Tutorato di Automi e Linguaggi Formali

Homework 5: Grammatiche Context-Free e Forma Normale di Chomsky

Gabriel Rovesti
Corso di Laurea in Informatica - Università degli Studi di Padova

Tutorato 5 - 14-04-2025

1 Grammatiche Context-Free (CFG)

Esercizio 1. Scrivere una grammatica context-free per ciascuno dei seguenti linguaggi:

a) $L_1 = \{a^n b^m c^n \mid n, m \geq 1\}$

b) $L_2 = \{w \in \{a, b\}^* \mid w \text{ contiene lo stesso numero di } a \text{ e } b\}$

c) $L_3 = \{a^i b^j c^k \mid i = j \text{ oppure } j = k, \text{ dove } i, j, k \geq 1\}$

d) $L_4 = \{a^n b^m \mid n \neq m, \text{ dove } n, m \geq 1\}$

e) $L_5 = \{wcw^R \mid w \in \{a, b\}^*\}$, dove w^R è il reverse di w

Nota: Per ogni linguaggio, spiegare brevemente la logica dietro la grammatica proposta.

Soluzione. **a)** $L_1 = \{a^n b^m c^n \mid n, m \geq 1\}$

La grammatica context-free per L_1 è:

$$S \to aSc \mid aT$$
$$T \to bT \mid b$$

La logica di questa grammatica è:

- La produzione $S \to aSc$ genera coppie di a e c in modo bilanciato (uno a sinistra, uno a destra)
- La produzione $S \to aT$ assicura almeno un a e un c (attraverso le produzioni precedenti), e passa alla generazione dei b

- Le produzioni $T \to bT \mid b$ generano almeno un b

Questa grammatica genera esattamente le stringhe nella forma $a^n b^m c^n$ con $n, m \geq 1$.
b) $L_2 = \{w \in \{a, b\}^* \mid w$ **contiene lo stesso numero di** a **e** $b\}$
La grammatica context-free per L_2 è:

$$S \to aSb \mid bSa \mid SS \mid \varepsilon$$

La logica di questa grammatica è:

- Le produzioni $S \to aSb$ e $S \to bSa$ aggiungono sempre un'occorrenza bilanciata di a e b
- La produzione $S \to SS$ permette di combinare due sottostringhe bilanciate
- La produzione $S \to \varepsilon$ fornisce il caso base (zero a e zero b)

Questa grammatica genera esattamente le stringhe con lo stesso numero di a e b.
c) $L_3 = \{a^i b^j c^k \mid i = j$ **oppure** $j = k,$ **dove** $i, j, k \geq 1\}$
La grammatica context-free per L_3 è:

$$\begin{aligned} S &\to A \mid B \\ A &\to aAb \mid abC \\ B &\to aBc \mid Dbc \\ C &\to cC \mid c \\ D &\to aD \mid a \end{aligned}$$

La logica di questa grammatica è:

- Il non-terminale S sceglie tra le due condizioni ($i = j$ o $j = k$)
- Il non-terminale A genera stringhe con $i = j$ seguito da $k \geq 1$ occorrenze di c
- Il non-terminale B genera stringhe con $j = k$ preceduto da $i \geq 1$ occorrenze di a
- I non-terminali C e D generano rispettivamente sequenze di c e a

Questa grammatica genera esattamente le stringhe nella forma $a^i b^j c^k$ con $i = j$ o $j = k$, dove $i, j, k \geq 1$.
d) $L_4 = \{a^n b^m \mid n \neq m,$ **dove** $n, m \geq 1\}$
La grammatica context-free per L_4 è:

$$\begin{aligned} S &\to A \mid B \\ A &\to aAb \mid aab \\ B &\to aBb \mid ab^2 \end{aligned}$$

La logica di questa grammatica è:

- Il non-terminale S sceglie tra le due possibili disuguaglianze ($n > m$ o $n < m$)

- Il non-terminale A genera stringhe con $n > m$, terminando con la produzione $A \to aab$ che garantisce almeno un a in più
- Il non-terminale B genera stringhe con $n < m$, terminando con la produzione $B \to ab^2$ che garantisce almeno un b in più

Questa grammatica genera esattamente le stringhe nella forma $a^n b^m$ con $n \neq m$ e $n, m \geq 1$.

e) $L_5 = \{wcw^R \mid w \in \{a, b\}^*\}$**, dove** w^R **è il reverse di** w

La grammatica context-free per L_5 è:

$$S \to aSa \mid bSb \mid c$$

La logica di questa grammatica è:

- La produzione $S \to c$ genera il simbolo centrale della stringa
- Le produzioni $S \to aSa$ e $S \to bSb$ aggiungono caratteri simmetricamente a sinistra e a destra, garantendo che la parte destra sia il reverse della parte sinistra

Questa grammatica genera esattamente le stringhe nella forma wcw^R dove $w \in \{a, b\}^*$.

Esercizio 2. Per ciascuna delle seguenti grammatiche, determinare il linguaggio generato e dimostrare formalmente che la grammatica genera quel linguaggio:

a) G_1:

$$\begin{aligned} S &\to aB \mid bA \\ A &\to a \mid aS \mid bAA \\ B &\to b \mid bS \mid aBB \end{aligned}$$

b) G_2:

$$\begin{aligned} S &\to AB \\ A &\to aA \mid \varepsilon \\ B &\to Bb \mid \varepsilon \end{aligned}$$

c) G_3:

$$\begin{aligned} S &\to aSb \mid T \\ T &\to aTb \mid \varepsilon \end{aligned}$$

Suggerimento: Utilizzare l'induzione sulla lunghezza della derivazione per dimostrare l'inclusione in entrambe le direzioni.

Soluzione. a) Grammatica G_1

Analizziamo sistematicamente la grammatica G_1:

$$S \to aB \mid bA$$
$$A \to a \mid aS \mid bAA$$
$$B \to b \mid bS \mid aBB$$

Un'analisi dettagliata mostra che questa grammatica genera tutte le stringhe non vuote su $\{a, b\}$. Formalmente: $L(G_1) = \{a, b\}^+$

Dimostrazione: Dimostriamo prima che $L(G_1) \subseteq \{a, b\}^+$, il che è ovvio poiché tutte le produzioni generano stringhe di a e b.

Per dimostrare che $\{a, b\}^+ \subseteq L(G_1)$, usiamo l'induzione sulla lunghezza delle stringhe.

Base: Per stringhe di lunghezza 1:

- $a \in L(G_1)$ perché $S \Rightarrow bA \Rightarrow ba$
- $b \in L(G_1)$ perché $S \Rightarrow aB \Rightarrow ab$

Passo induttivo: Supponiamo che tutte le stringhe di lunghezza $k \geq 1$ siano in $L(G_1)$. Consideriamo una stringa w di lunghezza $k + 1$.

Se $w = aw'$, dove $|w'| = k$, allora per ipotesi induttiva $w' \in L(G_1)$, quindi $S \Rightarrow^* w'$. Possiamo derivare w come: $S \Rightarrow bA \Rightarrow baS \Rightarrow ba(w') = aw'$

Analogamente, se $w = bw'$, possiamo derivare w utilizzando le produzioni appropriate.

Quindi, per induzione, ogni stringa in $\{a, b\}^+$ è generata da G_1.

b) Grammatica G_2

Il linguaggio generato dalla grammatica G_2:

$$S \to AB$$
$$A \to aA \mid \varepsilon$$
$$B \to Bb \mid \varepsilon$$

è $L(G_2) = \{a^n b^m \mid n, m \geq 0\}$.

Dimostrazione: Osserviamo che:

- A genera $\{a\}^*$, cioè $A \Rightarrow^* a^n$ per ogni $n \geq 0$
- B genera $\{b\}^*$, cioè $B \Rightarrow^* b^m$ per ogni $m \geq 0$

Dato che $S \Rightarrow AB$, la grammatica concatena queste due parti, generando $a^n b^m$ per ogni $n, m \geq 0$.

Per dimostrare formalmente che $L(G_2) = \{a^n b^m \mid n, m \geq 0\}$, procediamo in due direzioni:

1. Dimostriamo che $L(G_2) \subseteq \{a^n b^m \mid n, m \geq 0\}$: Qualsiasi derivazione in G_2 inizia con $S \Rightarrow AB$. Poi A genera una sequenza di a e B genera una sequenza di b. Quindi, ogni stringa in $L(G_2)$ ha la forma $a^n b^m$ per qualche $n, m \geq 0$.

2. Dimostriamo che $\{a^n b^m \mid n, m \geq 0\} \subseteq L(G_2)$: Per ogni $n, m \geq 0$, possiamo derivare $a^n b^m$ come segue:

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow a^n B \quad \text{(applicando } A \Rightarrow aA \text{ } n \text{ volte e poi } A \Rightarrow \varepsilon) \\ &\Rightarrow a^n b^m \quad \text{(applicando } B \Rightarrow Bb \text{ } m \text{ volte e poi } B \Rightarrow \varepsilon) \end{aligned}$$

Quindi, $L(G_2) = \{a^n b^m \mid n, m \geq 0\}$.

c) Grammatica G_3

Il linguaggio generato dalla grammatica G_3:

$$S \to aSb \mid T$$
$$T \to aTb \mid \varepsilon$$

è $L(G_3) = \{a^n b^n \mid n \geq 0\}$.

Dimostrazione: Analizziamo prima ciò che genera ciascun non-terminale:

- T genera $\{a^n b^n \mid n \geq 0\}$, che può essere dimostrato per induzione
- S genera $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^n \mid n \geq 0\} = \{a^n b^n \mid n \geq 0\}$

Per dimostrare formalmente che $L(G_3) = \{a^n b^n \mid n \geq 0\}$, procediamo in due direzioni:

1. Dimostriamo che $L(G_3) \subseteq \{a^n b^n \mid n \geq 0\}$: Per induzione sulla lunghezza della derivazione, possiamo dimostrare che:

- Se $T \Rightarrow^* w$, allora $w = a^n b^n$ per qualche $n \geq 0$
- Se $S \Rightarrow^* w$, allora $w = a^n b^n$ per qualche $n \geq 0$

2. Dimostriamo che $\{a^n b^n \mid n \geq 0\} \subseteq L(G_3)$: Per ogni $n \geq 0$, possiamo derivare $a^n b^n$ come segue:

$$\begin{aligned} S &\Rightarrow T \\ &\Rightarrow aTb \\ &\Rightarrow a^n T b^n \quad \text{(applicando } T \Rightarrow aTb \ n-1 \text{ volte)} \\ &\Rightarrow a^n \varepsilon b^n \quad \text{(applicando } T \Rightarrow \varepsilon) \\ &= a^n b^n \end{aligned}$$

Quindi, $L(G_3) = \{a^n b^n \mid n \geq 0\}$.

Esercizio 3. Si considerino le seguenti operazioni su linguaggi:

$$\text{MIRROR}(L) = \{w^R \mid w \in L\}$$
$$\text{CENTER}(L) = \{xay \mid xy \in L, a \in \Sigma\}$$

a) Dimostrare che se L è un linguaggio context-free, anche MIRROR(L) è context-free. Fornire un algoritmo che, data una grammatica G per L, costruisce una grammatica G' per MIRROR(L).

b) Determinare se CENTER(L) è sempre context-free quando L è context-free. Se sì, fornire un algoritmo per la costruzione della grammatica. Se no, fornire un controesempio.

c) Sia $L_{\text{pair}} = \{ww \mid w \in \{a, b\}^*\}$. Dimostrare che L_{pair} non è context-free utilizzando il pumping lemma per linguaggi context-free.

Soluzione. a) Dimostrazione che MIRROR(L) è context-free se L è context-free

Teorema: Se L è un linguaggio context-free, anche $\text{MIRROR}(L) = \{w^R \mid w \in L\}$ è context-free.

Dimostrazione costruttiva: Sia $G = (V, \Sigma, R, S)$ una grammatica context-free che genera L. Costruiamo una grammatica $G' = (V, \Sigma, R', S)$ per $\text{MIRROR}(L)$ come segue:

Per ogni regola di produzione $A \to \alpha$ in R, aggiungiamo la regola $A \to \alpha^R$ in R', dove α^R è il reverse di α.

Formalmente:

$$R' = \{A \to \alpha^R \mid A \to \alpha \in R\}$$

dove α^R è definito ricorsivamente:

- Se $\alpha = a$ (un terminale), allora $\alpha^R = a$
- Se $\alpha = A$ (un non-terminale), allora $\alpha^R = A$
- Se $\alpha = \beta\gamma$, allora $\alpha^R = \gamma^R\beta^R$
- Se $\alpha = \varepsilon$, allora $\alpha^R = \varepsilon$

Questa costruzione inverte l'ordine dei simboli nelle parti destre delle produzioni.

Dimostrazione di correttezza: Per induzione sulla lunghezza della derivazione, si può dimostrare che per ogni $A \in V$ e $w \in \Sigma^*$, $A \Rightarrow_G^* w$ se e solo se $A \Rightarrow_{G'}^* w^R$.

Quindi, $w \in L(G)$ se e solo se $w^R \in L(G')$, cioè $L(G') = \text{MIRROR}(L(G))$.

b) Determinare se CENTER(L) è sempre context-free quando L è context-free

Teorema: Se L è un linguaggio context-free, anche $\text{CENTER}(L) = \{xay \mid xy \in L, a \in \Sigma\}$ è context-free.

Dimostrazione costruttiva: Sia $G = (V, \Sigma, R, S)$ una grammatica context-free in Forma Normale di Chomsky che genera L. Costruiamo una grammatica $G' = (V', \Sigma, R', S')$ per $\text{CENTER}(L)$ come segue:

$V' = V \cup \{S'\} \cup \{X_a \mid X \in V, a \in \Sigma\}$

R' contiene le seguenti produzioni:

$$\begin{aligned}
&S' \to X_aY \quad \text{per ogni } a \in \Sigma \text{ e per ogni } X, Y \in V \text{ tali che } S \to XY \in R \\
&X_a \to a \quad \text{per ogni } X \in V, a \in \Sigma \\
&X_a \to Z_aW \quad \text{per ogni } X, Z, W \in V, a \in \Sigma \text{ tali che } X \to ZW \in R
\end{aligned}$$

La logica di questa costruzione è che X_a genera una stringa derivabile da X con un simbolo a inserito in qualche posizione.

Dimostrazione di correttezza: Si può dimostrare per induzione che questa grammatica genera esattamente $\text{CENTER}(L)$.

c) Dimostrazione che $L_{\text{pair}} = \{ww \mid w \in \{a, b\}^*\}$ non è context-free

Teorema: Il linguaggio $L_{\text{pair}} = \{ww \mid w \in \{a, b\}^*\}$ non è context-free.

Dimostrazione per contraddizione utilizzando il pumping lemma: Supponiamo per assurdo che L_{pair} sia context-free. Allora, per il pumping lemma, esiste una costante $p > 0$ tale che ogni stringa $z \in L_{\text{pair}}$ con $|z| \geq p$ può essere scomposta come $z = uvwxy$ con:

1. $|vwx| \leq p$
2. $|vx| > 0$
3. Per ogni $i \geq 0$, $uv^i wx^i y \in L_{\text{pair}}$

Consideriamo la stringa $z = a^p b^p a^p b^p \in L_{\text{pair}}$, dove la prima metà è $a^p b^p$ e la seconda metà è anche $a^p b^p$.

Poiché $|vwx| \leq p$, la sottostringa vwx è interamente contenuta nei primi $2p$ caratteri di z. Analizziamo i possibili casi:

Caso 1: vwx è interamente contenuto nel primo a^p. Allora v e x contengono solo a. Scegliendo $i = 2$, otteniamo una stringa in cui il numero di a nella prima metà è maggiore di p, mentre nella seconda metà rimane p. Quindi, la stringa ottenuta non è della forma ww.

Caso 2: vwx include caratteri sia dal primo a^p che dal primo b^p. Scegliendo $i = 0$, otteniamo una stringa in cui la proporzione tra a e b nella prima metà è alterata rispetto alla seconda metà. Quindi, la stringa non è della forma ww.

Caso 3: vwx è interamente contenuto nella prima metà $a^p b^p$. In qualsiasi configurazione, scegliendo $i = 2$ aumentiamo il numero di alcuni caratteri nella prima metà senza modificare la seconda metà, ottenendo una stringa che non è della forma ww.

Caso 4: vwx attraversa il confine tra la prima e la seconda metà. Questo caso è impossibile perché $|vwx| \leq p$ e la prima metà ha lunghezza $2p$.

In tutti i casi, troviamo un valore di i tale che $uv^i wx^i y \notin L_{\text{pair}}$, contraddicendo l'ipotesi che L_{pair} sia context-free.

Quindi, $L_{\text{pair}} = \{ww \mid w \in \{a, b\}^*\}$ non è context-free.

2 Ambiguità nelle Grammatiche Context-Free

Esercizio 4.

a) Dimostrare che la seguente grammatica è ambigua, trovando una stringa che ammette due diversi alberi di derivazione:

$$S \to S + S \mid S * S \mid (S) \mid a$$

b) Trasformare la grammatica precedente in una grammatica non ambigua che rispecchi le usuali regole di precedenza degli operatori aritmetici (dove $*$ ha precedenza su $+$).

c) Fornire una grammatica non ambigua per il linguaggio delle espressioni condizionali nella forma `if E then S else S`, dove:

- E rappresenta una condizione
- S rappresenta uno statement
- Si deve evitare l'ambiguità del "dangling else"

Soluzione. a) Dimostrazione che la grammatica è ambigua

La grammatica:

$$S \to S + S \mid S * S \mid (S) \mid a$$

è ambigua perché esistono stringhe che ammettono più di un albero di derivazione.

Esempio: Consideriamo la stringa $a + a * a$.

Questa stringa può essere derivata in due modi diversi:

Prima derivazione:

$$\begin{aligned} S &\Rightarrow S + S && [\text{usando } S \to S + S] \\ &\Rightarrow a + S && [\text{usando } S \to a] \\ &\Rightarrow a + S * S && [\text{usando } S \to S * S] \\ &\Rightarrow a + a * S && [\text{usando } S \to a] \\ &\Rightarrow a + a * a && [\text{usando } S \to a] \end{aligned}$$

Questa derivazione corrisponde all'interpretazione $(a+a)*a$, dove l'operazione $+$ viene eseguita prima.

Seconda derivazione:

$$\begin{aligned} S &\Rightarrow S * S && [\text{usando } S \to S * S] \\ &\Rightarrow S + S * S && [\text{usando } S \to S + S] \\ &\Rightarrow a + S * S && [\text{usando } S \to a] \\ &\Rightarrow a + a * S && [\text{usando } S \to a] \\ &\Rightarrow a + a * a && [\text{usando } S \to a] \end{aligned}$$

Questa derivazione corrisponde all'interpretazione $a+(a*a)$, dove l'operazione $*$ viene eseguita prima.

Poiché la stessa stringa ammette due alberi di derivazione che corrispondono a interpretazioni diverse, la grammatica è ambigua.

b) Trasformazione in una grammatica non ambigua con precedenza

Per trasformare la grammatica in una non ambigua che rispetti le precedenze (dove $*$ ha precedenza su $+$), introduciamo non-terminali per gestire i livelli di precedenza:

$$\begin{aligned} &S \to E \\ &E \to E + T \mid T \\ &T \to T * F \mid F \\ &F \to (E) \mid a \end{aligned}$$

In questa grammatica:

- E (espressione) può contenere addizioni
- T (termine) può contenere moltiplicazioni
- F (fattore) rappresenta atomi (variabili o espressioni parentesizzate)

La grammatica impone che la moltiplicazione abbia precedenza sull'addizione perché un'espressione E può contenere termini T, ma un termine T può contenere solo fattori F.

Per eliminare la ricorsione a sinistra (che può causare problemi in alcuni algoritmi di parsing):

$$
\begin{aligned}
S &\to E \\
E &\to TE' \\
E' &\to +TE' \mid \varepsilon \\
T &\to FT' \\
T' &\to *FT' \mid \varepsilon \\
F &\to (E) \mid a
\end{aligned}
$$

Questa grammatica genera lo stesso linguaggio ma è non ricorsiva a sinistra e mantiene le regole di precedenza.

c) Grammatica non ambigua per espressioni condizionali

Il "dangling else" è un problema di ambiguità che sorge quando non è chiaro a quale istruzione "if" sia associato un "else" in una sequenza di istruzioni if annidate.

Una grammatica non ambigua per le espressioni condizionali è:

$$
\begin{aligned}
S &\to \text{MatchedStmt} \mid \text{UnmatchedStmt} \\
\text{MatchedStmt} &\to \text{if } E \text{ then MatchedStmt else MatchedStmt} \mid \text{OtherStmt} \\
\text{UnmatchedStmt} &\to \text{if } E \text{ then } S \mid \text{if } E \text{ then MatchedStmt else UnmatchedStmt} \\
\text{OtherStmt} &\to \text{assignment} \mid \text{read} \mid \text{write} \mid \ldots
\end{aligned}
$$

La logica di questa grammatica è:

- MatchedStmt rappresenta istruzioni in cui ogni "if" ha un "else" corrispondente
- UnmatchedStmt rappresenta istruzioni in cui almeno un "if" non ha un "else"
- La regola UnmatchedStmt $\to$ if E then MatchedStmt else UnmatchedStmt garantisce che un "else" sia sempre associato all'istruzione "if" più vicina senza "else"

Con questa grammatica, la stringa "if E1 then if E2 then S1 else S2" ha un'interpretazione unica: l'"else" è associato al secondo "if", coerente con la semantica tipica nei linguaggi di programmazione.

Esercizio 5. Un linguaggio si dice inerentemente ambiguo se non esiste alcuna grammatica non ambigua che lo genera.

a) Dimostrare che il linguaggio $L = \{a^i b^j c^k \mid i = j \text{ o } j = k, \text{ dove } i, j, k \geq 1\}$ è inerentemente ambiguo.

b) Sia $L = \{a^n b^m c^p \mid n = m \text{ o } m = p, \text{ dove } n, m, p \geq 0\}$. Discutere se L è inerentemente ambiguo.

Suggerimento: Utilizzare il teorema dell'intersezione per linguaggi context-free.

Soluzione. a) Dimostrazione che $L = \{a^i b^j c^k \mid i = j$ o $j = k,$ dove $i, j, k \geq 1\}$ è inerentemente ambiguo

Teorema: Il linguaggio $L = \{a^i b^j c^k \mid i = j$ o $j = k,$ dove $i, j, k \geq 1\}$ è inerentemente ambiguo.

Dimostrazione: Definiamo due linguaggi context-free:

$$L_1 = \{a^i b^j c^k \mid i = j, i, j, k \geq 1\}$$
$$L_2 = \{a^i b^j c^k \mid j = k, i, j, k \geq 1\}$$

È facile verificare che entrambi sono context-free:

- L_1 è generato dalla grammatica: $S \to aS_1c \mid S_1c,\ S_1 \to aS_1b \mid ab$
- L_2 è generato dalla grammatica: $S \to aS \mid S_1,\ S_1 \to bS_1c \mid bc$

Osserviamo che $L = L_1 \cup L_2$.

Se L non fosse inerentemente ambiguo, esisterebbe una grammatica non ambigua G con $L(G) = L$. Consideriamo la stringa $w = a^n b^n c^n$ per $n \geq 1$. Questa stringa appartiene sia a L_1 (poiché $i = j = n$) che a L_2 (poiché $j = k = n$).

Poiché G è non ambigua, esiste un unico albero di derivazione per w. Questo albero dovrebbe in qualche modo "decidere" se w appartiene a L perché $i = j$ o perché $j = k$. Ma questa è una contraddizione, perché w è in L per entrambe le ragioni.

Più formalmente, definiamo:

$$L_{eq} = \{a^i b^j c^k \mid i = j = k, i, j, k \geq 1\}$$
$$L_{diff} = \{a^i b^j c^k \mid i = j \neq k \text{ o } i \neq j = k, i, j, k \geq 1\}$$

Abbiamo $L = L_{eq} \cup L_{diff}$ e $L_{eq} \cap L_{diff} = \emptyset$.

Per ogni stringa $w \in L_{eq}$, una grammatica non ambigua dovrebbe avere un unico albero di derivazione che "decide" se $w \in L$ a causa di $i = j$ o a causa di $j = k$. Ma questa è una contraddizione perché $w \in L$ per entrambe le condizioni.

Pertanto, L è inerentemente ambiguo.

b) Discussione su $L = \{a^n b^m c^p \mid n = m$ o $m = p,$ dove $n, m, p \geq 0\}$

Questo linguaggio è una variante di quello analizzato in (a), con la differenza che n, m, p possono essere zero. La dimostrazione dell'ambiguità inerente segue lo stesso ragionamento.

Definiamo:

$$L_1 = \{a^n b^m c^p \mid n = m, n, m, p \geq 0\}$$
$$L_2 = \{a^n b^m c^p \mid m = p, n, m, p \geq 0\}$$

Entrambi sono linguaggi context-free e $L = L_1 \cup L_2$.

Considerando la stringa $w = a^n b^n c^n$ per $n \geq 0$, abbiamo $w \in L_1 \cap L_2$.

Utilizzando lo stesso ragionamento del punto (a), possiamo dimostrare che L è inerentemente ambiguo.

La presenza di stringhe come ε (dove $n = m = p = 0$) non modifica l'argomentazione, poiché l'ambiguità deriva dalle stringhe del tipo $a^n b^n c^n$ che sono in $L_1 \cap L_2$.

3 Forma Normale di Chomsky

Esercizio 6. Convertire le seguenti grammatiche in Forma Normale di Chomsky, mostrando tutti i passaggi della trasformazione:

a) G_1:

$$S \to AB \mid a$$
$$A \to BA \mid \varepsilon$$
$$B \to b$$

b) G_2:

$$S \to aSb \mid ab$$

c) G_3:

$$S \to AB \mid C$$
$$A \to aAb \mid ab$$
$$B \to cB \mid c$$
$$C \to aCa \mid a$$

Ricordare i passaggi: (1) Aggiungere un nuovo simbolo iniziale, (2) Eliminare le produzioni ε, (3) Eliminare le produzioni unitarie, (4) Convertire le restanti produzioni nella forma corretta.

Soluzione. a) Conversione di G_1 in Forma Normale di Chomsky

Grammatica originale G_1:

$$S \to AB \mid a$$
$$A \to BA \mid \varepsilon$$
$$B \to b$$

Passo 1: Aggiungere un nuovo simbolo iniziale

$$S_0 \to S$$
$$S \to AB \mid a$$
$$A \to BA \mid \varepsilon$$
$$B \to b$$

Passo 2: Eliminare le produzioni ε Abbiamo la produzione $A \to \varepsilon$. Identifichiamo tutte le produzioni che contengono A e creiamo nuove produzioni sostituendo A con ε:

Da $S \to AB$, otteniamo $S \to B$ Da $A \to BA$, otteniamo $A \to B$

Risultato dopo l'eliminazione di ε-produzioni:

$$S_0 \to S$$
$$S \to AB \mid B \mid a$$
$$A \to BA \mid B$$
$$B \to b$$

Passo 3: Eliminare le produzioni unitarie Le produzioni unitarie sono $S_0 \to S$, $S \to B$ e $A \to B$.

Sostituendo $S_0 \to S$ con le produzioni di S: $S_0 \to AB \mid B \mid a$
Sostituendo $S \to B$ con le produzioni di B: $S \to AB \mid b \mid a$
Sostituendo $A \to B$ con le produzioni di B: $A \to BA \mid b$
Risultato dopo l'eliminazione delle produzioni unitarie:

$$\begin{aligned} S_0 &\to AB \mid b \mid a \\ S &\to AB \mid b \mid a \\ A &\to BA \mid b \\ B &\to b \end{aligned}$$

Passo 4: Convertire le produzioni nella forma corretta Introduciamo non-terminali per i terminali:

$$\begin{aligned} T_a &\to a \\ T_b &\to b \end{aligned}$$

Sostituendo nelle produzioni:

$$\begin{aligned} S_0 &\to AB \mid T_b \mid T_a \\ S &\to AB \mid T_b \mid T_a \\ A &\to BT_a \mid T_b \\ B &\to T_b \\ T_a &\to a \\ T_b &\to b \end{aligned}$$

Grammatica finale in FNC:

$$\begin{aligned} S_0 &\to AB \mid T_b \mid T_a \\ S &\to AB \mid T_b \mid T_a \\ A &\to BT_a \mid T_b \\ B &\to T_b \\ T_a &\to a \\ T_b &\to b \end{aligned}$$

b) Conversione di G_2 in Forma Normale di Chomsky
Grammatica originale G_2:

$$S \to aSb \mid ab$$

Passo 1: Aggiungere un nuovo simbolo iniziale

$$\begin{aligned} S_0 &\to S \\ S &\to aSb \mid ab \end{aligned}$$

Passo 2: Eliminare le produzioni ε Non ci sono produzioni ε, quindi questo passo non è necessario.

Passo 3: Eliminare le produzioni unitarie L'unica produzione unitaria è $S_0 \to S$.

Sostituendo $S_0 \to S$ con le produzioni di S:

$$\begin{aligned} S_0 &\to aSb \mid ab \\ S &\to aSb \mid ab \end{aligned}$$

Passo 4: Convertire le produzioni nella forma corretta Introduciamo non-terminali per i terminali e per le catene di simboli:

$$\begin{aligned} T_a &\to a \\ T_b &\to b \\ X &\to T_a S \\ Y &\to T_a T_b \end{aligned}$$

Sostituendo nelle produzioni:

$$\begin{aligned} S_0 &\to XT_b \mid Y \\ S &\to XT_b \mid Y \\ X &\to T_a S \\ Y &\to T_a T_b \\ T_a &\to a \\ T_b &\to b \end{aligned}$$

Grammatica finale in FNC:

$$\begin{aligned} S_0 &\to XT_b \mid Y \\ S &\to XT_b \mid Y \\ X &\to T_a S \\ Y &\to T_a T_b \\ T_a &\to a \\ T_b &\to b \end{aligned}$$

c) Conversione di G_3 in Forma Normale di Chomsky

Grammatica originale G_3:

$$\begin{aligned} S &\to AB \mid C \\ A &\to aAb \mid ab \\ B &\to cB \mid c \\ C &\to aCa \mid a \end{aligned}$$

Passo 1: Aggiungere un nuovo simbolo iniziale

$$\begin{aligned} S_0 &\to S \\ S &\to AB \mid C \\ A &\to aAb \mid ab \\ B &\to cB \mid c \\ C &\to aCa \mid a \end{aligned}$$

Passo 2: Eliminare le produzioni ε Non ci sono produzioni ε, quindi questo passo non è necessario.

Passo 3: Eliminare le produzioni unitarie Le produzioni unitarie sono $S_0 \to S$ e $S \to C$.

Sostituendo $S_0 \to S$ con le produzioni di S: $S_0 \to AB \mid C$

Sostituendo $S_0 \to C$ e $S \to C$ con le produzioni di C:

$$\begin{aligned} S_0 &\to AB \mid aCa \mid a \\ S &\to AB \mid aCa \mid a \end{aligned}$$

Risultato dopo l'eliminazione delle produzioni unitarie:

$$\begin{aligned} S_0 &\to AB \mid aCa \mid a \\ S &\to AB \mid aCa \mid a \\ A &\to aAb \mid ab \\ B &\to cB \mid c \\ C &\to aCa \mid a \end{aligned}$$

Passo 4: Convertire le produzioni nella forma corretta Introduciamo non-terminali per i terminali e per le catene:

$$\begin{aligned} T_a &\to a \\ T_b &\to b \\ T_c &\to c \\ X_1 &\to T_a A \\ X_2 &\to X_1 T_b \\ X_3 &\to T_a T_b \\ X_4 &\to T_a C \\ X_5 &\to X_4 T_a \\ X_6 &\to T_c B \end{aligned}$$

Sostituendo nelle produzioni:

$$
\begin{aligned}
S_0 &\to AB \mid X_5 \mid T_a \\
S &\to AB \mid X_5 \mid T_a \\
A &\to X_2 \mid X_3 \\
B &\to X_6 \mid T_c \\
C &\to X_5 \mid T_a \\
X_1 &\to T_a A \\
X_2 &\to X_1 T_b \\
X_3 &\to T_a T_b \\
X_4 &\to T_a C \\
X_5 &\to X_4 T_a \\
X_6 &\to T_c B \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

Grammatica finale in FNC:

$$
\begin{aligned}
S_0 &\to AB \mid X_5 \mid T_a \\
S &\to AB \mid X_5 \mid T_a \\
A &\to X_2 \mid X_3 \\
B &\to X_6 \mid T_c \\
C &\to X_5 \mid T_a \\
X_1 &\to T_a A \\
X_2 &\to X_1 T_b \\
X_3 &\to T_a T_b \\
X_4 &\to T_a C \\
X_5 &\to X_4 T_a \\
X_6 &\to T_c B \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

Esercizio 7. Sia data la seguente grammatica context-free:

$$
\begin{aligned}
S &\to ABA \mid BSB \\
A &\to a \mid aA \mid \varepsilon \\
B &\to b \mid bB \mid \varepsilon
\end{aligned}
$$

a) Determinare il linguaggio generato da G.

b) Eliminare le produzioni ε dalla grammatica.

c) Eliminare le produzioni unitarie.

d) Completare la trasformazione in FNC.

e) Costruire un esempio di derivazione per la stringa "ababab" nella grammatica originale e nella grammatica in FNC.

Soluzione. a) Determinare il linguaggio generato da G

Analizziamo sistematicamente i linguaggi generati dai non-terminali A e B:

- A genera $\{a\}^*$ (zero o più occorrenze di a)
- B genera $\{b\}^*$ (zero o più occorrenze di b)

Considerando le produzioni del simbolo iniziale:

- $S \to ABA$ genera stringhe della forma $a^i b^j a^k$ dove $i, j, k \geq 0$
- $S \to BSB$ genera stringhe della forma $b^i S b^j$ dove $i, j \geq 0$

La produzione $S \to BSB$ può essere applicata ricorsivamente, inserendo S tra sequenze di b. Successivamente, S può essere sostituito con ABA, generando stringhe con blocchi alternati di a e b.

Il linguaggio generato da G è l'insieme di stringhe che possono essere espresse come una concatenazione alternata di blocchi di a e b, dove i blocchi possono essere vuoti eccetto il primo e l'ultimo se sono dello stesso tipo.

Formalmente:

$$L(G) = \{w \in \{a, b\}^* \mid w \text{ è una sequenza alternata di blocchi di } a \text{ e } b\}$$

Questo include stringhe come a^*, b^*, $a^*b^*a^*$, $b^*a^*b^*$, $a^*b^*a^*b^*a^*$, ecc.

b) Eliminare le produzioni ε dalla grammatica

Abbiamo due produzioni ε: $A \to \varepsilon$ e $B \to \varepsilon$.

Passo 1: Identificare i non-terminali che possono derivare ε (chiamati "nullable"). A e B sono nullable.

Passo 2: Per ogni produzione contenente non-terminali nullable, aggiungere nuove produzioni sostituendo ogni possibile combinazione di questi non-terminali con ε.

Da $S \to ABA$, otteniamo:

- $S \to ABA$ (originale)
- $S \to AB$ (sostituendo A finale con ε)
- $S \to BA$ (sostituendo A iniziale con ε)
- $S \to A$ (sostituendo B con ε)
- $S \to B$ (sostituendo entrambi gli A con ε)
- $S \to \varepsilon$ (sostituendo tutti i non-terminali con ε)

Da $S \to BSB$, otteniamo:

- $S \to BSB$ (originale)
- $S \to BS$ (sostituendo B finale con ε)
- $S \to SB$ (sostituendo B iniziale con ε)
- $S \to S$ (sostituendo entrambi i B con ε)

Da $A \to aA$, otteniamo:

- $A \to aA$ (originale)
- $A \to a$ (sostituendo A con ε)

Da $B \to bB$, otteniamo:

- $B \to bB$ (originale)
- $B \to b$ (sostituendo B con ε)

Passo 3: Rimuovere le produzioni ε, tranne se $S \to \varepsilon$ è l'unica produzione e vogliamo generare ε.
La grammatica senza produzioni ε è:

$$\begin{aligned} &S \to ABA \mid AB \mid BA \mid A \mid B \mid BSB \mid BS \mid SB \mid S \\ &A \to a \mid aA \\ &B \to b \mid bB \end{aligned}$$

c) Eliminare le produzioni unitarie
Le produzioni unitarie sono: $S \to A \mid B \mid S$.
Per eliminare $S \to A$, sostituiamo con le produzioni di A: $S \to a \mid aA$
Per eliminare $S \to B$, sostituiamo con le produzioni di B: $S \to b \mid bB$
Per eliminare $S \to S$ (ricorsione diretta), semplicemente la rimuoviamo.
La grammatica senza produzioni unitarie è:

$$\begin{aligned} &S \to ABA \mid AB \mid BA \mid a \mid aA \mid b \mid bB \mid BSB \mid BS \mid SB \\ &A \to a \mid aA \\ &B \to b \mid bB \end{aligned}$$

d) Completare la trasformazione in FNC
Passo 1: Introdurre non-terminali per sostituire i terminali:

$$\begin{aligned} &T_a \to a \\ &T_b \to b \end{aligned}$$

Passo 2: Introdurre non-terminali per le produzioni con più di due non-terminali:

$$
\begin{aligned}
X_1 &\to AB \\
X_2 &\to X_1 A \\
X_3 &\to BA \\
X_4 &\to BS \\
X_5 &\to X_4 B
\end{aligned}
$$

Passo 3: Sostituire nelle produzioni originali:

$$
\begin{aligned}
S &\to X_2 \mid X_1 \mid X_3 \mid T_a \mid Y_1 \mid T_b \mid Y_2 \mid X_5 \mid X_4 \mid SB \\
A &\to T_a \mid Y_1 \\
B &\to T_b \mid Y_2 \\
X_1 &\to AB \\
X_2 &\to X_1 A \\
X_3 &\to BA \\
X_4 &\to BS \\
X_5 &\to X_4 B \\
Y_1 &\to T_a A \\
Y_2 &\to T_b B \\
T_a &\to a \\
T_b &\to b
\end{aligned}
$$

La produzione $S \to SB$ non è ancora in FNC. Introduciamo un nuovo non-terminale:

$$X_6 \to SB$$

Sostituendo:

$$
\begin{aligned}
S &\to X_2 \mid X_1 \mid X_3 \mid T_a \mid Y_1 \mid T_b \mid Y_2 \mid X_5 \mid X_4 \mid X_6 \\
X_6 &\to SB
\end{aligned}
$$

Grammatica finale in FNC:

$$
\begin{aligned}
S &\to X_2 \mid X_1 \mid X_3 \mid T_a \mid Y_1 \mid T_b \mid Y_2 \mid X_5 \mid X_4 \mid X_6 \\
A &\to T_a \mid Y_1 \\
B &\to T_b \mid Y_2 \\
X_1 &\to AB \\
X_2 &\to X_1 A \\
X_3 &\to BA \\
X_4 &\to BS \\
X_5 &\to X_4 B \\
X_6 &\to SB \\
Y_1 &\to T_a A \\
Y_2 &\to T_b B \\
T_a &\to a \\
T_b &\to b
\end{aligned}
$$

e) Costruire un esempio di derivazione per la stringa "ababab" nella grammatica originale e nella grammatica in FNC

Derivazione nella grammatica originale:

Per la stringa "ababab", possiamo costruire la seguente derivazione:

$$
\begin{aligned}
S &\Rightarrow ABA \\
&\Rightarrow aA \cdot B \cdot A \\
&\Rightarrow a \cdot B \cdot A \\
&\Rightarrow a \cdot bB \cdot A \\
&\Rightarrow a \cdot b \cdot A \\
&\Rightarrow a \cdot b \cdot aA \\
&\Rightarrow a \cdot b \cdot a \cdot b \cdot a \cdot b
\end{aligned}
$$

Derivazione nella grammatica in FNC:

Per la stessa stringa "ababab", la derivazione nella grammatica in FNC è più complessa:

$$
\begin{aligned}
S &\Rightarrow X_2 \\
&\Rightarrow X_1 A \\
&\Rightarrow ABA \\
&\Rightarrow T_a BA \\
&\Rightarrow aBA \\
&\Rightarrow aT_b A \\
&\Rightarrow abA \\
&\Rightarrow abT_a \\
&\Rightarrow aba \\
&\Rightarrow abab \\
&\Rightarrow ababa \\
&\Rightarrow ababab
\end{aligned}
$$

(Nota: questa è una versione semplificata; la derivazione completa includerebbe ogni passaggio di sostituzione attraverso i non-terminali aggiuntivi introdotti nella FNC.)

Esercizio 8. Sia L un linguaggio context-free generato da una grammatica G in Forma Normale di Chomsky.

a) Dimostrare che ogni derivazione di una stringa w di lunghezza n in G richiede esattamente $2n - 1$ passi di derivazione.

b) Dato che per ogni stringa $w \in L$ con $|w| = n$ si richiede $O(n)$ passi di derivazione, discutere le implicazioni di questo fatto sull'efficienza degli algoritmi di parsing per linguaggi context-free.

c) Utilizzare la FNC per dimostrare che per ogni linguaggio context-free L, l'insieme $\{w \in L \mid |w| \leq n\}$ è finito per ogni $n \geq 0$.

Soluzione. a) Dimostrazione che ogni derivazione di una stringa w di lunghezza n in una grammatica in FNC richiede esattamente $2n - 1$ passi

Teorema: Sia G una grammatica in Forma Normale di Chomsky e sia w una stringa di lunghezza $n \geq 1$. Ogni derivazione di w in G richiede esattamente $2n - 1$ passi.

Dimostrazione per induzione sulla lunghezza n della stringa:

Base: $n = 1$ (stringa di lunghezza 1) Se $|w| = 1$, allora $w = a$ per qualche terminale a. In FNC, l'unica produzione che può generare un terminale è della forma $A \to a$. Quindi, la derivazione è $S \Rightarrow a$, che richiede 1 passo. Verifichiamo: $2n - 1 = 2(1) - 1 = 1$.

Ipotesi induttiva: Supponiamo che l'affermazione sia vera per tutte le stringhe di lunghezza minore di $n > 1$.

Passo induttivo: Sia w una stringa di lunghezza n. Poiché G è in FNC, la prima produzione applicata deve essere della forma $S \Rightarrow AB$ per qualche non-terminali A e B.

Questa produzione divide il problema in due sottoproblemi: 1. Derivare una stringa u da A 2. Derivare una stringa v da B

tali che $w = uv$.

Siano $|u| = k$ e $|v| = n - k$ per qualche $1 \leq k \leq n - 1$.

Per l'ipotesi induttiva:

- La derivazione di u da A richiede $2k-1$ passi
- La derivazione di v da B richiede $2(n-k)-1$ passi

Quindi, il numero totale di passi per derivare w è:

$$\begin{aligned} 1 + (2k-1) + (2(n-k)-1) &= 1 + 2k - 1 + 2n - 2k - 1 \\ &= 2n - 1 \end{aligned}$$

Il primo 1 rappresenta la produzione iniziale $S \Rightarrow AB$.

Pertanto, ogni derivazione di una stringa w di lunghezza n in una grammatica in FNC richiede esattamente $2n-1$ passi.

b) Implicazioni sull'efficienza degli algoritmi di parsing

Il fatto che ogni derivazione in FNC richieda esattamente $2n-1 = O(n)$ passi ha diverse implicazioni per l'efficienza degli algoritmi di parsing:

1. *Limite inferiore di complessità*: Poiché sono necessari almeno $2n-1$ passi per generare una stringa di lunghezza n, qualsiasi algoritmo di parsing completo deve avere una complessità di almeno $\Omega(n)$.

2. *Complessità degli algoritmi standard*: Gli algoritmi di parsing più comuni per linguaggi context-free (CYK, Earley) hanno complessità $O(n^3)$ nel caso peggiore, che è significativamente superiore al limite inferiore $\Omega(n)$.

3. *Struttura dell'albero di parsing*: Poiché ogni derivazione richiede $2n-1$ passi, l'albero di parsing per una stringa di lunghezza n ha esattamente n foglie (i terminali) e $n-1$ nodi interni (le applicazioni di produzioni della forma $A \to BC$). Questo comporta un albero binario con n foglie e un totale di $2n-1$ nodi.

4. *Ambiguità*: Per grammatiche ambigue, possono esistere molteplici derivazioni (e quindi alberi di parsing) per la stessa stringa. Anche se ogni singola derivazione richiede solo $O(n)$ passi, il numero di derivazioni possibili può crescere esponenzialmente, rendendo il problema di trovare tutte le derivazioni molto più complesso.

5. *Tabellatura e memoizzazione*: Gli algoritmi di parsing come CYK costruiscono una tabella di dimensione $O(n^2)$ per memorizzare i non-terminali che possono derivare ogni sottostringa. Per ogni cella (i, j) della tabella, l'algoritmo considera $O(n)$ possibili divisioni, portando alla complessità totale di $O(n^3)$. Questo approccio evita di ripetere calcoli già effettuati.

In conclusione, sebbene ogni singola derivazione richieda solo $O(n)$ passi, gli algoritmi di parsing efficienti devono considerare tutte le possibili strutture di derivazione, portando a complessità superiori. Questo è un esempio di come il limite inferiore teorico di un problema possa essere inferiore alla complessità degli algoritmi pratici noti.

c) Dimostrazione che $\{w \in L \mid |w| \leq n\}$ è finito per ogni linguaggio context-free L

Teorema: Per ogni linguaggio context-free L e per ogni $n \geq 0$, l'insieme $\{w \in L \mid |w| \leq n\}$ è finito.

Dimostrazione: Sia $G = (V, \Sigma, R, S)$ una grammatica in FNC che genera L.

Ogni stringa in Σ^* di lunghezza al più n appartiene all'insieme finito $\Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^n$. Il numero totale di queste stringhe è:

$$\sum_{i=0}^{n} |\Sigma|^i = \frac{|\Sigma|^{n+1} - 1}{|\Sigma| - 1}$$

se $|\Sigma| > 1$, o $n + 1$ se $|\Sigma| = 1$.

Poiché $\{w \in L \mid |w| \leq n\} \subseteq \{w \in \Sigma^* \mid |w| \leq n\}$, e quest'ultimo insieme è finito, anche $\{w \in L \mid |w| \leq n\}$ è finito.

Dimostrazione alternativa utilizzando FNC: Dal punto (a), sappiamo che una derivazione di una stringa di lunghezza m in FNC richiede esattamente $2m - 1$ passi. Quindi, per generare stringhe di lunghezza al più n, sono necessari al più $2n - 1$ passi.

Il numero totale di possibili derivazioni di lunghezza al più $2n - 1$ è limitato, poiché in ogni passo possiamo applicare una delle produzioni in R (che è un insieme finito). Pertanto, il numero di stringhe distinte che possono essere generate con al più $2n - 1$ passi è finito.

Poiché ogni stringa in $\{w \in L \mid |w| \leq n\}$ richiede al più $2n-1$ passi per essere derivata, e il numero di tali derivazioni è finito, anche $\{w \in L \mid |w| \leq n\}$ è finito.