Tutorato di Automi e Linguaggi Formali

PDA, CFG ed Esercizi di approfondimento

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

Anno Accademico 2024-2025

1 Esercizi su Grammatiche Context-Free

CFG da PDA 1. Si consideri il seguente PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ dove:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{a, b, c\}$
- $\Gamma = \{Z_0, A, B\}$
- $F = \{q_3\}$

E con funzione di transizione δ definita come:

- $\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$
- $\delta(q_0, a, A) = \{(q_0, AA)\}$
- $\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$
- $\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$
- $\delta(q_1, c, Z_0) = \{(q_2, Z_0)\}$
- $\delta(q_2, c, Z_0) = \{(q_3, Z_0)\}$

Convertire questo PDA in una grammatica context-free equivalente utilizzando l'algoritmo standard di conversione.

Soluzione. Per convertire un PDA in una grammatica context-free equivalente, utilizziamo l'algoritmo standard che crea una variabile A_{pXq} per ogni tripla di stati p, q e simbolo di pila X. Questa variabile genera tutte le stringhe che permettono al PDA di passare

dallo stato p con X in cima alla pila allo stato q con la pila nello stesso stato (eccetto per X).

Iniziamo definendo la grammatica $G = (V, \Sigma, R, S)$ dove:

- V include le variabili A_{pXq} per ogni $p, q \in Q$ e $X \in \Gamma$
- $\Sigma = \{a, b, c\}$ (lo stesso alfabeto del PDA)
- $S = A_{q_0 Z_0 q_3}$ (la variabile di partenza che rappresenta l'intera computazione)
- R è l'insieme delle regole di produzione che costruiremo

Ora, per ogni transizione del PDA, aggiungiamo regole alla grammatica:

1. Per la transizione $\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$: - Aggiungiamo $A_{q_0 Z_0 q} \to a A_{q_0 A r} A_{r Z_0 q}$ per ogni $q, r \in Q$ - Questo rappresenta: leggiamo 'a', rimpiazziamo Z_0 con AZ_0, e poi dobbiamo elaborare prima A (raggiungendo un qualche stato r) e poi Z_0 per arrivare infine allo stato q

2. Per la transizione $\delta(q_0, a, A) = \{(q_0, AA)\}$: - Aggiungiamo $A_{q_0 A q} \to a A_{q_0 A r} A_{r A q}$ per ogni $q, r \in Q$

3. Per la transizione $\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$: - Aggiungiamo $A_{q_0 A q_1} \to b$ - Questo rappresenta: leggiamo 'b', rimuoviamo A dalla pila e passiamo direttamente allo stato q_1

4. Per la transizione $\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$: - Aggiungiamo $A_{q_1 A q_1} \to b$

5. Per la transizione $\delta(q_1, c, Z_0) = \{(q_2, Z_0)\}$: - Aggiungiamo $A_{q_1 Z_0 q_2} \to c$

6. Per la transizione $\delta(q_2, c, Z_0) = \{(q_3, Z_0)\}$: - Aggiungiamo $A_{q_2 Z_0 q_3} \to c$

Inoltre, per ogni stato $q \in Q$ e ogni simbolo di pila $X \in \Gamma$, aggiungiamo la regola: $A_{qXq} \to \varepsilon$ (che rappresenta la possibilità di non fare alcuna mossa)

La grammatica così costruita è molto grande con molte produzioni. Possiamo semplificarla osservando il comportamento specifico del PDA.

Analizzando il linguaggio accettato dal PDA, possiamo vedere che accetta stringhe della forma $a^n b^n c^2$ per $n \geq 1$. Il PDA funziona così: - Legge un certo numero di 'a' e impila un simbolo A per ciascuno - Poi legge lo stesso numero di 'b' rimuovendo un simbolo A per ciascuno - Infine legge esattamente due 'c' e termina nello stato finale q_3

Una grammatica context-free equivalente semplificata potrebbe essere:

$$
\begin{aligned}
S &\to AC \\
A &\to aAb \mid ab \\
C &\to cc
\end{aligned}
$$

Questa grammatica genera il linguaggio $\{a^n b^n c^2 \mid n \geq 1\}$, che è esattamente il linguaggio accettato dal PDA dato.

Per verificare che questa semplificazione sia corretta, possiamo analizzare i passi di computazione del PDA: 1. Partendo dallo stato q_0 con Z_0 sulla pila, leggiamo 'a' e impiliamo A sopra Z_0 2. Per ogni 'a' aggiuntivo, impiliamo un altro A 3. Per ogni 'b', rimuoviamo un A e passiamo allo stato q_1 dopo il primo 'b' 4. Dopo aver letto tutti i 'b' (tanti quanti gli 'a'), rimaniamo con solo Z_0 sulla pila 5. Leggiamo il primo 'c' e passiamo allo stato q_2 6. Leggiamo il secondo 'c' e passiamo allo stato finale q_3

Questa sequenza è esattamente catturata dalla semplice grammatica che abbiamo fornito.

Proprietà di chiusura 2. Data una grammatica context-free G che genera il linguaggio $L(G)$, considera il seguente linguaggio:

$$L_{half} = \{w \mid \exists v \text{ tale che } |v| = |w| \text{ e } wv \in L(G)\}$$

Ovvero, L_{half} è l'insieme delle metà sinistre delle stringhe in $L(G)$ di lunghezza pari.

Dimostra che se $L(G)$ è context-free, allora anche L_{half} è context-free.

Suggerimento: Modifica la grammatica G in modo che generi solo le metà sinistre delle stringhe in $L(G)$.

Soluzione. Per dimostrare che L_{half} è context-free quando $L(G)$ è context-free, costruiremo un PDA che accetta L_{half} partendo da un PDA che accetta $L(G)$.

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA che accetta $L(G)$ per stato finale.

Costruiamo un nuovo PDA $P' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$ che accetta L_{half} come segue:

- $Q' = Q \cup \{q_0', q_1', q_f'\}$ (aggiungiamo tre nuovi stati)
- $\Gamma' = \Gamma \cup \{\$, \#\}$ (aggiungiamo due nuovi simboli di pila)
- q_0' è il nuovo stato iniziale
- $Z_0' = Z_0$ (stesso simbolo iniziale di P)
- $F' = \{q_f'\}$ (un nuovo stato finale)

La funzione di transizione δ' include le seguenti transizioni:

- $\delta'(q_0', \varepsilon, Z_0) = \{(q_1', \$Z_0)\}$ (inizializziamo la pila con un contatore)
- Per ogni $a \in \Sigma$: $\delta'(q_1', a, \gamma) = \{(q_1', a\gamma)\}$ (leggiamo e impiliamo la prima parte)
- $\delta'(q_1', \varepsilon, \gamma) = \{(q_0, \#\gamma)\}$ (segniamo la metà e passiamo allo stato iniziale del PDA originale)
- Includiamo tutte le transizioni originali di δ
- Per ogni $q \in F$, aggiungiamo $\delta'(q, \varepsilon, \#) = \{(q_f', \varepsilon)\}$ (se il PDA originale accetta e siamo a metà, accettiamo)

Inoltre, dobbiamo aggiungere transizioni che verifichino che la lunghezza della seconda parte sia uguale alla prima:

- Per ogni $a \in \Sigma$ e $q \in Q$: $\delta'(q, a, a) = \{(q, \varepsilon)\}$ (verifichiamo che la seconda parte sia della stessa lunghezza della prima, consumando caratteri dalla pila)

L'idea è che P' legge la prima parte dell'input (che potrebbe essere in L_{half}), la memorizza sulla pila, e poi simula P sulla seconda parte. Accetta se P accetta e se la prima parte e la seconda parte hanno la stessa lunghezza, cioè se la stringa completa è in $L(G)$ e la prima metà è proprio w.

Alternativamente, possiamo costruire una grammatica context-free per L_{half} partendo da una grammatica per $L(G)$.

Sia $G = (V, \Sigma, P, S)$ una grammatica context-free che genera $L(G)$. Costruiamo una nuova grammatica $G' = (V', \Sigma, P', S')$ che genera L_{half}.

Per ogni non-terminale $A \in V$, introduciamo due nuovi non-terminali A_L e A_R che rappresentano, rispettivamente, la metà sinistra e la metà destra delle stringhe derivabili da A. Inoltre, introduciamo un nuovo non-terminale A_M che rappresenta il punto di divisione all'interno delle stringhe derivabili da A.

$V' = V \cup \{A_L, A_R, A_M \mid A \in V\} \cup \{S'\}$

Il simbolo iniziale è S' e le regole di produzione P' includono:

- $S' \to S_L$ (iniziamo generando la metà sinistra)
- Per ogni regola $A \to BC$ in P, aggiungiamo:
  - $A_L \to B_L$ (la divisione è nella seconda parte)
  - $A_L \to B_M C_L$ (la divisione è alla fine della prima parte)
  - $A_M \to B_R C_L$ (la divisione è tra B e C)
  - $A_R \to C_R$ (la divisione è nella prima parte)
- Per ogni regola $A \to a$ in P, aggiungiamo:
  - $A_L \to a$ (il terminale è nella metà sinistra)
  - $A_M \to \varepsilon$ (il punto di divisione è esattamente qui)
  - $A_R \to a$ (il terminale è nella metà destra)

Questa grammatica genera tutte le possibili metà sinistre delle stringhe in $L(G)$ che hanno una lunghezza pari. Il ragionamento è che A_L genera la parte della stringa che si trova interamente nella metà sinistra, A_M genera il punto di divisione, e A_R genera la parte della stringa che si trova interamente nella metà destra.

Quindi, $L_{half} = L(G')$, dimostrando che L_{half} è context-free quando $L(G)$ è context-free.

Ambiguità 3. Considera la grammatica:

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

(a) Dimostra che questa grammatica è ambigua mostrando due derivazioni diverse per la stringa "$a + a * a$".

(b) Riscrivere la grammatica in forma non ambigua per modellare le normali precedenze degli operatori (la moltiplicazione ha precedenza sull'addizione).

Soluzione. (a) Per dimostrare che la grammatica è ambigua, dobbiamo mostrare due derivazioni diverse che producono la stessa stringa.

Consideriamo la stringa "$a + a * a$". Possiamo derivarla in due modi diversi:

Derivazione 1 (associatività sinistra):

$$\begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow a + S \\ &\Rightarrow a + S * S \\ &\Rightarrow a + a * S \\ &\Rightarrow a + a * a \end{aligned}$$

Derivazione 2 (moltiplicazione ha precedenza):

$$\begin{aligned} S &\Rightarrow S + S \\ &\Rightarrow S + S * S \\ &\Rightarrow S + a * S \\ &\Rightarrow S + a * a \\ &\Rightarrow a + a * a \end{aligned}$$

Le due derivazioni sono diverse perché producono alberi sintattici diversi. - Nella prima derivazione, l'operazione di somma viene applicata al primo "a" e al risultato dell'espressione "$a * a$", quindi corrisponde a $a + (a * a)$. - Nella seconda derivazione, l'operazione di somma viene applicata al risultato dell'espressione "$a + a$" e al terzo "a", quindi corrisponde a $(a + a) * a$.

Questo dimostra che la grammatica è ambigua, poiché la stessa stringa può essere derivata in modi diversi che corrispondono a interpretazioni semantiche diverse dell'espressione.

(b) Per riscrivere la grammatica in forma non ambigua che rispetti le normali precedenze degli operatori (moltiplicazione ha precedenza sull'addizione), utilizziamo più simboli non-terminali per distinguere i diversi 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: - S è il simbolo iniziale - E rappresenta le espressioni che possono contenere l'operatore $+$ - T rappresenta i termini, che possono contenere l'operatore $*$ - F rappresenta i fattori, che possono essere atomi o espressioni tra parentesi

Questa grammatica non è ambigua e modella correttamente le normali precedenze degli operatori: - La moltiplicazione ha precedenza sull'addizione perché i termini T sono uniti da $+$ a livello di espressione E - I fattori F sono uniti da $*$ a livello di termine T - Le parentesi possono essere usate per modificare la precedenza

Usando questa grammatica, l'espressione "$a + a * a$" ha un'unica derivazione che corrisponde alla corretta interpretazione $a + (a * a)$:

$$\begin{aligned} S &\Rightarrow E \\ &\Rightarrow E + T \\ &\Rightarrow T + T \\ &\Rightarrow F + T \\ &\Rightarrow a + T \\ &\Rightarrow a + T * F \\ &\Rightarrow a + F * F \\ &\Rightarrow a + a * F \\ &\Rightarrow a + a * a \end{aligned}$$

Questa grammatica è non ambigua e genera lo stesso linguaggio della grammatica originale, con l'ulteriore proprietà di rispettare le normali precedenze degli operatori.

2 Esercizi su Automi a Pila

Costruzione di PDA 4. Progetta un PDA che accetti il linguaggio $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ e } i + j = k\}$.

Descrivi dettagliatamente tutti i componenti del PDA (stati, alfabeti, funzione di transizione) e spiega come funziona.

Soluzione. Progettiamo un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ che accetti il linguaggio $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ e } i + j = k\}$.

Definiamo i componenti del PDA:

- $Q = \{q_0, q_1, q_2, q_3\}$ (gli stati del PDA)
- $\Sigma = \{a, b, c\}$ (l'alfabeto di input)
- $\Gamma = \{Z_0, X\}$ (l'alfabeto della pila, dove Z_0 è il simbolo iniziale e X è un simbolo generico)
- q_0 è lo stato iniziale
- Z_0 è il simbolo iniziale della pila
- $F = \{q_3\}$ (lo stato finale)

La funzione di transizione δ è definita come:

- $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$ (leggiamo 'a', impiliamo X sopra Z_0)
- $\delta(q_0, a, X) = \{(q_0, XX)\}$ (leggiamo 'a', impiliamo X sopra X)
- $\delta(q_0, b, Z_0) = \{(q_1, XZ_0)\}$ (leggiamo 'b', impiliamo X sopra Z_0 e passiamo allo stato q_1)
- $\delta(q_0, b, X) = \{(q_1, XX)\}$ (leggiamo 'b', impiliamo X sopra X e passiamo allo stato q_1)
- $\delta(q_1, b, Z_0) = \{(q_1, XZ_0)\}$ (leggiamo 'b', impiliamo X sopra Z_0)
- $\delta(q_1, b, X) = \{(q_1, XX)\}$ (leggiamo 'b', impiliamo X sopra X)
- $\delta(q_1, c, Z_0) = \{(q_2, Z_0)\}$ (leggiamo 'c', manteniamo Z_0 e passiamo allo stato q_2)
- $\delta(q_1, c, X) = \{(q_2, \varepsilon)\}$ (leggiamo 'c', rimuoviamo X e passiamo allo stato q_2)
- $\delta(q_0, c, Z_0) = \{(q_2, Z_0)\}$ (leggiamo 'c', manteniamo Z_0 e passiamo allo stato q_2)
- $\delta(q_0, c, X) = \{(q_2, \varepsilon)\}$ (leggiamo 'c', rimuoviamo X e passiamo allo stato q_2)
- $\delta(q_2, c, Z_0) = \{(q_2, Z_0)\}$ (leggiamo 'c', manteniamo Z_0)

- $\delta(q_2, c, X) = \{(q_2, \varepsilon)\}$ (leggiamo 'c', rimuoviamo X)
- $\delta(q_2, \varepsilon, Z_0) = \{(q_3, Z_0)\}$ (se rimane solo Z_0 sulla pila, passiamo allo stato finale q_3)

Spiegazione del funzionamento: 1. Nello stato q_0, leggiamo e impiliamo un simbolo X per ogni 'a' nell'input. 2. Quando incontriamo il primo 'b', possiamo sia rimanere in q_0 e continuare a impilare, sia passare allo stato q_1. 3. Nello stato q_1, leggiamo e impiliamo un simbolo X per ogni 'b' nell'input. 4. Quando incontriamo il primo 'c', passiamo allo stato q_2 e iniziamo a rimuovere un simbolo X dalla pila per ogni 'c' nell'input. 5. Se dopo aver letto tutte le 'c', la pila contiene solo il simbolo iniziale Z_0 (ovvero, abbiamo rimosso tanti X quanti ne avevamo impilati, cioè $i + j = k$), allora passiamo allo stato finale q_3 e accettiamo.

Questo PDA accetta esattamente il linguaggio $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ e } i + j = k\}$.

Verifichiamo con un esempio: consideriamo la stringa $aabccc \in L$ con $i = 2, j = 1, k = 3$ (infatti $2+1 = 3$). 1. Iniziamo nello stato q_0 con la pila contenente solo Z_0. 2. Leggiamo 'a', impiliamo X sopra Z_0: pila $= XZ_0$. 3. Leggiamo 'a', impiliamo X sopra X: pila $= XXZ_0$. 4. Leggiamo 'b', impiliamo X sopra X e passiamo allo stato q_1: pila $= XXXZ_0$. 5. Leggiamo 'c', rimuoviamo X e passiamo allo stato q_2: pila $= XXZ_0$. 6. Leggiamo 'c', rimuoviamo X: pila $= XZ_0$. 7. Leggiamo 'c', rimuoviamo X: pila $= Z_0$. 8. La pila contiene solo Z_0, passiamo allo stato finale q_3 e accettiamo.

Quindi, il PDA accetta correttamente la stringa *aabccc*.

Nota: questo PDA accetta per stato finale. Se volessimo un PDA che accetti per pila vuota, dovremmo modificare leggermente la definizione.

Trasformazione PDA a CFG 5. Sia dato il seguente PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ dove:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{Z_0, X\}$
- $F = \{q_2\}$

Con funzione di transizione δ:

- $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$
- $\delta(q_0, a, X) = \{(q_0, XX)\}$
- $\delta(q_0, b, X) = \{(q_1, \varepsilon)\}$
- $\delta(q_1, b, X) = \{(q_1, \varepsilon)\}$
- $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$

Converti questo PDA in una grammatica context-free equivalente e semplifica la grammatica risultante il più possibile.

Soluzione. Per convertire il PDA dato in una grammatica context-free equivalente, utilizziamo l'algoritmo standard che crea una variabile per ogni tripla (p, A, q) dove p, q sono stati e A è un simbolo di pila.

Definiamo la grammatica $G = (V, \Sigma, R, S)$ dove:

- V è l'insieme dei non-terminali
- $\Sigma = \{a, b\}$ è l'alfabeto
- R è l'insieme delle regole di produzione
- S è il simbolo iniziale

Il simbolo iniziale S sarà $A_{q_0 Z_0 q_2}$, che rappresenta tutte le stringhe che permettono al PDA di passare dallo stato iniziale q_0 con Z_0 in cima alla pila allo stato finale q_2 con la pila invariata eccetto per Z_0.

Per ogni tripla (p, A, q) con $p, q \in Q$ e $A \in \Gamma$, creiamo un non-terminale A_{pAq} che genera tutte le stringhe che permettono al PDA di passare dallo stato p con A in cima alla pila allo stato q con la pila nello stesso stato (eccetto per A).

Ora generiamo le regole di produzione basate sulle transizioni del PDA:

1. Per la transizione $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$: - Per ogni $p, q \in Q$, aggiungiamo $A_{q_0 Z_0 q} \to a A_{q_0 X p} A_{p Z_0 q}$
2. Per la transizione $\delta(q_0, a, X) = \{(q_0, XX)\}$: - Per ogni $p, q \in Q$, aggiungiamo $A_{q_0 X q} \to a A_{q_0 X p} A_{p X q}$
3. Per la transizione $\delta(q_0, b, X) = \{(q_1, \varepsilon)\}$: - Aggiungiamo $A_{q_0 X q_1} \to b$
4. Per la transizione $\delta(q_1, b, X) = \{(q_1, \varepsilon)\}$: - Aggiungiamo $A_{q_1 X q_1} \to b$
5. Per la transizione $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$: - Aggiungiamo $A_{q_1 Z_0 q_2} \to \varepsilon$

Inoltre, per ogni stato $q \in Q$, aggiungiamo $A_{qAq} \to \varepsilon$ per ogni simbolo di pila $A \in \Gamma$.

Quindi, abbiamo le seguenti regole iniziali:

$$
\begin{aligned}
A_{q_0 Z_0 q_2} &\to a A_{q_0 X q_0} A_{q_0 Z_0 q_2} \mid a A_{q_0 X q_1} A_{q_1 Z_0 q_2} \mid a A_{q_0 X q_2} A_{q_2 Z_0 q_2} \\
A_{q_0 X q_0} &\to a A_{q_0 X q_0} A_{q_0 X q_0} \mid a A_{q_0 X q_1} A_{q_1 X q_0} \mid a A_{q_0 X q_2} A_{q_2 X q_0} \mid \varepsilon \\
A_{q_0 X q_1} &\to a A_{q_0 X q_0} A_{q_0 X q_1} \mid a A_{q_0 X q_1} A_{q_1 X q_1} \mid a A_{q_0 X q_2} A_{q_2 X q_1} \mid b \\
A_{q_0 X q_2} &\to a A_{q_0 X q_0} A_{q_0 X q_2} \mid a A_{q_0 X q_1} A_{q_1 X q_2} \mid a A_{q_0 X q_2} A_{q_2 X q_2} \mid \varepsilon \\
A_{q_1 X q_0} &\to \varepsilon \\
A_{q_1 X q_1} &\to b \mid \varepsilon \\
A_{q_1 X q_2} &\to \varepsilon \\
A_{q_2 X q_0} &\to \varepsilon \\
A_{q_2 X q_1} &\to \varepsilon \\
A_{q_2 X q_2} &\to \varepsilon \\
A_{q_1 Z_0 q_2} &\to \varepsilon \\
A_{q_0 Z_0 q_0} &\to a A_{q_0 X q_0} A_{q_0 Z_0 q_0} \mid a A_{q_0 X q_1} A_{q_1 Z_0 q_0} \mid a A_{q_0 X q_2} A_{q_2 Z_0 q_0} \mid \varepsilon \\
A_{q_0 Z_0 q_1} &\to a A_{q_0 X q_0} A_{q_0 Z_0 q_1} \mid a A_{q_0 X q_1} A_{q_1 Z_0 q_1} \mid a A_{q_0 X q_2} A_{q_2 Z_0 q_1} \\
A_{q_2 Z_0 q_0} &\to \varepsilon \\
A_{q_2 Z_0 q_1} &\to \varepsilon \\
A_{q_2 Z_0 q_2} &\to \varepsilon
\end{aligned}
$$

Questa grammatica è piuttosto complessa. Analizziamo il comportamento del PDA per semplificarla.

Il PDA dato accetta stringhe della forma $a^n b^n$ per $n \geq 1$. Funziona così: - Legge una serie di simboli 'a' e per ciascuno impila un simbolo X. - Poi legge una serie di simboli 'b' e per ciascuno rimuove un simbolo X. - Alla fine, se la pila contiene solo Z_0, passa allo stato finale q_2 e accetta.

Una grammatica context-free equivalente molto più semplice è:

$$S \to aAb$$
$$A \to aAb \mid \varepsilon$$

Questa grammatica genera il linguaggio $\{a^n b^n \mid n \geq 1\}$, che è esattamente il linguaggio accettato dal PDA dato.

Per verificare: - La regola $S \to aAb$ genera un 'a' all'inizio e un 'b' alla fine, con A nel mezzo. - La regola $A \to aAb$ genera coppie bilanciate di 'a' e 'b'. - La regola $A \to \varepsilon$ permette di terminare la generazione.

Possiamo confermare che questa grammatica semplificata è equivalente al PDA analizzando i passi di computazione: 1. Il PDA, partendo dallo stato q_0 con Z_0 sulla pila, legge 'a' e impila X sopra Z_0. 2. Per ogni 'a' aggiuntivo, il PDA impila un altro X. 3. Quando inizia a leggere 'b', il PDA passa allo stato q_1 e rimuove un X per ogni 'b'. 4. Dopo aver rimosso tutti gli X (che accade esattamente quando il numero di 'b' è uguale al numero di 'a'), rimane solo Z_0 sulla pila. 5. Il PDA passa quindi allo stato finale q_2 con ε-transizione e accetta.

Questa sequenza corrisponde alle derivazioni nella grammatica semplificata, dove ogni coppia (a, b) generata rappresenta un'operazione di impilare e poi rimuovere un simbolo X.

3 Esercizi Avanzati

Pumping lemma esteso 6. In questo esercizio esploreremo una versione più forte del pumping lemma per linguaggi context-free.

Teorema (Pumping lemma esteso): Se L è un linguaggio context-free, allora esiste una costante $p > 0$ tale che ogni stringa $z \in L$ con $|z| \geq p$ può essere scritta come $z = uvwxy$ con:

1. $|vwx| \leq p$
2. $|vx| > 0$
3. Per ogni $i \geq 0$ e per ogni $j \geq 0$, $uv^i wx^j y \in L$

Dimostra che i seguenti linguaggi non sono context-free utilizzando questo pumping lemma esteso:

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

(b) $L_2 = \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Soluzione. Il pumping lemma esteso è una versione più forte del pumping lemma standard per linguaggi context-free e ci permette di "pompare" indipendentemente due parti di una stringa.

(a) Dimostriamo che $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ non è context-free.

Supponiamo per assurdo che L_1 sia context-free. Allora, per il pumping lemma esteso, esiste una costante $p > 0$ tale che ogni stringa $z \in L_1$ con $|z| \geq p$ può essere scritta come $z = uvwxy$ con:

1. $|vwx| \leq p$
2. $|vx| > 0$
3. Per ogni $i \geq 0$ e per ogni $j \geq 0$, $uv^i wx^j y \in L_1$

Consideriamo la stringa $z = a^p b^p c^p \in L_1$. Per il pumping lemma esteso, possiamo scrivere $z = uvwxy$ con le proprietà sopra elencate.

Poiché $|vwx| \leq p$, la sottostringa vwx può contenere al massimo caratteri di due dei tre blocchi (al massimo, potrebbe contenere tutti gli a, o tutti gli a e alcuni b, o alcuni a e alcuni b, o tutti i b, o alcuni b e alcuni c, o tutti i b e alcuni c, o tutti i c).

Consideriamo i diversi casi possibili:

Caso 1: vwx contiene solo caratteri 'a'. Allora $v = a^r$ e $x = a^s$ per qualche $r, s \geq 0$ con $r + s > 0$ (poiché $|vx| > 0$). Consideriamo la stringa $uv^2wx^2y = uv \cdot v \cdot w \cdot x \cdot x \cdot y = a^{p+r+s}b^p c^p$. Questa stringa non è in L_1 perché il numero di 'a' $(p + r + s)$ è diverso dal numero di 'b' (p) e di 'c' (p).

Caso 2: vwx contiene caratteri 'a' e 'b'. Allora $v = a^r b^t$ e $x = a^s b^u$ per qualche $r, s, t, u \geq 0$ con $r + s + t + u > 0$ e almeno uno tra r, s, t, u deve essere positivo. Consideriamo la stringa $uv^0wx^0y = uwy = a^{p-r-s}b^{p-t-u}c^p$. Questa stringa non è in L_1 perché il numero di 'a' $(p - r - s)$ o il numero di 'b' $(p - t - u)$ è diverso dal numero di 'c' (p).

Caso 3: vwx contiene solo caratteri 'b'. Analogo al Caso 1, arriviamo a una contraddizione.

Caso 4: vwx contiene caratteri 'b' e 'c'. Analogo al Caso 2, arriviamo a una contraddizione.

Caso 5: vwx contiene solo caratteri 'c'. Analogo al Caso 1, arriviamo a una contraddizione.

In tutti i casi, possiamo costruire una stringa che dovrebbe essere in L_1 secondo il pumping lemma esteso, ma che non è in L_1. Questo è una contraddizione.

Pertanto, $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ non è context-free.

(b) Dimostriamo che $L_2 = \{a^n b^m c^m d^n \mid n, m \geq 1\}$ non è context-free.

Supponiamo per assurdo che L_2 sia context-free. Allora, per il pumping lemma esteso, esiste una costante $p > 0$ tale che ogni stringa $z \in L_2$ con $|z| \geq p$ può essere scritta come $z = uvwxy$ con le proprietà descritte sopra.

Consideriamo la stringa $z = a^p b^p c^p d^p \in L_2$. Per il pumping lemma esteso, possiamo scrivere $z = uvwxy$.

Poiché $|vwx| \leq p$, la sottostringa vwx può contenere al massimo caratteri di due dei quattro blocchi. Analizziamo i diversi casi:

Caso 1: vwx contiene solo caratteri 'a' o solo caratteri 'd', o una combinazione di 'a' e 'd'. Questo caso è simile al Caso 1 dell'analisi di L_1. Possiamo prendere $i = 2$ e $j = 0$ per ottenere una stringa con un numero sbilanciato di 'a' e 'd', che non è in L_2.

Caso 2: vwx contiene solo caratteri 'b' o solo caratteri 'c', o una combinazione di 'b' e 'c'. Possiamo prendere $i = 0$ e $j = 2$ per ottenere una stringa con un numero sbilanciato di 'b' e 'c', che non è in L_2.

Caso 3: vwx contiene una combinazione di 'a' e 'b', o 'b' e 'c', o 'c' e 'd'. In tutti questi casi, possiamo trovare valori di i e j che producono una stringa non in L_2, poiché alterando le quantità di caratteri adiacenti separatamente, rompiamo le relazioni che definiscono L_2.

Per esempio, se vwx contiene 'a' e 'b', possiamo scegliere $i = 2$ e $j = 0$ per aumentare il numero di 'a' e 'b' in modo non bilanciato rispetto a 'c' e 'd'.

In ogni caso, possiamo costruire una stringa che dovrebbe essere in L_2 secondo il pumping lemma esteso, ma che non è in L_2. Questo è una contraddizione.

Pertanto, $L_2 = \{a^n b^m c^m d^n \mid n, m \geq 1\}$ non è context-free.

Nota: il pumping lemma esteso è particolarmente utile per dimostrare che linguaggi come L_2 non sono context-free, poiché permette di "pompare" parti della stringa indipendentemente, rompendo le relazioni tra blocchi di caratteri distanti tra loro nella stringa.

Prefissi propri 7. Sia L un linguaggio context-free sull'alfabeto Σ. Definiamo l'insieme di tutti i prefissi propri di L come:

$$Prefix(L) = \{x \in \Sigma^* \mid \exists y \in \Sigma^+ : xy \in L\}$$

Dimostra che $Prefix(L)$ è sempre context-free se L è context-free.

Suggerimento: Modifica la grammatica che genera L in modo che ogni derivazione possa "terminare prematuramente".

Soluzione. Per dimostrare che $Prefix(L)$ è context-free quando L è context-free, costruiremo una grammatica per $Prefix(L)$ a partire da una grammatica per L.

Sia $G = (V, \Sigma, P, S)$ una grammatica context-free che genera L. Definiamo una nuova grammatica $G' = (V', \Sigma, P', S)$ che genera $Prefix(L)$ come segue:

$V' = V \cup \{S'\}$, dove S' è un nuovo simbolo non-terminale che fungerà da simbolo iniziale di G'.

Le regole di produzione P' includono:

- Tutte le regole originali di P
- Per ogni regola $A \to \alpha$ in P, dove $\alpha = X_1 X_2 \dots X_n$ (con $X_i \in V \cup \Sigma$ per $1 \leq i \leq n$), aggiungiamo le seguenti regole a P':
  - $A \to X_1 X_2 \dots X_{i-1} S'$ per ogni $1 \leq i \leq n+1$
- $S' \to \varepsilon$

Il simbolo iniziale di G' è S (lo stesso di G).

L'idea è che le nuove regole permettono di "terminare prematuramente" qualsiasi derivazione in G, effettivamente generando tutti i possibili prefissi delle stringhe in L. Il simbolo S' ci permette di "tagliare" una derivazione in qualsiasi punto e terminare.

Tuttavia, questa costruzione presenta un problema: genera tutti i prefissi di L, inclusi quelli che sono già in L, mentre vogliamo solo i prefissi propri. Per risolvere questo problema, modifichiamo leggermente la costruzione.

Definiamo una grammatica $G'' = (V'', \Sigma, P'', S'')$ che genera $Prefix(L)$ come segue:

$V'' = V \cup \{S'', T\}$, dove S'' e T sono nuovi simboli non-terminali.

Le regole di produzione P'' includono:

- Tutte le regole originali di P

- Una nuova regola $S'' \to T$
- Per ogni regola $A \to \alpha$ in P, dove $\alpha = X_1X_2 \dots X_n$ (con $X_i \in V \cup \Sigma$ per $1 \leq i \leq n$), aggiungiamo le seguenti regole a P'':
  - $T \to X_1X_2 \dots X_iT$ per ogni $0 \leq i < n$
- $T \to \varepsilon$

Il simbolo iniziale di G'' è S''.

Questa costruzione genera tutti i prefissi di stringhe in L, ma per assicurarci di generare solo i prefissi propri, dobbiamo fare un'ulteriore modifica.

Definiamo una grammatica $G''' = (V''', \Sigma, P''', S''')$ che genera esattamente $Prefix(L) - L$ (i prefissi propri di L):

$V''' = V \cup \{S''', A_1, A_2, \dots, A_k\}$, dove S''' è un nuovo simbolo iniziale e $A_1, A_2, \dots, A_k$ sono nuovi simboli non-terminali corrispondenti alle regole in P.

Le regole di produzione P''' includono:

- $S''' \to A_i$ per ogni regola $S \to \alpha_i$ in P
- Per ogni regola $A \to \alpha$ in P, dove $\alpha = X_1X_2 \dots X_n$ (con $X_i \in V \cup \Sigma$ per $1 \leq i \leq n$), aggiungiamo:
  - $A_i \to X_1X_2 \dots X_j$ per ogni $1 \leq j < n$ se $X_j \in \Sigma$ (genera un prefisso proprio terminando con un terminale)
  - $A_i \to X_1X_2 \dots X_jA_l$ per ogni $1 \leq j < n$ e per ogni regola $X_j \to \beta_l$ in P se $X_j \in V$ (continua la derivazione)

Tuttavia, questa costruzione diventa rapidamente complessa e difficile da formalizzare completamente.

Un approccio alternativo, più semplice, è utilizzare un PDA. Dato che L è context-free, esiste un PDA P che accetta L. Possiamo costruire un nuovo PDA P' che accetta $Prefix(L)$ come segue:

P' simula P su ogni input, ma in qualsiasi momento può non deterministicamente decidere di accettare la stringa corrente, purché rimanga da leggere almeno un simbolo dell'input originale (per garantire che sia un prefisso proprio).

Formalmente, se $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, allora $P' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$ dove:

- $Q' = Q \cup \{q_f\}$, dove q_f è un nuovo stato
- $\Gamma' = \Gamma \cup \{\$\}$, dove \$ è un nuovo simbolo di pila
- $q_0' = q_0$ (stesso stato iniziale)
- $Z_0' = Z_0$ (stesso simbolo iniziale della pila)
- $F' = \{q_f\}$ (il nuovo stato è l'unico stato finale)
- δ' include tutte le transizioni di δ, più:
  - $\delta'(q, \varepsilon, \gamma) = \{(q_f, \gamma)\}$ per ogni $q \in Q, \gamma \in \Gamma$

Questo PDA simula il PDA originale, ma può non deterministicamente passare allo stato q_f in qualsiasi momento, accettando così ogni prefisso di una stringa in L.

Per ottenere solo i prefissi propri, modifichiamo ulteriormente P':

- P' accetta solo se ha letto almeno un simbolo e c'è ancora almeno un simbolo nell'input che non è stato letto

Poiché P' è un PDA, il linguaggio $Prefix(L)$ è context-free.

Un altro approccio ancora è osservare che $Prefix(L) = \{x \in \Sigma^* \mid x\Sigma^+ \cap L \neq \emptyset\}$. Dato che Σ^+ è regolare e l'intersezione con un linguaggio regolare preserva la context-free-ness, così come l'operazione di proiezione (che elimina la parte Σ^+), possiamo concludere che $Prefix(L)$ è context-free quando L è context-free.

linguaggio delle shufflazioni 8. Data una stringa w, definiamo una *2-shufflazione* di w come una stringa ottenuta dividendo w in due parti e poi intercalando i caratteri delle due parti in modo arbitrario. Ad esempio, se $w = abcd$, una possibile 2-shufflazione è $acbd$ (ottenuta intercalando ab e cd).

Dato un linguaggio L, definiamo il linguaggio delle 2-shufflazioni di L come:

$$Shuffle_2(L) = \{z \mid z \text{ è una 2-shufflazione di qualche stringa } w \in L\}$$

Dimostra che se L è context-free, allora anche $Shuffle_2(L)$ è context-free.

Soluzione. Per dimostrare che $Shuffle_2(L)$ è context-free quando L è context-free, costruiremo un PDA che accetta $Shuffle_2(L)$ utilizzando un PDA per L.

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA che accetta L per stato finale. Descriviamo la costruzione di un PDA P' che accetta $Shuffle_2(L)$.

L'idea principale è che P' deve: 1. Indovinare la divisione della stringa originale $w \in L$ in due parti u e v 2. Indovinare, per ogni carattere dell'input, se esso proviene da u o da v 3. Simulare il PDA P su una ricostruzione di w per verificare che $w \in L$

Il PDA P' utilizza la pila per memorizzare i caratteri di input che provengono dalla seconda parte v, mentre elabora immediatamente i caratteri che provengono dalla prima parte u. Quando ha letto tutti i caratteri di u, passa a una seconda fase in cui elabora i caratteri memorizzati nella pila, simulando così l'elaborazione di $w = uv$ da parte di P.

Definiamo formalmente $P' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$ dove:

- $Q' = Q \cup \{q_0', q_1', q_2'\} \cup \{q_1, q_2 \mid q \in Q\}$ (aggiungiamo nuovi stati)
- $\Gamma' = \Gamma \cup (\Sigma \times \{1, 2\}) \cup \{\$\}$ (estendiamo l'alfabeto della pila con coppie (simbolo, parte) e un nuovo simbolo \$)
- q_0' è il nuovo stato iniziale
- $Z_0' = Z_0$ (stesso simbolo iniziale della pila)
- $F' = F$ (stessi stati finali)
- δ' include le seguenti transizioni:
  - $\delta'(q_0', \varepsilon, Z_0) = \{(q_1', Z_0)\}$ (inizializza la simulazione)
  - Per ogni $a \in \Sigma$:

        * $\delta'(q_1', a, \gamma) = \{(q_1', (a, 2)\gamma), (q_2', \gamma)\}$ (non deterministicamente decide se a appartiene alla prima o alla seconda parte)
    - $\delta'(q_2', \varepsilon, \gamma) = \{(q_0, \gamma)\}$ (passa alla simulazione di P)
    - Per ogni transizione $(p, b, \gamma) \to (q, \eta)$ in δ:
        * $\delta'(p, a, \gamma) = \{(q, \eta)\}$ per ogni $a \in \Sigma$ (simula le transizioni di P)
    - Per ogni $(a, 2) \in \Sigma \times \{2\}$ e per ogni stato $q \in Q$:
        * $\delta'(q, \varepsilon, (a, 2)) = \{(q_a, \varepsilon)\}$ (simula la lettura di a dalla seconda parte)
        * Per ogni transizione $(q, a, \gamma) \to (p, \eta)$ in δ:
            · $\delta'(q_a, \varepsilon, \gamma) = \{(p, \eta)\}$ (continua la simulazione di P dopo aver "letto" a)

Questa costruzione è piuttosto complessa, quindi spieghiamo intuitivamente come funziona:

1. Nello stato q_1', il PDA legge i caratteri dell'input e non deterministicamente decide se ciascun carattere appartiene alla prima parte u o alla seconda parte v della stringa originale w. 2. Se decide che un carattere appartiene alla prima parte, simula immediatamente il PDA P su quel carattere. 3. Se decide che un carattere appartiene alla seconda parte, lo memorizza nella pila con un'etichetta $(a, 2)$. 4. A un certo punto, non deterministicamente decide che ha finito di leggere la prima parte e passa allo stato q_2'. 5. Da q_2', passa allo stato iniziale q_0 di P e inizia la simulazione vera e propria. 6. Mentre legge il resto dell'input (che corrisponde a caratteri della prima parte mescolati con caratteri della seconda parte), simula il comportamento di P sui caratteri provenienti dalla prima parte. 7. Ogni volta che incontra un carattere della seconda parte nella pila (identificato dall'etichetta $(a, 2)$), simula il comportamento di P su quel carattere senza consumare input. 8. Il PDA accetta se, dopo aver letto tutto l'input e processato tutti i caratteri memorizzati nella pila, il PDA P sarebbe in uno stato finale.

Questo garantisce che P' accetta una stringa se e solo se è una 2-shufflazione di una stringa accettata da P, cioè una stringa in $Shuffle_2(L)$.

Un approccio alternativo, più elegante, è costruire una grammatica per $Shuffle_2(L)$ a partire da una grammatica per L.

Sia $G = (V, \Sigma, P, S)$ una grammatica context-free che genera L. Definiamo una nuova grammatica $G' = (V', \Sigma, P', S')$ che genera $Shuffle_2(L)$ come segue:

$V' = V \cup \{S'\} \cup \{A_1, A_2 \mid A \in V\}$, dove S' è un nuovo simbolo iniziale e A_1, A_2 sono nuovi non-terminali corrispondenti al non-terminale A di G per le due parti della stringa.

Le regole di produzione P' includono:

- $S' \to S_1 S_2$ (inizia la generazione con le due parti)
- Per ogni regola $A \to BC$ in P, aggiungiamo:
    - $A_1 \to B_1 C_1$ (applica la regola alla prima parte)
    - $A_2 \to B_2 C_2$ (applica la regola alla seconda parte)
- Per ogni regola $A \to a$ in P (dove $a \in \Sigma$), aggiungiamo:
    - $A_1 \to a$ (genera il terminale nella prima parte)

- $A_2 \to a$ (genera il terminale nella seconda parte)

- Aggiungiamo anche regole per permettere l'intreccio delle due parti:
  - $A_1 B_2 \to B_2 A_1$ per ogni $A, B \in V$ (permette di scambiare l'ordine)
  - $A_1 a \to a A_1$ per ogni $A \in V, a \in \Sigma$ (permette di scambiare un non-terminale con un terminale)
  - $a A_2 \to A_2 a$ per ogni $A \in V, a \in \Sigma$ (analogamente)

Questa grammatica genera tutte le possibili 2-shufflazioni delle stringhe in L. L'idea è che: 1. Generiamo separatamente le due parti della stringa originale utilizzando i non-terminali indicizzati con 1 e 2 2. Permettiamo ai terminali generati dalle due parti di mescolarsi liberamente usando le regole di scambio 3. Alla fine, ogni derivazione che termina con soli terminali rappresenta una possibile 2-shufflazione di una stringa in L

Poiché G' è una grammatica context-free, $Shuffle_2(L)$ è un linguaggio context-free.

PDA minimale 9. Si definisce la *dimensione* di un PDA come la somma del numero di stati e del numero di simboli di pila.

(a) Trova un PDA di dimensione minima che accetti il linguaggio $L = \{a^n b^n \mid n \geq 1\}$.

(b) Dimostra che il PDA da te costruito è effettivamente di dimensione minima (cioè, non esiste un PDA con dimensione inferiore che accetti lo stesso linguaggio).

Soluzione. (a) Costruiamo un PDA per il linguaggio $L = \{a^n b^n \mid n \geq 1\}$ e cerchiamo di minimizzare la sua dimensione (somma del numero di stati e del numero di simboli di pila).

Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ per questo linguaggio può essere definito come:

- $Q = \{q_0, q_1, q_2\}$ (tre stati: iniziale, di lettura delle 'b', finale)
- $\Sigma = \{a, b\}$ (l'alfabeto di input)
- $\Gamma = \{Z_0, X\}$ (due simboli di pila: il marcatore di fondo e un simbolo per contare)
- q_0 è lo stato iniziale
- Z_0 è il simbolo iniziale della pila
- $F = \{q_2\}$ (un solo stato finale)
- La funzione di transizione δ è definita come:
  - $\delta(q_0, a, Z_0) = \{(q_0, XZ_0)\}$ (leggiamo 'a' e impiliamo X sopra Z_0)
  - $\delta(q_0, a, X) = \{(q_0, XX)\}$ (leggiamo 'a' e impiliamo X sopra X)
  - $\delta(q_0, b, X) = \{(q_1, \varepsilon)\}$ (leggiamo 'b', rimuoviamo X e passiamo allo stato q_1)
  - $\delta(q_1, b, X) = \{(q_1, \varepsilon)\}$ (leggiamo 'b' e rimuoviamo X)
  - $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ (se rimane solo Z_0, passiamo allo stato finale)

La dimensione di questo PDA è $|Q| + |\Gamma| = 3 + 2 = 5$.

Possiamo ridurre questa dimensione? Proviamo a minimizzare il numero di stati e di simboli di pila.

Un PDA minimale $P' = (Q', \Sigma, \Gamma', \delta', q_0', Z_0', F')$ per L può essere:

- $Q' = \{q'_0, q'_1\}$ (due stati sono sufficienti: uno per la fase di lettura 'a' e uno per la fase di lettura 'b')
- $\Sigma = \{a, b\}$ (l'alfabeto di input rimane lo stesso)
- $\Gamma' = \{Z'_0\}$ (un solo simbolo di pila è sufficiente se usiamo la pila in modo binario)
- q'_0 è lo stato iniziale
- Z'_0 è il simbolo iniziale della pila
- $F' = \{q'_1\}$ (lo stato di lettura 'b' è anche lo stato finale)
- La funzione di transizione δ' è definita come:
  - $\delta'(q'_0, a, Z'_0) = \{(q'_0, Z'_0 Z'_0)\}$ (leggiamo 'a' e raddoppiamo il simbolo in cima alla pila)
  - $\delta'(q'_0, b, Z'_0) = \{(q'_1, \varepsilon)\}$ (leggiamo 'b', rimuoviamo un simbolo e passiamo allo stato q'_1)
  - $\delta'(q'_1, b, Z'_0) = \{(q'_1, \varepsilon)\}$ (leggiamo 'b' e rimuoviamo un simbolo)

La dimensione di questo PDA è $|Q'| + |\Gamma'| = 2 + 1 = 3$.

Questo PDA funziona come segue: 1. Per ogni 'a' letto, raddoppia il contenuto della pila (inizialmente un solo Z'_0) 2. Dopo aver letto n caratteri 'a', la pila contiene 2^n simboli Z'_0 3. Per ogni 'b' letto, rimuove un simbolo dalla pila 4. Accetta se, dopo aver letto tutti i caratteri 'b', la pila è vuota

Il problema di questo approccio è che genera un numero esponenziale di simboli sulla pila (per n caratteri 'a', genera 2^n simboli), il che non corrisponde esattamente al linguaggio $L = \{a^n b^n \mid n \geq 1\}$.

Quindi, dobbiamo utilizzare almeno due simboli di pila e due stati. Un PDA minimale corretto è:

- $Q'' = \{q''_0, q''_1\}$ (due stati)
- $\Gamma'' = \{Z''_0, X''\}$ (due simboli di pila)
- La funzione di transizione:
  - $\delta''(q''_0, a, Z''_0) = \{(q''_0, X''Z''_0)\}$
  - $\delta''(q''_0, a, X'') = \{(q''_0, X''X'')\}$
  - $\delta''(q''_0, b, X'') = \{(q''_1, \varepsilon)\}$
  - $\delta''(q''_1, b, X'') = \{(q''_1, \varepsilon)\}$
  - $\delta''(q''_1, \varepsilon, Z''_0) = \{(q''_1, Z''_0)\}$ (questa transizione non modifica la pila ma ci assicura di poter rimanere nello stato q''_1 anche dopo aver raggiunto Z''_0)

La dimensione di questo PDA è $|Q''| + |\Gamma''| = 2 + 2 = 4$.

(b) Dimostriamo che non esiste un PDA di dimensione inferiore a 4 che accetti il linguaggio $L = \{a^n b^n \mid n \geq 1\}$.

Supponiamo per assurdo che esista un PDA P con dimensione totale 3, cioè $|Q|+|\Gamma| = 3$.

Caso 1: $|Q| = 1$ e $|\Gamma| = 2$.

Con un solo stato, il PDA non può distinguere tra la fase di lettura delle 'a' e quella delle 'b'. Inoltre, per il pumping lemma per linguaggi context-free, un linguaggio come L richiede almeno un simbolo di pila oltre al simbolo iniziale per "contare" il numero di 'a'. Quindi, questo caso non è possibile.

Caso 2: $|Q| = 2$ e $|\Gamma| = 1$.

Con un solo simbolo di pila (il simbolo iniziale Z_0), il PDA non può contare correttamente il numero di 'a'. Infatti, l'unico modo per memorizzare informazione nella pila sarebbe replicare Z_0 un certo numero di volte. Ma senza altri simboli, non c'è modo di distinguere tra i diversi livelli della pila. Questo PDA potrebbe al massimo accettare linguaggi come $\{a^n b^m \mid n, m \geq 1\}$, ma non $L = \{a^n b^n \mid n \geq 1\}$.

Caso 3: $|Q| = 3$ e $|\Gamma| = 0$.

Questo caso è impossibile, poiché un PDA deve avere almeno un simbolo di pila (il simbolo iniziale).

Quindi, non esiste un PDA di dimensione inferiore a 4 che accetti il linguaggio L.

Il PDA che abbiamo costruito con $|Q| = 2$ e $|\Gamma| = 2$ ha dimensione 4 ed è quindi un PDA di dimensione minima per il linguaggio $L = \{a^n b^n \mid n \geq 1\}$.