***Automi e Linguaggi* (M. Cesati)**

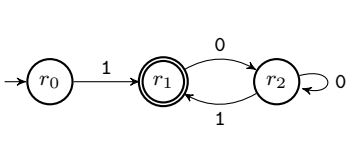
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 16 luglio 2024

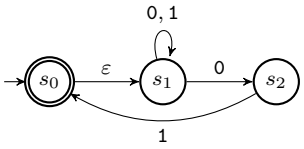
Esercizio 1 [7] Si considerino le REX $R_1 = \mathtt{1}(\mathtt{0}^*\mathtt{01})^*$ e $R_2 = [(\mathtt{0} \cup \mathtt{1})^*\mathtt{01}]^*$. Determinare una REX per il linguaggio $L(R_1) \cap L(R_2)$.

Soluzione: La chiusura dei linguaggi regolari rispetto all'intersezione garantisce l'esistenza della espressione regolare richiesta. Un procedimento meccanico per risolvere l'esercizio consiste nel ricavare dalle REX date R_1 e R_2 i corrispondenti automi a stati finiti N_1 e N_2, calcolare poi da questi un NFA N corrispondente all'intersezione dei rispettivi linguaggi, ed infine derivare una REX R equivalente all'automa N.

Determiniamo innanzi tutto gli NFA N_1 e N_2 per i linguaggi $L(R_1)$ e $L(R_2)$. Semplificando gli automi derivati meccanicamente si ottengono:

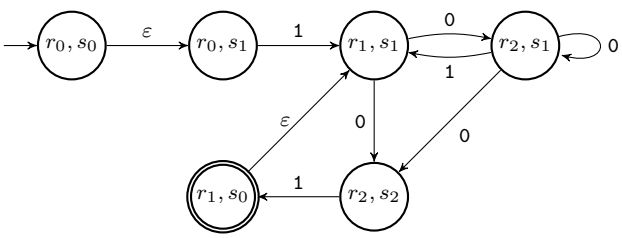


(N_1)

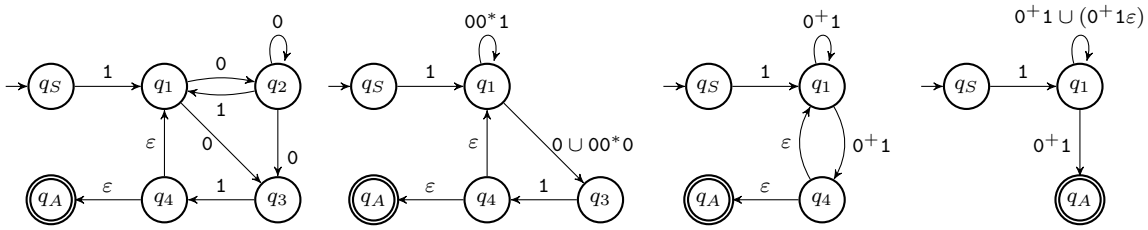


(N_2)

Da N_1 e N_2 possiamo derivare un NFA N tale che $L(N) = L(N_1) \cap L(N_2)$; N deve eseguire "in parallelo" la computazione di N_1 e N_2, quindi il suo insieme degli stati è costituito dal prodotto cartesiano tra gli stati di N_1 e N_2.

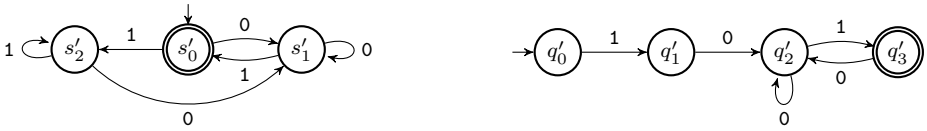


Infine possiamo trasformare N in un GNFA e derivare una REX equivalente eliminando nell'ordine q_2, q_3, q_4 e q_1:



Si osservi che 00^*1 è equivalente a 0^+1, che $0 \cup 00^*0$ è equivalente a 0^+, e che $[0^+1 \cup (0^+1\varepsilon)]^*0^+1$ è equivalente a $(0^+1)^+$. Pertanto, una REX per il linguaggio $L(R_1) \cap L(R_2)$ è $1(0^+1)^+$.

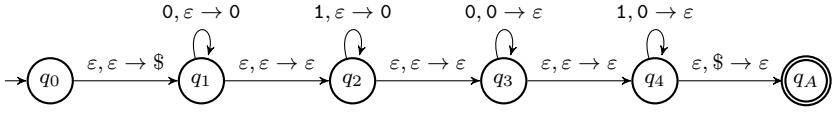
Una REX diversa, ma ovviamente equivalente, si ottiene trasformando gli NFA iniziali in DFA prima di determinare l'automa intersezione. In particolare, un DFA equivalente a N_2 ed il DFA intersezione risultante dall'intersezione con N_1 (già un DFA) sono i seguenti:



Quindi una REX equivalente è $10(0 \cup 10)^*1$.

Esercizio 2 [6.5] Si consideri il linguaggio $A = \{0^h1^k0^l1^m \mid h,k,l,m \geq 0 \text{ e } h+k = l+m\}$. A è un linguaggio libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A è CFL, in quanto può essere riconosciuto dal seguente automa a pila (PDA) P:



Infatti, supponiamo che $w \in A$; pertanto $w = 0^h1^k0^l1^m$ con $h,k,l,m \geq 0$ e $h+k = l+m$. La computazione $P(w)$ è non deterministica, ma è sufficiente determinare un singolo ramo di computazione deterministica accettante per concludere che $w \in L(P)$. Consideriamo dunque il ramo di computazione deterministica che itera h volte nello stato q_1, k volte nello stato q_2, l volte nello stato q_3 e m volte nello stato q_4. Il passaggio da q_i a q_{i+1} non dipende in effetti dall'input o dal contenuto dello stack, dunque è solo necessario verificare che le transizioni che portano da q_i a q_i siano legali, per ogni $1 \leq i \leq 4$. In effetti, vi possono essere h transizioni da q_1 a q_1 perché in ciascuna di essi viene letto un carattere '0' dalla parte iniziale 0^h di w. Ciascuna di queste transizioni aggiunge un simbolo sullo stack. Analogamente, vi possono

essere k transizioni da q_2 a q_2 perché in ciascuna di esse viene letto un carattere '1' dalla successiva parte 1^k di w; ciascuna di queste aggiunge un simbolo sullo stack. Quando poi si transita in q_3 si cominciano a rimuovere simboli dallo stack, ma continua ad essere possibile effettuare l transizioni da q_3 a q_3 perché si legge la parte 0^l di w, e contemporaneamente si tolgono $l \leq h + k$ simboli dallo stack. Infine è possibile effettuare m transizioni da q_4 a q_4 leggendo la parte 1^m di w e togliendo dallo stack i rimanenti $h + k - l = m$ simboli. Al termine la cima dello stack contiene '$' e si può quindi transitare nello stato di accettazione q_A. Pertanto, $w \in L(P)$.

Per la direzione contraria, supponiamo che $w \in L(P)$. L'accettazione di w da parte di P implica che deve esistere un ramo di computazione deterministica di $P(w)$ che, partendo da q_0, legga interamente la stringa in input w e termini nello stato q_A; pertanto, in questo ramo di computazione deterministica si devono attraversare nell'ordine gli stati da q_0 a q_A. Sia dunque h il numero di transizioni da q_1 a q_1, k il numero di transizioni da q_2 a q_2, l il numero di transizioni da q_3 a q_3, e m il numero di transizioni da q_4 a q_4. Per poter effettuare le $h \geq 0$ transizioni da q_1 a q_1 è necessario leggere 0^h dalla parte iniziale dell'input, quindi $w = 0^h \cdots$. Allo stesso tempo vengono aggiungi h simboli nello stack. Per poter effettuare le $k \geq 0$ transizioni da q_2 a q_2 è necessario leggere 1^k dalla parte successiva dell'input, quindi $w = 0^h 1^k \cdots$; contemporanemente si aggiungono altri k simboli allo stack. Per poter effettuare le $l \geq 0$ transizioni da q_3 a q_3 è necessario leggere 0^l dalla parte successiva dell'input, quindi $w = 0^h 1^k 0^l \cdots$; inoltre è necessario togliere l simboli dallo stack, quindi necessariamente $l \leq h + k$, altrimenti le transizioni non potrebbero essere eseguite. Per poter effettuare le $m \geq 0$ transizioni da q_4 a q_4 è necessario leggere 0^m dalla parte successiva dell'input, quindi $w = 0^h 1^k 0^l 1^m \cdots$; inoltre si devono togliere m simboli dallo stack, quindi necessariamente $m \leq h + k - l$. Infine, per poter effettuare la transizione finale da q_4 a q_A è necessario che sulla cima dello stack si trovi '$', e dunque $m = h + k - l$, ossia $h + k = l + m$. Inoltre, poiché w è accettata e q_A non ha transizioni che consumano altri simboli dell'input, $w = 0^h 1^k 0^l 1^m$. Pertanto, $w \in A$.

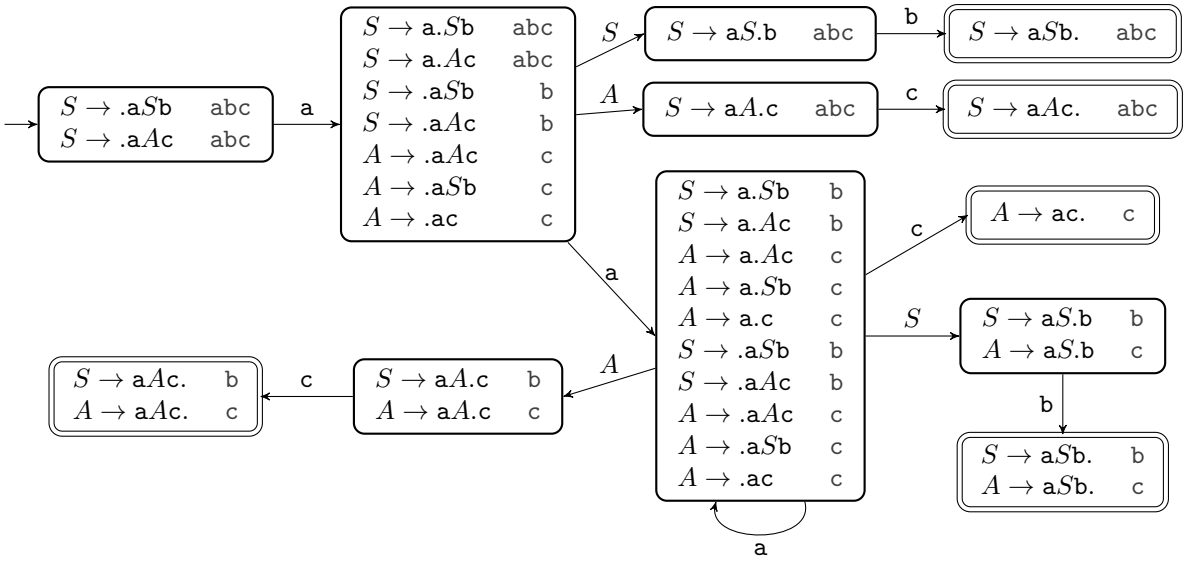
Avendo dimostrato che $A \subseteq L(P) \subseteq A$, ne consegue che $A = L(P)$. Pertanto A è un linguaggio libero dal contesto.

Esercizio 3 [6] Determinare se la grammatica G con variabile iniziale S e regole

$$S \to \mathtt{a}\, S\, \mathtt{b} \mid \mathtt{a}\, A\, \mathtt{c}, \quad A \to \mathtt{a}\, A\, \mathtt{c} \mid \mathtt{a}\, S\, \mathtt{b} \mid \mathtt{a}\, \mathtt{c}$$

è LR(0), LR(1) oppure né LR(0) né LR(1).

Soluzione: Per verificare se G è LR(1) eseguiamo il DK_1-test. L'automa così costruito potrà essere utilizzato anche per determinare se G è LR(0).



L'automa evidenzia che G non è LR(0), ossia non è deterministica; infatti esistono due stati di accettazione contenenti ciascuno più di una regola completata. D'altra parte, l'automa dimostra che G è LR(1), poiché il DK_1-test ha successo. Infatti, in due stati di accettazione esiste una sola regola completata e nessun'altra regola; negli altri due stati di accettazione esistono due regole completate, che però non sono consistenti, in quanto i rispettivi simboli di lookahead non si sovrappongono.

Esercizio 4 [6.5] Sia $A \setminus B = \{x \in A \,|\, x \notin B\}$. Consideriamo A Turing-riconoscibile (ossia ricorsivamente enumerabile) e B decidibile. Dimostrare che (a) $A \setminus B$ è Turing-riconoscibile, (b) $A \setminus B$ non è necessariamente decidibile, e (c) $B \setminus A$ non è necessariamente Turing-riconoscibile.

Soluzione: (a) Poiché A è Turing-riconoscibile, esiste un enumeratore E che stampa tutti i suoi elementi. Poiché B è decidibile, esiste una TM D che decide sull'appartenza o meno dei suoi elementi. A partire da E e D possiamo derivare il seguente enumeratore E':

E'="On empty input:
1. Run the emulator E, and for each string w on the print tape:
   2. Run the decisor D on input w
   3. If $D(w)$ rejects, then print w."

Se $w \in A \setminus B$, allora $w \in A$, dunque dopo un numero finito di passi l'emulatore E deve stampare w. Nel passo 2 il decisore D deve rifiutare dopo un numero finito di passi, in quanto $w \notin B$, dunque la stringa w verrà stampata anche da E'. Viceversa, se E' stampa una stringa w, allora innanzi tutto questa deve essere stampata anche da E nel passo 1, e successivamente

deve essere rifiutata dal decisore D. Pertanto $w \in A$ e $w \notin B$, ossia $w \in A \setminus B$. Concludiamo che E' è un enumeratore per $A \setminus B$, e quindi che $A \setminus B$ è Turing-riconoscibile.

(b) Dimostriamo l'asserto con un controesempio: sia A un linguaggio Turing-riconoscibile ma non decidibile, ad esempio $\mathcal{A}_{\text{TM}}$, e sia $B = \emptyset$ l'insieme vuoto (che è banalmente decidibile). Poiché $A \setminus B = A$, $A \setminus B$ non è necessariamente decidibile.

(c) Dimostriamo l'asserto con un controesempio: sia A un linguaggio Turing-riconoscibile ma non decidibile, ad esempio $\mathcal{A}_{\text{TM}}$, e sia $B = \Sigma^*$, ove Σ è l'alfabeto su cui insiste il linguaggio A; naturalmente B è un linguaggio decidibile. Poiché $B \setminus A = A^c$ è il complemento del linguaggio A, esso non può essere Turing-riconoscibile. Se infatti lo fosse, allora lo sarebbero sia A che A^c, e dunque A sarebbe decidibile. Quindi $B \setminus A$ non è necessariamente Turing-riconoscibile.

Esercizio 5 [7] È decidibile il problema di stabilire se una data macchina di Turing U è universale (ossia tale che per ogni TM M e input w, $U(\langle M, w\rangle) = M(w)$)? Giustificare la risposta con una dimostrazione.

Soluzione: Formuliamo il problema tramite un linguaggio $\mathcal{U}$ così definito:

$$\mathcal{U} = \{\langle U\rangle \mid U \text{ is a universal TM}\}.$$

Apparentemente sembra possibile applicare il Teorema di Rice per dimostrare l'indecidibilità di $\mathcal{U}$. Infatti, la proprietà che caratterizza il linguaggio $\mathcal{U}$ non è banale, poiché esistono certamente sia TM che sono universali che TM che non lo sono. Le difficoltà nascono quando si debba verificare che la proprietà che definisce $\mathcal{U}$ sia caratteristica del linguaggio degli elementi di $\mathcal{U}$. Infatti, gli elementi di $\mathcal{U}$ non sono TM che decidono un linguaggio, ossia che accettano o rifiutano una stringa in input, e pertanto non possiamo applicare la definizione di "linguaggio associato alla TM" come l'insieme delle stringhe accettate dalla TM. In altri termini, supponiamo che $U \in \mathcal{U}$: per definizione di TM universale, per ogni TM M e input w, $U(\langle M, w\rangle) = M(w)$, e non ha senso definire $\langle M, w\rangle$ come "accettato" o "rifiutato" da U. Dunque non ha senso definire il linguaggio associato ad U.

Evitiamo dunque di applicare il Teorema di Rice e procediamo invece con una dimostrazione per assurdo. Supponiamo che $\mathcal{U}$ sia decidibile, e quindi che esista per esso un decisore D. Consideriamo anche una TM universale U (ossia un elemento $U \in \mathcal{U}$). Dalle TM D e U possiamo determinare la seguente TM:

T="On input $\langle M, w\rangle$, where M is a TM and w is a string:
1. Build the encoding $\langle U'\rangle$ of the following TM:
   U' = "On input $\langle N, z\rangle$, where N is a TM and z is a string:
   (a) Run M on w
   (b) If $M(w)$ rejects, then halt
   (c) Run U on $\langle N, z\rangle$."

2. Run the decisor D on input $\langle U' \rangle$
3. If $D(\langle U' \rangle)$ accepts, then accept
4. Otherwise, reject."

Supponiamo che la computazione $M(w)$ termini accettando. Allora in una generica computazione $U'(\langle N, z \rangle)$ il passo (a) si completa in tempo finito, la condizione al passo (b) non è verificata, e quindi viene eseguito il passo (c); pertanto $U'(\langle N, z \rangle) = U(\langle N, z \rangle)$. Di conseguenza, nel passo 2 della computazione $T(\langle M, w \rangle)$ il decisore D accetta l'input $\langle U' \rangle$, perché U' emula la macchina di TM universale U, e quindi anche U' è universale, ossia $U' \in \mathcal{U}$. Perciò $T(\langle M, w \rangle)$ accetta al passo 3.

Supponiamo al contrario che la computazione $M(w)$ non accetti, ossia che termini rifiutando oppure non termini. La generica computazione $U'(\langle N, z \rangle)$ non arriva mai ad eseguire il passo (c), in un caso perché il passo (a) non termina, nell'altro perché la condizione al passo (b) è soddisfatta e quindi U' termina immediatamente. Di conseguenza, $U(\langle N, z \rangle)$, per ogni generico input $\langle N, z \rangle$, non emula il comportamento di una TM universale. Così $U' \notin \mathcal{U}$ e $D(\langle U' \rangle)$ termina rifiutando; di conseguenza anche $T(\langle M, w \rangle)$ rifiuta al passo 4.

In conclusione, la TM T è un decisore per il linguaggio $\mathcal{A}_{\text{TM}}$; questa è una contraddizione poiché tale linguaggio non è decidibile. L'assurdo deriva dall'aver supposto che $\mathcal{U}$ è decidibile, dunque resta dimostrata la sua indecidibilità.

Esercizio 6 [7] Dimostrare che se P=NP allora ogni linguaggio in P diverso da $\emptyset$ e da Σ^* è NP-completo.

Soluzione: La dimostrazione è in effetti molto semplice considerando che una riduzione polinomiale è basata su una macchina di Turing deterministica che ha la capacità di risolvere direttamente i problemi in P, e quindi in NP, poiché assumiamo che P=NP.

Formalmente, consideriamo un qualunque linguaggio $A \in$ P diverso dagli insiemi banali, ossia dall'insieme vuoto $\emptyset$ e dall'insieme contenente tutte le stringhe Σ^*, e dimostriamo che A è NP-completo. Innanzi tutto dobbiamo provare che $A \in$ NP, ma questo è immediato perché per ipotesi $A \in$ P e P=NP. Mostriamo adesso che A è NP-hard. Sia dunque $B \in$ NP. Poiché P=NP, $B \in$ P, dunque esiste una DTM M che decide B. Trasformiamo M in un'altra DTM N che, per ogni istanza x, simula l'esecuzione di $M(x)$. Se $M(x)$ accetta, N si ferma lasciando sul nastro la codifica di un elemento $I_y \in A$ (esiste certamente perché $A \neq \emptyset$). Se invece $M(x)$ rifiuta, N si ferma lasciando sul nastro la codifica di un elemento $I_n \notin A$ (esiste certamente perché $A \neq \Sigma^*$). La DTM N esegue in tempo polinomiale e calcola una istanza-sì di A per ogni istanza-sì di B, ed una istanza-no di A per ogni istanza-no di B. Dunque N costituisce una riduzione polinomiale da B ad A. Poiché B è un generico problema in NP, A è NP-hard. La conclusione è che A è NP-completo.