Esercizi di Automi e Linguaggi Formali

Prima Parte: Linguaggi Regolari, Pumping Lemma, e Linguaggi Context-Free

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

Aprile 2025

1 Chiusura dei Linguaggi Regolari

Esercizio 1. Dati i linguaggi A e B, lo shuffle perfetto di A e B è il linguaggio

$$S = \{w \mid w = a_1 b_1 a_2 b_2 \dots a_k b_k, \text{ dove } a_1 a_2 \dots a_k \in A \text{ e } b_1 b_2 \dots b_k \in B, \text{ ogni } a_i, b_i \in \Sigma\}$$

Mostrare che la classe dei linguaggi regolari è chiusa rispetto allo shuffle perfetto, cioè che se A e B sono linguaggi regolari allora anche il loro shuffle perfetto è un linguaggio regolare.

Soluzione. Per dimostrare che lo shuffle perfetto di due linguaggi regolari è regolare, costruiremo un automa che riconosce tale linguaggio a partire dagli automi che riconoscono A e B.

Siano $M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ e $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ due DFA che riconoscono rispettivamente A e B.

Costruiamo un nuovo NFA $M = (Q, \Sigma, \delta, q_0, F)$ che riconosce lo shuffle perfetto di A e B come segue:

- $Q = Q_A \times Q_B \times \{0, 1\}$, dove il terzo componente indica se il prossimo simbolo dovrebbe essere da A (valore 0) o da B (valore 1)
- $q_0 = (q_{0A}, q_{0B}, 0)$
- $F = \{(q_A, q_B, 1) \mid q_A \in F_A \text{ e } q_B \in F_B\}$
- La funzione di transizione δ è definita come:
  - $\delta((q_A, q_B, 0), a) = \{(\delta_A(q_A, a), q_B, 1)\}$ per ogni $a \in \Sigma$
  - $\delta((q_A, q_B, 1), b) = \{(q_A, \delta_B(q_B, b), 0)\}$ per ogni $b \in \Sigma$

L'automa M funziona alternando la lettura di simboli per A e B. Inizia leggendo un simbolo per A (stato con terzo componente 0), poi passa a leggere un simbolo per B (stato con terzo componente 1), e così via.

L'automa accetta solo se, dopo aver letto un numero pari di simboli (l'ultimo da B), entrambi gli stati per A e B sono di accettazione.

Poiché gli NFA riconoscono esattamente i linguaggi regolari, e M è un NFA, lo shuffle perfetto di A e B è un linguaggio regolare.

Esercizio 2. Sia A un linguaggio, e sia DROP_OUT(A) il linguaggio contenente tutte le stringhe che possono essere ottenute togliendo un simbolo da una stringa di A:

$$DROP\_OUT(A) = \{xz \mid xyz \in A \text{ dove } x, z \in \Sigma^* \text{ e } y \in \Sigma\}.$$

Mostrare che la classe dei linguaggi regolari è chiusa rispetto all'operazione DROP_OUT, cioè che se A è un linguaggio regolare allora DROP_OUT(A) è un linguaggio regolare.

Soluzione. Per dimostrare che DROP_OUT(A) è regolare quando A è regolare, costruiremo un NFA che riconosce DROP_OUT(A) a partire dal DFA che riconosce A.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce il linguaggio regolare A. Costruiamo un NFA $M' = (Q', \Sigma, \delta', q_0', F')$ che riconosce DROP_OUT(A):

- $Q' = Q \times \{0, 1\}$, dove il secondo componente indica se abbiamo già eliminato un carattere (valore 1) o no (valore 0)
- $q_0' = (q_0, 0)$
- $F' = \{(q, 1) \mid q \in F\}$
- La funzione di transizione δ' è definita come:
  - $\delta'((q, 0), a) = \{(\delta(q, a), 0), (\delta(\delta(q, a), b), 1) \mid b \in \Sigma\}$ per ogni $a \in \Sigma$
  - $\delta'((q, 1), a) = \{(\delta(q, a), 1)\}$ per ogni $a \in \Sigma$
  - $\delta'((q, 0), \varepsilon) = \{(\delta(q, b), 1) \mid b \in \Sigma\}$ (transizione ε per gestire l'eliminazione del primo carattere)

Spieghiamo come funziona questo NFA:

- Iniziamo nello stato $(q_0, 0)$, indicando che non abbiamo ancora eliminato alcun carattere.
- Per ogni carattere che leggiamo, abbiamo due possibilità:
  - Manteniamo il carattere, passando allo stato $(\delta(q, a), 0)$
  - Eliminiamo il carattere e passiamo direttamente al prossimo stato come se avessimo letto un carattere aggiuntivo b (per ogni possibile $b \in \Sigma$), passando allo stato $(\delta(\delta(q, a), b), 1)$
- Una volta che abbiamo eliminato un carattere (secondo componente è 1), ci comportiamo come il DFA originale.

- Aggiungiamo anche una transizione ε iniziale per gestire il caso in cui eliminiamo il primo carattere della stringa.
- Accettiamo se, dopo aver eliminato esattamente un carattere, raggiungiamo uno stato di accettazione del DFA originale.

Poiché gli NFA riconoscono esattamente i linguaggi regolari, e M' è un NFA, DROP_OUT(A) è un linguaggio regolare quando A è regolare.

Esercizio 3. Per una stringa $w = w_1 w_2 \dots w_n$, l'inversa di w è la stringa $w^R = w_n \dots w_2 w_1$. Per ogni linguaggio A, sia $A^R = \{w^R \mid w \in A\}$. Mostrare che se A è regolare allora lo è anche A^R.

Soluzione. Per dimostrare che A^R è regolare quando A è regolare, costruiremo un automa che riconosce A^R a partire dall'automa che riconosce A.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce il linguaggio regolare A. Costruiamo un NFA $M' = (Q, \Sigma, \delta', F, \{q_0\})$ che riconosce A^R come segue:

- Usiamo gli stessi stati Q
- Lo stato iniziale di M' è l'insieme degli stati finali F di M
- Gli stati finali di M' contengono solo lo stato iniziale $\{q_0\}$ di M
- La funzione di transizione δ' è inversa rispetto a δ:
  - $\delta'(q, a) = \{p \in Q \mid \delta(p, a) = q\}$ per ogni $q \in Q$ e $a \in \Sigma$

In pratica, abbiamo invertito la direzione di tutte le transizioni, scambiato lo stato iniziale con gli stati finali, e ora M' accetta la stringa w^R se e solo se M accetta w.

L'NFA M' può essere trasformato in un DFA equivalente utilizzando la costruzione standard di subset. Poiché gli NFA riconoscono esattamente i linguaggi regolari, anche A^R è un linguaggio regolare.

Esercizio 4. Sia $A/b = \{w \mid wb \in A\}$. Mostrare che se A è un linguaggio regolare e $b \in \Sigma$, allora A/b è regolare.

Soluzione. Per dimostrare che A/b è regolare quando A è regolare, costruiremo un automa che riconosce A/b a partire dall'automa che riconosce A.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce il linguaggio regolare A. Costruiamo un DFA $M' = (Q, \Sigma, \delta, q_0, F')$ che riconosce A/b come segue:

- Utilizziamo gli stessi stati Q, la stessa funzione di transizione δ e lo stesso stato iniziale q_0
- Modifichiamo solo l'insieme degli stati finali:
  - $F' = \{q \in Q \mid \delta(q, b) \in F\}$

In altre parole, M' accetta uno stato q se e solo se, leggendo il simbolo b da questo stato in M, si raggiunge uno stato di accettazione.

Dimostriamo che $L(M') = A/b$:

- Se $w \in A/b$, allora $wb \in A$. Questo significa che $\delta^*(q_0, wb) \in F$, dove δ^* è l'estensione di δ alle stringhe. Ma $\delta^*(q_0, wb) = \delta(\delta^*(q_0, w), b)$, quindi $\delta(\delta^*(q_0, w), b) \in F$. Per la definizione di F', abbiamo che $\delta^*(q_0, w) \in F'$, quindi $w \in L(M')$.

- Se $w \in L(M')$, allora $\delta^*(q_0, w) \in F'$. Per la definizione di F', abbiamo che $\delta(\delta^*(q_0, w), b) \in F$. Ma $\delta(\delta^*(q_0, w), b) = \delta^*(q_0, wb)$, quindi $\delta^*(q_0, wb) \in F$, il che significa che $wb \in A$. Di conseguenza, $w \in A/b$.

Poiché M' è un DFA, A/b è un linguaggio regolare quando A è regolare.

Esercizio 5. Sia $A/B = \{w \mid wx \in A \text{ per qualche } x \in B\}$. Mostrare che se A è un linguaggio regolare e B un linguaggio qualsiasi, allora A/B è regolare.

Soluzione. Per dimostrare che A/B è regolare quando A è regolare (indipendentemente da B), useremo le proprietà degli automi a stati finiti.

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce il linguaggio regolare A. Definiamo un nuovo DFA $M' = (Q, \Sigma, \delta, q_0, F')$ che riconosce A/B come segue:

- $F' = \{q \in Q \mid \exists x \in B \text{ tale che } \delta^*(q, x) \in F\}$

Dove δ^* è l'estensione di δ alle stringhe.

In altre parole, M' accetta uno stato q se e solo se esiste qualche stringa $x \in B$ tale che, leggendo x da q in M, si raggiunge uno stato di accettazione.

Dimostriamo che $L(M') = A/B$:

- Se $w \in A/B$, allora esiste $x \in B$ tale che $wx \in A$. Questo significa che $\delta^*(q_0, wx) \in F$. Ma $\delta^*(q_0, wx) = \delta^*(\delta^*(q_0, w), x)$, quindi $\delta^*(\delta^*(q_0, w), x) \in F$. Per la definizione di F', abbiamo che $\delta^*(q_0, w) \in F'$, quindi $w \in L(M')$.

- Se $w \in L(M')$, allora $\delta^*(q_0, w) \in F'$. Per la definizione di F', esiste $x \in B$ tale che $\delta^*(\delta^*(q_0, w), x) \in F$. Ma $\delta^*(\delta^*(q_0, w), x) = \delta^*(q_0, wx)$, quindi $\delta^*(q_0, wx) \in F$, il che significa che $wx \in A$. Di conseguenza, $w \in A/B$.

Osserviamo che F' è ben definito anche se B è un linguaggio infinito o non regolare. L'insieme F' dipende solo dalle proprietà dell'automa M e non richiede di costruire un automa per B. In particolare, uno stato q è in F' se e solo se l'intersezione tra il linguaggio $L_q = \{x \mid \delta^*(q, x) \in F\}$ e B è non vuota.

Poiché M' è un DFA (differisce da M solo per l'insieme degli stati finali), A/B è un linguaggio regolare quando A è regolare, indipendentemente dalla natura di B.

Esercizio 6. Il pumping lemma afferma che ogni linguaggio regolare ha una lunghezza del pumping p, tale che ogni stringa del linguaggio può essere iterata se ha lunghezza maggiore o uguale a p. La lunghezza minima del pumping per un linguaggio A è il più piccolo p che è una lunghezza del pumping per A. Per ognuno dei seguenti linguaggi, dare la lunghezza minima del pumping e giustificare la risposta.

(a) 110^*

(b) $1^*0^*1^*$

(c) $0^*1^*0^*1^* + 10^*1$

(d) $(01)^*$

(e) $\emptyset$

(f) $0^*01^*01^*$

(g) $10(11^*0)^*0$

(h) 101101

(i) $\{w \in \Sigma^* \mid w \neq 101101\}$

(j) 0001^*

(k) 0^*1^*

(l) $001 + 0^*1^*$

(m) ε

(n) $1^*01^*01^*$

(o) 1011

(p) Σ^*

Soluzione. Ricordiamo che la lunghezza minima del pumping per un linguaggio regolare è il numero di stati del DFA minimo che lo riconosce.

(a) 110^*: La lunghezza minima del pumping è 3.

Un DFA minimo per questo linguaggio ha 3 stati: uno per riconoscere il prefisso vuoto, uno per riconoscere il prefisso "1", e uno stato di accettazione per riconoscere il prefisso "11" seguito da qualsiasi numero di "0". Non può essere implementato con meno stati perché dobbiamo contare esattamente due "1" all'inizio.

(b) $1^*0^*1^*$: La lunghezza minima del pumping è 3.

Un DFA minimo per questo linguaggio ha 3 stati: uno stato per riconoscere il prefisso di soli "1", uno stato per riconoscere un prefisso di "1" seguito da "0", e uno stato per riconoscere un prefisso di "1" seguito da "0" seguito da "1". Non può essere implementato con meno stati perché dobbiamo assicurarci che i caratteri appaiano nell'ordine corretto (prima tutti gli "1", poi tutti gli "0", infine tutti gli "1").

(c) $0^*1^*0^*1^* + 10^*1$: La lunghezza minima del pumping è 5.

Un DFA minimo per questo linguaggio deve gestire separatamente i due casi. Per $0^*1^*0^*1^*$ servono 4 stati come nell'esercizio precedente plus estensione, e per 10^*1 servono 3 stati. Combinandoli ottimamente, otteniamo un DFA minimo con 5 stati.

(d) $(01)^*$: La lunghezza minima del pumping è 2.

Un DFA minimo per questo linguaggio ha 2 stati: uno stato iniziale che accetta la stringa vuota e che si aspetta "0", e uno stato che si aspetta "1". Non può essere implementato con meno stati perché dobbiamo verificare l'alternanza dei caratteri.

(e) $\emptyset$: La lunghezza minima del pumping è 1.

Un DFA minimo per il linguaggio vuoto ha un solo stato che non è di accettazione e nessuna transizione che porta a stati di accettazione.

(f) $0^*01^*01^*$: La lunghezza minima del pumping è 4.

Un DFA minimo per questo linguaggio ha 4 stati: uno per riconoscere il prefisso di soli "0", uno per riconoscere il prefisso che termina con il primo "0" non seguito da altri "0", uno per riconoscere il prefisso che termina con il secondo "0", e infine uno stato di accettazione. Non può essere implementato con meno stati perché dobbiamo contare esattamente due occorrenze di "0" separate da "1".

(g) $10(11^*0)^*0$: La lunghezza minima del pumping è 4.

Un DFA minimo per questo linguaggio ha 4 stati: uno per riconoscere il prefisso "1", uno per riconoscere il prefisso "10", uno per gestire il ciclo $(11^*0)^*$, e uno stato di accettazione per riconoscere l'ultimo "0".

(h) 101101: La lunghezza minima del pumping è 7.

Un DFA minimo per questo linguaggio ha 7 stati: uno per ogni prefisso della stringa, più uno stato di "trappola". Ogni linguaggio finito con una stringa di lunghezza n ha lunghezza minima del pumping $n + 1$.

(i) $\{w \in \Sigma^* \mid w \neq 101101\}$: La lunghezza minima del pumping è 7.

Questo è il complemento del linguaggio precedente, e la lunghezza minima del pumping per il complemento di un linguaggio è la stessa del linguaggio originale.

(j) 0001^*: La lunghezza minima del pumping è 4.

Un DFA minimo per questo linguaggio ha 4 stati: uno per riconoscere il prefisso vuoto, uno per riconoscere il prefisso "0", uno per riconoscere il prefisso "00", e uno stato di accettazione per riconoscere il prefisso "000" seguito da qualsiasi numero di "1". Non può essere implementato con meno stati perché dobbiamo contare esattamente tre "0" all'inizio.

(k) 0^*1^*: La lunghezza minima del pumping è 2.

Un DFA minimo per questo linguaggio ha 2 stati: uno stato per riconoscere il prefisso di soli "0", e uno stato per riconoscere un prefisso di "0" seguito da "1". Non può essere implementato con meno stati perché dobbiamo assicurarci che i caratteri appaiano nell'ordine corretto.

(l) $001 + 0^*1^*$: La lunghezza minima del pumping è 4.

Un DFA minimo per questo linguaggio deve riconoscere sia 001 che 0^*1^*. Per il primo servono 4 stati, per il secondo servono 2 stati. Combinandoli ottimalmente, otteniamo un DFA minimo con 4 stati.

(m) ε: La lunghezza minima del pumping è 1.

Un DFA minimo per il linguaggio che contiene solo la stringa vuota ha un solo stato che è sia iniziale che di accettazione, e nessuna transizione che porta ad altri stati.

(n) $1^*01^*01^*$: La lunghezza minima del pumping è 3.

Un DFA minimo per questo linguaggio ha 3 stati: uno per riconoscere il prefisso di soli "1", uno per riconoscere il prefisso che contiene un "0", e uno stato di accettazione per riconoscere il prefisso che contiene due "0". Non può essere implementato con meno stati perché dobbiamo contare esattamente due occorrenze di "0".

(o) 1011: La lunghezza minima del pumping è 5.

Un DFA minimo per questo linguaggio ha 5 stati: uno per ogni prefisso della stringa, più uno stato di "trappola". Come per il caso del linguaggio 101101, ogni linguaggio finito con una stringa di lunghezza n ha lunghezza minima del pumping $n + 1$.

(p) Σ^*: La lunghezza minima del pumping è 1.

Un DFA minimo per il linguaggio di tutte le stringhe ha un solo stato che è sia iniziale che di accettazione, e transizioni che rimangono nello stesso stato per ogni simbolo dell'alfabeto.

Esercizio 7. Sia $\Sigma = \{0, 1\}$, e considerate il linguaggio

$$D = \{w \mid w \text{ contiene un ugual numero di occorrenze di 01 e di 10}\}$$

Mostrare che D è un linguaggio regolare.

Soluzione. Per dimostrare che D è un linguaggio regolare, costruiremo un automa a stati finiti che lo riconosce.

Osserviamo innanzitutto alcune proprietà del linguaggio D. In una stringa w, le sottostringhe "01" e "10" rappresentano transizioni tra i caratteri: "01" rappresenta una transizione da 0 a 1, mentre "10" rappresenta una transizione da 1 a 0.

Se consideriamo la prima e l'ultima lettera della stringa w, possiamo notare che:

- Se w inizia e finisce con lo stesso carattere (entrambi 0 o entrambi 1), allora il numero di transizioni da 0 a 1 deve essere uguale al numero di transizioni da 1 a 0, perché ogni volta che passiamo da 0 a 1, dobbiamo eventualmente tornare da 1 a 0, e viceversa.
- Se w inizia con 0 e finisce con 1, allora il numero di transizioni da 0 a 1 deve essere maggiore di 1 rispetto al numero di transizioni da 1 a 0.
- Se w inizia con 1 e finisce con 0, allora il numero di transizioni da 1 a 0 deve essere maggiore di 1 rispetto al numero di transizioni da 0 a 1.

Quindi, $w \in D$ se e solo se:

- w inizia e finisce con lo stesso carattere, oppure
- w è la stringa vuota.

Possiamo costruire un DFA con 4 stati che riconosce D:

- q_0: Stato iniziale e di accettazione (rappresenta la stringa vuota o stringhe con ugual numero di "01" e "10")

- q_1: Stato che rappresenta stringhe che terminano con 0 e hanno un "01" in più rispetto a "10"
- q_2: Stato che rappresenta stringhe che terminano con 1 e hanno un "10" in più rispetto a "01"
- q_3: Stato che rappresenta stringhe che terminano con 1 e hanno un "01" in più rispetto a "10"

Le transizioni sono definite come segue:

- $\delta(q_0, 0) = q_1$
- $\delta(q_0, 1) = q_2$
- $\delta(q_1, 0) = q_1$
- $\delta(q_1, 1) = q_3$
- $\delta(q_2, 0) = q_1$
- $\delta(q_2, 1) = q_2$
- $\delta(q_3, 0) = q_0$
- $\delta(q_3, 1) = q_2$

Questo DFA accetta esattamente le stringhe che hanno lo stesso numero di occorrenze di "01" e "10", quindi D è un linguaggio regolare.

Esercizio 8. Sia $\Sigma = \{0, 1\}$.

- Mostrare che il linguaggio $A = \{0^k u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ è regolare.
- Mostrare che il linguaggio $B = \{0^k 1 u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ non è regolare.

Soluzione. **Parte 1**: Dimostriamo che $A = \{0^k u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ è regolare.

L'idea chiave è che il linguaggio A può essere riscritto come

$$A = \{0^k u 0^j \mid k, j \geq 1 \text{ e } u \in \Sigma^* \text{ e } k = j\}$$

Ma poiché un automa a stati finiti non può "contare" e memorizzare il valore di k per confrontarlo con j quando questi possono essere arbitrariamente grandi, sembra che A non sia regolare. Tuttavia, c'è un'osservazione importante: la definizione di A richiede che la stringa inizi con almeno un carattere 0 e termini con almeno un carattere 0, ma non pone vincoli sulla lunghezza della stringa u o sul suo contenuto.

Quindi A può essere equivalentemente descritto come

$$A = \{0w0 \mid w \in \Sigma^*\}$$

che è chiaramente un linguaggio regolare. Possiamo costruire un DFA con 3 stati che riconosce A:

- q_0: Stato iniziale
- q_1: Stato raggiunto dopo aver letto il primo 0
- q_2: Stato di accettazione raggiunto dopo aver letto almeno un altro 0 dopo q_1

Le transizioni sono definite come segue:

- $\delta(q_0, 0) = q_1$
- $\delta(q_0, 1)$ non è definita (o porta a uno stato di "trappola")
- $\delta(q_1, 0) = q_2$
- $\delta(q_1, 1) = q_1$
- $\delta(q_2, 0) = q_2$
- $\delta(q_2, 1) = q_1$

Questo DFA accetta esattamente le stringhe in A, quindi A è un linguaggio regolare.

Parte 2: Dimostriamo che $B = \{0^k 1 u 0^k \mid k \geq 1 \text{ e } u \in \Sigma^*\}$ non è regolare.

Utilizzeremo il Pumping Lemma per linguaggi regolari. Supponiamo per assurdo che B sia regolare. Allora esiste una costante $p > 0$ tale che ogni stringa $w \in B$ con $|w| \geq p$ può essere scritta come $w = xyz$ con $|xy| \leq p$, $|y| > 0$, e per ogni $i \geq 0$, $xy^i z \in B$.

Consideriamo la stringa $w = 0^p 1 0^p \in B$ (con $u =$ stringa vuota). Poiché $|w| > p$, possiamo applicare il Pumping Lemma. Dato che $|xy| \leq p$, xy è un prefisso di w contenente solo caratteri 0. Quindi $y = 0^j$ per qualche $j > 0$.

Scegliendo $i = 0$, otteniamo $xy^0 z = xz = 0^{p-j} 1 0^p$. Questa stringa non è in B perché il numero di 0 prima dell'1 è diverso dal numero di 0 dopo l'ultimo carattere (avremmo $0^{p-j} 1 0^p$ con $p - j < p$).

Questo contraddice l'ipotesi che B sia regolare, quindi B non è un linguaggio regolare.

Esercizio 9. Dimostrare che i seguenti linguaggi non sono regolari.

(a) $\{0^n 1^m 0^m \mid m, n \geq 0\}$

(b) $\{0^n 1^m \mid n \neq m\}$

(c) $\{w \in \{0,1\}^* \mid w \text{ non è palindroma}\}$

(d) $\{wtw \mid w, t \in \{0,1\}^+\}$, dove $\{0,1\}^+$ è l'insieme di tutte le stringhe binarie di lunghezza maggiore o uguale a 1

Soluzione. **(a)** Dimostriamo che $\{0^n 1^m 0^m \mid m, n \geq 0\}$ non è regolare.

Utilizzeremo il Pumping Lemma. Supponiamo per assurdo che il linguaggio sia regolare. Allora esiste una costante $p > 0$ tale che ogni stringa w nel linguaggio con $|w| \geq p$ può essere scritta come $w = xyz$ con $|xy| \leq p$, $|y| > 0$, e per ogni $i \geq 0$, $xy^i z \in L$.

Consideriamo la stringa $w = 0^p 1^p 0^p$. Poiché $|w| > p$, possiamo applicare il Pumping Lemma. Dato che $|xy| \leq p$, xy è un prefisso di w contenente solo caratteri 0. Quindi $y = 0^j$ per qualche $j > 0$.

Scegliendo $i = 2$, otteniamo $xy^2z = 0^{p+j}1^p0^p$. Questa stringa non è nel linguaggio perché il numero di 0 all'inizio è diverso da p, ma il numero di 1 e il numero di 0 alla fine sono entrambi p.

Questo contraddice l'ipotesi che il linguaggio sia regolare, quindi $\{0^n1^m0^m \mid m, n \geq 0\}$ non è regolare.

(b) Dimostriamo che $\{0^n1^m \mid n \neq m\}$ non è regolare.

Utilizzeremo il Pumping Lemma. Supponiamo per assurdo che il linguaggio sia regolare. Allora esiste una costante $p > 0$ tale che ogni stringa w nel linguaggio con $|w| \geq p$ può essere scritta come $w = xyz$ con $|xy| \leq p$, $|y| > 0$, e per ogni $i \geq 0$, $xy^iz \in L$.

Consideriamo la stringa $w = 0^{p+1}1^p$. Poiché $|w| > p$, possiamo applicare il Pumping Lemma. Dato che $|xy| \leq p$, xy è un prefisso di w contenente solo caratteri 0. Quindi $y = 0^j$ per qualche $j > 0$.

Scegliendo i in modo che $p+1-j \cdot i = p$, che equivale a $i = 1/j$, otteniamo $xy^iz = 0^p1^p$. Questa stringa non è nel linguaggio perché il numero di 0 è uguale al numero di 1.

Poiché $j > 0$, esiste sempre un valore intero di $i \geq 0$ tale che $xy^iz \notin L$, il che contraddice l'ipotesi che il linguaggio sia regolare. Quindi $\{0^n1^m \mid n \neq m\}$ non è regolare.

(c) Dimostriamo che $\{w \in \{0,1\}^* \mid w \text{ non è palindroma}\}$ non è regolare.

Ricordiamo che una stringa è palindroma se è uguale alla sua inversa, cioè se $w = w^R$.

Invece di usare direttamente il Pumping Lemma, useremo il fatto che il complemento di un linguaggio regolare è regolare. Quindi, se il linguaggio delle stringhe non palindrome fosse regolare, anche il linguaggio delle stringhe palindrome sarebbe regolare.

Consideriamo il linguaggio delle stringhe palindrome $P = \{w \in \{0,1\}^* \mid w = w^R\}$. Utilizzeremo il Pumping Lemma per dimostrare che P non è regolare.

Supponiamo per assurdo che P sia regolare. Allora esiste una costante $p > 0$ tale che ogni stringa $w \in P$ con $|w| \geq p$ può essere scritta come $w = xyz$ con $|xy| \leq p$, $|y| > 0$, e per ogni $i \geq 0$, $xy^iz \in P$.

Consideriamo la stringa $w = 0^p10^p$. Questa stringa è palindroma e ha lunghezza $2p + 1 > p$, quindi possiamo applicare il Pumping Lemma. Dato che $|xy| \leq p$, xy è un prefisso di w che contiene solo caratteri 0 (eventualmente tutti 0 se $|xy| < p$, o includendo l'unico 1 se $|xy| = p$).

Scegliendo $i = 2$, otteniamo $xy^2z = 0^{p+j}10^p$ o $xy^2z = 0^p10^p$ a seconda di cosa contiene y. Se y contiene solo 0, la stringa risultante non è palindroma (poiché ci sono più 0 all'inizio che alla fine). Se y contiene il 1, la stringa risultante non è ben formata per il nostro ragionamento.

In entrambi i casi, contraddiamo l'ipotesi che P sia regolare, quindi P non è regolare. Di conseguenza, il complemento di P, cioè $\{w \in \{0,1\}^* \mid w \text{ non è palindroma}\}$, non è regolare.

(d) Dimostriamo che $\{wtw \mid w, t \in \{0,1\}^+\}$ non è regolare.

Utilizzeremo il Pumping Lemma. Supponiamo per assurdo che il linguaggio sia regolare. Allora esiste una costante $p > 0$ tale che ogni stringa s nel linguaggio con $|s| \geq p$ può essere scritta come $s = xyz$ con $|xy| \leq p$, $|y| > 0$, e per ogni $i \geq 0$, $xy^iz \in L$.

Consideriamo la stringa $s = 0^p10^p$ (dove $w = 0^p$ e $t = 1$). Poiché $|s| = 2p + 1 > p$, possiamo applicare il Pumping Lemma. Dato che $|xy| \leq p$, xy è un prefisso di s contenente solo caratteri 0. Quindi $y = 0^j$ per qualche $j > 0$.

Scegliendo $i = 2$, otteniamo $xy^2z = 0^{p+j}10^p$. Questa stringa non può essere scritta nella forma wtw con $w, t \in \{0,1\}^+$, perché per qualsiasi scelta di w e t, o l'ultima parte

non corrisponde a w, o la prima parte non corrisponde a w.

Questo contraddice l'ipotesi che il linguaggio sia regolare, quindi $\{wtw \mid w, t \in \{0,1\}^+\}$ non è regolare.

Esercizio 10. Per ogni linguaggio A, sia $SUFFIX(A) = \{v \mid uv \in A \text{ per qualche stringa } u\}$. Mostrare che la classe dei linguaggi context-free è chiusa rispetto all'operazione di $SUFFIX$.

Soluzione. Per dimostrare che la classe dei linguaggi context-free è chiusa rispetto all'operazione $SUFFIX$, mostreremo che se A è un linguaggio context-free, allora $SUFFIX(A)$ è anch'esso un linguaggio context-free.

Sia A un linguaggio context-free. Allora esiste una grammatica context-free $G = (V, \Sigma, R, S)$ tale che $L(G) = A$, dove:

- V è l'insieme delle variabili (o non-terminali)
- Σ è l'alfabeto (o terminali)
- R è l'insieme delle regole di produzione
- S è il simbolo di partenza (o assioma)

Costruiamo una nuova grammatica context-free $G' = (V', \Sigma, R', S')$ tale che $L(G') = SUFFIX(A)$.

G' è definita come segue:

- $V' = V \cup \{S'\}$, dove S' è un nuovo simbolo non terminale
- R' include tutte le regole di R, più le seguenti nuove regole:
  - $S' \to S$
  - Per ogni produzione $X \to \alpha$ in R e per ogni scomposizione $\alpha = \beta\gamma$ con $\beta \in (V \cup \Sigma)^+$ e $\gamma \in (V \cup \Sigma)^*$, aggiungiamo la produzione $S' \to \gamma$

La grammatica G' funziona generando stringhe in due modi:

- Usando la regola $S' \to S$ seguita dalle regole di G, G' può generare tutte le stringhe in A.
- Usando le regole $S' \to \gamma$, G' può "saltare" direttamente a un punto intermedio nella derivazione di qualsiasi stringa in A, in modo da generare qualsiasi suffisso di una stringa in A.

Tuttavia, questa costruzione potrebbe introdurre un numero infinito di regole se G contiene cicli. Una costruzione alternativa utilizza una Push-Down Automaton (PDA).

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA che accetta A. Costruiamo un nuovo PDA $M' = (Q', \Sigma, \Gamma, \delta', q_0', Z_0, F')$ che accetta $SUFFIX(A)$:

- $Q' = Q \cup \{q_0'\}$, dove q_0' è un nuovo stato iniziale
- $F' = F$
- δ' include tutte le transizioni di δ, più le seguenti nuove transizioni:

- $\delta'(q_0', \varepsilon, Z_0) = \{(q_0, Z_0)\}$
- Per ogni stato $q \in Q$, aggiungiamo la transizione $\delta'(q_0', \varepsilon, Z_0) = \{(q, Z_0)\}$

Questo PDA può iniziare la computazione da qualsiasi stato del PDA originale, accettando così qualsiasi suffisso di una stringa accettata da M.

Poiché i linguaggi accettati dai PDA sono esattamente i linguaggi context-free, e M' accetta $SUFFIX(A)$, concludiamo che $SUFFIX(A)$ è un linguaggio context-free quando A è un linguaggio context-free.

Esercizio 11. Dimostrare che i seguenti linguaggi sono context-free. Salvo quando specificato diversamente, l'alfabeto è $\Sigma = \{0, 1\}$.

(a) $\{w \mid w$ contiene almeno tre simboli uguali a $1\}$

(b) $\{w \mid$ la lunghezza di w è dispari$\}$

(c) $\{w \mid w$ inizia e termina con lo stesso simbolo$\}$

(d) $\{w \mid$ la lunghezza di w è dispari e il suo simbolo centrale è $0\}$

(e) $\{w \mid w = w^R,$ cioè w è palindroma$\}$

(f) $\{w \mid w$ contiene un numero maggiore di 0 che di $1\}$

(g) Il complemento di $\{0^n 1^n \mid n \geq 0\}$

(h) Sull'alfabeto $\Sigma = \{0, 1, \#\}$, $\{w\#x \mid w^R$ è una sottostringa di x e $w, x \in \{0, 1\}^*\}$

(i) $\{x\#y \mid x, y \in \{0, 1\}^*$ e $x \neq y\}$

(j) $\{xy \mid x, y \in \{0, 1\}^*$ e $|x| = |y|$ ma $x \neq y\}$

(k) $\{a^i b^j \mid i \neq j$ e $2i \neq j\}$

Soluzione. **(a)** $\{w \mid w$ contiene almeno tre simboli uguali a $1\}$

Questo linguaggio è regolare, e quindi anche context-free. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to A1B1C1D \mid A1B1D1C \mid A1C1B1D$$

$$A \to 0A \mid 1A \mid \varepsilon$$
$$B \to 0B \mid 1B \mid \varepsilon$$
$$C \to 0C \mid 1C \mid \varepsilon$$
$$D \to 0D \mid 1D \mid \varepsilon$$

Questa grammatica genera tutte le stringhe che contengono almeno tre simboli 1.

(b) $\{w \mid$ la lunghezza di w è dispari$\}$

Questo linguaggio è regolare, e quindi anche context-free. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to 0 \mid 1 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$$

Questa grammatica genera tutte le stringhe di lunghezza dispari.

(c) $\{w \mid w$ inizia e termina con lo stesso simbolo$\}$

Questo linguaggio è regolare, e quindi anche context-free. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to 0A0 \mid 1B1 \mid 0 \mid 1$$

$$A \to 0A \mid 1A \mid \varepsilon$$

$$B \to 0B \mid 1B \mid \varepsilon$$

Questa grammatica genera tutte le stringhe che iniziano e terminano con lo stesso simbolo.

(d) $\{w \mid$ la lunghezza di w è dispari e il suo simbolo centrale è $0\}$

Questo linguaggio è regolare, e quindi anche context-free. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to A0A$$

$$A \to 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid \varepsilon$$

Questa grammatica genera tutte le stringhe di lunghezza dispari con il simbolo centrale 0.

(e) $\{w \mid w = w^R,$ cioè w è palindroma$\}$

Questo linguaggio è context-free ma non regolare. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

Questa grammatica genera tutte le stringhe palindrome.

(f) $\{w \mid w$ contiene un numero maggiore di 0 che di $1\}$

Questo linguaggio è context-free. Possiamo costruire una grammatica context-free G che tiene traccia della differenza tra il numero di 0 e il numero di 1:

$$S \to A \mid 0S \mid S0$$

$$A \to 0A1 \mid 1A0 \mid \varepsilon$$

Questa grammatica genera tutte le stringhe che contengono più 0 che 1.

(g) Il complemento di $\{0^n1^n \mid n \geq 0\}$

Il complemento di $\{0^n1^n \mid n \geq 0\}$ è il linguaggio di tutte le stringhe che non sono nella forma 0^n1^n. Questo può essere descritto come l'unione dei seguenti linguaggi:

- Stringhe che contengono almeno un 1 seguito da almeno un 0
- Stringhe che contengono un numero diverso di 0 e 1

Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to A \mid B$$

$$A \to C1D0E$$

$$B \to F0G \mid H1I$$
$$C \to 0C \mid 1C \mid \varepsilon$$
$$D \to 0D \mid 1D \mid \varepsilon$$
$$E \to 0E \mid 1E \mid \varepsilon$$
$$F \to 0F \mid 1F \mid \varepsilon$$
$$G \to 0G \mid 1G \mid \varepsilon$$
$$H \to 0H \mid 1H \mid \varepsilon$$
$$I \to 0I \mid 1I \mid \varepsilon$$

Questa grammatica genera tutte le stringhe che non sono nella forma 0^n1^n.

(h) Sull'alfabeto $\Sigma = \{0, 1, \#\}$, $\{w\#x \mid w^R$ è una sottostringa di x e $w, x \in \{0, 1\}^*\}$

Questo linguaggio è context-free. Possiamo costruire una grammatica context-free G con le seguenti produzioni:

$$S \to A\#B$$
$$A \to 0A \mid 1A \mid \varepsilon$$
$$B \to BD \mid D$$
$$D \to CDC$$
$$C \to 0C \mid 1C \mid \varepsilon$$

Dove B genera stringhe che contengono w^R come sottostringa.

(i) $\{x\#y \mid x, y \in \{0, 1\}^*$ e $x \neq y\}$

Questo linguaggio è context-free. Possiamo costruire una grammatica context-free G che genera tutte le coppie di stringhe che differiscono in almeno una posizione:

$$S \to A\#B \mid A0C\#A1D \mid A1C\#A0D$$
$$A \to 0A \mid 1A \mid \varepsilon$$
$$B \to 0B \mid 1B \mid \varepsilon$$
$$C \to 0C \mid 1C \mid \varepsilon$$
$$D \to 0D \mid 1D \mid \varepsilon$$

Questa grammatica genera tutte le stringhe della forma $x\#y$ dove $x \neq y$.

(j) $\{xy \mid x, y \in \{0, 1\}^*$ e $|x| = |y|$ ma $x \neq y\}$

Questo linguaggio è context-free. Possiamo costruire una grammatica context-free G che genera coppie di stringhe di uguale lunghezza che differiscono in almeno una posizione:

$$S \to 0A0 \mid 0A1 \mid 1A0 \mid 1A1$$
$$A \to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 0D1 \mid 1D0$$
$$B \to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 0D1 \mid 1D0 \mid \varepsilon$$
$$D \to 0D0 \mid 0D1 \mid 1D0 \mid 1D1 \mid \varepsilon$$

Questa grammatica genera tutte le stringhe della forma xy dove $|x| = |y|$ e $x \neq y$.

(k) $\{a^ib^j \mid i \neq j$ e $2i \neq j\}$

Questo linguaggio è context-free. Possiamo esprimere il linguaggio come l'unione di due linguaggi:

- $L_1 = \{a^i b^j \mid i > j\}$
- $L_2 = \{a^i b^j \mid i < j \text{ e } 2i \neq j\}$

Per L_1, possiamo costruire una grammatica context-free:

$$S_1 \to aS_1 \mid aT$$

$$T \to aT \mid aTb \mid \varepsilon$$

Per L_2, possiamo costruire una grammatica context-free:

$$S_2 \to aS_2b \mid aS_2bb \mid abbb \mid bb$$

La grammatica per l'intero linguaggio è:

$$S \to S_1 \mid S_2$$

Questa grammatica genera tutte le stringhe della forma $a^i b^j$ dove $i \neq j$ e $2i \neq j$.

Esercizio 12. Se A e B sono linguaggi, definiamo $A \circ B = \{xy \mid x \in A, y \in B \text{ e } |x| = |y|\}$. Mostrare che se A e B sono linguaggi regolari, allora $A \circ B$ è un linguaggio context-free.

Soluzione. Per dimostrare che $A \circ B$ è un linguaggio context-free quando A e B sono linguaggi regolari, costruiremo una grammatica context-free che genera $A \circ B$.

Siano $M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ e $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ due DFA che riconoscono rispettivamente A e B.

Costruiamo una grammatica context-free $G = (V, \Sigma, R, S)$ che genera $A \circ B$:

- $V = \{S\} \cup \{[p, q] \mid p \in Q_A, q \in Q_B\}$
- R contiene le seguenti produzioni:
  - $S \to [q_{0A}, q_{0B}]$
  - Per ogni $p \in Q_A, q \in Q_B, a \in \Sigma, b \in \Sigma$: $[p, q] \to a[p', q']b$ dove $p' = \delta_A(p, a)$ e $q' = \delta_B(q, b)$
  - Per ogni $p \in F_A, q \in F_B$: $[p, q] \to \varepsilon$

Questa grammatica funziona nel modo seguente:

- Iniziamo con il simbolo non terminale $[q_{0A}, q_{0B}]$, che rappresenta gli stati iniziali dei due DFA.
- Ad ogni passo, la grammatica genera simultaneamente un simbolo per la prima stringa (da A) e un simbolo per la seconda stringa (da B), aggiornando gli stati dei due DFA.
- La derivazione termina quando entrambi i DFA raggiungono uno stato di accettazione.

Questa costruzione garantisce che:

- Le stringhe generate hanno la forma xy dove x è la prima metà e y è la seconda metà.

- Le stringhe x sono accettate dal DFA M_A e quindi appartengono ad A.
- Le stringhe y sono accettate dal DFA M_B e quindi appartengono a B.
- Per costruzione, $|x| = |y|$.

Poiché abbiamo costruito una grammatica context-free che genera esattamente $A \circ B$, concludiamo che $A \circ B$ è un linguaggio context-free quando A e B sono linguaggi regolari.

Esercizio 13. Dimostrare che se G è una CFG in forma normale di Chomsky, allora per ogni stringa $w \in L(G)$ di lunghezza $n \geq 1$, ogni derivazione di w richiede esattamente $2n - 1$ passi.

Soluzione. Ricordiamo che una grammatica context-free $G = (V, \Sigma, R, S)$ è in forma normale di Chomsky se tutte le sue produzioni sono della forma:

- $A \to BC$ dove $A, B, C \in V$ (regole che generano due non-terminali)
- $A \to a$ dove $A \in V$ e $a \in \Sigma$ (regole che generano un terminale)
- $S \to \varepsilon$ (solo se $\varepsilon \in L(G)$, e S non appare nel lato destro di nessuna regola)

Per dimostrare che ogni derivazione di una stringa $w \in L(G)$ di lunghezza $n \geq 1$ richiede esattamente $2n - 1$ passi, useremo l'induzione sulla lunghezza n di w.

Base dell'induzione: $n = 1$, cioè w è un singolo carattere. Se $w = a$ è un singolo carattere, l'unica derivazione possibile è: $S \Rightarrow a$ Questa derivazione richiede 1 passo, che è esattamente $2n - 1 = 2 \cdot 1 - 1 = 1$.

Ipotesi induttiva: Supponiamo che il risultato sia vero per tutte le stringhe di lunghezza k con $1 \leq k < n$.

Passo induttivo: Dimostriamo che il risultato è vero per una stringa w di lunghezza $n > 1$.

Poiché G è in forma normale di Chomsky, il primo passo nella derivazione di w deve essere della forma $S \Rightarrow AB$ per qualche $A, B \in V$.

Questa derivazione continua con la generazione di due sottostringe u e v tali che $w = uv$, dove u è derivata da A e v è derivata da B. Siano $|u| = i$ e $|v| = j$. Poiché $w = uv$, abbiamo $|w| = |u| + |v| = i + j = n$.

Per l'ipotesi induttiva, la derivazione di u da A richiede $2i - 1$ passi, e la derivazione di v da B richiede $2j - 1$ passi.

Il numero totale di passi nella derivazione di w è quindi:

$$\begin{aligned} 1 + (2i - 1) + (2j - 1) &= 1 + 2i - 1 + 2j - 1 && (1)\\ &= 2i + 2j - 1 && (2)\\ &= 2(i + j) - 1 && (3)\\ &= 2n - 1 && (4) \end{aligned}$$

Questo dimostra che ogni derivazione di una stringa $w \in L(G)$ di lunghezza $n \geq 1$ richiede esattamente $2n - 1$ passi.

Una spiegazione intuitiva è che per generare una stringa di lunghezza n, dobbiamo:

- Utilizzare n regole del tipo $A \to a$ per generare gli n terminali.

- Utilizzare $n-1$ regole del tipo $A \to BC$ per combinare questi n terminali. Questo perché abbiamo bisogno di $n-1$ "giunzioni" per collegare n elementi in sequenza.

Il numero totale di passi è quindi $n + (n-1) = 2n - 1$.