

# Teoria dei linguaggi

## Indice

<b>1. Introduzione .....</b>	<b>2</b>
1.1. Storia .....	2
1.2. Ripasso .....	2
<b>2. Gerarchia di Chomsky .....</b>	<b>3</b>
2.1. Rappresentazione .....	3
2.2. Gerarchia .....	3
<b>3. Grammatiche di tipo 3 .....</b>	<b>4</b>
3.1. Automi a stati finiti .....	4
3.1.1. Definizione informale .....	4
3.1.2. Definizione formale .....	4
3.1.3. Linguaggio .....	5
3.1.4. Rappresentazione grafica .....	5
3.2. Automi a stati finiti non deterministici .....	5
3.2.1. Definizione informale .....	5
3.2.2. Definizione formale .....	5
3.3. Distinguibilità .....	6

# 1. Introduzione

## 1.1. Storia

Un **linguaggio** è uno strumento di comunicazione usato da membri di una stessa comunità, ed è composto da due elementi:

- **sintassi**: insieme di simboli (o *parole*) che devono essere combinati con una serie di regole
- **semantica**: associazione frase-significato

Per i linguaggi naturali è difficile dare delle regole sintattiche: vista questa difficoltà, nel 1956 **Noam Chomsky** introduce il concetto di **grammatiche formali**, che si servono di regole matematiche per la definizione della sintassi di un linguaggio

Il primo utilizzo dei linguaggi risale agli stessi anni con il **compilatore Fortran**, ovvero un traduttore da un linguaggio di alto livello ad uno di basso livello, ovvero il *linguaggio macchina*

## 1.2. Ripasso

Un **alfabeto** è un insieme *non vuoto e finito* di simboli, di solito indicato con  $\Sigma$  o  $\Gamma$

Una **stringa** (o **parola**) è una sequenza *finita* di simboli appartenenti a  $\Sigma$

Data una parola  $w$ , possiamo definire:

- $|w|$  *numero di caratteri* di  $w$
- $|w|_a$  *numero di occorrenze* della lettera  $a \in \Sigma$  in  $w$

Una parola molto importante è la **parola vuota**  $\varepsilon$ , che, come dice il nome, ha simboli, ovvero  $|\varepsilon| = 0$

L'insieme di tutte le possibili parole su  $\Sigma$  è detto  $\Sigma^*$

Un'importante operazione sulle parole è la **concatenazione** (o *prodotto*), ovvero se  $x, y \in \Sigma^*$  allora la concatenazione  $w$  è la parola  $w = xy$

Questo operatore di concatenazione:

- *non è commutativo*, infatti  $w_1 = xy \neq yz = w_2$  in generale
- *è associativo*, infatti  $(xy)z = x(yz)$

La struttura  $(\Sigma^*, \cdot, \varepsilon)$  è un **monoide** libero generato da  $\Sigma$

Vediamo ora alcune proprietà delle parole

- **prefisso**:  $x$  si dice *prefisso* di  $w$  se esiste  $y \in \Sigma^*$  tale che  $xy = w$ 
  - **prefisso proprio** se  $y \neq \varepsilon$
  - **prefisso non banale** se  $x \neq \varepsilon$
  - il numero di prefissi è uguale a  $|w| + 1$
- **suffisso**:  $y$  si dice *suffisso* di  $w$  se esiste  $x \in \Sigma^*$  tale che  $xy = w$ 
  - **suffisso proprio** se  $x \neq \varepsilon$
  - **suffisso non banale** se  $y \neq \varepsilon$
  - il numero di suffissi è uguale a  $|w| + 1$
- **fattore**:  $y$  si dice *fattore* di  $w$  se esistono  $x, z \in \Sigma^*$  tali che  $xyz = w$ 
  - il numero di fattori è al massimo  $\frac{|w| \cdot (|w| + 1)}{2} + 1$
- **sottosequenza**:  $x$  si dice *sottosequenza* di  $w$  se  $x$  è ottenuta eliminando 0 o più caratteri da  $w$ 
  - un *fattore* è una sottosequenza ordinata

Un **linguaggio**  $L$  definito su un alfabeto  $\Sigma$  è un qualunque sottoinsieme di  $\Sigma^*$

## 2. Gerarchia di Chomsky

### 2.1. Rappresentazione

Vogliamo rappresentare in maniera finita un oggetto infinito come un linguaggio

Abbiamo a nostra disposizione due modelli molto potenti

- **generativo**: date delle regole, si parte da *un certo punto* e si generano tutte le parole di quel linguaggio con le regole date
- **ricognoscitivo**: si usano dei *modelli di calcolo* che prendono in input una parola e dicono se appartiene o meno al linguaggio

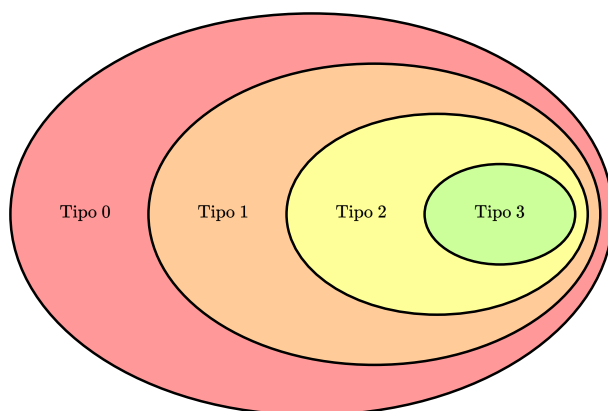
Considerando il linguaggio sull'alfabeto  $\{(, )\}$  delle parole ben bilanciate, proviamo a dare due modelli

- **generativo**: a partire da una sorgente  $S$  devo applicare delle regole per derivare tutte le parole appartenenti a questo linguaggio
  - la parola vuota  $\varepsilon$  è ben bilanciata
  - se  $x$  è ben bilanciata, allora anche  $(x)$  è ben bilanciata
  - se  $x, y$  sono ben bilanciate, allora anche  $xy$  sono ben bilanciate
- **ricognoscitivo**: abbiamo una *black-box* che prende una parola e ci dice se appartiene o meno al linguaggio (in realtà potrebbe non terminare mai la sua esecuzione)
  - $|x|_(< = |x|_>$
  - per ogni prefisso,  $|x|_(< \geq |x|_>$

### 2.2. Gerarchia

Negli anni 50 Noam Chomsky studia la generazione dei linguaggi formali e crea una **gerarchia di grammatiche formali**

- **tipo 0**: grammatiche che generano tutti i linguaggi, sono senza restrizioni e come modello equivalente hanno le **macchine di Turing**
- **tipo 1**: grammatiche *context-sensitive* (dipendenti dal contesto), come modello equivalente hanno le **Linear Bounded Automata**
- **tipo 2**: grammatiche *context-free* (libere dal contesto), hanno come modello equivalente gli **automi a pila**
- **tipo 3**: grammatiche regolari, hanno come modello equivalente gli **automi a stati finiti**



### 3. Grammatiche di tipo 3

#### 3.1. Automi a stati finiti

Gli **automi a stati finiti** sono un modello riconoscitivo usato per caratterizzare i *linguaggi regolari*

##### 3.1.1. Definizione informale

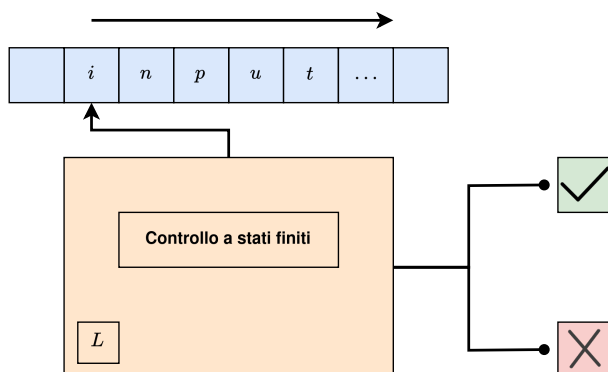
Gli automi a stati finiti sono delle macchine molto semplici: hanno un **controllo a stati finiti** che legge l'input da un **nastro**, formato da una serie di celle, ognuna delle quali contiene un carattere dell'input

Per leggere l'input si utilizza una **testina di lettura**, posizionata inizialmente sulla cella più a sinistra (ovvero sul primo carattere di input), e poi spostata iterazione dopo iterazione da sinistra verso destra

Il controllo a stati finiti, prima della lettura dell'input, è allo **stato iniziale**

Ad ogni iterazione, a partire dallo stato corrente e dal carattere letto dal nastro, ci si muove con la testina a destra sul simbolo successivo e si cambia stato

Quando si arriva alla fine del nastro, in base allo stato corrente dell'automa, quest'ultimo risponde "sì", ovvero la parola in input appartiene al linguaggio, oppure "no", ovvero la parola in input non appartiene al linguaggio



##### 3.1.2. Definizione formale

Un automa è una tupla  $\{Q, \Sigma, \delta, q_I, F\}$ , con

- $Q$  insieme degli stati
- $\Sigma$  alfabeto di input
- $\delta : Q \times \Sigma \rightarrow Q$  funzione di transizione
- $q_I \in Q$  stato iniziale
- $F \subseteq Q$  insieme degli stati finali

La parte dinamica dell'automa è la **funzione di transizione** che, dati lo stato iniziale e un simbolo del linguaggio, calcola lo stato successivo

Possiamo estendere la funzione di transizione affinché utilizzi una parola del linguaggio, ovvero definiamo  $\bar{\delta} : Q \times \Sigma^* \rightarrow Q$  tale che

- $\bar{\delta}(q, \epsilon) = q \quad \forall q \in Q$
- $\bar{\delta}(q, wa) = \delta(\bar{\delta}(q, w), a) \quad \forall q \in Q, w \in \Sigma^*, a \in \Sigma$

Per semplicità useremo  $\delta$  al posto di  $\bar{\delta}$  nella notazione

### 3.1.3. Linguaggio

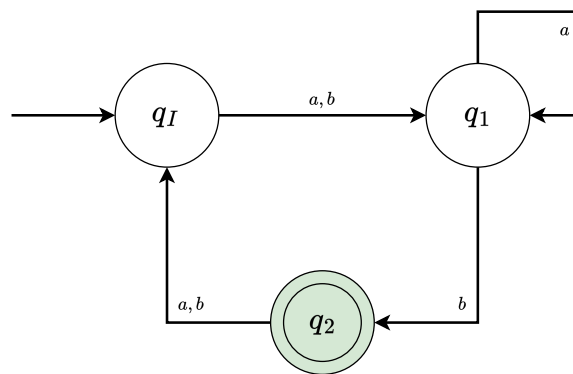
Chiamiamo  $L(A) = \{w \in \Sigma^* \mid \delta(q_I, w) \in F\}$  il **linguaggio riconosciuto dall'automa**, ovvero l'insieme delle parole che applicate alla funzione di transizione, a partire dallo stato iniziale, mi mandano in uno stato finale

### 3.1.4. Rappresentazione grafica

Possiamo vedere un automa come un grafo, dove

- gli **stati** sono nodi etichettati con il nome dello stato
- le **transizioni** sono archi orientati ed etichettati con la lettera dell'alfabeto che causa quella transizione

Lo **stato iniziale** è indicato con una freccia entrante nello stato, mentre gli **stati finali** sono nodi doppiamente cerchiati



## 3.2. Automi a stati finiti non deterministici

Questi particolari automi sono utili nei linguaggi  $R_n$  (fatti come esempio a lezione) perché necessitano di molto meno stati ( $n - 1$ ) rispetto ad un automa deterministico ( $2^n$ )

### 3.2.1. Definizione informale

Gli **automi a stati finiti non deterministici (NFA)** sono particolari automi che hanno *almeno* uno stato dal quale escono 2 o più archi con la stessa lettera

Negli automi **deterministici (DFA)** invece da *ogni* stato esce al più un arco con la stessa lettera

La differenza principale sta nella complessità computazionale: se negli automi deterministici devo controllare se la parola ci porta in uno stato finale, negli automi non deterministici devo controllare se tra *tutti* i possibili cammini dell'**albero di computazione** ne esiste uno che ci porta in uno stato finale

Possiamo vedere il non determinismo come *una scommessa che va sempre a buon fine*

### 3.2.2. Definizione formale

Un automa non deterministico differisce da un automa deterministico solo per la funzione di transizione: infatti, quest'ultima diventa  $\delta : Q \times \Sigma \longrightarrow \mathbb{P}(Q)$ , ovvero ritorna un elemento dell'**insieme delle parti** di  $Q$ , quindi un sottoinsieme di stati nei quali possiamo finire applicando un carattere di  $\Sigma$  allo stato corrente

Come prima, definiamo l'estensione della funzione di transizione come la funzione  $\bar{\delta} : Q \times \Sigma^* \longrightarrow \mathbb{P}(Q)$  tale che

- $\bar{\delta}(q, \varepsilon) = \{q\}$
- $\bar{\delta}(q, wa) = \bigcup_{r \in \bar{\delta}(q, w)} \delta(r, a)$

Cambia anche il linguaggio riconosciuto dall'automa: infatti,  $L(A)$  diventa  $\{w \in \Sigma^* \mid \bar{\delta}(q_I, w) \cap F \neq \emptyset\}$

### 3.3. Distinguibilità

Dato un linguaggio  $L \subseteq \Sigma^*$ , due parole  $x, y \in \Sigma^*$  sono **distinguibili** rispetto ad  $L$  se  $\exists z \in \Sigma^*$  tale che  $(xz \in L \wedge yz \notin L) \vee (xz \notin L \wedge yz \in L)$

**Teorema 3.3.1** Siano  $L \subseteq \Sigma^*$  un linguaggio,  $X \subseteq \Sigma^*$  un insieme di parole tutte distinguibili tra loro rispetto ad  $L$  e  $A$  un automa DFA per  $L$ , allora  $A$  non può avere meno di  $|X|$  stati

#### Dimostrazione

Per assurdo, sia  $X = \{x_1, \dots, x_k\}$  insieme di  $k$  parole distinguibili tra loro rispetto ad  $L$  e che  $A$  abbia meno di  $k$  stati

Partendo dallo stato corrente  $q_I$  ho due situazioni

- leggendo il carattere  $x_i$ , con  $1 \leq i \leq k - 2$ , l'automa finisce nello stato  $q_i$
- leggendo i caratteri  $x_{k-1}$  e  $x_k$  l'automa finisce nello stato  $q_t$ , visto che  $A$  ha meno di  $k$  stati

Prendiamo ora una parola  $z$  e la applichiamo allo stato  $q_t$ , che ci porta in uno stato  $r$  qualsiasi, ma questo è assurdo: infatti, abbiamo due parole distinguibili ( $x_{k-1}$  e  $x_k$ ) che però sono entrambe accettate o entrambe rifiutate, in base allo stato  $r$  □