Alfabeto: Insieme finito di simboli Σ={a1, a2, … , an}
Parola: (o stringa) su un alfabeto Σ è una sequenza finita di simboli appartenenti a Σ
Σ^* = Insieme delle parole su Σ compresa ε ---- Σ^+ = Insieme delle parole su Σ senza ε
prefisso di y se y=x·z **suffisso** di y se y=zx **fattore** di y se y=z·x·w
Linguaggio formale: Un linguaggio L sull'alfabeto Σ è un insieme di parole di Σ*, cioè un qualunque sottoinsieme (finito o infinito) L ⊆ Σ*.
Operazioni tra linguaggi:
Unione: $A \cup B = \{w \in \Sigma^* \mid w \in A \lor w \in B\}$ **Intersezione**:$A \cap B = \{w \in \Sigma^* \mid w \in A \land w \in B\}$
Complemento: $A^c = \{w \in \Sigma^* \mid w \notin A\}$ **Prodotto**: dati i linguaggi L1 e L2 sull'alfabeto Σ , il loro prodotto è il linguaggio L1 · L2 = {w∈ Σ| w= xy , con x ∈ L1 e y ∈ L2 }
Potenza: Poiché il prodotto è associativo, possiamo definire la potenza L^k, dove L^0 = { ε } e
$L^{k+1} = L^k \cdot L$
Chiusura di Kleene: dato un linguaggio L, la sua chiusura è il linguaggio $L^* = L^0 \cup L^1 \cup \ldots \cup L^k \cup \ldots = \cup_{k=0}^{\infty} L^k$

$$L^+ = \bigcup_{k=1}^{\infty} L^k$$

dove $L^n = L \cdot L^{n-1}$e, per convenzione, $L^0 = \varepsilon$.
Codice: Un linguaggio L è un codice quando ogni parola in L^+è ottenuta in un unico modo come prodotto di parole di L. **Codice prefisso:** L è un codice prefisso quando è un codice e ogni parola in L non è prefisso di altre parole in L
Procedura: una sequenza finita di passi che può o meno terminare con un risultato.
$F_w(x)$ indica il risultato dell'esecuzione della procedura **w** su input **x**:
$F_w(x)$ ↓ procedura termina; $F_w(x) = 1$ uscita 1; $F_w(x) = 0$ uscita 0;
$F_w(x)$ ↑ indica che la procedura **w** su input **x** genera una computazione che non termina (un loop).
Algoritmo: un algoritmo è semplicemente una procedura **w** che su qualsiasi ingresso **x** genera una computazione che termina (dando come risultato, nel nostro caso, 0 oppure 1).
Linguaggio ricorsivo: Un linguaggio L è detto ricorsivo(o decidibile) se esiste un algoritmo ***w*** tale che $F_w(x) = 1\ se\ x \in L, 0\ se\ x \notin L$; tale algoritmo è anche detto **riconoscitore**, e calcola dunque la **Funzione caratteristica del linguaggio L**, cioè la funzione $\chi_L$tale che $\chi_L(x)$= 1 se x ∈ L, 0 altrimenti. (ammette un sistema riconoscitivo: automa). AMMETTE RICONOSCITORE
Linguaggio ricorsivamente numerabile: Un linguaggio L è detto ricorsivamente numerabile(o semidecidibile) se esiste una procedura ***w*** tale che $F_w(x) = 1\ se\ x \in L$, mentre $F_w(x)$ ↑ altrimenti. (ammette un sistema generativo: grammatica)
Teorema: $\boldsymbol{L\ ricorsivo\ \Rightarrow L^c\ ricorsivo}$
Infatti, se L è ricorsivo esisterà un algoritmo **w** tale che $F_w(x) = 1\ se\ x \in L$, mentre $F_w(x) = 0\ se\ x \notin L$; possiamo costruire allora un algoritmo **w2** tale che, su input **x,** calcola $F_w(x)$ e, se **w** ritorna 1, **w2** ritorna 0, mentre se **w** ritorna 0, **w2** ritorna 1.
Teorema: $\boldsymbol{L\ ricorsivo\ \Rightarrow L\ ricorsivamente\ numerabile}$
Infatti se L è ricorsivo esiste un algoritmo ***w*** che calcola la sua funzione caratteristica; costruiamo ora una nuova procedura che prima simula **w** e poi, se l'uscita è 0, genera una computazione che non termina. Questo prova che L è anche ricorsivamente numerabile.
Questo fa notare il fatto che un algoritmo può sempre essere peggiorato in una procedura.
$\boldsymbol{L\ ricorsivamente\ numerabile\ \Rightarrow L\ ricorsivo}$**?** **NO**
Infatti non è possibile sostituire una computazione che non termina con una che termina e dà come risultato 0.
Dunque esistono linguaggi che sono ricorsivamente numerabili, ma non ricorsivi. Per dare un esempio è necessario richiamare il concetto di interprete.
Interprete: un interprete è un programma **u** che accetta in ingresso due parole **x, w** ∈{0,1}* (due parole binarie) e simula l'esecuzione della procedura <u>codificata con **w**</u> su input **x**; in simboli:
$F_u(x\$w) = F_w(x)\ se\ w$ è $un\ programma$ $F_u(x\$w) = \perp\ altrimenti$
Ciò è possibile in quanto w è sia un programma(semanticamente) sia una parola binaria(sintatticamente).
nota bene: l'esempio di linguaggio ricorsivamente numerabile ma non ricorsivo fatto dalla prof. Palano è il seguente: L = {x : $F_u(x\ \$x)$ ↓}

cioè il linguaggio L delle parole tali per cui l'esecuzione della procedura codificata binariamente nella parola stessa, avendo in input la parola stessa, termina. Il linguaggio è detto:

Linguaggio dell'arresto ristretto: $\boldsymbol{D} = \{x \in \{0,1\}^* | \ F_u(x \ \$x) \downarrow\}$
Il suo complemento: $\boldsymbol{D^c} = \{x \in \{0,1\}^* | \ F_u(x \ \$x) \uparrow\}$
Proprietà: <u>$\boldsymbol{D}$ è ricorsivamente enumerabile</u>; $\boldsymbol{D}$ non è ricorsivo; $\boldsymbol{D^c}$ non è ricorsivamente enumerabile

```
RICNUM(x) {
        y = Fu(x$x);
        return 1;
} // return 1 sse appartiene al linguaggio e e fa loop se non appartiene
```

Calcolo logico: dato da una funzione che permette di "dimostrare" tutte e sole le affermazioni vere(f) di un linguaggio L.
Calcolo logico per il linguaggio L: dato un linguaggio $\boldsymbol{L} \subseteq \boldsymbol{\Sigma}^*$, un calcolo logico per L è un calcolo logico V corretto e completo per L. In tal caso sarà $\boldsymbol{L} = \{\boldsymbol{x} \mid \exists \boldsymbol{d} \ \ \boldsymbol{V}(\boldsymbol{x}, \boldsymbol{d}) = \boldsymbol{1}\}$.
Grammatica: una grammatica G è una quadrupla $<\Sigma, Q, P, S>$ dove:
1. **Σ** e **Q** sono due alfabeti finiti disgiunti, rispettivamente di simboli terminali e metasimboli;
2. **P** è un insieme finito di regole di produzione;
3. **S** è un elemento in Q, detto **assioma** o simbolo di partenza.

Un linguaggio ammette più grammatiche che lo generano se **è ricorsivamente numerabile**.
Due grammatiche G1 e G2 sono dette **equivalenti** se generano lo stesso linguaggio, cioè se L(G1)=L(G2).
Teorema: *Il linguaggio L è generato da una grammatica* ⇔ *L è ricorsivamente numerabile.*
Questo significa che, se per un linguaggio L esiste un calcolo logico corretto e completo, allora L è generabile da una grammatica. <u>Le grammatiche risultano dunque sistemi formali per esprimere calcoli logici</u>.

Grammatica tipo 0	**Grammatica tipo 1**	**Grammatica tipo 2**	**Grammatica tipo 3**
Regole di produzione arbitrarie	$\alpha \to \beta$ e |l(β)| ≥ l(α)| regola $S \to \varepsilon$	$\alpha \to \beta$ tale che α è un metasimbolo	$A \to \sigma B$, $A \to \sigma$ $A \to \varepsilon$
Genera L di tipo 0 <u>L ricorsivamente enumerabili</u>	Genera L di tipo 1 <u>L dipendenti da contesto</u>	Genera L di tipo 2 <u>L liberi da contesto (acontestuali)</u>	Genera L di tipo 3 <u>L regolari</u>
L ammette un calcolo logico ⇔ *L è ricorsivamente numerabile*		*L generato da G 2* ⇔L è riconosciuto da un riconoscitore a pila.	⇔ L è riconosciuto da un automa a stati finiti
L ricorsivamente enumerabile ⇔ ammette G che lo genera		L libero da contesto ⇔ è *accettato da un riconoscitore a pila*	Teorema di Kleene: L è denotato da una espressione regolare ⇔L è riconosciuto da un automa a stati finiti
		Alberi di derivazione è possibile utilizzare una pila per simulare una derivazione left-most in una grammatica di tipo 2 in fng	Generato da una G lineare a dx

Teorema di inclusione degli Rk

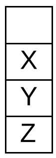
$$\boldsymbol{R_3 \subset R_2 \subset R_1 \subset R_0}$$

Se A e B sono linguaggi regolari allora anche il complemento di A, e A intersecato B lo sono .

Grammatica ambigua / non ambigua: una grammatica G = <Σ, Q, P, S> di tipo 2 è detta **ambigua** se esiste una parola w∈L(G) che ammette due diversi alberi di derivazione; viceversa, una grammatica G = <Σ, Q, P, S> di tipo 2 è detta **non ambigua** se ogni parola w∈L(G) ammette un unico albero di derivazione.
Automa a stati: Un automa a stati è un sistema A = < Q , Σ, δ, q_0, F> dove:
1) **Q** è un insieme di stati;
2) **Σ** è un alfabeto finito;
3) $\delta: \Sigma \times Q \to Q$ è la **funzione di transizione**;
4) q_0∈ Q è lo **stato iniziale**;
5) **F** ⊆ Q è l'insieme degli <u>stati finali</u> che definisce una funzione λ : Q →{ 0,1 } , dove: λ (q) = 1 se q ∈ F, altrimenti λ (q) = 0.
Se l'insieme Q è finito, l'automa è detto a stati finiti.
Automa a stati finiti non deterministico (NFA): sistema A = <Σ, Q, q0, R, F> dove **Q è un insieme finito di stati**, **Σ è un alfabeto** e **R è l'insieme delle relazioni di transizione**, non si parla quindi più di funzione bensì di relazione di transizione non deterministica:

- **R (q, σ, p) = 0** "non si può raggiungere p da q tramite la lettura di σ "
- **R (q, σ, p) = 1** "si può raggiungere p da q tramite la lettura di σ "

Automa a stati finiti deterministico (DFA): è un particolare automa non deterministico A = <Σ, Q, q0, R, F> dove è sempre possibile sostituire la funzione di transizione δ con una relazione di transizione R
Teorema: Per ogni L riconosciuto da un NFA esiste un DFA che lo riconosce.
Parola ambigua: Una parola è ambigua se ammette due alberi di derivazione diversi.
Grammatica ambigua: Una grammatica G si dice ambigua se genera almeno una parola ambigua.
Linguaggio inerentemente ambiguo: Un linguaggio si dice inerentemente ambigui se ogni G che lo genera è ambigua.
Forma Normale di Chomsky (FNC): $A \to BC \quad A \to \sigma$ con $A, B, C \in V \quad e \quad \sigma \in T$
Forma Normale di Greibach (FNG): $A \to \sigma W$ con $A \in V$, $\sigma \in T \quad e \quad W \in V^*$
Pila: Memoria ad accesso limitato con politica (LIFO - *Last in first out*). Si può lavorare solo su x perché è quello più in alto.



Riconoscitore a pila:
Un riconoscitore a pila è una tupla $A = (\Sigma, K, S, \delta)$ dove:

- Σ alfabeto di input
- K alfabeto della pila $\Sigma \cap K = \emptyset$
- S simbolo iniziale della pila
- δ funzione di evoluzione della pila $\delta: K \times \Sigma \to 2^{k^*} \quad \delta(x, \sigma) = \{w_1, w_2, \ldots, w_s\}$ con $w_j \in K^*$

Si indica che:

- X è letto in cima alla pila **(TOP)**
- σè letto sul nastro di input
- X viene cancellato dalla pila **(POP)**
- Viene scelto un $W_j \in K^*$ in maniera **NON DETERMINISTICA** da inserire nella pila (**PUSH)**

Criterio di accettazione: La parola x si dirà accettata se nel grafo di computazione di x esiste un cammino da S a ε
Pumping Lemma: Esprime una condizione necessaria per i linguaggi di tipo 2.
L non soddisfa il lemma ⇒L non è di tipo 2.
L soddisfa il lemma ⇒L può essere di tipo 2 o no.
Per ogni L di tipo 2 esiste una costante H tale che per ogni $z \in L$ con |z| > H esiste una scomposizione in $uvwxy = z$ che soddisfa:

1. $|vx| \geq 1$
2. $|vwx| \leq H$
3. $\forall k \geq 0 \quad uv^k wx^k y \in L$