

# Linguagens Formais e Autômatos

Humberto Longo

Instituto de Informática  
Universidade Federal de Goiás

Bacharelado em Ciência da Computação, 2022/1



## Linguagens formais e autômatos

J.D.Ullman, J.E.Hopcroft, R. Motwani - Introdução à Teoria de Autômatos, Linguagens e Computação

"A Teoria dos Autômatos é o estudo dos dispositivos de computação abstratos, ou 'máquinas'. Antes de existirem os computadores, na década de 1930, **Alan Turing** estudou uma máquina abstrata que tinha todas as características dos computadores atuais, pelo menos no que se refere ao quanto eles poderiam calcular. O objetivo de Turing era descrever com exatidão o limite entre o que uma máquina de computação podia fazer e aquilo que ela não podia fazer.

Nas décadas de 1940 e 1950, tipos de máquinas mais simples, que hoje chamamos 'autômatos finitos', foram estudados por diversos pesquisadores. Esses autômatos, propostos originalmente para modelar a função do cérebro (através de modelos conhecidos como 'Redes Neurais'), se mostraram extremamente úteis para uma grande variedade de outros propósitos.

Também no final dos anos 50, o linguista **Noam Chomsky** iniciou o estudo de 'gramáticas' formais. Embora não sejam estritamente máquinas, essas gramáticas têm relacionamentos estreitos com os autômatos abstratos e hoje servem como a base de alguns importantes componentes de *software*, incluindo algumas partes dos compiladores.

Todos estes desenvolvimentos teóricos têm relação direta com aquilo que os cientistas da computação fazem hoje. Alguns conceitos, como autômatos finitos e certos tipos de gramáticas formais, são usados no projeto e na construção de componentes de *software*. Outros conceitos, como a máquina de Turing, ajudam a entender o que podemos esperar de nosso *software*. Em especial, a teoria de problemas intratáveis nos permite deduzir se temos a chance de, ao nos depararmos com um problema, sermos capazes de escrever um programa para resolvê-lo (porque ele não pertence à classe intratável), ou se teremos de descobrir algum modo de contornar o problema intratável: encontrar uma aproximação, usar uma heurística ou empregar algum outro método para limitar o período de tempo que o programa despende para resolver o problema".



## Dúvidas e perguntas frequentes!

### Linguagem???

- ▶ No contexto da disciplina LFA, uma linguagem é simplesmente um conjunto de *palavras (cadeias)*.
- ▶ Dado um alfabeto  $\Sigma$ , então  $\Sigma^*$  é o conjunto de todas as cadeias possíveis contendo apenas os símbolos em  $\Sigma$ .
  - ▶ Por exemplo,  $\{0, 1\}^*$  é o conjunto de todas as sequências binárias de qualquer comprimento.
  - ▶ No entanto, um alfabeto não precisa ser binário e pode ser unário, ternário, ...
- ▶ **Uma linguagem sobre um alfabeto  $\Sigma$  é qualquer subconjunto de  $\Sigma^*$ .**



## Dúvidas e perguntas frequentes!

### Problema???

- ▶ Um problema é uma pergunta, sobre alguma entrada/dado/informação, que gostaríamos que fosse respondida!
  - ▶ Especificamente, um **Problema de Decisão** é uma pergunta que indaga: "A entrada dada satisfaz a propriedade  $X$ ?"
- ▶ Uma linguagem é a concepção formal de um problema: ao se raciocinar teoricamente sobre um problema de decisão, pode-se examinar a linguagem correspondente.
- ▶ Para um problema de decisão  $D$ , a linguagem correspondente é:

$$\mathcal{L} = \{w \mid w \text{ é a codificação de uma entrada } e \text{ do problema } D, \text{ e a resposta para a entrada } e \text{ do problema } D \text{ é "Sim"}\}.$$

- ▶ Determinar se a resposta para uma entrada de um problema de decisão é "Sim" é equivalente a determinar se a codificação dessa entrada, sobre um determinado alfabeto, pertence à linguagem correspondente.



## Dúvidas e perguntas frequentes!

### Algoritmo???

- ▶ Um modo de resolver algum problema passo a passo!
- ▶ Um algoritmo pode ser expresso de muitas maneiras e formalismos distintos.
- ▶ Existem muitos algoritmos diferentes que **resolvem qualquer problema**.



## Dúvidas e perguntas frequentes!

### Máquina de Turing???

- ▶ Análogo formal de um algoritmo!
- ▶ Uma máquina de Turing, para cada cadeia formada com símbolos de um determinado alfabeto, para ou não em um estado de aceitação.
- ▶ Para cada máquina de Turing  $M$ , existe uma linguagem correspondente:
$$\mathcal{L}(M) = \{w \mid M, \text{ ao processar a cadeia } w, \text{ para em um estado de aceitação}\}.$$
- ▶ Há uma diferença sutil entre máquinas de Turing que param com todas as cadeias de entrada (aceitando-as ou rejeitando-as) e máquinas que garantidamente param apenas quando aceitam as entradas!



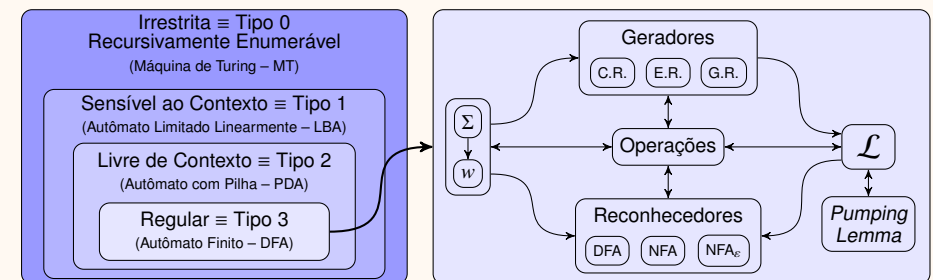
## Dúvidas e perguntas frequentes!

### Máquina de Turing × linguagens???

1. Cada máquina de Turing corresponde a exatamente uma linguagem.
2. Pode haver mais de uma máquina de Turing que corresponda a uma determinada linguagem.
3. **Pode não existir uma máquina de Turing que corresponda a uma determinada linguagem!!!**



## Roteiro



# Linguagem

## Definição?

- ▶ Uma forma de comunicação.
- ▶ Um conjunto de elementos (*símbolos*) e um conjunto de métodos (*regras*) para combinar estes elementos, usado e entendido por uma determinada comunidade:
  - ▶ Linguagens naturais (ou idiomas).
  - ▶ Linguagens de programação.
  - ▶ Protocolos de comunicação.



# Linguagem

- ▶ Uma linguagem é um meio de comunicação, utilizado por elementos de uma determinada comunidade, formado por:
  - ▶ *Conjunto* de palavras.
  - ▶ *Conjunto* de regras gramaticais.
- ▶ Uma linguagem é **formal** quando pode ser representada através de um sistema, com sustentação matemática, formado por:
  - ▶ Sintaxe (estrutura) – foco da disciplina.
  - ▶ Semântica (significado) – não interessa para nós.



# Linguagens formais

- ▶ Mecanismos formais para representação/especificação de linguagens, baseados na chamada “Teoria da Computação”.

**Reconhecedores:** dispositivos formais que servem para verificar se uma palavra pertence ou não à determinada linguagem (autômatos finitos, autômatos de pilha e máquina de Turing).

**Geradores:** dispositivos formais que permitem a geração sistemática de todas as palavras de uma linguagem. Os principais sistemas geradores disponíveis são as gramáticas, onde se destacam as gramáticas de Chomsky.



# Linguagens formais e autômatos

## Objetivo

- ▶ Estudo de modelos matemáticos que possibilitam a especificação e o reconhecimento de linguagens (no sentido formal do termo), suas classificações, estruturas, propriedades, características e inter-relacionamentos.

## Importância na Ciência da Computação

- ▶ Apoio a outros aspectos teóricos da Ciência da Computação (decidibilidade, computabilidade, complexidade computacional, ...).
- ▶ Fundamenta diversas aplicações computacionais (processamento de linguagens, reconhecimento de padrões, modelagem de sistemas, ...).



## Definições básicas

**Alfabeto:** conjunto finito de *símbolos* ou *caracteres*.

▶ Ex:  $\{0, 1\}$ ,  $\{a, b\}$ ,  $\{a, b, c, \dots, z\}$ .

**Cadeia de símbolos:** sequência de zero ou mais símbolos (de um alfabeto) justapostos.

▶ Ex: 01100110, *abababab*..., *abc*...*z*.

**Palavra:** cadeia *finita* de símbolos.

▶ Exemplos: 01, *a*, *b*, *abc*.

## Notação

$\Sigma$  : conjunto de símbolos (um alfabeto).

$\varepsilon$  : cadeia ou palavra vazia.

$\Sigma^*$  : conjunto de todas as palavras possíveis sobre  $\Sigma$ .

$\Sigma^+ = \Sigma^* - \{\varepsilon\}$ .

$|w|$  : comprimento ou tamanho da palavra  $w$  (número de símbolos em  $w$ ).



## Linguagem formal

▶ Dado um alfabeto  $\Sigma$ , uma linguagem  $\mathcal{L}$  em  $\Sigma$  é um conjunto de sequências de símbolos (palavras) do alfabeto.

▶ Se  $\Sigma = \{a, b\}$ , então são linguagens sobre  $\Sigma$ :

▶ Infinitas: o conjunto  $\{\varepsilon, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, \dots\}$  de palíndromos sobre  $\Sigma$ .

▶ Finitas:  $\{a, b, aa, ab, ba, bb\}$ ,  $\{\varepsilon, aaa, bbb\}$ ,  $\{aaa, aab, aba, abb\}$ .

▶ Finitas: o conjunto vazio e o conjunto formado pela palavra vazia.

(**Atenção:**  $\{\}$   $\neq$   $\{\varepsilon\} \neq \varepsilon$ ).

▶ Linguagem  $\Sigma^*$  : conjunto de todas as sequências de símbolos do alfabeto  $\Sigma$ .

▶  $\varepsilon \in \Sigma^*$ .

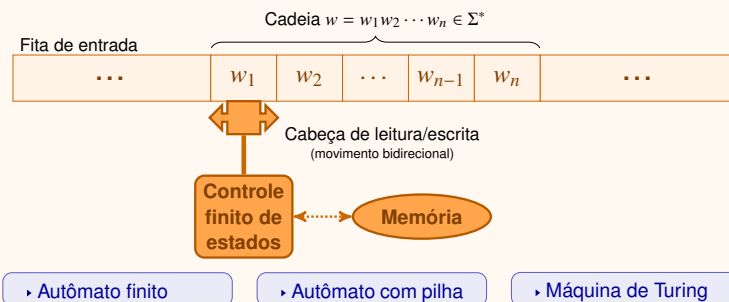
▶  $\mathcal{L} \subseteq \Sigma^*$ , se  $\mathcal{L}$  é uma linguagem em  $\Sigma$ .



## Reconhecedor

▶ Um reconhecedor de uma linguagem  $\mathcal{L} \subseteq \Sigma^*$  é a representação de um procedimento que, quando apresentado a uma cadeia  $w \in \Sigma^*$ :

1. se  $w \in \mathcal{L}$ , para e responde sim, após um número finito de passos; ou
2. se  $w \notin \mathcal{L}$ , para e responde não, após um número finito de passos; ou
3. não para!



## Reconhecedor

▶ Uma configuração do reconhecedor é uma descrição:

1. do estado do controle finito,
2. do conteúdo da fita de entrada e da posição da cabeça de leitura/escrita, e
3. do conteúdo da memória.

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

1. parte de uma configuração inicial;
2. a cabeça de leitura/escrita faz uma sequência finita de movimentos; e
3. termina em uma configuração final.

▶ A linguagem aceita por um reconhecedor  $R$  é:

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



## Geradores – Gramáticas

► Uma gramática (formal) consiste em:

1. um conjunto finito de regras de derivação da forma  $\alpha \rightarrow \beta$ , onde  $\alpha$  e  $\beta$  são sequências de símbolos;
2. um símbolo especial “inicial”  $S$  ( $S$  vem de “sentença”);
3. um conjunto finito de “símbolos” (símbolos terminais) usados para escrever “palavras” válidas ; e
4. um conjunto finito de símbolos auxiliares (símbolos não-terminais).

► Dada uma gramática, para gerar uma “sentença”:

1. começar pelo símbolo inicial  $S$ ;
2. aplicar uma das regras de derivação para formar uma “forma sentencial” ( $S \rightarrow \alpha_1$ );
3. aplicar outra regra para formar uma nova “forma sentencial”  $\alpha_2$ ;
4. repetir o passo 3 até chegar a uma “palavra”  $\alpha_n$  que consiste apenas de símbolos terminais.



### Exemplo 1.1

► Considere a gramática:

$$S \rightarrow AB \quad (1)$$

$$A \rightarrow C \quad (2)$$

$$CB \rightarrow Cb \quad (3)$$

$$C \rightarrow a \quad (4)$$

onde  $\{a, b\}$  são os símbolos terminais e  $\{S, A, B, C\}$  são os não-terminais.

► A palavra “ab” pode ser derivada nessa gramática da seguinte forma:

$$S \Rightarrow AB, \quad \text{a partir de (1)}$$

$$\Rightarrow CB, \quad \text{a partir de (2)}$$

$$\Rightarrow Cb, \quad \text{a partir de (3)}$$

$$\Rightarrow ab, \quad \text{a partir de (4).}$$



## Gramáticas

### Definição 1.2

► Uma gramática é uma 4-upla  $G = (V, \Sigma, P, S)$  onde:

- $V$  é um conjunto finito não vazio de **símbolos**, chamados de **não-terminais**;
- $\Sigma$  é um conjunto finito não vazio de **símbolos**, chamados de **terminais**, tal que  $\Sigma \cap V = \emptyset$ ;
- $S$  é o **símbolo** (não terminal) **inicial** ( $S \in V$ ); e
- $P$  é um conjunto de **regras de derivação** (ou de produção) da forma  $\alpha \rightarrow \beta$ , onde:
  - $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ ;
  - $\beta \in (V \cup \Sigma)^*$ .



## Gramáticas

### Exemplo 1.3

►  $G = (V, \Sigma, P, S)$  onde:

$$V = \{S, N_p, V_p, V_i, V_t, V_s, C, P_f\}$$

$$\Sigma = \{\text{João, Maria, sorri, ama, pensa, que}\}$$

$$P = \left\{ \begin{array}{l} S \rightarrow N_p V_p P_f \\ V_p \rightarrow V_i \mid V_t N_p \mid V_s C S \\ N_p \rightarrow \text{João} \mid \text{Maria} \\ V_i \rightarrow \text{sorri} \\ V_t \rightarrow \text{ama} \\ V_s \rightarrow \text{pensa} \\ C \rightarrow \text{que} \\ P_f \rightarrow . \end{array} \right.$$



## Gramáticas

### Exemplo 1.4

- ▶  $G = (V, \Sigma, P, S)$ , onde:
  - ▶  $V = \{A, S\}$ ,
  - ▶  $S$ : símbolo (não-terminal) inicial,
  - ▶  $\Sigma = \{a, b\}$ ,
  - ▶  $P = \{S \rightarrow ab, S \rightarrow aASb, S \rightarrow bSb, AS \rightarrow bSb, A \rightarrow \varepsilon, aASAb \rightarrow aa\}$ .

### Exemplo 1.5

- ▶  $\mathcal{L} = \{0^n 1^n \mid n \geq 1\} = \{01, 0011, 000111, \dots\}$ .
- ▶ Gramática que gera  $\mathcal{L}$ :
  - ▶  $\Sigma = \{0, 1\}$ ,
  - ▶ Símbolo inicial:  $S$ ,
  - ▶  $V = \{S\}$ ,
  - ▶  $P = \{S \rightarrow 0S1, S \rightarrow 01\}$ .



## Gramáticas

- ▶ No contexto usual de linguagens de programação:
  - $\Sigma$  (símbolos)  $\equiv$  palavras reservadas, variáveis definidas, símbolos numéricos, operadores, delimitadores ...
  - $w$  (cadeia)  $\equiv$  programa sintaticamente correto,
  - $\mathcal{L}$  (linguagem)  $\equiv$  conjunto de programas sintaticamente corretos,
  - $G$  (gramática)  $\equiv$  estrutura sintática dos programas.



## BNF – Backus Normal Form

### Backus-Naur Form

- ▶ Outro modo de se representar algumas gramáticas (GLC).
- ▶ Símbolo  $\rightarrow$  é substituído por  $::=$ .
- ▶ Símbolos não terminais são ladeados por  $\langle \rangle$ .
- ▶ As várias regras de derivação de um mesmo símbolo não terminal  $A$  são escritas como:  $\langle A \rangle ::= \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ .
- ▶ Os símbolos  $\langle, \rangle, ::=$  e  $|$  formam a metalinguagem, ou seja, são símbolos que não fazem parte da linguagem mas ajudam a descrevê-la!



## BNF – Backus Normal Form

### Exemplo 1.6

- ▶  $G = (V, \Sigma, P, S)$  onde:

$V = \{\langle \text{sentença} \rangle, \langle \text{sn} \rangle, \langle \text{sv} \rangle, \langle \text{artigo} \rangle, \langle \text{substantivo} \rangle, \langle \text{verbo} \rangle\};$

$\Sigma = \{o, a, \text{peixe}, \text{mordeu}, \text{isca}\};$

$S = \langle \text{sentença} \rangle$  e

$$P = \left\{ \begin{array}{l} \langle \text{sentença} \rangle ::= \langle \text{sn} \rangle \langle \text{sv} \rangle \\ \langle \text{sn} \rangle ::= \langle \text{artigo} \rangle \langle \text{substantivo} \rangle \\ \langle \text{sv} \rangle ::= \langle \text{verbo} \rangle \langle \text{sn} \rangle \\ \langle \text{artigo} \rangle ::= o \mid a \\ \langle \text{verbo} \rangle ::= \text{mordeu} \\ \langle \text{substantivo} \rangle ::= \text{peixe} \mid \text{isca} \end{array} \right\}.$$



## BNF – Backus Normal Form

### Exemplo 1.7 (Subconjunto da gramática da linguagem Pascal)

#### ► Regras léxicas:

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid \dots \mid 9$   
 $\langle \text{number} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{number} \rangle \mid - \langle \text{number} \rangle$   
 $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle$   
 $\langle \text{letter} \rangle ::= a \mid b \mid c \mid \dots \mid z$   
 $\langle \text{operator} \rangle ::= + \mid - \mid * \mid /$   
 $\langle \text{relation} \rangle ::= <= \mid >= \mid == \mid > \mid < \mid !=$



## BNF – Backus Normal Form

### Exemplo 1.7 (Subconjunto da gramática da linguagem Pascal)

#### ► Regras sintáticas:

$\langle \text{program} \rangle ::= \text{program } \langle \text{identifier} \rangle; \langle \text{block} \rangle$   
 $\langle \text{block} \rangle ::= \langle \text{declaration seq} \rangle \text{ begin } \langle \text{command seq} \rangle \text{ end.}$   
 $\langle \text{declaration seq} \rangle ::= \varepsilon \mid \langle \text{declaration} \rangle \langle \text{declaration seq} \rangle$   
 $\langle \text{declaration} \rangle ::= \text{var } \langle \text{variable list} \rangle : \langle \text{type} \rangle;$   
 $\langle \text{type} \rangle ::= \text{boolean} \mid \text{integer} \mid \text{real} \mid \text{string}$   
 $\langle \text{command seq} \rangle ::= \langle \text{command} \rangle \mid \langle \text{command} \rangle \langle \text{command seq} \rangle$   
 $\langle \text{command} \rangle ::= \langle \text{variable} \rangle = \langle \text{expression} \rangle \mid$   
 $\quad \text{if } \langle \text{boolean expr} \rangle \text{ then } \langle \text{command seq} \rangle \text{ end} \mid$   
 $\quad \text{while } \langle \text{boolean expr} \rangle \text{ do } \langle \text{command seq} \rangle \text{ end} \mid$   
 $\quad \dots$



## Gramáticas

### Definição 1.8

- Uma gramática é uma 4-upla  $G = (V, \Sigma, P, S)$  onde:
  - $V$  é um conjunto finito não vazio de **símbolos**, chamados de **não-terminais**;
  - $\Sigma$  é um conjunto finito não vazio de **símbolos**, chamados de **terminais**, tal que  $\Sigma \cap V = \emptyset$ ;
  - $S$  é o **símbolo** (não terminal) **inicial** ( $S \in V$ ); e
  - $P$  é um conjunto de **regras de derivação** (ou de produção) da forma  $\alpha \rightarrow \beta$ , onde:
    - $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ ;
    - $\beta \in (V \cup \Sigma)^*$ .



## Tipos de gramáticas

- Dependendo das regras de derivação  $\alpha \rightarrow \beta$ , as gramáticas podem ser caracterizadas em:
  - Irrestrita**: sem restrição a  $\alpha$  e  $\beta$ ;
  - Sensível ao contexto**: as regras são da forma  $rAs \rightarrow r\gamma s$ , onde  $A$  é um não-terminal,  $r$ ,  $s$  e  $\gamma$  são cadeias de terminais e não-terminais e  $\gamma$  não é vazio.
  - Livre de contexto**: as regras são forma  $A \rightarrow \gamma$ , onde  $A$  é um não-terminal e  $\gamma$  é uma cadeia (potencialmente vazia) de terminais e não-terminais.
  - Regular**: as regras são forma  $A \rightarrow aB$  ou  $A \rightarrow a$ , onde  $A$  e  $B$  são não-terminais e  $a$  é um símbolo (potencialmente vazio) terminal.



## Tipos de gramáticas

### Definição 1.9 (Gramática irrestrita (GI) ou Tipo-0)

- ▶  $G = (V, \Sigma, P, S)$ , onde:
  - $V$  = conjunto finito não vazio de símbolos, chamados de não-terminais;
  - $\Sigma$  = conjunto finito não vazio de símbolos, chamados de terminais, tal que  $\Sigma \cap V = \emptyset$ ;
  - $S$  = símbolo (não terminal) inicial ( $S \in V$ ); e
  - $P$  = conjunto de regras (de produção) da forma  $\alpha \rightarrow \beta$ , onde:
    - ▶  $\alpha \in (V \cup \Sigma)^+$ ,
    - ▶  $\beta \in (V \cup \Sigma)^*$ .
- ▶ Gramática *semi-thue* ou gramática de estrutura de frase.



## Tipos de gramáticas

### Definição 1.10 (Gramática sensível ao contexto (GSC) ou Tipo-1)

- ▶  $G = (V, \Sigma, P, S)$ , com toda produção  $\alpha \rightarrow \beta$  de  $P$  da forma:
  - ▶  $\alpha \in (V \cup \Sigma)^*(V \cup \Sigma)(V \cup \Sigma)^*$ ,
  - ▶  $\beta \in (V \cup \Sigma)^*(V \cup \Sigma)^+(V \cup \Sigma)^*$ ,
  - ▶  $S \rightarrow \varepsilon$ .
  - ▶ Se esta regra de produção ocorrer,  $S$  não pode aparecer no lado direito de qualquer outra produção.



## Tipos de gramáticas

### Definição 1.11 (Gramática livre de contexto (GLC) ou Tipo-2)

- ▶  $G = (V, \Sigma, P, S)$ , com toda produção  $\alpha \rightarrow \beta$  de  $P$  da forma:
  - ▶  $\alpha \in V$ ,
  - ▶  $\beta \in (V \cup \Sigma)^*$ .



## Tipos de gramáticas

### Definição 1.12 (Gramática regular (GR) ou Tipo-3)

- ▶  $G = (V, \Sigma, P, S)$ , com toda produção  $\alpha \rightarrow \beta$  de  $P$  da forma:
  - ▶  $\alpha \in V$ ,
  - ▶  $\beta \in \Sigma V \cup V \Sigma \cup \Sigma \cup \varepsilon$ .





## Tipos de Gramáticas

### Observações

- ▶ Tipos 2 e 3: só existe 1 não-terminal no lado esquerdo das produções.
- ▶ Tipo-1: pode haver mais de 1 não-terminal no lado esquerdo, mas só 1 é transformado em cada regra de produção.
- ▶ Tipo-0: qualquer quantidade de não-terminais no lado esquerdo das produções.

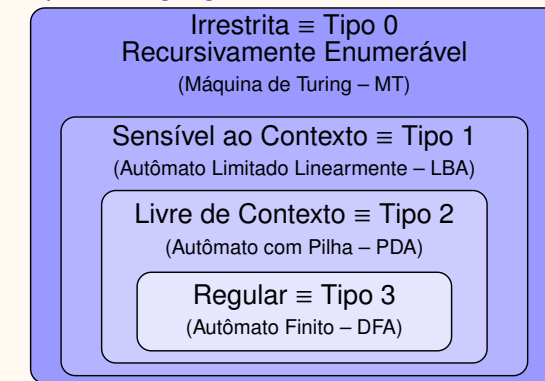
### Definição 1.13

Uma linguagem  $\mathcal{L}$  é do Tipo- $x$ , se existe uma gramática  $G$  do Tipo- $x$  tal que  $\mathcal{L} = \mathcal{L}(G)$  (isto é, a linguagem  $\mathcal{L}$  é gerada pela gramática  $G$ ).



## Hierarquia de Chomsky

### Tipos de linguagens e reconhecedores



## Hierarquia de Chomsky

### Tipos de linguagens e reconhecedores

**Tipo-0** Produções arbitrárias.

**M.T.** Memória arbitrariamente grande.

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

**L.B.A.** Memória proporcional ao comprimento da cadeia de entrada.

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








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

**Tipo-3** Produções da forma (unitária)  $A \rightarrow wB$  ou  $A \rightarrow w$  tal que  $A, B \in V$ ,  $w \in (\Sigma \cup \{\epsilon\})$ .

**D.F.A.** Sem memória.



## Livros texto

-  **R. P. Grimaldi**  
*Discrete and Combinatorial Mathematics – An Applied Introduction.*  
Addison Wesley, 1994.
-  **D. J. Velleman**  
*How To Prove It – A Structured Approach.*  
Cambridge University Press, 1996.
-  **J. E. Hopcroft; J. Ullman.**  
*Introdução À Teoria de Autômatos, Linguagens e Computação.*  
Ed. Campus.
-  **T. A. Sudkamp.**  
*Languages and Machines – An Introduction to the Theory of Computer Science.*  
Addison Wesley Longman, Inc. 1998.
-  **J. Carroll; D. Long.**  
*Theory of Finite Automata – With an Introduction to Formal Languages.*  
Prentice-Hall, 1989.
-  **M. Sipser.**  
*Introduction to the Theory of Computation.*  
PWS Publishing Company, 1997.
-  **H. R. Lewis; C. H. Papadimitriou**  
*Elementos de Teoria da Computação.*  
Bookman, 2000.

