4. Sia L un linguaggio regolare su un alfabeto Σ e dimostrate che il seguente linguaggio è regolare:

$$\mathit{suffixes}(L) = \{y \mid xy \in L \text{ per qualche stringa } x \in \Sigma^*\}$$

Intuitivamente, $\mathit{suffixes}(L)$ è il linguaggio di tutti i suffissi delle parole che stanno in L.

Per dimostrare che $\mathit{suffixes}(L)$ è regolare basta mostrare come costruire un automa a stati finiti che riconosce $\mathit{suffixes}(L)$ a partire dall'automa a stati finiti che riconosce L.

Sia quindi $A = (Q, \Sigma, q_0, \delta, F)$ un automa a stati finiti che riconosce il linguaggio L. Costruiamo un ε-NFA $B = (Q \cup \{q_0'\}, \Sigma, q_0', \delta_B, F)$ che ha uno stato in più di A. Chiamiamo q_0' questo nuovo stato e gli assegnamo il ruolo di stato iniziale di B. La funzione di transizione del nuovo automa aggiunge una ε-transizione dal nuovo stato iniziale q_0' verso ogni stato di Q che è raggiungibile dal vecchio stato iniziale. Il resto delle transizioni rimane invariato. Gli stati finali sono gli stessi di A.

4. Sia L un linguaggio regolare su un alfabeto Σ. Dimostrare che anche il seguente linguaggio è regolare:

$$\mathit{init}(L) = \{w \in \Sigma^* : \text{ esiste } x \in \Sigma^* \text{ tale che } wx \in L\}$$

Soluzione: Per dimostrare che $\mathit{init}(L)$ è regolare vediamo come è possibile costruire un automa a stati finiti che riconosce $\mathit{init}(L)$ a partire dall'automa a stati finiti che riconosce L.

Sia quindi $A = (Q, \Sigma, q_0, \delta, F)$ un DFA che riconosce il linguaggio L. Costruiamo il DFA $B = (Q, \Sigma, q_0, \delta, G)$ che ha gli stessi stati, le stesse transizioni e lo stesso stato iniziale di A. Definiamo l'insieme G degli stati finali del nuovo automa come $G = \{q \in Q : \text{esiste una sequenza di transizioni da } q \text{ ad uno stato finale } f \in F\}$, ossia come tutti gli stati a partire dai quali possiamo raggiungere uno stato finale di A. Dobbiamo dimostrare che $L(B) = \mathit{init}(L)$.

- Sia $w \in \mathit{init}(L)$: allora per la definizione deve esistere $x \in \Sigma^*$ tale che $wx \in L$. Poiché A è un automa deterministico, esiste una sola sequenza di transizioni che parte da q_0 e accetta la parola wx in A. Possiamo spezzare questa sequenza in due parti: una prima sequenza che parte da q_0, legge w e arriva in uno stato intermedio q, e una seconda sequenza che parte da q, legge x e arriva ad uno stato finale $f \in F$. Ma allora lo stato intermedio q deve appartenere agli stati finali G di B! Quindi la parola w viene accettata dall'automa B.
- Prendiamo ora una parola $w \in L(B)$. Poiché B è un automa deterministico, esiste una sola sequenza di transizioni che parte da q_0 e accetta la parola w arrivando ad uno stato finale $q \in G$. Per la definizione di G, esiste una sequenza di transizioni che porta da q ad uno stato finale f di A. Quindi deve esistere una parola x i cui simboli etichettano le transizioni della sequenza da q a f. Ma allora possiamo creare una sequenza di transizioni da q_0 a f che riconosce la parola wx, che quindi appartiene a L. Dalla definizione di $\mathit{init}(L)$ segue che $w \in \mathit{init}(L)$.

1. Un *all-ε-NFA* M è una quintupla $(Q, \Sigma, \delta, q_0, F)$ che accetta una parola $w \in \Sigma^*$ se *ogni* possibile stato in cui M si può trovare dopo aver consumato l'input w è uno stato in F. Nota che un normale ε-NFA accetta la stringa se *qualcuno* di questi stati è uno stato finale. Prova che gli all-ε-NFA riconoscono esattamente la classe dei linguaggi regolari.

- Per dimostrare che ogni linguaggio regolare è riconosciuto da un all-ε-NFA si parte dal fatto che ogni linguaggio regolare ha un DFA che lo riconosce. È facile vedere che ogni DFA è anche un all-ε-NFA, che non ha ε-transizioni e dove $\delta(q, a) = \{q'\}$ per ogni stato $q \in Q$ e simbolo dell'alfabeto $a \in \Sigma$. In un DFA c'è una sola computazione possibile, quindi lo stato in cui si trova l'automa dopo aver consumato l'input è unicamente determinato: se questo stato è finale il DFA accetta, altrimenti rifiuta. Questo è coerente con la condizione di accettazione degli all-ε-NFA quando c'è un solo possibile stato dove si può trovare l'automa dopo aver consumato l'input.

  Soluzione alternativa: in alternativa all'algoritmo si può dare la definizione del DFA $D = (Q_D, \Sigma, S_0, \delta_D, F_D)$ equivalente all'all-ε-NFA $N = (Q_N, \Sigma, q_0, \delta_N, F_N)$ specificando le componenti del DFA:
  - l'insieme degli stati è l'insieme delle parti di Q_N: $Q_D = \{S \mid S \subseteq Q_N\}$;
  - lo stato iniziale è la ε-chiusura di q_0: $S_0 = \textsc{Eclose}(q_0)$;
  - la funzione di transizione "simula" le transzioni di N:

$$\delta_D(S, a) = \textsc{Eclose}(\bigcup_{p \in S} \delta_N(p, a))$$

  - l'insieme degli stati finali è l'insieme delle parti di F_N: $F_D = \{S \mid S \subseteq F_N\}$.

  Anche questa definizione è analoga a quella che trasforma un ε-NFA in un DFA, con la sola differenza della definizione degli stati finali.

1. Considera la funzione ricorsiva

$$faro(x,z) = \begin{cases} z & \text{se } x = \varepsilon \\ a.faro(z,y) & \text{se } x = ay \end{cases}$$

Per esempio, $faro(00110, 0101) = 000110110$. Dimostra che se L e M sono linguaggi regolari definiti sullo stesso alfabeto Σ, allora anche il linguaggio $faro(L,M) = \{faro(x,z) \mid x \in L \text{ e } z \in M\}$ è regolare.

1. Supponiamo che L ed M siano due linguaggi regolari, e mostriamo che anche $faro(L,M)$ è regolare. Poiché L ed M sono regolari, sappiamo che esiste un DFA A_L che riconosce L ed un DFA A_M che riconosce M. Per dimostrare che $faro(L,M)$ è regolare esibiamo un automa a stati finiti A che riconosce $faro(L,M)$.

La funzione ricorsiva $faro(x,z)$ rimescola le parole x e z in questo modo:

- se $|x| = |z|$, allora il risultato è una parola che alterna i simboli di x nelle posizioni dispari coni i simboli di z nelle posizioni pari: $faro(x,z) = x_1z_1x_2z_2 \dots x_nz_n$;
- se x è più corta di z, allora $faro(x,z)$ alterna simboli di x con simboli di z finché possibile, e poi continua con la parte rimanente di z;
- se z è più corta di x, allora $faro(x,z)$ alterna simboli di x con simboli di z finché possibile, e poi continua con la parte rimanente di x.

L'automa che accetta $faro(L,M)$ procede alternando transizioni di A_L con transizioni di A_M per ottenere l'alternanza dei simboli della parola $x \in L$ con quelli della parola $z \in M$. Per poter simulare l'alternanza l'automa deve memorizzare lo stato corrente di A_L, lo stato corrente di A_M e quale automa simulare alla prossima transizione. Gli stati A saranno quindi delle triple (r_L, r_M, t) dove r_L è uno stato di A_L, r_M è uno stato di A_M e $t \in \{L, M\}$ un valore che rappresenta il turno della simulazione. Le transizioni sono definite come segue:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, M)$ se $\delta_L(r_L, a) = s_L$: quando è il turno di A_L si simula la transizione di A_L, si mantiene inalterato lo stato di A_M e si cambia il turno a M per la prossima transizione;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, L)$ se $\delta_M(r_M, a) = s_M$: quando è il turno di A_M si simula la transizione di A_M, si mantiene inalterato lo stato di A_L e si cambia il turno a L per la prossima transizione.

Dobbiamo ora stabilire quali sono gli stati finali di A. Quando la simulazione raggiunge uno stato finale di A_L possono succedere due cose: si continua ad alternare transizioni di A_L con transizioni di A_M, oppure "scommettere" che abbiamo terminato di consumare i simboli della parola x, e continuare con la parte rimanente di z fino a raggiungere uno stato finale di A_M. Viceversa, quando la simulazione raggiunge uno stato finale di A_M, sceglie se continuare con l'alternanza oppure con la parte rimanente di x. Useremo il nondeterminismo per rappresentare queste scelte: A sarà un NFA anche se A_L e A_M sono dei DFA. Oltre al nondeterminismo dobbiamo aggiungere altri due tipi di turno: L_{suff} e M_{suff} per rappresentare le computazioni sulla parte rimanente di parola in L o sulla parte rimanente di parola in M. Quindi gli stati di A saranno delle triple (r_L, r_M, t) con $t \in \{L, M, L_{suff}, M_{suff}\}$, e dobbiamo aggiungere le seguenti transizioni all'automa:

- $(r_L, r_M, L) \xrightarrow{a} (s_L, r_M, L_{suff})$ se $\delta_L(r_L, a) = s_L$ e r_M è uno stato finale di A_M;
- $(r_L, r_M, M) \xrightarrow{a} (r_L, s_M, M_{suff})$ se $\delta_M(r_M, a) = s_M$ e r_L è uno stato finale di A_L;
- $(r_L, r_M, L_{suff}) \xrightarrow{a} (s_L, r_M, L_{suff})$ se $\delta_L(r_L, a) = s_L$;
- $(r_L, r_M, M_{suff}) \xrightarrow{a} (r_L, s_M, M_{suff})$ se $\delta_M(r_M, a) = s_M$.

Si può notare che se il turno diventa uguale a L_{suff} allora A prosegue simulando solamente le transizioni di A_L, senza cambiare lo stato r_M. Viceversa, quando il turno diventa uguale a M_{suff} A prosegue simulando solamente le transizioni di A_M, senza cambiare lo stato r_L.

Gli stati finali di A sono tutte le triple (r_L, r_M, t) tali che r_L è uno stato finale di A_L e r_M è uno stato finale di M. Lo stato iniziale è la tripla (q_L^0, q_M^0, L).

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

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

*dove $x \sqcap y$ rappresenta l'*and bit a bit *di x e y. Per esempio, $0011 \sqcap 0101 = 0001$.*

Poiché L e M sono regolari, sappiamo che esiste un DFA $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ che riconosce L e un DFA $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ che riconosce M.

Costruiamo un NFA A che riconosce il linguaggio $L \sqcap M$:

- L'insieme degli stati è $Q = Q_L \times Q_M$, che contiene tutte le coppie composte da uno stato di A_L e uno stato di A_M.
- L'alfabeto è lo stesso di A_L e di A_M, $\Sigma = \{0,1\}$.
- La funzione di transizione δ è definita come segue:

$$\delta((r_L, r_M), 0) = \{(\delta_L(r_L, 0), \delta_M(r_M, 0)), (\delta_L(r_L, 1), \delta_M(r_M, 0)), (\delta_L(r_L, 0), \delta_M(r_M, 1))\}$$
$$\delta((r_L, r_M), 1) = \{(\delta_L(r_L, 1), \delta_M(r_M, 1))\}$$

  La funzione di transizione implementa le regole dell'and tra due bit: l'and di due 1 è 1, mentre è 0 se entrambi i bit sono 0 o se un bit è 0 e l'altro è 1.
- Lo stato iniziale è (q_L, q_M).
- Gli stati finali sono $F = F_L \times F_M$, ossia tutte le coppie di stati finali dei due automi.

1. Considera la seguente funzione da $\{0,1\}^*$ a $\{0,1\}^*$:

$$\text{stutter}(w) = \begin{cases} \varepsilon & \text{se } w = \varepsilon \\ aa.\text{stutter}(x) & \text{se } w = ax \text{ per qualche simbolo } a \text{ e parola } x \end{cases}$$

Dimostra che se L è un linguaggio regolare sull'alfabeto $\{0,1\}$, allora anche il seguente linguaggio è regolare:

$$\text{stutter}(L) = \{\text{stutter}(w) \mid w \in L\}.$$

Per dimostrare che stutter(L) è un linguaggio regolare se L è regolare, possiamo procedere nel seguente modo:

1. Dato che L è un linguaggio regolare, esiste un DFA A = (Q, Σ, δ, q0, F) che lo riconosce.
2. Costruiamo un nuovo DFA A' = (Q', Σ, δ', q0', F') che riconosce stutter(L): Q' = Q × {0,1} (gli stati di A' sono coppie formate da uno stato di A e un bit) Σ = {0,1} q0' = (q0, 0) F' = {(q,1) | q ∈ F} La funzione di transizione δ' è definita come: δ'((q,0), a) = (δ(q,a), 1) δ'((q,1), a) = (δ(q,a), 0)
3. L'idea è che il bit aggiuntivo tiene traccia se stiamo leggendo il primo o il secondo simbolo di una coppia "stutterata".
4. Quando il bit è 0, stiamo leggendo il primo simbolo di una coppia, quindi passiamo allo stato corrispondente in A e settiamo il bit a 1.
5. Quando il bit è 1, stiamo leggendo il secondo simbolo di una coppia, quindi passiamo allo stato corrispondente in A e resettiamo il bit a 0.
6. L'automa accetta solo quando si trova in uno stato finale di A e ha appena letto il secondo simbolo di una coppia (bit = 1).

Questo DFA A' riconosce esattamente le parole nella forma aa.stutter(x) dove ax ∈ L, che corrisponde alla definizione di stutter(L).

Poiché abbiamo costruito un DFA che riconosce stutter(L), abbiamo dimostrato che stutter(L) è un linguaggio regolare.

1. **(12 punti)** Se L è un linguaggio e a un simbolo, allora a/L, la *derivata* di L e a, è l'insieme delle stringhe

$$a/L = \{w \mid aw \in L\}.$$

Per esempio, se $L = \{a, aab, baa\}$, allora $a/L = \{\varepsilon, ab\}$. Dimostra che se L è regolare allora anche a/L è regolare.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Data una parola $aw \in L$ dove $w = w_1 \dots w_n$, la computazione di A su aw è una sequenza di stati $r_0 r_1 \dots r_{n+1}$ tali che:

- $r_0 = q_0$;
- $\delta(r_0, a) = r_1$;
- $\delta(r_i, w_i) = r_{i+1}$ per ogni $i = 1, \dots, n$;
- $r_{n+1} \in F$.

Siccome A è un DFA, lo stato r_1 in cui l'automa si trova è unicamente determinato, e possiamo costruire un automa A' che accetta il linguaggio a/L cambiando lo stato iniziale di A in r_1. Formalmente, $A' = (Q, \Sigma, \delta, r_1, F)$ dove Q, Σ, δ e F sono gli stessi di A e lo stato iniziale è $r_1 = \delta(q_0, a)$. In questo modo abbiamo che per ogni $aw \in L$ la sequenza di stati $r_1 \dots r_{n+1}$ descritta sopra è una computazione di A' che accetta w, ed abbiamo dimostrato che se $aw \in L$ allora $w \in L(A')$.

Viceversa, se $w \in L(A')$ allora esiste una computazione $s_1 \dots s_{n+1}$ di A' tale che:

- $s_1 = r_1$;
- $\delta(s_i, w_i) = s_{i+1}$ per ogni $i = 1, \dots, n$;
- $s_{n+1} \in F$.

Di conseguenza, la sequenza di stati $q_0 s_1 \dots s_{n+1}$ è una computazione di A su aw, ed abbiamo dimostrato che se $w \in L(A')$ allora $aw \in L$. Quindi possiamo concludere che il linguaggio di A' è precisamente a/L, come richiesto.

3. **(12 punti)** Una CFG è detta *lineare a destra* se il corpo di ogni regola ha al massimo una variabile, e la variabile si trova all'estremità di destra. In altre parole, tutte le regole di una grammatica lineare a destra sono nella forma $A \to wB$ o $A \to w$, dove A e B sono variabili e w è una stringa di zero o più simboli terminali.

Dimostra che ogni grammatica lineare a destra genera un linguaggio regolare. *Suggerimento:* costruisci un ε-NFA che simula le derivazioni della grammatica.

Soluzione: Data una grammatica lineare a destra $G = (V, \Sigma, R, S)$, possiamo notare che le derivazioni di G hanno una struttura particolare. Siccome le regole sono nella forma $A \to wB$ o $A \to w$, dove A e B sono variabili e w è una stringa di zero o più simboli terminali, ogni derivazione di una parola della grammatica sarà formata da n applicazioni di una regola del tipo $A \to wB$ seguita da un'applicazione finale di una regola $A \to w$. A partire da questa osservazione possiamo costruire un ε-NFA $A = (Q, \Sigma, \delta, q_0, F)$ che simula le derivazioni della grammatica. Per rendere più chiara la costruzione usiamo una notazione abbreviata per la funzione di transizione, che ci fornisce un modo per far consumare un'intera stringa all'automa in un singolo passo. Possiamo simulare queste transizioni estese introducendo stati aggiuntivi per consumare la stringa un simbolo alla volta. Siano p e q stati dell'ε-NFA e sia $w = w_1 \dots w_n$ una stringa sull'alfabeto Σ. Per fare in modo che l'automa vada da p a q consumando l'intera stringa w introducendo nuovi stati intermedi $r_1 \dots r_{n-1}$ e definendo la funzione di transizione come segue:

$$\begin{aligned} \delta(p, w_1) &\text{ contiene } r_1, \\ \delta(r_1, w_2) &= \{r_2\}, \\ &\dots \\ \delta(r_{n-1}, w_n) &= \{q\}. \end{aligned}$$

Useremo la notazione $q \in \delta(p, w)$ per denotare quando l'automa può andare da p a q consumando la stringa di input w. Gli stati dell'automa sono $Q = V \cup \{q_f\} \cup E$ dove V è l'insieme delle variabili della grammatica, q_f è l'unico stato finale dell'automa e E è l'insieme di stati necessari per realizzare le transizioni estese appena descritte. Lo stato iniziale è S, variabile iniziale della grammatica. La funzione di transizione è definita come segue:

- $B \in \delta(A, w)$ per ogni regola $A \to wB$ della grammatica;
- $q_f \in \delta(A, w)$ per ogni regola $A \to w$ della grammatica.

L'automa A che abbiamo costruito è in grado di riconoscere lo stesso linguaggio della grammatica G perché simula le derivazioni di G nel modo descritto di seguito. L'automa inizia la computazione dallo stato che corrisponde alla variabile iniziale di G, e ripete i seguenti passi:

(a) se lo stato corrente corrente corrisponde alla variabile A, sceglie non deterministicamente una delle regole per A;
(b) se la regola scelta è $A \to wB$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato B, altrimenti la computazione si blocca su questo ramo;
(c) se la regola scelta è $A \to w$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato finale q_f e accetta se ha consumato tutto l'input. La computazione si blocca su questo ramo se l'automa non riesce a consumare w o se non ha consumato tutto l'input quando arriva in q_f.

1. **(12 punti)** Se L è un linguaggio e a un simbolo, allora L/a, il *quoziente* di L e a, è l'insieme delle stringhe

$$L/a = \{w \mid wa \in L\}.$$

Per esempio, se $L = \{a, aab, baa\}$, allora $L/a = \{\varepsilon, ba\}$. Dimostra che se L è regolare allora anche L/a è regolare.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Data una parola $wa \in L$ dove $w = w_1 \dots w_n$, la computazione di A su aw è una sequenza di stati $r_0 r_1 \dots r_{n+1}$ tali che:

- $r_0 = q_0$;
- $\delta(r_{i-1}, w_i) = r_i$ per ogni $i = 1, \dots, n$;
- $\delta(r_n, a) = r_{n+1}$;
- $r_{n+1} \in F$.

Data questa osservazione possiamo costruire un automa A' che accetta il linguaggio L/a cambiando gli stati finali di A. Formalmente, $A' = (Q, \Sigma, \delta, q_0, F')$ dove Q, Σ, δ e q_0 sono gli stessi di A e l'insieme degli stati finali contiene tutti gli stati che raggiungono uno stato finale di A dopo aver consumato a:

$$F' = \{q \mid \delta(q, a) \in F\}.$$

In questo modo abbiamo che per ogni $wa \in L$ la sequenza di stati $r_0 \dots r_n$ descritta sopra è una computazione di A' che accetta w (perché r_n diventa uno stato finale di A'), ed abbiamo dimostrato che se $wa \in L$ allora $w \in L(A')$. Viceversa, se $w \in L(A')$ allora esiste una computazione $s_0 \dots s_n$ di A' tale che:

- $s_0 = q_0$;
- $\delta(s_{i-1}, w_i) = s_i$ per ogni $i = 1, \dots, n$;
- $s_n \in F'$.

Di conseguenza, $s_{n+1} = \delta(s_n, a) \in F$ e la sequenza di stati $s_1 \dots s_{n+1}$ è una computazione di A su wa, ed abbiamo dimostrato che se $w \in L(A')$ allora $wa \in L$. Quindi possiamo concludere che il linguaggio di A' è precisamente L/a, come richiesto.

1. **(12 punti)** Se L è un linguaggio sull'alfabeto $\{0, 1\}$, la *rotazione a destra* di L è l'insieme delle stringhe

$$\mathrm{ROR}(L) = \{aw \mid wa \in L, w \in \{0,1\}^*, a \in \{0,1\}\}.$$

Per esempio, se $L = \{0, 001, 10010\}$, allora $\mathrm{ROR}(L) = \{0, 100, 01001\}$. Dimostra che se L è regolare allora anche $\mathrm{ROR}(L)$ è regolare.

Per dimostrare che ROR(L) è regolare se L è regolare, costruiamo un automa a stati finiti deterministico (DFA) che riconosce ROR(L) a partire dal DFA che riconosce L.

Sia A = (Q, Σ, δ, q0, F) il DFA che riconosce L. Costruiamo A' = (Q', Σ, δ', q0', F') che riconosce ROR(L) come segue:

Q' = Q × Σ ∪ {q0'} q0' è un nuovo stato iniziale F' = {(q, a) | δ(q, a) ∈ F} δ' è definita come: δ'(q0', a) = (q0, a) per ogni a ∈ Σ δ'((q, b), a) = (δ(q, a), b) per ogni q ∈ Q, a,b ∈ Σ

L'idea è che A' simula A tenendo traccia nell'ultimo simbolo letto nello stato. Quando A' raggiunge uno stato finale di A, accetta se l'ultimo simbolo corrisponde a quello memorizzato nello stato.

Poiché A' è un DFA finito, ROR(L) è regolare.

1. **(12 punti)** Se L è un linguaggio sull'alfabeto $\{0,1\}$, la *rotazione a sinistra* di L è l'insieme delle stringhe

$$\mathrm{ROL}(L) = \{wa \mid aw \in L, w \in \{0,1\}^*, a \in \{0,1\}\}.$$

Per esempio, se $L = \{0, 01, 010, 10100\}$, allora $\mathrm{ROL}(L) = \{0, 10, 100, 01001\}$. Dimostra che se L è regolare allora anche $\mathrm{ROL}(L)$ è regolare.

Sia A = (Q, Σ, δ, q0, F) il DFA che riconosce L. Costruiamo A' = (Q', Σ, δ', q0', F') che riconosce ROL(L) come segue:

Q' = Q × Σ ∪ {q0'} q0' è un nuovo stato iniziale F' = {(q, a) | q ∈ F, a ∈ Σ} δ' è definita come:

- δ'(q0', a) = (q0, a) per ogni a ∈ Σ
- δ'((q, b), a) = (δ(q, a), b) per ogni q ∈ Q, a,b ∈ Σ

L'idea è che A' simula A, ma "ritarda" la lettura del primo simbolo, memorizzandolo nello stato. Quando A' raggiunge uno stato finale di A, accetta se l'ultimo simbolo letto porta A in uno stato finale.

A' accetta wa se e solo se aw ∈ L. Quindi, L(A') = ROL(L).

Poiché A' è un DFA finito, ROL(L) è regolare.

1. **(12 punti)** Se L è un linguaggio e a un simbolo, allora L/a, il *quoziente* di L e a, è l'insieme delle stringhe

$$L/a = \{w \mid wa \in L\}.$$

Per esempio, se $L = \{0, 001, 100\}$, allora $L/0 = \{\varepsilon, 10\}$. Dimostra che se L è regolare allora anche L/a è regolare.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Data una parola $wa \in L$ dove $w = w_1 \dots w_n$, la computazione di A su aw è una sequenza di stati $r_0 r_1 \dots r_{n+1}$ tali che:

- $r_0 = q_0$;
- $\delta(r_{i-1}, w_i) = r_i$ per ogni $i = 1, \dots, n$;
- $\delta(r_n, a) = r_{n+1}$;
- $r_{n+1} \in F$.

Data questa osservazione possiamo costruire un automa A' che accetta il linguaggio L/a cambiando gli stati finali di A. Formalmente, $A' = (Q, \Sigma, \delta, q_0, F')$ dove Q, Σ, δ e q_0 sono gli stessi di A e l'insieme degli stati finali contiene tutti gli stati che raggiungono uno stato finale di A dopo aver consumato a:

$$F' = \{q \mid \delta(q, a) \in F\}.$$

In questo modo abbiamo che per ogni $wa \in L$ la sequenza di stati $r_0 \dots r_n$ descritta sopra è una computazione di A' che accetta w (perché r_n diventa uno stato finale di A'), ed abbiamo dimostrato che se $wa \in L$ allora $w \in L(A')$. Viceversa, se $w \in L(A')$ allora esiste una computazione $s_0 \dots s_n$ di A' tale che:

- $s_0 = q_0$;
- $\delta(s_{i-1}, w_i) = s_i$ per ogni $i = 1, \dots, n$;
- $s_n \in F'$.

Di conseguenza, $s_{n+1} = \delta(s_n, a) \in F$ e la sequenza di stati $s_1 \dots s_{n+1}$ è una computazione di A su wa, ed abbiamo dimostrato che se $w \in L(A')$ allora $wa \in L$. Quindi possiamo concludere che il linguaggio di A' è precisamente L/a, come richiesto.

3. (12 punti) Una CFG è detta *lineare a destra* se il corpo di ogni regola ha al massimo una variabile, e la variabile si trova all'estremità di destra. In altre parole, tutte le regole di una grammatica lineare a destra sono nella forma $A \to wB$ o $A \to w$, dove A e B sono variabili e w è una stringa di zero o più simboli terminali.

Dimostra che ogni grammatica lineare a destra genera un linguaggio regolare. *Suggerimento:* costruisci un ε-NFA che simula le derivazioni della grammatica.

Soluzione: Data una grammatica lineare a destra $G = (V, \Sigma, R, S)$, possiamo notare che le derivazioni di G hanno una struttura particolare. Siccome le regole sono nella forma $A \to wB$ o $A \to w$, dove A e B sono variabili e w è una stringa di zero o più simboli terminali, ogni derivazione di una parola della grammatica sarà formata da n applicazioni di una regola del tipo $A \to wB$ seguita da un'applicazione finale di una regola $A \to w$. A partire da questa osservazione possiamo costruire un ε-NFA $A = (Q, \Sigma, \delta, q_0, F)$ che simula le derivazioni della grammatica. Per rendere più chiara la costruzione usiamo una notazione abbreviata per la funzione di transizione, che ci fornisce un modo per far consumare un'intera stringa all'automa in un singolo passo. Possiamo simulare queste transizioni estese introducendo stati aggiuntivi per consumare la stringa un simbolo alla volta. Siano p e q stati dell'ε-NFA e sia $w = w_1 \dots w_n$ una stringa sull'alfabeto Σ. Per fare in modo che l'automa vada da p a q consumando l'intera stringa w introducendo nuovi stati intermedi $r_1 \dots r_{n-1}$ e definendo la funzione di transizione come segue:

$$\begin{aligned} \delta(p, w_1) &\text{ contiene } r_1, \\ \delta(r_1, w_2) &= \{r_2\}, \\ &\dots \\ \delta(r_{n-1}, w_n) &= \{q\}. \end{aligned}$$

Useremo la notazione $q \in \delta(p, w)$ per denotare quando l'automa può andare da p a q consumando la stringa di input w. Gli stati dell'automa sono $Q = V \cup \{q_f\} \cup E$ dove V è l'insieme delle variabili della grammatica, q_f è l'unico stato finale dell'automa e E è l'insieme di stati necessari per realizzare le transizioni estese appena descritte. Lo stato iniziale è S, variabile iniziale della grammatica. La funzione di transizione è definita come segue:

- $B \in \delta(A, w)$ per ogni regola $A \to wB$ della grammatica;
- $q_f \in \delta(A, w)$ per ogni regola $A \to w$ della grammatica.

L'automa A che abbiamo costruito è in grado di riconoscere lo stesso linguaggio della grammatica G perché simula le derivazioni di G nel modo descritto di seguito. L'automa inizia la computazione dallo stato che corrisponde alla variabile iniziale di G, e ripete i seguenti passi:

(a) se lo stato corrente corrente corrisponde alla variabile A, sceglie non deterministicamente una delle regole per A;
(b) se la regola scelta è $A \to wB$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato B, altrimenti la computazione si blocca su questo ramo;
(c) se la regola scelta è $A \to w$, prova a consumare la stringa w dall'input. Se ci riesce va nello stato finale q_f e accetta se ha consumato tutto l'input. La computazione si blocca su questo ramo se l'automa non riesce a consumare w o se non ha consumato tutto l'input quando arriva in q_f.

1. **(12 punti)** Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio regolare allora anche il linguaggio

$$substring(L) = \{y \in \Sigma^* \mid xyz \in L \text{ con } x, z \in \Sigma^*\}$$

è un linguaggio regolare.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Costruiamo un ε-NFA A' che accetta il linguaggio $substring(L)$. L'automa A' è costituto dagli stessi stati di A a cui viene aggiunto un nuovo stato iniziale q_0'. La funzione di transizione contiene una ϵ-transizione dal nuovo stato iniziale q_0' verso tutti gli stati di A che sono raggiungibili da q_0, e si comporta come A per gli altri stati. Gli stati finali dell'automa sono tutti gli stati di A a partire dai quali esiste una computazione che raggiunge uno stato finale di A.

Formalmente, $A' = (Q', \Sigma, \delta', q_0', F')$ è definito come segue.

- $Q' = Q \cup \{q_0'\}$, dove q_0' è uno stato nuovo che non appartiene a Q.
- L'alfabeto Σ rimane lo stesso.
- $\delta'(q_0', \varepsilon) = \{q \in Q \mid \text{esiste una computazione da } q_0 \text{ a } q \text{ in } A\}$.
- $\delta'(q, a) = \{\delta(q, a)\}$ per ogni stato $q \neq q_0'$ e $a \in \Sigma$.
- $F' = \{q \in Q \mid \text{esiste una computazione da } q \text{ ad uno stato di } F\}$.

Per dimostrare che A' riconosce il linguaggio $substring(L)$, dobbiamo considerare due casi.

- Se $w \in L$, e data una qualsiasi suddivisione $w = xyz$, esiste una computazione di A che accetta la parola. Definiamo q_1 come lo stato raggiunto da A dopo aver consumato y, e q_2 come lo stato raggiunto da A dopo aver consumato xy. Allora possiamo costruire una computazione di A' che accetta y in questo modo:

$$q_0' \xrightarrow{\varepsilon} q_1 \xrightarrow{y_1} \dots \xrightarrow{y_n} y_2.$$

  Siccome $y_2 \in F'$, la computazione è accettante per A'.

- Se y è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma

$$q_0' \xrightarrow{\varepsilon} q_1 \xrightarrow{y_1} \dots \xrightarrow{y_n} y_{n+1},$$

  con $y_{n+1} \in F'$. Per la defnizione della funzione di transizione δ', abbiamo che esiste una computazione da q_0 a q_1 di A, che consuma una certa parola x. Per la definizione dell'insieme di stati finali F', esiste una computazione di A che raggiunge uno stato finale di A a partire da y_{n+1}, che consuma una certa parola z. Di conseguenza, la parola xyz è accettata da A.

1. **(12 punti)** Data una parola $w \in \Sigma^*$, definiamo $evens(w)$ come la sottosequenza di w che contiene solo i simboli in posizione pari (iniziando a contare da 1). Per esempio, $evens(INDICEPARI) = NIEAI$. Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio regolare allora anche il linguaggio

$$evens(L) = \{evens(w) \mid w \in L\}$$

è un linguaggio regolare.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Costruiamo un ε-NFA A' che accetta il linguaggio $evens(L)$, aggiungendo un flag $0, 1$ agli stati di A, dove flag 0 corrisponde a "simbolo in posizione pari" e flag 1 corrisponde a "simbolo in posizione dispari". La funzione di transizione è fatta in modo da alternare i flag 0 e 1 dopo ogni transizione dell'automa. Se lo stato corrente ha flag 0, l'automa consuma il prossimo simbolo della stringa di input e prosegue nello stesso stato che raggiungerebbe A dopo aver consumato lo stesso simbolo. Se lo stato corrente ha flag 1, l'automa prosegue con una ε-transizione verso uno degli stati raggiungibili dall'automa A consumando un simbolo: simulando in questo modo il fatto che i simboli in posizione dispari vanno "saltati".

Formalmente, $A' = (Q', \Sigma, \delta', q_0', F')$ è definito come segue.

- $Q' = Q \times \{0, 1\}$.
- L'alfabeto Σ rimane lo stesso.
- $\delta'((q, 0), a) = \{(\delta(q, a), 1)\}$. Con il flag 0 l'automa consuma un simbolo di input ed ha una sola alternativa possibile: lo stato $\delta(q, a)$ raggiunto da A dopo aver consumato a. Il flag diventa 1.
- $\delta'((q, 1), \varepsilon) = \{(q', 0) \mid \text{esiste } a \in \Sigma \text{ tale che } \delta(q, a) = q'\}$. Con il flag 1 l'automa procede nondeterministicamente con una ε-transizione verso uno degli stati raggiungibili da A dopo aver consumato un simbolo arbitrario.
- $q'_0 = (q_0, 1)$. Lo stato iniziale corrisponde allo stato iniziale di A con flag 1 (simbolo in posizione dispari).
- $F' = F \times \{0, 1\}$. L'insieme degli stati finali di A' corrisponde all'insieme degli stati finali di A con qualsiasi flag.

Per dimostrare che A' riconosce il linguaggio $evens(L)$, data una parola $w = w_1 w_2 \dots w_n$, dobbiamo considerare due casi.

- Se $w \in L$, allora esiste una computazione di A che accetta la parola. Di conseguenza, esiste una computazione di A che accetta la parola:

  $$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

  con $s_0 = q_0$ e $s_n \in F$. Se w è di lunghezza pari, la computazione

  $$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_2} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_4} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1)$$

  è una computazione accettante per A' sulla parola $evens(w) = w_2 w_4 w_6 \dots w_n$, perché $(s_n, 1) \in F'$. Se w è di lunghezza dispari, allora la computazione accettante per A' su $evens(w)$ è

  $$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_2} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_4} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_{n-1}} (s_{n-1}, 1) \xrightarrow{\varepsilon} (s_n, 1).$$

- Se w è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma

  $$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_1} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_2} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1),$$

  se $(s_n, 1)$ è uno stato finale, oppure la forma

  $$(s_0, 1) \xrightarrow{\varepsilon} (s_1, 0) \xrightarrow{w_1} (s_2, 1) \xrightarrow{\varepsilon} (s_3, 0) \xrightarrow{w_2} (s_4, 1) \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} (s_n, 1) \xrightarrow{\varepsilon} (s_{n+1}, 0),$$

  quando è necessaria una ε-transizione finale per raggiungere uno stato finale. In entrambi i casi possiamo costruire una computazione accettante per A sostituendo le ϵ-transizioni con transizioni che consumano un carattere. Nel primo caso si ottiene una computazione che ha la forma

  $$s_0 \xrightarrow{u_1} s_1 \xrightarrow{w_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{w_2} s_4 \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} s_n,$$

  e accetta una parola $u = u_1 w_1 u_2 w_2 \dots u_n w_n$. Nel secondo caso si ottiene una computazione

  $$s_0 \xrightarrow{u_1} s_1 \xrightarrow{w_1} s_2 \xrightarrow{u_2} s_3 \xrightarrow{w_2} s_4 \xrightarrow{\varepsilon} \dots \xrightarrow{w_n} s_n \xrightarrow{u_{n+1}} s_{n+1},$$

  che accetta la parola $u = u_1 w_1 u_2 w_2 \dots u_n w_n u_{n+1}$. In entrambi i casi $evens(u) = w$, come richiesto.

1. **(12 punti)** Diciamo che una stringa x è un *prefisso* della stringa y se esiste una stringa z tale che $xz = y$, e che è un *prefisso proprio* di y se vale anche $x \neq y$. Dimostra che se $L \subseteq \Sigma^*$ è un linguaggio regolare allora anche il linguaggio

   $$NOPREFIX(L) = \{w \in L \mid \text{ nessun prefisso proprio di } w \text{ appartiene ad } L\}$$

   è un linguaggio regolare.

   Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Costruiamo un DFA A' che accetta il linguaggio $NOPREFIX(L)$, aggiungendo uno stato "pozzo" agli stati di A, ossia uno stato non finale q_s tale che la funzione di transizione obbliga l'automa a rimanere per sempre in q_s una volta che lo si raggiunge. Lo stato iniziale e gli stati finali rimangono invariati. La funzione di transizione di A' si comporta come quella di A per gli stati non finali, mentre va verso lo stato pozzo per qualsiasi simbolo dell'alfabeto a partire dagli stati finali. In questo modo le computazioni accettanti di A' sono sempre sequenze di stati dove solo l'ultimo stato è finale, mentre tutti quelli intermedi sono non finali. Di conseguenza le parole che A' accetta sono accettate anche da A, mentre tutti i prefissi propri sono parole rifiutate da A, come richiesto dalla definizione del linguaggio $NOPREFIX(L)$.

   Formalmente, $A' = (Q', \Sigma, \delta', q'_0, F')$ è definito come segue.

- $Q' = Q \cup \{q_s\}$, con $q_s \notin Q$.
- L'alfabeto Σ rimane lo stesso.
- $\delta'(q, a) = \begin{cases} \delta(q, a) & \text{se } q \notin F \\ q_s & \text{altrimenti} \end{cases}$
- $q'_0 = q_0$. Lo stato iniziale non cambia.
- $F' = F$. Gli stati finali rimangono invariati.

Per dimostrare che A' riconosce il linguaggio $NOPREFIX(L)$, dobbiamo considerare due casi.

- Se $w \in NOPREFIX(L)$, allora sappiamo che $w \in L$, mentre nessun prefisso proprio di w appartiene ad L. Di conseguenza esiste una computazione di A che accetta la parola:
$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$
con $s_0 = q_0$ e $s_n \in F$. Siccome tutti i prefissi propri di w sono rifiutati da A, allora gli stati $s_0, \dots, s_{n-1}$ sono tutti non finali. Per la definizione di A', la computazione che abbiamo considerato è anche una computazione accettante per A', e di conseguenza, $w \in L(A')$.
- Viceversa, se w è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma
$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$
con $s_0 = q_0$, $s_n \in F$ e dove tutti gli stati intermedi $s_0, \dots, s_{n-1}$ sono non finali. Di conseguenza, la computazione è una computazione accettante anche per A, quindi $w \in L$. Siccome tutti gli stati intermedi della computazione sono non finali, allora A rifiuta tutti i prefissi propri di w, e quindi $w \in NOPREFIX(L)$.

3. (12 punti) Una grammatica context-free è *lineare* se ogni regola in R è nella forma $A \to aBc$ o $A \to a$ per qualche $a, c \in \Sigma \cup \{\varepsilon\}$ e $A, B \in V$. I linguaggi generati dalle grammatiche lineari sono detti *linguaggi lineari*. Dimostra che i linguaggi regolari sono un sottoinsieme proprio dei linguaggi lineari.

Soluzione. Per risolvere l'esercizio dobbiamo dimostrare che ogni linguaggio regolare è anche un linguaggio lineare (i linguaggi regolari sono un sottoinsieme dei linguaggi lineari), e che esistono linguaggi lineari che non sono regolari (l'inclusione è propria).

- Dato un linguaggio regolare L, sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Inoltre, sappiamo che ogni DFA può essere convertito in una grammatica context-free dove le regole sono del tipo $R_i \to aR_j$ per ogni transizione $\delta(q_i, a) = q_j$ del DFA, e del tipo $R_i \to \varepsilon$ per ogni stato finale del q_i del DFA. Entrambi i tipi di regola rispettano le condizioni di linearità, quindi la grammatica equivalente al DFA è lineare, e questo implica che L è un linguaggio lineare.

- Consideriamo il linguaggio non regolare $L = \{0^n 1^n \mid n \geq 0\}$. La seguente grammatica lineare genera L:
$$S \to 0S1 \mid \varepsilon$$
Quindi, esiste un linguaggio lineare che non è regolare.

1. **(12 punti)** Data una stringa w di 0 e 1, il *flip* di w si ottiene cambiando tutti gli 0 in w con 1 e tutti gli 1 in w con 0. Dato un linguaggio L, il flip di L è il linguaggio

$$\mathit{flip}(L) = \{w \in \{0,1\}^* \mid \text{ il flip di } w \text{ appartiene ad } L\}.$$

Dimostra che la classe dei linguaggi regolari è chiusa rispetto all'operazione di flip.

Soluzione: Se L è un linguaggio regolare, allora sappiamo che esiste un DFA $A = (Q, \Sigma, \delta, q_0, F)$ che riconosce L. Costruiamo un DFA $A' = (Q', \Sigma, \delta', q_0', F')$ che accetta il linguaggio $\mathit{flip}(L)$ come segue.

- $Q' = Q$. L'insieme degli stati rimane lo stesso.
- L'alfabeto Σ rimane lo stesso.
- Per ogni stato $q \in Q$, $\delta'(q,0) = \delta(q,1)$ e $\delta'(q,1) = \delta(q,0)$. La funzione di transizione scambia gli 0 con 1 e gli 1 con 0.
- $q_0' = q_0$. Lo stato iniziale non cambia.
- $F' = F$. Gli stati finali rimangono invariati.

Per dimostrare che A' riconosce il linguaggio $\mathit{flip}(L)$, dobbiamo considerare due casi.

- Se $w \in \mathit{flip}(L)$, allora sappiamo che il flip di w appartiene ad L. Chiamiamo $\overline{w}$ il flip di w. Siccome A riconosce L, allora esiste una computazione di A che accetta $\overline{w}$:

$$s_0 \xrightarrow{\overline{w}_1} s_1 \xrightarrow{\overline{w}_2} \dots \xrightarrow{\overline{w}_n} s_n$$

con $s_0 = q_0$ e $s_n \in F$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A' sulla parola w. Di conseguenza, abbiamo dimostrato che $w \in L(A')$.

- Viceversa, se w è accettata dal nuovo automa A', allora esiste una computazione accettante che ha la forma

$$s_0 \xrightarrow{w_1} s_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} s_n$$

con $s_0 = q_0$, $s_n \in F'$. Se scambiamo gli zeri e gli uni nella computazione, otteniamo una computazione accettante per A sul flip di w. Di conseguenza, il flip di w appartiene ad L e abbiamo dimostrato che $w \in \mathit{flip}(L)$.

1. **(12 punti)** La *traslitterazione* è un tipo di conversione di un testo da una scrittura a un'altra che prevede la sostituzione di lettere secondo modalità prevedibili. Per esempio, il sistema di traslitterazione Hepburn permette di convertire la scrittura Kana giapponese nell'alfabeto latino usando la seguente tabella:

あ	ア	か	が	カ	ガ	さ	ざ	サ	ザ	た	だ	タ	ダ	な	ナ	は	ば	ぱ	ハ	バ	パ	ま	マ	や	ヤ	ら	ラ	わ	ワ
a	*a*	ka	ga	*ka*	*ga*	sa	za	*sa*	*za*	ta	da	*ta*	*da*	na	*na*	ha	ba	pa	*ha*	*ba*	*pa*	ma	*ma*	ya	*ya*	ra	*ra*	wa	*wa*
い	イ	き	ぎ	キ	ギ	し	じ	シ	ジ	ち	ぢ	チ	ヂ	に	ニ	ひ	び	ぴ	ヒ	ビ	ピ	み	ミ			り	リ		
i	*i*	ki	gi	*ki*	*gi*	shi	ji	*shi*	*ji*	chi	ji	*chi*	*ji*	ni	*ni*	hi	bi	pi	*hi*	*bi*	*pi*	mi	*mi*			ri	*ri*		
う	ウ	く	ぐ	ク	グ	す	ず	ス	ズ	つ	づ	ツ	ヅ	ぬ	ヌ	ふ	ぶ	ぷ	フ	ブ	プ	む	ム	ゆ	ユ	る	ル		
u	*u*	ku	gu	*ku*	*gu*	su	zu	*su*	*zu*	tsu	zu	*tsu*	*zu*	nu	*nu*	f u	bu	pu	*f u*	*bu*	*pu*	mu	*mu*	yu	*yu*	ru	*ru*		
え	エ	け	げ	ケ	ゲ	せ	ぜ	セ	ゼ	て	で	テ	デ	ね	ネ	へ	べ	ぺ	へ	べ	ぺ	め	メ			れ	レ		
e	*e*	ke	ge	*ke*	*ge*	se	ze	*se*	*ze*	te	de	*te*	*de*	ne	*ne*	he	be	pe	*he*	*be*	*pe*	me	*me*			re	*re*		
お	オ	こ	ご	コ	ゴ	そ	ぞ	ソ	ゾ	と	ど	ト	ド	の	ノ	ほ	ぼ	ぽ	ホ	ボ	ポ	も	モ	よ	ヨ	ろ	ロ	を	ヲ
o	*o*	ko	go	*ko*	*go*	so	zo	*so*	*zo*	to	do	*to*	*do*	no	*no*	ho	bo	po	*ho*	*bo*	*po*	mo	*mo*	yo	*yo*	ro	*ro*	o	*o*

ん	ン
n	*n*

Dati due alfabeti Σ e Γ, possiamo definire formalmente una traslitterazione come una funzione $T : \Sigma \mapsto \Gamma^*$ che mappa ogni simbolo di Σ in una stringa di simboli in Γ.

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

$$T(L) = \{w \in \Gamma^* \mid w = T(a_0)T(a_1)\dots T(a_n) \text{ per qualche } a_0a_1\dots a_n \in L\}.$$

Dimostrazione: Sia A = (Q, Σ, δ, q0, F) un automa a stati finiti deterministico (DFA) che riconosce L. Costruiamo un nuovo DFA A' = (Q, Γ*, δ', q0, F) che riconosce T(L) come segue:

- Gli stati Q e lo stato iniziale q0 rimangono gli stessi
- L'insieme degli stati finali F rimane lo stesso
- Per ogni transizione δ(q, a) = p in A, definiamo in A' la transizione δ'(q, T(a)) = p

A' simula A, ma invece di leggere un simbolo a ∈ Σ, legge la sua traslitterazione T(a).

Per dimostrare che A' riconosce T(L), mostriamo che per ogni w ∈ Σ*: w ∈ L se e solo se T(w) ∈ T(L)

(⇒) Se w = a1a2...an ∈ L, allora esiste una sequenza di stati q0, q1, ..., qn in A tale che: δ(qi-1, ai) = qi per i = 1, ..., n, e qn ∈ F

Per costruzione di A', avremo: δ'(qi-1, T(ai)) = qi per i = 1, ..., n

Quindi, T(w) = T(a1)T(a2)...T(an) è accettato da A', e T(w) ∈ T(L)

(⇐) Se T(w) ∈ T(L), allora T(w) è accettato da A'. Seguendo il ragionamento inverso, w deve essere accettato da A, quindi w ∈ L.

Poiché A' è un DFA, T(L) è regolare.

1. **(12 punti)** 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, $x \sqcup y$ rappresenta l'or bit a bit di x e y. Per esempio, $0011 \sqcup 0101 = 0111$.

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

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

Dimostrazione: Poiché L ed M sono regolari, esistono DFA A_L = (Q_L, {0,1}, δ_L, q_0L, F_L) e A_M = (Q_M, {0,1}, δ_M, q_0M, F_M) che li riconoscono rispettivamente.

Costruiamo un NFA N = (Q, {0,1}, δ, q_0, F) che riconosce L ⊔ M:

- Q = Q_L × Q_M × {0, 1, 2}
- q_0 = (q_0L, q_0M, 0)
- F = {(q_L, q_M, 2) | q_L ∈ F_L, q_M ∈ F_M}
- δ è definita come segue:
  1. δ((q_L, q_M, 0), 0) = {(δ_L(q_L, 0), q_M, 0), (q_L, δ_M(q_M, 1), 1)}
  2. δ((q_L, q_M, 0), 1) = {(δ_L(q_L, 1), q_M, 0), (q_L, δ_M(q_M, 0), 1)}
  3. δ((q_L, q_M, 1), 0) = {(q_L, δ_M(q_M, 1), 1)}
  4. δ((q_L, q_M, 1), 1) = {(q_L, δ_M(q_M, 0), 1)}
  5. δ((q_L, q_M, 1), ε) = {(q_L, q_M, 2)} se |x| = |y|

N simula A_L e A_M in parallelo, usando il terzo componente dello stato per tenere traccia della fase: 0 per leggere x, 1 per leggere y, e 2 per accettare se le lunghezze sono uguali.

Poiché N è un NFA e riconosce L ⊔ M, L ⊔ M è regolare.

1. **(12 punti)** Lo *xor bit a bit* è un'operazione binaria che prende due stringhe binarie di uguale lunghezza ed esegue lo xor logico su ogni coppia di bit corrispondenti. Il risultato è una stringa binaria in cui ogni posizione è 0 se i due bit sono uguali, 1 se sono diversi. Date due stringhe binarie x e y di uguale lunghezza, $x \oplus y$ rappresenta lo xor bit a bit di x e y. Per esempio, $0011 \oplus 0101 = 0110$.

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

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

Dimostrazione: Poiché L ed M sono regolari, esistono DFA A_L = (Q_L, {0,1}, δ_L, q_0L, F_L) e A_M = (Q_M, {0,1}, δ_M, q_0M, F_M) che li riconoscono rispettivamente.

Costruiamo un NFA N = (Q, {0,1}, δ, q_0, F) che riconosce L ⊕ M:

- Q = Q_L × Q_M × {0,1}
- q_0 = (q_0L, q_0M, 0)
- F = {(q_L, q_M, i) | q_L ∈ F_L, q_M ∈ F_M, i ∈ {0,1}}
- δ è definita come segue:
  1. δ((q_L, q_M, 0), 0) = {(δ_L(q_L, 0), δ_M(q_M, 0), 0), (δ_L(q_L, 0), δ_M(q_M, 1), 1)}
  2. δ((q_L, q_M, 0), 1) = {(δ_L(q_L, 1), δ_M(q_M, 1), 0), (δ_L(q_L, 1), δ_M(q_M, 0), 1)}
  3. δ((q_L, q_M, 1), 0) = {(δ_L(q_L, 0), δ_M(q_M, 1), 1), (δ_L(q_L, 0), δ_M(q_M, 0), 0)}
  4. δ((q_L, q_M, 1), 1) = {(δ_L(q_L, 1), δ_M(q_M, 0), 1), (δ_L(q_L, 1), δ_M(q_M, 1), 0)}

N simula A_L e A_M in parallelo, usando il terzo componente dello stato per tenere traccia del risultato dello XOR bit a bit. Poiché N è un NFA e riconosce L ⊕ M, L ⊕ M è regolare.

1. **(12 punti)** Data una stringa $w \in \Sigma^*$, definiamo una operazione che scambia di posizione i caratteri della stringa a due a due:

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

   Per esempio, $\text{SWAP}(\texttt{ABCDE}) = \texttt{BADCE}$.

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

$$\text{SWAP}(L) = \{\text{SWAP}(w) \mid w \in L\}.$$

Dimostrazione: Sia A = (Q, Σ, δ, q0, F) un DFA che riconosce L. Costruiamo un NFA A' = (Q', Σ, δ', q0', F') che riconosce SWAP(L):

Q' = Q × {0, 1, 2} q0' = (q0, 0) F' = {(q, 2) | q ∈ F} δ' è definita come segue:

1. δ'((q, 0), ε) = {(q, 2)} (per gestire le stringhe vuote o di lunghezza 1)
2. δ'((q, 0), a) = {(δ(q, a), 1), (δ(q, a), 2)} per ogni a ∈ Σ
3. δ'((q, 1), a) = {(δ(q, a), 2)} per ogni a ∈ Σ
4. δ'((q, 2), a) = {(δ(q, a), 0)} per ogni a ∈ Σ

L'idea è che lo stato 0 rappresenta l'inizio della stringa o dopo uno scambio, lo stato 1 rappresenta dopo aver letto il primo carattere di una coppia, e lo stato 2 rappresenta dopo aver completato uno scambio o alla fine della stringa.

Poiché A' è un NFA che riconosce SWAP(L), SWAP(L) è regolare