Tutorato di Automi e Linguaggi Formali

Esercizi in preparazione al parziale

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

Anno Accademico 2024-2025

1 Linguaggi Regolari

Operazioni logiche bit a bit (da appello 16/4/2024) 1. L'or bit a bit è un'operazione binaria che prende due stringhe binarie di uguale lunghezza ed esegue l'or logico su ogni coppia di bit corrispondenti. Il risultato è una stringa binaria in cui ogni posizione è 0 se entrambi i bit sono 0, 1 altrimenti.

Date due stringhe binarie x e y di uguale lunghezza, $x \vee y$ rappresenta l'or bit a bit di x e y. Per esempio, $0011 \vee 0101 = 0111$.

Dimostra che se L ed M sono linguaggi regolari sull'alfabeto $\{0, 1\}$, allora anche il seguente linguaggio è regolare:

$$L \vee M = \{x \vee y \mid x \in L, y \in M \text{ e } |x| = |y|\}.$$

Suggerimento: Considerare il prodotto cartesiano degli automi per L e M.

Soluzione. Sia $A_L = (Q_L, \Sigma, \delta_L, q_{0L}, F_L)$ un DFA che accetta L e $A_M = (Q_M, \Sigma, \delta_M, q_{0M}, F_M)$ un DFA che accetta M.

Costruiamo un nuovo DFA $A = (Q, \Sigma, \delta, q_0, F)$ per il linguaggio $L \vee M$ come segue:

- $Q = Q_L \times Q_M$ (prodotto cartesiano degli stati)
- $\Sigma = \{0, 1\}$ (stesso alfabeto degli automi originali)
- Lo stato iniziale è $q_0 = (q_{0L}, q_{0M})$
- Gli stati finali sono $F = F_L \times F_M$ (coppie di stati finali)
- Per la funzione di transizione, per ogni $(p, q) \in Q$ e per ogni bit $b \in \{0, 1\}$ risultante dall'operazione OR, definiamo:

$$\delta((p,q),b) = \begin{cases} (\delta_L(p,0), \delta_M(q,0)) & \text{se } b = 0 \\ (\delta_L(p,0), \delta_M(q,1)) \text{ OR } (\delta_L(p,1), \delta_M(q,0)) \text{ OR } (\delta_L(p,1), \delta_M(q,1)) & \text{se } b = 1 \end{cases}$$

Osserviamo che abbiamo costruito un NFA, perché per $b = 1$ possiamo avere più transizioni possibili.

Correttezza: L'automa A simula l'esecuzione simultanea di A_L e A_M su input che, quando combinati con OR, producono la stringa di input. Quando legge un simbolo b, A considera tutte le possibili coppie di input ai DFA originali che, quando combinati con OR, danno b:

- Se $b = 0$, allora sia il bit di L che quello di M devono essere 0
- Se $b = 1$, allora almeno uno tra il bit di L e quello di M deve essere 1

Quindi, A accetta una stringa z se e solo se esistono stringhe $x \in L$ e $y \in M$ con $|x| = |y| = |z|$ tali che $z = x \vee y$.

Poiché possiamo costruire questo NFA e sappiamo che ogni NFA può essere convertito in un DFA equivalente, il linguaggio $L \vee M$ è regolare.

Esercizio 2. Considera l'operazione SWAP che scambia di posizione i caratteri della stringa a due a due:

$$\textsc{Swap}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ a & \text{se } w = a \text{ con } a \in \Sigma \\ a_1 a_0 \textsc{Swap}(u) & \text{se } w = a_0 a_1 u \text{ con } a_0, a_1 \in \Sigma, u \in \Sigma^* \end{cases}$$

Per esempio, $\textsc{Swap}(\texttt{ABCDE}) = \texttt{BADCE}$.

Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio regolare, allora anche il seguente linguaggio è regolare:

$$\textsc{Swap}(L) = \{\textsc{Swap}(w) \mid w \in L\}.$$

Suggerimento: Costruire un NFA che simula gli stati dell'automa per L mentre legge i caratteri nell'ordine corretto dopo lo swap.

Soluzione. Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA che accetta il linguaggio regolare L. Costruiamo un NFA $A' = (Q', \Sigma, \delta', q_0', F')$ che accetta $\textsc{Swap}(L)$.

L'idea principale è che mentre leggiamo una stringa $w' = \textsc{Swap}(w)$, vogliamo simulare l'elaborazione di w (la stringa originale) nell'automa A. Per farlo, dobbiamo invertire l'ordine delle coppie di caratteri mentre leggiamo la stringa.

Definiamo A' come segue:

- $Q' = Q \cup (Q \times \Sigma)$ (ogni stato può avere anche un carattere "in attesa")
- $q_0' = q_0$ (stesso stato iniziale)
- $F' = F$ (stessi stati finali)
- La funzione di transizione δ' è definita come:
  - Per $q \in Q$ e $a \in \Sigma$: $\delta'(q, a) = \{(q, a)\}$ (memorizza il primo carattere di una coppia)
  - Per $(q, a) \in Q \times \Sigma$ e $b \in \Sigma$: $\delta'((q, a), b) = \delta(\delta(q, b), a)$ (processa prima b poi a, invertendo l'ordine)

Per gestire correttamente le stringhe di lunghezza dispari, aggiungiamo anche:

- Per $(q,a) \in Q \times \Sigma$: se $q \in F$, allora $(q,a) \in F'$ (permette di accettare stringhe con un carattere rimanente)

Correttezza:

- Se $w = \varepsilon$, allora $\textsc{Swap}(w) = \varepsilon$ e A' accetta perché $q_0 \in F$ se e solo se $\varepsilon \in L$.
- Se $w = a$ (lunghezza 1), allora $\textsc{Swap}(w) = a$ e A' accetta a se e solo se $\delta(q_0, a) \in F$, come richiesto.
- Se $w = a_0a_1a_2...a_{n-1}$ con n pari, allora $\textsc{Swap}(w) = a_1a_0a_3a_2...a_{n-1}a_{n-2}$. L'automa A' elabora le coppie $(a_1, a_0), (a_3, a_2), ..., (a_{n-1}, a_{n-2})$ invertendo l'ordine di ogni coppia, simulando effettivamente l'elaborazione di w in A.
- Se $w = a_0a_1a_2...a_{n-1}a_n$ con n dispari, allora $\textsc{Swap}(w) = a_1a_0a_3a_2...a_na_{n-1}$. L'automa gestisce anche questo caso grazie alla regola aggiuntiva per gli stati (q, a).

Quindi, A' accetta una stringa w' se e solo se esiste una stringa $w \in L$ tale che $w' = \textsc{Swap}(w)$, il che significa che A' accetta esattamente $\textsc{Swap}(L)$.

Poiché abbiamo costruito un NFA e ogni NFA può essere convertito in un DFA equivalente, il linguaggio $\textsc{Swap}(L)$ è regolare.

Chiusura rispetto al complemento di prefissi 3. Ricordiamo che una stringa x è un *prefisso* di una stringa y se esiste una stringa z tale che $xz = y$, e che x è un *prefisso proprio* di y se, in aggiunta, $x \neq y$.

Dimostra che la classe dei linguaggi regolari è chiusa sotto l'operazione:

$$\textsc{NoPrefix}(A) = \{w \in A \mid \text{nessun prefisso proprio di } w \text{ è un membro di } A\}.$$

Suggerimento: Considerare l'insieme degli stati attraversati durante l'elaborazione di w nell'automa deterministico per A.

Soluzione. Sia A un linguaggio regolare e sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che accetta A.

Costruiamo un nuovo DFA $M' = (Q', \Sigma, \delta', q_0', F')$ che accetta $\textsc{NoPrefix}(A)$ come segue:

- $Q' = Q \times \{0, 1\}$ (aggiungiamo un bit per tenere traccia se abbiamo incontrato uno stato finale)
- $q_0' = (q_0, 0)$ se $q_0 \notin F$, altrimenti $q_0' = (q_0, 1)$
- La funzione di transizione è definita come:

$$\delta'((q,b),a) = \begin{cases} (\delta(q,a), 1) & \text{se } \delta(q,a) \in F \text{ o } b = 1 \\ (\delta(q,a), 0) & \text{altrimenti} \end{cases}$$

- $F' = \{(q, 0) \mid q \in F\}$ (accettiamo solo se non abbiamo incontrato stati finali lungo il percorso)

L'idea è che il bit aggiuntivo (il secondo componente degli stati di M') viene impostato a 1 non appena incontriamo uno stato finale di M durante l'elaborazione della stringa. Se raggiungiamo uno stato finale di M e il bit è ancora 0, significa che non abbiamo incontrato prefissi propri che appartengono ad A, quindi la stringa è in $\textsc{NoPrefix}(A)$.

Correttezza: Sia $w \in \Sigma^*$. Dimostriamo che $w \in \textsc{NoPrefix}(A)$ se e solo se M' accetta w.

($\Rightarrow$) Supponiamo che $w \in \textsc{NoPrefix}(A)$. Allora:

- $w \in A$ (per definizione di $\textsc{NoPrefix}(A)$)
- Nessun prefisso proprio di w è in A

Quindi, quando M' elabora w, il bit rimane 0 fino a quando non raggiunge l'ultimo stato (che è in F perché $w \in A$). Poiché il bit è 0 e lo stato è in F, l'automa M' accetta w.

($\Leftarrow$) Supponiamo che M' accetti w. Allora:

- w termina in uno stato $(q, 0)$ con $q \in F$
- Il bit è 0, quindi nessun prefisso proprio di w porta a uno stato finale in M

Questo significa che $w \in A$ (perché $q \in F$) e nessun prefisso proprio di w è in A (perché il bit è 0). Quindi, $w \in \textsc{NoPrefix}(A)$.

Poiché M' è un DFA che accetta esattamente $\textsc{NoPrefix}(A)$, il linguaggio $\textsc{NoPrefix}(A)$ è regolare.

Perfect Shuffle 4. Per i linguaggi A e B, definiamo il *perfect shuffle* di A e B come il linguaggio

$$\textsc{PerfShuffle}(A, B) = \{a_1b_1a_2b_2 \cdots a_nb_n \mid a_1a_2 \cdots a_n \in A, b_1b_2 \cdots b_n \in B, a_i, b_i \in \Sigma\}.$$

Dimostra che la classe dei linguaggi regolari è chiusa sotto perfect shuffle.

Soluzione. Siano A e B linguaggi regolari. Sia $M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ un DFA che accetta A e $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ un DFA che accetta B.

Costruiamo un NFA $M = (Q, \Sigma, \delta, q_0, F)$ che accetta $\textsc{PerfShuffle}(A, B)$ come segue:

- $Q = Q_A \times Q_B \times \{0, 1\}$ (dove il terzo componente indica se dobbiamo leggere un carattere per A (0) o per B (1))
- $q_0 = (q_{0A}, q_{0B}, 0)$ (iniziamo leggendo un carattere per A)
- $F = F_A \times F_B \times \{1\}$ (l'automa deve terminare dopo aver letto un carattere per B)
- La funzione di transizione è definita come:

$$\delta((p, q, 0), a) = \{(\delta_A(p, a), q, 1)\}$$
$$\delta((p, q, 1), a) = \{(p, \delta_B(q, a), 0)\}$$

L'idea è che l'automa M tiene traccia dello stato corrente in entrambi gli automi M_A e M_B, e alterna tra il processamento di caratteri per A e B. Inizia leggendo un carattere per A (stato con terzo componente 0), poi legge un carattere per B (stato con terzo componente 1), e così via.

Correttezza: Sia $w = c_1 c_2 \cdots c_{2n} \in \Sigma^*$ una stringa di lunghezza pari. Dimostriamo che $w \in \textsc{PerfShuffle}(A, B)$ se e solo se M accetta w.

($\Rightarrow$) Supponiamo che $w \in \textsc{PerfShuffle}(A, B)$. Allora esistono stringhe $x = a_1 a_2 \cdots a_n \in A$ e $y = b_1 b_2 \cdots b_n \in B$ tali che $w = a_1 b_1 a_2 b_2 \cdots a_n b_n$.

Quando M elabora w, legge prima a_1 e aggiorna lo stato di M_A, poi legge b_1 e aggiorna lo stato di M_B, e così via. Dopo aver letto l'intera stringa, lo stato di M_A sarà quello raggiunto dopo aver letto x, che è uno stato finale (perché $x \in A$), e lo stato di M_B sarà quello raggiunto dopo aver letto y, che è uno stato finale (perché $y \in B$). Quindi, M termina in uno stato $(p, q, 1)$ con $p \in F_A$ e $q \in F_B$, cioè uno stato finale.

($\Leftarrow$) Supponiamo che M accetti $w = c_1 c_2 \cdots c_{2n}$. Allora M termina in uno stato $(p, q, 1)$ con $p \in F_A$ e $q \in F_B$.

Durante l'elaborazione di w, M ha letto i caratteri in posizioni dispari $(c_1, c_3, \ldots, c_{2n-1})$ per aggiornare lo stato di M_A, e i caratteri in posizioni pari $(c_2, c_4, \ldots, c_{2n})$ per aggiornare lo stato di M_B. Poiché M termina con $p \in F_A$, la stringa $x = c_1 c_3 \cdots c_{2n-1}$ è accettata da M_A, quindi $x \in A$. Analogamente, poiché M termina con $q \in F_B$, la stringa $y = c_2 c_4 \cdots c_{2n}$ è accettata da M_B, quindi $y \in B$.

Quindi, $w = c_1 c_2 \cdots c_{2n} = c_1 c_3 \cdots c_{2n-1} \text{shuffle} c_2 c_4 \cdots c_{2n} = x \text{shuffle} y$, con $x \in A$ e $y \in B$, cioè $w \in \textsc{PerfShuffle}(A, B)$.

Nel caso di stringhe di lunghezza dispari, l'automa M non può raggiungere uno stato finale, quindi non accetta tali stringhe, coerentemente con la definizione di $\textsc{PerfShuffle}(A, B)$.

Poiché M è un NFA che accetta esattamente $\textsc{PerfShuffle}(A, B)$, e ogni NFA può essere convertito in un DFA equivalente, il linguaggio $\textsc{PerfShuffle}(A, B)$ è regolare.

Trasformazione automi 5. Sia M un DFA con insieme degli stati Q, alfabeto Σ, funzione di transizione δ, stato iniziale q_0 e insieme degli stati finali F. Definisci un nuovo DFA M' che accetta esattamente le stringhe che vengono accettate da M dopo esattamente k passi, dove k è un numero naturale fissato. Dimostrane la correttezza.

Soluzione. Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA. Vogliamo costruire un DFA $M' = (Q', \Sigma, \delta', q'_0, F')$ che accetta esattamente le stringhe che vengono accettate da M dopo esattamente k passi.

Definiamo M' come segue:

- $Q' = Q \times \{0, 1, 2, \ldots, k\}$ (aggiungiamo un contatore per tenere traccia del numero di passi)
- $q'_0 = (q_0, 0)$ (iniziamo con 0 passi)
- La funzione di transizione è definita come:

$$\delta'((q, i), a) = \begin{cases} (\delta(q, a), i + 1) & \text{se } i < k \\ \text{non definita} & \text{se } i = k \end{cases}$$

- $F' = \{(q, k) \mid q \in F\}$ (accettiamo solo dopo esattamente k passi, e solo se lo stato è in F)

Correttezza: Sia $w \in \Sigma^*$ una stringa. Dimostriamo che w è accettata da M' se e solo se $|w| = k$ e w è accettata da M.

($\Rightarrow$) Supponiamo che w sia accettata da M'. Allora:

- M' termina in uno stato $(q, j) \in F'$ dopo aver letto w
- Per definizione di F', abbiamo $j = k$ e $q \in F$
- Il contatore j è stato incrementato esattamente $|w|$ volte (una volta per ogni carattere letto)
- Quindi, $|w| = k$
- Inoltre, M termina nello stato $q \in F$ dopo aver letto w, quindi w è accettata da M

($\Leftarrow$) Supponiamo che $|w| = k$ e w sia accettata da M. Allora:

- M termina in uno stato $q \in F$ dopo aver letto w
- Durante l'elaborazione di w in M', il contatore viene incrementato a ogni passo, raggiungendo k dopo aver letto l'intera stringa
- Quindi, M' termina nello stato (q, k) dopo aver letto w
- Poiché $q \in F$, abbiamo $(q, k) \in F'$, quindi w è accettata da M'

Pertanto, M' accetta esattamente le stringhe di lunghezza k che sono accettate da M.

Problema di decisione 6. È dato un linguaggio L definito da un'espressione regolare e una stringa w. Descrivere un algoritmo che determini se esistono stringhe x e y tali che $xwy \in L$. L'algoritmo deve avere complessità polinomiale nella dimensione dell'espressione regolare e nella lunghezza di w.

Soluzione. Dato un linguaggio L definito da un'espressione regolare e una stringa w, vogliamo decidere se esistono stringhe x e y tali che $xwy \in L$.

Algoritmo:

1. Costruiamo un DFA $M = (Q, \Sigma, \delta, q_0, F)$ che riconosce L a partire dall'espressione regolare.
2. Definiamo l'insieme degli stati raggiungibili $R = \{q \in Q \mid \exists x \in \Sigma^* \text{ tale che } \delta^*(q_0, x) = q\}$.
3. Per ogni stato $q \in R$, calcoliamo $\delta^*(q, w)$, cioè lo stato raggiunto partendo da q e leggendo w.
4. Definiamo l'insieme degli stati da cui è possibile raggiungere uno stato finale $A = \{q \in Q \mid \exists y \in \Sigma^* \text{ tale che } \delta^*(q, y) \in F\}$.
5. Verifichiamo se esiste almeno un $q \in R$ tale che $\delta^*(q, w) \in A$. Se sì, allora esistono stringhe x e y tali che $xwy \in L$; altrimenti, tali stringhe non esistono.

Correttezza:

- Se esiste $q \in R$ tale che $\delta^*(q, w) \in A$, allora esiste una stringa x tale che $\delta^*(q_0, x) = q$, e una stringa y tale che $\delta^*(\delta^*(q, w), y) \in F$. Quindi, $\delta^*(q_0, xwy) \in F$, cioè $xwy \in L$.
- Se non esiste tale q, allora per ogni stringa x, lo stato $\delta^*(q_0, x)$ non soddisfa la condizione che $\delta^*(\delta^*(q_0, x), w) \in A$. Quindi, non esistono stringhe x e y tali che $xwy \in L$.

Complessità:

- La costruzione del DFA a partire dall'espressione regolare richiede tempo polinomiale nella dimensione dell'espressione regolare.
- Il calcolo degli insiemi R e A può essere effettuato con algoritmi di accessibilità standard su grafi, che hanno complessità polinomiale nel numero di stati di M.
- Per ogni stato $q \in R$, il calcolo di $\delta^*(q, w)$ richiede tempo $O(|w|)$.
- Ci sono al più $|Q|$ stati in R, quindi questa fase richiede tempo $O(|Q| \cdot |w|)$.
- La verifica finale richiede tempo $O(|Q|)$.

Complessivamente, l'algoritmo ha complessità polinomiale nella dimensione dell'espressione regolare e nella lunghezza di w.

Alternativa: possiamo anche utilizzare il concetto di "quoziente" di linguaggi. Definiamo:

- $L/w = \{y \in \Sigma^* \mid wy \in L\}$ (quoziente destro)
- $w \backslash L = \{x \in \Sigma^* \mid xw \in L\}$ (quoziente sinistro)

Il problema si riduce quindi a verificare se $(w \backslash L) \cap (\Sigma^*) \neq \emptyset$ e $(L/w) \cap (\Sigma^*) \neq \emptyset$, ossia se entrambi i quozienti non sono vuoti.

Per calcolare i quozienti, possiamo utilizzare le proprietà degli automi a stati finiti:

1. Per L/w, costruiamo un DFA $M_w = (Q, \Sigma, \delta, q_w, F)$, dove $q_w = \delta^*(q_0, w)$ è lo stato raggiunto dopo aver letto w a partire da q_0. Il linguaggio riconosciuto da M_w è esattamente L/w.
2. Per $w \backslash L$, costruiamo un DFA $M' = (Q, \Sigma, \delta, q_0, F')$, dove $F' = \{q \in Q \mid \delta^*(q, w) \in F\}$. Il linguaggio riconosciuto da M' è esattamente $w \backslash L$.

Verifichiamo se $L(M_w) \neq \emptyset$ e $L(M') \neq \emptyset$. Questo è equivalente a verificare se esiste un percorso dallo stato iniziale a uno stato finale in ciascun automa, il che può essere fatto con un algoritmo di accessibilità standard.

La complessità totale rimane polinomiale nella dimensione dell'espressione regolare e nella lunghezza di w.

2 Pumping Lemma e Non-Regolarità

Divisori binari (da appello 16/4/2024) 7. Considera il linguaggio

$$L = \{x\#y \mid x, y \in \{0,1\}^* \text{ sono numeri binari tali che } x \text{ è un divisore di } y\}.$$

L'alfabeto di questo linguaggio è $\{0, 1, \#\}$. Ad esempio, $10\#100 \in L$ perché 2 è un divisore di 4, mentre $10\#0101 \notin L$ perché 2 non è divisore di 5. Dimostra che L non è regolare.

Suggerimento: Utilizzare il pumping lemma considerando stringhe della forma $1^n\#1^{n^2}$.

Soluzione. Utilizziamo il pumping lemma per dimostrare che il linguaggio L non è regolare.

Supponiamo per assurdo che L sia regolare. Allora esiste una costante $p > 0$ (lunghezza del pumping) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere suddivisa in tre parti $s = xyz$ tali che:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Consideriamo la stringa $s = 1^p\#1^{p^2} \in L$, dove interpretiamo 1^p come il numero binario $2^p - 1$ e 1^{p^2} come il numero binario $2^{p^2} - 1$. Poiché $2^p - 1$ è un divisore di $2^{p^2} - 1$, abbiamo $s \in L$.

Per il pumping lemma, possiamo suddividere $s = xyz$ con le proprietà sopra descritte. Poiché $|xy| \leq p$, sappiamo che y è composto solo da '1' e si trova interamente nella prima parte della stringa (prima del simbolo '#').

Sia $y = 1^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $xy^2z = 1^p \cdot 1^k\#1^{p^2} = 1^{p+k}\#1^{p^2}$.

In questa stringa, la prima parte rappresenta il numero binario $2^{p+k} - 1$, mentre la seconda parte rimane $2^{p^2} - 1$.

Dimostriamo che $2^{p+k} - 1$ non è un divisore di $2^{p^2} - 1$ per $k > 0$, il che contraddirà il pumping lemma.

Se $2^{p+k} - 1$ fosse un divisore di $2^{p^2} - 1$, allora esisterebbe un intero m tale che $(2^{p+k} - 1) \cdot m = 2^{p^2} - 1$.

Consideriamo il numero $2^{p+k} - 1$. Per $k > 0$, abbiamo $2^{p+k} - 1 > 2^p - 1$. Questo significa che la nuova prima parte è un numero più grande rispetto alla prima parte originale.

Sappiamo che $2^{p^2} - 1$ è divisibile per $2^p - 1$ perché p divide p^2. Infatti, abbiamo la formula: $(2^p - 1)$ divide $(2^m - 1)$ se e solo se p divide m.

Poiché $p + k > p$ e $k > 0$, $p + k$ non divide p^2 a meno che $p + k = p$, il che è impossibile per $k > 0$.

Quindi, $2^{p+k} - 1$ non è un divisore di $2^{p^2} - 1$, il che significa che $xy^2z \notin L$, contraddicendo il pumping lemma.

Pertanto, L non può essere regolare.

Permutazioni (da appello 15/7/2024) 8. Date due stringhe u e v, diciamo che u è una permutazione di v se u ha gli stessi simboli di v con ugual numero di occorrenze, ma eventualmente in un ordine diverso. Per esempio, le stringhe 01011 e 00111 sono entrambe permutazioni di 11001.

Dimostra che il seguente linguaggio non è regolare:

$$L = \{uv \mid u, v \in \{0,1\}^* \text{ e } u \text{ è una permutazione di } v\}.$$

Suggerimento: Considerare stringhe della forma 0^n1^nw dove w è una permutazione di 0^n1^n.

Soluzione. Utilizziamo il pumping lemma per dimostrare che il linguaggio L non è regolare.

Supponiamo per assurdo che L sia regolare. Allora esiste una costante $p > 0$ (lunghezza del pumping) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere suddivisa in tre parti $s = xyz$ tali che:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Consideriamo la stringa $s = 0^p1^p0^p1^p \in L$. Possiamo vedere questa stringa come uv dove $u = 0^p1^p$ e $v = 0^p1^p$. Chiaramente, u è una permutazione di v (in questo caso, sono identici), quindi $s \in L$.

Per il pumping lemma, possiamo suddividere $s = xyz$ con le proprietà sopra descritte. Poiché $|xy| \leq p$, sappiamo che y si trova interamente nella prima parte della stringa, cioè nei primi p caratteri, quindi $y = 0^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $xy^0z = xz$ (rimuoviamo completamente y).

Sia $x = 0^j$ con $j + k = p$ (poiché $xy = 0^p$). Quindi: $xz = 0^j1^p0^p1^p$

Questa stringa deve essere della forma $u'v'$ dove u' è una permutazione di v'. Dobbiamo determinare come suddividere la stringa in u' e v'.

Dato che $|u'| = |v'|$ (perché una permutazione deve avere la stessa lunghezza), e la lunghezza totale di xz è $2p + 2p - k = 4p - k$, dobbiamo avere $|u'| = |v'| = (4p - k)/2 = 2p - k/2$.

Se k è dispari, già abbiamo un problema perché $k/2$ non è un intero, quindi la stringa non può essere suddivisa correttamente.

Se k è pari, abbiamo $|u'| = |v'| = 2p - k/2$. Ma anche in questo caso, nessuna suddivisione della stringa $0^j1^p0^p1^p$ in due parti di uguale lunghezza può dare due parti che sono permutazioni l'una dell'altra, a meno che non dividiamo esattamente a metà, cioè $u' = 0^j1^{p-j}$ e $v' = 0^{p-j}1^p$.

Ma u' ha j zeri e $p - j$ uni, mentre v' ha $p - j$ zeri e p uni. Per essere permutazioni, dovrebbero avere lo stesso numero di zeri e uni, il che è possibile solo se $j = p - j$ e $p - j = p$, cioè $j = p/2$ e $p - j = p$, il che è una contraddizione.

Quindi, $xz \notin L$, il che contraddice il pumping lemma.

Pertanto, L non può essere regolare.

Concatenazione potenze 9. Dimostra che il linguaggio $L = \{a^nb^m \mid n, m \geq 1, n \text{ è divisibile per } m\}$ non è regolare.

Soluzione. Utilizziamo il pumping lemma per dimostrare che il linguaggio L non è regolare.

Supponiamo per assurdo che L sia regolare. Allora esiste una costante $p > 0$ (lunghezza del pumping) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere suddivisa in tre parti $s = xyz$ tali che:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Consideriamo la stringa $s = a^{p!}b^1 \in L$. Poiché $n = p!$ è divisibile per $m = 1$, abbiamo $s \in L$.

Per il pumping lemma, possiamo suddividere $s = xyz$ con le proprietà sopra descritte. Poiché $|xy| \leq p$, sappiamo che y si trova interamente nella prima parte della stringa, cioè $y = a^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $xy^2z = a^{p!+k}b^1 \in L$. Per appartenere a L, deve essere vero che $p! + k$ è divisibile per 1, il che è banalmente vero.

Ora consideriamo la stringa $xy^0z = xz = a^{p!-k}b^1$. Per appartenere a L, deve essere vero che $p! - k$ è divisibile per 1, il che è banalmente vero.

Proviamo invece con una stringa più complessa. Consideriamo $s' = a^{p!}b^p \in L$. Poiché $n = p!$ è divisibile per $m = p$ (infatti, $p! = p \cdot (p-1)!$), abbiamo $s' \in L$.

Per il pumping lemma, possiamo suddividere $s' = x'y'z'$ con le proprietà sopra descritte. Poiché $|x'y'| \leq p$, sappiamo che y' si trova interamente nella prima parte della stringa, cioè $y' = a^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $x'y'^2z' = a^{p!+k}b^p$. Per appartenere a L, deve essere vero che $p! + k$ è divisibile per p.

Sappiamo che $p!$ è divisibile per p. Quindi, $p! + k$ è divisibile per p se e solo se k è divisibile per p. Ma poiché $1 \leq k \leq p$, l'unico valore per cui k è divisibile per p è $k = p$.

Se $k < p$, allora $p! + k$ non è divisibile per p, il che significa che $x'y'^2z' \notin L$, contraddicendo il pumping lemma.

Quindi, dobbiamo avere $k = p$. Ma in questo caso, $y' = a^p$, il che significa che $|x'y'| = |x'| + |y'| \geq p$. Poiché abbiamo anche $|x'y'| \leq p$, dobbiamo avere $|x'y'| = p$, il che implica $|x'| = 0$, cioè $x' = \varepsilon$.

Ora, consideriamo la stringa $x'y'^0z' = z' = a^{p!-p}b^p$. Per appartenere a L, deve essere vero che $p! - p$ è divisibile per p.

Sappiamo che $p! = p \cdot (p-1)!$, quindi $p! - p = p \cdot (p-1)! - p = p \cdot ((p-1)! - 1)$. Questo è divisibile per p se e solo se $(p-1)! - 1$ è un intero, il che è vero.

Quindi, $x'y'^0z' \in L$, il che è coerente con il pumping lemma. Ma questo è un caso speciale in cui $k = p$ e $x' = \varepsilon$.

Consideriamo una terza stringa $s'' = a^{p!(p+1)}b^{p+1} \in L$. Poiché $n = p!(p+1)$ è divisibile per $m = p + 1$, abbiamo $s'' \in L$.

Per il pumping lemma, possiamo suddividere $s'' = x''y''z''$ con le proprietà sopra descritte. Poiché $|x''y''| \leq p$, sappiamo che y'' si trova interamente nella prima parte della stringa, cioè $y'' = a^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $x''y''^2z'' = a^{p!(p+1)+k}b^{p+1}$. Per appartenere a L, deve essere vero che $p!(p+1) + k$ è divisibile per $p + 1$.

Sappiamo che $p!(p+1)$ è divisibile per $p+1$. Quindi, $p!(p+1)+k$ è divisibile per $p+1$ se e solo se k è divisibile per $p+1$. Ma poiché $1 \leq k \leq p < p+1$, k non può essere divisibile per $p+1$.

Quindi, $x''y''^2z'' \notin L$, contraddicendo il pumping lemma.

Pertanto, L non può essere regolare.

Stringhe palindrome 10. Dimostra che il linguaggio delle stringhe palindrome su $\{0,1\}$ non è regolare. Una stringa w è palindroma se $w = w^R$.

Soluzione. Sia $L = \{w \in \{0,1\}^* \mid w = w^R\}$ il linguaggio delle stringhe palindrome su $\{0,1\}$. Utilizziamo il pumping lemma per dimostrare che L non è regolare.

Supponiamo per assurdo che L sia regolare. Allora esiste una costante $p > 0$ (lunghezza del pumping) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere suddivisa in tre parti $s = xyz$ tali che:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Consideriamo la stringa $s = 0^p10^p \in L$. Questa è chiaramente una stringa palindroma (poiché è simmetrica rispetto al carattere centrale '1'), quindi $s \in L$.

Per il pumping lemma, possiamo suddividere $s = xyz$ con le proprietà sopra descritte. Poiché $|xy| \leq p$, sappiamo che y si trova interamente nella prima parte della stringa, cioè $y = 0^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $xy^2z = 0^{p+k}10^p$. Per essere palindroma, questa stringa dovrebbe essere uguale al suo reverse, cioè 0^p10^{p+k}. Ma questo è vero solo se $p = p+k$, cioè se $k = 0$, il che contraddice la condizione $|y| > 0$ del pumping lemma.

Quindi, $xy^2z \notin L$, contraddicendo il pumping lemma.

Pertanto, L non può essere regolare.

Potenze di 2 11. Dimostra che il linguaggio $L = \{a^n \mid n = 2^k \text{ per qualche } k \geq 0\}$ non è regolare.

Soluzione. Sia $L = \{a^n \mid n = 2^k \text{ per qualche } k \geq 0\}$. Utilizziamo il pumping lemma per dimostrare che L non è regolare.

Supponiamo per assurdo che L sia regolare. Allora esiste una costante $p > 0$ (lunghezza del pumping) tale che ogni stringa $s \in L$ con $|s| \geq p$ può essere suddivisa in tre parti $s = xyz$ tali che:

1. $|xy| \leq p$
2. $|y| > 0$
3. Per ogni $i \geq 0$, $xy^iz \in L$

Sia m il più piccolo intero tale che $2^m \geq p$. Consideriamo la stringa $s = a^{2^m} \in L$.

Per il pumping lemma, possiamo suddividere $s = xyz$ con le proprietà sopra descritte. Poiché $|xy| \leq p$, sappiamo che $|y| \leq p$. Inoltre, $y = a^k$ per qualche $1 \leq k \leq p$.

Consideriamo la stringa $xy^2z = a^{2^m+k}$. Per appartenere a L, deve essere vero che $2^m + k = 2^j$ per qualche $j \geq 0$.

Ma $2^m < 2^m + k < 2^m + p \leq 2^m + 2^m = 2^{m+1}$ (poiché $1 \leq k \leq p \leq 2^m$).

Quindi, $2^m < 2^m + k < 2^{m+1}$, il che significa che $2^m + k$ non è una potenza di 2, quindi $xy^2z \notin L$, contraddicendo il pumping lemma.

Pertanto, L non può essere regolare.

3 Linguaggi Context-Free

Operazione MIX (da appello 16/4/2024) 12. Dati due linguaggi A, B, definiamo il linguaggio $\textsc{Mix}(A, B)$ come

$$\textsc{Mix}(A, B) = \{x_1y_1x_2y_2 \dots x_ny_n \mid n \geq 0, x_i \in A, y_i \in B\}.$$

Si noti che ciascun x_i, y_i è una stringa. Dimostra che la classe dei linguaggi context-free è chiusa per l'operazione $\textsc{Mix}$.

Suggerimento: Modificare la grammatica per A e B in modo da intrecciarle.

Soluzione. Siano A e B linguaggi context-free. Vogliamo dimostrare che $\textsc{Mix}(A, B)$ è anch'esso context-free.

Dato che A e B sono context-free, esistono grammatiche context-free $G_A = (V_A, \Sigma, R_A, S_A)$ e $G_B = (V_B, \Sigma, R_B, S_B)$ tali che $L(G_A) = A$ e $L(G_B) = B$.

Senza perdita di generalità, possiamo assumere che $V_A \cap V_B = \emptyset$ (se necessario, rinominiamo i non-terminali per garantire questa condizione).

Costruiamo una nuova grammatica $G = (V, \Sigma, R, S)$ per $\textsc{Mix}(A, B)$ come segue:

- $V = V_A \cup V_B \cup \{S\}$, dove S è un nuovo simbolo non-terminale
- R consiste delle seguenti regole:
  - $S \to \epsilon$ (per gestire il caso $n = 0$)
  - $S \to S_AS_B$ (per iniziare con una coppia x_1y_1)
  - $S \to S_AS_BS$ (per concatenare più coppie x_iy_i)
  - Tutte le regole in R_A (per generare stringhe in A)
  - Tutte le regole in R_B (per generare stringhe in B)

Dimostriamo che $L(G) = \textsc{Mix}(A, B)$.

($L(G) \subseteq \textsc{Mix}(A, B)$): Sia $w \in L(G)$. Allora w può essere derivata da S usando le regole di G. Se la derivazione parte con $S \to \epsilon$, allora $w = \epsilon$, che è in $\textsc{Mix}(A, B)$ per $n = 0$. Altrimenti, la derivazione coinvolge una sequenza di applicazioni di $S \to S_AS_BS$ (possibilmente 0 volte) seguite da $S \to S_AS_B$ e infine $S \to \epsilon$. Questo porta a una derivazione della forma $S \Rightarrow^* S_AS_BS_AS_B \dots S_AS_B$.

Ogni S_A può derivare una stringa $x_i \in A$ (usando le regole di R_A), e ogni S_B può derivare una stringa $y_i \in B$ (usando le regole di R_B). Quindi, $w = x_1y_1x_2y_2 \dots x_ny_n$ con $x_i \in A$ e $y_i \in B$ per $1 \leq i \leq n$, il che significa che $w \in \textsc{Mix}(A, B)$.

($\textsc{Mix}(A, B) \subseteq L(G)$): Sia $w \in \textsc{Mix}(A, B)$. Allora $w = x_1y_1x_2y_2 \dots x_ny_n$ per qualche $n \geq 0$, con $x_i \in A$ e $y_i \in B$ per $1 \leq i \leq n$.

Se $n = 0$, allora $w = \epsilon$ e possiamo derivare w in G con $S \Rightarrow \epsilon$.

Se $n > 0$, possiamo derivare w in G come segue:

- Se $n = 1$: $S \Rightarrow S_A S_B \Rightarrow^* x_1 S_B \Rightarrow^* x_1 y_1$
- Se $n > 1$:

$$
\begin{aligned}
S &\Rightarrow S_A S_B S \\
&\Rightarrow^* x_1 S_B S \\
&\Rightarrow^* x_1 y_1 S \\
&\Rightarrow \dots \\
&\Rightarrow^* x_1 y_1 x_2 y_2 \dots x_{n-1} y_{n-1} S \\
&\Rightarrow x_1 y_1 x_2 y_2 \dots x_{n-1} y_{n-1} S_A S_B \\
&\Rightarrow^* x_1 y_1 x_2 y_2 \dots x_{n-1} y_{n-1} x_n S_B \\
&\Rightarrow^* x_1 y_1 x_2 y_2 \dots x_{n-1} y_{n-1} x_n y_n
\end{aligned}
$$

Quindi, $w \in L(G)$.

Soluzione. Poiché abbiamo dimostrato che $L(G) = \textsc{Mix}(A, B)$ e G è una grammatica context-free, concludiamo che $\textsc{Mix}(A, B)$ è un linguaggio context-free. Quindi, la classe dei linguaggi context-free è chiusa rispetto all'operazione $\textsc{Mix}$.

Palindromizzazione (da appello 15/7/2024) 13. Dimostra che se B è un linguaggio regolare, allora il linguaggio

$$\textsc{Palindromize}(B) = \{ww^R \mid w \in B\}$$

è un linguaggio context-free.

Suggerimento: Costruire una grammatica o un PDA.

Soluzione. Sia B un linguaggio regolare. Vogliamo dimostrare che $\textsc{Palindromize}(B) = \{ww^R \mid w \in B\}$ è un linguaggio context-free.

Poiché B è regolare, esiste un DFA $M = (Q, \Sigma, \delta, q_0, F)$ tale che $L(M) = B$.

Costruiamo un PDA $P = (Q', \Sigma, \Gamma, \delta', q_0', \bot, F')$ che accetta $\textsc{Palindromize}(B)$ per accettazione di stato finale:

- $Q' = Q \cup \{q_0', q_f\}$, dove q_0' è un nuovo stato iniziale e q_f è un nuovo stato finale
- $\Gamma = \Sigma \cup \{Z\}$, dove Z è un simbolo speciale per il fondo dello stack
- $F' = \{q_f\}$
- La funzione di transizione δ' include le seguenti transizioni:
  - $\delta'(q_0', \varepsilon, Z) = \{(q_0, Z)\}$ (inizializza lo stack con Z e passa allo stato iniziale di M)
  - Per ogni $q \in Q$, $a \in \Sigma$, $\delta'(q, a, \alpha) = \{(\delta(q, a), a\alpha)\}$ (simula M e impila il carattere letto)
  - Per ogni $q \in F$, $\delta'(q, \varepsilon, \alpha) = \{(q_f, \alpha)\}$ (passa allo stato finale se si è in uno stato di accettazione di M)
  - Per ogni $a \in \Sigma$, $\delta'(q_f, a, a) = \{(q_f, \varepsilon)\}$ (consuma un carattere se corrisponde al top dello stack)

Il PDA P funziona come segue:

1. Inizialmente, lo stack contiene solo Z.
2. Durante la lettura della prima parte dell'input, P simula M e impila ogni carattere letto.
3. Quando decide di passare alla seconda parte (corrispondente a w^R), deve essere in uno stato finale di M (per garantire che la prima parte $w \in B$) e passa allo stato q_f.
4. Nello stato q_f, legge i caratteri rimanenti dell'input e li confronta con i caratteri in cima allo stack, consumandoli se corrispondono. Questo garantisce che la seconda parte sia w^R.
5. Se riesce a consumare tutti i caratteri rimanenti dell'input e lo stack contiene solo Z, allora accetta.

Formalmente, dimostriamo che $L(P) = \textsc{Palindromize}(B)$:

($L(P) \subseteq \textsc{Palindromize}(B)$): Sia $s \in L(P)$. Allora P accetta s, il che significa che P può leggere s e terminare nello stato q_f con lo stack vuoto (a parte Z).

Per accettare s, P deve:

1. Leggere una prima parte w, simulando M e impilando i caratteri di w.
2. Raggiungere uno stato finale di M, il che significa che $w \in B$.
3. Passare allo stato q_f.
4. Leggere la parte rimanente x dell'input, confrontandola con i caratteri in cima allo stack.

Per accettare, lo stack deve essere vuoto alla fine, il che significa che x deve essere esattamente w^R. Quindi, $s = wx = ww^R$ con $w \in B$, cioè $s \in \textsc{Palindromize}(B)$.

($\textsc{Palindromize}(B) \subseteq L(P)$): Sia $s \in \textsc{Palindromize}(B)$. Allora $s = ww^R$ per qualche $w \in B$.

Poiché $w \in B$, M accetta w, quindi esiste una sequenza di stati $q_0, q_1, \ldots, q_n$ tale che $q_n \in F$ e $\delta(q_i, w_{i+1}) = q_{i+1}$ per $0 \leq i < n$.

Allora P può elaborare $s = ww^R$ come segue:

1. Inizia nello stato q_0' con lo stack contenente solo Z.
2. Passa allo stato q_0 (lo stato iniziale di M), mantenendo Z nello stack.
3. Legge w carattere per carattere, simulando M e impilando ogni carattere. Dopo aver letto w, P è nello stato $q_n \in F$ e lo stack contiene $w^R Z$ (con w^R in cima).
4. Passa allo stato q_f.
5. Legge w^R carattere per carattere, confrontando ogni carattere con il carattere in cima allo stack. Poiché w^R è esattamente il reverse di w, ogni carattere corrisponde al carattere in cima allo stack.

6. Dopo aver letto w^R, lo stack contiene solo Z e P è nello stato q_f, quindi accetta.

Quindi, $s \in L(P)$.

Poiché $L(P) = \textsc{Palindromize}(B)$ e P è un PDA, concludiamo che $\textsc{Palindromize}(B)$ è un linguaggio context-free.

Concatenazione con potenze 14. Siano A e B linguaggi su Σ. Definiamo $A \cdot B^n = \{xy \mid x \in A, y \in B^n\}$, dove B^n è il linguaggio ottenuto concatenando B con se stesso n volte.

Dimostra che se A e B sono linguaggi context-free, allora per ogni $n \geq 0$ fissato, anche $A \cdot B^n$ è context-free.

Soluzione. Siano A e B linguaggi context-free su Σ. Vogliamo dimostrare che per ogni $n \geq 0$ fissato, anche $A \cdot B^n$ è context-free.

Poiché A e B sono context-free, esistono grammatiche context-free $G_A = (V_A, \Sigma, R_A, S_A)$ e $G_B = (V_B, \Sigma, R_B, S_B)$ tali che $L(G_A) = A$ e $L(G_B) = B$.

Senza perdita di generalità, possiamo assumere che $V_A \cap V_B = \emptyset$ (se necessario, rinominiamo i non-terminali per garantire questa condizione).

Procediamo per induzione su n:

Base ($n = 0$): Per convenzione, $B^0 = \{\epsilon\}$ (il linguaggio contenente solo la stringa vuota). Quindi, $A \cdot B^0 = A \cdot \{\epsilon\} = A$. Poiché A è context-free per ipotesi, anche $A \cdot B^0$ è context-free.

Passo induttivo: Supponiamo che $A \cdot B^k$ sia context-free per qualche $k \geq 0$. Vogliamo dimostrare che $A \cdot B^{k+1}$ è anch'esso context-free.

Osserviamo che $A \cdot B^{k+1} = A \cdot B^k \cdot B = (A \cdot B^k) \cdot B$.

Per l'ipotesi induttiva, $A \cdot B^k$ è context-free. E poiché B è context-free per ipotesi, e la classe dei linguaggi context-free è chiusa rispetto alla concatenazione, concludiamo che $(A \cdot B^k) \cdot B = A \cdot B^{k+1}$ è context-free.

Quindi, per induzione, $A \cdot B^n$ è context-free per ogni $n \geq 0$ fissato.

Context-free o no? 15. Determina quali dei seguenti linguaggi sono context-free, giustificando la risposta:

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

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

c) $L_3 = \{a^n b^m \mid n \neq m, \text{ dove } n, m \geq 1\}$

d) $L_4 = \{a^i b^j c^k \mid i + j = k, \text{ dove } i, j, k \geq 1\}$

e) $L_5 = \{a^i b^j c^k d^l \mid i = j \text{ e } k = l, \text{ dove } i, j, k, l \geq 1\}$

Suggerimento: Utilizzare il pumping lemma per CFG o costruire una CFG/PDA dove possibile.

Soluzione. Analizziamo ciascun linguaggio per determinare se è context-free:

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

L_1 è context-free. Possiamo esprimere L_1 come l'unione di due linguaggi context-free:

$$L_{1a} = \{a^n b^m c^k \mid n = m, n, m, k \geq 1\}$$
$$L_{1b} = \{a^n b^m c^k \mid m = k, n, m, k \geq 1\}$$

Entrambi L_{1a} e L_{1b} sono context-free:

- Per L_{1a}, possiamo costruire una grammatica con le produzioni:

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

- Per L_{1b}, possiamo costruire una grammatica con le produzioni:

$$\begin{aligned} S &\to AC \\ A &\to aA \mid a \\ C &\to bCc \mid bc \end{aligned}$$

Poiché la classe dei linguaggi context-free è chiusa rispetto all'unione, $L_1 = L_{1a} \cup L_{1b}$ è context-free.

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

L_2 è context-free. Possiamo costruire una grammatica context-free che genera L_2:

$$\begin{aligned} S &\to AB \\ A &\to aAb \mid ab \\ B &\to cBd \mid cd \end{aligned}$$

La grammatica genera stringhe della forma $a^n b^n c^m d^m$ dove $n, m \geq 1$, quindi $L(G) = L_2$.

c) $L_3 = \{a^n b^m \mid n \neq m, \text{ dove } n, m \geq 1\}$

L_3 non è context-free. Utilizziamo il pumping lemma per linguaggi context-free per dimostrarlo.

Supponiamo per assurdo che L_3 sia context-free. Allora, per il pumping lemma, esiste una costante $p > 0$ tale che ogni stringa $s \in L_3$ con $|s| \geq p$ può essere suddivisa in cinque parti $s = uvwxy$ tali che:

1. $|vwx| \leq p$
2. $|vx| > 0$
3. Per ogni $i \geq 0$, $uv^i wx^i y \in L_3$

Consideriamo la stringa $s = a^p b^{p+1} \in L_3$ (poiché $p \neq p + 1$). Per il pumping lemma, possiamo suddividere $s = uvwxy$ con le proprietà sopra descritte.

Poiché $|vwx| \leq p$, la sottostringa vwx è contenuta interamente nella parte a^p di s. Quindi, v e x contengono solo il carattere a.

Sia $v = a^r$ e $x = a^t$ con $r + t > 0$ (poiché $|vx| > 0$).

Consideriamo la stringa $s' = uv^0 wx^0 y = uwy$. Poiché v e x contengono solo a, eliminandoli riduciamo il numero di a nella stringa. Quindi, $s' = a^{p-(r+t)} b^{p+1}$.

Per il pumping lemma, $s' \in L_3$, il che significa che $p - (r + t) \neq p + 1$. Ma possiamo sempre trovare un valore di i tale che $p - (r + t) + i(r + t) = p + 1$, cioè $i = \frac{2p+1}{r+t} > 0$ (poiché $r + t > 0$).

Per questo valore di i, la stringa $s'' = uv^iwx^iy$ avrebbe $p - (r+t) + i(r+t) = p+1$ occorrenze di a e $p+1$ occorrenze di b, quindi $s'' \notin L_3$ (perché il numero di a e b è uguale), contraddicendo il pumping lemma.

Pertanto, L_3 non può essere context-free.

d) $L_4 = \{a^ib^jc^k \mid i+j=k,\ \text{dove } i,j,k \geq 1\}$

L_4 è context-free. Possiamo costruire un PDA che accetta L_4:

1. Legge e impila un carattere a per ogni a nell'input.
2. Legge e impila un carattere b per ogni b nell'input.
3. Per ogni c nell'input, toglie un carattere dallo stack (o un a o un b).
4. Accetta se e solo se ha letto tutti i caratteri dell'input e lo stack è vuoto.

Alternativamente, possiamo costruire una grammatica context-free per L_4:

$$S \to aSc \mid T$$
$$T \to bTc \mid bc$$

Questa grammatica genera stringhe della forma $a^ib^jc^k$ dove $i+j=k$ e $i,j,k \geq 1$, quindi $L(G) = L_4$.

e) $L_5 = \{a^ib^jc^kd^l \mid i=j \text{ e } k=l,\ \text{dove } i,j,k,l \geq 1\}$

L_5 è context-free. Possiamo esprimere L_5 come la concatenazione di due linguaggi context-free:

$$L_{5a} = \{a^ib^j \mid i=j, i,j \geq 1\}$$
$$L_{5b} = \{c^kd^l \mid k=l, k,l \geq 1\}$$

Entrambi L_{5a} e L_{5b} sono context-free:

- Per L_{5a}, possiamo costruire una grammatica con le produzioni:

$$S_1 \to aS_1b \mid ab$$

- Per L_{5b}, possiamo costruire una grammatica con le produzioni:

$$S_2 \to cS_2d \mid cd$$

Poiché la classe dei linguaggi context-free è chiusa rispetto alla concatenazione, $L_5 = L_{5a} \cdot L_{5b}$ è context-free. La grammatica per L_5 è:

$$S \to S_1S_2$$
$$S_1 \to aS_1b \mid ab$$
$$S_2 \to cS_2d \mid cd$$

Forma Normale di Chomsky 16. Converti la seguente grammatica in Forma Normale di Chomsky, mostrando tutti i passaggi della trasformazione:

$$G: \begin{aligned} S &\to AB \mid aSb \\ A &\to aA \mid \varepsilon \\ B &\to bB \mid \varepsilon \end{aligned}$$

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. Convertiamo la grammatica G in Forma Normale di Chomsky (FNC).

Passo 1: Aggiungere un nuovo simbolo iniziale.

$$\begin{aligned} S_0 &\to S \\ S &\to AB \mid aSb \\ A &\to aA \mid \varepsilon \\ B &\to bB \mid \varepsilon \end{aligned}$$

Passo 2: Eliminare le produzioni ε.

Le produzioni ε sono $A \to \varepsilon$ e $B \to \varepsilon$. Per eliminarle, dobbiamo considerare tutte le produzioni che contengono A e B e creare nuove produzioni sostituendo ogni possibile combinazione di A e B con ε.

Per $S \to AB$, sostituendo A con ε otteniamo $S \to B$, e sostituendo B con ε otteniamo $S \to A$. Sostituendo sia A che B con ε, otteniamo $S \to \varepsilon$.

Per $S \to aSb$, non ci sono A o B da sostituire.

Quindi, le nuove produzioni sono:

$$\begin{aligned} S_0 &\to S \\ S &\to AB \mid A \mid B \mid \varepsilon \mid aSb \\ A &\to aA \mid a \\ B &\to bB \mid b \end{aligned}$$

Osserviamo che ora S può derivare ε, quindi dobbiamo considerare anche le produzioni che contengono S e creare nuove produzioni sostituendo S con ε. L'unica produzione che contiene S è $S_0 \to S$, che ci dà la nuova produzione $S_0 \to \varepsilon$.

Inoltre, per la produzione $S \to aSb$, sostituendo S con ε otteniamo $S \to ab$.

Quindi, le nuove produzioni sono:

$$\begin{aligned} S_0 &\to S \mid \varepsilon \\ S &\to AB \mid A \mid B \mid \varepsilon \mid aSb \mid ab \\ A &\to aA \mid a \\ B &\to bB \mid b \end{aligned}$$

Passo 3: Eliminare le produzioni unitarie.

Le produzioni unitarie sono $S_0 \to S$, $S \to A$, e $S \to B$.

Per $S_0 \to S$, sostituiamo S con tutte le produzioni di S eccetto quelle unitarie: $S_0 \to AB \mid aSb \mid ab \mid \varepsilon$

Per $S \to A$, sostituiamo A con tutte le produzioni di A: $S \to aA \mid a$

Per $S \to B$, sostituiamo B con tutte le produzioni di B: $S \to bB \mid b$

Quindi, le nuove produzioni sono:

$$\begin{aligned}
S_0 &\to AB \mid aSb \mid ab \mid \varepsilon \\
S &\to AB \mid aSb \mid ab \mid \varepsilon \mid aA \mid a \mid bB \mid b \\
A &\to aA \mid a \\
B &\to bB \mid b
\end{aligned}$$

Passo 4: Convertire le produzioni nella forma corretta.

Le produzioni devono essere della forma $X \to YZ$ o $X \to a$, dove X, Y, Z sono non-terminali e a è un terminale.

Per la produzione $S \to ab$, introduciamo nuovi non-terminali A_a e A_b tali che $A_a \to a$ e $A_b \to b$. Quindi, $S \to ab$ diventa $S \to A_aA_b$.

Per la produzione $S \to aSb$, introduciamo un nuovo non-terminale S_b tale che $S_b \to Sb$. Quindi, $S \to aSb$ diventa $S \to aS_b$. Ma questa non è ancora nella forma corretta, quindi introduciamo un altro non-terminale A_a tale che $A_a \to a$. Quindi, $S \to aS_b$ diventa $S \to A_aS_b$.

Per la produzione $A \to aA$, introduciamo un nuovo non-terminale A_a tale che $A_a \to a$. Quindi, $A \to aA$ diventa $A \to A_aA$.

Similmente, per $B \to bB$, introduciamo un nuovo non-terminale A_b tale che $A_b \to b$. Quindi, $B \to bB$ diventa $B \to A_bB$.

Per $A \to a$, possiamo usare $A \to A_a$ e $A_a \to a$.

Per $B \to b$, possiamo usare $B \to A_b$ e $A_b \to b$.

Inoltre, dobbiamo occuparci delle produzioni $S_0 \to \varepsilon$ e $S \to \varepsilon$. In FNC, l'unica produzione che può generare ε è $S_0 \to \varepsilon$, e solo se $\varepsilon \in L(G)$. Quindi, manteniamo $S_0 \to \varepsilon$ e rimuoviamo $S \to \varepsilon$.

Per la produzione $S_b \to Sb$, introduciamo un nuovo non-terminale A_b tale che $A_b \to b$. Quindi, $S_b \to Sb$ diventa $S_b \to SA_b$.

Quindi, le nuove produzioni sono:

$$\begin{aligned}
S_0 &\to AB \mid A_aS_b \mid A_aA_b \mid \varepsilon \\
S &\to AB \mid A_aS_b \mid A_aA_b \mid A_aA \mid A_a \mid A_bB \mid A_b \\
A &\to A_aA \mid A_a \\
B &\to A_bB \mid A_b \\
S_b &\to SA_b \\
A_a &\to a \\
A_b &\to b
\end{aligned}$$

Eliminiamo anche le produzioni unitarie rimanenti: Per $S \to A_a$, sostituiamo A_a con le sue produzioni: $S \to a$, che diventa $S \to A_a$. Per $S \to A_b$, sostituiamo A_b con le sue produzioni: $S \to b$, che diventa $S \to A_b$. Per $A \to A_a$, sostituiamo A_a con le sue produzioni: $A \to a$, che diventa $A \to A_a$. Per $B \to A_b$, sostituiamo A_b con le sue produzioni: $B \to b$, che diventa $B \to A_b$.

Quindi, la grammatica finale in Forma Normale di Chomsky è:

$$\begin{aligned}
S_0 &\to AB \mid A_aS_b \mid A_aA_b \mid \varepsilon \\
S &\to AB \mid A_aS_b \mid A_aA_b \mid A_aA \mid A_bB \\
A &\to A_aA \mid A_a \\
B &\to A_bB \mid A_b \\
S_b &\to SA_b \\
A_a &\to a \\
A_b &\to b
\end{aligned}$$

Questa grammatica è in Forma Normale di Chomsky perché ogni produzione è della forma $X \to YZ$ o $X \to a$, eccetto per la produzione $S_0 \to \varepsilon$ che è consentita solo per il simbolo iniziale e solo se $\varepsilon \in L(G)$.

4 Problemi Avanzati

Chiusura rotazionale 17. Abbiamo definito la chiusura rotazionale di un linguaggio A come $RC(A) = \{yx \mid xy \in A\}$.

a) Dimostra che la classe dei linguaggi regolari è chiusa sotto chiusura rotazionale.

b) Dimostra che la classe dei linguaggi context-free è chiusa sotto chiusura rotazionale.

Soluzione. a) **Dimostriamo che la classe dei linguaggi regolari è chiusa sotto chiusura rotazionale.**

Sia A un linguaggio regolare. Vogliamo dimostrare che $RC(A) = \{yx \mid xy \in A\}$ è anch'esso regolare.

Poiché A è regolare, esiste un DFA $M = (Q, \Sigma, \delta, q_0, F)$ tale che $L(M) = A$.

Costruiamo un NFA $M' = (Q', \Sigma, \delta', q_0', F')$ che accetta $RC(A)$ come segue:

- $Q' = Q \times Q \cup \{q_0'\}$ (aggiungiamo coppie di stati e un nuovo stato iniziale)
- q_0' è il nuovo stato iniziale
- $F' = \{(q, q_0) \mid q \in F\}$ (accettiamo se il secondo componente è q_0 e il primo è uno stato finale)
- La funzione di transizione δ' include le seguenti transizioni:
  - $\delta'(q_0', \varepsilon) = \{(q_0, q) \mid q \in Q\}$ (dal nuovo stato iniziale, possiamo passare a qualsiasi coppia che inizia con q_0)
  - $\delta'((p, q), a) = \{(\delta(p, a), q)\}$ (aggiorniamo solo il primo componente della coppia)

L'idea è che M' "indovina" il punto di divisione della stringa originale $xy \in A$. Il primo componente della coppia (p, q) tiene traccia dello stato corrente di M mentre elabora l'intera stringa xy, mentre il secondo componente q rappresenta lo stato di M dopo aver elaborato la prima parte x.

L'NFA M' legge la seconda parte y della rotazione, aggiornando solo il primo componente della coppia. Accetta se, dopo aver letto y, il primo componente è uno stato finale (indicando che $xy \in A$) e il secondo componente è q_0 (indicando che siamo pronti a leggere la prima parte x).

Formalmente, dimostriamo che $L(M') = RC(A)$:

($L(M') \subseteq RC(A)$): Sia $w \in L(M')$. Allora esiste un'esecuzione accettante di M' su w che termina in uno stato $(q_f, q_0) \in F'$. Questa esecuzione deve partire da q'_0, passare a uno stato (q_0, q) per qualche $q \in Q$, e poi leggere w carattere per carattere.

Dopo aver letto w, M' è nello stato (q_f, q_0) con $q_f \in F$. Questo significa che esiste una stringa v tale che $\delta(q_0, v) = q$ e $\delta(q, w) = q_f$. Cioè, $\delta(q_0, vw) = q_f \in F$, quindi $vw \in A$.

Inoltre, poiché il secondo componente dello stato finale è q_0, abbiamo $q = q_0$. Quindi, w è esattamente la seconda parte di una rotazione, e $vw \in A$. Cioè, $w = yx$ e $vw = xy \in A$ per qualche x, y, quindi $w \in RC(A)$.

($RC(A) \subseteq L(M')$): Sia $w \in RC(A)$. Allora $w = yx$ per qualche stringhe x, y tali che $xy \in A$. Poiché $xy \in A$, abbiamo $\delta(q_0, xy) \in F$.

Ora, M' può elaborare $w = yx$ come segue:

1. Inizia nello stato q'_0.
2. Passa allo stato (q_0, q_0) (poiché $q_0 \in Q$).
3. Legge y carattere per carattere, raggiungendo lo stato $(\delta(q_0, y), q_0)$.
4. Legge x carattere per carattere, raggiungendo lo stato $(\delta(q_0, yx), q_0) = (\delta(q_0, xy), q_0)$ (poiché δ è una funzione e yx contiene gli stessi caratteri di xy, anche se in ordine diverso).
5. Poiché $xy \in A$, abbiamo $\delta(q_0, xy) \in F$, quindi $(\delta(q_0, xy), q_0) \in F'$.

Quindi, $w \in L(M')$.

Poiché abbiamo dimostrato che $L(M') = RC(A)$ e M' è un NFA, concludiamo che $RC(A)$ è regolare. Quindi, la classe dei linguaggi regolari è chiusa sotto chiusura rotazionale.

b) **Dimostriamo che la classe dei linguaggi context-free è chiusa sotto chiusura rotazionale.**

Sia A un linguaggio context-free. Vogliamo dimostrare che $RC(A) = \{yx \mid xy \in A\}$ è anch'esso context-free.

Poiché A è context-free, esiste una grammatica context-free $G = (V, \Sigma, R, S)$ tale che $L(G) = A$.

Costruiamo una nuova grammatica $G' = (V', \Sigma, R', S')$ che genera $RC(A)$ come segue:

- $V' = V \cup \{S'\} \cup \{X_a \mid X \in V, a \in \Sigma\}$ (aggiungiamo un nuovo simbolo iniziale e nuovi non-terminali)
- S' è il nuovo simbolo iniziale

- R' include le seguenti regole:
  - $S' \to XY$ per ogni regola $S \to XY$ in R (simuliamo le regole originali)
  - $S' \to Xa$ per ogni regola $S \to aX$ in R (gestiamo il caso in cui la rotazione avviene dopo il primo carattere)
  - $X_a \to YZ_a$ per ogni regola $X \to YZ$ in R (propaghiamo il punto di rotazione)
  - $X_a \to Y_a b$ per ogni regola $X \to bY$ in R (gestiamo il caso in cui la rotazione avviene all'interno di una regola)
  - $X_a \to \varepsilon$ per ogni regola $X \to a\varepsilon$ in R (gestiamo il caso in cui la rotazione è proprio in corrispondenza di un terminale)

Tuttavia, questa costruzione è molto complicata e non è facile dimostrare formalmente che $L(G') = RC(A)$. Un approccio più semplice è utilizzare un teorema noto: ogni linguaggio context-free può essere riconosciuto da un PDA (automa a pila).

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

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

- $Q' = Q \cup \{q_0', q_1'\}$ (aggiungiamo due nuovi stati)
- $\Gamma' = \Gamma \cup \{\$\}$ (aggiungiamo un nuovo simbolo di stack)
- q_0' è il nuovo stato iniziale
- $Z_0' = Z_0$ (stesso simbolo iniziale dello stack)
- $F' = F$ (stessi stati finali)
- La funzione di transizione δ' include le seguenti transizioni:
  - $\delta'(q_0', \varepsilon, Z_0) = \{(q_1', \$Z_0)\}$ (inizializza lo stack con il simbolo speciale \$)
  - $\delta'(q_1', a, \alpha) = \{(q_1', a\alpha)\}$ per ogni $a \in \Sigma$ (legge e impila la prima parte della rotazione)
  - $\delta'(q_1', \varepsilon, \alpha) = \{(q_0, \alpha)\}$ (passa allo stato iniziale per elaborare la seconda parte)
  - Tutte le transizioni originali di δ (per elaborare la seconda parte)

L'idea è che P' "indovina" il punto di rotazione. Prima legge e impila la prima parte della rotazione (che corrisponde alla seconda parte della stringa originale in A), poi passa allo stato iniziale di P e continua a elaborare la seconda parte della rotazione (che corrisponde alla prima parte della stringa originale in A).

Quando P' riconosce la fine della seconda parte (raggiungendo il simbolo speciale \$ nello stack), può accettare se P accetterebbe la stringa originale.

Questo PDA accetta esattamente $RC(A)$, dimostrando che $RC(A)$ è context-free. Pertanto, la classe dei linguaggi context-free è chiusa sotto chiusura rotazionale.

SCRAMBLE di linguaggi 18. Per le stringhe w e t, scriviamo $w \cong t$ se i simboli di w sono una permutazione dei simboli di t. In altre parole, $w \cong t$ se t e w hanno gli stessi simboli nelle stesse quantità, ma possibilmente in un ordine diverso.

Per ogni stringa w, definiamo $\textsc{Scramble}(w) = \{t \mid t \cong w\}$. Per ogni linguaggio A, sia $\textsc{Scramble}(A) = \{t \mid t \in \textsc{Scramble}(w) \text{ per qualche } w \in A\}$.

a) Dimostra che se $\Sigma = \{0, 1\}$, allora lo Scramble di un linguaggio regolare è un linguaggio context-free.

b) Cosa succede nella parte (a) se Σ contiene tre o più simboli? Dimostra la risposta.

Soluzione. a) **Dimostriamo che se $\Sigma = \{0, 1\}$, allora lo Scramble di un linguaggio regolare è un linguaggio context-free.**

Sia A un linguaggio regolare su $\Sigma = \{0, 1\}$. Osserviamo che per qualsiasi stringa w su $\{0, 1\}$, tutte le permutazioni di w sono completamente determinate dal numero di '0' e dal numero di '1' in w.

Definiamo il linguaggio ausiliario $B = \{0^n1^m \mid \text{esiste } w \in A \text{ con esattamente } n \text{ '0' e } m \text{ '1'}\}$. Dimostriamo che B è regolare e che $\text{Scramble}(A) = \text{Scramble}(B)$.

Poiché A è regolare, esiste un DFA $M = (Q, \Sigma, \delta, q_0, F)$ che accetta A. Costruiamo un nuovo DFA $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ che accetta B:

- $Q_B = Q \times \mathbb{N} \times \mathbb{N}$ (ogni stato tiene traccia dello stato originale e del numero di '0' e '1' incontrati)
- $q_{0B} = (q_0, 0, 0)$ (inizialmente, non abbiamo incontrato né '0' né '1')
- $F_B = \{(q, n, m) \mid q \in F, n, m \in \mathbb{N}\}$ (accettiamo se lo stato originale è accettante)
- La funzione di transizione δ_B è definita come:

$$\delta_B((q, n, m), a) = \begin{cases} (\delta(q, 0), n + 1, m) & \text{se } a = 0 \\ (\delta(q, 1), n, m + 1) & \text{se } a = 1 \end{cases}$$

M_B simula M e conta il numero di '0' e '1' nella stringa di input. Accetta se M accetta la stringa originale.

Poiché il numero di stati di M è finito, e per la regolarità di A, esiste un insieme finito di valori di n e m che possono portare a Stati accettanti in M (quando leggono stringhe di lunghezza $n + m$). Inoltre, poiché M è un DFA, il comportamento su stringhe con lo stesso conteggio di '0' e '1' dipende solo da questi conteggi e non dall'ordine specifico dei simboli.

Quindi, possiamo semplificare Q_B considerando solo un insieme finito di triple (q, n, m), e B è effettivamente regolare.

Ora, dimostriamo che $\text{Scramble}(A) = \text{Scramble}(B)$:

($\text{Scramble}(A) \subseteq \text{Scramble}(B)$): Sia $t \in \text{Scramble}(A)$. Allora esiste $w \in A$ tale che $t \cong w$. Sia n il numero di '0' in w (e quindi in t) e m il numero di '1' in w (e quindi in t). Per costruzione, $0^n1^m \in B$. Poiché t ha esattamente n '0' e m '1', abbiamo $t \cong 0^n1^m$, quindi $t \in \text{Scramble}(B)$.

($\text{Scramble}(B) \subseteq \text{Scramble}(A)$): Sia $t \in \text{Scramble}(B)$. Allora esiste $v \in B$ tale che $t \cong v$. Per definizione di B, $v = 0^n1^m$ per qualche n, m tali che esiste $w \in A$ con esattamente n '0' e m '1'. Quindi, $w \cong 0^n1^m \cong t$, il che significa che $t \in \text{Scramble}(A)$.

Quindi, $\text{Scramble}(A) = \text{Scramble}(B)$.

Ora, dimostriamo che $\text{Scramble}(B)$ è context-free. Poiché B è regolare ed è formato solo da stringhe della forma 0^n1^m, possiamo definire l'insieme $C = \{(n, m) \mid 0^n1^m \in B\}$.

L'insieme C è infinito in generale, ma poiché B è regolare, C può essere rappresentato come un'unione finita di espressioni lineari della forma:

$$C = \bigcup_{i=1}^{k} \{(n, m) \mid n = a_i + b_i x, m = c_i + d_i x, x \in \mathbb{N}\}$$

per alcune costanti a_i, b_i, c_i, d_i.

Quindi, $\textsc{Scramble}(B) = \{w \in \{0,1\}^* \mid w$ ha esattamente n '0' e m '1' per qualche $(n, m) \in C\}$.

Per ogni espressione lineare $\{(n, m) \mid n = a_i + b_i x, m = c_i + d_i x, x \in \mathbb{N}\}$ in C, possiamo costruire una grammatica context-free che genera le stringhe con esattamente $a_i + b_i x$ '0' e $c_i + d_i x$ '1' per $x \in \mathbb{N}$.

Ad esempio, se $a_i = 1, b_i = 2, c_i = 0, d_i = 3$, la grammatica genererebbe stringhe con $(1+2x)$ '0' e $(0+3x)$ '1', cioè stringhe con numero dispari di '0' e numero di '1' multiplo di 3. Una possibile grammatica sarebbe:

$$\begin{aligned} S &\to 0T \\ T &\to 00U \mid \varepsilon \\ U &\to 111T \end{aligned}$$

Combinando le grammatiche per tutte le espressioni lineari in C, otteniamo una grammatica context-free per $\textsc{Scramble}(B) = \textsc{Scramble}(A)$.

Pertanto, se $\Sigma = \{0, 1\}$, allora lo Scramble di un linguaggio regolare è un linguaggio context-free.

b) **Cosa succede se Σ contiene tre o più simboli?**

Se Σ contiene tre o più simboli, lo Scramble di un linguaggio regolare non è necessariamente context-free.

Consideriamo il caso $\Sigma = \{a, b, c\}$ e il linguaggio regolare $A = \{a^n b^n c^n \mid n \geq 1\}$.

Il linguaggio $\textsc{Scramble}(A)$ consiste di tutte le stringhe che contengono esattamente lo stesso numero di 'a', 'b' e 'c'. Formalmente:

$$\textsc{Scramble}(A) = \{w \in \{a, b, c\}^* \mid w \text{ contiene esattamente } n \text{ 'a', } n \text{ 'b' e } n \text{ 'c' per qualche } n \geq 1\}$$

Dimostriamo che $\textsc{Scramble}(A)$ non è context-free utilizzando il pumping lemma per linguaggi context-free.

Supponiamo per assurdo che $\textsc{Scramble}(A)$ sia context-free. Allora, per il pumping lemma, esiste una costante $p > 0$ tale che ogni stringa $s \in \textsc{Scramble}(A)$ con $|s| \geq p$ può essere suddivisa in cinque parti $s = uvwxy$ tali che:

1. $|vwx| \leq p$
2. $|vx| > 0$
3. Per ogni $i \geq 0$, $uv^i wx^i y \in \textsc{Scramble}(A)$

Consideriamo la stringa $s = a^p b^p c^p \in \textsc{Scramble}(A)$. Per il pumping lemma, possiamo suddividere $s = uvwxy$ con le proprietà sopra descritte.

Poiché $|vwx| \leq p$, la sottostringa vwx deve essere contenuta interamente in uno dei blocchi ($a^p, b^p,$ o c^p) o al massimo in due blocchi adiacenti. Senza perdita di generalità, supponiamo che vwx sia contenuta nei blocchi a^p e b^p.

Questo significa che v e x contengono solo i caratteri 'a' e 'b', e non contengono 'c'. Sia $v = a^{n_a}b^{n_b}$ e $x = a^{m_a}b^{m_b}$, dove almeno uno tra n_a, n_b, m_a, m_b è positivo (poiché $|vx| > 0$).

Consideriamo la stringa $s' = uv^2wx^2y = uv \cdot v \cdot w \cdot x \cdot x \cdot y$. Applicando v e x una volta in più, aggiungiamo $n_a + m_a$ 'a' e $n_b + m_b$ 'b', ma nessun 'c'. Quindi, s' contiene $p + n_a + m_a$ 'a', $p + n_b + m_b$ 'b', e ancora p 'c'.

Per appartenere a $\textsc{Scramble}(A)$, s' dovrebbe contenere lo stesso numero di 'a', 'b' e 'c'. Ma questo è possibile solo se $n_a + m_a = 0$ e $n_b + m_b = 0$, il che contraddice $|vx| > 0$.

Pertanto, $\textsc{Scramble}(A)$ non può essere context-free.

Quindi, se Σ contiene tre o più simboli, lo Scramble di un linguaggio regolare non è necessariamente context-free.

All-NFA 19. Un *all-NFA* M è una 5-tupla $(Q, \Sigma, \delta, q_0, F)$ che accetta $x \in \Sigma^*$ se *ogni* possibile stato in cui M potrebbe trovarsi dopo aver letto l'input x è uno stato appartenente a F. Nota che questa definizione differisce da quella standard di un NFA.

Dimostra che gli all-NFA riconoscono esattamente la classe dei linguaggi regolari.

Soluzione. Dimostreremo che gli all-NFA riconoscono esattamente la classe dei linguaggi regolari. Lo faremo in due passaggi:

1. Dimostrare che ogni linguaggio regolare può essere riconosciuto da un all-NFA.
2. Dimostrare che ogni linguaggio riconosciuto da un all-NFA è regolare.

Passo 1: Ogni linguaggio regolare può essere riconosciuto da un all-NFA.

Sia L un linguaggio regolare. Allora esiste un DFA $D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$ tale che $L(D) = L$.

Costruiamo un all-NFA $M = (Q, \Sigma, \delta, q_0, F)$ che accetta L come segue:

- $Q = Q_D$ (stessi stati del DFA)
- $q_0 = q_{0D}$ (stesso stato iniziale)
- $F = F_D$ (stessi stati finali)
- La funzione di transizione δ è definita come:

$$\delta(q, a) = \begin{cases} \{\delta_D(q, a)\} & \text{se } q \in Q_D \\ \emptyset & \text{altrimenti} \end{cases}$$

In altre parole, M è essenzialmente identico a D, con la differenza che le transizioni restituiscono insiemi (singleton) anziché singoli stati.

Per ogni stringa $w \in \Sigma^*$, se D accetta w, allora $\delta_D^*(q_{0D}, w) \in F_D$. Poiché M simula esattamente D, l'unico stato in cui M può trovarsi dopo aver letto w è $\delta_D^*(q_{0D}, w)$. Se questo stato è in $F_D = F$, allora M accetta w.

Viceversa, se M accetta w, allora tutti gli stati in cui M può trovarsi dopo aver letto w (in questo caso, solo $\delta_D^*(q_{0D}, w)$) sono in $F = F_D$. Quindi, $\delta_D^*(q_{0D}, w) \in F_D$, cioè D accetta w.

Pertanto, $L(M) = L(D) = L$, dimostrando che ogni linguaggio regolare può essere riconosciuto da un all-NFA.

Passo 2: Ogni linguaggio riconosciuto da un all-NFA è regolare.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un all-NFA e sia $L(M)$ il linguaggio riconosciuto da M.

Osserviamo che una stringa w è accettata da M se e solo se tutti gli stati in $\delta^*(q_0, w)$ sono in F. Equivalentemente, w è rifiutata da M se e solo se esiste almeno uno stato in $\delta^*(q_0, w)$ che non è in F.

Sia M' un NFA standard costruito modificando M come segue:

- Gli stati, l'alfabeto e la funzione di transizione sono gli stessi di M.
- Lo stato iniziale è lo stesso di M.
- Gli stati finali sono $Q \setminus F$ (il complemento di F rispetto a Q).

Allora, M' accetta una stringa w se e solo se esiste almeno uno stato in $\delta^*(q_0, w)$ che è in $Q \setminus F$, cioè se e solo se w è rifiutata da M.

Quindi, $L(M') = \Sigma^* \setminus L(M)$ (il complemento di $L(M)$).

Poiché M' è un NFA standard, $L(M')$ è regolare. E poiché la classe dei linguaggi regolari è chiusa rispetto alla complementazione, anche $\Sigma^* \setminus L(M') = L(M)$ è regolare.

Pertanto, ogni linguaggio riconosciuto da un all-NFA è regolare.

Combinando i due passaggi, concludiamo che gli all-NFA riconoscono esattamente la classe dei linguaggi regolari.

Operazioni con lunghezze uguali 20. Se A e B sono linguaggi, definiamo $A \circ B = \{xy \mid x \in A \text{ e } y \in B \text{ e } |x| = |y|\}$.

a) Dimostra che se A e B sono linguaggi regolari, allora $A \circ B$ non è necessariamente regolare. Fornisci un controesempio.

b) Dimostra che se A e B sono linguaggi regolari, allora $A \circ B$ è context-free.

c) Cosa puoi dire se A è regolare e B è context-free?

Soluzione. a) **Dimostriamo che se A e B sono linguaggi regolari, allora $A \circ B$ non è necessariamente regolare.**

Consideriamo il seguente controesempio:

$$A = \{a^n \mid n \geq 0\}$$
$$B = \{b^n \mid n \geq 0\}$$

Entrambi A e B sono chiaramente linguaggi regolari (possono essere riconosciuti da semplici DFA).

Calcoliamo $A \circ B$:

$$\begin{aligned} A \circ B &= \{xy \mid x \in A, y \in B, |x| = |y|\} \\ &= \{a^n b^n \mid n \geq 0\} \end{aligned}$$

Ma $\{a^n b^n \mid n \geq 0\}$ è il linguaggio standard di esempio che non è regolare (può essere dimostrato con il pumping lemma come abbiamo fatto in esercizi precedenti).

Quindi, anche se A e B sono regolari, $A \circ B$ non è necessariamente regolare.

b) **Dimostriamo che se A e B sono linguaggi regolari, allora $A \circ B$ è context-free.**

Sia A e B linguaggi regolari. Vogliamo dimostrare che $A \circ B = \{xy \mid x \in A, y \in B, |x| = |y|\}$ è context-free.

Poiché A e B sono regolari, esistono DFA $M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ e $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ tali che $L(M_A) = A$ e $L(M_B) = B$.

Costruiamo un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ che accetta $A \circ B$ come segue:

- $Q = \{q_0, q_1, q_f\} \cup Q_A \cup Q_B$ (aggiungiamo tre nuovi stati)
- $\Gamma = \Sigma \cup \{Z_0\}$ (usiamo i simboli dell'alfabeto di input come simboli di stack, più un simbolo iniziale)
- q_0 è lo stato iniziale
- $F = \{q_f\}$ (un singolo stato finale)
- La funzione di transizione δ include le seguenti transizioni:
  - $\delta(q_0, \varepsilon, Z_0) = \{(q_{0A}, Z_0)\}$ (iniziamo simulando M_A)
  - Per ogni $q \in Q_A$, $a \in \Sigma$, $\gamma \in \Gamma$, $\delta(q, a, \gamma) = \{(\delta_A(q, a), a\gamma)\}$ (simuliamo M_A e impiliamo il carattere letto)
  - Per ogni $q \in F_A$, $\gamma \in \Gamma$, $\delta(q, \varepsilon, \gamma) = \{(q_{0B}, \gamma)\}$ (se raggiungiamo uno stato finale di M_A, passiamo a simulare M_B)
  - Per ogni $q \in Q_B$, $a \in \Sigma$, $b \in \Sigma$, $\delta(q, a, b) = \{(\delta_B(q, a), \varepsilon)\}$ (simuliamo M_B e togliamo un carattere dallo stack per ogni carattere letto)
  - Per ogni $q \in F_B$, $\delta(q, \varepsilon, Z_0) = \{(q_f, Z_0)\}$ (se raggiungiamo uno stato finale di M_B e lo stack è vuoto eccetto Z_0, accettiamo)

L'idea è che P prima simula M_A su una parte dell'input, impilando ogni carattere letto. Quando decide di passare alla seconda parte, deve essere in uno stato finale di M_A (per garantire che la prima parte $x \in A$). Poi simula M_B sulla parte rimanente dell'input, rimuovendo un carattere dallo stack per ogni carattere letto. Se riesce a raggiungere uno stato finale di M_B e lo stack è vuoto eccetto Z_0, significa che ha letto stringhe $x \in A$ e $y \in B$ con $|x| = |y|$, quindi accetta.

Formalmente, dimostriamo che $L(P) = A \circ B$:

($L(P) \subseteq A \circ B$): Sia $w \in L(P)$. Allora P accetta w, il che significa che P può leggere w e terminare nello stato q_f con lo stack contenente solo Z_0.

Per accettare w, P deve:

1. Leggere una prima parte x di w, simulando M_A e impilando i caratteri di x.
2. Raggiungere uno stato finale di M_A, il che significa che $x \in A$.
3. Passare a simulare M_B sulla parte rimanente $y = w - x$ di w.
4. Raggiungere uno stato finale di M_B con lo stack vuoto (eccetto Z_0), il che significa che $y \in B$ e $|y| = |x|$ (poiché abbiamo rimosso tanti caratteri dallo stack quanti ne abbiamo letti in y).

Quindi, $w = xy$ con $x \in A$, $y \in B$ e $|x| = |y|$, cioè $w \in A \circ B$.

($A \circ B \subseteq L(P)$): Sia $w \in A \circ B$. Allora $w = xy$ per qualche $x \in A$ e $y \in B$ con $|x| = |y|$.

Poiché $x \in A$, M_A accetta x, quindi esiste una sequenza di stati $q_{0A}, q_1, \ldots, q_n$ tale che $q_n \in F_A$ e $\delta_A(q_i, x_{i+1}) = q_{i+1}$ per $0 \leq i < n$.

Analogamente, poiché $y \in B$, M_B accetta y, quindi esiste una sequenza di stati $q_{0B}, r_1, \ldots, r_m$ tale che $r_m \in F_B$ e $\delta_B(r_i, y_{i+1}) = r_{i+1}$ per $0 \leq i < m$.

Allora P può elaborare $w = xy$ come segue:

1. Inizia nello stato q_0 con lo stack contenente solo Z_0.
2. Passa allo stato q_{0A}, mantenendo Z_0 nello stack.
3. Legge x carattere per carattere, simulando M_A e impilando ogni carattere. Dopo aver letto x, P è nello stato $q_n \in F_A$ e lo stack contiene $x^R Z_0$ (con x^R in cima).
4. Passa allo stato q_{0B}.
5. Legge y carattere per carattere, simulando M_B e rimuovendo un carattere dallo stack per ogni carattere letto. Poiché $|y| = |x|$, dopo aver letto y lo stack contiene solo Z_0.
6. Poiché $r_m \in F_B$, P può passare allo stato q_f, mantenendo Z_0 nello stack.

Quindi, P accetta w, cioè $w \in L(P)$.

Poiché $L(P) = A \circ B$ e P è un PDA, concludiamo che $A \circ B$ è context-free.

c) **Cosa succede se A è regolare e B è context-free?**

Se A è regolare e B è context-free, allora $A \circ B = \{xy \mid x \in A, y \in B, |x| = |y|\}$ è ancora context-free.

La dimostrazione è simile a quella della parte (b), ma invece di simulare un DFA per B, il PDA deve simulare un PDA per B. Questo rende la costruzione più complessa, ma concettualmente simile.

Sia $P_B = (Q_B, \Sigma, \Gamma_B, \delta_B, q_{0B}, Z_{0B}, F_B)$ un PDA che accetta B per stato finale.

Costruiamo un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ che accetta $A \circ B$ come segue:

- $Q = \{q_0, q_1\} \cup Q_A \cup Q_B$ (aggiungiamo due nuovi stati)
- $\Gamma = \Sigma \cup \Gamma_B \cup \{Z_0, \#\}$ (aggiungiamo un simbolo speciale # per separare le due parti dello stack)
- q_0 è lo stato iniziale
- $F = F_B$ (stessi stati finali di P_B)
- La funzione di transizione δ include le seguenti transizioni:
  - $\delta(q_0, \varepsilon, Z_0) = \{(q_{0A}, Z_0)\}$ (iniziamo simulando M_A)
  - Per ogni $q \in Q_A$, $a \in \Sigma$, $\gamma \in \Gamma$, $\delta(q, a, \gamma) = \{(\delta_A(q, a), a\gamma)\}$ (simuliamo M_A e impiliamo il carattere letto)

- Per ogni $q \in F_A$, $\gamma \in \Gamma$, $\delta(q, \varepsilon, \gamma) = \{(q_1, \gamma)\}$ (se raggiungiamo uno stato finale di M_A, passiamo a uno stato intermedio)
- $\delta(q_1, \varepsilon, Z_0) = \{(q_{0B}, \#Z_{0B}Z_0)\}$ (prepariamo lo stack per simulare P_B)
- Per ogni transizione $\delta_B(q, a, \gamma) = \{(p, \alpha)\}$ in P_B, includiamo $\delta(q, a, \gamma) = \{(p, \alpha)\}$ (simuliamo P_B)
- Per ogni $q \in F_B$, $a \in \Sigma$, $\gamma \in \Gamma$, se $\gamma \neq \#$, includiamo $\delta(q, a, \gamma) = \emptyset$ (rifiutiamo se c'è altro nello stack oltre al separatore #)
- Per ogni $q \in F_B$, includiamo $\delta(q, \varepsilon, \#) = \{(q, \varepsilon)\}$ (rimuoviamo il separatore)
- Per ogni $q \in F_B$, $a \in \Sigma$, $\delta(q, a, a) = \{(q, \varepsilon)\}$ (rimuoviamo i caratteri impilati durante la simulazione di M_A)
- Per ogni $q \in F_B$, $\delta(q, \varepsilon, Z_0) = \{(q, Z_0)\}$ (accettiamo se lo stack è vuoto eccetto Z_0)

L'idea è che P prima simula M_A su una parte dell'input, impilando ogni carattere letto. Quando passa a simulare P_B, aggiunge un separatore # e il simbolo iniziale Z_{0B} di P_B allo stack. Poi simula P_B sulla parte rimanente dell'input. Se P_B accetta e lo stack sotto il separatore # contiene esattamente tanti caratteri quanti ne ha letti nella seconda parte, allora P accetta.

La dimostrazione formale che $L(P) = A \circ B$ è analoga a quella della parte (b).

Pertanto, se A è regolare e B è context-free, allora $A \circ B$ è context-free.