Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 1 settembre 2021

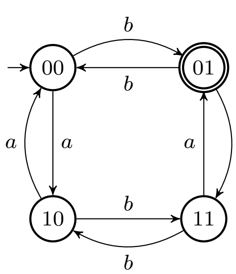
Esercizio 1 [6] Determinare una espressione regolare per il linguaggio

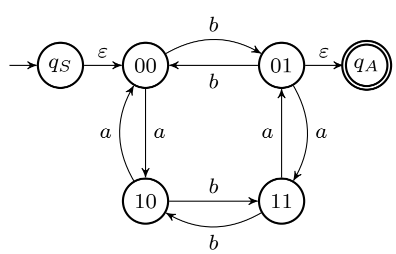
$$L = \{x \in \{a, b\}^* \mid x \text{ contiene un numero pari di } a \text{ e dispari di } b\}$$

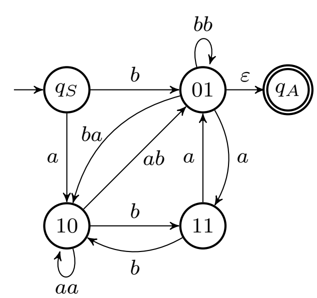
ovvero dimostrare che tale espressione regolare non esiste.

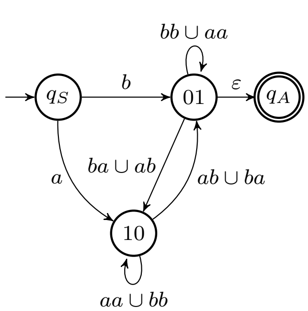
Soluzione: Il linguaggio L è regolare, perché per riconoscere l'appartenza di una stringa in L non è necessario contare le occorrenze di simboli a e b, ma solo memorizzare la parità o disparità del numero di occorrenze. Cercare di applicare il pumping lemma per i linguaggi regolari non può riuscire a dimostrare che L non è regolare: infatti esisterà sempre una suddivisione che contiene un numero pari di a e/o un numero pari di b che può essere pompata arbitrariamente verso l'alto o verso il basso.

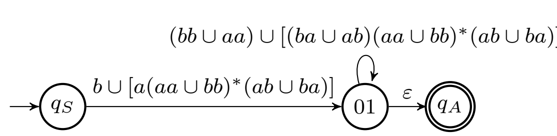
Una strategia per determinare una espressione regolare per il linguaggio richiesto consiste nel determinare dapprima un DFA che riconosca il linguaggio, e successivamente derivare dal DFA una REX. Il linguaggio L può essere riconosciuto da un DFA con quattro stati corrispondenti alle quattro possibili condizioni di parità/disparità per i simboli a e b. Trasformando in GNFA e rimuovendo nell'ordine i nodi 00, 11, 10 e 01 si ottiene:











ed infine

$$\{b \cup [a\,(aa \cup bb)^*\,(ab \cup ba)]\}\ \{(aa \cup bb) \cup [(ab \cup ba)\ (aa \cup bb)^*\ (ab \cup ba)]\}^*\,.$$

Esercizio 2 [6] Siano A e B linguaggi regolari. Il linguaggio

$$\begin{aligned} C = \{\, w \mid \quad & w = a_1 b_1 \cdots a_n b_n, n > 0, |a_i| = |b_i| = 1 \text{ per } 1 \le i \le n, \\ & a = a_1 \cdots a_n \in A, b = b_1 \cdots b_n \in B \,\} \end{aligned}$$

è regolare? Giustificare la risposta con una dimostrazione. (Ad esempio, $0a1b2c \in C$ se e solo se $012 \in A$ e $abc \in B$.)

Soluzione: L'operazione tra i linguaggi A e B che ottiene il linguaggio C è detta *rimescolamento perfetto* (*perfect shuffle*). Assumiamo di conoscere un DFA $M_A = (Q_A, \Sigma_A, \delta_A, q_0^A, F_A)$ per il linguaggio A ed un DFA $M_B = (Q_B, \Sigma_B, \delta_B, q_0^B, F_B)$ per B, e dimostriamo che C è regolare esibendo un DFA M derivato da M_A e M_B.

Sia $M = (Q, \Sigma, \delta, q_0, F)$, ove:

- $\Sigma = \Sigma_A \cup \Sigma_B$ è l'unione degli alfabeti di A e B
- $Q = Q_A \times Q_B \times \{f, t\}$ è il prodotto cartesiano degli stati di M_A e M_B ed un flag booleano
- $q_0 = (q_0^A, q_0^B, f)$ è la tripla corrispondente agli stati iniziali di M_A e M_B ed al valore falso per il flag
- $F = F_A \times F_B \times \{f\}$ è il prodotto cartesiano degli stati di accettazione di M_A e M_B e del valore falso per il flag
- $\delta = \delta' \cup \delta''$, ove
  - per ogni transizione $\delta_A(q, \sigma) = q'$, δ' include la transizione $\delta'((q, \overline{q}, f), \sigma) = (q', \overline{q}, t)$, per ogni $\overline{q} \in Q_B$
  - per ogni transizione $\delta_B(q, \sigma) = q'$, δ'' include la transizione $\delta''((\overline{q}, q, t), \sigma) = (\overline{q}, q', f)$, per ogni $\overline{q} \in Q_A$

Dimostriamo che M riconosce il linguaggio C. Sia $w \in C$, perciò la lunghezza di w è pari; la stringa x costituita dai caratteri in posizione pari di w appartiene ad A, mentre la stringa y costituita dai caratteri in posizione dispari appartiene a B. Pertanto $M_A(x)$ accetta con una successione di stati $q_0^A, \ldots, q_F^A \in F_A$, e analogamente $M_B(y)$ accetta con una successione di stati $q_0^B, \ldots, q_F^B \in F_B$. Consideriamo ora la computazione di M sulla stringa w. Innanzi tutto, la variabile booleana codificata nello stato interno di M garantisce che leggendo la stringa w venga applicata innanzi tutto una transizione in δ' derivata da δ_A, poi una transizione in δ'' derivata da δ_B, e così via. Il ruolo della variabile booleana è duplice: evita che una regola di

M_A possa essere applicata leggendo un carattere in posizione dispari, oppure una regola di M_B possa essere applicata leggendo un carattere in posizione pari; inoltre garantisce che M possa accettare solo dopo aver letto un numero pari di caratteri in ingresso. Poiché M_A è un DFA, esiste una sola transizione in δ applicabile al primo carattere di w, e precisamente quella inserita in δ' in corrispondenza della transizione di δ_A applicata leggendo il primo carattere di x. La transizione imposta la variabile al valore vero, dunque le uniche transizioni applicabili per leggere il secondo carattere di w sono quelle in δ''; poiché anche M_B è un DFA, esiste una sola transizione applicabile, corrispondente alla transizione applicata da M_B per leggere il primo carattere di y. La variabile booleana torna al valore falso, e M si prepara dunque a leggere il terzo carattere di w corrispondente al secondo carattere di x. Alla fine, dopo aver letto l'ultimo carattere di w corrispondente all'ultimo carattere di y, lo stato interno di M è $(q_F^A, q_F^B, f) \in F$, e quindi $M(w)$ accetta.

Viceversa, supponiamo che $M(w)$ accetti, dunque termina di leggere la stringa w in uno stato $(q_F^A, q_F^B, f) \in F$. Poiché per costruzione la variabile booleana ha valore falso, w ha lunghezza pari: $|w| = 2n$. Siano x e y le due sottostringhe nelle posizioni pari e dispari di w, con $|x| = |y| = n$. Per leggere w il DFA M ha utilizzato la successione di $2n$ stati (q_0^A, q_0^B, f), (q_1^A, q_0^B, t), (q_1^A, q_1^B, f), $\ldots$, (q_F^A, q_{n-2}^B, t), (q_F^A, q_F^B, f). Ora è immediato verificare che la sequenza di stati q_0^A, q_1^A, $\ldots$, q_{n-2}^A, q_F^A corrisponde alla successione di stati di M_A durante la lettura della sottostringa x; poiché per costruzione $q_F^A \in F_A$, $M_A(a)$ accetta e dunque $x \in A$. Analogamente la sequenza di stati q_0^B, q_1^B, $\ldots$, q_{n-2}^B, q_F^B corrisponde alla successione di stati di M_B mentre legge y, e poiché $q_F^B \in F_B$, $M_B(y)$ accetta, e quindi $y \in B$. Pertanto $w \in C$.

Esercizio 3 [8] Siano A e B linguaggi liberi dal contesto (CFL). Il linguaggio

$$C = \{\, w \mid \quad \begin{array}{l} w = a_1 b_1 \cdots a_n b_n, n > 0, |a_i| = |b_i| = 1 \text{ per } 1 \le i \le n, \\ a = a_1 \cdots a_n \in A, b = b_1 \cdots b_n \in B \,\} \end{array}$$

è CFL? Giustificare la risposta con una dimostrazione.

Soluzione: I linguaggi liberi dal contesto (CFL) non sono chiusi rispetto all'operazione di *rimescolamento perfetto* (*perfect shuffle*). Per dimostrarlo è sufficiente esibire un controesempio, ossia due linguaggi CFL A e B tali che il loro rimescolamento perfetto C non è CFL.

Consideriamo $A = \{0^k 1^{2k} \mid k \ge 0\}$ e $B = \{c^{2k} d^k \mid k \ge 0\}$. È immediato verificare che sia A che B sono CFL. Ad esempio, una CFG per A è $S \to 0S11 \mid \varepsilon$, mentre una CFG per B è $S \to ccSd \mid \varepsilon$.

Consideriamo dunque il rimescolamento perfetto C di A e B. Se $w \in C$, allora w ha lunghezza pari $|w| = 2n > 0$, ed è costituito necessariamente da una sottostringa $a \in A$ di lunghezza n nelle posizioni pari, con n multiplo di 3, ed una sottostringa $b \in B$ di lunghezza n nelle posizioni dispari. Pertanto $a = 0^k 1^{2k}$ e $b = c^{2k} d^k$, con $k = n/3$, e quindi $w = \underbrace{0c \cdots 0c}_{2k} \underbrace{1c \cdots 1c}_{2k} \underbrace{1d \cdots 1d}_{2k}$. In generale quindi $C = \{(0c)^k (1c)^k (1d)^k \mid k > 0\}$.

Dimostriamo che C non è CFL utilizzando il pumping lemma. Supponiamo per assurdo che C sia CFL; dunque esiste una lunghezza p tale che se $w \in C$ ha lunghezza $\geq p$, deve esistere una suddivisione $w = uvxyz$ tale che $uv^i xy^i z \in C$ per ogni $i \geq 0$, $|vy| > 0$ e $|vxy| \leq p$. Consideriamo come elemento di C la stringa $s = (0c)^p (1c)^p (1d)^p$ di lunghezza $6p > p$. Si osservi che in ogni elemento di C il numero di occorrenze dei simboli 0, 1, c e d è fissato rispetto alla lunghezza totale dell'elemento: in particolare, 1/6 dei simboli sono 0, 1/3 dei simboli sono 1, 1/3 dei simboli sono c, ed infine 1/6 dei simboli sono d. Supponiamo dunque che esista una suddivisione $s = uvxyz$ che soddisfi le condizioni del pumping lemma. Poiché la stringa uv^2xy^2z deve appartenere a C, essa deve rispettare la proporzione del numero di simboli suddetta, dunque v ed y devono contenere tutti i quattro tipi di simboli 0, 1, c e d. Poiché 0 e d sono posizionati agli estremi di s, $|vxy| > 2p+3$, il che contraddice la condizione del lemma $|vxy| \leq p$. Dunque il lemma non è valido, e ciò implica che C non è CFL, come volevasi dimostrare.

Esercizio 4 [6] Sia $L \subseteq \Sigma^*$ un linguaggio Turing-riconoscibile (ossia r.e.), e sia $L^p = \{x \mid \exists\, y \in \Sigma^+ \text{ tale che } xy \in L\}$. Dimostrare che L^p è Turing-riconoscibile.

Soluzione: Il linguaggio L^p è costituito da tutti i prefissi propri degli elementi di L. Se ad esempio $abcd \in L$, allora certamente a, ab e abc fanno parte di L^p.

Per dimostrare che L^p è ricorsivamente enumerabile è sufficiente esibire un enumeratore E' per esso, ossia una TM che produce in uscita tutti e soli gli elementi del linguaggio, anche se possibilmente con ripetizioni e senza ordine prefissato. In effetti, poiché L è ricorsivamente enumerabile, esiste un enumeratore E per esso, ed è semplice costruire E' basandosi su E:

E'= "On any input:
1. Simulate the enumerator E for L
2. for each element $x \in L$ printed by E:
3. for each proper prefix y of x:
4. print y"

Sia $z \in L^p$; dunque z è il prefisso proprio di un elemento $x \in L$, ossia $x = yz$, con $|z| > 0$. Poiché E enumera L, dopo un tempo finito genererà l'elemento x di L. Pertanto E' stamperà tutti i prefissi propri di x, e dunque anche z. Viceversa, se E' stampa una stringa y, allora necessariamente tale stringa è un prefisso proprio di una stringa x stampata da E, e poiché E è un enumeratore per L, $x \in L$. Pertanto $y \in L^p$. Concludiamo che E' è un enumeratore per L^p, e dunque L^p è ricorsivamente enumerabile.

Esercizio 5 [5] Siano $A, B \subseteq \Sigma^*$ linguaggi Turing-riconoscibili (ossia r.e.). Dimostrare che $A \setminus B = \{x \in A \mid x \notin B\}$ non è necessariamente Turing-riconoscibile.

Soluzione: Possiamo utilizzare un contro-esempio per dimostrare che $A \setminus B$ non è necessariamente r.e. anche se A e B lo sono. Consideriamo come linguaggio B qualunque linguaggio r.e. ma non decidibile, ad esempio $B = \mathcal{A}_{\mathrm{TM}}$. Sia invece $A = \Sigma^*$, ove Σ è l'alfabeto di supporto di B (ossia $B \subseteq \Sigma^*$); si osservi che Σ^* è decidibile e quindi r.e. Pertanto $A \setminus B = \Sigma^* \setminus B = B^c$. Se ora $A \setminus B$ fosse necessariamente r.e., allora B ($\mathcal{A}_{\mathrm{TM}}$) sarebbe decidibile, in quanto sia B che B^c sarebbero r.e.: una contraddizione. Pertanto resta assodato che $A \setminus B$ non è necessariamente r.e. anche se A e B lo sono.

Esercizio 6 [9] Dimostrare che NL è chiuso rispetto alle operazioni di unione e concatenazione, ossia dimostrare che se $L_1, L_2 \in \mathrm{NL}$ allora $L_1 \cup L_2 \in \mathrm{NL}$ e $L_1 \circ L_2 = \{xy \mid x \in L_1, y \in L_2\} \in \mathrm{NL}$.

Soluzione: Poiché $L_1, L_2 \in \mathrm{NL}$, esistono due TM nondeterministiche N_1 e N_2 che decidono L_1 e L_2, rispettivamente. Ciascuna NTM N_i possiede un nastro di ingresso di sola lettura ed un nastro di lavoro che può essere letto e scritto liberamente. Per definizione della classe NL, la quantità di celle di lavoro utilizzate in ciascun ramo di computazione deterministico è in $O(\log(n))$, ove n è la dimensione della stringa sul nastro di ingresso.

Per dimostrare che $L_1 \cup L_2 \in \mathrm{NL}$ consideriamo la seguente NTM $N_\cup$ dello stesso tipo di N_1 e N_2:

$N_\cup$= "On input w:

1. Nondeterministically choose either L_1 or L_2
2. Simulate the NTM N_i corresponding to the chosen language
3. Accept or reject according to $N_i(w)$

Ovviamente $N_\cup$ è una NTM che decide il linguaggio $L_1 \cup L_2$: infatti, $w \in L_1 \cup L_2$ se e solo se w appartiene ad almeno uno dei due linguaggi, e quindi se e solo se esiste un ramo di computazione deterministico in $N_\cup$ che accetta. È immediato verificare che in ciascun ramo di computazione deterministico la quantità di celle di lavoro utilizzate da $N_\cup$ è in $O(\log(|w|))$. Infatti i passi 1 e 3 utilizzano una quantità costante ($O(1)$) di celle di lavoro. La quantità di celle di lavoro utilizzate nel passo 2 è essenzialmente quella della NTM N_i simulata; per definizione, in ciascun ramo di computazione deterministico si utilizzano al più $O(\log(|w|))$ celle di lavoro. In conclusione, $L_1 \cup L_2 \in \mathrm{NL}$.

Per dimostrare che $L_1 \circ L_2 \in \mathrm{NL}$ consideriamo la seguente NTM $N_\circ$ dello stesso tipo di N_1 e N_2:

$N_\circ$= "On input w:

1. Nondeterministically choose a number c between 0 and $|w|$
2. Simulate N_1 on the leftmost substring of w of length c
3. If N_1 on the leftmost substring rejects, then reject
4. Simulate N_2 on the rightmost substring of w of length $|w| - c$
5. If N_2 on the right substring accepts, then accept; otherwise, reject"

È facile verificare che $N_\circ$ è una NTM che decide il linguaggio $L_1 \circ L_2$. Infatti $w \in L_1 \circ L_2$ se e solo se $w = xy$, con $x \in L_1$ e $y \in L_2$; dunque se e solo se esiste una lunghezza $c = |x|$ tale che N_1 accetta la porzione sinistra di w di lunghezza c e N_2 accetta la restante porzione destra di w; dunque se e solo se esiste un ramo di computazione deterministica per $N_\circ$ che "indovina" il valore c, accetta la sottostringa sinistra nella simulazione di N_1 ed accetta la sottostringa destra nella simulazione di N_2.

Il passo 1 deve memorizzare il valore di c sul nastro di lavoro; poiché $c < |w|$, il numero di celle utilizzato è $O(\log(|w|))$. Le simulazioni delle NTM N_1 e N_2 utilizzano essenzialmente, in ciascun ramo di computazione deterministico, lo stesso numero di celle di lavoro dei corrispondenti rami nella computazione di N_1 e N_2; pertanto, i passi 2 e 4 utilizzano un numero di celle di lavoro in $O(\log(|w|))$. Si osservi che queste simulazioni non possono operare su copie della stringa di ingresso w, e quindi $N_\circ$ controlla continuamente la posizione della testina sul nastro in ingresso in modo da riconoscere i limiti delle sottostringhe. È possibile compiere questo lavoro memorizzando la posizione della testina sul nastro di lavoro, ossia un valore numerico non superiore a $|w|$, e quindi con un costo di $O(\log(|w|))$ celle di lavoro. Infine i restanti passi utilizzano un numero costante di celle di lavoro. In conclusione, $L_1 \circ L_2 \in \mathrm{NL}$.