

5 - Grammatiche e macchine

(Forniti gentilmente dal pasticcino alla crema di Pice)

Gerarchia di Chomsky

Sia $G = (X, V, S, P)$ una grammatica, dalla sua definizione si ha:

$$P = \{v \rightarrow w \mid v \in (X \cup V)^+ \text{ e } v \text{ contiene almeno un NT}, w \in (X \cup V)^*\}$$

Classificazione

A seconda delle restrizioni imposte sulle regole di produzione, si distinguono le varie classi di grammatiche:

- **Tipo 0:** quando le stringhe che appaiono nella produzione non sono soggette ad alcuna limitazione
- **Tipo 1 - Dipendenti da contesto:** quando le produzioni sono limitate alla forma
 1. $yAz \rightarrow ywz$ con $A \in V$, $y, z \in (X \cup V)^*$, $w \in (X \cup V)^*$
 2. $S \rightarrow \lambda$, purché S non compaia nella parte destra di alcuna produzione
- **Tipo 2 - Libera da contesto:** quando le produzioni sono limitate alla forma

$$v \rightarrow w \text{ con } v \in V$$
- **Tipo 3 - Lineare destra:** quando le produzioni sono limitate alla forma
 1. $A \rightarrow bC$ con $A, C \in V$ e $b \in X$
 2. $A \rightarrow b$ con $A \in V$ e $b \in X \cup \{\lambda\}$

Una grammatica di tipo '3' è detta **lineare destra** perché il simbolo NT , se c'è, compare nella parte destra della produzione.

Un linguaggio generato da una tale grammatica è detto di tipo '3' o lineare a destro

Teorema della gerarchia

Il **Teorema della Gerarchia di Chomsky** dimostra che le quattro classi di linguaggi formali (classificate come tipo 0, 1, 2 e 3) formano una gerarchia strettamente inclusiva, dove ogni classe è un sottoinsieme proprio della precedente.

Denotiamo con \mathcal{L}_i (insieme dei linguaggi di tipo i) il seguente insieme:

$$\mathcal{L}_i = \{L \subset X^* \mid L = L(G), G \text{ di tipo } i\}$$

La gerarchia di Chomsky è una gerarchia in senso stretto di classi di linguaggi:

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$$

Dimostrazione

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2$$

- **Inclusione:** Ogni grammatica di tipo 3 (lineare destra) è anche di tipo 2 (libera da contesto), poiché le produzioni $A \rightarrow bC$ o $A \rightarrow b$ soddisfano la definizione di grammatica libera da contesto.
- **Inclusione stretta:** Esiste almeno un linguaggio di tipo 2 che non è di tipo 3.

Esempio:

$$L = a^n b^n \mid n > 0$$

Questo linguaggio è generato da una grammatica libera da contesto ma non può essere generato da una grammatica lineare destra.

- $\mathcal{L}_3 \subseteq \mathcal{L}_2$ discende dalle definizioni di linguaggio di tipo 3 e di grammatica di tipo 2. Infatti, si osserva facilmente che ogni grammatica di tipo 3 è anche una grammatica di tipo 2

$$\mathcal{L}_2 \subsetneq \mathcal{L}_1$$

- **Inclusione:** Ogni grammatica libera da contesto è anche dipendente da contesto, con l'eccezione delle produzioni $A \rightarrow \lambda$ (dove $A \neq S$).
Lo definiamo come:

$$\forall L : L \in \mathcal{L}_2 \Leftrightarrow \exists G, G \text{ è C.F.} : L = L(G)$$

Tuttavia, il **Lemma della stringa vuota** permette di eliminare queste produzioni senza alterare il linguaggio generato, rendendo la grammatica di tipo 1

- **Inclusione stretta:** Esiste almeno un linguaggio di tipo 1 che non è di tipo 2.

Esempio:

$$L = a^n b^n c^n \mid n > 0$$

Questo linguaggio è dipendente da contesto ma non libero da contesto.

Lemma della stringa vuota

Sia $G = (X, V, S, P)$ una grammatica C.F. con almeno una λ -produzione, allora esiste una grammatica C.F. G' tale che:

1. $L(G) = L(G')$ (con le due grammatiche che si equivalgono)
2. Se $\lambda \notin L(G)$ allora in G' non esistono produzioni del tipo $A \rightarrow \lambda$
3. Se $\lambda \in L(G)$ allora in G' esiste un'unica produzione $S' \rightarrow \lambda$, ove S' è il simbolo iniziale di G' ed S' non compare nella parte destra di alcuna produzione di G'

Se G ha almeno una λ -produzione, utilizziamo il Lemma della stringa vuota per determinare una grammatica C.F. G' equivalente a G , ma priva di λ -produzioni (al più, in G' compare la produzione $S' \rightarrow \lambda$, ed S' non compare nella parte destra di alcuna produzione di G'). G' è di tipo 1, dimostrando che $\mathcal{L}_2 \subseteq \mathcal{L}_1$

$$\mathcal{L}_1 \subsetneq \mathcal{L}_0$$

- **Inclusione:** Ogni grammatica di tipo 1 è anche di tipo 0, poiché le produzioni dipendenti da contesto sono un caso particolare delle produzioni non ristrette (tipo 0).
- **Inclusione stretta:** Esistono linguaggi ricorsivamente enumerabili (tipo 0) che non sono dipendenti da contesto. La dimostrazione formale richiede nozioni avanzate come le macchine di Turing.

Operazioni sui linguaggi

Siano L_1 ed L_2 due linguaggi definiti su uno stesso alfabeto X ($L_1, L_2, \subseteq X^*$), le operazioni attuabili su essi sono:

Unione insiemistica

L'unione di due linguaggi è l'insieme di tutte le stringhe che appartengono **almeno a uno** dei due linguaggi, anche se definiti su alfabeti diversi.

$$L_1 \cup L_2 = \{w | w \in L_1 \vee w \in L_2\}$$

Concatenazione

La concatenazione genera tutte le possibili combinazioni prima una stringa di L_1 , poi una di L_2

$$L_1 \cdot L_2 = \{w | w = w_1 w_2, w \in L_1 \vee w \in L_2\}$$

Iterazione

L'iterazione è un'operazione unaria, si parte da un linguaggio e si fa una generalizzazione dalle parole di uno stesso linguaggio. Si ottiene un linguaggio infinito, definito con la formula:

$$L_1^* = \{w | w = w_1 w_2 \dots w_n, n \geq 0 \text{ e } \forall i : w_i \in L_1\}$$

L^* = potenza all'ennesimo di tutti i linguaggi

Complemento

Il complemento è l'insieme di tutte le parole meno l'insieme di partenza, definito con la formula

$$\overline{L_1} = X^* - L_1$$

Intersezione

L'intersezione tra due linguaggi è l'operazione di prendere due elementi in comune tra due linguaggi, quindi stringhe presenti in entrambi.

$$L_1 \cap L_2 = \{w | w \in L_1 \wedge w \in L_2\}$$

L'intersezione, la concatenazione e l'unione sono dette operazioni binarie, in quanto prevedono l'uso di due insiemi. Complemento e iterazione sono invece operazioni unarie.

Proprietà

L'operazione di concatenazione gode delle seguenti proprietà:

Dati $L_1, L_2, L_3 \subseteq X^*$ ($\equiv L_1, L_2, L_3 \in 2^{X^*}$), si possono avere:

- **Associatività**, l'ordine in cui concateni tre linguaggi non cambia il risultato:
 $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$
- **Non commutativa**, l'ordine dei linguaggi **influenza** il risultato: $L_1 \cdot L_2 \neq L_2 \cdot L_1$
- **Elemento neutro**: $L_1 \cdot \{\lambda\} = \{\lambda_1\} \cdot L_1 = L_1$

$(2^{X^*}, \cdot)$ è anch'esso un **monoide**, in quanto presenta:

- $L_1 \cdot \emptyset = \emptyset \cdot L_1 = \emptyset$, (\emptyset) è l'**elemento assorbente**
- Se un linguaggio contiene la stringa vuota ($\lambda \in L_1$ oppure $\lambda \in L_2$), valgono queste inclusioni:
 - $L_2 \subseteq L_1 \cdot L_2$
 - $L_2 \subseteq L_2 \cdot L_1$
 - $L_1 \subseteq L_1 \cdot L_2$
 - $L_1 \subseteq L_2 \cdot L_1$

Potenza di un linguaggio

Sia L un linguaggio definito su un alfabeto X , dicesi **potenza n-esima** di L , e si denota con L^n , $n \geq 0$, il seguente linguaggio:

$$L^n = \begin{cases} \{\lambda\} & \text{se } n=0 \\ L^{n-1} \cdot L & \text{altrimenti} \end{cases}$$

Si ha dunque che:

$L^+ = \bigcup_{i \geq 1} L^i$, (unione di tutte le potenze maggiori di 1, quindi deve avere almeno una concatenazione)

Si può definire l'unione di tutte le potenze anche con la stringa vuota:

$$L^* = \{\lambda\} \cup L^+ = \bigcup_{i \geq 0} L^i$$

Proprietà di chiusura delle classi di linguaggi

Un linguaggio definito su un alfabeto è un insieme di parole, una classe di linguaggi è un insieme di linguaggi.

Definizione di chiusura

Si suppone di avere un operazione binaria, definita su una coppia di linguaggi

Teorema di chiusura

La classe dei linguaggi di tipo $i, i = 0, 1, 2, 3$ è chiusa rispetto alle operazioni di unione, concatenazione ed iterazione.

Dati quindi due linguaggi quindi, dopo aver effettuato una di queste operazioni tra i due linguaggi, si ottiene sempre un linguaggio della stessa classe.

Dimostrazione del teorema

Lo schema generale della dimostrazione è il seguente:

- consideriamo una certa operazione, denotata con α ;
- date G_1 e G_2 , costruiamo una nuova grammatica G :

$$G = (X, V, S, P)$$

Per la quale si dimostra che:

- se G_1 e G_2 sono di tipo i , allora G è di tipo i ;
- $L(G) = \alpha(L_1, L_2)$

Assumendo che non abbiano non terminali in comune.

Poniamo che: $V = V_1 \cup V_2 \cup \{S\}$

Lo schema generale della dimostrazione è il seguente:

- consideriamo una certa operazione, denotata con α
- costruiamo una nuova grammatica G per cui dimostriamo che
 - se G_1 e G_2 sono di tipo i , allora G è di tipo i ;
 - $L(G) = \alpha(L_1, L_2)$

Unione (per \mathcal{L}_2):

Costruiamo la grammatica $G_3 = (X, V, S, P_3)$ ove:

$$P_3 = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2$$

Osserviamo che, se G_1 e G_2 sono entrambe di tipo 2, lo è anche G_3 in quanto abbiamo aggiunto due produzioni libere da contesto:

- $S \rightarrow S_1$
- $S \rightarrow S_2$

Nel primo caso si avrà la derivazione: $S \Rightarrow S_1 \xRightarrow{*} w_1 \in L_1$

Nel secondo caso si avrà la derivazione: $S \Rightarrow S_2 \xRightarrow{*} w_2 \in L_2$

E' pertanto dimostrato che \mathcal{L}_2 è chiusa rispetto all'unione.

Unione (per \mathcal{L}_3):

Se G_1 e G_2 sono di tipo '3', G_3 non è lineare destra, perché le produzioni che abbiamo introdotto non sono lineari destre: $S \rightarrow S_1 \quad S \rightarrow S_2$

Per risolvere il problema dobbiamo introdurre produzioni lineari destre che simulino i passi iniziali delle derivazioni in G_1 ed in G_2 .

Costruiamo la grammatica $G_4 = (X, V, S, P_4)$ ove P_4 :

- per ogni regola $S_1 \rightarrow w \in P_1$ aggiungiamo a P_4 la regola: $S \rightarrow w$
- per ogni regola $S_2 \rightarrow w \in P_2$ aggiungiamo a P_4 la regola: $S \rightarrow w$

$$P_4 = \{S \rightarrow w | S_1 \rightarrow w \in P_1\} \cup \{S \rightarrow w | S_2 \rightarrow w \in P_2\} \cup P_1 \cup P_2.$$

Tutte le regole di P_4 sono lineari destre in quanto abbiamo aggiunto regole la cui parte destra rispetta il vincolo delle grammatiche di tipo '3': G_4 è di tipo '3'.

Concatenazione (per \mathcal{L}_2):

Costruiamo la grammatica $G_5 = (X, V, S, P_5)$, nella quale $P_5 = \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2$.

Osservazione:

- se G_1 e G_2 sono di tipo '2', anche G_5 è di tipo '2'
- $L(G_5) = L_1 \cdot L_2$, poiché tutte le derivazioni sono del tipo:

$$S \Rightarrow S_1 S_2 \xRightarrow{*} w_1 S_2 \xRightarrow{*} w_1 w_2 \in L_1 \cdot L_2$$

È pertanto dimostrato che ℓ_2 è chiusa rispetto alla concatenazione.

Concatenazione (per \mathcal{L}_3):

Osservazione:

Data una grammatica di tipo '3', ogni forma di frase derivata dal suo simbolo iniziale ha due peculiarità:

1. in essa compare al più un NT
2. se in essa compare un NT, questo è il simbolo più a destra

Quindi, se G_1 e G_2 sono di tipo '3', G_5 non è di tipo '3', per la presenza della produzione: $S \rightarrow S_1 S_2$. C'è pertanto bisogno di una nuova grammatica in grado di simulare l'effetto di tale produzione.

Osserviamo che per generare una parola del linguaggio $L_1 \cdot L_2$ senza usare la produzione $S \rightarrow S_1 S_2$, dovremmo "chiudere" (ovvero sostituire l'ultimo non terminale della forma di frase, ottenendo dunque una stringa terminale) la derivazione di una parola di L_1 (ovvero derivare S_1 fino ad ottenere solo caratteri terminali) per poi generare l'assioma di L_1 .

Studiando una generica derivazione di una parola di L_1 notiamo che:

$$S_1 \Rightarrow x_1 A \Rightarrow x_1 x_2 A \xRightarrow{*} x_1 x_2 \dots x_{n-1} N \Rightarrow x_1 x_2 \dots x_n$$

Avremmo quindi due possibili derivazioni del nonterminale N:

$$1. N \rightarrow \alpha$$

$$2. N \rightarrow \lambda$$

Caso 1:

Le regole del tipo $N \rightarrow \alpha$ vengono modificate in: $N \rightarrow \alpha S_2$, e quindi:

$$S_1 \Rightarrow x_1 A \Rightarrow x_1 x_2 A \xRightarrow{*} x_1 x_2 \dots x_{n-1} N \Rightarrow x_1 x_2 \dots x_{n-1} \alpha \text{ diventa:}$$

$$S_1 \Rightarrow x_1 A \Rightarrow x_1 x_2 A \xRightarrow{*} x_1 x_2 \dots x_{n-1} N \Rightarrow x_1 x_2 \dots x_{n-1} \alpha S_2$$

Dall'assioma di S_2 si potrà successivamente generare una parola $w_2 \in L_2$, ottenendo una parola $\in L_1 \cdot L_2$

Caso 2:

Le regole del tipo $N \rightarrow \lambda$ non possono essere trasformate in $N \rightarrow S_2$ in quanto non sarebbe lineare destra.

Per tale regola dobbiamo risalire al non terminale che ha generato N ovvero alle regole del tipo: $M \rightarrow \alpha N$, con poi $N \rightarrow \lambda$, chiudendo quindi la derivazione di una parola di L_1 .

In pratica si deve intervenire su ogni λ -produzione e relative regole che generano il non terminale presente nella parte sinistra della λ -produzione.

Costruiamo quindi la grammatica $G_6 = (X, V - \{S\}, S_1, P_6)$. Le sue produzioni sono del tipo:

$$P_6 = \{A \rightarrow bB \mid A \rightarrow bB \in P_1\} \cup \{A \rightarrow bS_2 \mid A \rightarrow b \in P_1, b \neq \lambda\} \cup \{A \rightarrow bS_2 \mid B \rightarrow \lambda \in P_1, A \rightarrow bB \in P_1\}$$

Questa grammatica tuttavia ha un problema, in quanto non è possibile derivare solo le parole di L_2 . (Dovremmo dunque implementare una sorta di $S_1 \rightarrow \lambda$, non implementabile in quanto non sarebbe lineare destra.)

Per risolvere tale problema non dovremmo fare altro che innescare anche da S_1 la derivazione di parole di S_2 . Aggiungiamo quindi una nuova regola alle produzioni:

$$P_6 = \{A \rightarrow bB \mid A \rightarrow bB \in P_1\} \cup \{A \rightarrow bS_2 \mid A \rightarrow b \in P_1, b \neq \lambda\} \cup \{A \rightarrow bS_2 \mid B \rightarrow \lambda \in P_1, A \rightarrow bB \in P_1\} \cup \{S_1 \rightarrow S_1 S_2\}$$

Con l'ultima regola non andiamo a fare altro che a trascrivere S_1 con i non terminali di L_2 qualora ci sia una λ -produzione.

È pertanto dimostrato che L_3 è chiusa rispetto alla concatenazione

Iterazione (per \mathcal{L}_2)

Costruiamo la grammatica G_7 partendo da G_1 : $G_7 = (X, V_1 \cup \{S\}, S, P_7)$

dove $P_7 = \{S \rightarrow \lambda, S \rightarrow S_1 S\} \cup P_1$.

Osserviamo che se G_1 è di tipo 2, lo è anche G_3 in quanto abbiamo aggiunto due produzioni libere da contesto

Iterazione (per \mathcal{L}_3)

Anche qui nasce il problema che $S \rightarrow S_1 S$ non è lineare destra.

Dobbiamo costruire una nuova grammatica G_8 il cui assioma S produca λ e tutte le parti destre dell'assioma di G_1 , in modo da garantire che ogni derivazione di G_8 inizi esattamente come una di G .

Osservazione preliminare:

L'operatore \mathcal{L}_3 produce stringhe costituite da **concatenazioni di zero o più stringhe di** $L(G_1)$, quindi serve una grammatica che permetta sia di generare una singola stringa di G_1 , sia di ripeterla quante volte si vuole, sia di fermarsi (producendo λ).

Algoritmo per costruire $G_8 = (V_8, \Sigma, P_8, S)$ a partire da $G_1 = (V_1, \Sigma, P_1, S_1)$:

1. Aggiungiamo una nuova variabile $S \notin V_1$ come nuovo assioma.
2. Poniamo $V_8 = V_1 \cup \{S\}$
3. Inizializziamo $P_8 = \{S \rightarrow \lambda\}$
4. Per ogni produzione $S_1 \rightarrow w \in P_1$, aggiungiamo **due produzioni** a P_8 :
 - $S \rightarrow w$
 - $S \rightarrow wS$
 In simboli:
 - $P_8 = \{S \rightarrow \lambda\} \cup \{S \rightarrow w, S \rightarrow wS \mid S_1 \rightarrow w \in P_1\}$

Verifica della correttezza: due casi**1. Caso 1: G_1 non produce λ**

Allora $L(G_8) = L(G_1)^*$

Ogni stringa generata da G_8 è ottenuta concatenando zero o più stringhe generate da G_1 .

2. Caso 2: G_1 produce λ

Allora λ è già incluso in $L(G_8)$ tramite la regola $S \rightarrow \lambda$,
ma attenzione: se $S_1 \rightarrow \lambda$ è una regola in G_1 , allora:

- $S \rightarrow \lambda$ (già presente)
- $S \rightarrow \lambda S$ (aggiunta come $S_1 \rightarrow \lambda \Rightarrow S \rightarrow \lambda S$)

Quindi bisogna notare che:

- $S \Rightarrow \lambda$
- $S \Rightarrow \lambda S \Rightarrow \lambda \lambda S \Rightarrow \dots$

Ovvero, si generano **infinite derivazioni** della parola vuota, ma l'insieme delle stringhe generate rimane $L(G_1)^*$, come desiderato.

Altri teoremi di chiusura

1. La classe dei linguaggi lineari destri (tipo '3') è chiusa rispetto al complemento ed all'intersezione
2. La classe dei linguaggi liberi da contesto (tipo '2') non è chiusa rispetto al complemento ed all'intersezione
3. La classe dei linguaggi dipendenti da contesto (tipo '1') è chiusa rispetto al complemento e all'intersezione
4. La classe dei linguaggi di tipo '0' non è chiusa rispetto al complemento

Per ℓ_1 e ℓ_0 non lo dimostriamo

Dimostrazioni:

1. Per la classe di tipo 3 (lineari destri):

Assumiamo dimostrata la chiusura di ℓ_3 rispetto al complemento.

Secondo le Leggi di De Morgan:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

Allora, poiché i linguaggi regolari (tipo 3) sono chiusi per:

- **complemento:** \mathcal{L}_3 è chiuso rispetto al complemento
- **unione:** \mathcal{L}_3 è chiuso rispetto all'unione

ne segue che anche l'intersezione è chiusa:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} \in \ell_3$$

Poiché tutte le operazioni usate sono chiuse in \mathcal{L}_3 , anche $L_1 \cap L_2$ appartiene a \mathcal{L}_3 .

2. Per la classe di tipo 2 (liberi da contesto):

Non vale la chiusura né per il **complemento** né per l'**intersezione**.

È possibile dimostrarlo con un controesempio:

- Sia $L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$
(libero da contesto)
- Sia $L_2 = \{a^m b^n c^n \mid m, n \geq 0\}$
(libero da contesto)

Allora:

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$$

che **non è un linguaggio libero da contesto**.

Quindi la classe dei CFL (tipo 2) **non è chiusa** rispetto all'intersezione.

Inoltre, se fosse chiusa rispetto al **complemento**, allora anche l'intersezione lo sarebbe (per De Morgan), il che porterebbe a una contraddizione.

Quindi anche **la chiusura per complemento fallisce**.

Riflessione

Definizione di stringa riflessa

Sia w una parola su un alfabeto $X = \{x_1, x_2, \dots, x_k\}$, con $w = x_{i_1} x_{i_2} \dots x_{i_{n-1}} x_{i_n}$, si dice **stringa riflessa** o **riflessione** di w la stringa:

$$w^R = x_{i_n} x_{i_{n-1}} \dots x_{i_2} x_{i_1}$$

Operazione di riflessione

Sia w una parola su un alfabeto $X = \{x_1, x_2, \dots, x_k\}$ e sia w^R la stringa riflessa di w , l'operazione di trasformazione si chiama **operazione di riflessione**

Definizione di parola palindromica

Un **palindromo** o **parola palindromica** è una parola la cui lettera a ritroso riproduce la parola di partenza:

$$w \text{ palindromo} \stackrel{def}{\iff} w = w^R$$

Un palindromo è dunque una parola che coincide con la sua riflessione

I palindromi possono essere di due tipi:

- **Lunghezza pari**, con asse di simmetria costituito dalla parola vuota
- **Lunghezza dispari**, con asse di simmetria costituito da uno dei simboli dell'alfabeto

Più precisamente si ha la seguente caratterizzazione:

Teorema sulla parola palindroma

Sia w una parola su un alfabeto X , w è un palindromo se e solo se:

$$w = axa^R, x \in X \cup \{\lambda\}$$

Teorema sulla riflessione

La classe dei linguaggi non contestuali (tipo '2') è **chiusa** rispetto all'operazione di **riflessione**. In altre parole, se un linguaggio L è generato da una grammatica libera da contesto (CFG), allora anche il linguaggio riflesso $L^R = \{w^R \mid w \in L\}$ è libero da contesto.

Dimostrazione

Sia $G_1 = (X, V_1, S_1, P_1)$ una grammatica CFG che genera L . Costruiamo una nuova grammatica G_9 come segue:

$$G_9 = (X, V_1, S_1, P_9), \quad \text{dove:}$$

$$P_9 = \{A \rightarrow \alpha^R \mid A \rightarrow \alpha \in P_1\}.$$

Passaggi:

1. Inversione delle Produzioni:

Per ogni produzione $A \rightarrow \alpha$ in P_1 , aggiungiamo a P_9 la produzione $A \rightarrow \alpha^R$.

- *Esempio:* Se P_1 contiene $A \rightarrow aBb$, allora P_9 conterrà $A \rightarrow bBa$.

2. Preservazione del Tipo '2':

Poiché G_1 è CFG, ogni produzione ha un singolo non terminale a sinistra (es. $A \rightarrow \alpha$).

Invertire α non cambia questo vincolo, dunque G_9 rimane di tipo '2'.

3. Correttezza:

- Ogni derivazione in G_1 che genera w corrisponde a una derivazione in G_9 che genera w^R , grazie all'inversione delle produzioni.
- *Struttura induttiva*: Se $S_1 \Rightarrow^* w$ in G_1 , allora $S_1 \Rightarrow^* w^R$ in G_9 , poiché ogni passo di derivazione riflette l'ordine dei simboli.

Esempio

- **Grammatica Originale (G_1):**
 $S \rightarrow aSb \mid \lambda$ genera $L = \{a^n b^n \mid n \geq 0\}$.
- **Grammatica Riflessa (G_9):**
 $S \rightarrow bSa \mid \lambda$ genera $L^R = \{b^n a^n \mid n \geq 0\}$.