 4, 5: & \text{Não acessíveis} \\ 0, 1, 2, 3: & \text{Acessíveis} \end{cases}$

Portanto:

$A^c = (Q', \Sigma, \delta', q_0, F')$ onde: $Q' = \{0, 1, 2, 3\}$

$F' = \{3\}$

$\Sigma = \{a, b, c\}$

$q_0 = 0$

δ' : Tabela de transição de estados

δ	a	b	c
0	1	2	0
1	2	1	3
2	0	2	3
3	3	1	2

Def. 3.10:

Dois estados q e q' são equivalentes ($q \equiv q'$) sse $(\forall x \in \Sigma^*) (\delta(q, x) \in F \Leftrightarrow \delta(q', x) \in F)$.

Consequentemente também vale que: $(\forall x \notin \Sigma^*) (\delta(q, x) \notin F \Leftrightarrow \delta(q', x) \notin F)$.

A relação \equiv é uma relação de equivalência. Portanto, ela particiona o conjunto Q (de estados). A importância disso é que ela permite identificar elementos redundantes de Q (do ponto de vista de reconhecimento de linguagem), pois, se $q \equiv q'$ (q e q' na mesma classe de equivalência), não irá fazer diferença se o autômato se encontra no estado q ou no estado q' quando uma cadeia $x \in \Sigma^*$ começar a ser processada. Logo, o autômato pode ser minimizado, escolhendo-se apenas um elemento de cada uma das classes de equivalência da relação \equiv .

Def. 3.11:

Dois estados q e q' são k -equivalentes ($q \stackrel{k}{\equiv} q'$) sse $(\forall x \in \Sigma^*, |x| \leq k) (\delta(q, x) \in F \Leftrightarrow \delta(q', x) \in F)$

OBS.:

- A relação $\stackrel{k}{\equiv}$ também é uma relação de equivalência.
- Da definição, segue que: se $q \stackrel{k}{\equiv} q'$, então $q \stackrel{k'}{\equiv} q'$ para todo $k' \leq k$.
- Portanto, π_k (partição de Q induzida por $\stackrel{k}{\equiv}$) refina $\pi_{k'}$ (ou seja, $i(\pi_k) \geq i(\pi_{k'})$, $k' \leq k$).

Teorema 3.4:

Pode-se encontrar a relação \equiv , considerando-se sucessivamente as relações $\stackrel{0}{\equiv}$, $\stackrel{1}{\equiv}$, $\stackrel{2}{\equiv}$ e assim por diante, até que $\pi_k = \pi_{k+1}$, $k \geq 0$.

Teorema 3.5:

O processo a que se refere o teorema anterior sempre pára, i.e., pode-se construir um algoritmo para construir a relação \equiv .

Ou seja, em conjunto:

$$q \stackrel{0}{\equiv} q' \Rightarrow \pi_0$$

$$\begin{array}{ccc} q \stackrel{1}{\equiv} q' & \Rightarrow & \pi_1 \\ \vdots & & \vdots \end{array}$$

$$q \stackrel{k}{\equiv} q' \Rightarrow \pi_k$$

$$q \stackrel{k+1}{\equiv} q' \Rightarrow \pi_{k+1} = \pi_k \Rightarrow q \equiv q'$$

Quando dois π_k subsequentes são iguais, fica determinada a equivalência.

OBS.:

Uma descrição do algoritmo pode encontrada na p.67 do Hopcroft & Ullman, ou no livro do P. Blauth.

3.3 EXPRESSÕES REGULARES

Def. 3.12:

Seja Σ um alfabeto. As expressões regulares sobre Σ , e os conjuntos que elas representam, são definidos, recursivamente, como:

- a) ϕ é uma expressão regular, e representa o conjunto ϕ .
- b) ϵ é uma expressão regular e representa o conjunto $\{\epsilon\}$.
- c) Se $a \in \Sigma$, então a é uma expressão regular e representa o conjunto $\{a\}$.
- d) Se r e s são expressões regulares representando, respectivamente, os conjuntos R e S , então $r+s$, rs , r^* e s^+ são expressões regulares representando, respectivamente, $R \cup S$, RS , R^* e S^+ .

OBS.: Para se poder eliminar alguns parênteses, assume-se que a prioridade das operações é:

- 1) Fechamento (de Kleene, e positivo)
- 2) Concatenação
- 3) Soma

EXPRESSÃO REGULAR	LINGUAGEM REPRESENTADA
$aa \equiv a^2$	Somente a palavra aa
$aa + b$	Somente as palavras aa e b
a^+	Todas as palavras com a 's concatenados.
ba^*	Todas as palavras que iniciam por b , seguido por zero ou mais a 's
$(a+b)^+$	Todas as palavras sobre $\{a, b\}$
$(a+b)^*$	Todas as palavras sobre $\{a, b\}$, e também a cadeia vazia
$(a+b)^*aa(a+b)^*$	Todas as palavras contendo aa como subpalavra
$a^*ba^*ba^*$	Todas as palavras contendo exatamente 2 b 's
$(a+b)^*(aa+bb)$	Todas as palavras que terminam com aa ou bb
$(a+\epsilon)(b+ba)^*$	Todas as palavras que não possuem 2 a 's consecutivos (incluindo: $\epsilon + a + b$)

Exercício:

É possível construir um autômato finito que reconheça a linguagem $L = \{ 0^n 1^n \mid n \geq 1 \}$?

3.4 AUTÔMATO FINITO NÃO-DETERMINÍSTICO

Def. 3.13:

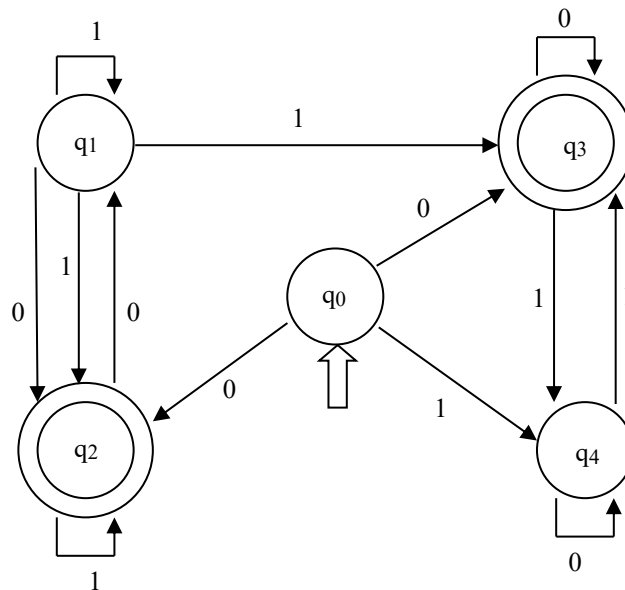
Um autômato finito não determinístico é uma 5-upla $A = (Q, \Sigma, \delta, q_0, F)$, com Q, Σ, q_0 e F definidos como no caso determinístico, e δ da forma

$$\delta: Q \times \Sigma \rightarrow 2^Q,$$

onde 2^Q é o conjunto potência de Q .

$\delta(q, a) = \{q_1, q_2, \dots, q_n\} \Leftrightarrow$ O autômato estando no estado q e tendo o símbolo a na fita de entrada, move sua cabeça leitora uma posição para a direita, transitando para cada um dos q_i estados, $i=1, \dots, n$; conceitualmente, é equivalente a imaginar que o autômato se subdivide em n cópias, cada uma das quais transita para um q_i distinto, $i=1, \dots, n$.

Exemplo:



Def. 3.14:

Seja $x \in \Sigma^*$; x é aceita pelo AFND $A = (Q, \Sigma, \delta, q_0, F)$, se $\delta(q_0, x) \cap F \neq \emptyset$. Ou seja, \exists algum estado final em $\delta(q_0, x)$.

Portanto, $L(A) = \{ x \in \Sigma^* \mid \delta(q_0, x) \cap F \neq \emptyset \}$.

Teorema 3.6:

Seja L um conjunto aceito por um AFND. Então existe um autômato finito determinístico (AFD) que aceita L .

“Prova” informal, por construção. Sejam

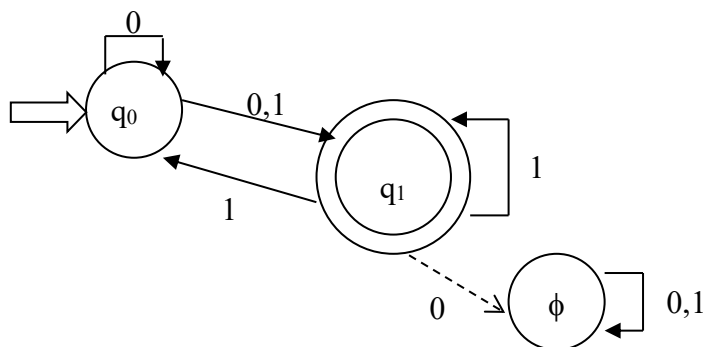
$$A = (Q, \Sigma, \delta, q_0, F) \quad \text{AFND}$$

$$A' = (Q', \Sigma, \delta', q_0', F') \quad \text{AFD}$$

tal que A' é equivalente a A . Como construir A' , a partir A ?

- 1) Cada elemento de Q' será representado por $[q_1 q_2 \dots q_i]$ onde $q_1, q_2, \dots, q_i \in Q$.
- 2) $q_0' = [q_0]$.
- 3) $Q' = 2^Q$, com a notação do passo anterior.
- 4) $F' =$ conjunto formado pelos estados de Q' que possuem pelo menos um estado em F .
- 5) $\delta'([q_1 q_2 \dots q_i], a) = [r_1 r_2 \dots r_j] \Leftrightarrow \delta(q_1, a) \cup \delta(q_2, a) \cup \dots \cup \delta(q_i, a) = \{r_1, r_2, \dots, r_j\}$.

Exemplo: Seja $A = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$, com δ dada por:



OBS:

Transições de estado não-especificadas no AFND devem levar ao estado ϕ , o qual terá, no AFD equivalente, o significado de um estado de erro, i.e., um estado que, se atingido durante o processamento de alguma cadeia de entrada, significará seu não reconhecimento.

No exemplo acima:

$$\delta(q_1, 0) = \phi$$

$$\delta(\phi, 0) = \phi$$

$$\delta(\phi, 1) = \phi$$

Determinar A' equivalente a A (i.e., ambos devem reconhecer a mesma linguagem):

$A'(Q', \{0,1\}, q_0', F', \delta')$

$$2^Q = \{ \phi, \{q_0\}, \{q_1\}, \{q_1, q_0\} \} \rightarrow Q' = \{ \phi, [q_0], [q_1], [q_0q_1] \}$$

$$q_0' = [q_0]$$

$$F' = \{ [q_1], [q_0q_1] \}$$

$\delta(\phi, 0) = \phi$	\rightarrow	$\delta'(\phi, 0) = \phi$
$\delta(\phi, 1) = \phi$	\rightarrow	$\delta'(\phi, 1) = \phi$
$\delta(q_0, 0) = \{q_0, q_1\}$	\rightarrow	$\delta'([q_0], 0) = [q_0q_1]$
$\delta(q_0, 1) = \{q_1\}$	\rightarrow	$\delta'([q_0], 1) = [q_1]$
$\delta(q_1, 0) = \phi$	\rightarrow	$\delta'([q_1], 0) = \phi$
$\delta(q_1, 1) = \{q_0, q_1\}$	\rightarrow	$\delta'([q_1], 1) = [q_0q_1]$

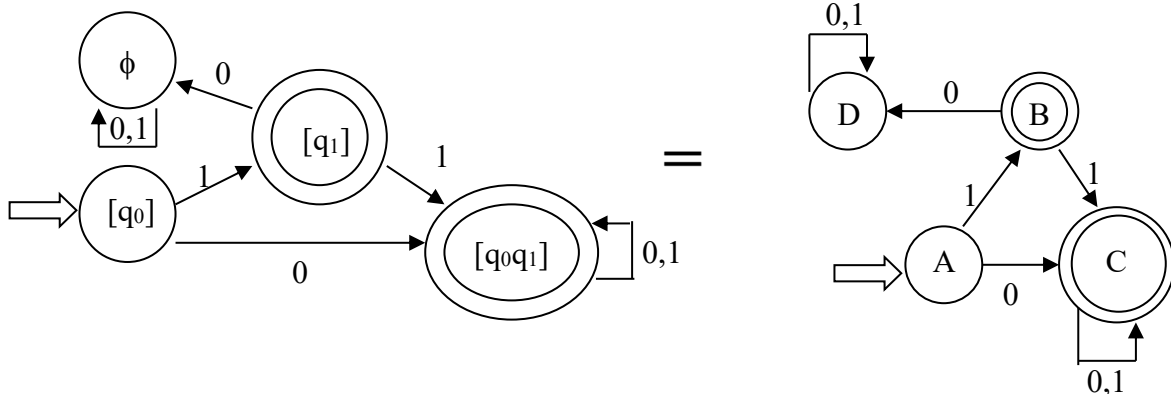
E ainda:

$$\underbrace{\delta(q_0, 0)}_{\{q_0, q_1\}} \cup \underbrace{\delta(q_1, 0)}_{\phi} = \{q_0, q_1\} \rightarrow \delta'([q_0q_1], 0) = [q_0q_1]$$

$$\underbrace{\delta(q_0, 1)}_{\{q_1\}} \cup \underbrace{\delta(q_1, 1)}_{\{q_0, q_1\}} = \{q_0, q_1\} \rightarrow \delta'([q_0q_1], 1) = [q_0q_1]$$

OBS: $\{A, B\} \cup \phi = \{A, B\}$
 $\{A, B\} \cup \{\phi\} = \{A, B, \phi\}$

AFD $A' \equiv$ AFND A



$$L(A) = L(A') = \{ 1, 11x, 0x \mid x \in \Sigma^* \} = L(A_{\min})$$

$$1 + 11(0+1)^* + 0(0+1)^*$$

Teorema 3.7:

Seja G uma gramática regular $\Rightarrow \exists$ um AFD $A \mid L(G) = L(A)$.

Teorema 3.8:

Seja A um AFD $\Rightarrow \exists$ uma gramática regular $G \mid L(G) = L(A)$.

Exercício:

Seja $A = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ um autômato finito não-determinístico, onde δ é dada por:

δ	a	b
q₀	$\{q_0, q_1\}$	\emptyset
q₁	\emptyset	$\{q_0, q_1, q_2\}$
q₂	\emptyset	\emptyset

1.a) Desenhe o diagrama de transição de estados e determine $L(A)$.

1.b) Construa o autômato finito determinístico mínimo, equivalente a A .

3.5 AUTÔMATO FINITO NÃO-DETERMINÍSTICO COM ϵ -MOVIMENTO

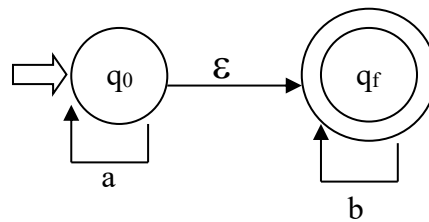
Um AFND- ϵ é uma 5-tupla $M = (\Sigma, Q, \delta, q_0, F)$, com δ da forma:

$$\delta: Q \times (\Sigma \cup \{ \epsilon \}) \rightarrow 2^Q$$

$\delta(q_1, \epsilon) = \{ q_1, q_2, \dots, q_k \} \Leftrightarrow$ Ao processar ϵ , o autômato assume, simultaneamente, o estado de origem q_1 , e todos os estados de destino q_1, q_2, \dots, q_k . Ou seja, mesmo sem ler nada na fita de entrada, num determinado momento, ele é capaz de mudar de estado.

Exemplo :

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

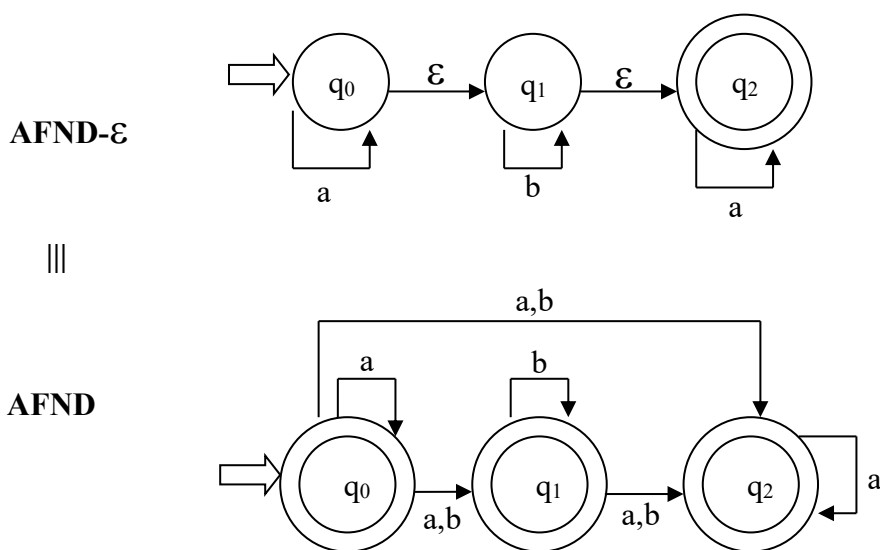


$\delta(q_0, \epsilon) = \{ q_0, q_f \} \Leftrightarrow$ Ao processar ϵ , o autômato assume, simultaneamente, o estado inicial q_0 e o estado final q_f .

$$L(M) = a^* \epsilon b^* \equiv a^* b^*$$

Teorema 3.9:

Para todo AFND- ϵ , é possível construir um AFND equivalente (Blauth, 2001).



$$L = a^* \epsilon b^* \epsilon a^* \equiv a^* b^* a^*$$

Teorema 3.10:

Seja r uma expressão regular e $L(r)$ a linguagem representada por r . Então existe um AFND- ϵ A , tal que $L(A) = L(r)$. A prova do teorema (Blauth, 2001; Hopcroft & Ullman, 1979) é construtiva, permitindo construir o AFND- ϵ diretamente a partir de r .

OBS.1: A capacidade de fazer o ϵ -movimento não se traduz em maior poder computacional, já que os AFND- ϵ continuam reconhecendo apenas linguagens regulares.

OBS.2: No Hopcroft & Ullman, a partir de $r = 01^* + 1$, mostra-se como construir diretamente um AFND- ϵ que reconhece $L(r)$.

Teorema 3.11:

Obtenção de uma Gramática Regular (Linear Unitária), a partir de um Autômato Finito (Determinístico ou Não-Determinístico) correspondente.

Executam-se os 4 passos abaixo (apesar de que produções desnecessárias poderão ser geradas):

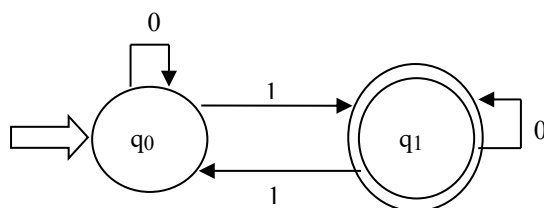
- 1) Cada estado do autômato dá origem a um símbolo não-terminal da gramática.
- 2) O estado inicial do autômato dá origem ao não-terminal inicial da gramática.
- 3) Transições para estados finais e não-finais no autômato, dão origem, na gramática, a regras de produção do tipo:

$$A \rightarrow s B \quad \begin{cases} A, B \in N \\ s \in \Sigma \end{cases}$$

- 4) Transições para estados finais no autômato, dão origem, na gramática, a regras de produção do tipo:

$$A \rightarrow s, \quad s \in \Sigma$$

Exemplo:



$$G = (\Sigma, N, S, P) \Rightarrow \quad (1) \quad \begin{cases} q_0 \text{ dá origem ao não-terminal inicial } S \\ q_1 \text{ dá origem ao não-terminal } A \end{cases} \quad (2)$$

$$(3) \quad \left\{ \begin{array}{l} S \rightarrow 0S / 1A, \\ A \rightarrow 0A / 1S \end{array} \right\}$$

$$(4) \quad \left\{ \begin{array}{l} S \rightarrow 1, \\ A \rightarrow 0 \end{array} \right\}$$

Teorema 3.12:

Obtenção de um Autômato Finito Não-Determinístico, a partir de uma Gramática Regular correspondente.

São os passos inversos do Teorema anterior (3.11), apenas observando-se que uma regra de produção do tipo $B \rightarrow \epsilon$, B não-terminal, dará origem a uma transição de estado com a cadeia vazia (i.e., um AFND- ϵ será obtido).

Exemplo:

Seja $r = 1+01^*$. A partir de r , construir G que gera $L(r)$, e daí, construir um AFND que reconhece $L(r)$.

Apresentam-se 2 soluções, obtidas a partir das gramáticas $G_1(N, \Sigma, P_1, S)$ e $G_2(N, \Sigma, P_2, S)$, tais que:

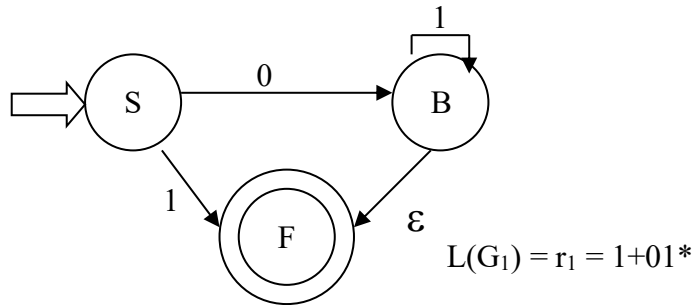
$N = \{ S, B \}$

S: Símbolo não-terminal inicial

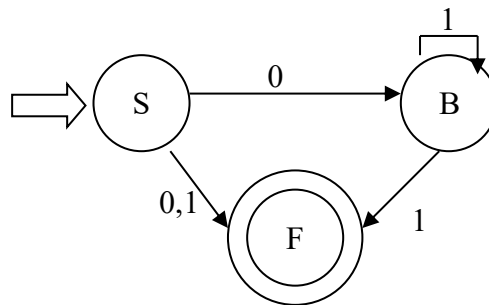
$\Sigma = \{ 0, 1 \}$

P_1 e P_2 : Conjuntos de regras de produção

$P_1 = \{ S \rightarrow 1, \\ S \rightarrow 0B, \\ B \rightarrow \epsilon, \\ B \rightarrow 1B \}$

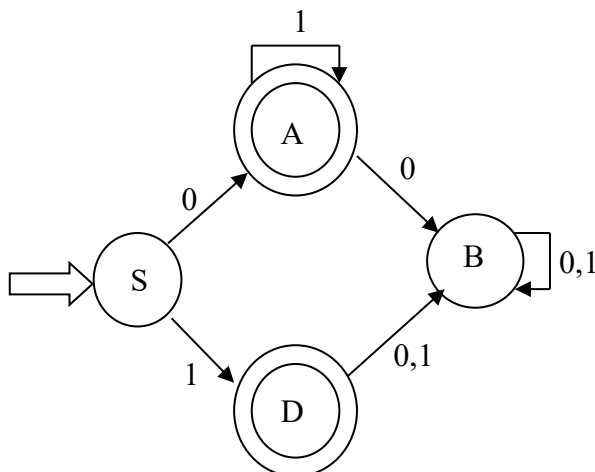


$P_2 = \{ S \rightarrow 0, \\ S \rightarrow 1, \\ S \rightarrow 0B, \\ B \rightarrow 1B, \\ B \rightarrow 1 \}$



$L(G_2) = r_2 = (0+1)+01^*1 = (0+1)+01^+ = 1+(0+01^+) = 1+(01^0+01^+) = 1+01^*$

Autômato finito determinístico, equivalente aos AFNDs acima:



$P = \{ S \rightarrow 0A / 1D / 0 / 1, \\ A \rightarrow 1A / 1 / 0B, \\ D \rightarrow 0B / 1B, \\ B \rightarrow 0B / 1B \}$

Removendo B e D, pois não há como trocá-los por símbolos terminais ($B \rightarrow 0 / 1 / \epsilon$):

$P = \{ S \rightarrow 0A / 0 / 1, \\ A \rightarrow 1A / 1 \}$

Notação: Transição no autômato finito

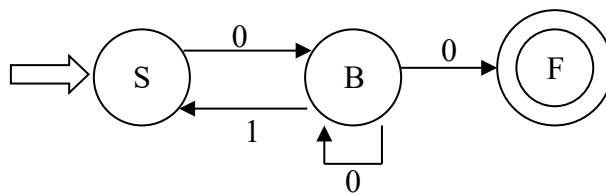
$$(q, ax) \vdash (q', x) \Leftrightarrow \delta(q, a) = q'$$

Com essa notação, pode-se escrever: $L(A) = \{ x \in \Sigma^* \mid (q_0, x) \vdash^* (q, \epsilon), q \in F \}$

Seja a gramática

$$G = (\{S, B\}, \{0,1\}, P, S), \text{ com } P = \{ S \rightarrow 0B, B \rightarrow 0B, B \rightarrow 1S, B \rightarrow 0 \}$$

e o AFND correspondente (construído a partir do procedimento anterior):



Assim, pode-se representar os processos de geração e de reconhecimento da cadeia 00100, respectivamente, como:

$$S \Rightarrow 0B \Rightarrow 00B \Rightarrow 001S \Rightarrow 0010B \Rightarrow \mathbf{00100}$$

Na geração

$$(S, \mathbf{00100}) \vdash (B, 0100) \vdash (B, 100) \vdash (S, 00) \vdash (B, 0) \vdash (F, \epsilon)$$

No reconhecimento

OBSERVAÇÕES:

- 1) Além do Teorema de Myhill-Nerode, há um outro teorema importante para ajudar a provar que uma determinada linguagem não é regular. Trata-se do chamado **Pumping Lemma** (ou Teorema do Bombeamento). Com ele pode-se provar, por exemplo, que a linguagem L abaixo não é regular:

$$L = \{ 0^n 1^n \mid n \geq 1 \}$$

- 2) Os conjuntos regulares apresentam várias propriedades interessantes. Por exemplo:
 - Se X e Y são conjuntos regulares $\Rightarrow X \cup Y$ também é regular.
 - A classe das linguagens regulares é fechada sob a complementação, isto é, se L é regular sobre Σ^* , então a linguagem $\Sigma^* - L$ também é regular.
- 3) As linguagens regulares e os autômatos finitos são bem comportados no contexto de problemas de decisão, ou seja, tipicamente existem algoritmos para várias questões que envolvem os autômatos finitos e as linguagens regulares.

Por exemplo:

- Se X e Y são linguagens regulares “ $X \subseteq Y$?” ou “ $X \equiv Y$?” são decidíveis.
- Se A_1 e A_2 são autômatos finitos “ $A_1 \equiv A_2$?” é decidível.

3.6 VARIANTES DE AUTÔMATOS FINITOS

Autômatos Finitos com saída:

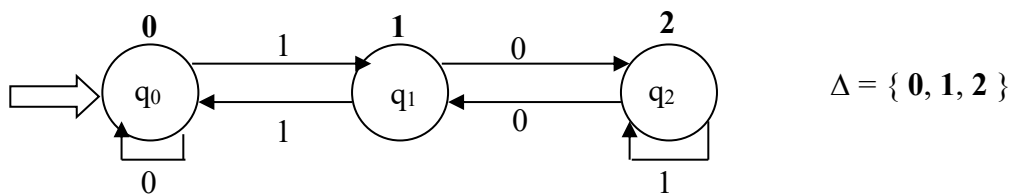
- Máquina de Moore
- Máquina de Meally

Def. 3.15: Máquina de Moore

Constitui-se de uma 6-upla $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, onde:

Δ : Alfabeto de saída ($\#\Delta \geq 2$)
 $\lambda: Q \rightarrow \Delta$ Função de saída: a saída se define exclusivamente a partir do estado da máquina.

Exemplo:



O que faz esta máquina?

Dado um número natural que lhe é fornecido, em representação binária, ela calcula o resto da divisão inteira desse número por 3 (i.e., ela implementa a operação MOD3 sobre o número).

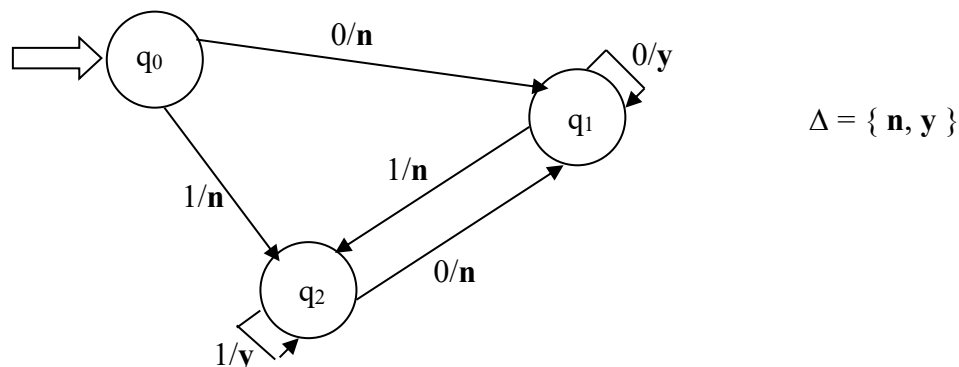
Exemplo: In: 0 1 0 1 (0101)₂ = (5)₁₀ => Mod(5, 3) = ?
 Out: 0

Def. 3.16: Máquina de Meally

Também constitui-se de uma 6-upla $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, mas a função λ tem agora uma caracterização mais refinada:

$\lambda: Q \times \Sigma \rightarrow \Delta$ Função de saída: a saída se define não só a partir do estado da máquina, mas também em função do símbolo lido na fita.

Exemplo: In: 1 0 1 1
 Out:



O que faz esta máquina? Ela reconhece a linguagem representada por $(0+1)^*(00+11)$.

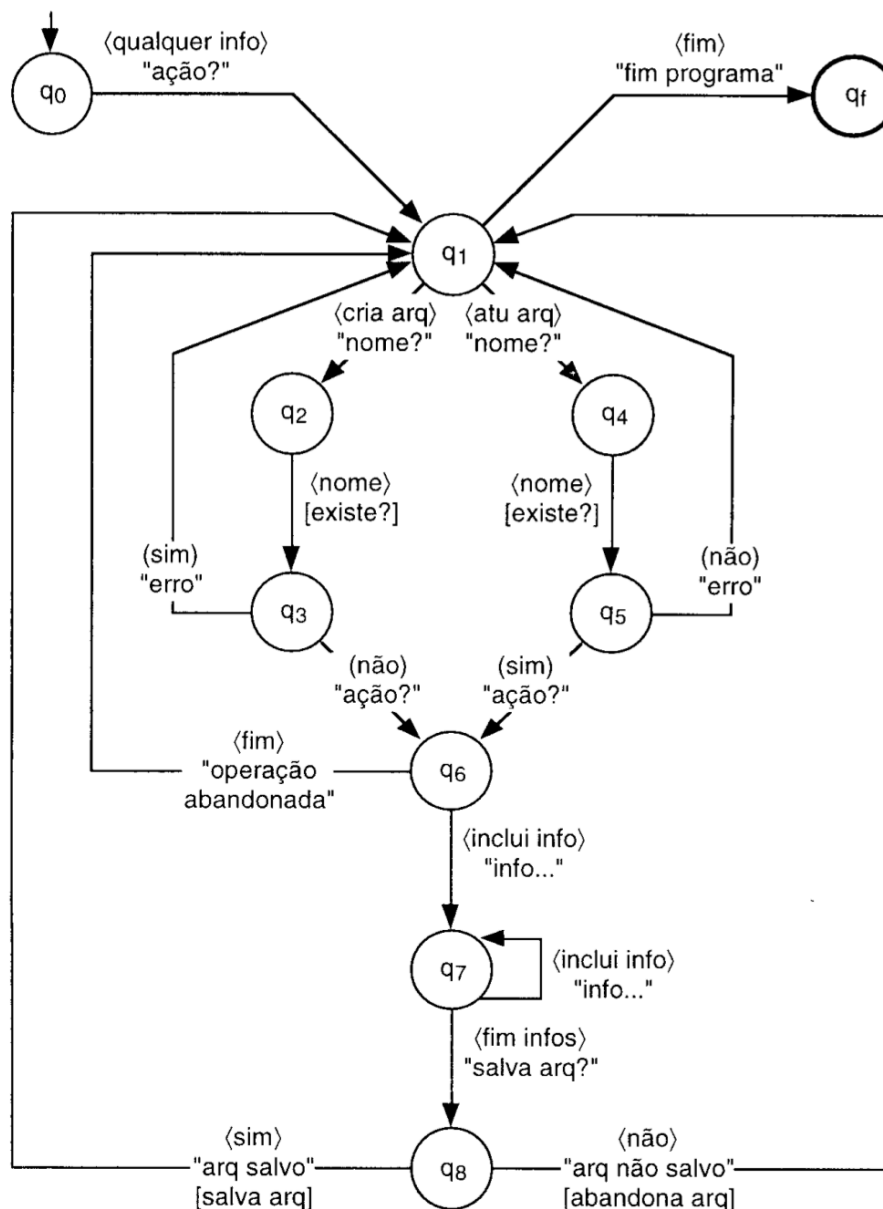
OBS: O AFD_{min} requer 5 estados!

3.7 Autômatos Finitos como Formalismo de Modelagem de Sistemas:

Exemplo 1 (P. B. Menezes):

Neste exemplo, uma Máquina de Mealy trata algumas situações típicas de um diálogo que cria e atualiza arquivos. A seguinte simbologia é adotada no grafo da função de transição:

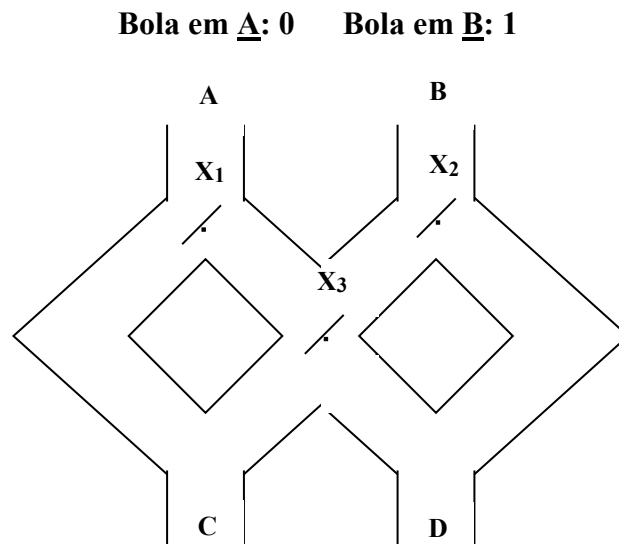
- $\langle \dots \rangle$: Entrada fornecida pelo usuário (em um teclado, por exemplo).
- “...” : Saída gerada pelo programa (em um vídeo, por exemplo).
- [...] : Ação interna ao programa, sem comunicação com o usuário.
- (...) : Resultado de uma ação interna ao programa; é usado como entrada no grafo.



Exemplo 2 (Hopcroft & Ullman): duas canaletas verticais interligadas e três desvios

Uma cadeia binária representa uma sequência de bolas que entrarão no sistema mecânico abaixo, onde:

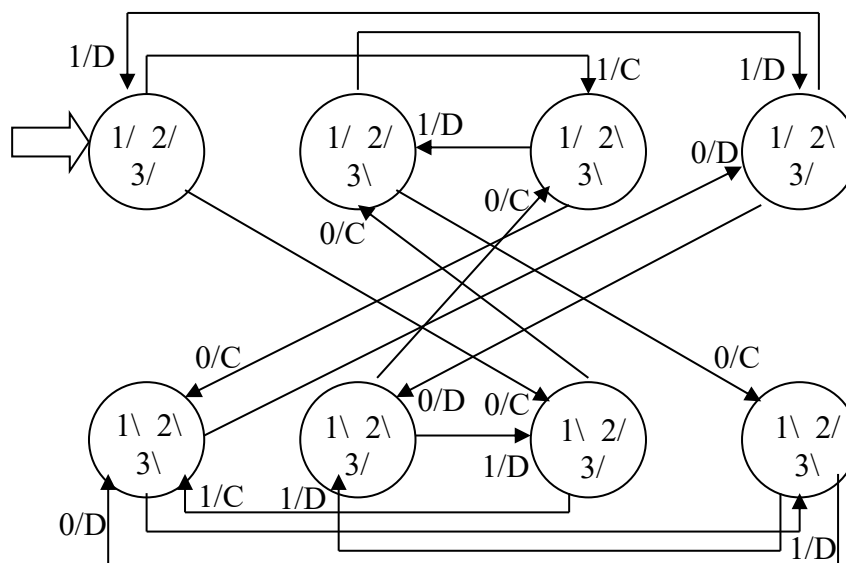
A, B: Entradas das bolas
C, D: Saídas das bolas
X₁, X₂, X₃: Desvios que mudam de posição a cada bola que passa por elas.



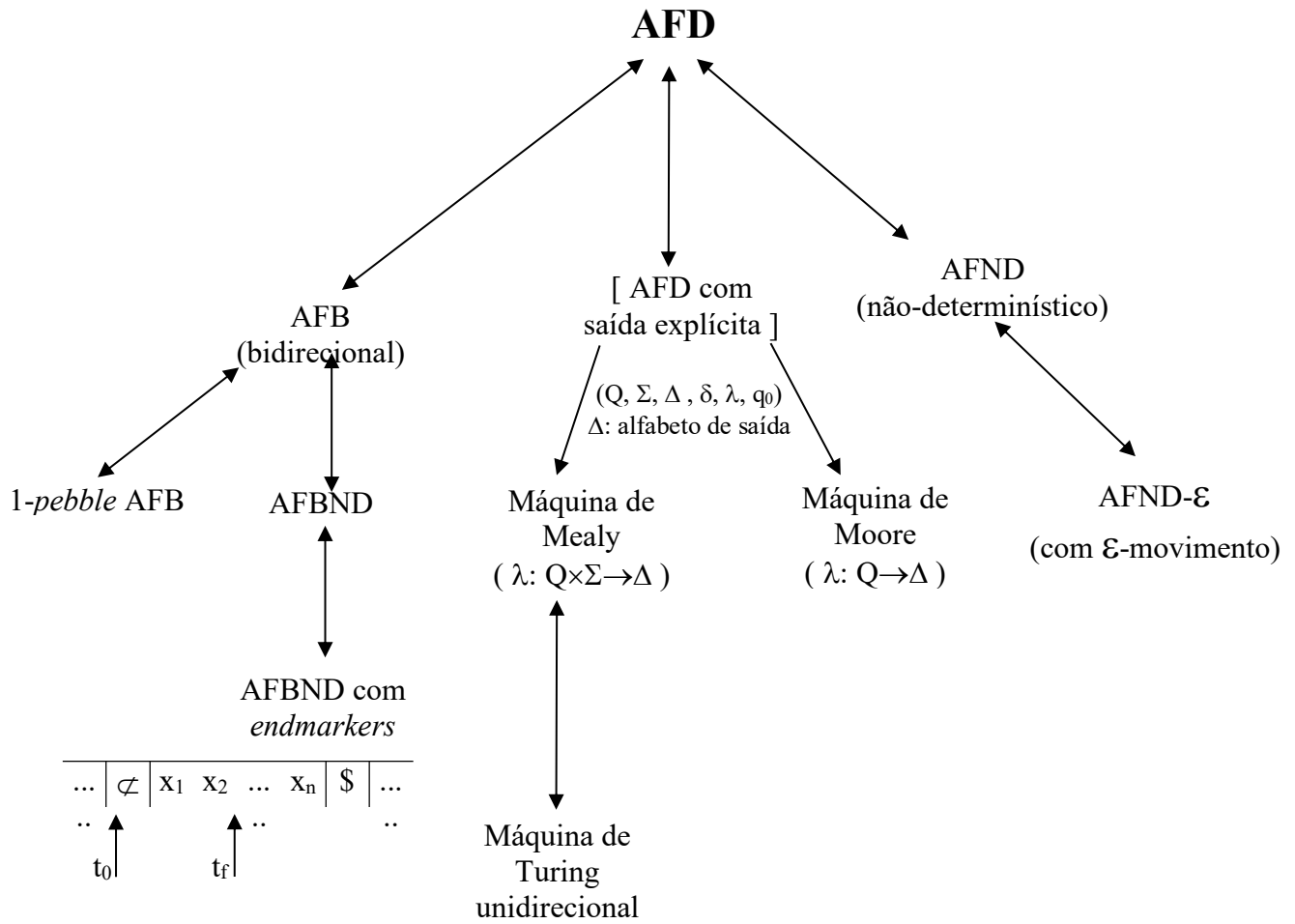
Questão que deve ser respondida:

Assumindo as alavancas na posição inicial em que se encontram no desenho, quais são as cadeias binárias de entrada, tais que a última bola sempre saia pela saída **D**?

A Máquina de Mealy a seguir modela o mecanismo, representando os 8 estados possíveis do conjunto de 3 alavancas. Em cada estado, os símbolos / ou \ afixados aos identificadores das alavancas, indicam o posicionamento de cada alavanca.



AUTÔMATO FINITO E VARIAÇÕES: QUADRO RESUMO



UNIDADE 4

MÁQUINAS COM PILHA

Recordação: Uma gramática livre de contexto (GLC) é uma 4-upla

$$G = (N, \Sigma, P, S)$$

onde:

N – Conjunto de símbolos não-terminais

Σ – Alfabeto (símbolos terminais) ($N \cap \Sigma = \emptyset$, $V = N \cup \Sigma$)

S – Símbolo não termina inicial: ($S \in N$)

P – Conjunto de produções da forma:

$$A \rightarrow \alpha, \quad A \in N, \quad \alpha \in V^*$$

Exemplo de GLC, representada na BNF (*Backus-Naur Form*)

$$G = (N, \Sigma, P, S)$$

$$N = \{S, \langle \text{op} \rangle, \langle \text{var} \rangle, \langle \text{cte} \rangle\}$$

$$\Sigma = \{1, 0, x, y, z, (,), +, *\}$$

$$P = \{ S ::= \langle \text{var} \rangle \mid \langle \text{cte} \rangle \mid S \langle \text{op} \rangle S \mid (S),$$

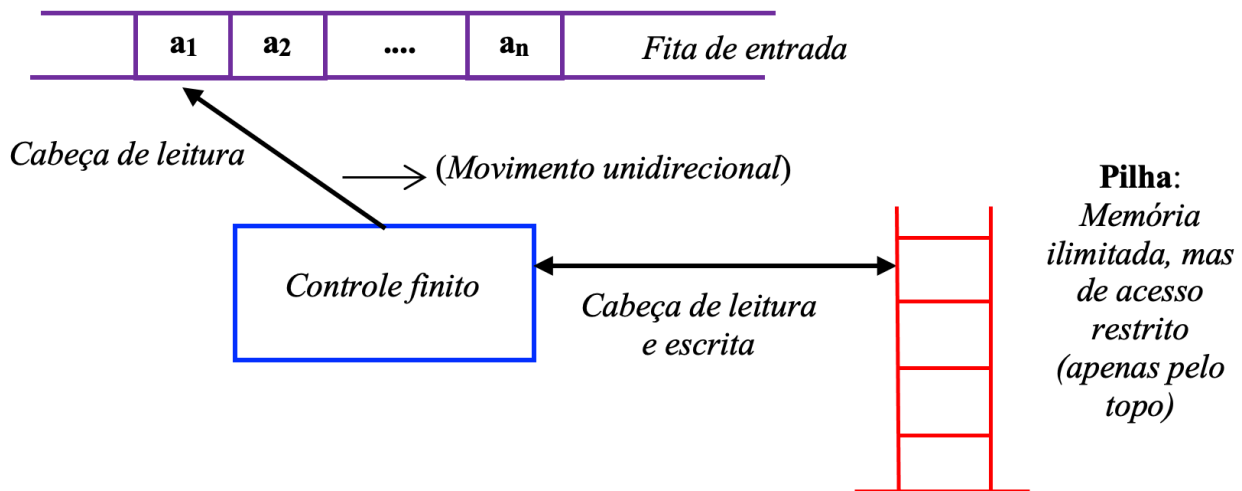
$$\langle \text{op} \rangle ::= + \mid *,$$

$$\langle \text{var} \rangle ::= x \mid y \mid z,$$

$$\langle \text{cte} \rangle ::= 0 \mid 1 \mid 0 \langle \text{cte} \rangle \mid 1 \langle \text{cte} \rangle \}$$

$$(x + 101)^*y \in L(G)$$

4.1 AUTÔMATOS DE “EMPILHAMENTO” (*Pushdown Automata* – PDA)



Linguagem Livres de Contexto são reconhecidas pelos PDAs

Intuitivamente:

Dependendo do:

- Estado do controle finito,
- Símbolo que está sendo lido na fita de entrada,
- Símbolo no topo da pilha,

o autômato de empilhamento:

- **Muda de estado,**
- Escreve **um número finito** de símbolos na pilha (escrever ϵ corresponde a apagar o símbolo no topo), e
- Move sua cabeça leitora **uma** posição para a direita.

O autômato de empilhamento pode também efetuar ϵ –movimentos (movimentos com a cadeia vazia) que corresponde a mudar de estado e mudar o conteúdo da pilha, sem ler o símbolo na fita de entrada (isto é, sem mover sua cabeça leitora para a direita).

Definição 4.1: Um autômato de empilhamento é uma 7-upla

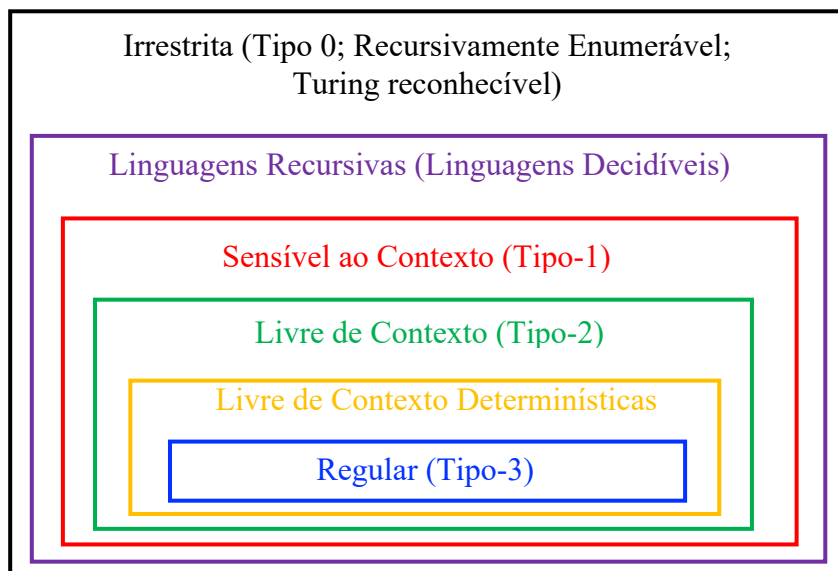
$$A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F),$$

onde:

Q	Conjunto finito não vazio de estados
Σ	Alfabeto de entrada
Γ	Alfabeto da pilha
$q_0 \in Q$	Estado inicial
$z_0 \in \Gamma$	Símbolo inicial da pilha
$F \subseteq Q$	Conjunto de estados finais

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \rightarrow 2^{(Q \times \Gamma^*)} \quad (\text{não determinístico!})$$

OBS: Ou seja, PDAs são não-determinísticos e, já adiantando, possuem poder computacional maior que a versão determinística (PDAD).



Definição 4.2:

Uma **configuração** de um PDA é um elemento de $Q \times \Sigma^* \times \Gamma^*$.

Definição 4.3:

Uma **transição** do autômato de empilhamento é representada por

$$(q, ax, zw) \vdash (q', x, yw) \Leftrightarrow (q', y) \in \delta(q, a, z)$$

com: $q, q' \in Q$; $a \in \Sigma \cup \{\epsilon\}$; $x \in \Sigma^*$; $w, y \in \Gamma^*$; $z \in \Gamma$

OBS: A condição $z \in \Gamma$ impõe que o PDA não faz transição se sua pilha estiver vazia.

Definição 4.4:

A linguagem aceita por $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$, **por estado final** é:

$$L(A) = \{ w \in \Sigma^* / (q_0, w, z_0) \vdash^* (q, \epsilon, x) ; q \in F ; x \in \Gamma^* \}$$

Definição 4.5:

A linguagem aceita por $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ **por pilha vazia** é:

$$N(A) = \{ w \in \Sigma^* / (q_0, w, z_0) \vdash^* (q, \epsilon, \epsilon) , q \in Q \}$$

OBS: $L(A) = N(A)$

Definição 4.7: Seja Σ um alfabeto e A um autômato de empilhamento.

$L_\Sigma = \{ L \subseteq \Sigma^* / L = L(A) \}$ Conjunto das linguagens aceitas por estado final

$N_\Sigma = \{ L \subseteq \Sigma^* / L = N(A) \}$ Conjunto das linguagens aceitas por pilha vazia

Proposição 4.1: $L_\Sigma = N_\Sigma$

$$L = L(A) \Rightarrow L = N(B)$$

Lema 4.1:

Se $L = L(A)$, para algum autômato de empilhamento A , então existe autômato de empilhamento B tal que $L = N(B)$.

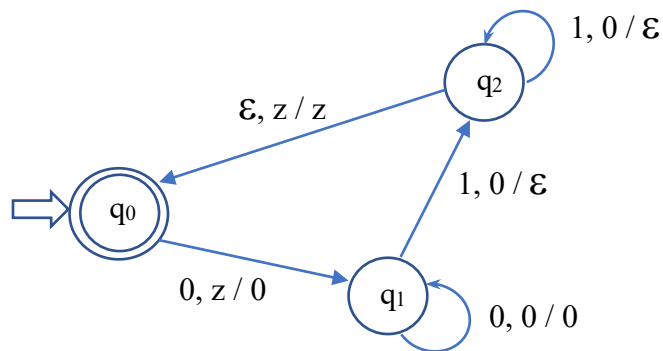
Proposição 4.2: $L_\Sigma = N_\Sigma = \{ L \subseteq \Sigma^* / L \text{ é linguagem livre de contexto} \}$

Exemplo: Construir autômato de empilhamento que aceita $L = \{ 0^n 1^n / n \geq 0 \}$

$$A = (\{q_0, q_1, q_2\}, \{0,1\}, \{z,0\}, \delta, q_0, z, \{q_0\})$$

onde δ é dado por:

$\delta(q_0, 0, z) = \{(q_1, 0)\}$	Empilha 0s.
$\delta(q_1, 0, 0) = \{(q_1, 0)\}$	
$\delta(q_1, 1, 0) = \{(q_2, \epsilon)\}$	Para cada 1 encontrado, desempilha um 0.
$\delta(q_2, 1, 0) = \{(q_2, \epsilon)\}$	
$\delta(q_2, \epsilon, z) = \{(q_0, z)\}$	



Pode-se provar que a linguagem $\{0^n 1^n / n \geq 0\}$ é livre de contexto **determinística**. O autômato A do exemplo acima (reconhecedor da linguagem $\{0^n 1^n / n \geq 0\}$) é **determinístico**.

Definição 4.6:

Um autômato de empilhamento $A = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ é **determinístico** se, para todo $(q, a, z) \in Q \times \Sigma \times \Gamma$, tem-se:

- 1) $|\delta(q, a, z)| \leq 1$
- 2) Apenas 1 das configurações a seguir é válida (ou seja, $\neq \emptyset$):
 $\delta(q, a, z), \delta(q, \varepsilon, z), \delta(q, a, \varepsilon), \delta(q, \varepsilon, \varepsilon)$.

OBS:

- Diferentemente dos AFDs, **PDADs admitem ε -movimentos** sobre a fita, ou seja, $\delta(_, \varepsilon, _)$, já que sua configuração também envolve a pilha.
- Há ainda **ε -movimentos** na pilha, i.e., $\delta(_, _, \varepsilon)$.
- A condição 1 estabelece que cada transição leva a no máximo 1 configuração.
- A condição 2 estabelece que isso deve ocorrer mesmo quando ε -movimentos são possíveis na fita e na pilha.

Ou seja:

$\delta(q, a, z)$ e $\delta(q, \varepsilon, z)$: seria um movimento ignorando a **fita**

$\delta(q, a, z)$ e $\delta(q, a, \varepsilon)$: seria um movimento ignorando a **pilha**

$\delta(q, a, z)$ e $\delta(q, \varepsilon, \varepsilon)$: seria um movimento ignorando a **fita** e a **pilha**

$\delta(q, \varepsilon, z)$ e $\delta(q, a, \varepsilon)$: seria um movimento ignorando a **fita** e a **pilha**

$\delta(q, \varepsilon, z)$ e $\delta(q, \varepsilon, \varepsilon)$: seria um movimento ignorando a **fita** e a **pilha**

$\delta(q, a, \varepsilon)$ e $\delta(q, \varepsilon, \varepsilon)$: seria um movimento ignorando a **fita** e a **pilha**

Um exemplo de como intuir que a computabilidade dos PDAs possa ser maior que a dos PDADs:

Considere-se a linguagem

$$L = \{ ww^R / w \in \{0,1\}^* ; w^R \text{ é o reverso de } w \}$$

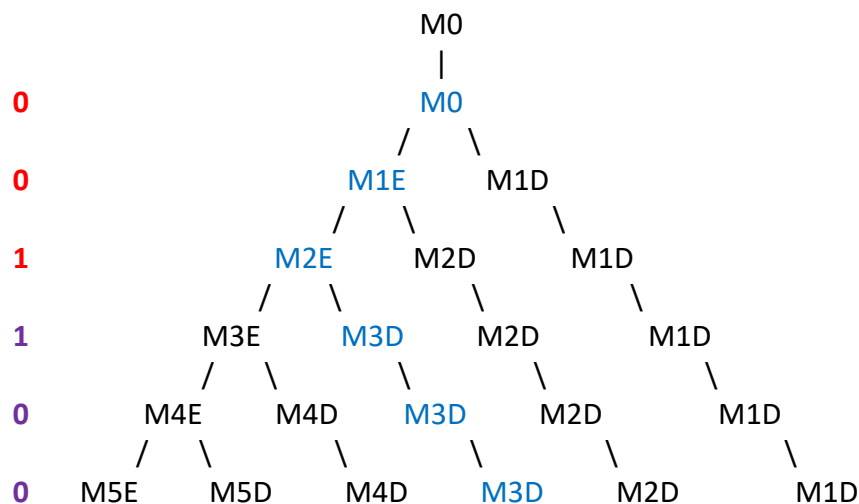
(ou seja, L é formada por palíndromes *pares*)

Intuitivamente, um autômato de empilhamento para reconhecer L deverá ser não determinístico porque, durante o processamento de uma cadeia ww^R , não há maneira de saber quando termina a cadeia w e começa a cadeia w^R . Assim, para cada símbolo lido, o autômato deve prever duas possíveis situações:

- a cadeia w ainda não terminou
- a cadeia w terminou e os próximos símbolos serão de w^R

Para a linguagem $L' = \{ waw^R / w \in \{0,1\}^*, \Sigma = \{0, 1, a\} \}$, pode-se construir um **PDAD** que a reconhece. No entanto, não é possível construir um PDAD para reconhecer L; para isso é necessário um PDA, pois L é livre de contexto (mas **não** l.c. determinística).

OBS: Replicações de um PDA que reconhece a linguagem L de palíndromes pares, ao processar a cadeia **001100**.



Exercício: Pode-se provar que a linguagem $\{ a^i b^j c^k \mid i, j, k \geq 0 \text{ com } i = j \text{ or } i = k \}$ é livre de contexto, mas **não** é livre de contexto determinística. Construir um PDA que reconhece L.

Exemplo [Hopcroft, Motwani e Ullman, 3rd ed., 2007, p.230]:
PDA que reconhece a linguagem de palíndromes binárias pares
 $L = \{ ww^R / w \in \{0,1\}^*; w^R \text{ é o reverso de } w \}$.

A notação empregada no exemplo para as transições de estado é mais compacta do que a que definimos:

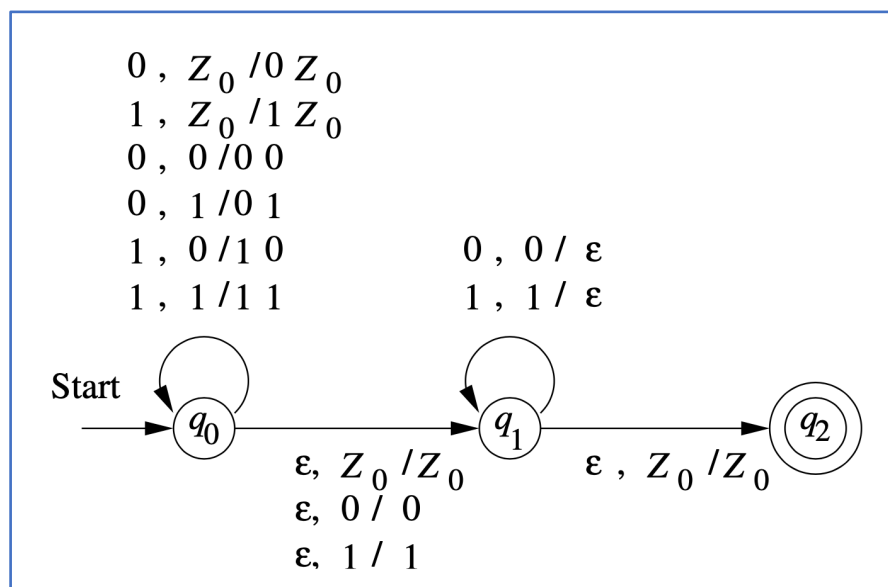
$a, b / b \Rightarrow$ lendo o símbolo **a** na fita e **b** na pilha, a pilha não se altera

$a, b / Xb \Rightarrow$ lendo o símbolo **a** na fita e **b** na pilha, adicione a cadeia **X** à pilha

Ou seja, cada uma das transições de estado acima corresponde na nossa notação a duas transições:

$a, b / b \Rightarrow \{ a, b / \varepsilon \Rightarrow a, b / b \}$

$a, b / Xb \Rightarrow \{ a, b / \varepsilon \Rightarrow a, b / Xb \}$



Configuração inicial: $(q_0, 0, Z_0)$

Sequência de transições que levam à aceitação da cadeia de entrada: **0 1 1 0**
(mas há transições alternativas, que não levariam ao reconhecimento)

$(q_0, 0, Z_0) \Rightarrow (q_0, 0 Z_0)$

Pilha: 0 Z_0

$(q_0, 1, 0) \Rightarrow (q_0, 1 0)$

Pilha: 1 0 Z_0

Neste ponto ocorre um ε -movimento para o estado q_1 :

$(q_0, \varepsilon, 1) \Rightarrow (q_1, 1)$

Pilha: 1 0 Z_0

$(q_1, 1, 1) \Rightarrow (q_1, \varepsilon)$

Pilha: 0 Z_0

$(q_1, 0, 0) \Rightarrow (q_1, \varepsilon)$

Pilha: Z_0

Neste ponto ocorre um ε -movimento para o estado q_2 :

$(q_1, \varepsilon, Z_0) \Rightarrow (q_2, Z_0)$

Pilha: Z_0

Definição 4.8: Derivações em gramáticas livres de contexto:

Gramática de *balanceamento de parênteses*:

$$G_{bp} = (N = \{S\}, \Sigma = \{ (,) \}, S, P_{bp}), \text{ com } P_{bp} = \{ S \rightarrow SS \mid (S) \mid () \}$$

1. Derivações **Mais-à-Esquerda** (*leftmost derivations*):

$$S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$$

2. Derivações **Mais-à-Direita** (*rightmost derivations*):

$$S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$$

3. Derivações **arbitrárias**:

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$$

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$$

Não são derivações mais-à-direita nem mais-à-esquerda.

Definição 4.9: Gramática Ambígua:

Uma gramática livre de contexto é **ambígua**, se alguma das cadeias geradas a partir dela possuir mais de uma derivação mais-à-esquerda (ou mais-à-direita), o que acaba gerando mais de uma árvore sintática (*parsing tree*) associada.

OBS:

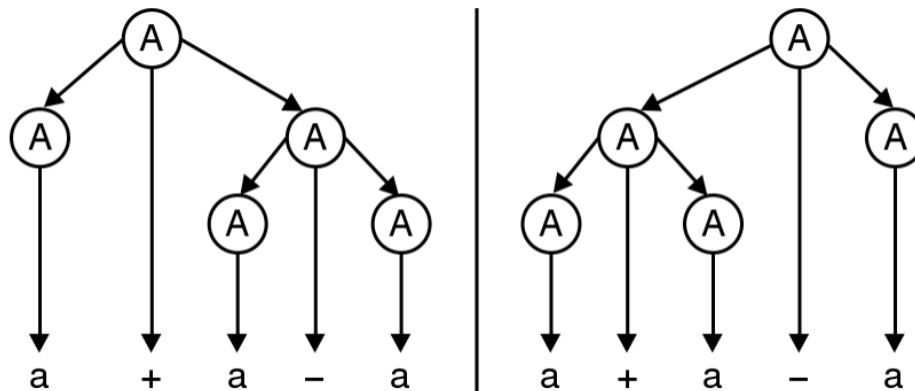
- 1) O fato de G_{bp} ter duas derivações diferentes para a cadeia $(())()$ não garante que é G_{bp} ambígua (pois essas derivações não são mais-à-esquerda).
- 2) A multiplicidade de derivações mais-à-esquerda tem uma certa maior importância, pois escrevemos e lemos código computacional a partir da esquerda.
- 3) Não existe (e nunca existirá) um algoritmo para determinar se uma gramática livre de contexto arbitrária é ambígua ou não!

Exemplo 1:

A gramática livre de contexto de somas e subtrações $A \rightarrow A + A \mid A - A \mid a$ é ambígua pois a cadeia $a + a - a$ possui 2 derivações mais-à-esquerda:

$$\begin{array}{ll} A \Rightarrow \underline{A} + A & A \Rightarrow \underline{A} - A \\ \Rightarrow a + \underline{A} & \Rightarrow \underline{A} + A - A \\ \Rightarrow a + \underline{A} - A & \Rightarrow a + \underline{A} - A \\ \Rightarrow a + a - \underline{A} & \Rightarrow a + a - \underline{A} \\ \Rightarrow a + a - a & \Rightarrow a + a - a \end{array}$$

Ou, a cadeia $a + a - a$ possui 2 árvores sintáticas ($A \rightarrow A + A \mid A - A \mid a$):



Exemplo 2: O problema do **else** pendurado (*dangling else*).

Numa gramática contendo as regras

```
Comando  $\rightarrow$  If Condição then Comando |
           If Condição then Comando else Comando |
           ...
Condição  $\rightarrow$  ...
```

A cadeia (expressão)

If a **then** **If** b **then** s1 **else** s2

é ambígua (tem duas interpretações sintáticas possíveis), dependendo se o **else** é associado ao primeiro ou ao segundo **if**:

```
If a then begin If b then s1 end else s2
If a then begin If b then s1 else s2 end
```

Definição 4.10: Linguagem Inerentemente Ambígua:

Uma linguagem livre de contexto é **inerentemente ambígua** se todas as gramáticas livres de contexto geradoras desta linguagem são ambíguas.

Exercícios:

(1) Mostre que a gramática $G_1 = \{ \{S, A\}, \{0, 1, +\}, S, P_1 \}$, com

$$P_1 = \{ \begin{array}{l} S \rightarrow A + A, \\ A \rightarrow 0 / 1 \end{array} \}$$

é não ambígua.

(2) Mostre que a gramática $G_2 = \{ \{S, E\}, \{1, 2, 3, +, *\}, S, P_2 \}$, com

$$P_2 = \{ \begin{array}{l} S \rightarrow E, \\ E \rightarrow E + E, \\ E \rightarrow E * E, \\ E \rightarrow 1 / 2 / 3 \end{array} \}$$

é ambígua.

Dica: Tome como base a derivação da cadeia $1 + 2 * 3$.

(3) Com base na gramática G_2 do exercício anterior, como o conjunto de regras de produção P_2 poderia ser alterado de forma que G_2 deixasse de ser ambígua?

Dica: Considere a adição de dois símbolos terminais novos: (e).

Análise Sintática Descendente com Retorno:

Seja a gramática G livre de contexto definida com:

$$\begin{aligned}\Sigma &= \{ a, +, \times, (,) \} \\ N &= \{ E, T, F \} \\ S &= E \\ P &= \{ \\ &\quad E \rightarrow E+T / T, \\ &\quad T \rightarrow T \times F / F, \\ &\quad F \rightarrow a / (E) \\ &\quad \} \end{aligned}$$

Eis um PDA que reconhece sua linguagem correspondente (por pilha vazia!):

$PDA_G = (\{q\}, \Sigma, \Gamma, \delta, q, E, \emptyset)$, com

$$\Sigma = \{ a, +, \times, (,) \},$$

$$\Gamma = \Sigma \cup \{ E, T, F \}$$

e δ dado por:

$$\begin{aligned}\delta(q, \varepsilon, E) &= \{ (q, E+T), (q, T) \} && \{ 1a, 1b \} \\ \delta(q, \varepsilon, T) &= \{ (q, T \times F), (q, F) \} && \{ 2a, 2b \} \\ \delta(q, \varepsilon, F) &= \{ (q, a), (q, (E)) \} && \{ 3a, 3b \} \\ \delta(q, x, x) &= \{ (q, \varepsilon) \}, \quad x \in \Sigma && \{ 4 \} \end{aligned}$$

Notação usada: $\delta(_, _, \textit{topo da pilha}) = (_, \textit{cadeia que substituirá o topo da pilha})$

$$\delta(_, _, x) = (_, Y) \equiv (\delta(_, _, x) = (_, \varepsilon) \Rightarrow \delta(_, _, _) = (_, Y))$$

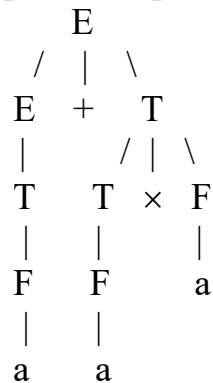
A cadeia **a+a×a** pode ser reconhecida com a seguinte sequência de transições:

	(q, a+a×a, <u>E</u>)	
1a	(q, a+a×a, <u>E</u> +T)	(OBS: O topo da pilha é E.)
1b	(q, a+a×a, <u>T</u> +T)	
2b	(q, a+a×a, <u>F</u> +T)	
3a	(q, a+a×a, a+T)	
4	(q, a+a×a, +T)	
4	(q, a+a×a, T)	
2a	(q, a+a×a, <u>T</u> ×F)	
2b	(q, a+a×a, <u>F</u> ×F)	
3a	(q, a+a×a, a×F)	
4	(q, a+a×a, ×F)	
4	(q, a+a×a, <u>F</u>)	
3a	(q, a+a×a, a)	
4	(q, a+a×a, ε)	

Observando as transições do PDA_G , nota-se que a fita de entrada contém, a cada instante, a parte da cadeia de entrada que falta ser analisada, e que o conteúdo da pilha simula uma derivação mais à esquerda:

$$\underline{E} \Rightarrow \underline{E}+T \Rightarrow \underline{T}+T \Rightarrow \underline{F}+T \Rightarrow a+\underline{T} \Rightarrow a+\underline{T}\times F \Rightarrow a+\underline{F}\times F \Rightarrow a+a\times \underline{F} \Rightarrow a+a\times a$$

Este autômato de pilha é conhecido como **Reconhecedor Descendente** ou **Analisador Sintático Descendente** (*top-down parser*), porque, como simula derivações mais à esquerda, constrói a árvore sintática da cadeia de entrada, de cima para baixo (e da esquerda para a direita), que no exemplo corresponde a:


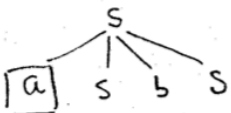
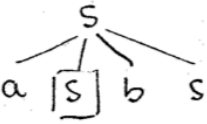
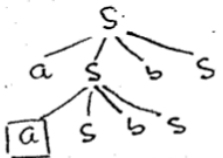
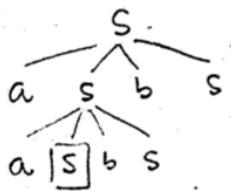
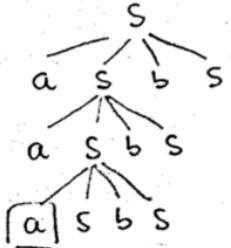
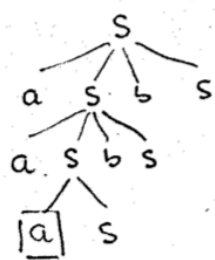
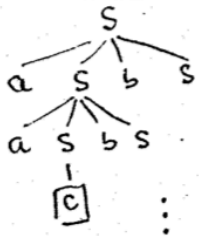


Entretanto, como pode ser observado da definição da função de transição de estados δ , o autômato PDA_G é não-determinístico, sendo que as transições mostradas foram convenientemente escolhidas a cada passo. Uma questão interessante é: Seria possível automatizar este processo? A resposta é sim, através de um algoritmo que implementa o processo conhecido como **Análise Sintática Descendente com Retorno** (*top-down backtrack parsing*), a qual é descrita pelos 4 passos a seguir:

- (1) Começar a árvore sintática pelo símbolo inicial da pilha (o símbolo inicial da gramática). Esse símbolo é o nó ativo inicial. Em seguida, executar os passos (2) e (3), recursivamente.
- (2) Se o nó ativo é um não-terminal **A**, escolher a primeira **A**-produção ainda não utilizada $A \rightarrow X_1X_2...X_k$ e criar na árvore sintática $X_1, X_2, ..., X_k$ como descendentes diretos de **A**. Marcar o primeiro destes descendentes (o mais à esquerda) como ativo.
- (3) Se o nó ativo é um terminal **a**, então compará-lo com o símbolo atual de entrada. Se eles foram iguais, então tornar ativo o símbolo imediatamente à direita de **a** na árvore, e avançar com a cabeça leitora para o próximo símbolo de entrada. Se eles forem diferentes, retornar ao último não terminal **A** expandido, para o qual existe uma **A**-produção ainda não utilizada.
- (4) Se não existe nó ativo e todos os símbolos de entrada foram lidos, então a análise sintática termina com sucesso (a cadeia foi reconhecida). Caso contrário, rejeitar a cadeia (isto é, existe um erro sintático na cadeia).

Exemplo:

Seja a gramática $G = (\Sigma = \{a, b, c\}, N = \{S\}, S, P = \{S \rightarrow aSbS / aS / c\})$, com a cadeia de entrada aacbc.

ÁRVORE	CADEIA	OBSERVAÇÕES
	\downarrow aacbc	o símbolo \square mostra o nó atualmente ativo; \downarrow indica o símbolo de entrada que está sendo lido
	\downarrow aacbc	escolhe-se a primeira alternativa para expandir S.
	a \downarrow acbc	o símbolo de entrada "casa" com o nó da árvore; o próximo símbolo é considerado.
	a \downarrow acbc	escolhe-se a primeira alternativa para expandir S
	aa \downarrow cbc	"casamento" de símbolos; passa ao próximo nó da árvore
	aa \downarrow cbc	o símbolo S foi expandido; A escolha no estado é incorreta porque não há "casamento" de terminais. Testa a próxima alternativa
	aa \downarrow cbc	essa alternativa também é incorreta; tenta a próxima alternativa.
	aac \downarrow bc ⋮	"casamento" ⋮

OBS: Exemplos de linguagens que não são livres de contexto:

1) $L_1 = \{ ww \mid w \in \{a, b\}^* \}$

$G_1 = (\{S, A, B, C, D, E\}, \{a, b\}, P, S)$

$$P = \{ \begin{array}{l} S \rightarrow ABC, \\ AB \rightarrow aAD, \\ AB \rightarrow bAE, \\ DC \rightarrow BaC, \\ EC \rightarrow BbC, \\ Da \rightarrow aD, \\ Db \rightarrow bD, \\ Ea \rightarrow aE, \\ Eb \rightarrow bE, \\ AB \rightarrow \varepsilon, \\ C \rightarrow \varepsilon, \\ aB \rightarrow Ba, \\ bB \rightarrow Bb \end{array} \}$$

2) $L_2 = \{ w \in \{a, b, c\}^+ \mid \#a = \#b = \#c \}$

$G_2 = (\{S, A, B, C\}, \{a, b, c\}, P, S)$

$$P = \{ \begin{array}{l} S \rightarrow ABC, \\ S \rightarrow ABCS, \\ AB \rightarrow BA, \\ AC \rightarrow CA, \\ BC \rightarrow CB, \\ BA \rightarrow AB, \\ CA \rightarrow AC, \\ CB \rightarrow BC, \\ A \rightarrow a, \\ B \rightarrow b, \\ C \rightarrow c \end{array} \}$$

3) $L_3 = \{ 0^n 1^n 2^n, n \geq 0 \}$

4) $L_4 = \{ a^p \mid p \text{ é um número primo} \}$

OBS 1: Como saber se uma dada linguagem L é de determinado tipo?

- Não há solução geral, nem mesmo quando a pergunta é “**L é regular?**”
- Teoremas ajudam a determinar se L não é de determinado tipo:
 - Teorema de Myhill-Nerode
 - *Pumping lemma* para linguagens regulares
 - *Pumping lemma* para linguagens livre de contexto
 - etc...

OBS 2: Como saber se uma dada gramática G é de determinado tipo?

- Além das técnicas acima, é útil reescrever as regras de produção de G de forma que as novas regras sigam formas normais conhecidas.
- Outras formas normais existem para GLC ($a \in \Sigma$; $S, X, Y, Z \in N$; $W \in N^*$):
 - **Chomsky:** $X \rightarrow a$ / $X \rightarrow YZ$ / $S \rightarrow \varepsilon$
 - **Greibach:** $X \rightarrow a$ / $X \rightarrow aW$ / $S \rightarrow \varepsilon$

Ex.: Seja a linguagem livre de contexto $L = \{ ww^R \mid w \in \{a, b\}^* \}$
(palíndromes pares de a's de b's)

$G(L)$:

$\Sigma = \{a, b\}$

$N = \{I\}$

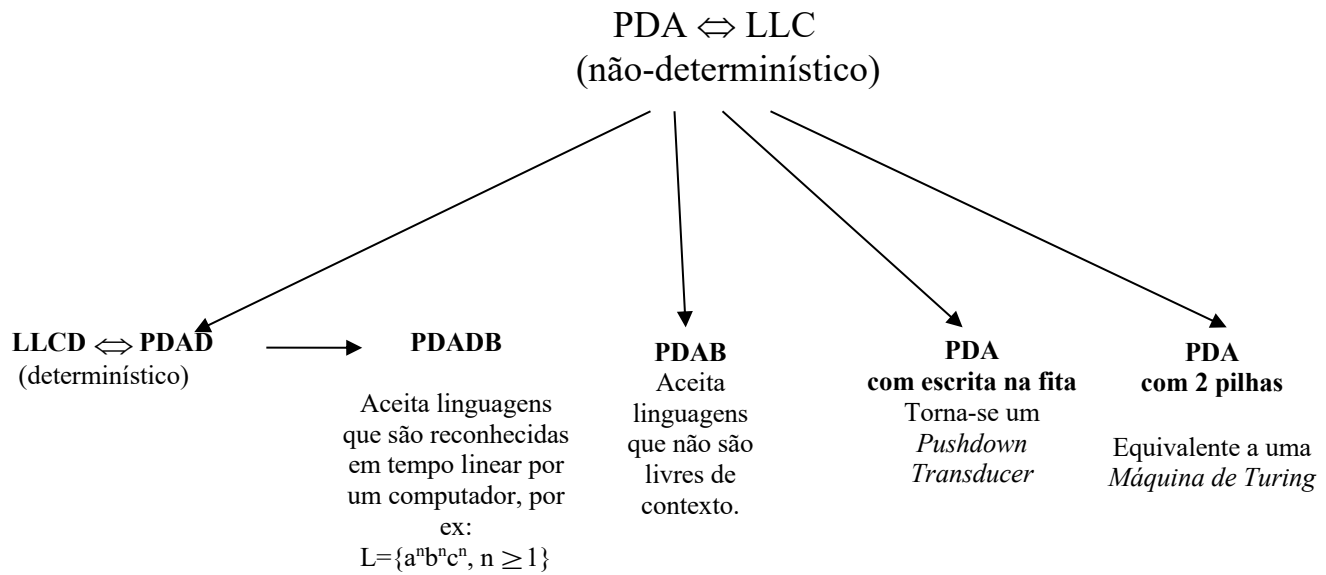
I: Símbolo inicial

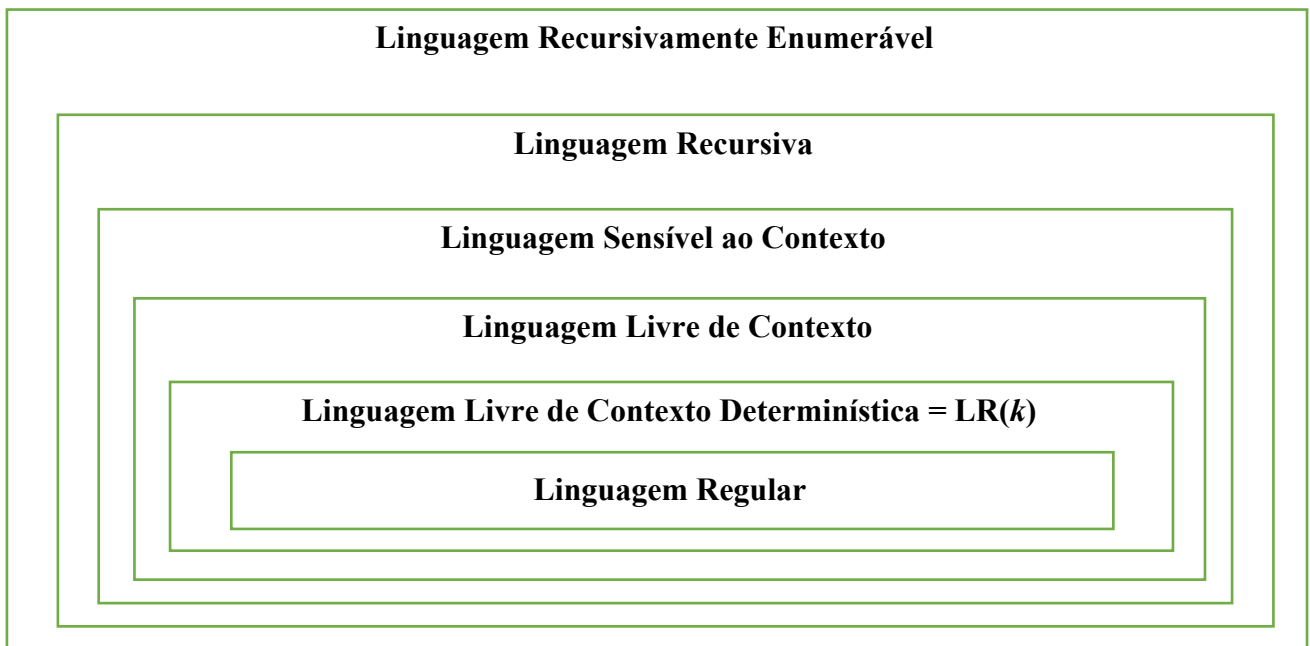
$P = \{ I \rightarrow \varepsilon ,$
 $I \rightarrow a / b,$
 $I \rightarrow aIa / bIb \}$

Exercício:

Reescrever a gramática acima segundo as formas normais de Chomsky e de Greibach.

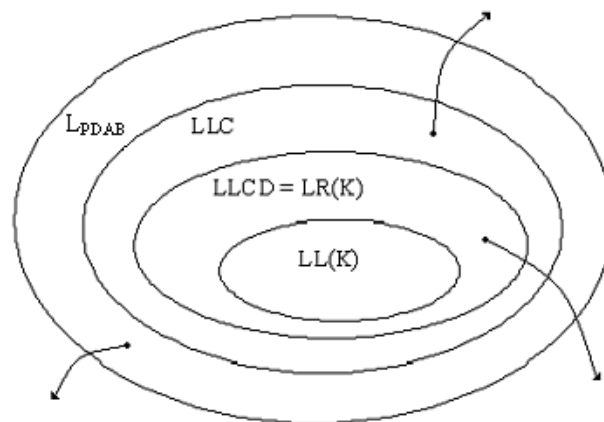
4.2 AUTÔMATO DE EMPILHAMENTO E VARIAÇÕES





LINGUAGENS

$$L = \{ww^R \mid w \in \{0,1\}^*; w^R \text{ é o reverso de } w\}$$



$$L'' = \{a^n b^n c^n, n \geq 1\}$$

$$L' = \{waw^R \mid w \in \{0,1\}^*\}$$

OBS:

LL(*k*) e **LR(*k*)** são apropriadas para análise sintática sem retorno (sem *backtracking*)

4.3 “STACK AUTOMATA” (Autômatos de Pilha)

- PDA + 1) Cabeça bidirecional

2) Fita com *endmarkers*

3) Modo adicional de deslocamento na pilha: **apenas-leitura** ao longo de toda a sua extensão (uma vez neste modo, ele só termina quando a cabeça da pilha voltar ao topo)

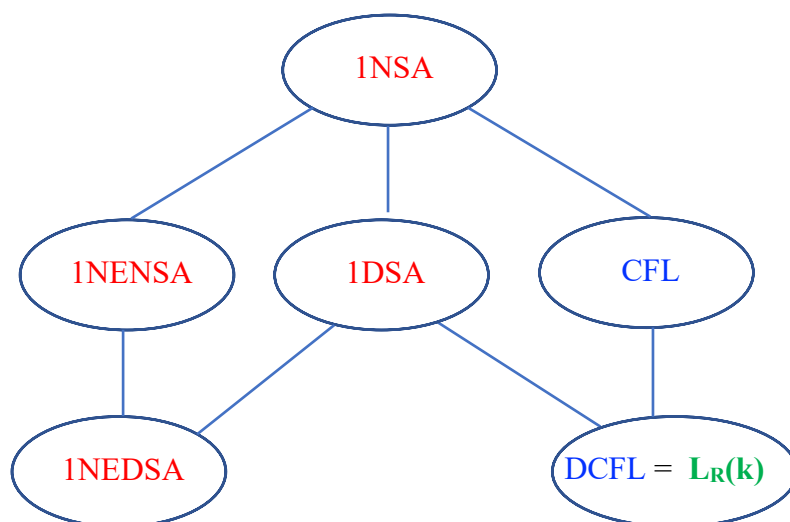
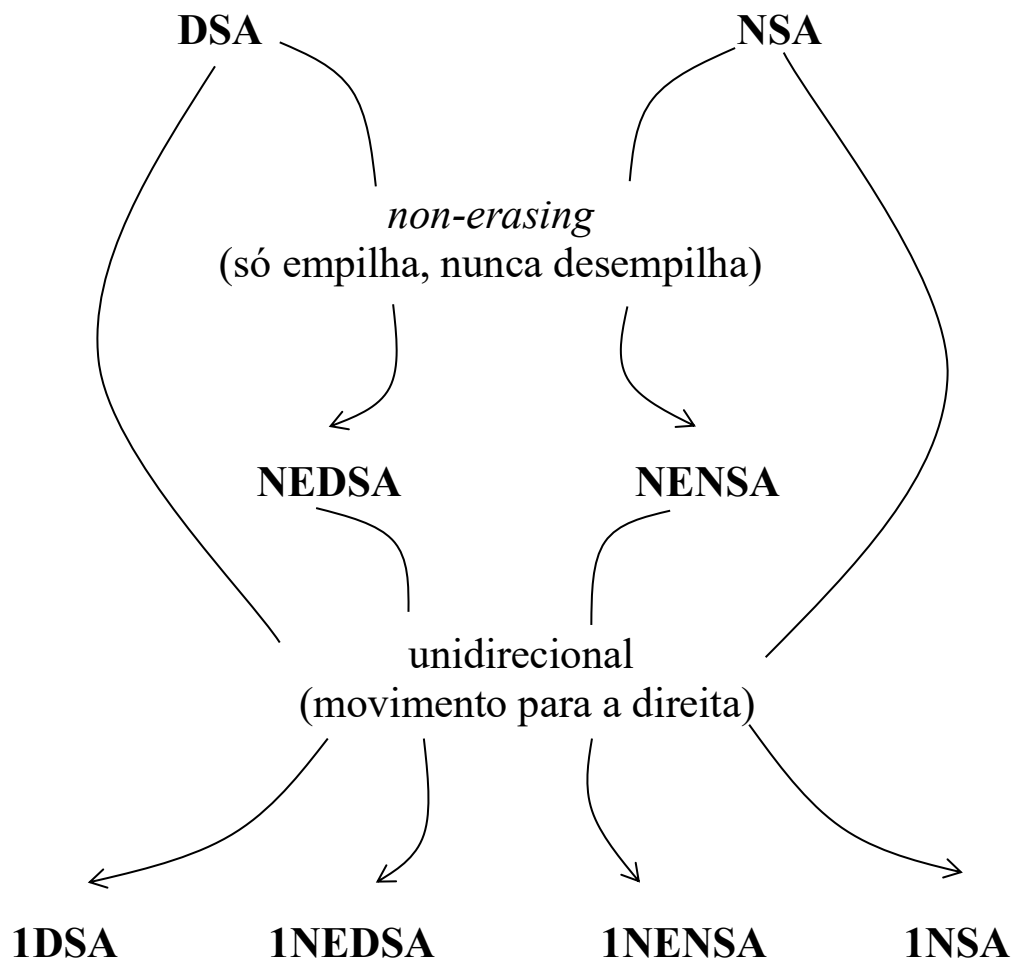
- Aceitação: por estado final

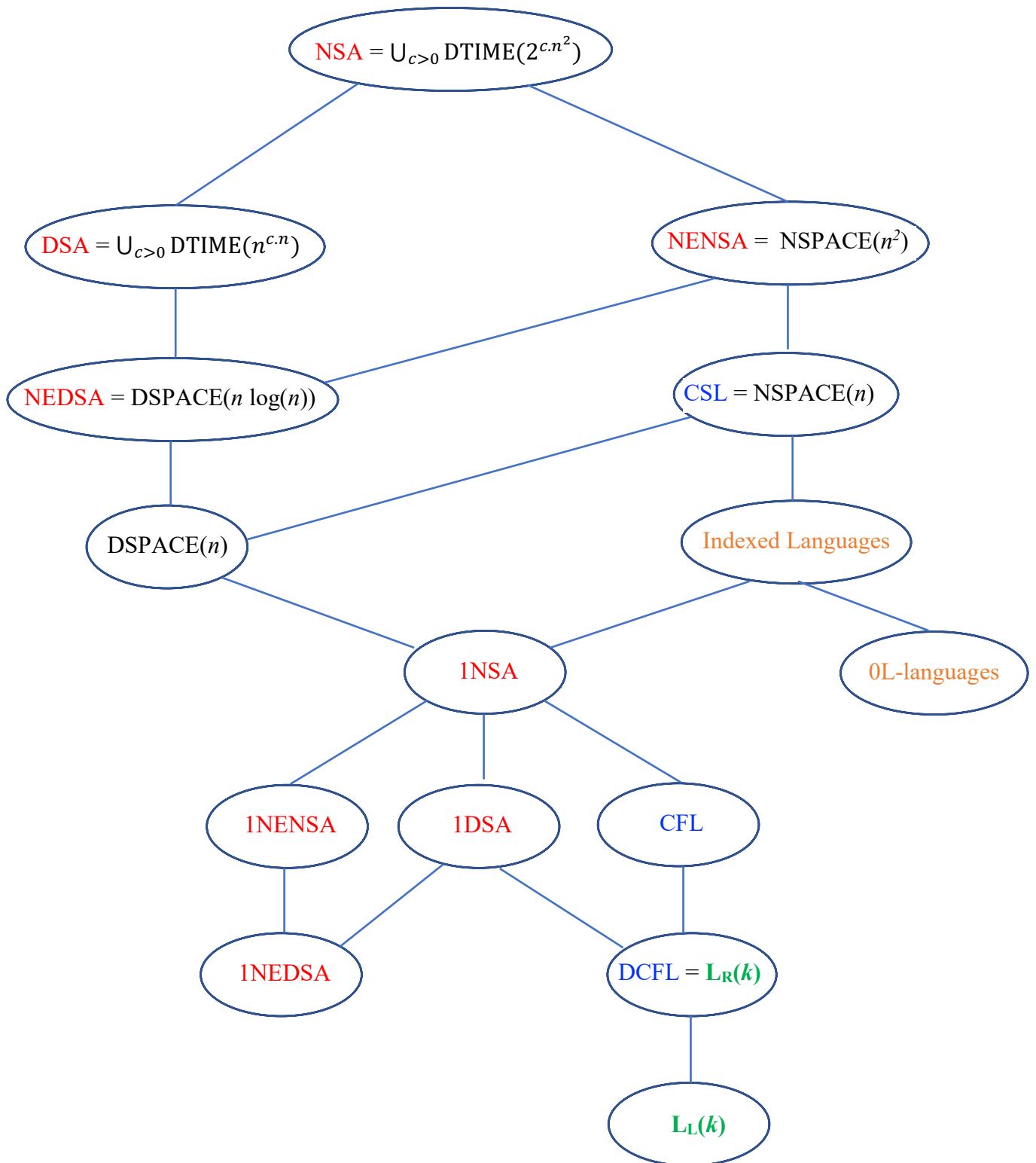
- No início:

— Cabeça de entrada está na extremidade à esquerda da fita.

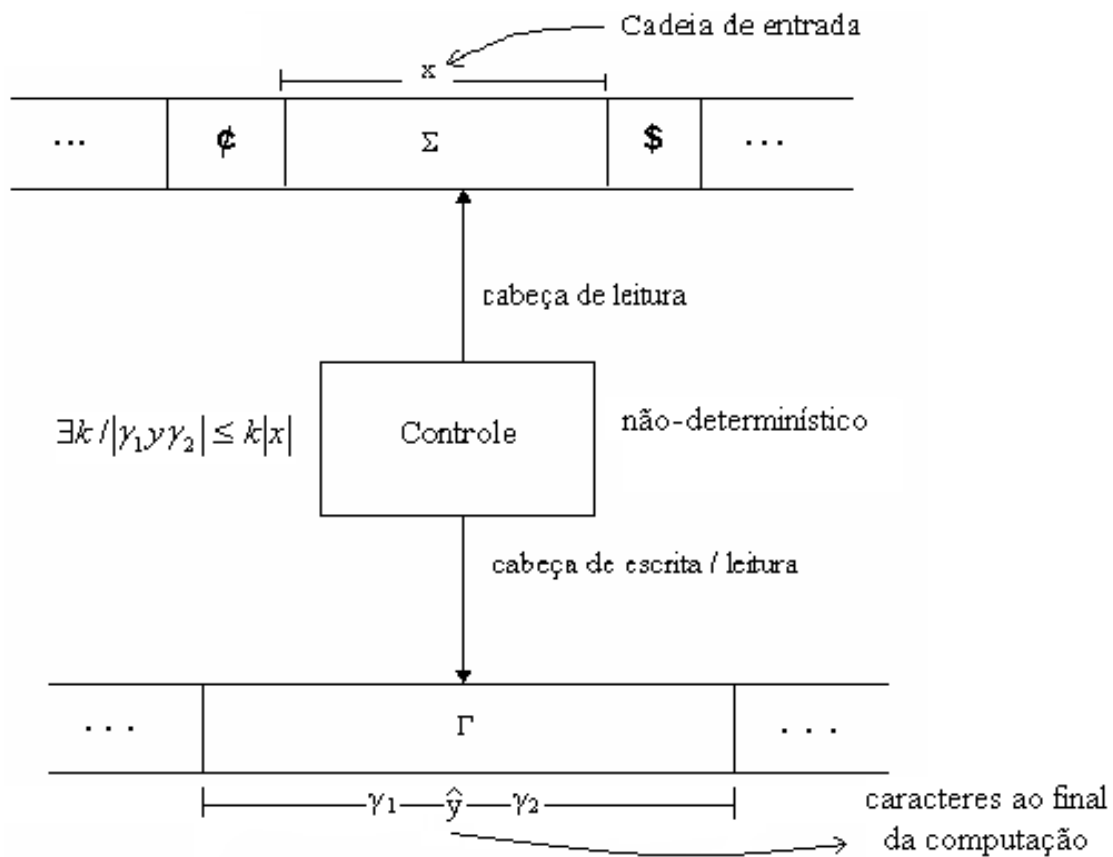
— Controle está em q_0 .

— Pilha tem um único símbolo (Γ_0), definido como de início do processo.





4.4 AUTÔMATO LIMITADO LINEARMENTE (*Linear Bounded Automata*)



- Equivale a uma Máquina de Turing não-determinística, com fita limitada
- Linguagens sensíveis ao contexto:

$$\text{A.L.L.Det.} \stackrel{?}{\equiv} \text{A.L.L. Não-det.}$$