

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 21 giugno 2019

Esercizio 1 [6] Siano $L_1, L_2 \subseteq \Sigma^*$ linguaggi regolari; dimostrare che

$$L_1 \setminus L_2 = \{w \in L_1 \mid w \notin L_2\}$$

è un linguaggio regolare.

Soluzione: La dimostrazione è immediata considerando che $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ e che la classe dei linguaggi regolari è chiusa rispetto alle operazioni di complemento ed intersezione.

Un'altra dimostrazione consiste nel derivare un DFA N che riconosce il linguaggio differenza a partire dai DFA $M = (Q, \Sigma, \delta, q_0, F)$ e $M' = (Q', \Sigma, \delta', q'_0, F')$ che riconoscono rispettivamente L_1 e L_2 . In particolare, $N = (Q \times Q', \Sigma, \delta'', (q_0, q'_0), F'')$, ove la funzione di transizione $\delta'': Q \times Q' \times \Sigma \rightarrow Q \times Q'$ è definita come $\delta''(q, q', x) = (\delta(q, x), \delta'(q', x))$, e l'insieme di stati di accettazione è $F'' = \{(q, q') \mid q \in F, q' \notin F'\}$.

Esercizio 2 [6] Sia $A = \{ww' \mid w, w' \in \{0, 1\}^*, w' = \overline{w}^R\}$, (w' è la stringa ottenuta da w rovesciando l'ordine dei bit e complementandone il valore). Dimostrare che A è un CFL esibendo una grammatica opportuna.

Soluzione: La più semplice CFG G che genera le stringhe di A è:

$$S \rightarrow 0S1 \mid 1S0 \mid \epsilon$$

Infatti è immediato verificare che ogni stringa terminale generata da G è costituita da un numero pari di bit, quindi può essere decomposta in due sottostringhe w e w' di pari lunghezza. Il primo bit di w e l'ultimo bit di w' devono essere diversi perché generati nell'applicazione di una singola regola $0S1$ o $1S0$. Allo stesso modo, il secondo bit di w ed il penultimo bit di w' devono essere differenti perché generati nell'applicazione di un'altra regola, e così via. Formalmente, ogni derivazione da S lunga un passo che genera una stringa terminale deve produrre $\epsilon \in A$. Sia dunque assunto per vero che ogni derivazione da S lunga n passi che genera una stringa terminale produca un elemento di A . In una derivazione da S lunga $n+1$ passi che genera una stringa terminale z deve iniziare applicando una regola $S \rightarrow xS\bar{x}$, e

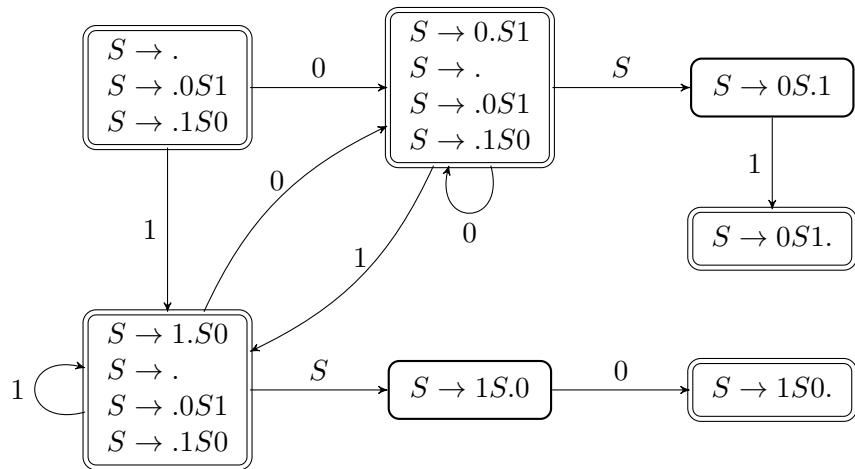
continuare derivando da S una stringa terminale in n passi. Per ipotesi induttiva dunque $z = xv\bar{v}^R\bar{x} = w\bar{w}^R \in A$.

Viceversa, consideriamo una qualunque stringa binaria $w\bar{w}^R$, e dimostriamo per induzione sulla lunghezza della stringa w che può essere generata da G . Se $|w| = 0$, l'asserto segue dalla regola $S \rightarrow \epsilon$. Supponiamo che l'asserto sia vero per $|w| \leq i$, e consideriamo la stringa $w\bar{w}^R$, ove $|w| = i + 1$. Si ha dunque che $w\bar{w}^R = xv\bar{v}^R\bar{x}$, con $x \in \{0, 1\}$. Per costruzione, è possibile derivare da S la stringa non terminale $xS\bar{x}$; per ipotesi induttiva, è possibile derivare da S la stringa $v\bar{v}^R$, poiché $|v| = i$; pertanto, la grammatica può generare tutte le stringhe $w\bar{w}^R$ arbitrariamente lunghe.

Esercizio 3 [7] Dimostrare che la grammatica ottenuta nel precedente esercizio è non deterministica.

Soluzione: Intuitivamente, ogni grammatica che genera le stringhe del linguaggio $A = \{ww' \mid w, w' \in \{0, 1\}^*, w' = \bar{w}^R\}$ deve essere nondeterministica perché un PDA che riconosce le stringhe del linguaggio deve necessariamente utilizzare il nondeterminismo per indovinare l'occorrenza nella sequenza di input del primo simbolo di $w' = \bar{w}^R$.

Formalmente, eseguiamo il DK-test sulla grammatica G , ottenendo:



Poiché almeno uno degli stati terminali contiene un “dot” seguito da un simbolo terminale, il test è fallito e quindi la grammatica è non deterministica.

Esercizio 4 [10] Sia $B = \{ww' \mid w, w' \in \{0, 1\}^*, w' = \bar{w}\}$, (w' è la stringa ottenuta da w complementando ogni bit). Dimostrare che B non è un CFL.

Soluzione: Per assurdo, supponiamo che B sia CFL e che quindi valga per esso il “Pumping lemma”. Sia dunque p la “pumping length” per B . In questo esercizio la scelta della stringa che contraddice le conclusioni del lemma deve essere fatta con attenzione. Ad esempio, la stringa $0^p 1^p 1^p 0^p$ non andrebbe bene, perché può essere pompata ponendo $u = 0^{p-1}$, $v = 0$, $x = \epsilon$, $y = 1$, $z = 1^{2p-1} 0^p$: infatti, $uv^i xy^i z = 0^{p-1} 0^i 1^i 1^{2p-1} 0^p = 0^{p+i-1} 1^p 1^{p+i-1} 0^p$.

Consideriamo invece la stringa $s = 0^p 1^p 0^p 1^p 0^p 1$: evidentemente $s \in B$ (in quanto $\overline{0^p 1^p 0} = 1^p 0^p 1$), quindi deve essere possibile determinare una suddivisione $s = uvxyz$ tale che per ogni $i \geq 0$, $uv^i xy^i z \in B$, $|vy| > 0$ e $|vxy| \leq p$. Distinguiamo i seguenti casi:

1. Se vy non contiene esattamente lo stesso numero di zero ed uno, pompando verso il basso o verso l’alto si otterebbe una stringa con una eccedenza di zero od uno, che dunque non potrebbe far parte di B . Ciò implica anche che $|vy| > 1$.
2. Se la sottostringa vxy fosse interamente nella prima metà di s , allora pompando verso il basso si otterebbe una stringa uxz in cui uno dei bit della seconda occorrenza di 1^p in s finirebbe nell’ultima posizione della prima metà di uxz (in quanto $|vxy| \leq p$ e $|vy| > 1$). Pertanto uxz non può essere in B perché il bit in ultima posizione della prima metà deve essere 0.
3. Analogamente, la sottostringa vxy non può essere interamente nella seconda metà di s , altrimenti pompando verso il basso uno dei bit della prima occorrenza di 1^p in s finirebbe nell’ultima posizione della prima metà di uxz , e ciò significa che tale stringa non potrebbe far parte di B perché essa termina con 1.
4. Pertanto, vxy deve contenere sia un bit della prima metà di s che un bit della seconda metà. Poichè però vy deve contenere un ugual numero di zero ed uno, l’unica possibilità è che vy sia o la stringa 01 oppure la stringa 10, ove lo 0 è quello in ultima posizione della prima metà di s . Pompando verso il basso si ottiene una stringa in cui nell’ultima posizione della prima metà vi è un 1, dunque tale stringa non può far parte di B .

Poichè non è possibile suddividere la stringa s in accordo al pumping lemma, concludiamo che il pumping lemma non può essere applicato, e pertanto che l’ipotesi che B fosse un CFL è falsa.

Esercizio 5 [11] Il problema DOMINATING SET è il seguente: dato un grafo non diretto $G = (V, E)$, ed un numero intero k , determinare se esiste un sottoinsieme di nodi $V' \subseteq V$ di cardinalità k tale che ogni nodo del grafo o appartiene a V' oppure è adiacente ad un nodo in V' (o entrambe le cose). Dimostrare che DOMINATING SET è NP-completo descrivendo una riduzione polinomiale da VERTEX COVER.

Soluzione: Per prima cosa dimostriamo che DOMINATING SET (DS) appartiene a NP. Il problema è polinomialmente verificabile perché ogni istanza che fa parte del linguaggio ha come certificato il sottoinsieme di nodi del grafo che costituisce il dominating set: è certamente di dimensione non superiore al numero di nodi del grafo, e verificare che ogni nodo del grafo fa parte od è adiacente al sottoinsieme può essere facilmente realizzato da un algoritmo che esegue in tempo polinomiale nella dimensione dell'istanza del problema:

M= “On input $\langle G = (V, E), k, V' \rangle$, where G is a graph, $V' \subseteq V$:

1. if $|V'| > k$: reject
2. for each node v in V :
 3. if $v \in V'$ then continue with next node in step 1
 4. for each edge $e \in E$:
 5. if e links v to a node in V' , continue with next node in step 1
 6. Reject, because node v is not dominated by V'
 7. Accept, because all nodes in V are dominated by V' ”

Il numero totale di passi eseguiti dall'algoritmo è $O(n m n) = O(n^4)$, ove $n = |V(G)|$ e $m = |E(G)| = O(n^2)$.

Consideriamo ora una riduzione polinomiale da VERTEX COVER (VC) a DS. Sia $(G = (V, E), k)$ una istanza di VC. Costruiamo un nuovo grafo $G' = (V', E')$ in questo modo: $V' = V \cup V_E$ è costituito dai nodi di G e da un nodo v_e per ciascun arco $e \in E$; in totale quindi $|V'| = n + m = |V| + |E|$. $E' = E \cup E_V$ è costituito dagli archi di G e, per ciascun nodo $v_e \in V_E$, da due archi (v_e, v) e (v_e, w) ove $e = (v, w)$; in totale quindi $|E'| = 3m = 3|E|$. Possiamo dimostrare che $(G, k) \in \text{VC}$ se e solo se $(G', k + s) \in \text{DS}$, ove s è il numero di nodi $S \subseteq V$ “isolati” (senza archi incidenti).



Supponiamo che $(G, k) \in \text{VC}$; dunque esiste $U \subseteq V$ tale che $|U| = k$ ed ogni arco di G è incidente ad almeno un nodo di U . Consideriamo il sottoinsieme di nodi $U' = U \cup S$ in G' (esiste perché tutti i nodi di G sono anche nodi di G'), e dimostriamo che è un dominating set di dimensione $k + s$. Infatti, sia $x \in V(G') = V \cup V_E$: se $x \in V$, allora o $x \in S$, e dunque $x \in U'$, oppure esiste un arco $e \in E$ incidente su x . Poiché U è un ricoprimento degli archi in G , esiste un nodo $y \in U$ tale che $e = (x, y)$; pertanto, il nodo x è dominato dal nodo $y \in U'$.

Se invece $x \in V_E$, allora $x = v_e$ per un certo arco $e \in E$: dunque, U deve contenere un nodo y che ricopre e ; allora, per costruzione di E_V , $(y, v_e) \in E_V$, e quindi x è dominato da $y \in U'$.

Per la direzione opposta, supponiamo che $(G', k + s) \in \text{DS}$, e quindi esiste un sottoinsieme U' , con $|U'| = k + s$, che domina ogni nodo di G' . Risulta evidente che $S \subseteq U'$, perché l'unico modo per dominare nodi isolati è inserirli nel sottoinsieme dominante. Un'altra osservazione è che se U' contiene un qualunque nodo $v_e \in V_E$, è possibile sostituire in U' il nodo v_e con uno qualunque dei due nodi v, w tali che $e = (v, w)$. Infatti per costruzione di G' il nodo v_e può dominare solo se stesso, v e w ; ma v (o equivalentemente w) domina almeno se stesso, w e v_e , quindi sostituendo v_e con v si ottiene un dominating set di dimensione pari od inferiore a quella di U' che domina almeno lo stesso insieme di nodi di G' . Consideriamo dunque il dominating set U'' in cui ogni nodo v_e è stato sostituito da un nodo in V come appena descritto, e sia $U = U'' \setminus S$. Il sottoinsieme U ha cardinalità $\leq k$, è un sottoinsieme di nodi di G , ed è un ricoprimento degli archi in G . Infatti, sia $e \in E$, con $e = (v, w)$. Poiché il nodo v_e deve essere dominato da qualche nodo in U'' , per costruzione v, w , od entrambi appartengono al sottoinsieme dominante U'' . Naturalmente $v, w \notin S$, quindi v, w oppure entrambi, appartengono ad U . Pertanto, l'arco e risulta ricoperto da un elemento di U .

Abbiamo dunque dimostrato che la trasformazione da (G, k) a $(G', k + s)$ è una riduzione tra problemi. È inoltre evidente che tale trasformazione può essere costruita in tempo polinomiale. Pertanto, $\text{VC} \leq_m \text{DS}$, e quindi DS è NP-hard in quanto VC è NP-completo. Ciò conclude la dimostrazione di NP-completezza di DS .

Automi e Linguaggi (M. Cesati)

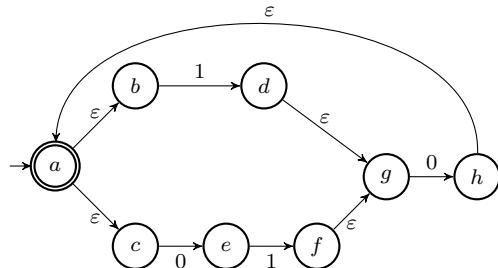
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 5 luglio 2019

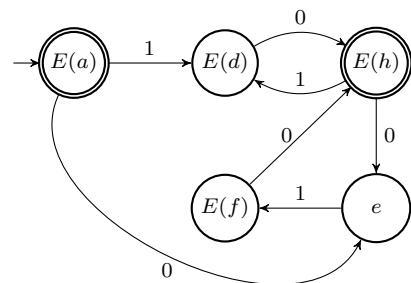
Esercizio 1 [5] Si consideri l'espressione regolare $R = ((1 \cup (01))0)^*$. Costruire un DFA che riconosce il linguaggio generato da R .

Soluzione: Il linguaggio è molto semplice e quindi l'automa deterministico può essere costruito direttamente senza problemi. Mostriamo però il procedimento meccanico che dapprima deriva da R un NFA N che riconosce il linguaggio, poi trasforma N in un equivalente DFA D .

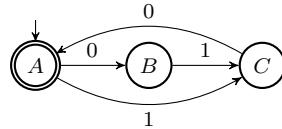
Adottando le regole di composizione degli NFA per le operazioni di concatenazione, unione, e star, ed eliminando qualche transizione ε ridondante, si costruisce lo NFA N seguente:



Calcoliamo gli insiemi chiusura di ciascuno stato rispetto alle transizioni ε : $E(a) = \{a, b, c\}$, $E(b) = \{b\}$, $E(c) = \{c\}$, $E(d) = \{d, g\}$, $E(e) = \{e\}$, $E(f) = \{f, g\}$, $E(g) = \{g\}$, $E(h) = \{h, a, b, c\}$. La procedura di conversione dall'NFA N al DFA D produce il seguente automa:



È immediato osservare che le transizioni uscenti dagli stati marcati con $E(a)$ e con $E(h)$ sono identiche, così come quelle uscenti dagli stati marcati con $E(d)$ e $E(f)$; dunque possiamo semplificare l'automa ed ottenere il DFA seguente:



Esercizio 2 [7] Sia C un linguaggio context-free e sia R un linguaggio regolare. Dimostrare che $C \cap R$ è un linguaggio context-free.

Soluzione: Cominciamo con l'osservare che considerazioni insiemistiche sulla natura di $C \cap R$ non portano da nessuna parte. Ad esempio, a nulla serve enunciare che $C = (C \cap R) \cup (C \cap \overline{R})$ e che l'unione di linguaggi CFL è CFL. Infatti, ciò non significa che un linguaggio CFL esprimibile come $C = A \cup B$ implichì necessariamente che A o B siano CFL. Od ancora, il fatto che il linguaggio regolare R sia anche CFL non implica che il suo sottoinsieme $C \cap R$ sia necessariamente CFL o regolare; questa linea di ragionamento, se applicata a due linguaggi in CFL, porterebbe a concludere che la loro intersezione è in CFL, cosa manifestamente non vera.

Poiché C è CFL, esiste un PDA $P = (Q_P, \Sigma_P, \Gamma_P, \delta_P, q_0^P, F_P)$ che accetta tutti e solo gli elementi di C . Analogamente, poiché R è regolare, esiste un DFA $D = (Q_D, \Sigma_D, \delta_D, q_0^D, F_D)$ che accetta tutti e solo gli elementi di R . L'idea della dimostrazione è di esibire un altro PDA che si comporta esattamente come P ma contemporaneamente simula l'esecuzione di D , ed accetta se e solo se sia P che D terminano in stato di accettazione.

Consideriamo dunque un altro PDA $P' = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ove $Q = Q_P \times Q_D$, $\Sigma = \Sigma_P \cup \Sigma_D$, $\Gamma = \Gamma_P$, $q_0 = (q_0^P, q_0^D)$ e $F = F_P \times F_D$.

Per ogni stato $q = (q^P, q^D) \in Q = Q_P \times Q_D$ e simboli $x \in \Sigma_\varepsilon$, $y \in \Gamma_\varepsilon$, sia

$$\delta(q, x, y) = \begin{cases} \{(q'^P, q^D), y'\} \mid (q'^P, y') \in \delta_P(q^P, \varepsilon, y) \} & \text{se } x = \varepsilon \\ \{(q^P, q'^D), y'\} \mid (q^P, y') \in \delta_P(q^P, x, y) \text{ e } \delta_D(q^D, x) = q'^D\} & \text{se } x \in \Sigma_P \cap \Sigma_D \\ \emptyset & \text{altrimenti.} \end{cases}$$

Si osservi che se P' legge un simbolo di input che non appartiene all'alfabeto di R allora non può accettare l'intera stringa perché non può applicare alcuna transizione. P' può anche effettuare delle transizioni senza consumare simboli di input, ed in questo caso lo stato di D registrato negli stati di P' non cambia. Infine, se il simbolo di input x appartiene sia

a Σ_P che Σ_D , allora può essere applicata una qualunque delle transizioni definite da P e contemporaneamente la transizione definita da D .

Supponiamo che $w \in C \cap R$; allora esiste una sequenza di transizioni di P che porta $P(w)$ in uno stato in F_P , ed esiste una sequenza di transizioni di D che porta $D(w)$ in uno stato di F_D . Pertanto per costruzione esiste una sequenza di transizioni di P' che porta $P'(w)$ in uno stato di $F = F_P \times F_D$. Viceversa, se P' accetta una stringa w , allora per costruzione di P' sia $P(w)$ che $D(w)$ accettano, e dunque $w \in C \cap R$.

Poiché esiste un PDA che riconosce $C \cap R$, concludiamo che esso è CFL.

Esercizio 3 [6] Dimostrare che il linguaggio

$$C = \left\{ w \in \{1, +, =\}^* \mid w = 1^i + 1^j = 1^{i+j}, i, j > 0 \right\}$$

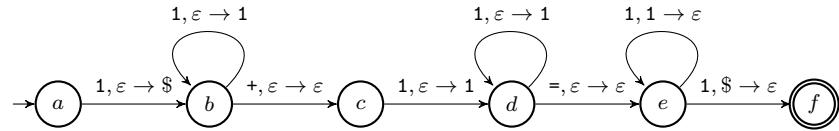
è DCFL esibendo un opportuno PDA deterministico.

Soluzione: L'automa a pila deterministico può svolgere efficacemente il riconoscimento delle stringhe di C utilizzando lo stack per contare il numero di 1 degli addendi che precedono il simbolo $=$ e confrontandolo con il numero di 1 che seguono il simbolo $=$.

Formalmente, la descrizione ad alto livello del DPDA N che decide C è la seguente:

- $N =$ “On input w , where $w = w_1 w_2 \cdots w_m \in \{1, +, =\}^*$:
1. if $w_1 \neq 1$ then reject
 2. while iterating over the symbols w_1, w_2, \dots, w_m of the input:
 3. if $w_i \neq 1$ then break the loop and go to step 5
 4. push 1 onto the stack
 - 5 if $w_i \neq +$ or $w_{i+1} \neq 1$ then reject
 6. while iterating over the remaining symbols w_{i+1}, \dots, w_m of the input:
 7. if $w_j \neq 1$ then break the loop and go to step 9
 8. push 1 onto the stack
 9. if $w_j \neq =$ then reject
 10. while iterating over the remaining symbols w_{j+1}, \dots, w_m of the input:
 11. if $w_k \neq 1$ then reject
 12. if the stack is empty then reject
 13. pop 1 from the stack
 14. if the stack is empty then accept, otherwise reject.”

In alternativa, e più semplicemente, possiamo descrivere il PDA tramite il grafo dei suoi stati e delle sue transizioni. Ad esempio, il seguente automa N a 6 stati riconosce C :



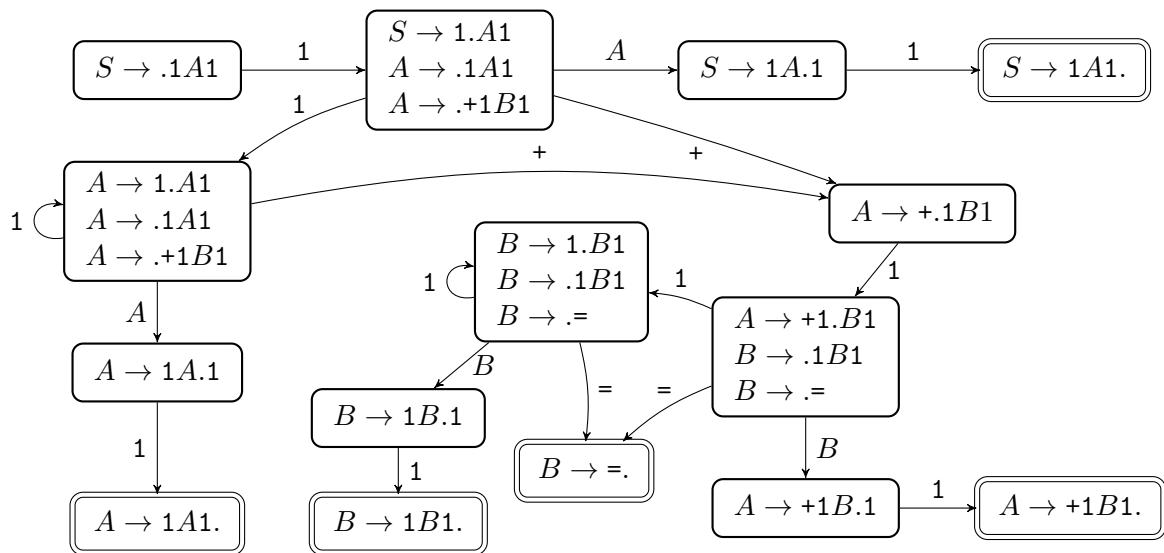
In entrambe le descrizioni di N è immediato verificare che N è un automa a pila deterministico che accetta la stringa in input se e solo se essa è della forma $1^+ + 1^+ = 1^+$ ed inoltre il numero di 1 che precedono il simbolo $=$ è uguale al numero di 1 che lo seguono. Pertanto, C è DCFL.

Esercizio 4 [11] Determinare una grammatica context-free deterministica per il linguaggio C dell'esercizio precedente, eseguendo su di essa il DK-test.

Soluzione: Una grammatica context-free che genera il linguaggio C è la seguente:

$$\begin{aligned} S &\rightarrow 1A1 \\ A &\rightarrow 1A1 \quad | \quad +1B1 \\ B &\rightarrow 1B1 \quad | \quad = \end{aligned}$$

Dimostriamo per prima cosa che questa grammatica è deterministica applicando il DK-test. Si ottiene il seguente diagramma:



Poiché tutti gli stati terminali contengono una singola regola completata, il test ha successo e quindi la grammatica è deterministica.

Dimostriamo ora che la grammatica genera tutti e soli gli elementi di C . Infatti, sia $w \in C$; allora esistono $i, j > 0$ tali che $1^i + 1^j = 1^{i+j}$. Possiamo ridurre la stringa w in questo modo:

$$1^i + 1^j = 1^j 1^i = 1^i + 1 \overbrace{1^{j-1} = 1^{j-1}}^{B\dots} 1 1^i \xrightarrow{*} 1^i \overbrace{+ 1B1}^A 1^i \xrightarrow{*} 1 \overbrace{1^{i-1} A 1^{i-1}}^{A\dots} 1 \xrightarrow{*} \overbrace{1 A 1}^S \xrightarrow{*} S.$$

Viceversa, sia w una stringa terminale generata dalla grammatica deterministica ($S \Rightarrow w$), e supponiamo per assurdo che $w \notin C$. Allora deve verificarsi uno dei seguenti casi:

1. w non inizia con 1: si ha una contraddizione perché la variabile iniziale S può espandersi solo in $1A1$.
2. w non contiene $=$: si ha una contraddizione perché l'unico modo per ottenere una stringa terminale dalla grammatica è eliminare la variabile B con la regola $B \rightarrow =$. Ogni altra regola introduce un'altra variabile non terminale.
3. w non contiene $+$: si ha una contraddizione perché l'unico modo per ottenere una stringa terminale dalla grammatica è eliminare la variabile B con la regola $B \rightarrow =$, e l'unico modo per ottenere la variabile B è tramite la regola $A \rightarrow +1B1$.
4. in w c'è più di una occorrenza di $+$: si ha una contraddizione perché l'unica regola che genera $+$ può essere applicata una sola volta: S genera una sola A nella stringa, ogni A genera una sola A oppure una sola B (ed in questo caso genera anche $+$), e le regole di B non possono generare $+$.
5. in w c'è più di una occorrenza di $=$: si ha una contraddizione perché l'unica regola che genera $=$ può essere applicata una sola volta: S genera una sola A nella stringa, ogni A genera una sola A oppure una sola B , e la regola di B che genera $=$ produce necessariamente la stringa terminale.
6. in w , $=$ precede $+$: si ha una contraddizione perché l'unico modo per ottenere una stringa terminale è tramite l'applicazione di $B \rightarrow =$ preceduta dall'applicazione di $A \rightarrow +1B1$. Quindi $+$ deve precedere $=$.
7. in w , $=$ segue immediatamente $+$: si ha una contraddizione perché la regola $A \rightarrow +1B1$ che genera $+$ pone immediatamente dopo il simbolo 1.
8. w termina con $=$: si ha una contraddizione perché la regola che genera $=$ è $B \rightarrow =$, e B può essere generata solo dalle regole $A \rightarrow +1B1$ oppure $B \rightarrow 1B1$: in entrambi i casi la stringa termina con 1, non $=$.
9. $w = 1^i + 1^j = 1^h$ con $h > i + j$: si ha una contraddizione, perché riducendo la stringa si otterrebbe:

$$1^i + 1^j = 1^h \mapsto 1^i + 1^j B 1^h \xrightarrow{*} 1^i + 1B1 1^{h-j} \mapsto 1^i A 1^{h-j} \xrightarrow{*} 1A1 1^{h-(i+j)} \mapsto S 1^{h-(i+j)}$$

ed ovviamente non esisterebbe modo di generare la sequenza di 1 a destra della variabile iniziale S .

10. $w = 1^i + 1^j = 1^h$ con $h < i + j$: si ha una contraddizione, perché riducendo la stringa si otterrebbe:

$$1^i + 1^j = 1^h \mapsto 1^i + 1^j B 1^h \xrightarrow{*} 1^i + 1^{j-h} B 1$$

e poiché $j - h > 0$ non ci sarebbe modo di ridurre ulteriormente applicando la regola A .

Poiché ogni motivo di esclusione di w da C porta ad una contraddizione, se ne conclude che $w \in C$ per ogni $S \Rightarrow w$.

Esercizio 5 [11] Il problema SET COVER è il seguente: dato un insieme U di elementi, una collezione $\mathcal{C} = \{S_1, \dots, S_m\}$ di sottoinsiemi di U ($S_i \subseteq U$ per ogni i), ed un intero $k > 0$, determinare se è possibile selezionare k sottoinsiemi S_{i_1}, \dots, S_{i_k} nella collezione \mathcal{C} tali che $\cup_{j=1}^k S_{i_j} = U$. Dimostrare che SET COVER è NP-completo descrivendo una riduzione polinomiale da VERTEX COVER.

Soluzione: Per prima cosa dimostriamo che SET COVER (SC) appartiene a NP. Il problema è polinomialmente verificabile perché ogni istanza $\langle U, \mathcal{C}, k \rangle$ che fa parte del linguaggio ha come certificato la collezione di sottoinsiemi che ricopre U : è certamente di dimensione non superiore alla collezione di tutti i sottoinsiemi in $\mathcal{C} = \{S_1, \dots, S_m\}$, e verificare che ogni elemento di U fa parte di uno dei sottoinsiemi può essere facilmente realizzato da un algoritmo che esegue in tempo polinomiale nella dimensione dell'istanza del problema:

M = “On input $\langle U, \mathcal{C}, k, S'_1, \dots, S'_h \rangle$, where \mathcal{C} is a collection of subsets $S_i \subseteq U$:

1. if $h > k$ or $S'_j \notin \mathcal{C}$ for any j then reject
2. for each element $x \in U$:
 3. for each S'_j ($1 \leq j \leq h$):
 4. if $x \in S'_j$ then continue with next element in step 2
 5. Reject, because element x is not covered by any subset S'_j
 6. Accept, because all elements in U are covered by the subsets S'_j ”

Il numero di passi totali eseguito dall'algoritmo è in $O(|U| \cdot h \cdot \max_{j=1}^h |S'_j|) = O(|U|^2 \cdot h)$, ossia è polinomiale nella dimensione dell'istanza $O(h \cdot |U|)$.

Consideriamo ora una riduzione polinomiale da VERTEX COVER (VC) a SC. Sia $(G = (V, E), k)$ una istanza di VC. La corrispondente istanza di SC ha come insieme U un elemento x_e per ciascun arco e di G ($U = \{x_e \mid e \in E(G)\}$), e come collezione \mathcal{C} un sottoinsieme

S_v per ciascun nodo v di G contenente tutti gli elementi x_e tali che l'arco e è incidente su v :
 $\mathcal{C} = \{S_v \mid v \in V(G)\}$ con $S_v = \{x_e \in U \mid e = (v, w) \in E(G)\}$.

Supponiamo che $(G, k) \in \text{VC}$; dunque esiste $V' \subseteq V$ tale che $|V'| \leq k$ e V' copre tutti gli archi di G . Consideriamo dunque come sottoinsiemi ricoprenti per SC quelli corrispondenti agli elementi di V' . Sia x_e un qualunque elemento di U : poiché $e \in E(G)$, esiste un nodo v in V' tale che $e = (v, w)$. Ma allora $x_e \in S_v$ e S_v è tra i sottoinsiemi selezionati. Dunque esistono al più k sottoinsiemi tra quelli in \mathcal{C} che coprono U .

Per la direzione opposta, supponiamo che esista un sottoinsieme di al più k sottoinsiemi in \mathcal{C} che coprono U , e siano $S'_{v_1}, S'_{v_2}, \dots, S'_{v_k}$. Sia $V' = \{v_1, \dots, v_k\}$. Per costruzione dell'istanza di SC, $V' \subseteq V(G)$ ed ovviamente $|V'| \leq k$. Dimostriamo che V' è un vertex cover. Sia dunque $e \in E(G)$ un qualunque arco di G ; il corrispondente elemento $x_e \in U$ deve far parte di almeno uno dei sottoinsiemi S'_{v_i} . Per costruzione di S'_{v_i} allora $e = (v_i, w)$, e dunque il nodo $v_i \in V'$ copre l'arco e .

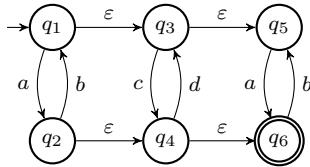
Abbiamo dunque dimostrato che la trasformazione da (G, k) a (U, S_1, \dots, S_m, k) è una riduzione tra problemi. È inoltre evidente che tale trasformazione può essere costruita in tempo polinomiale. Pertanto, $\text{VC} \leq_m \text{SC}$, e quindi SC è NP-hard, e di conseguenza NP-completo.

Automi e Linguaggi (M. Cesati)

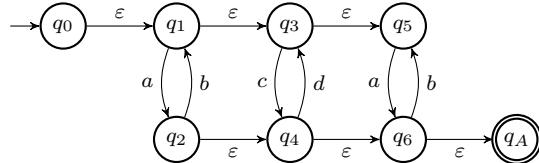
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 4 settembre 2019

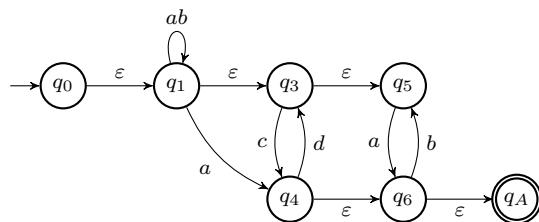
Esercizio 1 [8] Derivare l'espressione regolare che definisce il linguaggio riconosciuto dall'automa:



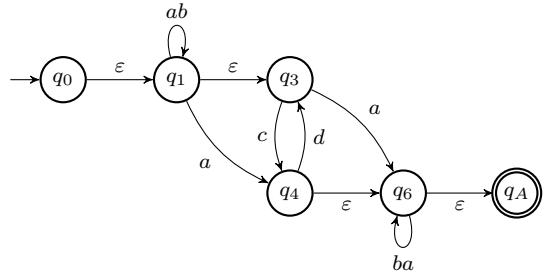
Soluzione: La derivazione di una espressione regolare comporta la trasformazione in un GNFA e la successiva eliminazione di tutti gli stati non iniziali e finali. Come primo passo consideriamo dunque un GNFA derivato dall'automa iniziale (tutti gli archi con etichetta \emptyset sono omessi dal grafico):



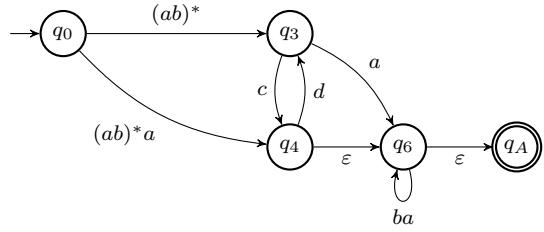
La scelta dell'ordine degli stati da eliminare è arbitraria, ma generalmente conviene iniziare dagli stati aventi un numero di archi entranti e/o uscenti piccolo. Cominciamo dallo stato q_2 : esso ha archi entranti da q_1 e archi uscenti verso q_1 e q_4 ; pertanto il nuovo automa ha etichette aggiornate per gli archi da q_1 e verso q_1 e q_4 :



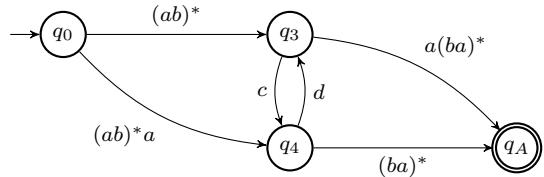
Eliminiamo ora q_5 , che ha archi entranti da q_3 e q_6 e uscenti verso q_6 :



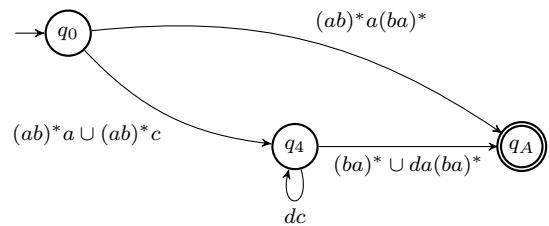
Lo stato q_1 ha un arco entrante da q_0 ed archi uscenti verso q_3 e q_4 :



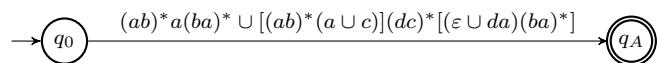
Lo stato q_6 ha archi entranti da q_3 e q_4 , ed un arco uscente verso q_A :



Lo stato q_3 ha archi entranti da q_0 e q_4 , ed archi uscenti verso q_4 e q_A :



Semplifichiamo l'etichetta $(ab)^*a \cup (ab)^*c$ in $(ab)^*(a \cup c)$, e l'etichetta $(ba)^* \cup da(ba)^*$ in $(\varepsilon \cup da)(ba)^*$. Rimuovendo l'ultimo stato q_4 si ottiene:



Semplificando ancora l'etichetta $(ab)^*a(ba)^* \cup [(ab)^*(a \cup c)](dc)^*[(\varepsilon \cup da)(ba)^*]$ si ottiene l'espressione regolare

$$(ab)^* [a \cup (a \cup c)(dc)^*(\varepsilon \cup da)] (ba)^*.$$

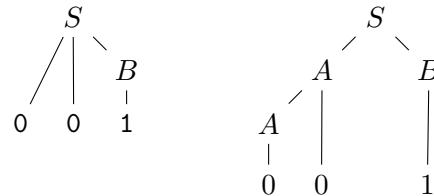
Esercizio 2 [6] Dimostrare che la grammatica context-free seguente è ambigua:

$$S \rightarrow AB \quad | \quad 00B \qquad A \rightarrow 0 \quad | \quad A0 \qquad B \rightarrow 1$$

Soluzione: Una grammatica si definisce ambigua se esistono due derivazioni “leftmost” (o equivalentemente “rightmost”) della stessa stringa terminale. Consideriamo dunque la stringa terminale 001: essa può essere prodotta con le seguenti due diverse derivazioni “leftmost”:

$$\begin{aligned} S &\rightarrow 00B \rightarrow 001 \\ S &\rightarrow AB \rightarrow A0B \rightarrow 00B \rightarrow 001 \end{aligned}$$

Equivalentemente, la grammatica è certamente ambigua perché la stringa terminale 001 può essere prodotta con due differenti alberi sintattici (“parse tree”):



Esercizio 3 [8] Dimostrare che il seguente linguaggio non è context-free:

$$A = \{w \in \{a, b, c\}^* \mid w \text{ ha un ugual numero di } a, b \text{ e } c\}$$

Soluzione: Una facile dimostrazione indiretta può essere basata su due risultati già stabiliti in precedenza. Il primo risultato è che l'insieme $\{0^n1^n2^n \mid n \geq 0\}$ non è un CFL (dimostrato durante le lezioni del corso), quindi l'insieme ad esso isomorfo $B = \{a^n b^n c^n \mid n \geq 0\}$ non è un CFL. Il secondo risultato è l'asserto di un esercizio di una sessione d'esami precedente, ossia che l'intersezione di un CFL ed un linguaggio regolare è un CFL. È facile verificare che $B = A \cap R$, ove R è il linguaggio generato dall'espressione regolare $a^*b^*c^*$. Pertanto, se per assurdo A fosse CFL, allora anche B dovrebbe esserlo, una contraddizione.

Per una dimostrazione diretta utilizziamo il pumping lemma. Supponiamo per assurdo che A sia un CFL, e sia p la corrispondente lunghezza garantita dal lemma. Scegliamo come stringa $s = a^p b^p c^p$; poiché vale il pumping lemma, deve esistere una suddivisione $s = uvxyz$ ove $|vxy| \leq p$, $|vy| > 0$, e $uv^i xy^i z \in A$ per ogni $i \geq 0$. Qualunque suddivisione per la quale vy non contenga un ugual numero di a , b e c non può preservare l'appartenza a A . D'altra parte, qualunque suddivisione tale che vy contenga sia a che b che c deve essere lunga almeno $p + 2$ simboli (un a , tutti i b ed un c). Pertanto risulterebbe $|vxy| > p$, mentre il pumping lemma garantisce l'esistenza di una suddivisione con $|vxy| \leq p$. La contraddizione deriva dall'aver supposto che A sia un CFL.

Esercizio 4 [9] Si consideri il linguaggio $U = \{\langle M \rangle \mid M \text{ è una TM e } n \in L(M)\}$ ove n è la codifica in binario del tuo numero di matricola universitaria. Il linguaggio U è decidibile? La risposta deve includere una dimostrazione.

Soluzione: Poiché il linguaggio è costituito da codifiche di macchine di Turing, è plausibile che ad esso possa applicarsi il teorema di Rice, che afferma che qualunque proprietà non banale relativa ai linguaggi riconosciuti da TM è indecidibile.

Per verificare se le ipotesi del teorema di Rice sono soddisfatte, consideriamo se la proprietà caratterizzante l'insieme U è banale o meno: nella definizione dell'insieme il numero di matricola corrisponde semplicemente ad una stringa di bit ben definita e costante. Quindi la proprietà non è banale, in quanto almeno un insieme (l'insieme vuoto) non include questa stringa binaria, mentre almeno un altro insieme ($\{0, 1\}^*$) la include. La seconda ipotesi del teorema di Rice è che la proprietà caratterizzante U deve riferirsi al linguaggio riconosciuto dalle macchine di Turing, e non alle TM stesse. Ciò è evidente dalla definizione stessa di U ; in altri termini, è evidente che se $M \in U$ allora per ogni altra TM M' tale che $L(M) = L(M')$, $n \in L(M')$, e dunque $M' \in U$.

Avendo quindi verificato le ipotesi del teorema di Rice, possiamo concludere direttamente con il suo asserto, ossia che il linguaggio U non è decidibile.

Una dimostrazione alternativa dell'indecidibilità di U può essere basata su di una riduzione da un problema indecidibile come A_{TM} . Supponiamo per assurdo che U sia decidibile, quindi esista una TM M_U che decida U . Sia inoltre la TM N un elemento di U , ovvero $n \in L(N)$. Data una istanza di A_{TM} (M, w), possiamo costruire la seguente TM P :

P=“On input $\langle M, w \rangle$, where M is a TM and w is its input:

1. Construct a TM M_w that accepts on input x if and only if M accepts w and N accepts x .

2. Run M_U on M_w . If it accepts, accept; otherwise, reject.”

Supponiamo che $w \in L(M)$: in questo caso $L(M_w) = L(N)$, perché M accetta w , quindi $n \in L(M_w)$ e pertanto $M_w \in U$; la TM P accetta (M, w) in quanto M_U accetta M_w . Nel caso invece che $w \notin L(M)$, $L(M_w) = \emptyset$, quindi $n \notin L(M_w)$; la TM P rifiuta (M, w) poiché M_U rifiuta M_w . L'esistenza di una TM M_U che decide U implica dunque l'esistenza di una TM P che decide A_{TM} ; poiché A_{TM} è indecidibile, possiamo concludere che anche U deve essere indecidibile.

Esercizio 5 [9] Il problema NOT-ALL-EQUAL SAT (NAESAT) è costituito dalle istanze di SAT in cui esiste una assegnazione di verità alle variabili tale per cui in ogni clausola esiste almeno un letterale vero ed un letterale falso (ossia, nessuna clausola ha i valori dei letterali tutti veri o tutti falsi). Dimostrare che NAESAT è NP-completo.

Soluzione: Per prima cosa dimostriamo l'appartenza di NAESAT in NP. Il problema è polinomialmente verificabile perché ogni istanza che fa parte del linguaggio ha come certificato l'assegnazione di verità alle variabili che garantisce la presenza in ciascuna clausola di un letterale falso e di un letterale vero. Tale certificato è certamente di dimensione non superiore alla dimensione dell'istanza, e verificare che l'assegnazione soddisfa i requisiti del problema è implementabile in modo immediato, in modo del tutto analogo alla verifica usata per SAT che una assegnazione di verità renda almeno un letterale vero in ogni clausola.

Per dimostrare la NP-hardness di NAESAT costruiamo una riduzione dal problema NP-completo 3SAT. Data una istanza di 3SAT

$$\Phi = \bigwedge_i \bigvee_{j=1}^3 l_{i,j},$$

la riduzione polinomiale costruisce la formula CNF

$$\Psi = \bigwedge_i \left(s \vee \bigvee_{j=1}^3 l_{i,j} \right),$$

ove s è una variabile che non appare in alcuno dei letterali $l_{i,j}$.

Supponiamo che Φ sia una “istanza sì” di 3SAT, quindi che esista un assegnazione di valori di verità alle variabili di Φ che rende almeno un letterale entro ogni clausola vero. La stessa assegnazione di valori, estesa con l'assegnazione del valore “falso” alla variable s , assicura che nella formula Ψ esista almeno un letterale vero ed un letterale falso in ogni clausola. Pertanto, Ψ derivata da Φ è una “istanza sì” di NAESAT.

Viceversa, supponiamo che una formula Ψ costruita con lo stesso procedimento sia una “istanza sì” di NAESAT, pertanto che esista una assegnazione di verità alle variabili di Ψ tale che ogni clausola di Ψ include almeno un letterale vero ed un letterale falso. Distinguiamo due casi: se l’assegnazione di verità ha posto la variabile s a “falso”, allora la stessa assegnazione di verità (senza s) soddisfa la formula Φ da cui Ψ è stata derivata, poiché le rimanenti variabili garantiscono la presenza di almeno un letterale vero in ogni clausola di Φ . Se invece l’assegnazione di verità ha posto la variabile s a “vero”, allora si consideri l’assegnazione di verità complementare, in cui tutte le variabili ricevono la negazione del valore precedentemente assegnato. È immediato verificare che la nuova assegnazione di verità continua a garantire la presenza in ogni clausola di Ψ di un letterale vero e di un letterale falso, perché il valore di ciascun letterale viene negato rispetto alla precedente assegnazione. Quindi anche la assegnazione complementare certifica che Ψ appartiene a NAESAT. Nella assegnazione complementare la variabile s ha il valore “falso”, dunque si ricade nel caso precedente: la formula Φ è una “istanza sì” di 3SAT.

In conclusione, ogni “istanza sì” di 3SAT corrisponde ad una “istanza sì” di NAESAT, ed ogni “istanza no” di 3SAT corrisponde ad una “istanza no” di NAESAT. La trasformazione dalle istanze di 3SAT a quelle di NAESAT è evidentemente costruibile in tempo polinomiale e costituisce una riduzione tra problemi. Quindi NAESAT è NP-hard, e pertanto è NP-completo, come volevasi dimostrare.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 23 giugno 2020

(La prova è stata svolta “a libri aperti”, perciò una piccola parte degli esercizi proposti presenta una difficoltà superiore a quella consueta. I testi di alcuni esercizi sono stati modificati rispetto ai compiti assegnati in sede d’esame; la tipologia dell’esercizio ed il relativo svolgimento rimangono comunque immutati.)

Esercizio 1. Sia $A \subseteq \{0, 1\}^*$, e sia

$$A_1 = \{x \in \{0, 1\}^* \mid \exists y \in A : |x| = |y| \text{ e } x \text{ e } y \text{ differiscono per al più un singolo bit}\}.$$

Ad esempio, se $A = \{0, 11, 010\}$ allora $A_1 = A \cup \{1, 01, 10, 110, 000, 011\}$. Dimostrare che se A è un linguaggio regolare allora anche A_1 è regolare.

Soluzione: Per la dimostrazione è sufficiente esibire un DFA oppure un NFA che riconosca il linguaggio A_1 , partendo dal presupposto dell’esistenza di un DFA M che riconosce il linguaggio A . In sostanza, A_1 contiene le stringhe di A ed inoltre le stringhe ottenibili invertendo un singolo bit delle stringhe di A . Costruiamo l’automa non deterministico N in modo che simuli il comportamento di M ; prima di leggere ogni simbolo dell’input, N può non deterministicamente decidere che quel simbolo è differente da quanto si aspetta M , e quindi entrare in un insieme di stati alternativo che simulano ancora il comportamento di M ma senza più mosse nondeterministiche.

Se dunque $M = (Q, \{0, 1\}, \delta, q_0, F)$, sia $N = (Q_1, \{0, 1\}, \delta_1, q_0, F_1)$, ove:

- l’insieme degli stati è $Q_1 = Q \cup Q'$, con $Q' = \{q' \mid q \in Q\}$;
- l’insieme degli stati di accettazione è $F_1 = F \cup F'$, ove $F' = \{q' \mid q \in F\}$;
- per ogni transizione $\delta(p, b) = q$ in M , in N esistono tre transizioni: $\delta_1(p, b) = q$ (emula il comportamento di N , ancora nessun bit invertito), $\delta_1(p, \bar{b}) = q'$ (qui si verifica una inversione di bit), e $\delta_1(p', b) = q'$ (emula il comportamento di N , inversione già avvenuta).

Supponiamo che $x \in A_1$. Se $x \in A$, $N(x)$ accetta perché esiste un percorso non deterministico che utilizza solo stati originariamente in Q ed arriva ad uno stato in F . Altrimenti se $x \notin A$, allora deve esistere una stringa $x' \in A$ che differisce da x per un singolo bit, supponiamo quello

in posizione k . Si consideri allora il percorso non deterministico entro $N(x)$ che segue gli stati originariamente in Q per i primi $k - 1$ simboli dell'input, poi segue la transizione verso gli stati Q' in corrispondenza del k -esimo simbolo (arrivando allo stato che M avrebbe raggiunto leggendo il complemento del bit presente in input), ed infine segue gli stati Q' emulando il comportamento di M per i restanti simboli di input. Poiché $M(x')$ accetta, $N(x)$ arriva necessariamente in uno stato in F' , quindi in uno stato di accettazione.

Supponiamo ora che esista un percorso non deterministico accettante per $N(x)$. Se lo stato di accettazione finale fa parte di F , allora lo stesso percorso esiste nella computazione deterministica $M(x)$, quindi $x \in A$, e pertanto $x \in A_1$. Se invece lo stato di accettazione finale fa parte di F' , allora nel percorso non deterministico è stata applicata una transizione dagli stati Q agli stati Q' ; supponiamo che sia avvenuta in corrispondenza della lettura del k -esimo simbolo. Allora la stringa x' ottenuta da x invertendo esattamente il k -esimo bit è accettata da $M(x')$, in quanto N replica nelle transizioni tra gli stati Q' il comportamento dell'automa M . Poiché dunque $x' \in A$, per definizione $x \in A_1$.

Esercizio 2. Sia $A \subseteq \Sigma^*$, e sia

$$A_{1/2} = \{x \in \Sigma^* \mid \exists y : |x| = |y| \text{ e } xy \in A\}.$$

Ad esempio, se $A = \{1, 001, 0100, 0110, 1010\}$ allora $A_{1/2} = \{01, 10\}$. Dimostrare che se A è un linguaggio regolare allora $A_{1/2}$ è regolare.

Soluzione: Per la dimostrazione è sufficiente esibire un DFA od un NFA che riconosce le stringhe di $A_{1/2}$, partendo dal presupposto dell'esistenza di un DFA M che riconosce il linguaggio A . In sintesi, $A_{1/2}$ contiene le metà superiori di tutte le stringhe di A che hanno lunghezza pari. L'idea della dimostrazione è quella di costruire un automa che simula il comportamento di N e, contemporaneamente, utilizza il non-determinismo per indovinare (1) lo stato finale di M dopo aver letto la prima metà della stringa e (2) la seconda metà della stringa di A mancante nelle stringhe di $A_{1/2}$.

Se dunque $M = (Q, \{0, 1\}, \delta, q_0, F)$, sia $N = (Q', \{0, 1\}, \delta', q'_0, F')$, ove:

- $Q' = (Q \times Q \times Q) \cup \{q'_0\}$ (ogni stato (p, q, r) di N codifica tre stati di M : p è lo stato ottenuto seguendo la stringa di ingresso x , q è uno stato indovinato non deterministicamente in cui si fermerà $M(x)$, ed r è lo stato ottenuto in M seguendo la seconda metà della stringa a partire da q);
- $F' = \{(p, p, q) \mid p \in Q, q \in F\}$ (gli stati di accettazione di N sono quelli in cui lo stato finale di $M(x)$ è esattamente quello indovinato non deterministicamente, ed inoltre la

simulazione di M sulla seconda parte della stringa indovinata non deterministicamente porta ad uno stato di accettazione di M);

- δ' include le transizioni non deterministiche

$$\delta'(q'_0) = \{(q_0, p, p) \mid p \in Q\}$$

(all'inizio $N(x)$ indovina uno stato finale p per $M(x)$ e si prepara a simulare la parte di stringa mancante a partire da esso); inoltre per ogni $p, q, r \in Q$ ed ogni $b \in \{0, 1\}$,

$$\delta'((p, q, r), b) = \{(\delta(p, b), q, \delta(r, c)) \mid c \in \{0, 1\}\}$$

(il primo elemento della terna di stati segue la computazione di $M(x)$, il secondo rimane fisso, ed il terzo simula M su di un simbolo generato non deterministicamente).

Supponiamo che $x \in A_{1/2}$. Sia \bar{q} lo stato in cui termina $M(x)$, e consideriamo in $N(x)$ il percorso non deterministico che inizia con la transizione da q'_0 verso (q_0, \bar{q}, \bar{q}) . Per ogni simbolo letto di x , N emula deterministicamente la computazione di $M(x)$ nel primo elemento della terna di stati, arrivando alla fine in \bar{q} . Poiché $x \in A_{1/2}$, deve esistere $y \in \{0, 1\}^{|x|}$ tale che $xy \in A$. Consideriamo dunque le transizioni non deterministiche che “indovinano” i bit di y , partendo dallo stato \bar{q} : poiché $M(xy)$ accetta, $N(x)$ arriva in uno stato (\bar{q}, \bar{q}, q_A) , ove $q_A \in F$ (si noti che $|x| = |y|$, quindi la prima componente “raggiunge” \bar{q} quando la terza componente “raggiunge” q_A). Poiché $(\bar{q}, \bar{q}, q_A) \in F'$, $N(x)$ accetta.

Viceversa, supponiamo che $N(x)$ accetti una stringa x , ossia che esista un percorso non deterministico che porti nello stato (\bar{q}, \bar{q}, q_A) , con $q_A \in F$. Allora per costruzione $M(x)$ termina nello stato \bar{q} , ed esiste una sequenza di $|x|$ simboli y che porta l'automa M dallo stato \bar{q} allo stato q_A . Ciò significa che la computazione deterministica $M(xy)$ accetta, quindi $xy \in A$. Pertanto per definizione $x \in A_{1/2}$.

Esercizio 3. Sia $A \subseteq \{0, 1\}^*$ tale che $x \in A$ se e solo se la somma di tutti i bit di x è minore della metà della lunghezza della stringa x (ossia, se $x = x_1, \dots, x_n$, allora $\sum_{i=1}^n x_i < \frac{n}{2}$). Dimostrare che A non è un linguaggio regolare.

Esercizio 4. Sia $A \subseteq \{0, 1\}^*$ tale che $x \in A$ se e solo se il numero di bit 1 in x è minore del numero di bit 0. Dimostrare che A non è un linguaggio regolare.

Soluzione: Questi problemi sono identici, anche se hanno una diversa formulazione. Per dimostrare che A non è un linguaggio regolare utilizziamo il “pumping lemma”. Supponiamo

dunque per assurdo che A sia regolare, e sia $p > 0$ la sua “pumping length”. Consideriamo la stringa

$$w = \overbrace{1 \cdots 1}^p \overbrace{0 \cdots 0}^{p+1} \in A.$$

A causa del “pumping lemma” deve esistere una suddivisione $w = xyz$ ove $|y| > 0$, $|xy| \leq p$, e per ogni intero i vale $xy^i z \in A$. Poiché $|xy| \leq p$ allora $xy \in 1^*$; considerato che $|y| > 0$, ne consegue che nella stringa $xy^2 z$ il numero di 1 deve crescere almeno di una unità mentre il numero di 0 rimane costante. Perciò $xy^2 z \notin A$. Questo contraddice il “pumping lemma”, e dunque si deve concludere che il linguaggio A non è regolare.

Esercizio 5. Siano $A, B \subseteq \Sigma^*$, e sia $A \setminus B = \{x \in A \mid x \notin B\}$. Dimostrare che se A è un linguaggio libero dal contesto (CFL) e B è un linguaggio regolare allora $A \setminus B$ è CFL.

Soluzione: Dando per dimostrato il lemma che afferma che se A è CFL e B è regolare allora $A \cap B$ è CFL, l'asserto segue immediatamente considerando l'uguaglianza $A \setminus B = A \cap B^c$ ed osservando che il complemento di un linguaggio regolare è regolare.

D'altra parte una dimostrazione diretta per l'esercizio è essenzialmente identica alla dimostrazione che $A \cap B$ è CFL. Poiché A è CFL, esiste un PDA $P = (Q_P, \Sigma, \Gamma_P, \delta_P, q_0^P, F_P)$ che accetta tutti e solo gli elementi di A . Analogamente, poiché B è regolare, esiste un DFA $D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$ che accetta tutti e solo gli elementi di B . L'idea della dimostrazione è di esibire un altro PDA che si comporta esattamente come P ma contemporaneamente simula l'esecuzione di D , ed accetta se e solo se P accetta mentre D rifiuta. (Nel caso di $A \cap B$, si deve accettare se e solo se sia P che B accettano.)

Consideriamo dunque un altro PDA $P' = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ove $Q = Q_P \times Q_D$, $\Gamma = \Gamma_P$, $q_0 = (q_0^P, q_0^D)$ e $F = F_P \times F_D^c$.

Per ogni stato $q = (q^P, q^D) \in Q_P \times Q_D$ e simboli $x \in \Sigma_\varepsilon$, $y \in \Gamma_\varepsilon$, sia

$$\delta(q, x, y) = \begin{cases} \{(q'^P, q^D), y'\} \mid (q'^P, y') \in \delta_P(q^P, \varepsilon, y) \} & \text{se } x = \varepsilon \\ \{(q'^P, q'^D), y'\} \mid (q'^P, y') \in \delta_P(q^P, x, y) \text{ e } \delta_D(q^D, x) = q'^D \} & \text{se } x \in \Sigma \end{cases}$$

Si osservi che P' può effettuare delle transizioni senza consumare simboli di input, ed in questo caso lo stato di D registrato negli stati di P' non cambia. Altrimenti può essere applicata una qualunque delle transizioni definite da P e contemporaneamente la transizione definita da D .

Supponiamo che $w \in A \setminus B$; allora esiste una sequenza di transizioni di P che porta $P(w)$ in uno stato in F_P , ed esiste una sequenza di transizioni di D che porta $D(w)$ in uno stato di non accettazione, ossia in F_D^c . Pertanto per costruzione esiste una sequenza di transizioni di

P' che porta $P'(w)$ in uno stato di $F = F_P \times F_D^c$. Viceversa, se P' accetta una stringa w , allora per costruzione di P' $P(w)$ accetta mentre $D(w)$ rifiuta, e dunque $w \in A \setminus B$.

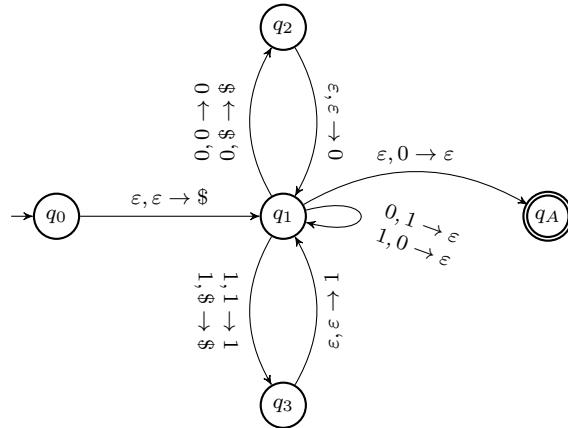
Poiché esiste un PDA che riconosce $A \cap B$, concludiamo che esso è CFL.

Esercizio 6. Sia $A \subseteq \{0,1\}^*$ tale che $x \in A$ se e solo se la somma di tutti i bit di x è minore della metà della lunghezza della stringa x (ossia, se $x = x_1, \dots, x_n$, allora $\sum_{i=1}^n x_i < \frac{n}{2}$). Dimostrare che il linguaggio A è libero dal contesto (CFL).

Soluzione: Per dimostrare che A è CFL possiamo esibire una grammatica libera dal contesto che genera tutte e sole le stringhe di A , oppure un automa a pila che riconosce gli elementi di A . Poiché $x \in A$ se e solo se il numero di bit 1 in x è strettamente inferiore al numero di bit 0, sceglieremo di costruire un PDA che tenga traccia dello “sbilanciamento” nelle quantità dei due tipi di simboli.

In pratica, l'automa memorizza nello stack esclusivamente i simboli in eccesso: ad esempio, ipotizzando che ad un certo punto della computazione siano stati letti 10 simboli ‘0’ e 7 simboli ‘1’, nello stack si troverebbero tre simboli ‘0’ ed il simbolo ‘\$’ (che serve a marcare il fondo dello stack). Se durante la scansione della stringa il numero di ‘0’ e di ‘1’ finora letti fosse identico, allora lo stack conterebbe soltanto il simbolo ‘\$’. In nessun caso nello stack potranno essere memorizzati sia simboli ‘0’ che ‘1’.

Un automa che implementa questo meccanismo è il seguente:

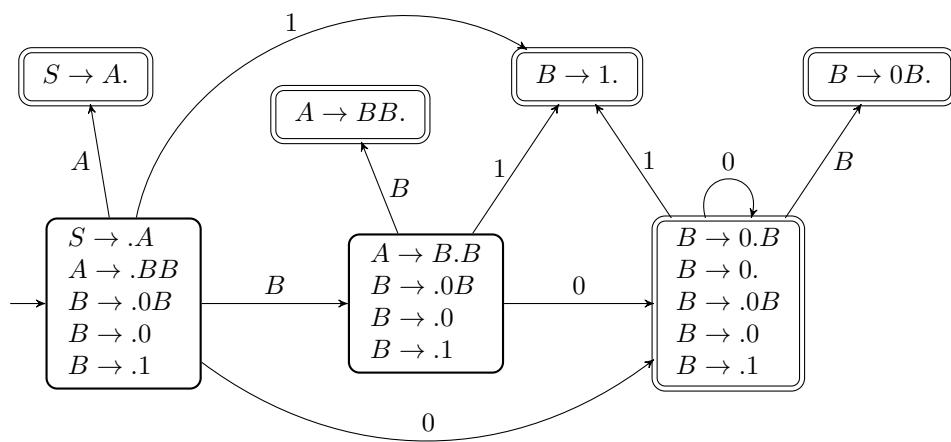


Lo stato iniziale q_0 scrive il simbolo ‘\$’ e fa transitare l'automa nello stato principale q_1 . In questo stato, quando viene letto un simbolo dell'input diverso dal simbolo sulla cima dello stack, allora viene rimosso il simbolo sulla cima dello stack (o meglio, poiché l'operazione di

lettura lo ha rimosso dallo stack, il simbolo *non* viene re-inserito), e si resta nello stato q_1 . Se invece il simbolo letto dall'input è uguale a quello sulla cima dello stack, oppure lo stack è vuoto (ossia ‘\$’ è sulla cima), allora viene aggiunto un simbolo in eccesso: per prima cosa si rimette sulla cima dello stack il simbolo tolto, poi se ne aggiunge un altro identico a quello letto dall'input. Per questo scopo vengono utilizzati gli stati q_2 e q_3 . Infine, in qualsiasi momento l'automa può transitare nello stato di accettazione q_A , purché però sulla cima dello stack sia presente il simbolo ‘0’, che indica che fino a quel momento si è avuto un eccesso di zero rispetto agli uno. Se il salto a q_A avviene quando l'input non è stato totalmente consumato, l'automa non potrà accettare perché dallo stato q_A non si può più leggere alcun simbolo di input. Il percorso non deterministico che salta a q_A in corrispondenza della fine dell'input assicura l'esistenza di un percorso deterministico accettante, a condizione ovviamente che nell'input fosse presente una eccedenza di ‘0’.

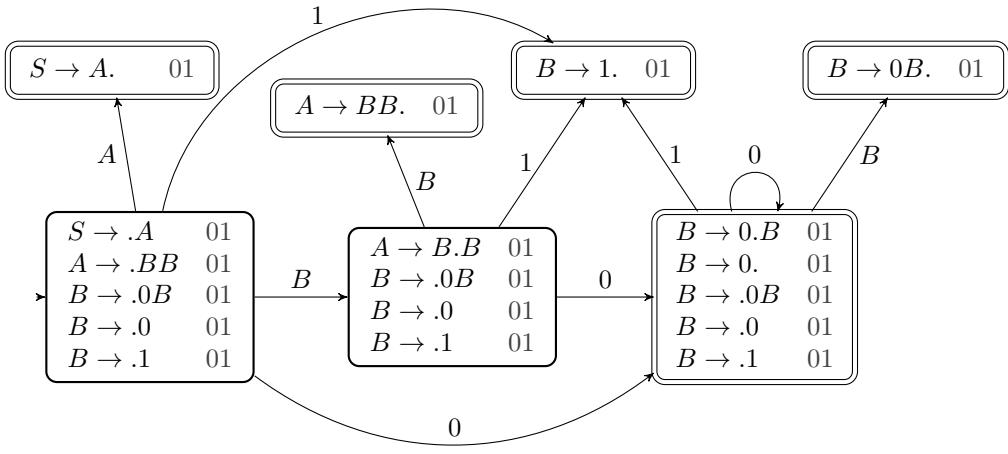
Esercizio 7. Si consideri la grammatica CFG con variabile iniziale S , simboli terminali 0 e 1, e avente le seguenti regole: $S \rightarrow A$, $A \rightarrow BB$, $B \rightarrow 0B$, $B \rightarrow 0$, $B \rightarrow 1$. Verificare se la grammatica è LR(0) oppure LR(1).

Soluzione: Iniziamo dimostrando che la grammatica non è LR(0) (ossia DCFG) utilizzando il DK-test:



Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo ‘0’ contiene una regola completata e diverse regole in cui il punto precede un simbolo terminale. Perciò il DK-test è fallito, e di conseguenza la grammatica non è DCFG.

Dimostriamo ora che la grammatica non è LR(1) utilizzando il DK₁-test: aggiungiamo i simboli di “lookahead” al grafo precedente, ottenendo così il seguente grafo:



Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo ‘0’ contiene diverse regole consistenti. Infatti la regola completata ha come simboli di lookahead sia ‘0’ che ‘1’, e nello stesso stato compaiono regole non completate in cui il simbolo ‘0’ od il simbolo ‘1’ seguono il punto. Pertanto il DK₁-test è fallito e la grammatica non è LR(1).

Esercizio 8. Dimostrare che il problema di stabilire se due automi a stati finiti deterministici (DFA) riconoscono lo stesso linguaggio è decidibile esibendo un algoritmo che confronta le decisioni dei due automi su tutte le stringhe fino ad una determinata lunghezza X . Quale è il valore di X superato il quale si può concludere che gli automi riconoscono lo stesso linguaggio?

Soluzione: Consideriamo i DFA $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che operano sullo stesso alfabeto Σ (nel caso in cui gli automi operassero su alfabeti differenti, consideriamo un alfabeto comune costituito dalla loro unione, ed aggiungiamo transizioni a ciascuno dei due automi che portano l'automa in uno stato di rifiuto definitivo ogni volta che legge un simbolo non incluso nel suo alfabeto originale).

Supponiamo di essere in grado di calcolare un valore limite X come richiesto dal testo dell'esercizio. La seguente TM rifiuta se esiste una stringa $w \in \Sigma^*$ con $|w| \leq X$ tale che $D_1(w)$ e $D_2(w)$ forniscono risposte discordanti; se tale stringa w non esiste, la TM accetta.

M = “On input $\langle D_1, D_2 \rangle$, where D_1 and D_2 are DFA’s:

1. Compute the value X (see below)
 2. For each string w with $|w| \leq X$:
 3. Emulate the DFA D_1 on input w

4. Emulate the DFA D_2 on input w
5. If $D_1(w)$ accepts while $D_2(w)$ rejects, then reject
6. If $D_1(w)$ rejects while $D_2(w)$ accepts, then reject
7. Accept (because D_1 and D_2 agree on all strings)"

Poiché M genera un numero finito di stringhe (limitato superiormente da $|\Sigma|^X$) e per ciascuna di esse simula l'esecuzione di due DFA che concludono ciascuna in al massimo X passi, M termina su ogni input ed è quindi un decisore. Rimane però da stabilire il valore di X che rende M un decisore per il linguaggio $\mathcal{EQ}_{\text{DFA}}$.

Lemma: se esiste una stringa $v \in \Sigma^*$ per la quale D_1 e D_2 non concordano (ossia, $v \in L(D_1) \Delta L(D_2) = (L(D_1) \cap L(D_2)^c) \cup (L(D_2) \cap L(D_1)^c)$), allora esiste una stringa $w \in L(D_1) \Delta L(D_2)$ di lunghezza non superiore a $|Q_1| \cdot |Q_2|$.

Infatti, supponiamo che $|v| > |Q_1| \cdot |Q_2|$, e consideriamo la sequenza di stati $q_1, q_2, \dots, q_{|v|}$ attraversati dall'automa D_1 leggendo la stringa v , e la analoga sequenza di stati $q'_1, q'_2, \dots, q'_{|v|}$ attraversati da D_2 leggendo v . Poiché il numero delle coppie di stati $(q, q') \in Q_1 \times Q_2$ è uguale a $|Q_1| \cdot |Q_2|$, devono esistere due simboli v_i e v_j di v (con $i < j$) aventi la stessa coppia di stati (q, q') . La stringa v' ottenuta da v rimuovendo tutti i simboli che seguono v_i fino al simbolo v_j incluso è ancora contenuta in $L(D_1) \Delta L(D_2)$ ed ha lunghezza strettamente inferiore a v . Ripetendo il procedimento si giunge ad identificare una stringa w di lunghezza non superiore a $|Q_1| \cdot |Q_2|$ ed inclusa in $L(D_1) \Delta L(D_2)$.

Pertanto, impostando $X = |Q_1| \cdot |Q_2|$ si ottiene per M un decisore del linguaggio $\mathcal{EQ}_{\text{DFA}}$.

Esercizio 9. Siano $A, B \subseteq \Sigma^*$ linguaggi ricorsivamente enumerabili. Dimostrare che il linguaggio $A \bar{\circ} B = \{x \in \Sigma^* \mid \exists y \in B : xy \in A\}$ è ricorsivamente enumerabile.

Soluzione: L'operatore “ $\bar{\circ}$ ” introdotto nel testo dell'esercizio può essere considerato come l'inverso dell'operatore di concatenazione: esso rimuove dalle stringhe del primo linguaggio A tutte le “code” costituite dalle stringhe del secondo linguaggio B .

I linguaggi ricorsivamente enumerabili possono essere caratterizzati dall'esistenza di un enumeratore, ossia una TM che iterativamente “stampa” su di un nastro speciale tutte le stringhe che fanno parte del linguaggio. In particolare, se una stringa fa parte del linguaggio, l'enumeratore la stamperà certamente dopo un numero finito di passi. Poiché per ipotesi i linguaggi A e B sono ricorsivamente enumerabili, esistono i corrispondenti enumeratori E_A e E_B . Dimostriamo che $A \bar{\circ} B$ è ricorsivamente enumerabile mostrando come è possibile costruire un enumeratore E per tale linguaggio basato su E_A e E_B .

E = “On any input (ignored):

1. For every increasing value $n = 1, 2, \dots, \infty$
2. Run the emulator E_A and collect the first n strings of A
3. For each string v printed by E_A :
 4. Run the emulator E_B and collect the first n strings of B
 5. For each string y printed by E_B :
 6. If y is the trailing substring of v ($v = xy$ for some x):
 7. Print the leading substring x ”

Verifichiamo che effettivamente E è un enumeratore per il linguaggio $A \bar{\circ} B$. Innanzi tutto, se E stampa una stringa x , allora necessariamente deve esistere una stringa $y \in B$ ed una stringa $v \in A$ tale che $v = xy$. Perciò $x \in A \bar{\circ} B$.

Viceversa, supponiamo che $x \in A \bar{\circ} B$. Per definizione deve allora esistere una stringa $y \in B$ tale che la concatenazione xy è un elemento di A . L'enumeratore E_A stamperà la stringa xy dopo un numero finito di passi; sia xy la n_A -esima stringa stampata da E_A . Analogamente, l'enumeratore E_B stamperà la stringa y dopo un numero finito di passi; sia y la n_B -esima stringa stampata da E_B . Nell'iterazione dell'enumeratore E con $n = \max(n_A, n_B)$, xy è inclusa nelle stringhe collezionate da E_A e y è inclusa nelle stringhe collezionate da E_B ; perciò E potrà verificare che y è la “coda” di xy e stamperà la stringa x .

Esercizio 10. Si consideri il linguaggio $A = \{x \in \{0,1\}^* \mid x \text{ ha un numero di bit 1 uguale al numero di bit 0}\}$. Sia $B = \{\langle M \rangle \mid M \text{ è una TM e } L(M) \subseteq A\}$. Il linguaggio B è decidibile oppure no? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio B non è decidibile, e per dimostrarlo è sufficiente verificare che le ipotesi del Teorema di Rice sono verificate. Si consideri come proprietà P del linguaggio della TM l'essere un sottinsieme del linguaggio A . Tale proprietà è non banale: infatti la TM che accetta tutte le stringhe non soddisfa la proprietà ($\Sigma^* \not\subseteq A$), mentre la TM che rifiuta tutte le stringhe soddisfa la proprietà ($\emptyset \subseteq A$). Inoltre, P è una proprietà del linguaggio riconosciuto dalla TM: infatti se due diverse TM riconoscono lo stesso linguaggio, per entrambe vale che il linguaggio è un sottinsieme di A , e dunque entrambe le TM soddisfano la priorità P . Poiché tutte le ipotesi del Teorema di Rice sono soddisfatte possiamo concludere immediatamente che il linguaggio B contenente codifiche di TM che soddisfano la proprietà P è indecidibile.

Esercizio 11. Il problema HITTING SET è costituito da un insieme di elementi U , da una collezione \mathcal{C} di sottoinsiemi di U , e da un numero intero k tali che esiste un sottoinsieme

$S \subseteq U$ (non necessariamente in \mathcal{C}) con $|S| \leq k$ e S contenente almeno un elemento di ciascun sottoinsieme in \mathcal{C} . Dimostrare che il problema è NP-completo (suggerimento: considerare il problema NP-completo VERTEX COVER).

Soluzione: Per dimostrare che il problema HITTING SET (HS) può essere verificato in tempo polinomiale, consideriamo la seguente TM:

$M =$ “On input $\langle U, \mathcal{C}, k, C \rangle$, where U is a set of elements, $\mathcal{C} \subseteq 2^U$, $k \in \mathbb{N}$:

1. Verify that C is a list of elements of U ; if not, reject.
2. Verify that C includes k or less elements; if not, reject.
3. For each element x in C :
 4. Scan the subsets in \mathcal{C} and mark every subset containing x .
 5. Scan the subsets in \mathcal{C} : if one of them is not marked, reject.
 6. Accept (all subsets in \mathcal{C} are marked)”

I primi due passi possono essere conclusi in tempo $O(k)$, e quindi in tempo $O(n)$. Il ciclo del passo 3 esegue $O(k) = O(n)$ iterazioni; in ciascuna di esse, vengono scanditi tutti gli elementi in \mathcal{C} (in numero certamente inferiore alla dimensione dell’istanza, poiché \mathcal{C} fa parte dell’istanza). Anche il passo 5 è eseguito in tempo lineare nella dimensione dell’istanza, mentre l’ultimo passo impiega tempo costante. Pertanto M esegue in tempo polinomiale nella dimensione dell’istanza, e quindi possiamo concludere che $HS \in \text{NP}$.

Per verificare che HS è NP-hard, è sufficiente considerarlo come una semplice generalizzazione del problema VERTEX COVER. Un grafo non diretto infatti può essere considerato come un insieme di elementi (i nodi) ed una collezione di sottoinsiemi di nodi con esattamente due elementi in ogni sottoinsieme (gli archi). Formalmente, per ogni istanza di VC, ossia un grafo $G = (V, E)$ ed un intero $k \in \mathbb{N}$, consideriamo l’istanza (U, \mathcal{C}, k') di HS così fatta: $U = V$, $\mathcal{C} = \{\{u, v\} \mid (u, v) \in E\}$ e $k' = k$. Il grafo G ammette un ricoprimento tramite vertici di dimensione $\leq k$ se e solo se questi k nodi ricoprono tutti gli archi, ovvero se e solo se lo stesso sottoinsieme di elementi di U ha un elemento in ciascun sottoinsieme di \mathcal{C} , ossia se e solo se l’istanza (U, \mathcal{C}, k) ammette un “hitting set” di dimensione al più k . Pertanto, VC riduce in tempo polinomiale a HS, e dunque HS è NP-hard.

Esercizio 12. Il problema GRAPH 3-COLORING è costituito da un grafo non diretto G tale che ogni nodo del grafo può essere “colorato” con uno di tre possibili colori in modo che nodi adiacenti abbiano colori differenti. Dimostrare che il problema è NP-completo (suggerimento: considerare il problema NP-completo 3SAT).

Soluzione: Per dimostrare che il problema GRAPH 3-COLORING (3COL) è verificabile in tempo polinomiale consideriamo la seguente TM:

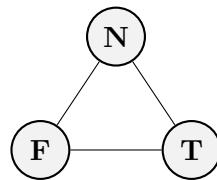
$M =$ “On input $\langle G, k, C \rangle$, where G is a graph and $k \in \mathbb{N}$:

1. Assign an ordering v_1, v_2, \dots, v_n to the n nodes of G .
2. Verify that C is a list of elements in $\{c_1, c_2, c_3\}$; if not, reject.
3. Verify that C includes exactly $n = |V(G)|$ elements; if not, reject.
4. For each edge $(v_i, v_j) \in E(G)$:
 6. If the i -th element and the j -th element of C are the same, reject.
7. Accept (all adjacent nodes have different colors)”

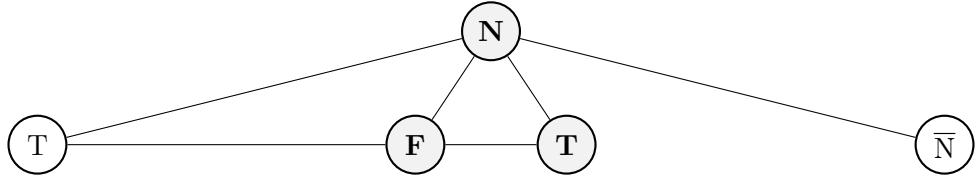
I passi 1, 2 e 3 possono essere completati in tempo lineare nel numero di nodi di G . Il ciclo nel passo 4 esegue $O(|E(G)|)$ iterazioni, ed in ciascuna di esse si controllano due elementi in una lista di n elementi. Pertanto M termina in un numero di passi polinomiale nella dimensione dell’input, e quindi $3\text{COL} \in \text{NP}$.

Per dimostrare che 3COL è NP-hard, consideriamo una riduzione dal problema 3SAT. Sia dunque Φ una formula CNF costituita da m clausole C_1, \dots, C_m e k variabili x_1, \dots, x_k . Ciascuna clausola è la disgiunzione di tre letterali. Vediamo come costruire da Φ un grafo G che è colorabile con 3 colori se e solo se Φ è soddisfacibile.

La struttura di base della riduzione sono 3 nodi mutuamente connessi, chiamiamo questo tipo di sottografo K_3 . È evidente che per colorare una struttura di tipo K_3 è necessario usare tutti e tre i colori a disposizione. Per comodità assegniamo ai 3 colori le etichette T (per ‘True’), F (per ‘False’) e N (per ‘Neutral’). Il grafo G include una struttura P di tipo K_3 che svolge il ruolo di tabella dei colori di riferimento:

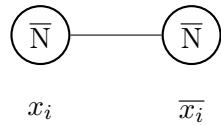


Per forzare un nodo del grafo ad assumere un determinato colore sarà sufficiente collegare il nodo ai due colori complementari nella tabella di riferimento. Un solo arco verso la struttura P impedirà al nodo di assumere il colore corrispondente:

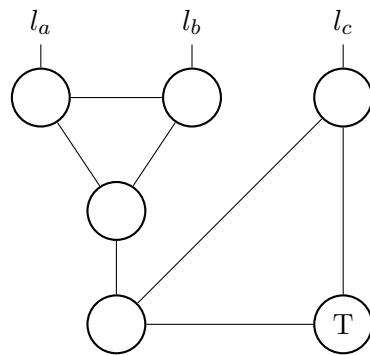


Per semplificare il grafico, una etichetta con un colore entro un nodo implica l'esistenza degli archi verso i nodi corrispondenti ai colori complementari nella struttura P . Un nodo con una etichetta con un colore negato implica l'esistenza di un arco verso il nodo della struttura P con il colore corrispondente.

Per ciascuna variabile x_i della formula Φ , il grafo G conterrà un gadget costituito da due nodi connessi tra loro ed il cui colore è forzato ad essere T oppure F. Uno dei due nodi sarà associato al letterale x_i , l'altro al letterale \bar{x}_i .



Per ciascuna clausola $C_i = l_a \vee l_b \vee l_c$ il grafo G include il seguente gadget:



I tre nodi superiori sono collegati da archi ai nodi dei gadget delle variabili corrispondenti ai letterali nella clausola (l'ordine è indifferente). È evidente che se i tre nodi superiori del gadget sono collegati a nodi tutti colorati con F, allora non è possibile colorare il gadget. Infatti in questo caso il nodo collegato al nodo letterale l_c dovrebbe avere colore N (perchè adiacente a due nodi colorati con T e F). Di conseguenza, il nodo in basso a sinistra nel gadget deve avere colore F. Ora ciascun nodo della struttura K_3 sopra quest'ultimo nodo è adiacente ad un nodo con colore F, quindi non è possibile colorarla. D'altra parte non è difficile convincersi che ogni altra configurazione permette di colorare con F almeno uno dei tre nodi superiori del gadget e di conseguenza tutto il gadget risulta colorabile.

Supponiamo che Φ sia una formula soddisfacibile, e consideriamo una assegnazione di verità alle variabili che rende la formula vera. Nel grafo corrispondente coloriamo con T i nodi corrispondenti alle variabili con valore vero, e coloriamo con F le variabili assegnate a falso. Di conseguenza, i colori dei nodi delle variabili negate è assegnato a T o F (poiché per costruzione queste variabili non possono avere il colore N). Si noti che per “colorare con T” o “colorare con F” si deve intendere assegnare lo stesso colore del nodo corrispondente nella struttura P . A questo punto risultano colorati i nodi della struttura P e i nodi dei gadget delle variabili, e dobbiamo verificare se questa colorazione può essere estesa ai nodi dei gadget delle clausole. Poiché Φ è soddisfatta, in ogni clausola esiste un letterale che assume il valore vero; quindi ogni gadget delle clausole ha un nodo nella fila superiore collegato ad un nodo con colore T, e che quindi può essere colorato con F. Perciò il resto del gadget può essere colorato, come sopra evidenziato.

Supponiamo ora che Φ sia una formula non soddisfacibile. Questo significa che ogni assegnazione di valori di verità alle variabili rende falsa almeno una clausola. Consideriamo quindi una qualunque colorazione del grafo: poiché i nodi dei gadget delle variabili devono avere il valore T o F, possiamo considerare l’assegnazione di verità corrispondente. Consideriamo ora un gadget delle clausole corrispondente ad una clausola non soddisfatta di Φ : tutti i nodi devono essere connessi a nodi con colore F, quindi come sopra evidenziato non è possibile colorare il resto del gadget. Perciò non esiste una colorazione per il grafo.

La costruzione descritta è una riduzione polinomiale dal problema 3SAT al problema 3COL, e quindi 3COL è NP-hard.

Esercizio 13. Dimostrare che il problema 2SAT contenente formule Booleane in forma normale congiuntiva con 2 letterali in ogni clausola appartiene alla classe P.

Soluzione: Benché esistano algoritmi per risolvere efficientemente (anche in tempo lineare) il problema 2SAT, questi sono generalmente sofisticati e poco intuitivi. Possiamo però convincerci che $2SAT \in P$ trasformandolo in un problema su grafi diretti.

Una formula Φ di 2SAT è la congiunzione di clausole costituite dalla disgiunzione di esattamente due letterali: $(l_1 \vee l_2)$, ove ogni letterale è una variabile diretta o negata. L’idea è che se ad un certo punto abbiamo determinato che il letterale l_1 , ad esempio, ha il valore falso, allora necessariamente il letterale l_2 deve assumere il valore vero, altrimenti la formula non potrà essere soddisfatta. Riformulando, la clausola $(l_1 \vee l_2)$ si traduce in due implicazioni: $(\neg l_1 \Rightarrow l_2)$ e $(\neg l_2 \Rightarrow l_1)$.

Data la formula Φ costruiamo il grafo diretto G avente 2 nodi per ciascuna variabile v , il primo associato al letterale v ed il secondo al letterale $\neg v$. Il grafo include inoltre due archi per ogni clausola $(l_1 \vee l_2)$: un arco dal nodo $\neg l_1$ al nodo l_2 , ed un altro arco dal nodo $\neg l_2$ al nodo l_1 . Gli archi rappresentano le “implicazioni”: assegnare ad un letterale l un valore vero implica dover assegnare il valore vero a tutti i letterali corrispondenti ai nodi raggiungibili con un singolo arco dal nodo l .

L’idea centrale è che la soddisfabilità di Φ è correlata con la struttura delle componenti fortemente connesse (CFC) del grafo. Una CFC è un sottoinsieme di nodi tale che per ciascuna coppia di nodi u, v nel CFC esistono i percorsi diretti da u a v e da v a u utilizzanti solo nodi del sottoinsieme. Se una assegnazione di verità alle variabili rende vero un certo letterale, allora tutti i letterali nella stessa CFC devono essere resi veri, altrimenti la formula non sarebbe soddisfatta.

Un’altra osservazione importante è che se il grafo G contiene il percorso da x a y , allora contiene anche il percorso da $\neg y$ a $\neg x$ (infatti se è stato inserito un arco $(u \rightarrow v)$ a causa della clausola $(\neg u \vee v)$ allora è stato inserito anche l’arco $(\neg v \rightarrow \neg u)$). Quindi per ogni CFC esiste una CFC “complementare” contenente tutti i nodi complementari della prima. Se CFC e CFC complementare coincidessero, allora in questa CFC troveremmo sia una variabile che la sua negazione. Assegnare un qualsiasi valore a questa variabile porterebbe ad una contraddizione.

Lemma: la formula Φ *non* è soddisfacibile se e solo se esiste una variabile x tale che il nodo x ed il nodo $\neg x$ sono nella stessa CFC.

Infatti, supponiamo che esistano nodi x e $\neg x$ nella stessa CFC. Dunque esiste nel grafo sia il percorso da x a $\neg x$ che il percorso da $\neg x$ a x . Supponiamo per assurdo che Φ sia soddisfacibile, e fissiamo pertanto una assegnazione di verità alle variabili che soddisfa la formula. Se alla variabile x viene assegnato il valore vero, consideriamo il percorso da x a $\neg x$. La catena di implicazioni porta a concludere che anche a $\neg x$ deve essere assegnato il valore vero, ma questo è impossibile perché non si può assegnare lo stesso valore di verità ad una variabile ed alla sua negazione. Lo stesso discorso si applica al percorso da $\neg x$ a x qualora alla variabile x sia assegnato il valore falso. La contraddizione prova che Φ non può essere soddisfacibile.

Viceversa, supponiamo che non esista alcun variabile x per la quale i nodi x e $\neg x$ sono nella stessa CFC. Ciò significa che esistono almeno due CFC complementari e distinte. Anche se possono esistere archi tra CFC differenti, questi archi non costituiscono mai un ciclo, altrimenti tutte le CFC toccate dal ciclo sarebbero un unica CFC. Di conseguenza esiste una CFC che non ha archi uscenti: impostare al valore vero tutti i letterali in questa CFC non comporta conseguenze verso altre variabili non incluse nella CFC, proprio perché non esistono archi uscenti. Rimuoviamo ora dal grafo questa CFC e la sua CFC complementare (ormai tutti i

letterali in questa CFC complementare avranno assegnato il valore falso). Di nuovo, esisterà una CFC del grafo rimanente senza archi uscenti, ed assegniamo il valore vero a tutti i suoi letterali. Ripetiamo il procedimento fino ad esaurire tutte le CFC. Si consideri ora una generica clausola $(l_1 \vee l_2)$ di Φ . Questa ha dato luogo a due distinti archi nel grafo: $(\neg l_1 \rightarrow l_2)$ e $(\neg l_2 \rightarrow l_1)$. Se uno dei due archi è contenuto in una CFC selezionata per assegnare il valore vero, l'altro arco è nella CFC complementare che è stata rimossa di conseguenza. La clausola è soddisfatta dal letterale l_1 o l_2 contenuto nella CFC selezionata per il valore vero. L'altra possibilità è che i due archi colleghino due CFC differenti, ma in questo caso la direzione è unica, altrimenti si avrebbe un ciclo tra CFC. Ora, l'assegnazione di verità deve avere dato il valore vero ai letterali dei nodi l_1 e l_2 , in quanto tra le due questa è la CFC sicuramente senza archi uscenti. Perciò la clausola è soddisfatta perché entrambi i suoi letterali sono veri. Dunque Φ è soddisfacibile.

Il lemma appena dimostrato ci consente di descrivere la seguente TM che decide il problema 2SAT:

$M =$ “On input $\langle \Phi \rangle$, a Boolean CNF formula with clauses of size 2:

1. Build the graph G corresponding to Φ
2. For every node $x \in V(G)$:
 3. Run the TM N deciding the PATH problem on $\langle G, x, \neg x \rangle$
 4. Run the TM N deciding the PATH problem on $\langle G, \neg x, x \rangle$
 5. If both $N(\langle G, x, \neg x \rangle)$ and $N(\langle G, \neg x, x \rangle)$ accept, then reject
6. Accept (no node and its complement in the same CFC)”

Poiché sappiamo che il problema PATH (relativo all'esistenza di un percorso tra due nodi in un grafo diretto) è in P, possiamo utilizzare una TM N che decide PATH in tempo polinomiale come procedura della TM M . Ogni passo della macchina può essere completato in tempo polinomiale nella dimensione dell'input, e quindi $2\text{SAT} \in P$.

Esercizio 14. Il problema GRAPH 2-COLORING è costituito da un grafo non diretto G tale che ogni nodo del grafo può essere “colorato” con uno di due possibili colori in modo che nodi adiacenti abbiano colori differenti. Dimostrare che il problema è in P.

Soluzione: Mostriamo una TM che decide direttamente le istanze del problema GRAPH 2-COLORING (2COL):

$M =$ “On input $\langle G \rangle$, where $G = (V, E)$ is a undirected graph:

1. Scan every node in V ; if all of them are colored, then accept

2. Pick a uncolored node $u \in V$
3. Assign an arbitrary color to u
4. Push u on the stack
5. While the stack is not empty:
 6. Pop a node u from the stack
 7. For every edge $e \in E$:
 8. If u is not one of the ends of e , continue
 9. If the other end v of e is already colored:
 10. If u and v have the same color, then reject
 11. Else (v is not colored):
 12. Assign to v the color opposite to the color of u
 13. Push v on the stack
 14. Restart from step 1"

La TM utilizza una porzione del nastro come uno stack. Ogni passo elementare dell'algoritmo richiede un tempo d'esecuzione costante oppure lineare nella dimensione del grafo. Ad ogni iterazione del ciclo più esterno viene colorato almeno un nodo, quindi il numero di iterazioni è al più $n = |V|$. Nello stack sono memorizzati, una sola volta, tutti i nodi appena colorati, quindi al massimo contiene n elementi. Possiamo dunque immediatamente concludere che M esegue in tempo polinomiale nella dimensione del grafo.

È immediato verificare che se $M(\langle G \rangle)$ accetta allora il grafo è 2-colorabile: infatti M accetta solo se tutti i nodi sono colorati, ed ogni assegnazione di un colore comporta la verifica della consistenza dei colori dei nodi adiacenti. Viceversa, supponiamo che il grafo è 2-colorabile. La procedura esegue una nuova iterazione del ciclo esterno per ogni componente连通的 del grafo. All'interno di una componente连通的 esistono due schemi di colorazione alternativi ed equivalentemente leciti; M sceglie uno dei due schemi assegnando un colore arbitrario al primo nodo non colorato della componente. Successivamente la scelta dei colori è forzata, in quanto ogni nodo adiacente ad uno già colorato deve necessariamente avere il colore opposto. M quindi si limita a completare la colorazione della componente连通的. Poiché in effetti il grafo è 2-colorabile, M alla fine accetterà.

Possiamo dare una dimostrazione alternativa se assumiamo di avere dimostrato che il problema 2SAT sulla soddisfacibilità di formule booleane in CNF con 2 letterali in ogni clausola è risolvibile in tempo polinomiale. In questo caso la dimostrazione dell'esercizio segue facilmente mostrando una riduzione dal problema GRAPH 2-COLORING (2COL) al problema 2SAT: $2\text{COL} \leq_p 2\text{SAT}$. Sia dunque $G = (V, E)$ un grafo non diretto. Costruiamo la formula Φ avente una variabile x_v per ogni nodo $v \in V$ e due clausole (x_u, x_v) e (\bar{x}_u, \bar{x}_v) per ogni arco $(u, v) \in E$. Le due clausole rappresentano le quattro implicazioni $(\bar{x}_u \Rightarrow x_v)$, $(\bar{x}_v \Rightarrow x_u)$, $(x_u \Rightarrow \bar{x}_v)$ e

$(x_v \Rightarrow \overline{x_u})$ che possono essere interpretate come “ x_u e x_v devono avere assegnazioni di verità differenti”. Supponiamo che esista una colorazione dei nodi di G con due colori. Allora l’assegnazione di verità alle variabili di Φ che assegna ad una variabile il valore vero se il nodo ha un determinato colore ed il valore falso se ha l’altro colore soddisfa la formula. Infatti se esistesse una clausola non soddisfatta in Φ questa avrebbe due letterali falsi. Ma ciò è impossibile perché le due variabili corrispondenti avrebbero lo stesso valore, e questo significa che due nodi adiacenti del grafo avrebbero lo stesso colore. Viceversa, se esiste una assegnazione di verità che soddisfa la formula, assegniamo i colori ai nodi in modo corrispondente. Se ora esistesse un arco nel grafo che collega due nodi colorati allo stesso modo, allora una delle due corrispondenti clausole nella formula sarebbe falsa, perché le variabili avrebbero lo stesso valore.

Automi e Linguaggi (M. Cesati)

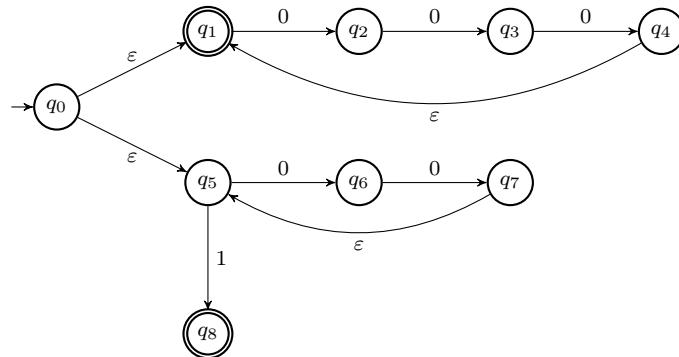
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 14 luglio 2020

(La prova è stata svolta “a libri aperti”. I testi di alcuni esercizi sono stati modificati rispetto ai compiti assegnati in sede d’esame; la tipologia dell’esercizio ed il relativo svolgimento rimangono comunque immutati.)

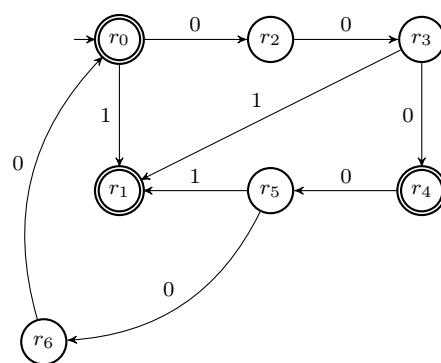
Esercizio 1. Determinare un automa a stati finiti deterministico (DFA) che riconosca il linguaggio generato dall’espressione regolare $(000)^* \cup (00)^*1$.

Soluzione: L’esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione di poco conto allo NFA derivato dalla espressione regolare si ottiene:

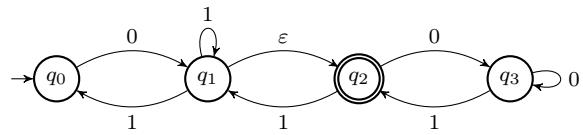


Un automa deterministico equivalente è il seguente:

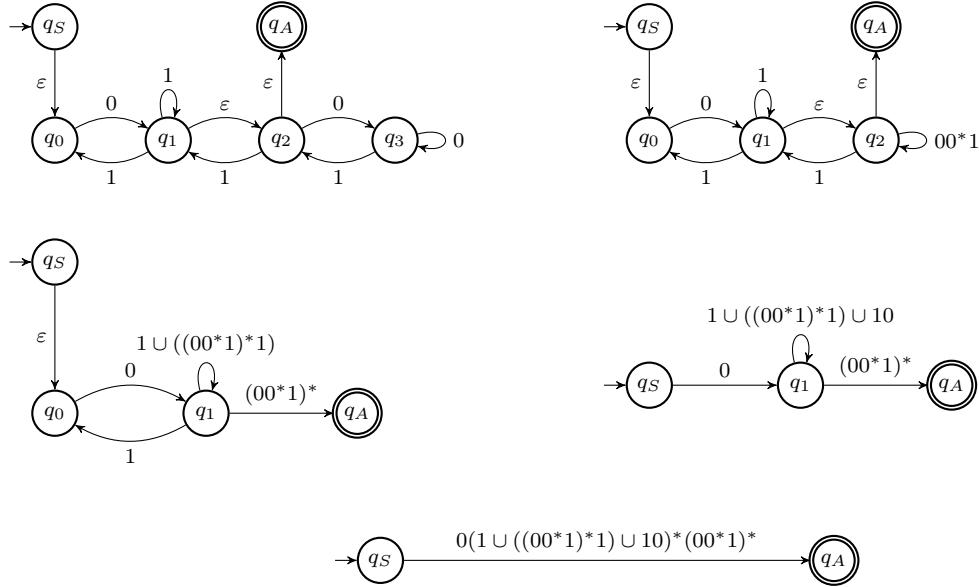
$$\begin{aligned}
 r_0 &= \{q_0, q_1, q_5\}, \{q_1, q_4, q_5, q_7\} \\
 r_1 &= \{q_8\} \\
 r_2 &= \{q_2, q_6\} \\
 r_3 &= \{q_3, q_5, q_7\} \\
 r_4 &= \{q_1, q_4, q_6\} \\
 r_5 &= \{q_2, q_5, q_7\} \\
 r_6 &= \{q_3, q_6\}
 \end{aligned}$$



Esercizio 2. Determinare una espressione regolare equivalente al seguente automa a stati finiti non deterministico (NFA):



Soluzione: Per derivare l'espressione regolare in modo meccanico trasformiamo l'automa in un GNFA e rimuoviamo uno alla volta gli stati intermedi. Nei diagrammi seguenti omettiamo gli archi con etichetta \emptyset .



Una espressione regolare che genera il linguaggio riconosciuto dall'automa è

$$0(1 \cup 10 \cup (00^*1)^*1)^*(00^*1)^*$$

Esercizio 3. Sia $A = \{u^R \# v \mid u, v \in \{0, 1\}^*\}$, u è la codifica binaria di un numero $n \geq 1$, e v è la codifica binaria del numero $n + 1$. Le codifiche binarie non hanno zeri non significativi a sinistra. Si noti che il linguaggio codifica u invertendo l'ordine dei suoi bit. Ad esempio fanno parte di A le stringhe $01\#11$ e $11\#100$, mentre non fanno parte di A le stringhe $10^R\#10$ (perché $10^R = 01$ ha uno zero non significativo) e $1\#11$. Dimostrare che A è un linguaggio libero dal contesto (CFL).

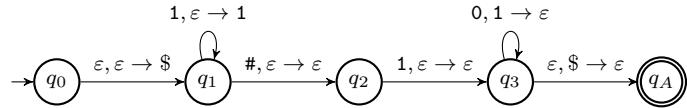
Soluzione:

Una semplice dimostrazione che A è CFL può essere ottenuta osservando che A è in effetti l'unione di due linguaggi:

$$A = B \cup C, B = \{1^n \# 10^n \mid n \geq 1\}, C = \{u^R \# v \mid u, v \in \{0, 1\}^*, |u| = |v|, (u)_2 + 1 = (v)_2\},$$

ove $(x)_2$ rappresenta il numero codificato in binario dalla stringa x . In altri termini, B rappresenta le istanze in cui è presente un riporto nella addizione $+1$, mentre C rappresenta le istanze di A in cui non è presente un riporto, e dunque tali che le stringhe u e v hanno la stessa lunghezza.

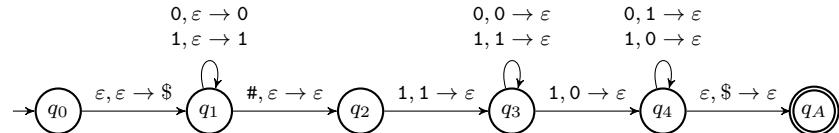
Per dimostrare che A è CFL è sufficiente dimostrare che sia B che C sono CFL (i linguaggi liberi dal contesto sono infatti chiusi rispetto all'operazione di unione). Il linguaggio B è una semplice variante del linguaggio $a^n b^n$, ed un PDA che lo riconosce è ad esempio:



Poiché le codifiche dei numeri nelle istanze-sì di A non devono contenere zeri non significativi, il linguaggio C può essere descritto come:

$$C = \{x 0 w 1 \# 1 y 1 z \mid w, x, y, z \in \{0, 1\}^*, w^R = y, x^R = \bar{z}\}$$

Un esempio di PDA che riconosce C è:



(Si osservi che la definizione di A richiede che il numero n codificato da u sia maggiore di zero. Se fosse ammesso anche il caso $n = 0$ allora bisognerebbe aggiungere a C la stringa $0\#\mathbf{1}$. Poiché un linguaggio costituito da una sola stringa è regolare, C continuerebbe comunque ad essere CFL.)

Esercizio 4. Sia $A = \{u\#v \mid u, v \in \{0, 1\}^*, u$ è la codifica binaria di un numero $n \geq 1$, e v è la codifica binaria del numero $n + 1\}$. Le codifiche binarie non hanno zeri non significativi a sinistra. Ad esempio fanno parte di A le stringhe $10\#11$ e $11\#100$, mentre non fanno parte di A le stringhe $01\#10$ e $1\#11$. Dimostrare che A non è un linguaggio libero dal contesto (CFL).

Soluzione: Per assurdo, supponiamo che A sia CFL, e che dunque esista per esso una “pumping length” $p > 0$. Consideriamo allora la stringa $s = 1^p 0 1^p \# 1^{p+1} 0^p \in A$. Poiché $|s| > p$, deve esistere una suddivisione $s = uvxyz$ tale che $|vy| > 0$, $|vxy| \leq p$, e $uv^i xy^i z \in A$ per ogni $i \geq 0$. Possono verificarsi solo i seguenti casi per la suddivisione di s :

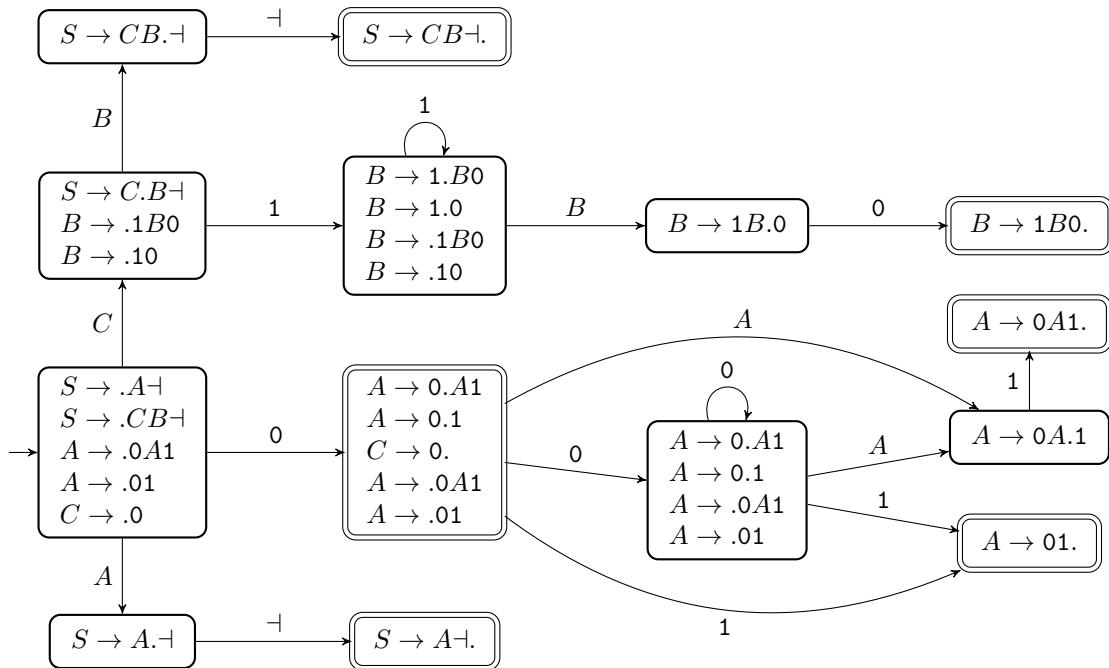
1. La sottostringa vxy non include ‘#’ ed è quindi tutta alla sinistra (o destra) di ‘#’: pompando verso il basso o verso l’alto si modifica la parte a sinistra (o destra) del simbolo ‘#’, quindi evidentemente la stringa risultante non può essere inclusa in A in quanto soltanto uno dei due numeri codificati nella stringa sarebbe variato rispetto a quelli codificati in s .
2. Il simbolo ‘#’ è incluso in v oppure y : in questo caso pompando verso l’alto si ottiene una stringa che include due o più simboli ‘#’, e che dunque non può far parte del linguaggio A .
3. Il simbolo ‘#’ è incluso in x , pertanto v include simboli a sinistra di ‘#’ e y include simboli a destra di ‘#’. Poiché $|vxy| \leq p$, nessun simbolo ‘0’ può apparire in v od in y . Poiché inoltre $|vy| > 0$, almeno uno dei seguenti due sottocasi deve verificarsi:
 - (a) $|v| > 0$: pompando verso l’alto la stringa assume la forma $1^p 0 1^q \# 1^r 0^p$ con $q > p$ e $r \geq p + 1$. Questa stringa non può far parte del linguaggio A perché sommando uno ad una sequenza di $q > p$ bit ‘1’ si deve ottenere una sequenza di $q > p$ bit ‘0’.
 - (b) $|y| > 0$: pompando verso il basso la stringa assume la forma $1^p 0 1^q \# 1^r 0^p$ con $q \leq p$ e $r < p + 1$. Se ora fosse $p + r < p + q + 1$ allora la stringa non potrebbe far parte del linguaggio A perché il numero a sinistra sarebbe maggiore del numero a destra. Dunque deve valere $p \geq r \geq q + 1 > q$, ossia $q < p$, e quindi $|v| > 0$. Si ricade pertanto nel caso precedente.

Poiché non è possibile trovare una suddivisione per la stringa s che soddisfa il pumping lemma, A non può essere un linguaggio libero dal contesto.

Esercizio 5. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{aligned} S &\rightarrow A \vdash \quad | \quad CB \dashv \\ A &\rightarrow 0A1 \quad | \quad 01 \\ B &\rightarrow 1B0 \quad | \quad 10 \\ C &\rightarrow 0 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:

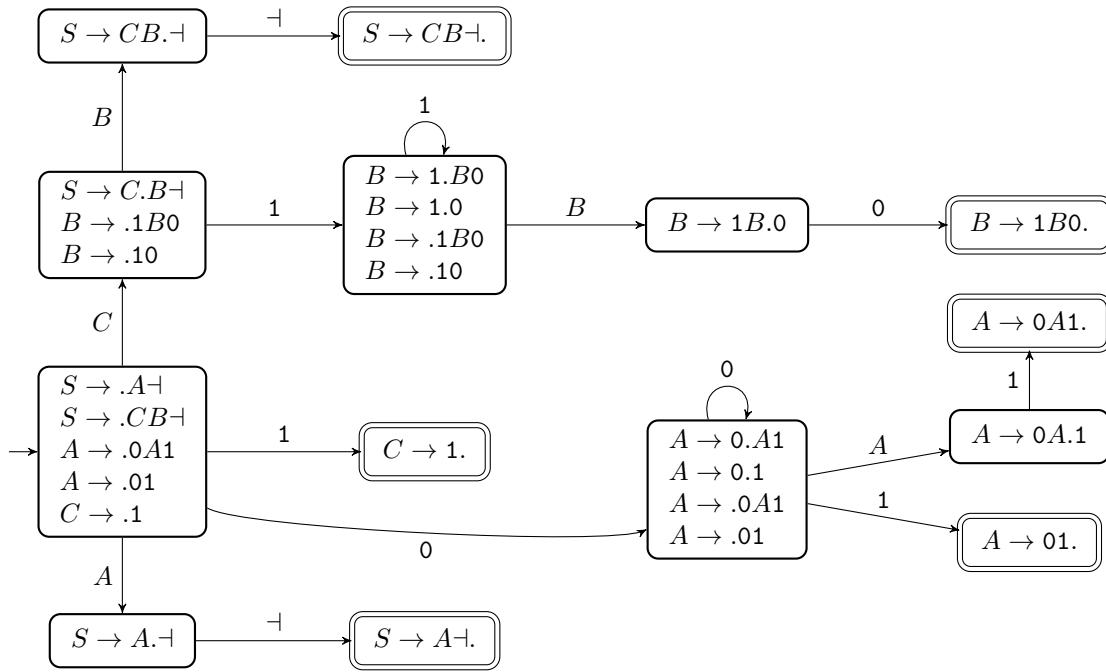


Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo ‘0’ contiene una regola completata e diverse regole in cui il punto precede un simbolo terminale. Perciò il DK-test è fallito, e di conseguenza la grammatica non è DCFG.

Esercizio 6. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{aligned}
 S &\rightarrow A⊣ \quad | \quad CB⊣ \\
 A &\rightarrow 0A1 \quad | \quad 01 \\
 B &\rightarrow 1B0 \quad | \quad 10 \\
 C &\rightarrow 1
 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:



Poiché ogni stato finale ha una sola regola, quella completata, il DK-test risulta soddisfatto, pertanto la grammatica analizzata è deterministica.

Esercizio 7. Sia $A \bar{\circ} B = \{x \mid \exists y \in B : xy \in A\}$, con $A, B \subseteq \Sigma^*$. Dimostrare che se A è un linguaggio libero dal contesto (CFL) e B è un linguaggio regolare allora $A \bar{\circ} B$ è libero dal contesto.

Soluzione: Poiché A è CFL, esiste un PDA $P = (Q_P, \Sigma, \Gamma, \delta_P, q_0^P, F_P)$ che lo riconosce. Analogamente, poiché B è un linguaggio regolare, esiste un DFA $D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$ che lo riconosce. Mostriamo come derivare da P e D un PDA P' che riconosce il linguaggio $A \bar{\circ} B$. L'idea è quella di simulare il comportamento di P fino alla fine della stringa (indovinata in modo non deterministico); successivamente P' simula il comportamento sia di P che di D su di una stringa y generata in modo non deterministico.

Costruiamo il PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ nel seguente modo. Gli stati interni di P sono $Q = Q_P \times (\{\hat{q}\} \cup Q_D)$, ove $\hat{q} \notin Q_D$. In sostanza, ciascuno stato di P' codifica uno stato di P (quando accoppiato con \hat{q}) oppure sia uno stato di P che uno stato di D . L'alfabeto di ingresso di P' coincide con il corrispondente alfabeto di P e D . L'alfabeto di stack di P' coincide con quello di P . Lo stato iniziale di P' è $q_0 = (q_0^P, \hat{q})$. Gli stati di accettazione di P' sono $F = F_P \times F_D$, ossia quelli che codificano stati di accettazione sia per P che per D .

La tabella di transizioni di P' è la seguente:

- Per ogni stato $(q^P, \hat{q}) \in Q$ e simboli $a \in \Sigma_\varepsilon$, $b \in \Gamma_\varepsilon$, δ contiene tutte le transizioni equivalenti in P :

$$\delta((q^P, \hat{q}), a, b) = \left\{ ((q'^P, \hat{q}), b') \mid (q'^P, b') \in \delta_P(q^P, a, b) \right\}$$

- Per ogni simbolo di ingresso $a \in \Sigma$, e per ogni coppia di transizioni $((q'^P, b') \in \delta_P(q^P, a, b)$ e $\delta_D(q^D, a) = q'^D$ che leggono quel simbolo a , δ contiene la transizione

$$((q'^P, q'^D), b') \in \delta((q^P, q^D), \varepsilon, b)$$

Si osservi che la transizione in P *non* legge il simbolo di ingresso a .

- Per ogni transizione $((q'^P, b') \in \delta_P(q^P, \varepsilon, b)$ di P che non legge simboli di ingresso e per ogni stato q^D , δ contiene la transizione

$$((q'^P, q^D), b') \in \delta((q^P, q^D), \varepsilon, b)$$

- Per ogni stato $q^P \in Q_P$, δ contiene la transizione

$$((q^P, q_0^D), \varepsilon) \in \delta((q^P, \hat{q}), \varepsilon, \varepsilon)$$

Sia $x \in A \bar{\sigma} B$, e consideriamo la computazione $P'(x)$. Poiché per definizione esiste un $y \in B$ tale che $xy \in A$, esiste una sequenza di transizioni in P' di tipo (1) che leggono l'intera stringa x e portano P' nello stato (q_x^P, \hat{q}) e con il contenuto dello stack S_x , ove q_x^P è esattamente lo stato raggiunto da P e S_x è il contenuto dello stack in una computazione accettante di $P(xy)$ dopo aver letto la stringa iniziale x . È possibile ora applicare una transizione di tipo (4) e transitare nello stato (q_x^P, q_0^D) . Poiché $P(xy)$ accetta, esiste una sequenza di transizioni di P dallo stato q_x che leggono i simboli della stringa y e portano in uno stato di accettazione di P . Se una di queste transizioni legge un simbolo di y si può applicare una transizione di tipo (2): l'automa P' dunque modifica lo stack ed entra nello stato interno corrispondente alla transizione applicata da P ; inoltre P' simula la corrispondente transizione di D applicata al simbolo di y . Se invece una di queste transizioni non legge un simbolo di y , P' può applicare una transizione di tipo (3) per modificare in modo appropriato stato interno di P e configurazione dello stack, lasciando invariato lo stato interno simulato di D . Poiché $y \in B$, quando la sequenza di transizioni corrispondente a quella di $P(xy)$ è terminata, lo stato interno di P' è uno stato di accettazione. Dunque $P'(x)$ accetta, in quanto l'intera stringa in ingresso è stata letta.

Viceversa, supponiamo che $P'(x)$ accetti una certa stringa $x \in \Sigma^*$, e consideriamo una sequenza di transizioni che porta in uno stato di accettazione. Notiamo innanzitutto che per poter accettare la stringa, questa deve essere stata letta completamente dall'automa. Inoltre

nessuno stato di accettazione può contenere lo stato \hat{q} come secondo componente della coppia. Si osservi anche che nessuna transizione in P' con lo stato \hat{q} come secondo componente può leggere simboli di ingresso. Infine, non esiste alcuna transizione in P' che possa portare da uno stato interno (q_x^P, q_0^D) ad uno stato interno con \hat{q} come secondo componente. Pertanto, i simboli della stringa x debbono essere letti da P' utilizzando esclusivamente transizioni di tipo (1). Poiché P' accetta, nella sequenza accettante deve esistere una transizione di tipo (4), sia questa $((q_x^P, q_0^D), \varepsilon) \in \delta((q_x^P, \hat{q}), \varepsilon, \varepsilon)$. Sia inoltre S_x la configurazione dello stack in questo punto della sequenza. Le transizioni di tipo (1) che portano da (q_0^D, \hat{q}) fino a (q_x^P, \hat{q}) corrispondono a transizioni di P ; pertanto esiste una computazione di $P(x)$ che termina nello stato q_x^P con configurazione dello stack S_x . Consideriamo ora le transizioni di tipo (2) e (3) seguenti: esse portano P' in uno stato di accettazione senza leggere alcun altro simbolo di ingresso. Per ogni transizione di tipo (2) che appare nella sequenza, esiste (almeno) un simbolo $a \in \Sigma$ che ha permesso di introdurre la transizione in δ . Sia y una stringa costituita dalla sequenza di simboli ‘ a ’ definita dalla sequenza di transizioni di tipo (2). Le transizioni di tipo (2) corrispondono a transizioni in D che portano l’automa D da q_0^D ad uno stato di accettazione in F_D leggendo la stringa y in ingresso; pertanto, $y \in B$. Le transizioni di tipo (2) e (3) invece corrispondono ad una sequenza di transizioni che portano l’automa P da q_x^P e configurazione dello stack S_x ad uno stato di accettazione in F_P ; pertanto $xy \in A$. Per definizione dunque $x \in A \bar{\circ} B$.

Esercizio 8. Siano A e B linguaggi Turing-riconoscibili (o in altri termini, ricorsivamente enumerabili). Dimostrare che l’intersezione $A \cap B$ è Turing-riconoscibile.

Soluzione: Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B , rispettivamente.

Consideriamo la seguente TM M :

$M =$ “On input x :

1. Emulate the TM M_A on input x
2. Emulate the TM M_B on input x
3. If both $M_A(x)$ and $M_B(x)$ accepted, then accept
4. Otherwise, reject”

Supponiamo che $M(x)$ accetti; di conseguenza sia l’emulazione di $M_A(x)$ che quella di $M_B(x)$ devono terminare, ed inoltre entrambe le computazioni devono accettare. Perciò $x \in A$ e $x \in B$, e quindi $x \in A \cap B$. D’altra parte, se $x \notin A$, allora o $M_A(x)$ non termina, e dunque

$M(x)$ non termina, oppure $M_A(x)$ termina e rifiuta, e quindi $M(x)$ non potrà accettare. Stesso ragionamento vale se $x \notin B$. Dunque se $x \notin A \cap B$, allora $M(x)$ non accetta (in quanto non termina oppure rifiuta).

Esercizio 9. Siano A e B linguaggi Turing-riconoscibili (o in altri termini, ricorsivamente enumerabili). Dimostrare che $AB = \{xy \mid x \in A, y \in B\}$ è Turing-riconoscibile.

Soluzione: Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B , rispettivamente.

Consideriamo la seguente NTM M :

M = “On input w :

1. Guess a subdivision x, y of w : $w = xy$
2. Emulate the TM M_A on input x
3. Emulate the TM M_B on input y
4. If both $M_A(x)$ and $M_B(y)$ accepted, then accept
5. Otherwise, reject”

Supponiamo che $M(w)$ accetti; di conseguenza esiste una computazione accettante che “indovina” una opportuna suddivisione $xy = w$, ed emula $M_A(x)$ e $M_B(y)$. Poiché $M(w)$ termina accettando, le due emulazioni devono terminare ed entrambe le computazioni devono accettare. Pertanto $x \in A$ e $y \in B$, e quindi $w \in AB$. D’altra parte, supponiamo che $w \in AB$; dunque esiste una suddivisione $w = xy$ tale che $x \in A$ e $y \in B$. Questa suddivisione corrisponde ad un ramo dell’albero di computazioni di $M(w)$; inoltre poiché M_A riconosce A , $M_A(x)$ termina accettando; analogamente $M_B(y)$ termina accettando. Perciò $M(w)$ accetta.

Si osservi che il non-determinismo di M non costituisce un problema, perché per ogni TM non deterministica esiste una TM deterministica equivalente. Perciò il linguaggio AB è ricorsivamente enumerabile.

Esercizio 10. Si consideri il problema codificato dal linguaggio $\{\langle \Phi \rangle \mid \Phi \text{ è una formula 3-CNF per la quale esiste una assegnazione di verità che rende un letterale vero ed un letterale falso in ciascuna clausola}\}$. Dimostrare che tale linguaggio è NP-completo esibendo una riduzione da 3SAT.

Soluzione: Questo problema è noto con diversi nomi, ad esempio \neq -SAT oppure “Heterogeneous SAT”. Essendo molto simile al problema NAESAT (NOT-ALL-EQUAL SAT) introdotto in un precedente esercizio d'esame, ci riferiremo ad esso con il nome NAE3SAT.

Per dimostrare che NAE3SAT è in NP osserviamo che esso è polinomialmente verificabile: infatti ogni istanza che fa parte del linguaggio ha come certificato l'assegnazione di verità alle variabili che garantisce la presenza in ciascuna clausola di un letterale falso e di un letterale vero. Tale certificato è certamente di dimensione non superiore alla dimensione dell'istanza, e verificare che l'assegnazione soddisfa i requisiti del problema è implementabile in modo immediato, in modo del tutto analogo alla verifica usata per SAT che una assegnazione di verità renda almeno un letterale vero in ogni clausola.

La riduzione polinomiale dal problema 3SAT è simile a quella utilizzata per NAESAT: introduciamo una nuova variabile s che non appare in alcuna clausola della formula originale e la cui assegnazione di verità assicurerà l'esistenza di un letterale falso in ogni clausola. A differenza però della riduzione per NAESAT non possiamo semplicemente aggiungere s ad ogni clausola perché il numero di letterali in ciascuna clausola diventerebbe uguale a quattro, mentre in NAE3SAT deve essere esattamente tre. Possiamo però sfruttare la stessa idea descritta nella riduzione da SAT a 3SAT: spezziamo ogni clausola di quattro letterali in due clausole di tre letterali ciascuna aggiungendo una nuova variabile per ciascuna clausola originale.

In pratica, sia Φ una formula 3-CNF costituita da m clausole: $\Phi = \wedge_{i=1}^m C_i$. Per ciascuna clausola $C_i = (l_a \vee l_b \vee l_c)$, definiamo due clausole: $(l_a \vee l_b \vee z_i)$ e $(l_c \vee \bar{z}_i \vee s)$. La variabile s è una nuova variabile comune a tutte le clausole, mentre z_i è una nuova variabile specifica per la clausola C_i . La formula 3-CNF Φ' costituita dalla congiunzione delle $2m$ clausole derivate da quelle di Φ è l'istanza del problema NAE3SAT. È evidente che la dimensione di Φ' è polinomialmente limitata dalla dimensione di Φ e che la sua costruzione è meccanica.

Supponiamo dunque che Φ sia soddisfacibile. Esiste perciò una assegnazione di verità alle variabili che rende vero un letterale in ciascuna clausola C_i . Consideriamo dunque un generico letterale $C_i = (l_a \vee l_b \vee l_c)$. La seguente tabella riassume le possibili assegnazioni di verità ai letterali, e le impostazioni delle variabili z_i e s che assicurano in ciascuna delle due clausole corrispondenti di Φ' l'esistenza di un letterale vero e di un letterale falso.

| l_a | l_b | l_c | z_i | s |
|-------|-------|-------|-------|---------|
| F | F | F | | escluso |
| F | F | T | T | * |
| F | T | F | F | * |
| F | T | T | T | * |
| T | F | F | F | * |
| T | F | T | T | * |
| T | T | F | F | * |
| T | T | T | F | F |

Si osservi che non è possibile che l_a , l_b e l_c siano tutti falsi, in quanto l'assegnazione di verità deve soddisfare la clausola originale. In tutti gli altri casi tranne l'ultimo il valore della variabile comune s non è importante, perché è sufficiente impostare opportunamente il valore di z_i per avere un letterale vero ed uno falso in ogni clausola. Solo nel caso in cui tutti e tre i letterali originali hanno il valore vero è necessario assegnare a s il valore falso.

Assegnando dunque alle variabili z_i il valore opportuno secondo la corrispondente tabella ed assegnando alla variabile s il valore falso si ottiene una estensione della assegnazione di verità per Φ' che soddisfa le condizioni del problema NAE3SAT.

Viceversa, consideriamo una formula Φ' che possiede una assegnazione di verità che assegna a ciascuna clausola un letterale vero ed uno falso. Si consideri il valore assunto dalla variabile s : se nella assegnazione s è vera, consideriamo l'assegnazione di verità complementare a quella data: essa necessariamente continuerà ad assegnare a ciascuna clausola un letterale vero ed un falso. In questa assegnazione il valore di s sarà falso. Assumiamo dunque che ad s sia assegnato il valore falso. Ciò significa che in ogni clausola con letterali \bar{z}_i , almeno uno tra l_c e \bar{z}_i deve assumere il valore vero. Se l_c assume il valore vero allora la corrispondente clausola $(l_a \vee l_b \vee l_c)$ è soddisfatta. Altrimenti \bar{z}_i deve essere vero, dunque z_i è falsa, perciò nella clausola $(l_a \vee l_b \vee z_i)$ almeno una tra l_a e l_b deve essere vera. Pertanto $(l_a \vee l_b \vee l_c)$ è ancora soddisfatta. Dunque la formula Φ è soddisfacibile.

Si può dunque concludere che $3SAT \leq_P NAE3SAT$, e dunque NAE3SAT è NP-completo.

Esercizio 11. Si consideri il problema FEEDBACK VERTEX SET: dato un grafo diretto $G = (V, A)$ ed un numero $k \in \mathbb{N}$, esiste un sottoinsieme $V' \subseteq V$ con $|V'| \leq k$ tale che ogni *ciclo* diretto entro G include almeno un nodo in V' ? Dimostrare che il problema è NP-completo.

Soluzione: Il problema FEEDBACK VERTEX SET (o FVS) è verificabile polinomialmente: un certificato è banalmente la lista di nodi che costituisce l'insieme V' . Si consideri infatti il seguente algoritmo.

M= “On input $\langle G, k, V' \rangle$:

1. Verify that V' is a subset of k or less nodes of G
2. Build the graph $G' = G \setminus V'$

3. For every pairs of nodes s, t in G' :
4. Run $\text{PATH}(G', s, t)$ to determine if there is a path from s to t
5. If the path exists:
 6. Let I be the nodes of the path except s and t
 7. Build the graph $G'' = G \setminus I$
 8. Run $\text{PATH}(G'', s, t)$
 - If the path exists, reject (cycle found)
9. There is no cycle in G' , hence accept"

La notazione $G \setminus V$ indica il grafo ottenuto da G rimuovendo tutti i nodi dell'insieme V e tutti gli archi incidenti su questi nodi. Poiché l'algoritmo PATH è polinomiale, il verificatore esegue in tempo polinomiale nel numero di nodi del grafo G in istanza.

Per dimostrare che FVS è NP-hard possiamo considerare una semplicissima riduzione dal problema NP-completo VERTEX COVER. Sia infatti (G, k) una istanza del problema VC, ove G è un grafo non diretto e $k \in \mathbb{N}$. Consideriamo il grafo diretto G' ottenuto sostituendo ad ogni arco non diretto di G una coppia di archi in direzione opposta tra gli stessi nodi in G' . Sia dunque (G', k) l'istanza ridotta di FVS, che ha ovviamente dimensione polinomiale ed è meccanicamente costruibile.

Supponiamo che (G, k) sia una istanza-sì di VC. Dunque esiste un sottoinsieme di al più k nodi che copre ogni arco di VC. Lo stesso sottoinsieme di nodi in G' copre ogni arco diretto, quindi la rimozione dei nodi di V' rende G' un grafo senza archi, e quindi senza cicli. Perciò (G', k) è una istanza-sì di FVS. Viceversa, supponiamo che (G', k) sia una istanza-sì di FVS, pertanto esiste un sottoinsieme di al più k nodi che copre ogni ciclo del grafo G' . Si consideri ora che ciascun arco del grafo G ha dato origine ad un ciclo di dimensione 2 in G' , e anche tali cicli devono essere coperti da V' . Pertanto l'insieme di nodi V' copre ogni arco del grafo G , e dunque (G, k) è una istanza-sì di VC.

Concludendo, $\text{VC} \leq_p \text{FVS}$, pertanto FVS è NP-hard, e dunque NP-completo.

Esercizio 12. Si consideri il problema LONGEST PATH: dato un grafo non diretto $G = (V, E)$ ed un numero $k \in \mathbb{N}$, esiste un percorso semplice (senza nodi ripetuti) che utilizza almeno k archi? Dimostrare che il problema è NP-completo.

Soluzione: Il problema LONGEST PATH (LP) è verificabile polinomialmente. Infatti un certificato consiste in una lista di nodi del grafo. Il verificatore deve controllare che la lista contenga k nodi, che non contenga nodi ripetuti, e che tra ciascun nodo ed il successivo esista

un arco nel grafo. È immediato osservare come ciascuna di queste operazioni può essere svolta in tempo polinomiale, dunque LP è in NP.

Per dimostrare che il problema è NP-hard consideriamo il problema UNDIRECTED HAMILTONIAN PATH (UHP), che già sappiamo essere NP-completo. Una istanza di UHP è costituita da un grafo non diretto G ed una coppia di nodi s e t ; la questione è se esiste un percorso semplice tra s e t che attraversa tutti i nodi del grafo. Per ridurre questo problema a LP possiamo osservare che un percorso semplice che attraversa n nodi deve avere $n - 1$ archi. Dunque richiedendo nell'istanza di LP che il percorso sia lungo almeno il numero di nodi del grafo meno uno restringe in pratica la ricerca ad un percorso hamiltoniano. Si ha però una complicazione: a differenza di UHP, una istanza di LP non contiene una coppia di nodi selezionati come termini del percorso.

Dato un grafo G ed una coppia di nodi s e t , consideriamo il grafo G' ottenuto da G aggiungendo due nodi s' e t' e due archi (s, s') e (t, t') . Consideriamo l'istanza di LP costituita da G' e dal numero $n + 1$, ove n è il numero di nodi del grafo G . Ovviamente la riduzione è calcolabile in tempo polinomiale.

Supponiamo che esista un percorso halmitoniano in G tra s e t . Dunque nel grafo G' esiste un percorso semplice tra i nodi s e t di lunghezza $n - 1$. Considerando i due archi aggiuntivi possiamo costruire un percorso semplice entro G' di lunghezza $n + 1$. Dunque $(G', n + 1)$ è una istanza-sì di LP. Viceversa, supponiamo che esista un percorso semplice di lunghezza $n + 1$ entro G' : poiché G ha esattamente $n + 2$ nodi, il percorso deve attraversare tutti i nodi del grafo. Poiché inoltre i nodi s' e t' hanno un solo arco incidente, essi devono necessariamente essere i terminali del percorso entro G' . Considerando dunque i successivi nodi s e t adiacenti a s' e t' , esiste in G' un percorso di lunghezza $n - 1$ tra s e t . Questo percorso esiste anche in G tra i nodi s e t , e dunque (G, s, t) è una istanza-sì di UHP.

Pertanto $\text{UHP} \leq_P \text{LP}$, e dunque LP è NP-completo.

Automi e Linguaggi (M. Cesati)

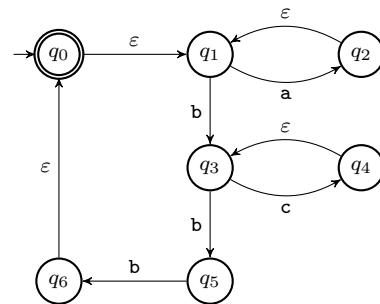
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 1 settembre 2020

(La prova è stata svolta “a libri aperti”.)

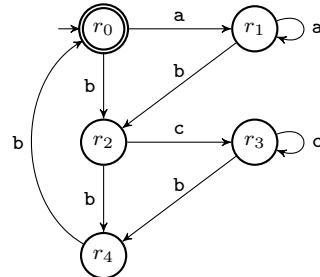
Esercizio 1. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((a^*bc^*)bb)^*$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



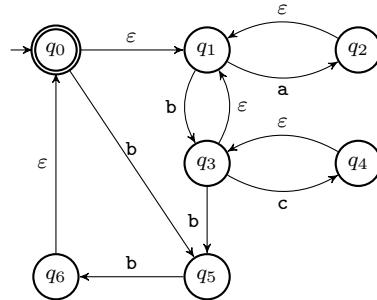
Un automa deterministico equivalente è il seguente:

$$\begin{aligned}
 r_0 &= \{q_0, q_1\}, \{q_1, q_0, q_6\} \\
 r_1 &= \{q_1, q_2\} \\
 r_2 &= \{q_3\} \\
 r_3 &= \{q_3, q_4\} \\
 r_4 &= \{q_5\}
 \end{aligned}$$



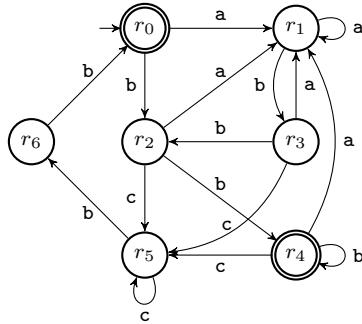
Esercizio 2. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((a^*bc^*)^*bb)^*$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



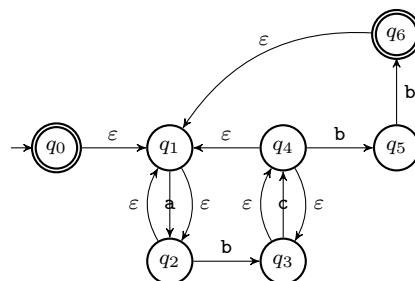
Un automa deterministico equivalente è il seguente:

$$\begin{aligned}
 r_0 &= \{q_0, q_1\} \\
 r_1 &= \{q_1, q_2\} \\
 r_2 &= \{q_1, q_3, q_5\} \\
 r_3 &= \{q_1, q_3\} \\
 r_4 &= \{q_0, q_1, q_3, q_5, q_6\} \\
 r_5 &= \{q_3, q_4\} \\
 r_6 &= \{q_5\}
 \end{aligned}$$



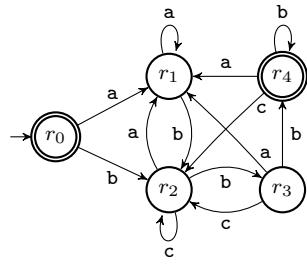
Esercizio 3. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((a^*bc^*)^+bb)^*$.

Soluzione: [Rev. 17.06.2022] La notazione x^+ è una abbreviazione per xx^* . L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



Un automa deterministico equivalente è il seguente:

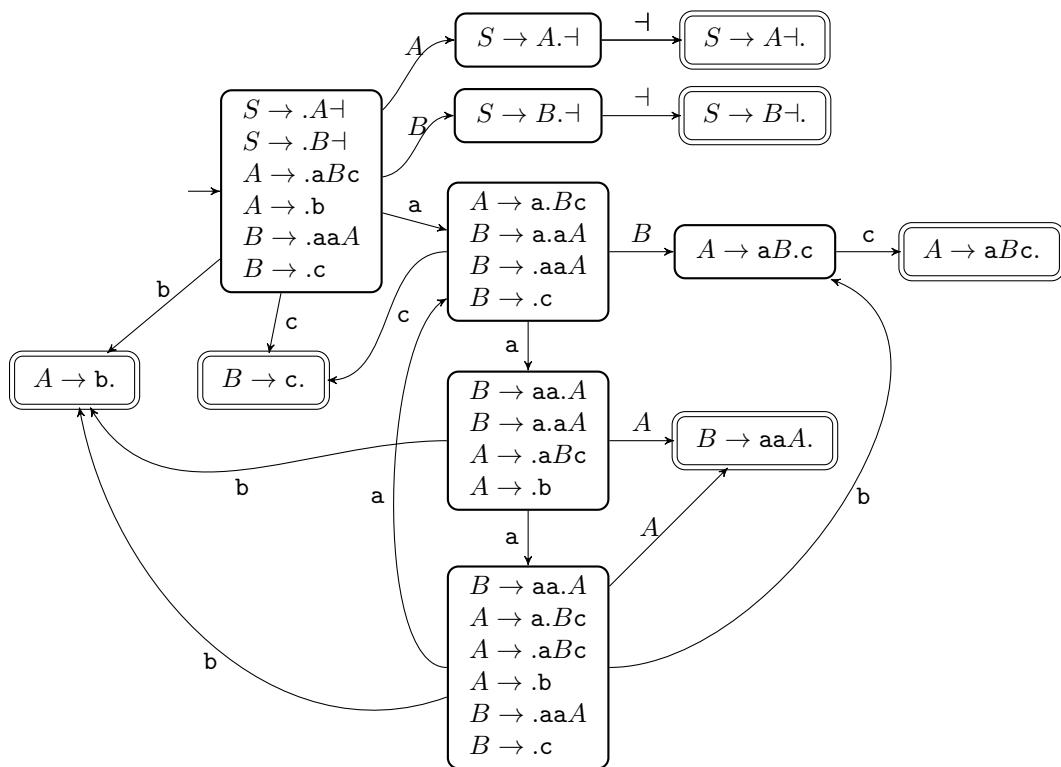
$$\begin{aligned}
 r_0 &= \{q_0, q_1, q_2\} \\
 r_1 &= \{q_1, q_2\} \\
 r_2 &= \{q_1, q_2, q_3, q_4\} \\
 r_3 &= \{q_1, q_2, q_3, q_4, q_5\} \\
 r_4 &= \{q_1, q_2, q_3, q_4, q_5, q_6\}
 \end{aligned}$$



Esercizio 4. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{aligned}
 S &\rightarrow A\dashv \quad | \quad B\dashv \\
 A &\rightarrow aBc \quad | \quad b \\
 B &\rightarrow aaA \quad | \quad c
 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:

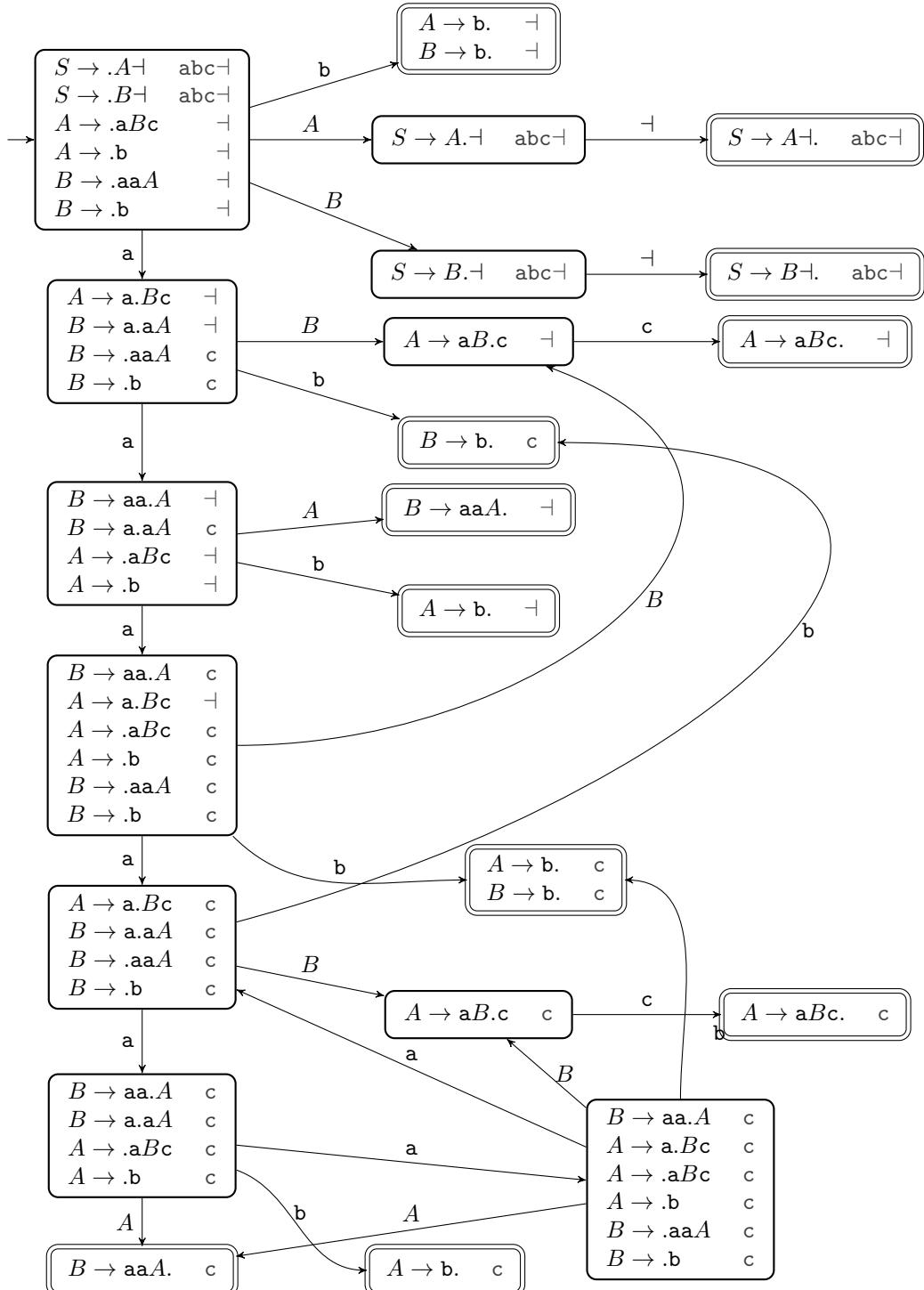


Poiché ogni stato finale ha una sola regola, quella completata, il DK-test risulta soddisfatto, pertanto la grammatica analizzata è deterministica.

Esercizio 5. Determinare se la seguente grammatica CFG con variabile iniziale S è LR(1):

$$\begin{aligned} S &\rightarrow A \dashv \quad | \quad B \dashv \\ A &\rightarrow aBc \quad | \quad b \\ B &\rightarrow aaA \quad | \quad b \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK_1 -test, ottenendo così il seguente diagramma:

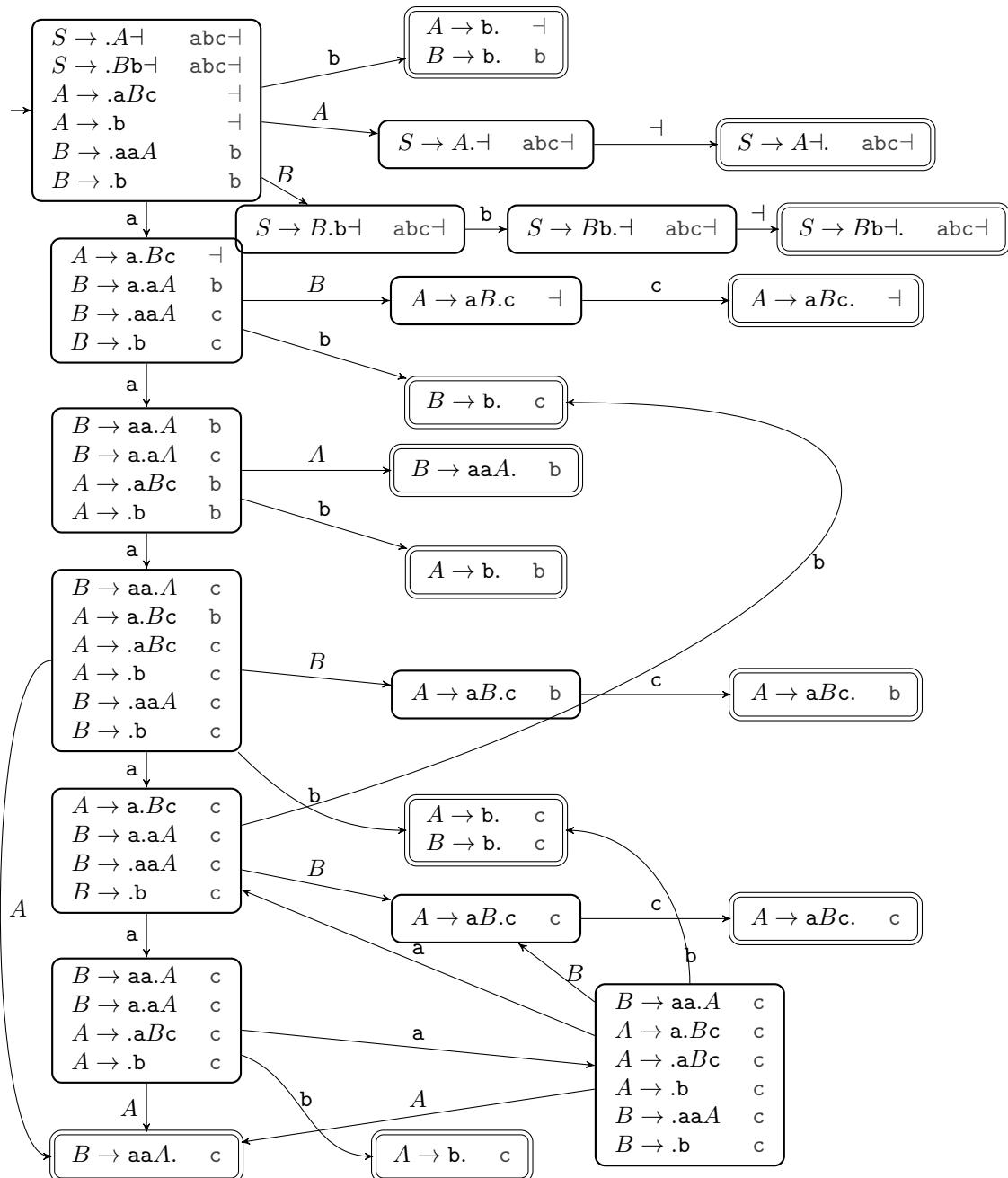


Poiché lo stato finale più in alto ha due regole completate consistenti, il DK₁-test non risulta soddisfatto, pertanto la grammatica analizzata non è LR(1).

Esercizio 6. Determinare se la seguente grammatica CFG con variabile iniziale S è LR(1):

$$\begin{aligned} S &\rightarrow A\vdash \quad | \quad Bb\vdash \\ A &\rightarrow aBc \quad | \quad b \\ B &\rightarrow aaA \quad | \quad b \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK₁-test:



Poiché lo stato finale al centro della figura ha due regole completate consistenti, il DK₁-test non risulta soddisfatto, pertanto la grammatica analizzata non è LR(1).

Esercizio 7. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing con alfabeto di ingresso Σ :

$L = \{\langle M \rangle \mid M \text{ è una DTM ed esiste } x \in \Sigma^* \text{ tale che } M(x) \text{ scrive un "blank" in una cella contenente un simbolo di } \Sigma\}$.

L è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Osserviamo preliminarmente che non è possibile applicare il teorema di Rice: infatti la proprietà che caratterizza il linguaggio L , pur essendo non banale, non è realmente del linguaggio riconosciuto dalle macchine di Turing ma delle macchine di Turing stesse. Infatti, si consideri $M \in L$, dunque esista un input $x \in \Sigma^*$ tale che $M(x)$ scrive il simbolo “blank” in una cella contenente un simbolo di Σ . Consideriamo ora una DTM M' identica ad M tranne che M' include un nuovo simbolo di nastro “#”. Inoltre ogni transizione di M in cui viene scritto il simbolo “blank” viene sostituita in M' da una analoga transizione in cui viene scritto il simbolo “#”. Infine, per ogni transizione in cui viene letto il simbolo “blank” viene aggiunta una analoga transizione in cui viene letto il simbolo “#”. È immediato ora verificare che $M' \notin L$ e che $L(M) = L(M')$.

L non è decidibile. Per dimostrarlo, supponiamo per assurdo che lo sia, e sia D una DTM che decide il linguaggio L . Consideriamo il problema indecidibile \mathcal{A}_{TM} : data una istanza $\langle M, w \rangle$, $\langle M, w \rangle \in \mathcal{A}_{\text{TM}}$ se e solo se $M(w)$ accetta. Si consideri ora la seguente DTM:

P= “On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. From $\langle M, w \rangle$, build the encoding of the following DTM R , which never writes a blank symbol over an input symbol except when explicitly stated:

R=“On any input x :

- a. If $x \neq w$, then halt
- b. Run $M(w)$
- d. If $M(w)$ rejects, then halt
- e. If $M(w)$ accepts, then:
 - f. Write an input symbol on tape
 - g. Write a blank symbol over the input symbol
 - h. Halt

2. Run D on input $\langle R \rangle$
3. If $D(\langle R \rangle)$ accepts, then accept, else reject"

La DTM R utilizza un simbolo di “blank” differente da quello utilizzato per codificare il “blank” di M . Se $D(\langle R \rangle)$ accetta, allora $\langle R \rangle \in L$, e dunque esiste un input x tale che $R(x)$ scrive un “blank”: necessariamente $x = w$ e $M(w)$ accetta. Se invece $D(\langle R \rangle)$ rifiuta, allora $\langle R \rangle \notin L$, dunque non esiste alcun input x per il quale $R(x)$ scrive un blank. In particolare, se $x = w$ allora $M(w)$ o rifiuta o non termina, quindi non accetta. La DTM P dunque decide \mathcal{A}_{TM} , ma questo è assurdo perché tale problema è indecidibile. Pertanto, resta dimostrato che L è indecidibile.

Esercizio 8. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing:

$$L = \{\langle M \rangle \mid M \text{ accetta almeno 15 stringhe di ingresso}\}.$$

Dimostrare che L è ricorsivamente enumerabile (ossia Turing-riconoscibile) ma non decidibile.

Soluzione: Per dimostrare che L è ricorsivamente enumerabile è sufficiente esibire un riconoscitore per tale linguaggio:

R = “On input $\langle M \rangle$, where M is a TM with input alphabet Σ :

1. Make room for two counters i, n on the tape
2. **for** $i = 1$ **to** $+\infty$:
3. Let $n \leftarrow 0$
4. For each string s among the first i strings in Σ^* :
 5. Run M on s for at most i steps
 6. If in the previous step $M(s)$ accepted, then let $n \leftarrow n + 1$
 7. If $n \geq 15$, then accept

R utilizza la classica tecnica per evitare di entrare in un ciclo infinito durante la simulazione di una computazione $M(s)$ non terminante: per ciascun valore di i , R esegue i passi di M sulle prime i stringhe in string order di Σ^* . Se esistono almeno 15 stringhe accettate da M , consideriamo la quindicesima di esse in ordine lessicografico, e sia il suo indice I . Sia inoltre N il numero massimo di passi che M impiega per accettare ciascuna delle prime 15 stringhe. Allora nella iterazione con $i = \max\{I, N\}$ R simula l'esecuzione accettante di M su ciascuna delle 15 stringhe, e dunque il valore del contatore n raggiunge il valore 15. Pertanto $R(\langle M \rangle)$ accetta. Se invece M accetta meno di 15 stringhe, il valore di n non potrà mai raggiungere il valore 15, e quindi R non accetterà mai.

Per dimostrare che L non è decidibile, è sufficiente evidenziare che esso ricade sotto le ipotesi del teorema di Rice. Infatti la proprietà che caratterizza L può essere sintetizzata come $|L(M)| \geq 15$, dunque è una proprietà dei linguaggi riconosciuti dalle TM piuttosto che delle TM stesse; è inoltre evidente che la proprietà non è banale (ad esempio, non vale per \emptyset , ma vale per Σ^*). Dunque si applica il teorema di Rice, che stabilisce che L non è decidibile.

Esercizio 9. Si consideri il seguente linguaggio contenente codifiche di macchine di Turing:

$$L = \{\langle M \rangle \mid M \text{ accetta meno di 15 stringhe di ingresso}\}.$$

Dimostrare che L non è ricorsivamente enumerabile (ossia non è Turing-riconoscibile).

Soluzione: Per dimostrare che L non è ricorsivamente enumerabile è conveniente ragionare sulla decidibilità del suo linguaggio complemento: $L^c = A \cup B$, ove

$$\begin{aligned} A &= \{\langle M \rangle \mid M \text{ accetta almeno 15 stringhe di ingresso}\} \\ B &= \{X \mid \text{la stringa } X \text{ non codifica una TM}\}. \end{aligned}$$

Possiamo assumere che il linguaggio B sia decidibile, in quanto il controllo sulla correttezza formale della codifica di una TM è una operazione di routine in tutte le TM che presentiamo.

Possiamo dimostrare che il linguaggio A è ricorsivamente enumerabile ma non decidibile. Per questa dimostrazione si veda l'esercizio precedente.

Poiché A è ricorsivamente enumerabile, allora anche L^c lo è, in quanto l'unione di un linguaggio ricorsivamente enumerabile e di un linguaggio decidibile è ricorsivamente enumerabile. Supponiamo per assurdo che L sia ricorsivamente enumerabile; di conseguenza, L^c sarebbe decidibile. Poiché $A = L^c \setminus B$ (in quanto $A \cap B = \emptyset$), allora anche A sarebbe decidibile. La contraddizione prova che L non è ricorsivamente enumerabile.

Esercizio 10. Si consideri il seguente problema: data una lista di n insiemi S_1, S_2, \dots, S_n ed un numero $k \in \mathbb{N}$, decidere se esiste una collezione di k insiemi dalla lista tali che nessun elemento è contenuto in due di questi k insiemi. Ad esempio, gli insiemi $\{1, 3, 5\}, \{1, 2, 3\}, \{2, 4\}, \{2, 5, 7\}, \{6\}$ e $k = 3$ sono una “istanza-sì” in quanto gli insiemi $\{1, 3, 5\}, \{2, 4\}$ e $\{6\}$ non hanno elementi in comune. Dimostrare che il linguaggio corrispondente a questo problema decisionale è NP-completo.

Soluzione: Il problema in questione, conosciuto come SET PACKING, è verificabile in tempo polinomiale. Infatti, data una istanza con n insiemi S_i e da un intero $k > 0$, un certificato per

l'istanza è costituito da k insiemi S_{i_1}, \dots, S_{i_k} tali che $S_{i_j} \cap S_{i_h} = \emptyset$ per ogni $1 \leq j < h \leq k$. Poiché il certificato include un sottoinsieme degli elementi della istanza, la sua dimensione non è superiore a quella della istanza. Inoltre una TM che riceve come input sul nastro una istanza del problema ed un certificato può facilmente controllare in tempo polinomiale che tutti gli insiemi del certificato siano insiemi della istanza e che nessuna coppia di insiemi del certificato abbia un elemento in comune. Pertanto SET PACKING appartiene alla classe NP.

Per dimostrare che SET PACKING è NP-hard, consideriamo una riduzione polinomiale dal problema NP-completo INDEPENDENT SET. Sia dunque (G, k) una istanza di INDEPENDENT SET, ove $G = (V, E)$ è un grafo con insieme di vertici V ed un insieme di archi E , e $k > 0$ è un intero. Fissiamo un ordinamento qualunque degli archi di G : $E = \{e_1, e_2, \dots, e_m\}$. Per ciascun nodo $v \in V$, includiamo nella collezione \mathcal{S} della istanza di SET PACKING l'insieme di numeri $S_v = \{x \in \mathbb{N} \mid \text{l'arco } e_x \text{ incide su } v\}$. In altri termini, \mathcal{S} contiene tanti insiemi quanti sono i nodi di G , e ciascuno di essi è costituito dagli indici degli archi incidenti sul nodo stesso. L'istanza ridotta di SET PACKING corrispondente a (G, k) è (\mathcal{S}, k) . È immediato verificare che è possibile costruire una istanza ridotta (\mathcal{S}, k) in tempo polinomiale nella dimensione di (G, k) .

Supponiamo che (G, k) sia una istanza-sì di INDEPENDENT SET: pertanto esiste un sottoinsieme di nodi $U \subseteq V$ di dimensione almeno k tale che non esiste alcun arco in G tra i nodi in U . Consideriamo quindi gli insiemi S_v in \mathcal{S} tali che $v \in U$: questi sottoinsiemi rappresentano un certificato valido per SET PACKING. Se infatti vi fosse un elemento x in comune tra due sottoinsiemi S_u e S_v nel certificato, allora tra i nodi u e v di G esisterebbe un arco e_x , ma questo è impossibile perché $u, v \in U$, che è una soluzione di INDEPENDENT SET. Viceversa, dato un certificato che testimonia sull'esistenza di una soluzione per l'istanza ridotta di SET PACKING, l'insieme di nodi di G corrispondenti agli insiemi nel certificato costituiscono necessariamente un insieme indipendente di dimensione k per G . Concludiamo pertanto che SET PACKING è NP-hard, e quindi NP-completo.

Esercizio 11. Si consideri il seguente problema: data una lista (multinsieme) di numeri positivi S , determinare se esiste una partizione di S in due multinsiemi A e B ($A \cup B = S$, $A \cap B = \emptyset$) tali che la somma degli elementi di A coincide con la somma degli elementi di B . Dimostrare che il linguaggio corrispondente è NP-completo.

Soluzione: Questo problema è noto, nella sua forma più generale in cui i numeri sono interi sia positivi che negativi, con il nome di PARTITION.

È facile dimostrare che PARTITION è verificabile in tempo polinomiale. Infatti data una istanza S consistente in una collezione di numeri interi (indifferentemente positivi o negativi), un

certificato consiste in un singolo multinsieme A . Il verificatore in tempo polinomiale controlla che il multinsieme A è contenuto in S (ossia che tutti gli elementi in A sono anche elementi di S), e che la somma degli elementi di A coincide con la somma degli elementi di S che non sono in A . Poiché la dimensione del certificato non è superiore alla dimensione dell'istanza ed il verificatore esegue semplici somme di numeri interi, si conclude che PARTITION (e quindi anche la sua restrizione agli interi positivi) è in NP.

Per dimostrare che PARTITION è NP-hard mostriamo una riduzione polinomiale dal problema NP-completo SUBSET SUM. Sia data dunque una istanza (S, t) di SUBSET SUM, ove S è un multinsieme di interi e t è un intero. Sia $\mu = \sum_{x \in S} x$ la somma di tutti gli elementi in S . L'istanza ridotta di PARTITION consiste nello stesso multinsieme S con un elemento y in più, precisamente il valore intero $y = 2t - \mu$: $P = S \cup \{y\}$. La somma di tutti gli elementi di P è $\mu + (2t - \mu) = 2t$. È evidente che la funzione che trasforma una istanza di SUBSET SUM in una istanza di PARTITION è calcolabile in tempo polinomiale.

Supponiamo che (S, t) sia una istanza-sì di SUBSET SUM, e quindi che esiste un sottoinsieme T degli elementi di S la cui somma è esattamente t : $\sum_{x \in T} x = t$. Si consideri allora la partizione di P costituita da T da una parte, e da $S \setminus T \cup \{y\}$ dall'altra. La somma degli elementi del secondo multinsieme è $\mu - t + (2t - \mu) = t$. Quindi P è una istanza-sì di PARTITION.

Supponiamo ora che P sia una istanza-sì di PARTITION. Pertanto esistono due sotto-multinsiemi A e B tali che $A \cup B = P$ e $A \cap B = \emptyset$, e tali che $\sum_{x \in A} x = \sum_{x \in B} x = t$. L'elemento y deve appartenere ad A oppure a B ; senza perdita di generalità supponiamo sia in A . Pertanto B è costituito da elementi di S la cui somma è esattamente t . Quindi (S, t) è una istanza-sì di SUBSET SUM.

La trasformazione mostrata è dunque una riduzione polinomiale, quindi PARTITION è NP-hard, e quindi NP-completo.

Esercizio 12. Si consideri il seguente problema decisionale: dato un insieme $X \subset \mathbb{N}^2$ di coppie (v, w) e due numeri $C, W \in \mathbb{N}$, esiste un sottoinsieme $S \subseteq X = \{(v_1, w_1), \dots, (v_m, w_m)\}$ tale che $\sum_{i=1}^m v_i \geq C$ e $\sum_{i=1}^m w_i \leq W$? Dimostrare che il linguaggio corrispondente è NP-completo.

Soluzione: Questo problema è noto con il nome KNAPSACK (“zaino” o “bisaccia”): infatti può essere considerato come l'equivalente del problema di riempire un contenitore avente una capacità finita W in modo che il valore degli oggetti inseriti sia il maggiore possibile (nella versione decisionale, sia maggiore di una soglia C).

Il problema KNAPSACK è verificabile in tempo polinomiale. Infatti un certificato per una istanza (X, C, W) è costituita da un insieme $S \subset \mathbb{N}^2$ con non più di $|X|$ coppie. Il verificatore controlla che tutte le coppie nel certificato siano incluse in X , che siano tutte distinte, che la somma dei primi elementi delle coppie sia non inferiore a C , e che la somma dei secondi elementi delle coppie sia non superiore a W . Considerata che la dimensione del certificato è inferiore alla dimensione dell'istanza e che il verificatore effetta semplici somme e confronti di interi, KNAPSACK è in NP.

Consideriamo una riduzione dal problema SUBSET SUM: sia (S, t) una istanza costituita da un multinsieme di numeri interi $S = (s_1, s_2, \dots, s_n)$ ed un numero “target” t . La corrispondente istanza ridotta di KNAPSACK è semplicemente (K, t, t) , ove $K = ((s_1, s_1), (s_2, s_2), \dots, (s_n, s_n))$ (quindi ogni intero S è trasformato in una coppia avente lo stesso intero come primo e come secondo elemento).

Sia (S, t) una istanza-sì di SUBSET SUM: dunque esiste un sotto-multinsieme T di S tale che $\sum_{x \in T} x = t$. Perciò il corrispondente sotto-multinsieme di coppie $L = ((x, x) | x \in T)$ è tale che la somma dei primi elementi è $t (= C)$ e la somma dei secondi elementi è $t (= W)$. L'esistenza di L implica che (K, t, t) è una istanza-sì di KNAPSACK.

Sia invece (K, t, t) una istanza-sì di KNAPSACK; dunque esiste un sotto-multinsieme L di K tale che $\sum_{(x,x) \in L} x \geq t$ e $\sum_{(x,x) \in L} x \leq t$. Perciò $\sum_{(x,x) \in L} x = t$. Se dunque consideriamo $T = (x | (x, x) \in L)$, la somma degli elementi del sotto-multinsieme T di S ha somma esattamente t . Dunque (S, t) è una istanza-sì di SUBSET SUM.

L'esistenza della riduzione polinomiale da SUBSET SUM a KNAPSACK dimostra che quest'ultimo è NP-hard, e di conseguenza NP-completo.

Automi e Linguaggi (M. Cesati)

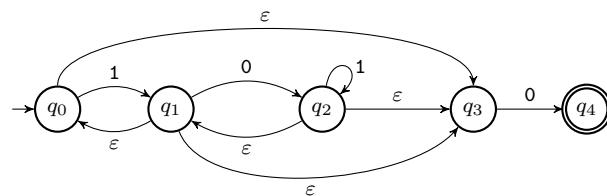
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 16 settembre 2020

(La prova è stata svolta “a libri aperti”.)

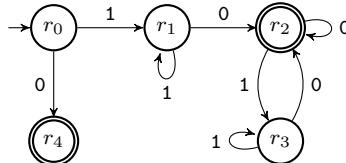
Esercizio 1. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $(1(01^*)^*)^*0$.

Soluzione: [Corr. 26.04.2022] L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:

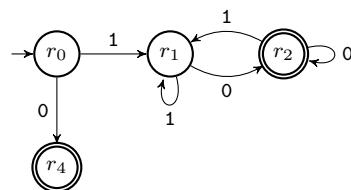


Un automa deterministico equivalente è il seguente:

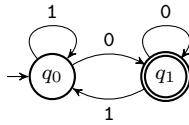
$$\begin{aligned}
 r_0 &= \{q_0, q_3\} \\
 r_1 &= \{q_0, q_1, q_3\} \\
 r_2 &= \{q_0, q_1, q_2, q_3, q_4\} \\
 r_3 &= \{q_0, q_1, q_2, q_3\} \\
 r_4 &= \{q_4\}
 \end{aligned}$$



In effetti gli stati r_1 e r_3 sono identici, e quindi è possibile ottenere un DFA equivalente con quattro stati:



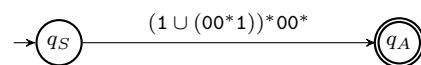
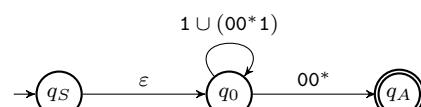
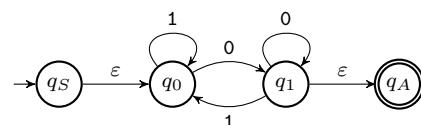
Esercizio 2. Determinare una espressione regolare equivalente all'automa



Soluzione: L'automa nel testo è molto semplice, ed è possibile determinare facilmente che il linguaggio riconosciuto dall'automa sono tutte le stringhe binarie che terminano con 0. Dunque una espressione regolare equivalente è ad esempio $(0 \cup 1)^*0$. Poiché però questa espressione non è stata derivata con un metodo formale, è necessario dimostrare la sua equivalenza con l'automa. In effetti, l'asserto segue dimostrando che:

1. Ogni stringa accettata dall'automa deve terminare con 0. Infatti, l'unico stato di accettazione è q_1 , e l'unico simbolo che può portare in q_1 è 0. Inoltre q_1 non è lo stato iniziale, dunque la stringa vuota (che non termina con 0) non è accettata.
2. Ogni stringa che termina con 0 è accettata dall'automa. Cominciamo con l'osservare che tutti gli stati hanno transizioni uscenti sia per il simbolo 0 che per il simbolo 1. Pertanto, tutte le stringhe con simboli in $\{0, 1\}$ sono totalmente “consumate” dall'automa. Si consideri ora una stringa s che termina con 0: lo stato seguente alla lettura dell'ultimo 0 non può essere diverso da q_1 , poiché questo è l'unico simbolo a cui si può arrivare leggendo uno 0. Poiché q_1 è uno stato di accettazione, la stringa s viene accettata.

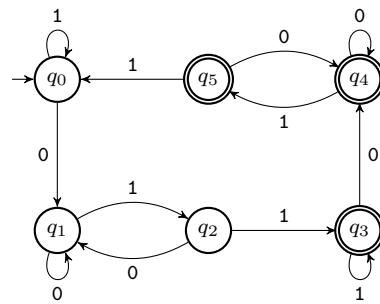
L'esercizio può essere svolto anche in modo più meccanico utilizzando la procedura di conversione da un GNFA ad una espressione regolare. Si consideri ad esempio la seguente conversione:



L'espressione regolare $(1 \cup (00^*1))^*00^*$ è una soluzione dell'esercizio. È possibile semplificare l'espressione considerando che 00^* è equivalente a 0^*0 , e che $(1 \cup 0^*01)$ è equivalente a (0^*1) . Dunque una espressione regolare equivalente è $(0^*1)^*0^*0$, che a sua volta è equivalente a $(0 \cup 1)^*0$.

Esercizio 3. Si consideri $A = \{x \in \{0, 1\}^* \mid \text{il numero di sequenze '011' entro } x \text{ è dispari}\}$. Ad esempio, $\varepsilon \notin A$, $10111 \in A$ e $101100110 \notin A$. Il linguaggio A è regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A è regolare. Per dimostrarlo, consideriamo il seguente DFA D :



Dimostriamo che D accetta $x \in \{0, 1\}^*$ se e solo se $x \in A$. La dimostrazione è per induzione sulla lunghezza $|x|$. Se $|x| \leq 2$, allora necessariamente $x \notin A$ perché x non può contenere la sequenza '011' lunga 3; d'altra parte, non è possibile raggiungere uno stato di accettazione in D a partire dallo stato iniziale q_0 utilizzando meno di 3 passi, quindi $D(x)$ rifiuta. Se $|x| = 3$, allora $x \in A$ se e solo se $x = 011$, ed in effetti è facile verificare che questa stringa di 3 bit porta nello stato di accettazione q_3 , mentre ogni altra stringa lascia l'automa in uno stato di non accettazione. Supponiamo dunque che l'ipotesi induttiva sia vera per $|x| < n$, e consideriamo una stringa x lunga esattamente $n > 3$ bit. Sia y la stringa costituita dai primi $n - 1$ bit di x , e distinguiamo i seguenti casi:

1. y termina con '00', '10' o '11': in questo caso il bit finale di x non può formare una sequenza '011' con gli ultimi due bit di y , pertanto $x \in A$ se e solo se $y \in A$.

Supponiamo che $x \in A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere q_5 , in quanto non esistono transizioni che portano in q_5 leggendo 0, e non esistono due transizioni consecutive che possano portare in q_5 leggendo 11. Dunque lo stato di $D(y)$ è in $\{q_3, q_4\}$. Se ora l'ultimo bit di x fosse 0, l'ultimo stato di $D(x)$ sarebbe necessariamente

q_4 , quindi $D(x)$ accetterebbe. Se invece l'ultimo bit di x fosse 1, l'ultimo stato di $D(x)$ sarebbe necessariamente in $\{q_3, q_5\}$, e dunque $D(x)$ accetterebbe ancora. Pertanto $D(x)$ accetta.

Supponiamo che $x \notin A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto $D(y)$ lascia l'automa in uno stato di non accettazione, che però non potrebbe essere q_2 . Infatti non esistono transizioni che portano in q_2 leggendo 0, e non esistono due transizioni consecutive che possano portare in q_2 leggendo 11. Dunque lo stato di $D(y)$ è in $\{q_0, q_1\}$. Se ora l'ultimo bit di x fosse 0, l'ultimo stato di $D(x)$ sarebbe necessariamente q_1 , quindi $D(x)$ non accetterebbe. Se invece l'ultimo bit di x fosse 1, l'ultimo stato di $D(x)$ sarebbe necessariamente in $\{q_0, q_2\}$, e dunque $D(x)$ non accetterebbe ancora. Pertanto $D(x)$ non accetta.

2. y termina con '01' e l'ultimo bit di x è 0: anche in questo caso il bit finale di x non può formare una sequenza '011' con gli ultimi due bit di y , pertanto $x \in A$ se e solo se $y \in A$.

Supponiamo che $x \in A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere né q_3 né q_4 , in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo 01. Dunque lo stato finale di $D(y)$ è q_5 , e quindi lo stato finale di $D(x)$ è q_4 . Perciò $D(x)$ accetta.

Supponiamo che $x \notin A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto y lascia l'automa in uno stato di non accettazione, che però non potrebbe essere né q_0 né q_1 , in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo 01. Dunque lo stato finale di $D(y)$ è q_2 , e quindi lo stato finale di $D(x)$ è q_1 . Perciò $D(x)$ rifiuta.

3. y termina con '01' e l'ultimo bit di x è 1: in questo caso il bit finale di x forma una nuova sequenza '011' con gli ultimi due bit di y , pertanto $x \in A$ se e solo se $y \notin A$.

Supponiamo che $x \in A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto y lascia l'automa in uno stato di non accettazione, che però non potrebbe essere né q_0 né q_1 , in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo 01. Dunque lo stato finale di $D(y)$ è q_2 , e quindi lo stato finale di $D(x)$ è q_3 . Perciò $D(x)$ accetta.

Supponiamo che $x \notin A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere né q_3 né q_4 , in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo 01. Dunque lo stato finale di $D(y)$ è q_5 , e quindi lo stato finale di $D(x)$ è q_0 . Perciò $D(x)$ rifiuta.

In sintesi, abbiamo dimostrato per induzione che l'automa D accetta la stringa x se e solo se $x \in A$. Pertanto il linguaggio A è regolare.

Esercizio 4. Sia $\Sigma = \{0, 1, +\}$, e sia E il linguaggio su Σ generato dall'espressione regolare $(0 \cup 1)^+ (+(0 \cup 1)^+)^*$. Sia inoltre $\varphi : E \rightarrow \mathbb{N}$ la funzione che associa ad una stringa di E il valore intero corrispondente alla somma dei numeri binari rappresentati. Ad esempio, $0100+11 \in E$ ed inoltre $\varphi(0100+11) = 7$. Si consideri il linguaggio sull'insieme $\Sigma \cup \{\equiv\}$

$$F = \{x \equiv y \mid x, y \in E \text{ e } \varphi(x) = \varphi(y)\}.$$

Ad esempio, $0100+11 \equiv 101+10 \in F$ mentre $0100+11 \equiv 101+01 \notin F$. Il linguaggio F è libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio F non è libero dal contesto. Per dimostrarlo, supponiamo per assurdo che lo sia ed utilizziamo il *pumping lemma* per i CFL. Sia dunque $p > 0$ la *pumping length* di F . Consideriamo la stringa $s = 10^p \equiv 1(+1)^{2^p-1}$; essa appartiene ad F perché $\varphi(10^p) = 2^p = \varphi(1(+1)^{2^p-1})$. Poiché $|s| > p$, deve esistere una suddivisione $s = uvxyz$ tale che $uv^i xy^i z \in F$ per ogni i , $|vy| > 0$ e $|vxy| \leq p$. Consideriamo i seguenti casi:

1. la stringa vy contiene il simbolo ‘ \equiv ’: la stringa pompata verso l'alto o verso il basso contiene un numero di ‘ \equiv ’ diverso da uno, e dunque non può far parte di F .
2. vxy è interamente alla sinistra di ‘ \equiv ’ in s : poiché $|vxy| \leq p$, la stringa vxy è interamente composta da ‘0’. Pompando verso l'alto il valore φ della sottostringa a sinistra di ‘ \equiv ’ viene moltiplicato per una potenza di due (in quanto $|vy| > 0$ il numero di ‘0’ aumenta almeno di una unità). D'altra parte il valore φ della sottostringa a destra non cambia, e quindi la stringa risultante non può far parte di F .
3. vxy è interamente alla destra di ‘ \equiv ’ in s : pompando verso l'alto la stringa risultante non appartiene a F , in quanto o non è valida (nel caso si producano ‘+’ consecutivi) oppure il valore φ della sottostringa a destra di ‘ \equiv ’ aumenta (poiché vy contiene almeno un ‘1’) mentre il valore di φ per la sottostringa di sinistra non cambia.
4. il simbolo ‘ \equiv ’ è incluso in x : osserviamo che sia v che y devono essere non nulle, altrimenti si ricadrebbe nel caso in cui aumenterebbe il valore φ di una sola delle due sottostringhe. Consideriamo $i = 1$: poiché $|v| > 0$, il valore di φ della sottostringa di sinistra come minimo deve raddoppiare (infatti come minimo si aggiunge un ‘0’ significativo al numero di sinistra), quindi diventa $\geq 2^{p+1}$. D'altra parte, poiché $|vxy| \leq p$, y deve contenere meno di $p/2$ occorrenze della sottostringa ‘1+’ o ‘+1’, quindi il valore di

φ della sottostringa a destra non può essere maggiore di $2^p + p/2$. Poiché $p/2 < 2^p$ per ogni $p > 0$, ne consegue che le due sottostringhe hanno valori φ differenti e la stringa pompata non fa parte di F .

Poiché non esiste modo di suddividere la stringa s in modo da soddisfare il *pumping lemma* si ha una contraddizione. Resta dunque dimostrato che F non è CFL.

Esercizio 5. Date due stringhe non nulle s_1 e s_2 su uno stesso alfabeto Σ , diciamo che le due stringhe sono *totalmente differenti* ($s_1 \not\approx s_2$) se, denotato con $s[i]$ l' i -esimo carattere della stringa s , $s_1[i] = s_2[j]$ implica che $i \neq j$.

Sia $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, e si consideri il linguaggio $B = \{s_1 \# s_2 \mid s_1, s_2 \in \Sigma^*, 0 < |s_1| \leq |s_2| \text{ e } s_1 \not\approx s_2^R\}$. Ad esempio, $\varepsilon \notin B$, $\# \notin B$, $\mathbf{ca} \# \mathbf{cba} \in B$, e $\mathbf{bc} \# \mathbf{ab} \notin B$. Il linguaggio B è libero dal contesto (CFL)? Si giustifichi la risposta con una dimostrazione.

Soluzione: Il linguaggio B è libero dal contesto. Per dimostrarlo, si consideri la seguente CFG G con variabile iniziale S :

$$\begin{array}{lcl} S & \longrightarrow & \mathbf{aAb} \quad | \quad \mathbf{aAc} \quad | \quad \mathbf{bAa} \quad | \quad \mathbf{bAc} \quad | \quad \mathbf{cAa} \quad | \quad \mathbf{cAb} \\ A & \longrightarrow & \mathbf{aAb} \quad | \quad \mathbf{aAc} \quad | \quad \mathbf{bAa} \quad | \quad \mathbf{bAc} \quad | \quad \mathbf{cAa} \quad | \quad \mathbf{cAb} \quad | \quad B \\ B & \longrightarrow & \mathbf{Ba} \quad | \quad \mathbf{Bb} \quad | \quad \mathbf{Bc} \quad | \quad \# \end{array}$$

Dimostriamo che la grammatica genera tutte e sole le stringhe in B . In primo luogo osserviamo che ogni derivazione di G inizia applicando una regola che espande la variabile iniziale S e produce il primo carattere di s_1 (ossia $s_1[1]$) e l'ultimo carattere di s_2 (ossia $s_2^R[1]$).

Cominciamo con la dimostrazione che $L(G) \subseteq B$, procedendo per induzione sulla lunghezza di s_1 . Come base per l'induzione consideriamo il caso $|s_1| = 1$. Poiché $s_1[1]$ è generato da una regola che espande la variabile iniziale S seguita immediatamente dalla regola $A \longrightarrow B$, è immediato verificare che i caratteri $s_1[1]$ e $s_2^R[1]$ devono necessariamente essere differenti. L'applicazione delle regole che espandono B può solo aggiungere caratteri alla stringa s_2 . Pertanto $s_1 \not\approx s_2$ e $|s_2| \geq 1 = |s_1|$, quindi ogni stringa prodotta da G con $|s_1| = 1$ appartiene a B . Assumiamo ora come ipotesi induttiva che l'asserto sia vero per ogni stringa $u \# v$ generata da G con $|u| < n$, e consideriamo una stringa $s_1 \# s_2 \in L(G)$ tale che $|s_1| = n$. In ogni derivazione da S ad una stringa terminale deve essere sempre applicata la regola $A \longrightarrow B$, perché B è l'unica variabile che può espandere con una regola senza variabili a destra. Consideriamo la sequenza di derivazioni ottenute da quella che produce la stringa

$s_1 \# s_2$ rimuovendo la regola applicata subito prima $A \rightarrow B$. Poiché per ipotesi $|s_1| = n > 1$, tale regola ha la forma $A \rightarrow xAy$, con $x, y \in \Sigma$. Pertanto la derivazione ottenuta togliendo questa regola è ancora valida e produce una stringa $u \# wv$ con $u, v, w \in \Sigma^*$, $|u| = |v| = n - 1$, $s_1 = ux$ e $s_2 = wyv$. Per ipotesi induttiva si ha che $u \# wv \in B$, dunque $u \not\approx (wv)^R = v^R w^R$. Ciò significa che le stringhe u e v^R sono totalmente differenti. Inoltre la regola rimossa $A \rightarrow xAy$ garantisce che $x \neq y$. Perciò $ux \not\approx v^R y$, quindi $s_1 = ux \not\approx v^R yw^R = s_2^R$, e finalmente $s_1 \# s_2 \in B$.

Dimostriamo ora che $B \subseteq L(G)$, ossia che la grammatica è in grado di generare qualunque stringa appartenente a B . Procediamo per induzione sulla lunghezza della sottostringa a sinistra del carattere '#'. La base dell'induzione è il caso in cui tale sottostringa è lunga un carattere, ossia il caso di una stringa in B con la forma $x \# vy$, con $x, y \in \Sigma$. L'appartenenza in B implica che $x \not\approx (vy)^R = yv^R$, o equivalentemente che $x \neq y$. Pertanto, detta $m = |v|$, la grammatica G può derivare questa stringa utilizzando in sequenza le regole $S \rightarrow xAy$, $A \rightarrow B$, $B \rightarrow Bv[m]$, ..., $B \rightarrow Bv[1]$, $B \rightarrow \#$. Quindi $x \# vy \in L(G)$. Assumiamo ora come ipotesi induttiva che l'asserto sia vero per ogni stringa in B avente una sottostringa a sinistra del '#' di lunghezza inferiore a $n > 1$. Consideriamo una stringa $s_1 \# s_2 \in B$ con $|s_1| = n$. Sia $x = s_1[n] \in \Sigma$ l'ultimo carattere di s_1 , e sia $y = s_2^R[n] \in \Sigma$ (esiste certamente perché $|s_2| \geq |s_1|$); siano inoltre $u, v, w \in \Sigma^*$ tali che $s_1 = ux$ e $s_2 = wyv$, con $|u| = |v| = n - 1$. L'appartenenza in B implica che $s_1 \not\approx s_2^R$, ossia che $ux \not\approx (wyv)^R = v^R yw^R$. Di conseguenza, $u \not\approx v^R$, quindi $u \not\approx v^R w^R = (wv)^R$. Perciò $u \# wv \in B$, dunque per ipotesi induttiva $u \# wv \in L(G)$. Consideriamo la derivazione da S alla stringa $u \# wv$: deve necessariamente essere applicata la regola $A \rightarrow B$, altrimenti non si potrebbe produrre una stringa di soli terminali. Modifichiamo ora la derivazione inserendo subito prima dell'applicazione della regola $A \rightarrow B$ la regola $A \rightarrow xAy$: la regola esiste certamente perché $x \neq y$, e si può applicare perché si applica alla medesima variabile A . La nuova derivazione produce esattamente la stringa $ux \# wyv$, e quindi $s_1 \# s_2 \in L(G)$.

Esercizio 6. Si consideri l'insieme $\Sigma = \{a, b\}$ e la seguente grammatica G con variabile iniziale S :

$$S \rightarrow aSb \mid bS \mid Sa \mid \epsilon$$

Quale è l'insieme di stringhe $L(G)$ generate da G ? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio generato dalla grammatica include tutte le stringhe sull'alfabeto di terminali Σ , ossia $L(G) = \Sigma^*$. La dimostrazione è per induzione. Come base dell'induzione,

consideriamo che la stringa vuota ε è generata dalla grammatica grazie alla regola $S \rightarrow \varepsilon$. Supponiamo dunque come ipotesi induttiva che la grammatica sia in grado di generare tutte le stringhe in Σ^* di lunghezza inferiore a $n \geq 0$, e consideriamo una stringa x di lunghezza n . Possiamo distinguere i seguenti tre casi:

1. il primo carattere di x è ‘b’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow bS$: la stringa ottenuta è pertanto bS . Poiché $x = by$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l’ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera il primo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.
2. l’ultimo carattere di x è ‘a’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow Sa$: la stringa ottenuta è pertanto Sa . Poiché $x = ya$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l’ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera l’ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.
3. il primo carattere di x è ‘a’ e l’ultimo carattere di x è ‘b’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow aSb$: la stringa ottenuta è pertanto aSb . Poiché $x = aby$ ove $y \in \Sigma^*$ e $|y| = n - 2$, per l’ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera il primo e l’ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.

In tutti i tre casi si conclude che $x \in L(G)$. Pertanto resta dimostrato che $x \in L(G)$ per ogni possibile stringa di Σ^* , e quindi $L(G) = \Sigma^*$.

Esercizio 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 $A \in \text{NP}$, ma questo è immediato perché per ipotesi $A \in \text{P}$ e $\text{P}=\text{NP}$. Mostriamo adesso che A è NP-hard. Sia dunque $B \in \text{NP}$. Poiché $\text{P}=\text{NP}$, $B \in \text{P}$, 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.

Esercizio 8. Siano dati un grafo non diretto $G = (V, E)$, una coppia di nodi $a, b \in G$, ed un intero $k \in \mathbb{N}$. Si dimostri che il problema di determinare se esiste un percorso da a a b in G di lunghezza non superiore a k è in P.

Soluzione: Il problema è noto come SHORTEST PATH, ed è una generalizzazione del problema polinomiale PATH che richiede l'esistenza di un percorso tra una coppia di nodi a prescindere dalla sua lunghezza. In effetti una idea per dimostrare che SHORTEST PATH è in P consiste nel modificare l'algoritmo utilizzato per dimostrare che PATH è in P, in modo da considerare anche la lunghezza dei percorsi. Si osservi inoltre che non esiste una reale differenza tra la versione su grafi non diretti e quella su grafi diretti.

M= “On input $\langle G, a, b, k \rangle$, where G is a graph, $a, b \in V(G)$, $k \in \mathbb{N}$

1. Allocate a vector of lengths having $|V(G)|$ elements (one for each node)
2. Initialize all lengths to ∞
3. Set the length of node a to 0
3. For $i = 0$ to $|E(G)|$:
 4. For each edge $e \in E(G)$:
 5. If e links a node u with length i and a node v with length ∞ :
 6. Set the length of v to $i + 1$
 7. If b has length $\leq k$, accept; otherwise, reject”

La DTM inizia a marcare il nodo a con la lunghezza 0; poi marca tutti i nodi adiacenti ad a con la lunghezza 1; poi procede a marcare tutti i nodi adiacenti ai nodi di lunghezza 1 che non erano stati già marcati con la lunghezza 2; e così via. Il ciclo esterno è ripetuto per un numero di volte pari al numero di archi nel grafo (la lunghezza del più lungo percorso possibile), e ad ogni iterazione vengono scanditi tutti gli archi del grafo. Pertanto la complessità temporale

dell'algoritmo è $O(|E|^2)$. È inoltre evidente che alla fine il nodo b viene marcato con un valore $\leq k$ se e solo se esiste un percorso entro il grafo da a a b di lunghezza non superiore a k . Pertanto M è un decisore in tempo polinomiale, e quindi SHORTEST PATH è in P.

Si osservi che l'analogo problema LONGEST PATH, che richiede di verificare l'esistenza di un percorso di lunghezza non inferiore ad un valore dato, è NP-completo.

Esercizio 9. Uno stato interno r di una macchina di Turing M è detto *inutile* se non esiste alcuna stringa di input che possa portare la macchina M ad assumere lo stato interno r . Dimostrare che il linguaggio contenente le codifiche di tutte le macchine di Turing con uno o più stati inutili è indecidibile.

Soluzione: Cominciamo con l'osservazione evidente che la proprietà di avere o meno uno stato inutile appartiene alla particolare macchina di Turing, e non è propria del linguaggio riconosciuto dalla macchina. Ad esempio, consideriamo una TM che non ha stati inutili, e modifichiamo la sua descrizione in modo da aggiungere uno stato che non è mai utilizzato in alcuna transizione. Ovviamente la macchina modificata riconosce lo stesso linguaggio della macchina originale ma possiede uno stato inutile. Dunque non possiamo dimostrare quanto richiesto utilizzando il Teorema di Rice.

Per dimostrare l'indecidibilità del linguaggio contenente le codifiche di TM aventi almeno uno stato inutile costruiamo una riduzione dal problema di accettazione delle TM \mathcal{A}_{TM} . Come al solito dunque consideriamo una istanza $\langle M, w \rangle$ di \mathcal{A}_{TM} e mostriamo come costruire una TM N che ha uno stato inutile se e solo se $M(w)$ accetta. La difficoltà di questa costruzione, però, risiede nella macchina M stessa: infatti, dovendo simulare M , N include nei suoi stati interni anche gli stati interni di M . Se M avesse stati inutili, la macchina N potrebbe avere stati inutili anche se in effetti $M(w)$ non accettasse. Dobbiamo quindi fare attenzione a costruire N in modo da rendere ogni stato di M “utile”.

Siano Σ e Γ rispettivamente gli alfabeti di input e di nastro della macchina M ; la TM N utilizzerà $\Sigma' = \Sigma \cup \{\#\}$ e $\Gamma' = \Gamma \cup \{\#\}$, ove ‘ $\#$ ’ è un nuovo simbolo non utilizzato in M . Il simbolo ‘ $\#$ ’ serve ad implementare un ciclo in cui si entra in ogni stato interno di M : in pratica viene aggiunta una nuova transizione per ogni stato interno di M in cui leggendo ‘ $\#$ ’ si passa in uno stato interno di N che continua il ciclo.

Nella descrizione di N seguente si deve assumere che non vengono introdotti stati interni propri di N inutili a meno che questo che non sia affermato esplicitamente. In particolare, lo stato interno r di N è utilizzato solo quando esplicitamente menzionato. Inoltre si tenga presente che gli stati di accettazione e rifiuto di M non sono stati finali per N .

N = “On input x , where $x \in \Sigma^*$:

1. If $x = \#$:
2. Execute a loop and enter in every internal state of M
3. Halt
4. Run M on input w
5. If $M(w)$ accepts, enter in internal state r
6. Halt”

Supponiamo per assurdo che il problema di decidere se una TM ha stati interni inutili sia decidibile, e sia dunque R un decisore per tale problema. Consideriamo allora la seguente TM D :

D = “On input $\langle M, w \rangle$, where M is a TM and $w \in \Sigma^*$:

1. Build from $\langle M, w \rangle$ the encoding of the TM N
2. Run R on $\langle N \rangle$
3. If $R(\langle N \rangle)$ accepts, then reject
4. Otherwise, if $R(\langle N \rangle)$ rejects, then accept”

Se $R(\langle N \rangle)$ accetta, allora N ha almeno uno stato interno inutile. Per costruzione di N questo stato può essere unicamente r . Ciò significa che $M(w)$ non accetta. Se invece $R(\langle N \rangle)$ rifiuta, allora ogni stato interno di N è utile, quindi anche lo stato interno di r deve essere visitato per un certo input x . In effetti l'input x di N viene ignorato se è diverso da ‘#’. L'unica possibilità è che $M(w)$ accetta e quindi r sia utilizzato nel passo 5 di N . Poiché D complementa la decisione di R , D è in effetti un decisore per \mathcal{A}_{TM} , ma questo è in contraddizione con il fatto che \mathcal{A}_{TM} è indecidibile. Pertanto resta dimostrato che il problema di decidere se una TM ha stati interni inutili è indecidibile.

Esercizio 10. Si consideri il linguaggio costituito dalle codifiche delle formule booleane che hanno almeno due assegnazioni di verità che soddisfano la formula. Dimostrare che tale linguaggio è NP-completo.

Soluzione: Questo problema è generalmente chiamato DOUBLE SAT, ed è una generalizzazione molto semplice del problema SAT.

Si dimostra facilmente che DOUBLE SAT è in NP. Infatti, data una qualunque istanza $\langle \Phi \rangle$ che codifica una formula booleana, un certificato per l'esistenza di una soluzione è costituito da due liste di valori di verità. Il verificatore controlla che ciascuna lista contenga esattamente

un valore (vero o falso) per ciascuna variabile di Φ , e che entrambe le assegnazioni di verità soddisfino la formula Φ .

Per dimostrare che DOUBLE SAT è NP-hard consideriamo la seguente riduzione dal problema SAT: considerata una generica istanza $\langle\Phi\rangle$ di SAT, sia Φ' la formula booleana ottenuta da Φ aggiungendo una nuova variabile v e ponendo

$$\Phi' = \Phi \wedge (v \vee \bar{v}).$$

È immediato verificare che se la formula Φ è soddisfacibile allora esistono almeno due assegnazioni di verità che soddisfano Φ' : la assegnazione che soddisfa Φ estesa con $v = T$ e la stessa assegnazione estesa con $v = F$. Viceversa, se la formula Φ non è soddisfacibile, allora nemmeno Φ' può essere soddisfacibile, perché la variabile v non appare nella formula Φ e quindi non può contribuire a rendere quella parte della formula Φ' soddisfacibile.

Da tutto ciò si può concludere che DOUBLE SAT è NP-completo.

Esercizio 11. Si consideri un grafo non diretto $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Si dimostri che il problema di stabilire se il grafo G contiene un sottoinsieme W di nodi con $|W| \geq \frac{n}{2}$ tale che non esiste alcun arco tra i nodi di W è NP-completo.

Soluzione: Questo problema è generalmente noto con il nome HALF INDEPENDENT SET (HIS). Dimostrare che HIS è in NP è molto semplice. Data una istanza $\langle G \rangle$, un certificato per l'esistenza di una soluzione è una lista di nodi del grafo G . Il verificatore controlla che nella lista non vi siano nodi ripetuti, che la lista contenga almeno $|V(G)|/2$ nodi, e che ciascuna coppia di nodi nella lista sia non adiacente nel grafo G , ossia non esista tra essi un arco. Tutti questi controlli possono naturalmente essere effettuati in tempo polinomiale nella dimensione del grafo.

Per dimostrare che HIS è NP-hard mostriamo una riduzione dall'analogo problema NP-completo INDEPENDENT SET (IS). Si presti attenzione che le istanze di IS e di HIS hanno una differente struttura: le istanze di IS codificano un grafo G ed un numero intero k , mentre le istanze di HIS codificano soltanto un grafo. Dunque la riduzione deve “eliminare” il parametro k dall'istanza di IS.

La riduzione considera il valore del parametro k ed opera differentemente nei casi $k = n/2$, $k < n/2$, e $k > n/2$. Si ricordi che una riduzione è una funzione calcolabile in tempo polinomiale, e confrontare due valori interi è una operazione eseguibile in tempo polinomiale nella lunghezza delle codifiche dei valori.

1. $k = n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G \rangle$ (si elimina semplicemente il valore k dall'istanza). Infatti, una istanza-sì di IS è tale per cui esiste un sottoinsieme $W \subseteq V$ con $|W| \geq k = n/2$ tale che non esiste alcun arco tra i nodi in W ; il grafo è dunque anche una istanza-sì di HIS. Ovviamente vale anche il viceversa.
2. $k < n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $n - 2k$ nodi isolati. Sia $\langle G, k \rangle$ una istanza-sì di IS, dunque esiste un insieme indipendente $W \subseteq V$ con $|W| \geq k$. Nel grafo G' , W unito ai nuovi nodi costituisce un insieme indipendente di dimensione $k + (n - 2k) = n - k$ nodi. D'altra parte, G' contiene $n + (n - 2k) = 2(n - k)$ nodi, quindi G' è una istanza-sì di HIS.

Se invece $\langle G, k \rangle$ è una istanza-no di HIS, non esiste alcun insieme indipendente di dimensione maggiore o uguale a k in G . Nel grafo G' pertanto non esiste alcun insieme indipendente di dimensione maggiore o uguale a $n - k$, perché G' è ottenuto da G aggiungendo solo $n - 2k$ nodi e non rimuovendo alcun arco. Dunque G' è una istanza-no di HIS, perché nessun insieme indipendente può essere di dimensione maggiore o uguale alla metà dei $2(n - k)$ nodi di G' .

3. $k > n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $2k - n$ nodi ed i seguenti archi: tutti i nuovi nodi sono collegati tra loro (quindi costituiscono un insieme completo), inoltre ciascuno dei nuovi nodi è collegato a ciascuno dei nodi del grafo G . Ovviamente, una istanza-sì di IS è anche una istanza-sì di G' . Infatti G' ha $n + (2k - n) = 2k$ nodi, e l'insieme indipendente W con $|W| \geq k$ è anche un insieme indipendente di G' con almeno la metà dei nodi di G' .

Consideriamo ora una istanza-no di IS, dunque supponiamo che nel grafo G non esista alcun insieme indipendente di dimensione maggiore o uguale a k . Il grafo G' è ottenuto da G aggiungendo un sottografo completo ed aggiungendo tutti gli archi tra i vecchi ed i nuovi nodi. Pertanto, nessuno dei nuovi nodi può essere aggiunto ad un insieme indipendente di G . Si conclude dunque che non può esistere in G' un insieme indipendente avente la metà dei nodi di G' , quindi $\langle G' \rangle$ è una istanza-no di HIS.

È facile verificare che la dimensione dell'istanza $\langle G' \rangle$ è polinomialmente limitata dalla dimensione dell'istanza $\langle G, k \rangle$ (infatti, $k \leq n$, quindi in ogni caso $|V(G')| \leq 2n$). Inoltre la riduzione è calcolabile in tempo polinomiale. Quindi HIS è NP-hard, e di conseguenza NP-completo.

Esercizio 12. Si consideri un grafo non diretto $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Si dimostri che il problema di stabilire se il grafo G contiene un sottoinsieme W di nodi con $|W| \leq \frac{n}{2}$ tale che ogni arco di G contiene almeno un nodo in W è NP-completo.

Soluzione: Questo problema è generalmente noto con il nome HALF VERTEX COVER (HVC). Dimostrare che HVC è in NP è molto semplice. Data una istanza $\langle G \rangle$, un certificato per l'esistenza di una soluzione è una lista di nodi del grafo G . Il verificatore controlla che nella lista non vi siano nodi ripetuti, che la lista contenga non più di $|V(G)|/2$ nodi, e che ciascun arco di G sia collegato ad un nodo della lista. Tutti questi controlli possono naturalmente essere effettuati in tempo polinomiale nella dimensione del grafo.

Assumendo come dimostrato che il problema HALF INDEPENDENT SET (introdotto nell'esercizio precedente) è NP-hard, è immediato verificare che HVC è NP-hard. Infatti, un insieme di nodi W in un grafo G è un insieme indipendente se e solo se l'insieme di nodi $U = V(G) \setminus W$ è un ricoprimento tramite vertici (“vertex cover”). Dunque esiste un insieme indipendente di dimensione almeno $|V(G)|/2$ se e solo se esiste un ricoprimento tramite vertice di dimensione al più $|V(G)|/2$.

In alternativa, mostriamo una riduzione tra INDEPENDENT SET (IS) e HVC. Si presti attenzione che le istanze di IS e di HVC hanno una differente struttura: le istanze di IS codificano un grafo G ed un numero intero k , mentre le istanze di HVC codificano soltanto un grafo. Dunque la riduzione deve “eliminare” il parametro k dall'istanza di IS.

La riduzione considera il valore del parametro k ed opera differentemente nei casi $k = n/2$, $k < n/2$, e $k > n/2$. Si ricordi che una riduzione è una funzione calcolabile in tempo polinomiale, e confrontare due valori interi è una operazione eseguibile in tempo polinomiale nella lunghezza delle codifiche dei valori.

1. $k = n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G \rangle$ (si elimina semplicemente il valore k dall'istanza). Infatti, una istanza-sì di IS è tale per cui esiste un sottoinsieme $W \subseteq V$ con $|W| \geq k = n/2$ tale che non esiste alcun arco tra i nodi in W ; l'insieme $U = V \setminus W$ è un ricoprimento tramite vertici con $|U| \leq n - k = n/2$; il grafo è dunque anche una istanza-sì di HVC. Ovviamente vale anche il viceversa.
2. $k < n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $n - 2k$ nodi isolati. Sia $\langle G, k \rangle$ una istanza-sì di IS, dunque esiste un insieme indipendente $W \subseteq V$ con $|W| \geq k$. Nel grafo G' , W unito ai nuovi nodi costituisce un insieme indipendente W' di dimensione $k + (n - 2k) = n - k$ nodi. Pertanto $U = V(G') \setminus W' = V \setminus W$ ha $|U| = n - (n - k) = k$ nodi ed è un ricoprimento tramite vertici per G' . D'altra parte, G' contiene $n + (n - 2k) = 2(n - k)$ nodi, e $k < n/2$ implica $k < n - k$, quindi G' è una istanza-sì di HVC.

Se invece $\langle G, k \rangle$ è una istanza-no di HIS, non esiste alcun insieme indipendente di dimensione maggiore o uguale a k in G . Nel grafo G' pertanto non esiste alcun insieme indipendente di dimensione maggiore o uguale a $n - k$, perché G' è ottenuto da G aggiun-

gendo solo $n - 2k$ nodi e non rimuovendo alcun arco. Dunque in G' nessun ricoprimento tramite vertici può essere di dimensione minore o uguale a $2(n - k) - (n - k) = n - k$, e quindi $\langle G' \rangle$ è una istanza-no di HVC.

3. $k > n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $2k - n$ nodi ed i seguenti archi: tutti i nuovi nodi sono collegati tra loro (quindi costituiscono un insieme completo), inoltre ciascuno dei nuovi nodi è collegato a ciascuno dei nodi del grafo G . Ovviamente, una istanza-sì di IS è anche una istanza-sì di G' . Infatti G' ha $n + (2k - n) = 2k$ nodi, e l'insieme indipendente W con $|W| \geq k$ è anche un insieme indipendente di G' con almeno la metà dei nodi di G' ; pertanto il complemento di questo insieme è un ricoprimento tramite vertici con al più la metà dei nodi di G' .

Consideriamo ora una istanza-no di IS, dunque supponiamo che nel grafo G non esista alcun insieme indipendente di dimensione maggiore o uguale a k . Il grafo G' è ottenuto da G aggiungendo un sottografo completo ed aggiungendo tutti gli archi tra i vecchi ed i nuovi nodi. Pertanto, nessuno dei nuovi nodi può essere aggiunto ad un insieme indipendente di G . Si conclude dunque che non può esistere in G' un insieme indipendente avente la metà dei nodi di G' . Pertanto, ogni ricoprimento tramite vertici in G' deve contenere più della metà dei nodi di G' , dunque $\langle G' \rangle$ è una istanza-no di HVC.

È facile verificare che la dimensione dell'istanza $\langle G' \rangle$ è polinomialmente limitata dalla dimensione dell'istanza $\langle G, k \rangle$ (infatti, $k \leq n$, quindi in ogni caso $|V(G')| \leq 2n$). Inoltre la riduzione è calcolabile in tempo polinomiale. Quindi HVC è NP-hard, e di conseguenza NP-completo.

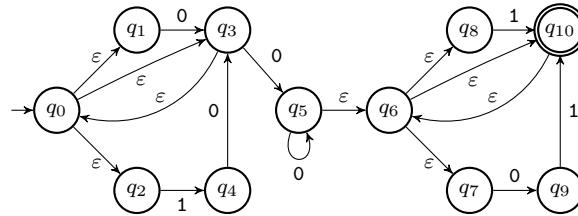
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 22 febbraio 2021

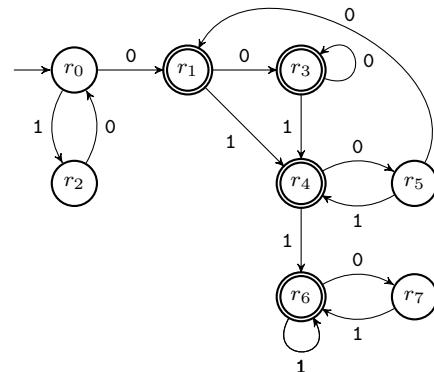
Esercizio 1 [6] Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $(0 \cup 10)^*00^*(1 \cup 01)^*$.

Soluzione: La costruzione di un automa non deterministico che riconosce il linguaggio regolare è meccanica; dopo qualche semplificazione minore si ottiene lo NFA:



Calcoliamo gli insiemi chiusura di ciascuno stato rispetto alle transizioni ϵ : $E(q_0) = E(q_3) = \{q_0, q_1, q_2, q_3\}$, $E(q_5) = \{q_5, q_6, q_7, q_8, q_{10}\}$, $E(q_6) = E(q_{10}) = \{q_6, q_7, q_8, q_{10}\}$; tutti gli altri insiemi chiusura contengono solo uno stato. La procedura di conversione dall'NFA al DFA produce il seguente automa:

$$\begin{aligned}
 r_0 &= \{q_0, q_1, q_2, q_3\} \\
 r_1 &= \{q_0, q_1, q_2, q_3, q_5, q_6, q_7, q_8, q_{10}\} \\
 r_2 &= \{q_4\} \\
 r_3 &= \{q_0, q_1, q_2, q_3, q_5, q_6, q_7, q_8, q_9, q_{10}\} \\
 r_4 &= \{q_4, q_6, q_7, q_8, q_{10}\} \\
 r_5 &= \{q_0, q_1, q_2, q_3, q_9\} \\
 r_6 &= \{q_6, q_7, q_8, q_{10}\} \\
 r_7 &= \{q_9\}
 \end{aligned}$$



Esercizio 2 [6] Dimostrare che la grammatica context-free seguente è ambigua:

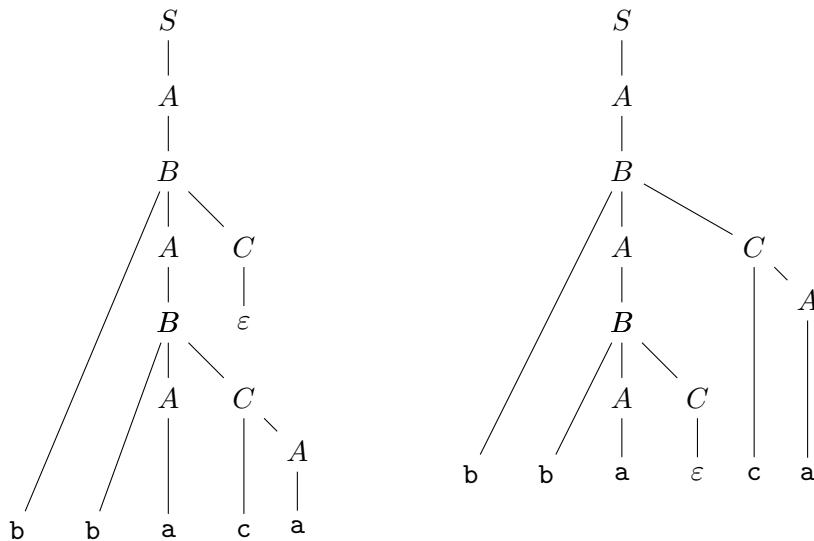
$$S \rightarrow A \quad A \rightarrow aA \mid B \mid a \quad B \rightarrow bAC \quad C \rightarrow cA \mid \epsilon$$

Soluzione: Una grammatica si definisce ambigua se esistono due derivazioni “leftmost” (o equivalentemente “rightmost”) della stessa stringa terminale. Consideriamo dunque la stringa terminale $bbaca$: essa può essere prodotta con le seguenti due derivazioni “leftmost”:

$$S \rightarrow A \rightarrow B \rightarrow bAC \rightarrow bBC \rightarrow bbACC \rightarrow bbaCC \rightarrow bbacAC \rightarrow bbacaC \rightarrow bbaca\epsilon$$

$$S \rightarrow A \rightarrow B \rightarrow bAC \rightarrow bBC \rightarrow bbACC \rightarrow bbaCC \rightarrow bba\epsilon C \rightarrow bbacA \rightarrow bbaca$$

Equivalentemente, la grammatica è certamente ambigua perché la stringa terminale $bbaca$ può essere prodotta con due differenti alberi sintattici (“parse tree”):



Esercizio 3 [6] Dimostrare che il linguaggio $L = \{0^n 1 0^m 1 0^q \mid n, m > 0, q = \max(n, m)\}$ non è context-free.

Soluzione: Supponiamo per assurdo che il linguaggio L sia context-free. In tal caso, per L varrebbe il pumping lemma per i linguaggi context-free, e dunque esisterebbe un valore $p > 0$ tale che tutte le stringhe $s \in L$ di lunghezza uguale o maggiore di p possono essere “pompatte” verso l’alto e verso il basso.

Consideriamo la stringa $s = 0^p 1 0^p 1 0^p \in L$, e dimostriamo che non è possibile determinare una suddivisione $s = uvxyz$ con $|vy| > 0$ e $|vxy| \leq p$ tale che $uv^i xy^i z \in L$ per ogni $i \geq 0$. Infatti una tale suddivisione deve necessariamente ricadere in uno dei seguenti casi:

1. La stringa vy include un carattere ‘1’: pompando verso l’alto o verso il basso la stringa risultante ha un numero di ‘1’ diverso da due, e quindi non fa parte di L .

2. La stringa vxy è totalmente inclusa nella sequenza di zeri più a sinistra (analogamente, nella sequenza di zeri centrale): pompando verso l'alto il numero di tali zeri aumenta, e dunque la stringa pompata ha la forma $0^s 1 0^p 0^p$ (ovvero $0^p 1 0^s 1 0^p$) con $s > p$; poiché $p \neq \max(s, p)$, $0^s 1 0^p 1 0^p \notin L$.
3. La stringa vxy include sia zeri nella sequenza più a sinistra che zeri nella sequenza centrale (considerando il caso 1, necessariamente $x = 1$). Pompando verso l'alto o verso il basso la stringa diventa $0^s 1 0^t 1 0^p$, con $s \neq p$ e $t \neq p$: in ogni caso non può far parte del linguaggio L perché $p \neq \max(s, t)$.
4. La stringa vxy include sia zeri nella sequenza centrale che zeri della sequenza a destra (e considerando il caso 1, $x = 1$). Pompando verso il basso si ottiene una stringa $0^p 1 0^s 1 0^t$ con $s < p$ e $t \neq \max(p, s) = p$, che dunque non può far parte di L .
5. La stringa vxy è totalmente inclusa nella sequenza di zeri più a destra: pompando verso l'alto o verso il basso si ottiene una stringa $0^p 1 0^p 1 0^t$ con $t \neq \max(p, p)$, che dunque non può appartenere a L .

In sintesi, per poter essere pompata la sottostringa vxy dovrebbe contenere elementi di tutte e tre le sequenze di zeri, ma ciò è impossibile perché la più corta di tali stringhe è costituita da $p + 4$ caratteri mentre $|vxy| \leq p$. Resta dunque dimostrato che il pumping lemma non è valido, e pertanto L non è context-free.

Esercizio 4 [10] Dimostrare che il linguaggio HALT_E , contenente le codifiche di tutte le macchine di Turing che su input vuoto terminano in un numero pari di passi, è indecidibile. Dare per assunto che il linguaggio della fermata delle macchine di Turing è indecidibile.

Soluzione: Innanzi tutto, poiché il problema della fermata delle macchine di Turing è indecidibile, è immediato stabilire che anche il problema della fermata delle TM che iniziano con il nastro vuoto è indecidibile: è sufficiente considerare che ogni istanza $(\langle M \rangle, x)$ è facilmente trasformabile in una istanza $\langle M' \rangle$ codificante una TM che prima scrive x sul nastro vuoto e poi simula il comportamento di M .

Si consideri dunque il linguaggio HALT contenente le codifiche di tutte le TM che terminano iniziando con il nastro vuoto: è evidente che ciascuna macchina in HALT termina in un numero finito di passi che può essere pari oppure dispari. Quindi:

$$\text{HALT} = \text{HALT}_E \cup \text{HALT}_O.$$

Supponiamo per assurdo che HALT_E sia decidibile, dunque che esista una TM D che decide tale linguaggio.

Si consideri ora una macchina di Turing R che riceve come input una codifica di una TM $T = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ e produce come output la codifica di una macchina di Turing $T' = (Q \cup \{q_h, q'_h\}, \Sigma, \Gamma, \delta', q_0, q_h, q'_h)$ in cui δ' contiene tutte le transizioni di δ ed in più, per ogni simbolo $\sigma \in \Gamma$, $\delta'(q_a, \sigma) = (q_h, \sigma, L)$ e $\delta'(q_r, \sigma) = (q'_h, \sigma, L)$. Si osservi che gli stati q_a e q_r di T' non sono più finali, mentre lo sono i nuovi stati $q_h, q'_h \notin Q$. Risulta immediato verificare che $T(\varepsilon)$ termina in n passi se e solo se $T'(\varepsilon)$ termina in $n + 1$ passi.

Si consideri ora la seguente macchina di Turing P :

$P =$ “On input $\langle T \rangle$, where T is a Turing machine:

1. Simulate the TM R on input $\langle T \rangle$ and get $\langle T' \rangle$
2. Simulate the TM D on input $\langle T' \rangle$
3. If $D(\langle T' \rangle)$ accepts, then accept; otherwise, reject.”

Poiché D è un decisore, P termina sempre. Supponiamo che $P(\langle T \rangle)$ accetti: allora $D(\langle T' \rangle)$ ha accettato, e dunque $T'(\varepsilon)$ termina in un numero pari di passi. Quindi $T(\varepsilon)$ termina in un numero dispari di passi, e dunque $\langle T \rangle \in \text{HALT}_O$. Viceversa, supponiamo che $P(\langle T \rangle)$ rifiuti: allora $D(\langle T' \rangle)$ ha rifiutato, e dunque $T'(\varepsilon)$ o non termina, oppure termina in un numero dispari di passi. Di conseguenza $T(\varepsilon)$ o non termina oppure termina in un numero pari di passi. Perciò $\langle T \rangle \notin \text{HALT}_O$. Pertanto, il linguaggio HALT_O è decidibile.

Poiché HALT è un linguaggio costituito dall'unione di due linguaggi decidibili, è esso stesso decidibile. Ovviamente ciò è assurdo perché sappiamo che il problema della fermata è indecidibile. La contraddizione deriva unicamente dall'aver supposto che HALT_E è decidibile, e resta così dimostrato l'asserto.

Esercizio 5 [12] Si consideri il linguaggio $\mathcal{I} = \{(R_1, R_2) \mid R_1$ e R_2 sono espressioni regolari senza “*” tali che $L(R_1) \neq L(R_2)\}$. In altri termini, le istanze-sì del linguaggio sono coppie di espressioni regolari che non fanno uso dell'operatore di Kleene * e che generano linguaggi differenti. Dimostrare che il linguaggio \mathcal{I} è NP-completo.

Soluzione: Per dimostrare che $\mathcal{I} \in \text{NP}$ è necessario verificare un opportuno certificato per ogni istanza-sì in tempo polinomiale. In effetti, data una istanza (R_1, R_2) del problema, un certificato è semplicemente una stringa w tale che, in alternativa, $w \in L(R_1)$ e $w \notin L(R_2)$ oppure $w \in L(R_2)$ e $w \notin L(R_1)$. È cruciale ora considerare che le espressioni regolari R_1 e R_2 non fanno uso dell'operatore *; pertanto ciascun simbolo occorrente in ciascuna espressione

regolare può generare al massimo un singolo simbolo terminale. Quindi i linguaggi $L(R_1)$ e $L(R_2)$ hanno dimensione finita e, per ogni stringa $v \in L(R_i)$, $|v| \leq |R_i|$. Il certificato w che testimonia l'appartenza di una istanza (R_1, R_2) al linguaggio \mathcal{I} ha dunque dimensione minore od uguale a $\max(|R_1|, |R_2|)$, e quindi è di dimensione polinomiale nella dimensione dell'istanza. Un verificatore per \mathcal{I} è dunque il seguente:

$V =$ “On input (R_1, R_2, w) , where R_1, R_2 are $*$ -free REX's and w is a string:

1. Build a DFA D_1 corresponding to R_1
2. Build a DFA D_2 corresponding to R_2
3. Run D_1 on input w
4. Run D_2 on input w
5. If $D_1(w)$ accepted and $D_2(w)$ rejected, then accept
6. Otherwise if $D_2(w)$ accepted and $D_1(w)$ rejected, then accept
7. Otherwise reject.”

I passi 1 e 2 di V sono eseguibili in tempo polinomiale perché R_1 e R_2 non includono $*$: gli automi deterministici corrispondenti non hanno cicli ed hanno un numero di stati lineare nella dimensione della espressione regolare. I passi 3 e 4 sono eseguiti in tempo proporzionale alla dimensione $|w|$ del certificato; poiché possiamo assumere che il certificato ha dimensione polinomiale in $|R_1| + |R_2|$, anche questi passi sono eseguiti in tempo polinomiale. Infine i restanti passi sono eseguiti in tempo costante. Possiamo dunque concludere che $\mathcal{I} \in \text{NP}$. Si noti che la clausola che R_1 e R_2 non includono “ $*$ ” è cruciale: l'appartenenza in NP del problema di ineguaglianza di espressioni regolari generali è ancora una questione aperta.

Dimostriamo ora che \mathcal{I} è NP-hard esibendo una riduzione polinomiale dal problema SAT. Consideriamo come istanza una formula CNF $\Phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$, ove ogni C_i è la disgiunzione di letterali (variabili o negazione di variabili). Sia $\text{Var}(\Phi) = \{x_1, x_2, \dots, x_n\}$ l'insieme di tutte le variabili booleane incluse in Φ . La riduzione polinomiale trasforma Φ in una istanza di \mathcal{I} (R_1, R_2) , ove:

- L'espressione regolare R_1 è

$$R_1 = \underbrace{(0 \cup 1)(0 \cup 1) \cdots (0 \cup 1)}_{n \text{ volte}},$$

ove n è il numero di variabili in Φ . Ne consegue che $L(R_1) = \{0, 1\}^n$, ossia l'insieme di tutte le possibili assegnazioni di verità alle variabili di Φ .

- L'espressione regolare R_2 è

$$R_2 = S_1 \cup S_2 \cup \dots \cup S_m,$$

ove S_i è derivato dalla clausola C_i di Φ ($1 \leq i \leq m$) in base alle variabili occorrenti in C_i . In particolare, $S_i = (S_i^1 \ S_i^2 \ \cdots \ S_i^n)$ con

$$S_i^k = \begin{cases} \emptyset & \text{se sia } x_k \text{ che } \bar{x}_k \text{ appaiono in } C_i \\ 0 & \text{se } x_k \text{ appare in } C_i \\ 1 & \text{se } \bar{x}_k \text{ appare in } C_i \\ (0 \cup 1) & \text{se né } x_k \text{ né } \bar{x}_k \text{ appaiono in } C_i. \end{cases}$$

S_i genera dunque tutte le stringhe in $\{0, 1\}^n$ che corrispondono ad assegnazioni di verità alle n variabili che rendono *falsa* la disgiunzione di letterali della clausola C_i . Infatti, se la clausola contiene sia una variabile che la sua negazione, allora sarà sempre vera, dunque $S_i = \emptyset$ (poiché $S_i^k = \emptyset$). Se invece una variabile non appare nella clausola, il suo valore non influisce sul valore della clausola, quindi entrambe le assegnazioni 0 e 1 sono permesse per falsificare la clausola. Se infine la variabile appare una volta sola nella clausola, il corrispondente bit nella sequenza generata da S_i rende il corrispondente letterale falso.

Supponiamo che Φ sia soddisfacibile, e dunque esista una assegnazione di verità che renda vera tutte le clausole C_i di Φ . La corrispondente stringa di bit $w \in \{0, 1\}^n$ non può far parte di $L(S_i)$, perché $L(S_i)$ include tutte e sole le stringhe che rendono falsa la clausola C_i , per ogni i da 1 a m . Poiché $L(R_2) = \bigcup_{i=1}^m L(S_i)$, $w \notin L(R_2)$, e dunque $L(R_2) \neq \{0, 1\}^n = L(R_1)$. Pertanto, (R_1, R_2) è una istanza-sì di \mathcal{I} .

Al contrario, supponiamo che $(R_1, R_2) \in \mathcal{I}$, ove R_1 e R_2 derivano da una formula CNF Φ come sopra descritto. Poiché $L(R_2) \neq L(R_1) = \{0, 1\}^n$, esiste una stringa $w \in \{0, 1\}^n$ tale che $w \notin L(R_2)$. Poiché $L(R_2) = \bigcup_{i=1}^m L(S_i)$, $w \notin L(S_i)$ per ogni i da 1 a m . Pertanto l'assegnazione di verità corrispondente a w rende vere tutte le clausole C_i , per $1 \leq i \leq m$, e dunque rende vera Φ . Pertanto Φ è soddisfacibile.

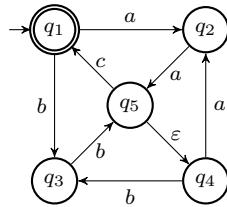
È immediato verificare che la trasformazione da Φ a (R_1, R_2) è eseguibile in tempo polinomiale in $|\Phi|$, e dunque costituisce una riduzione polinomiale tra SAT ed il linguaggio \mathcal{I} . Dunque \mathcal{I} è NP-hard.

Automi e Linguaggi (M. Cesati)

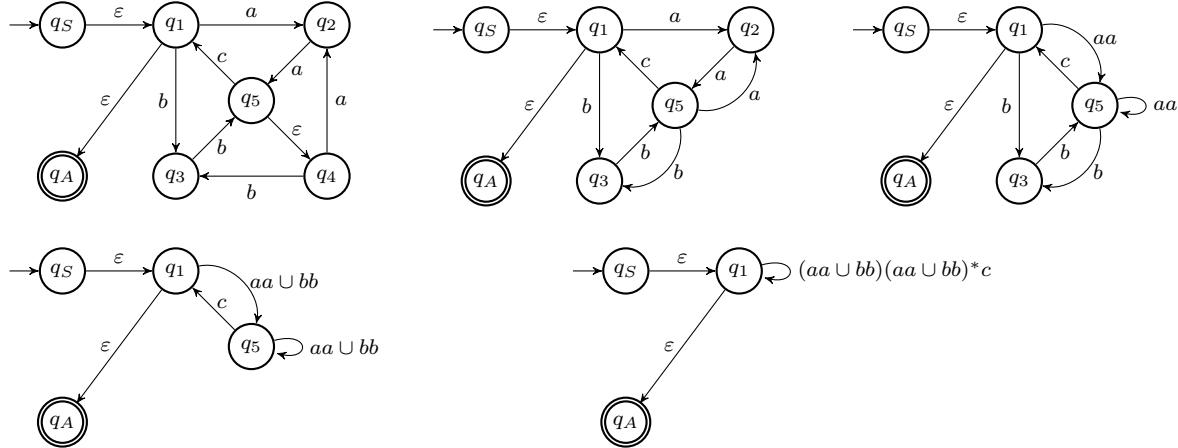
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 28 giugno 2021

Esercizio 1 [6] Derivare una espressione regolare per il linguaggio riconosciuto dall'automa:

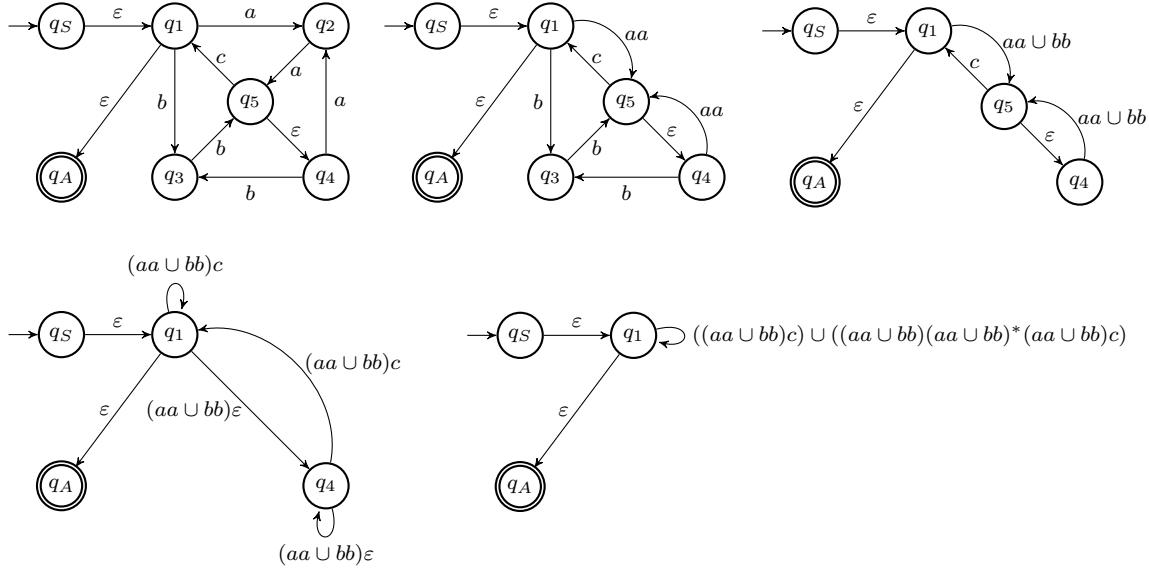


Soluzione: Convertiamo in un GNFA ed eliminiamo iterativamente i nodi dell'automa (per semplificare gli schemi non sono disegnati gli archi etichettati con ' \emptyset '):



Eliminando ora il nodo q_1 si ottiene l'espressione regolare $((aa \cup bb)(aa \cup bb)^* c)^*$, ovvero $((aa \cup bb)^+ c)^*$.

Cambiando l'ordine di eliminazione dei nodi si produce una diversa espressione regolare, comunque equivalente. Ad esempio:



Eliminando ora il nodo q_1 si ottiene l'espressione regolare

$$(((aa \cup bb) c) \cup ((aa \cup bb) (aa \cup bb)^* (aa \cup bb) c))^*$$

che può essere semplificata in $\left(((aa \cup bb) c) \cup ((aa \cup bb) (aa \cup bb)^+ c) \right)^*$ e successivamente in $((aa \cup bb)^+ c)^*$.

Esercizio 2 [6] Derivare un automa a pila (PDA) per il linguaggio $A = \{x\#y \mid x, y \in \{0, 1\}^*\}$ e x è la codifica binaria $\langle |y| \rangle$ della lunghezza di y , ovvero dimostrare che non è possibile derivare un tale PDA.

Soluzione: In effetti non è possibile costruire un PDA per il linguaggio A . È tuttavia estremamente difficile dimostrare questo fatto ragionando direttamente sul PDA, perché si dovrebbe dimostrare che *qualsiasi* algoritmo concepibile (tra gli infiniti possibili) fallirebbe nel riconoscere gli elementi di A . Possiamo invece ottenere il risultato ricordando che ogni linguaggio riconosciuto da un PDA è CFL e riconoscendo che A non è CFL. Per dimostrare l'ultimo asserto utilizziamo il “pumping lemma” per i CFL.

Supponiamo per assurdo che A sia CFL. A causa del “pumping lemma”, deve pertanto esistere una lunghezza $p > 0$ tale che, per ogni elemento s di lunghezza maggiore o uguale a p , s può essere suddiviso come $s = uvxyz$ in modo tale che $|vy| > 0$, $|vxy| \leq p$, e per ogni $i \geq 0$, $uv^i xy^i z \in A$. Consideriamo la stringa $s = 1^p \# 0^{2^p - 1}$: è immediato verificare che $|s| > p$ e $s \in A$. Dimostriamo che per ogni possibile suddivisione di s esiste un valore di $i \geq 0$ per il quale la stringa “pompata” *non* fa parte di A . In effetti una dimostrazione abbastanza semplice può

essere basata sul fatto che ogni simbolo 1 aggiunto a sinistra comporta almeno il raddoppio (in effetti la triplicazione) della lunghezza della stringa a destra; dunque il numero di simboli da aggiungere dovrebbe essere esponenziale in p , mentre la lunghezza della porzione y non può superare p . La stessa idea fondamentale viene sfruttata nella seguente dimostrazione formale in cui si utilizza esclusivamente il “pompaggio verso il basso”.

Cominciamo con l’osservare che ogni suddivisione in cui $'\#'$ $\notin x$ non può soddisfare il lemma. Infatti se $'\#'$ $\notin x$, e considerando che $|vy| > 0$, allora o vxy deve essere tutto a sinistra del simbolo $\#$ oppure deve essere tutto alla sua destra (altrimenti il simbolo $\#$ sarebbe rimosso o duplicato, e la stringa non potrebbe far parte di A in quanto malformata). In entrambi i casi pompare verso il basso produce una stringa in cui la codifica binaria a sinistra non coincide con il numero di simboli a destra.

Assumiamo dunque che $'\#'$ $\in x$, e pertanto $v \in 1^*$ e $y \in 0^*$. Analizziamo i seguenti casi basati sulla lunghezza della sottostringa v :

- $|v| = 0$: in questo caso, poiché $|vy| > 0$, deve valere $|y| > 0$. Pertanto pompando verso il basso, ossia considerando $i = 0$, si ottiene una stringa uxz in cui la codifica binaria a sinistra è identica ma il numero di elementi a destra è diminuito; tale stringa non può appartenere ad A .
- $|v| = 1$: consideriamo il valore $i = 0$, ossia la stringa $1^{p-1}\#0^l$ ottenuta pompando verso il basso, ove il valore l dipende dalla dimensione di y . Perché tale stringa possa far parte di A deve valere $l = 2^{p-1} - 1$, e quindi $|y| = 2^p - 1 - (2^{p-1} - 1) = 2^{p-1}$. Ma allora $|vxy| = 1 + 2^{p-1} + |x| \geq 2 + 2^{p-1}$; ricordando che $|vxy| \leq p$, si ottiene $2^{p-1} \leq p - 2$. In effetti non esiste alcun valore positivo per p che possa soddisfare tale disegualanza, e quindi non esiste suddivisione che possa conservare l’appartenza ad A pompando verso il basso.
- $|v| > 1$: si applica lo stesso ragionamento del caso precedente. Sia $j = |v|$, e consideriamo il valore $i = 0$. Necessariamente la lunghezza di y deve essere pari a $l = 2^p - 1 - (2^{p-j} - 1) = (1 - 2^{-j}) 2^p$. Pertanto: $p \geq |vxy| \geq j + 1 + (1 - 2^{-j}) 2^p$, e poiché $(1 - 2^{-j}) > 1/2$ per $j > 1$:

$$2^{p-1} < 2^p \left(1 - 2^{-j}\right) \leq p - j - 1 < p - 1.$$

Poiché questa disegualanza non ha alcuna soluzione per $p > 0$, non esiste alcuna suddivisione che possa conservare l’appartenza ad A pompando verso il basso.

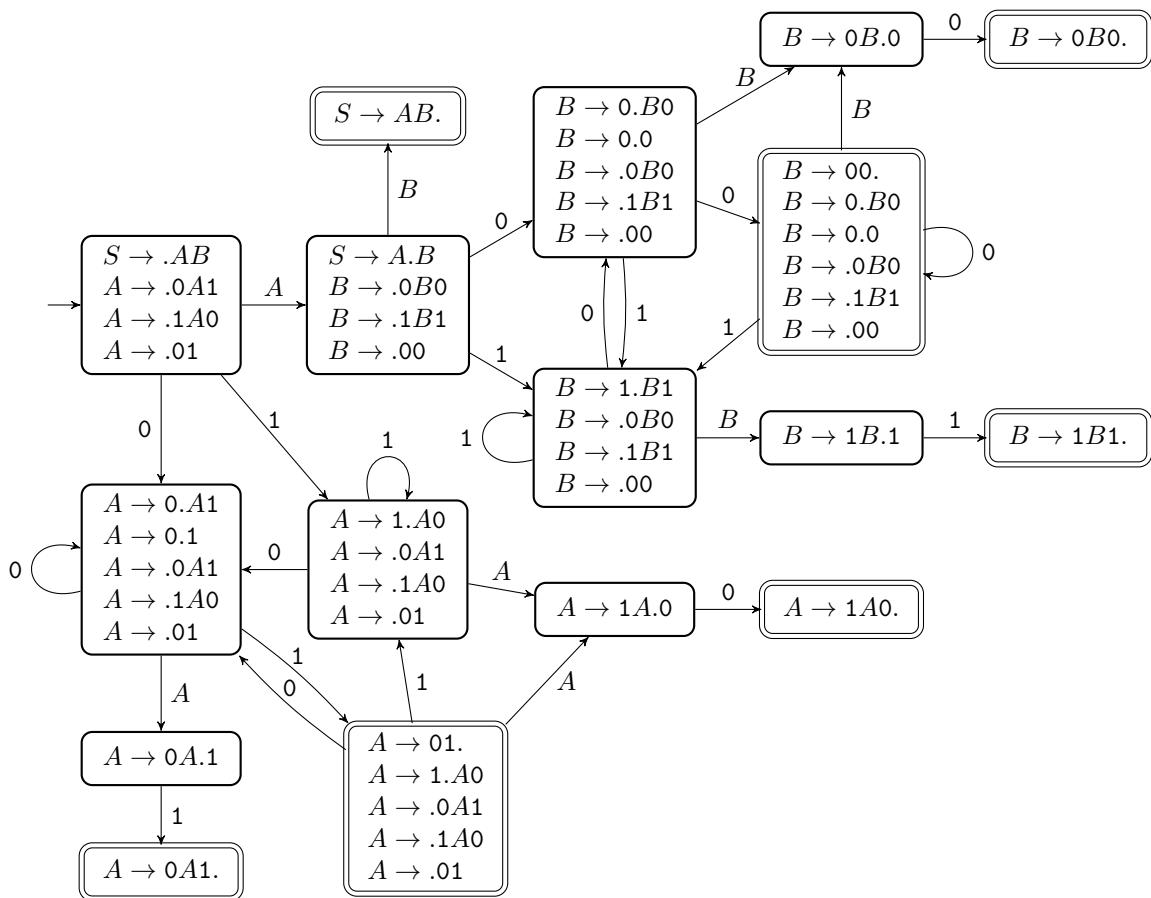
Poiché ogni possibile suddivisione di s risulta non appartente ad A quando pompata verso il basso, le conclusioni del “pumping lemma” non valgono. La contraddizione è dovuta alla

supposizione che A sia CFL. Concludiamo pertanto che non esiste alcun PDA che possa riconoscere gli elementi del linguaggio A .

Esercizio 3 [6] Determinare se la seguente grammatica libera dal contesto con variabile iniziale S è deterministica:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow 0A1 \quad | \quad 1A0 \quad | \quad 01 \\ B &\rightarrow 0B0 \quad | \quad 1B1 \quad | \quad 00 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica applichiamo il DK-test. Si ottiene il seguente automa:



Poiché esistono diversi stati finali che includono regole con il punto seguito da un carattere terminale, possiamo immediatamente concludere che la grammatica non è deterministica.

Esercizio 4 [6] Dimostrare che un linguaggio L è ricorsivamente enumerabile (ossia Turing-riconoscibile) se e solo se L riduce tramite funzione al problema di accettazione delle macchine di Turing (ossia, $L \leq_m \mathcal{A}_{\text{TM}}$).

Soluzione: Sappiamo che \mathcal{A}_{TM} è ricorsivamente enumerabile. Sappiamo inoltre che se un linguaggio X riduce tramite funzione ad un linguaggio ricorsivamente enumerabile Y , allora X è ricorsivamente enumerabile. Infatti, detta T la macchina di Turing che costituisce la riduzione e N la macchina di Turing che riconosce Y , è immediato derivare una macchina di Turing M che riconosce X : su ogni input w , $M(w)$ esegue $T(w)$, poi esegue N sull'output di $T(w)$. Ovviamente $w \in X$ se e solo se l'istanza prodotta da $T(w)$ è in Y ; dunque se $w \in X$ allora $M(w)$ accetta, mentre se $M(w)$ non accetta allora $w \notin X$. Perciò se $L \leq_m \mathcal{A}_{\text{TM}}$ allora L è ricorsivamente enumerabile.

Viceversa, supponiamo che L sia un linguaggio ricorsivamente enumerabile, dunque esista una macchina di Turing T che accetta un input w se e solo se $w \in L$ (e potrebbe non terminare altrimenti). Consideriamo la funzione calcolabile che mappa ogni istanza w nella istanza di \mathcal{A}_{TM} $\langle T, w \rangle$: poiché $w \in L$ se e solo se $\langle T, w \rangle \in \mathcal{A}_{\text{TM}}$, tale funzione costituisce una riduzione tramite funzione da L a \mathcal{A}_{TM} , ossia $L \leq_m \mathcal{A}_{\text{TM}}$.

Esercizio 5 [6] Dimostrare che un linguaggio L è decidibile se e solo se L riduce tramite funzione al linguaggio $B = \{x \in \{0, 1\}^* \mid x \text{ è la codifica in binario di un numero primo}\}$.

Soluzione: Il linguaggio B è decidibile; infatti consideriamo una macchina di Turing M che su input un numero codificato in binario x genera ogni numero intero compreso tra 2 e $x - 1$, ed accetta se e solo se nessuno di essi divide esattamente x . M è quindi un decisore per B .

Sappiamo che se $X \leq_m Y$ e Y è decidibile, allora X è decidibile. Infatti, detta T la macchina di Turing che costituisce la riduzione e N la macchina di Turing che decide Y , è immediato derivare una macchina di Turing M che decide X : su ogni input w , $M(w)$ esegue $T(w)$, poi esegue N sull'output di $T(w)$. Ovviamente $w \in X$ se e solo se l'istanza prodotta da $T(w)$ è in Y ; dunque se $w \in X$ allora $M(w)$ accetta, mentre se $w \notin X$ allora $M(w)$ rifiuta. Perciò se $L \leq_m B$ allora L è decidibile.

Viceversa, supponiamo che L sia decidibile, dunque esista una macchina di Turing M che decide L . Consideriamo la macchina di Turing T che su input w dapprima esegue $M(w)$; se $M(w)$ accetta allora $T(w)$ produce in output la codifica in binario del numero primo 3; se invece $M(w)$ rifiuta allora $T(w)$ produce in output la codifica in binario del numero composto 4. Perciò $T(w)$ produce un elemento in B se e solo se $w \in L$; pertanto $L \leq_m B$.

Esercizio 6 [10] Dato un grafo diretto, un *kernel* è un sottoinsieme di nodi K tale che (1) non esistono archi tra i nodi di K , e (2) per ogni nodo $u \notin K$, esiste un arco da u ad un nodo di K . Dimostrare che il problema $\text{KERNEL} = \{\langle G \rangle \mid \text{il grafo diretto } G \text{ contiene un kernel}\}$ è NP-completo.

Soluzione: Il problema KERNEL appartiene alla classe NP in quanto ogni istanza $\langle G \rangle$ che fa parte del linguaggio ammette come certificato il sottoinsieme di nodi K del grafo che costituiscono il kernel: è certamente di dimensione non superiore al numero di nodi di G , e dunque alla dimensione dell'istanza, e verificare che il sottoinsieme è effettivamente un kernel può essere facilmente realizzato da un algoritmo che esegue in tempo polinomiale nella dimensione dell'istanza del problema:

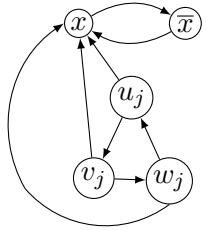
$M =$ “On input $\langle G, C \rangle$, where G is a directed graph:

1. If C is not a list of nodes $K \subseteq V(G)$, then reject
2. for each pair of elements $u, w \in K$:
3. if $(u, w) \in E(G)$ then reject
4. for each element $x \in V(G) \setminus K$:
5. for each element $y \in K$:
6. if $(x, y) \in E(G)$ then continue with next element in step 3
7. Reject, because element x is not adjacent to any node in K
8. Accept, because all elements in $V(G) \setminus K$ are adjacent to nodes in K ”

È immediato stabilire che M termina in un numero di passi in $O(n^4)$ nel numero di nodi n del grafo G , e quindi è un verificatore polinomiale.

Per dimostrare che KERNEL è NP-hard utilizziamo una riduzione polinomiale da un problema NP-hard. Poiché KERNEL è un problema riguardante grafi, potrebbe sembrare opportuno ridurre da un problema su grafi come VERTEX COVER oppure INDEPENDENT SET. Tuttavia questi problemi hanno nella istanza un parametro numerico (la dimensione del sottoinsieme di nodi), che manca completamente in KERNEL. Poiché la costruzione deve tenere in considerazione tale parametro numerico in modo sostanziale, la riduzione polinomiale non sembra essere immediata. Piuttosto, cerchiamo di ridurre da un problema senza parametro numerico nelle istanze, ad esempio il problema 3SAT. Sia dunque $\Phi = \bigwedge_j C_j$ una formula booleana in CNF in cui ogni clausola C_j è la disgiunzione di esattamente tre letterali. Sia $\text{Var}(\Phi)$ l'insieme delle variabili booleane utilizzate in Φ .

Il grafo diretto G_Φ risultante dalla riduzione polinomiale da Φ contiene per ogni $x \in \text{Var}(\Phi)$ due nodi (\underline{x}) e (\bar{x}) connessi da due archi diretti. Il grafo inoltre contiene per ogni clausola C_j in Φ tre nodi (u_j) , (v_j) e (w_j) connessi da tre archi diretti. Infine, per ogni letterale l incluso in una clausola C_j , G_Φ include tre archi dai tre nodi della clausola al nodo del letterale.



Supponiamo che la formula Φ sia soddisfacibile; dunque esiste una assegnazione di verità alle variabili in $\text{Var}(\Phi)$ che rende vera ogni clausola. Dimostriamo allora che il grafo G_Φ contiene un kernel K . Per ogni variabile x a cui è stato assegnato il valore *vero*, includiamo in K il nodo (\underline{x}) ; analogamente, per ogni variabile y con valore *falso*, includiamo in K il nodo (\bar{y}) . In totale quindi K contiene un nodo per ciascuna variabile della formula. Osserviamo che, per costruzione, non esiste alcun arco tra gli elementi in K . Infatti non esistono archi tra nodi di letterali corrispondenti a variabili differenti, ed inoltre una assegnazione di verità non può assegnare ad una variabile contemporaneamente il valore *vero* e quello *falso*. Consideriamo ora un qualsiasi nodo (z) di G_Φ non incluso in K , e distinguiamo due casi. Se il nodo (z) corrisponde ad un letterale, ossia ad una variabile diretta o negata, allora il nodo (\bar{z}) associato alla negazione del letterale deve necessariamente essere incluso in K perché l'assegnazione di verità include tutte le variabili di Φ . Pertanto, per costruzione esiste un arco da (z) (non in K) a (\bar{z}) (in K). Se invece il nodo (z) corrisponde al nodo di una clausola, allora tale clausola deve essere soddisfatta dalla assegnazione di verità; dunque deve esistere in K un nodo (l) corrispondente ad un letterale che rende vera la clausola. Per costruzione esiste un arco tra (z) (non in K) e (l) (in K). Si può concludere che K è effettivamente un kernel di G_Φ .

Supponiamo ora che nel grafo G_Φ esista un kernel K , e dimostriamo allora che la formula Φ sarebbe soddisfacibile. Supponiamo che K contenga un nodo (u_j) corrispondente ad una clausola C_j ; senza perdita di generalità supponiamo che da (u_j) esca un arco verso il nodo della stessa clausola (v_j) ed entri un arco proveniente dal nodo della stessa clausola (w_j) . Osserviamo che K non può includere né (v_j) né (w_j) poiché non possono esistere archi tra elementi del kernel. Pertanto, deve esistere un altro nodo (l) in K , necessariamente associato ad un letterale, che è la destinazione di un arco uscente da (v_j) . Per costruzione, il nodo (l) è anche la destinazione di archi uscenti da (u_j) e (w_j) . Dunque, il sottoinsieme di nodi ottenuto rimuovendo (u_j) da K è ancora un kernel per G_Φ . Possiamo dunque assumere che K contenga esclusivamente nodi associati a variabili (dirette o negate) di Φ . Consideriamo la assegnazione di verità (parziale) corrispondente ai nodi inclusi in K : se K include il nodo (\underline{x}) assegniamo ad x il valore *vero*; se invece include il nodo (\bar{x}) assegniamo ad x il valore *falso*. Poiché non possono esistere archi tra i nodi del kernel, non è possibile che ad una stessa variabile debba essere assegnato sia il valore *vero* che *falso*. Consideriamo ora una qualunque clausola C_j di Φ ed un generico nodo associato alla clausola (u_j) . Sappiamo che deve esistere un arco da (u_j) ad

un nodo (l) in K ; tale arco è stato inserito in G_Φ perché il letterale l è incluso nella clausola C_j . L'assegnazione di verità ha assegnato al letterale l il valore *vero*, quindi la clausola C_j è soddisfatta dalla assegnazione di verità. Tutte le clausole sono pertanto soddisfatte dalla assegnazione di verità corrispondente a K , e quindi la formula Φ è soddisfacibile. Si osservi che non è necessario che K includa un nodo per ciascuna variabile di Φ : è possibile che la formula possa essere soddisfatta a prescindere dal valore di alcune variabili, e K potrebbe non includere i nodi corrispondenti a variabili il cui valore non è determinante per soddisfare Φ .

È immediato verificare che la costruzione di G_Φ da Φ può essere completata in tempo polinomiale. Pertanto KERNEL è NP-hard, e dunque NP-completo, come volevasi dimostrare.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 14 luglio 2021

Esercizio 1 [5] Sia $A = \{xx^R \mid x \in \{0,1\}^*\}$; A è un linguaggio regolare? Giustificare la risposta con una dimostrazione.

Soluzione: È molto semplice dimostrare che A non è un linguaggio regolare per mezzo del pumping lemma. Infatti, supponiamo per assurdo che A sia regolare. Pertanto dovrebbe valere per esso il pumping lemma, quindi esisterebbe una lunghezza $p > 0$ tale che ogni stringa $s \in A$ con $|s| \geq p$ potrebbe essere “pompata” verso l’alto o verso il basso conservando l’appartenenza in A . Consideriamo la stringa $s = 0^p 1 1 0^p$, certamente appartenente ad A ($(10^p)^R = 0^p 1$) e di lunghezza $> p$. Per essa deve valere il pumping lemma, quindi deve esistere una suddivisione $s = xyz$ tale che (1) $xy^i z \in A$ per ogni $i \geq 0$, (2) $|y| > 0$ e (3) $|xy| \leq p$. Dalle ultime due disuguaglianze possiamo dedurre che $y \in 0^+$, ossia y è costituito da uno o più “0”. Pertanto per ogni $i \neq 1$ la stringa pompata $xy^i z$ avrebbe la forma $0^q 1 1 0^p$ con $q \neq p$. Dunque la stringa pompata non potrebbe evidentemente far parte di A (la sottostringa finale da rovesciare deve comunque iniziare con il carattere “1” altrimenti non potrebbe essere identica alla sottostringa iniziale; però ora non è possibile far coincidere le parti delle sottostringhe con i caratteri “0” perché hanno lunghezza differente). La contraddizione deriva dall’aver supposto che A sia regolare; è perciò dimostrato che A non è un linguaggio regolare.

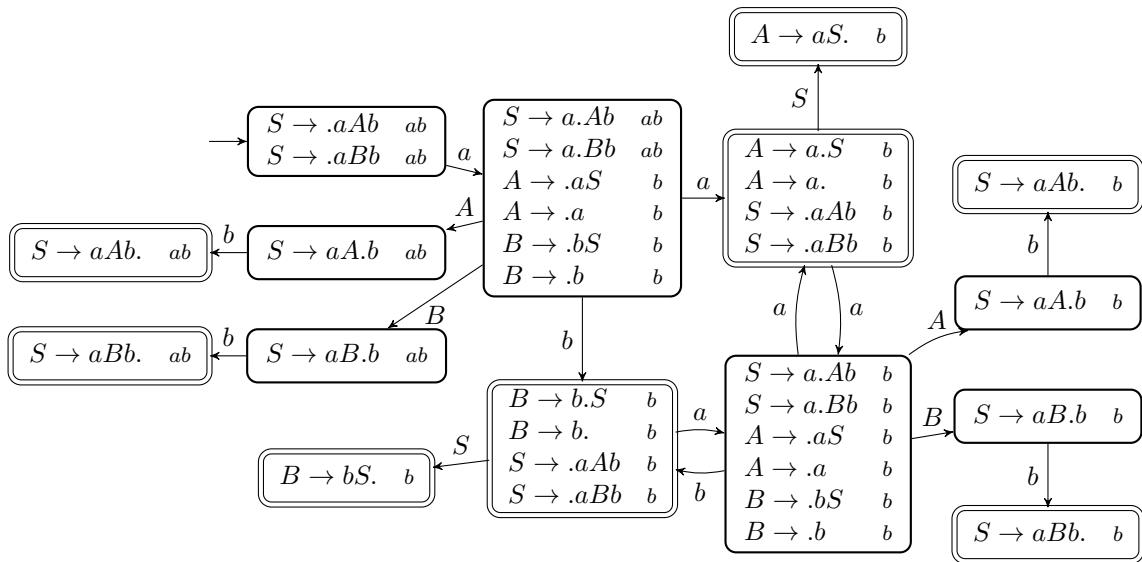
Esercizio 2 [5] Sia $B = \{xyx^R \mid x, y \in \{0,1\}^*\}$; B è un linguaggio regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Malgrado le apparenze, dovute alla forma in cui è stato definito il linguaggio B , esso è regolare. Affinché una stringa appartenga a B , essa deve poter essere formata da tre parti: una testa x , una parte centrale y , ed una coda che è il rovescio x^R della testa; in effetti però x e y possono essere qualunque elemento di $\{0,1\}^*$. Possiamo quindi considerare $x = \varepsilon \in \{0,1\}^*$: poiché $\varepsilon^R = \varepsilon$ (il rovescio della stringa vuota è ancora una stringa vuota), B contiene come sottoinsieme $\{\varepsilon y \varepsilon \mid y \in \{0,1\}^*\}$, ossia contiene $\{0,1\}^*$. Quindi $\{0,1\}^* \subseteq B \subseteq \{0,1\}^*$, ossia $B = \{0,1\}^*$. Naturalmente B è regolare, ad esempio perché generato dalla espressione regolare $(0 \cup 1)^*$.

Esercizio 3 [6] Determinare se la seguente grammatica con variabile iniziale S è LR(1), LR(0), oppure né LR(1) né LR(0):

$$S \rightarrow aAb \mid aBb \quad A \rightarrow aS \mid a \quad B \rightarrow bS \mid b$$

Soluzione: La grammatica è LR(1) ma non LR(0) (ossia non è DCFG). Per dimostrare ciò costruiamo l'automa per il DK₁-test:



È immediato verificare che nessuno stato accettante contiene regole tra loro consistenti, perciò la grammatica è LR(1). D'altra parte lo stesso automa, trascurando i simboli di lookahead, dimostra che il DK-test fallisce, in quanto esistono stati accettanti contenenti regole in cui il punto è seguito da un simbolo terminale. Perciò la grammatica non è LR(0), ossia non è DCFG.

Esercizio 4 [7] Derivare un automa a pila (PDA) per il linguaggio

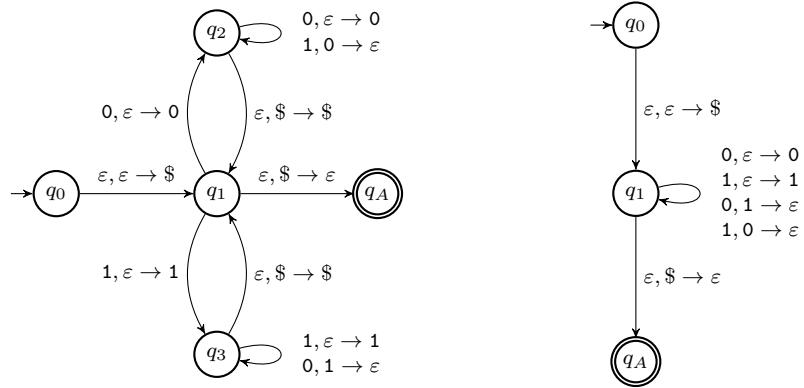
$$C = \{x \in \{0, 1\}^* \mid x \text{ ha lo stesso numero di } 0 \text{ e } 1\},$$

ovvero dimostrare che non è possibile derivare un tale PDA. Ad esempio, $\varepsilon \in C$, $011001 \in C$, $011 \notin C$.

Soluzione: Come di consueto nei casi in cui l'appartenenza al linguaggio è decisa “contando” il numero di occorrenze di due diversi tipi di simboli in ordine sparso, un PDA che riconosce gli elementi di C può essere basato sulla memorizzazione dei simboli “in eccedenza”. Se ad

un certo punto, leggendo i simboli della stringa in ingresso, si ha una eccedenza di “0”, allora lo stack conterrà un numero di simboli “0” uguale alla differenza tra il numero di “0” ed il numero di “1” letti fino a quel momento; analogamente nel caso vi sia eccedenza di simboli “1”.

Questa idea di base può essere implementata in molti modi diversi. A titolo di esempio, consideriamo questi PDA:



Dimostriamo che questi automi riconoscono il linguaggio C con il seguente ragionamento. Ogni regola che consuma un simbolo dell’input è una “push” dello stesso simbolo sullo stack, oppure una “pop” del simbolo alternativo dallo stack. Le regole che non consumano simboli della stringa di input sono esclusivamente legate al marcatore di fine stack “\$”. Un’altra considerazione comune a tutti gli automi è che essi accettano esclusivamente se sulla cima dello stack è presente “\$”, ossia se lo stack è “vuoto”.

Assumiamo dunque che un automa accetti una certa stringa x di lunghezza $n \geq 0$: tutti i caratteri devono essere stati letti, e lo stack non deve contenere altro che “\$”; dunque il numero di “push” e ”pop” dei caratteri diversi da “\$” deve essere stato identico e uguale in totale al numero n di simboli della stringa in input. Dunque sono state effettuate $n/2$ “pop”, ciascuna delle quali “accoppia” il simbolo appena letto con un simbolo letto precedentemente da una regola di tipo “push”; ma tali simboli sono alternativi. Perciò il numero di “0” letti con le regole “pop” è uguale al numero di “1” letti con le regole “push”, ed il numero di “0” letti con le regole “push” è uguale al numero di “1” letti con le regole “pop”. Di conseguenza, il numero di “0” letti in totale è uguale al numero di “1” letti in totale, e quindi $x \in C$.

Viceversa, se $x \in C$ allora il numero di “0” e di “1” deve coincidere; dimostriamo per induzione sulla lunghezza di x che ciascun automa accetta. L’asserto è immediato se $|x| = 0$, ossia $x = \varepsilon$. Supponiamo dunque che l’automa accetti qualunque stringa x di lunghezza fino a $n - 2$ con ugual numero di “0” e “1”, e consideriamo una stringa di lunghezza $n \geq 2$. Senza perdita di generalità assumiamo che il primo simbolo di x sia “0”. Poiché lo stack è “vuoto”, la regola

che consuma tale simbolo deve essere di tipo “push”. La prima occorrenza del simbolo “1” in x occorre avendo sulla cima dello stack un simbolo “0”. Consideriamo la stringa x' ottenuta rimuovendo questo simbolo “1” ed il simbolo “0” immediatamente precedente da x : x' è una stringa di lunghezza $n - 2$ con ugual numero di “0” e “1”, e pertanto per ipotesi induttiva esiste un ramo di computazione che porta l'automa ad accettare. D'altra parte, la stringa x differisce da x' unicamente per una sottosequenza “01”, e sappiamo che l'automa deve aver usato una regola di tipo “push” leggendo lo “0”, e dunque sulla cima dello stack si trova “0” nel momento in cui si legge “1”. L'automa può quindi applicare la regola di tipo “pop” per leggere il simbolo “1”: necessariamente questa regola è un “self-loop” che non cambia lo stato interno dell'automa, e lascia lo stato dello stack esattamente nella condizione precedente alla lettura dello “0” della sottosequenza “01”. Pertanto, a partire dal simbolo successivo della sottosequenza, è possibile applicare esattamente le stesse regole che hanno portato l'automa ad accettare x' , ed alla fine quindi l'automa giunge ad accettare x .

Esercizio 5 [7] Si consideri il linguaggio

$$D = \{\langle M \rangle \mid M \text{ è una TM che si ferma su input } \varepsilon \text{ e } |\langle M \rangle| \leq 10^9\};$$

in altri termini, D contiene le codifiche delle TM di lunghezza inferiore ad un miliardo di caratteri che si fermano quando eseguite con il nastro inizialmente vuoto. Il linguaggio D è decidibile? Giustificare la risposta.

Soluzione: Iniziamo con l'osservare che cercare di dimostrare che D è indecidibile esibendo una qualunque riduzione da un problema indecidibile, ad esempio il problema della fermata delle TM, non può funzionare. Infatti il problema da cui si riduce è costituito da infinite istanze di dimensioni arbitrariamente grandi. La riduzione deve deterministicamente costruire una istanza di D che, per essere accettata, deve avere lunghezza limitata. Non appare esserci modo di costruire tale riduzione senza decidere, all'interno della TM che costruisce la riduzione, l'istanza originale; ma sappiamo che poiché il problema originale è indecidibile, fare questo è impossibile.

In effetti il linguaggio D è decidibile, e la dimostrazione è molto semplice: è un linguaggio contenente un numero finito di elementi, e dunque è un linguaggio regolare. Infatti il numero di TM M la cui codifica $\langle M \rangle$ è una stringa di lunghezza non superiore a N è certamente inferiore al numero totale di stringhe di lunghezza non superiore a N . D'altra parte tale numero dipende dalla cardinalità S dell'alfabeto su cui insistono le stringhe, quindi è inferiore a S^{N+1} . Possiamo comunque assumere che $S \leq N$, altrimenti vi sarebbero caratteri inutilizzati dell'alfabeto. Pertanto esistono meno di N^{N+1} macchine di Turing la cui codifica è lunga al più N simboli. Ovviamente per $N = 10^9$ tale numero è gigantesco, ma il punto è che esso

è finito e dunque il linguaggio D è decidibile: deve esistere una TM (od anche un DFA) che semplicemente confronta la stringa in input con ciascuna delle codifiche degli elementi di D ; in tempo finito quindi accetta o rifiuta.

È opportuno però sottolineare che per poter costruire il decisore di D dovremmo stabilire una volta per tutte, per ciascuna delle TM la cui codifica è lunga meno di $N = 10^9$, se tale macchina si ferma o meno partendo dal nastro vuoto. Questo particolare problema è indecidibile, quindi non esiste procedura che possa portare alla costruzione della macchina che testimonia la decidibilità di D , anche se sappiamo che tale macchina deve esistere.

Esercizio 6 [10] Il problema HITTING SET è costituito da un insieme di elementi U , da una collezione \mathcal{C} di sottoinsiemi di U , e da un numero intero k tali che esiste un sottoinsieme $S \subseteq U$ (non necessariamente in \mathcal{C}) con $|S| \leq k$ e S contenente almeno un elemento di ciascun sottoinsieme in \mathcal{C} . Dimostrare che il problema è NP-completo.

Soluzione: Per dimostrare che il problema HITTING SET (HS) può essere verificato in tempo polinomiale, consideriamo la seguente TM:

M= “On input $\langle U, \mathcal{C}, k, C \rangle$, where U is a set of elements, $\mathcal{C} \subseteq 2^U$, $k \in \mathbb{N}$:

1. Verify that C is a list of elements of U ; if not, reject.
2. Verify that C includes k or less elements; if not, reject.
3. For each element x in C :
 4. Scan the subsets in \mathcal{C} and mark every subset containing x .
 5. Scan the subsets in \mathcal{C} : if one of them is not marked, reject.
 6. Accept (all subsets in \mathcal{C} are marked)”

I primi due passi possono essere conclusi in tempo $O(k)$, e quindi in tempo $O(n)$. Il ciclo del passo 3 esegue $O(k) = O(n)$ iterazioni; in ciascuna di esse, vengono scanditi tutti gli elementi in \mathcal{C} (in numero certamente inferiore alla dimensione dell’istanza, poiché \mathcal{C} fa parte dell’istanza). Anche il passo 5 è eseguito in tempo lineare nella dimensione dell’istanza, mentre l’ultimo passo impiega tempo costante. Pertanto M esegue in tempo polinomiale nella dimensione dell’istanza, e quindi possiamo concludere che $HS \in NP$.

Per verificare che HS è NP-hard, è sufficiente considerarlo come una semplice generalizzazione del problema VERTEX COVER. Un grafo non diretto infatti può essere considerato come un insieme di elementi (i nodi) ed una collezione di sottoinsiemi di nodi con esattamente due elementi in ogni sottoinsieme (gli archi). Formalmente, per ogni istanza di VC, ossia un grafo $G = (V, E)$ ed un intero $k \in \mathbb{N}$, consideriamo l’istanza (U, \mathcal{C}, k') di HS così fatta: $U = V$,

$\mathcal{C} = \{\{u, v\} \mid (u, v) \in E\}$ e $k' = k$. Il grafo G ammette un ricoprimento tramite vertici di dimensione $\leq k$ se e solo se questi k nodi ricoprono tutti gli archi, ovvero se e solo se lo stesso sottoinsieme di elementi di U ha un elemento in ciascun sottoinsieme di \mathcal{C} , ossia se e solo se l'istanza (U, \mathcal{C}, k) ammette un “hitting set” di dimensione al più k . Pertanto, VC riduce in tempo polinomiale a HS, e dunque HS è NP-hard.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 1 settembre 2021

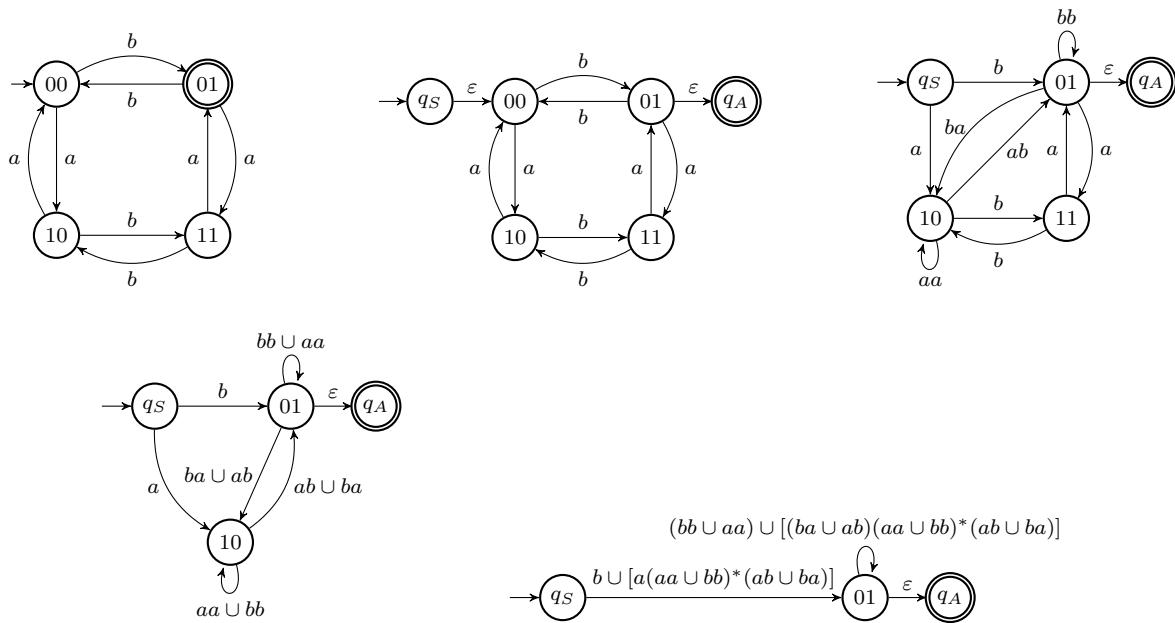
Esercizio 1 [6] Determinare una espressione regolare per il linguaggio

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

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

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

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 $\delta'((q, \bar{q}, f), \sigma) = (q', \bar{q}, t)$, per ogni $\bar{q} \in Q_B$
 - per ogni transizione $\delta_B(q, \sigma) = q'$, δ'' include la transizione $\delta''((\bar{q}, q, t), \sigma) = (\bar{q}, q', f)$, per ogni $\bar{q} \in Q_A$

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 $q_0^A, \dots, q_F^A \in F_A$, e analogamente $M_B(y)$ accetta con una successione di stati $q_0^B, \dots, q_F^B \in F_B$. Consideriamo ora la computazione di M sulla stringa w . Innanzitutto, la variabile booleana codificata nello stato interno di M garantisce che leggendo la stringa w venga applicata innanzitutto 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 $(q_0^A, q_0^B, f), (q_1^A, q_0^B, t), (q_1^A, q_1^B, f), \dots, (q_{n-2}^A, q_{n-2}^B, t), (q_F^A, q_F^B, f)$. Ora è immediato verificare che la sequenza di stati $q_0^A, q_1^A, \dots, q_{n-2}^A, q_F^A$ 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 $q_0^B, q_1^B, \dots, q_{n-2}^B, q_F^B$ 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

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

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

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 lun-

ghezza 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 $uv^2 xy^2 z$ 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 $B = \mathcal{A}_{\text{TM}}$. 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}_{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 $L_1, L_2 \in \text{NL}$ allora $L_1 \cup L_2 \in \text{NL}$ e $L_1 \circ L_2 = \{xy \mid x \in L_1, y \in L_2\} \in \text{NL}$.

Soluzione: Poiché $L_1, L_2 \in \text{NL}$, 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 $L_1 \cup L_2 \in \text{NL}$ 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, $L_1 \cup L_2 \in \text{NL}$.

Per dimostrare che $L_1 \circ L_2 \in \text{NL}$ 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, $L_1 \circ L_2 \in \text{NL}$.

Automi e Linguaggi (M. Cesati)

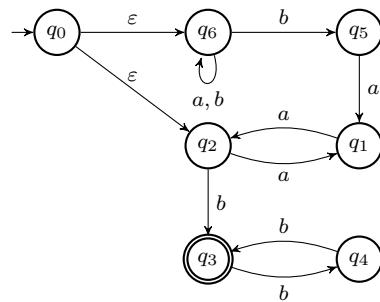
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 15 settembre 2021

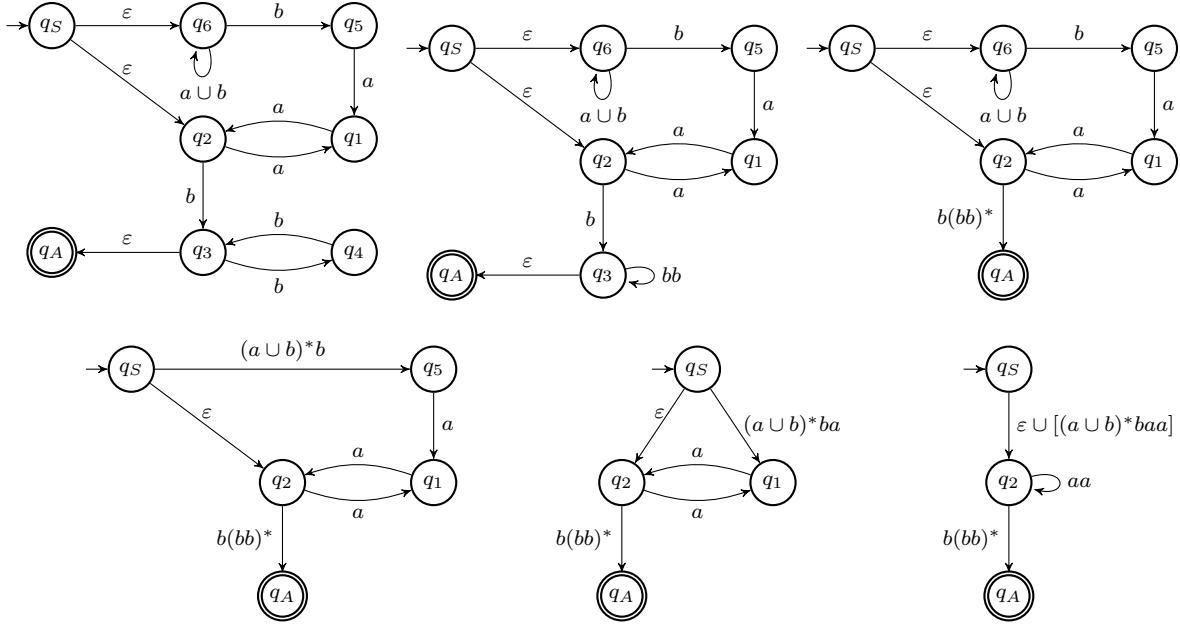
Esercizio 1 [6] Determinare una espressione regolare per il linguaggio $C = \{x \in \{a, b\}^* \mid$ l'ultima sequenza di a consecutive in x ha lunghezza pari, l'ultima sequenza di b consecutive in x ha lunghezza dispari, e l'ultimo carattere di x è $b\}$, ovvero dimostrare che tale espressione regolare non esiste. Ad esempio, $\varepsilon \notin C$, $b \in C$, $baa \notin C$ (l'ultimo carattere non è b), $abaabbb \in C$, $baaabab \notin C$.

Soluzione: Il linguaggio C è regolare, perché per riconoscere l'appartenza di una stringa in C non è necessario contare le occorrenze di simboli a e b , ma solo memorizzare la parità o disparità del numero di occorrenze nelle ultime sequenze.

Una strategia per determinare una espressione regolare per il linguaggio richiesto consiste nel determinare dapprima un NFA che riconosca il linguaggio, e successivamente derivare dal NFA una REX. Il linguaggio C può essere riconosciuto dal seguente NFA, in cui gli stati q_1 , q_2 , q_3 e q_4 servono a memorizzare la parità delle ultime sequenze di a e di b , mentre gli stati q_5 e q_6 sono utilizzati per “indovinare” la posizione dell’ultima sequenza di a nella stringa in ingresso. Dallo stato iniziale q_0 si può non deterministicamente saltare direttamente allo stato q_2 per considerare il caso in cui nella stringa in ingresso sia presente una unica sequenza di a (eventualmente di lunghezza zero) ed una unica sequenza finale di b .



Trasformando in GNFA e rimuovendo nell’ordine i nodi q_4 , q_3 , q_6 , q_5 , q_1 e q_2 si ottiene:



ed infine

$$\{\varepsilon \cup [(a \cup b)^*baa]\} (aa)^*b(bb)^*.$$

Esercizio 2 [5] Si consideri la grammatica G con variabile iniziale S

$$S \rightarrow PABb \quad P \rightarrow aP \mid bP \mid \varepsilon \quad A \rightarrow aaA \mid \varepsilon \quad B \rightarrow bbB \mid \varepsilon.$$

Il linguaggio generato dalla grammatica coincide con il linguaggio C dell'esercizio precedente? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio generato dalla grammatica è differente dal linguaggio C dell'esercizio precedente. Per dimostrarlo è sufficiente considerare la stringa ab : essa non appartiene a C in quanto l'ultima sequenza di a ha lunghezza dispari; d'altra parte $ab \in L(G)$, come dimostrato dalla seguente derivazione:

$$S \Rightarrow PABb \Rightarrow aPABb \Rightarrow aAABb \Rightarrow aBb \Rightarrow ab.$$

Esercizio 3 [6] Si consideri la grammatica G con variabile iniziale S

$$S \rightarrow PABb \quad P \rightarrow aP \mid bP \mid \varepsilon \quad A \rightarrow aaA \mid \varepsilon \quad B \rightarrow bbB \mid \varepsilon.$$

La grammatica G è LR(1)? Giustificare la risposta con una dimostrazione.

Soluzione: La grammatica G non è LR(1). Per dimostrarlo è sufficiente costruire lo stato iniziale dell'automa DK_1 :

| | |
|-----------------------|------|
| $S \rightarrow .PABb$ | ab |
| $P \rightarrow .aP$ | ab |
| $P \rightarrow .bP$ | ab |
| $P \rightarrow .$ | ab |

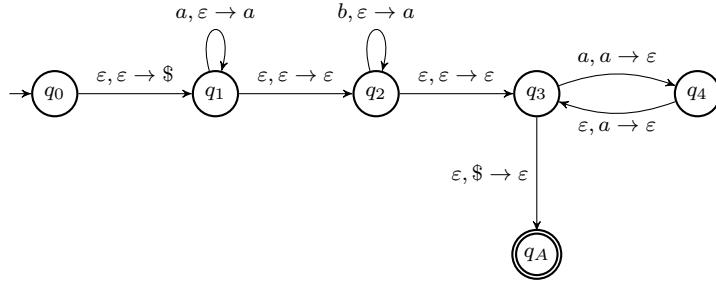
I simboli di lookahead delle regole “ P ” (seconda, terza e quarta nello stato dell'automa) sono a e b in quanto dalla sequenza ABb che segue P nella prima regola può essere generato come primo simbolo sia a (applicando $A \rightarrow aaA$) che b (applicando $A \rightarrow \varepsilon$). A causa della quarta regola, lo stato è accettante. D'altra parte, nella seconda regola a segue il punto ed è anche simbolo di lookahead della regola completata (stesso discorso per la terza regola). Pertanto, il DK_1 -test è fallito, e quindi la grammatica non è LR(1).

Esercizio 4 [6.5] Si consideri il linguaggio $D = \{a^h b^k a^l \mid k = h \wedge l = 2h\}$. D è un linguaggio libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio D non è CFL, come si può dimostrare applicando il “pumping lemma” per CFL. Supponiamo per assurdo che D sia CFL; varrebbe dunque per esso il lemma, e quindi dovrebbe esistere una lunghezza $p > 0$ tale che ogni stringa in D di lunghezza non inferiore a p può essere suddivisa in modo da poter essere “pompata” verso l'alto o verso il basso. Consideriamo dunque la stringa $s = a^p b^p a^{2p} \in D$, ed una sua suddivisione $s = uvxyz$ tale che $uv^i xy^i z \in D$ per ogni $i \geq 0$, $|vy| > 0$ e $|vxy| \leq p$. Affinché la stringa pompata appartenga ancora a D , vxy deve includere almeno una a a sinistra, tutte le b ed almeno una a a destra: infatti se pompando si modifica la lunghezza di una qualunque delle tre sequenze a^p , b^p e a^{2p} , allora anche le altre due sequenze devono essere opportunamente modificate affinché la stringa pompata appartenga a D . Perciò $|vxy| > |ab^p a| = p + 2$, il che è in contraddizione con la condizione $|vxy| \leq p$. La contraddizione è dovuta all'aver supposto che D è CFL. Resta dunque dimostrato che D non è CFL.

Esercizio 5 [6.5] Si consideri il linguaggio $E = \{a^h b^k a^l \mid h + k = 2l\}$. E è un linguaggio libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio E è CFL. Determinare una grammatica libera dal contesto che genera il linguaggio E non è facile; è molto più semplice invece esibire un PDA che riconosce il linguaggio E :



L'automa utilizza il non determinismo per “indovinare” la posizione delle tre sequenze che costituiscono ciascuna stringa in E . Le prime due sequenze vengono salvate sullo stack (utilizzando solo il simbolo a) negli stati q_1 e q_2 . Poi per ciascun simbolo della terza sequenza vengono rimossi due elementi dallo stack (stati q_3 e q_4). Si osservi che l'automa non può accettare se la stringa in ingresso è malformata (ossia non in $a^*b^*a^*$). Se inoltre $h + k < 2l$ allora non si potranno leggere tutti i simboli della terza sequenza nello stato q_3 , oppure non si potrà transitare dallo stato q_4 allo stato q_3 , quindi l'automa non potrà accettare. Se invece $h + k > 2l$, dopo aver terminato di leggere i simboli della terza sequenza nello stato q_3 non si potrà transitare nello stato di accettazione poiché sulla cima dello stack sarà presente a e non $\$$. In conclusione, l'automa accetta una stringa in ingresso x se e solo se $x \in E$.

Esercizio 6 [10] Un grafo diretto G è detto *fortemente connesso* (*strongly connected*) se per ogni coppia di nodi $u, v \in V(G)$, esiste un percorso diretto in G da u a v . Dimostrare che il linguaggio **STRONGLY CONNECTED** contenente le codifiche dei grafi diretti fortemente connessi è NL-completo.

Soluzione: Dimostriamo innanzi tutto che il problema **STRONGLY CONNECTED** è in NL. Per semplificare questa dimostrazione sfruttiamo il fatto che $\text{NL} = \text{coNL}$, ossia $X \in \text{NL}$ se e solo se $\overline{X} \in \text{NL}$. Il complemento di **STRONGLY CONNECTED** è costituito dalle codifiche di grafi diretti in cui esiste almeno una coppia di nodi u, v tale che *non* esiste alcun percorso orientato da u a v . Sfruttiamo inoltre il fatto che il problema **PATH** è NL-completo, e dunque che il suo complemento **PATH** è in NL: pertanto, esiste una TM nondeterministica N che su input $\langle G, u, v \rangle$ esegue con spazio di lavoro logaritmico e accetta se e solo se *non* esiste un percorso orientato in G dal nodo $u \in V(G)$ al nodo $v \in V(G)$. Consideriamo dunque la seguente TM non deterministica M :

$M =$ “On input $\langle G \rangle$, where G is a directed graph:

1. Nondeterministically guess a pair of nodes $u, v \in V(G)$
2. Nondeterministically guess a deterministic branch of $N(\langle G, u, v \rangle)$
3. If the deterministic branch of $N(\langle G, u, v \rangle)$ accepts then accept
4. Otherwise, reject”

È evidente che M può implementare il passo 1 utilizzando una quantità di spazio di lavoro logaritmica nella dimensione dell'input; infatti ha necessità di memorizzare due soli vertici del grafo G , e quindi utilizza $O(\log |V(G)|)$ celle di lavoro. Nel passo 2, M sceglie una computazione deterministica della TM non deterministica N su input $\langle G, u, v \rangle$; poiché N opera in spazio logaritmico, la lunghezza massima di tale computazione deterministica è lineare nella dimensione dell'input, e quindi M può tenere traccia dei passi della computazione deterministica utilizzando una quantità logaritmica di celle di lavoro (oltre a quelle utilizzate da N). Infine, i passi 3 e 4 richiedono una quantità costante di celle di lavoro. In conclusione, M è una NTM che accetterà se e solo se esistono $u, v \in V(G)$ tali che esiste un ramo di computazione deterministica accettante di $N(\langle G, u, v \rangle)$, ossia se e solo se $u, v \in V(G)$ tali che $(G, u, v) \in \overline{\text{PATH}}$, ossia se e solo se $\langle G \rangle \in \overline{\text{STRONGLY CONNECTED}}$. Quindi **STRONGLY CONNECTED** $\in \text{coNL} = \text{NL}$.

Dimostriamo ora che **STRONGLY CONNECTED** è NL-hard esibendo una riduzione in spazio logaritmico da **PATH** a **STRONGLY CONNECTED**. In particolare, consideriamo il seguente trasduttore logaritmico T :

$T =$ “On input $\langle G, s, t \rangle$, where G is a directed graph and $s, t \in V(G)$:

1. For each element $x \in V(G)$:
 2. Copy x on the output tape as an element in $V(G')$
 3. Copy (x, s) on the output tape as an element in $E(G')$
 4. Copy (t, x) on the output tape as an element in $E(G')$
5. For each element $(x, y) \in E(G)$:
 6. Copy (x, y) on the output tape as an element in $E(G')$ ”

Il trasduttore T trasforma l'istanza di **PATH** $\langle G, s, t \rangle$ in una istanza di **STRONGLY CONNECTED** $\langle G' \rangle$, ove G' è derivato da G aggiungendo tutti gli archi diretti da t a tutti i nodi di G , e tutti gli archi diretti da ogni nodo di G al nodo s (in sintesi, $V(G') = V(G)$ e $E(G') = E(G) \cup \{(x, s) \mid x \in V(G)\} \cup \{(t, x) \mid x \in V(G)\}$). È facile verificare che T può derivare la codifica di G' utilizzando spazio logaritmico nella dimensione di G : in effetti, deve implementare un ciclo che itera su tutti i nodi del grafo G , ed un altro ciclo che itera su tutti gli archi di G ; quindi necessita di un numero di celle di lavoro logaritmico in $|\langle G \rangle|$.

Supponiamo che $\langle G, s, t \rangle \in \text{PATH}$. Pertanto esiste un percorso diretto da s a t nel grafo G . Consideriamo due qualunque nodi $u, v \in V(G') = V(G)$. Per costruzione, esiste in G' un arco da u a s ed un arco da t a v ; dunque utilizzando il percorso da s a t che già era in G , esiste un percorso in G' da u a v . Pertanto, $\langle G' \rangle \in \text{STRONGLY CONNECTED}$.

Viceversa, supponiamo che $\langle G' \rangle \in \text{STRONGLY CONNECTED}$. Pertanto, per ogni possibile coppia $u, v \in V(G') = V(G)$, esiste un percorso da u a v . Di conseguenza, esiste in G' un percorso orientato da $u = s$ a $v = t$. Supponiamo che tale percorso utilizzi un arco e in $E(G')$ ma non in $E(G)$: per costruzione tale arco e deve essere entrante in s oppure uscente da t . Tutto il sottopercorso ad anello che porta da s a rientrare in s attraverso l'arco e , ovvero ad

uscire da t attraverso l'arco e per poi rientrare in t , può essere rimosso: il risultato è ancora un percorso in $E(G')$ da s a t che non contiene e . Dunque concludiamo che esiste un percorso da s a t che fa uso esclusivamente di archi in $E(G)$, e pertanto $\langle G, s, t \rangle \in \text{PATH}$.

In conclusione, STRONGLY CONNECTED è in NL ed è NL-hard, dunque è NL-completo, come volevasi dimostrare.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

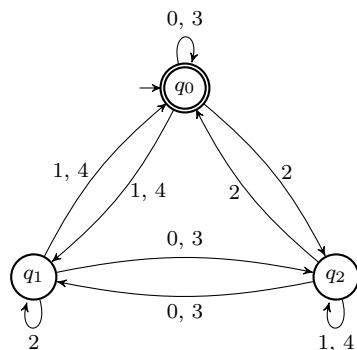
Compito scritto del 19 gennaio 2022

Esercizio 1 [5] Determinare un automa a stati finiti deterministico (DFA) per il linguaggio contenente la stringa vuota e tutti i numeri in base 5 multipli di 3.

Soluzione: L'automa a stati finiti richiesto può essere costruito considerando che l'unica informazione necessaria da memorizzare è il valore modulo 3 del numero rappresentato dalle cifre lette man mano. In altri termini, se v è il valore del numero rappresentato dalle cifre in base 5 lette fino ad un certo punto, e la successiva cifra letta è $b \in \{0, \dots, 4\}$, allora il nuovo valore rappresentato dalle cifre lette sarà $v' = v \times 5 + b$; tuttavia, è sufficiente memorizzare negli stati dell'automa soltanto il resto della divisione per tre, in quanto $v' \bmod 3 = ((v \bmod 3) \times 5 + b) \bmod 3$. Si ha dunque la seguente tabella:

| $v \bmod 3$ | $b = 0$ | $b = 1$ | $b = 2$ | $b = 3$ | $b = 4$ |
|-------------|---------|---------|---------|---------|---------|
| 0 | 0 | 1 | 2 | 0 | 1 |
| 1 | 2 | 0 | 1 | 2 | 0 |
| 2 | 1 | 2 | 0 | 1 | 2 |

È ora immediato derivare dalla tabella il seguente DFA:

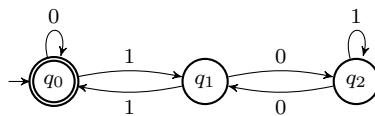


Si osservi che, come richiesto, l'automa accetta anche la stringa vuota ϵ .

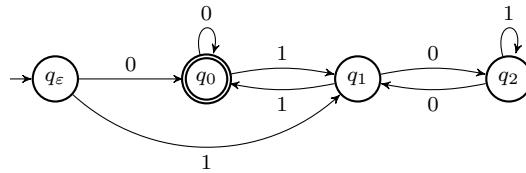
Esercizio 2 [5] Determinare una espressione regolare per il linguaggio contenente tutti i numeri in base 2 multipli di 3.

Soluzione: L'espressione regolare può essere derivata costruendo un automa a stati finiti che accetta le stringhe del linguaggio. Si osservi che il linguaggio non contiene la stringa vuota, in quanto questa non rappresenta alcun numero. Abbiamo due possibili alternative: (1) derivare l'espressione regolare da un automa che accetta esattamente tutti e soli gli elementi del linguaggio, oppure (2) derivare l'espressione di un automa che accetta anche la stringa vuota, poi modificare tale espressione regolare in modo da escludere ε .

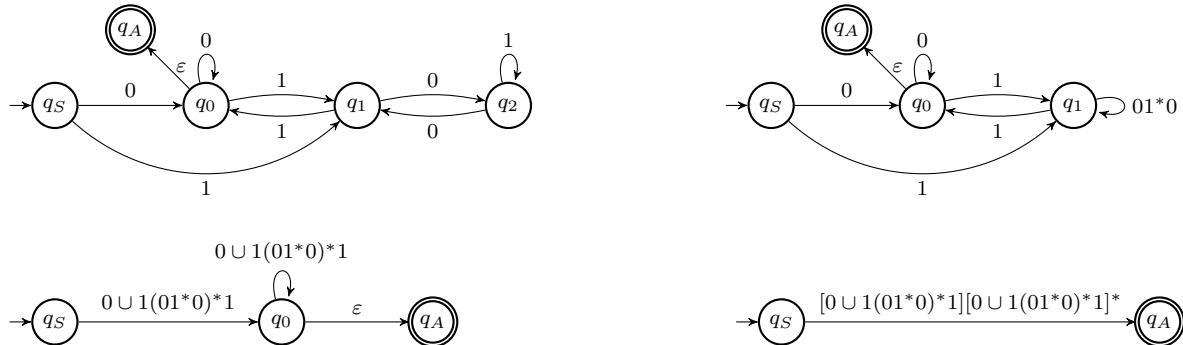
Un automa che accetta i numeri binari multipli di 3 e la stringa vuota deve semplicemente memorizzare i valori modulo 3 del numero man mano letto, assumendo inizialmente che la stringa vuota corrisponda al numero zero. Quindi:



Per escludere la stringa vuota è necessario introdurre uno stato iniziale q_ε non accettante:



Convertendo in GNFA e rimuovendo i vari nodi si ottiene:



Quindi l'espressione regolare per i numeri binari multipli di 3 è $[0 \cup 1(01^*0)^*1]^+$. Si osservi che utilizzando il secondo approccio ed eliminando i nodi nello stesso ordine si sarebbe ottenuta l'espressione $[0 \cup 1(01^*0)^*1]^*$, la cui unica differenza consiste nel generare anche la stringa vuota ε .

Esercizio 3 [6] Si consideri il linguaggio $\mathcal{L} = \{x y^n x y x^n y \mid x, y \in \{0, 1\}, x \neq y, n > 0\}$. \mathcal{L} è un linguaggio libero dal contesto? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio \mathcal{L} è libero dal contesto (CFL). Per dimostrarlo si può, in alternativa, esibire un PDA che riconosca gli elementi di \mathcal{L} oppure esibire una grammatica G tale che $L(G) = \mathcal{L}$.

Consideriamo la seguente grammatica G :

$$S \rightarrow 0A1 \mid 1B0 \quad A \rightarrow 1A0 \mid 1010 \quad B \rightarrow 0B1 \mid 0101.$$

Dimostriamo che $L(G) = \mathcal{L}$, ossia che $\mathcal{L} \subseteq L(G)$ e $L(G) \subseteq \mathcal{L}$.

Sia $w \in \mathcal{L}$, e dimostriamo per induzione che $w \in L(G)$. I più piccoli elementi di \mathcal{L} si ottengono per $n = 1$, e dunque sono 010101 e 101010. Entrambi possono essere derivati da S :

$$S \Rightarrow 0A1 \Rightarrow 010101 \quad S \Rightarrow 1B0 \Rightarrow 101010$$

Dunque $\{010101, 101010\} \subseteq L(G)$. Supponiamo ora che l'ipotesi induttiva valga per tutti gli elementi di \mathcal{L} fino al valore $n = N$, e dimostriamo che essa vale anche per $n = N + 1$. Sia dunque $w \in \mathcal{L}$ della forma $xy^{N+1}yx^{N+1}y$. Per semplificare la spiegazione, supponiamo anche che $x = 0$ e $y = 1$. Poiché vale l'ipotesi induttiva, $01^N 010^N 1 \in L(G)$, dunque $S \Rightarrow^* 01^N 010^N 1$. Ora la prima regola applicata deve essere necessariamente $S \rightarrow 0A1$, altrimenti non potrebbe essere possibile generare il primo simbolo 0 della stringa. Poiché inoltre A può generare solo stringhe terminali o stringhe contenenti A , l'ultima regola applicata deve essere stata $A \rightarrow 1010$. La sottostringa 1010 occorre in un solo punto, dunque si ha:

$$S \Rightarrow 0A1 \Rightarrow^* 01^{N-1} A 0^{N-1} 1 \Rightarrow 01^{N-1} 1010 0^{N-1} 1$$

Consideriamo ora la derivazione da S identica a questa, ma in cui l'ultima regola applicata è sostituita dalle due regole, in successione, $A \rightarrow 1A0$ e $A \rightarrow 1010$:

$$S \Rightarrow 0A1 \Rightarrow^* 01^{N-1} A 0^{N-1} 1 \Rightarrow 01^{N-1} 1 A 0 0^{N-1} 1 \Rightarrow 01^{N-1} 1 1010 0 0^{N-1} 1$$

Pertanto, $S \Rightarrow^* 01^{N+1} 010^{N+1} 1$. A causa della simmetria delle regole della grammatica, l'identico ragionamento si applica nel caso in cui $x = 1$ e $y = 0$, utilizzando le regole coinvolgenti il simbolo B . Pertanto resta dimostrato che la stringa $w = xy^{N+1}yx^{N+1}y \in L(G)$. Per induzione dunque si ha che $\mathcal{L} \subseteq L(G)$.

Dimostriamo ora che ogni stringa terminale generata dalla grammatica è inclusa in \mathcal{L} . Supponiamo che la prima regola applicata da S sia $S \rightarrow 0A1$. A causa della forma delle regole per espandere A , qualunque stringa terminale derivata da $0A1$ deve essere costituita da un certo numero $N \geq 0$ di applicazioni della regola $A \rightarrow 1A0$, seguite al termine dall'applicazione della regola $A \rightarrow 1010$. Si ha perciò:

$$S \Rightarrow 0A1 \Rightarrow^* 01^N A 0^N 1 \Rightarrow 01^N 1010 0^N 1$$

Tale stringa terminale appartiene ad \mathcal{L} (ponendo nella definizione del linguaggio $x = 0$, $y = 1$ e $n = N + 1 > 0$). Per la simmetria della grammatica e della definizione del linguaggio, lo stesso ragionamento si applica quando la prima regola applicata da S è $S \rightarrow 1B0$. Pertanto $L(G) \subseteq \mathcal{L}$.

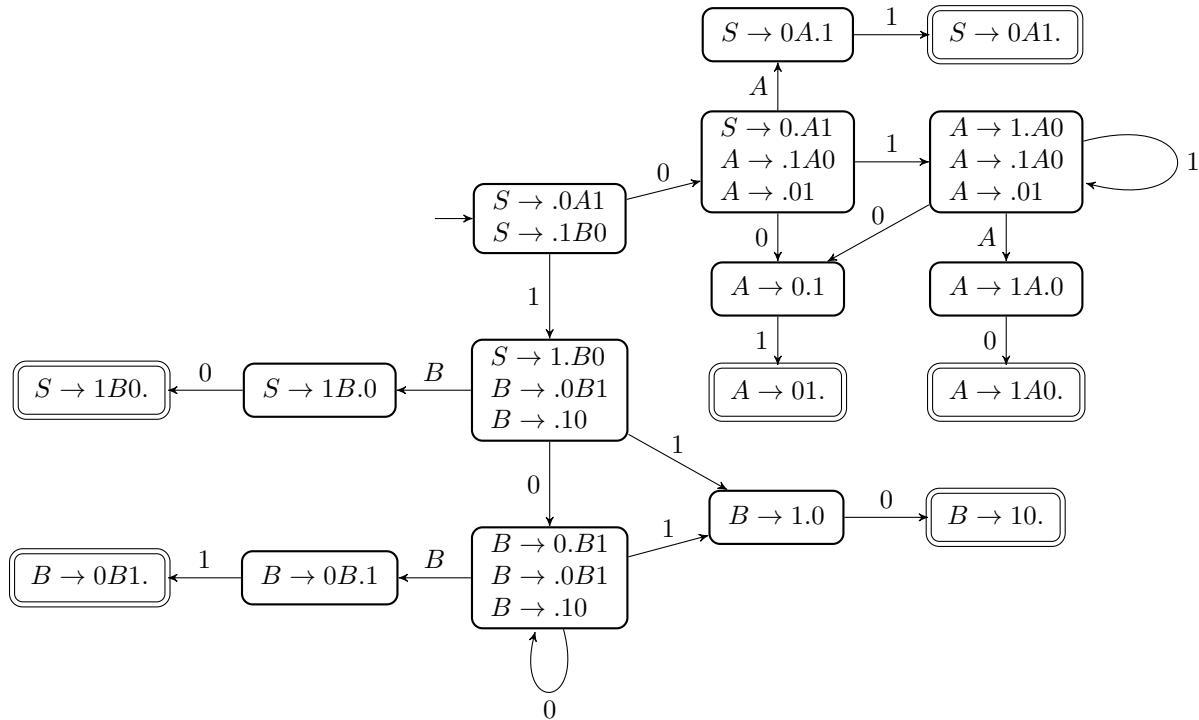
Concludiamo che $\mathcal{L} = L(G)$, e quindi che \mathcal{L} è CFL.

Esercizio 4 [6] Si consideri la grammatica G con variabile iniziale S :

$$S \rightarrow 0A1 \mid 1B0 \quad A \rightarrow 1A0 \mid 01 \quad B \rightarrow 0B1 \mid 10.$$

La grammatica è deterministica? Giustificare la risposta con una dimostrazione.

Soluzione: Per verificare se la grammatica G è deterministica utilizziamo il DK-test.



Poiché tutti gli stati finali dell'automa DK contengono una sola regola, la grammatica è deterministica.

Esercizio 5 [8] Siano A e B linguaggi Turing-riconoscibili (ricorsivamente enumerabili). Dimostrare che sia $A \# B = \{x \# y \mid x \in A, y \in B\}$ (ove $\#$ è un simbolo non incluso in A o B) che $A B = \{x y \mid x \in A, y \in B\}$ sono Turing-riconoscibili.

Soluzione: Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B , rispettivamente. È possibile costruire due diverse TM, basate su M_A e M_B , la prima deterministica per riconoscere se una stringa è in $A\#B$ e la seconda nondeterministica per riconoscere se una stringa in ingresso è in $A B$ (si osservi che il nondeterminismo, utilizzato per semplicità per “indovinare” una suddivisione della stringa in ingresso in due sottostringhe x e y , non costituisce un problema, perché per ogni TM non deterministica esiste una TM deterministica equivalente).

Per sottolineare come i due linguaggi $A\#B$ e $A B$ siano affini, svolgiamo l'esercizio utilizzando una tecnica di dimostrazione alternativa. Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM E_A e E_B che stampano in uscita tutte e sole le stringhe dei linguaggi A e B , rispettivamente. Consideriamo dunque la seguente TM E [ovvero E'] che enumera gli elementi di $A\#B$ [ovvero gli elementi di $A B$]:

- E [E'] = “On any input:
 - 1. Ignore the input
 - 2. for $k = 1$ to ∞ :
 - 3. Emulate the TM E_A for k steps
 - 4. For any string x printed by E_A :
 - 5. Emulate the TM E_B for k steps
 - 6. For any string y printed by E_B :
 - 7. Print the string $x\#y$ [the string xy]”

Si osservi che gli enumeratori in linea di principio non terminano mai, quindi non è corretto inserire nell'algoritmo una emulazione senza limiti sul numero di passi. In questo caso tipicamente l'enumeratore stamperebbe *soltanto* le stringhe $\bar{x}\#y$ [ovvero $\bar{x}y$], ove $y \in B$ ma \bar{x} è la *prima* stringa stampata da E_A .

È immediato verificare che ogni stringa stampata dall'enumeratore fa parte del linguaggio, in quanto composta da sottostringhe stampate da E_A e E_B . D'altra parte, supponiamo che $w \in A\#B$ [ovvero $w \in A B$]. Dunque w contiene due sottostringhe $x \in A$ e $y \in B$. L'enumeratore E_A stamperà certamente x dopo k_A di passi, mentre l'enumeratore E_B stamperà y dopo k_B passi. Perciò la stringa w verrà stampata dall'emulatore E [E'] nella iterazione corrispondente a $k = \max\{k_A, k_B\}$. Dunque E enumera $A\#B$, E' enumera $A B$, e pertanto entrambi i linguaggi sono ricorsivamente enumerabili.

Esercizio 6 [10] Il problema DOMINATING SET è il seguente: dato un grafo non diretto $G = (V, E)$, ed un numero intero k , determinare se esiste un sottoinsieme di nodi $V' \subseteq V$ di cardinalità k tale che ogni nodo del grafo o appartiene a V' oppure è adiacente ad un nodo in V' (o entrambe le cose). Dimostrare che DOMINATING SET è NP-completo.

Soluzione: Per prima cosa dimostriamo che DOMINATING SET (DS) appartiene a NP. Il problema è polinomialmente verificabile perché ogni istanza che fa parte del linguaggio ha come certificato il sottoinsieme di nodi del grafo che costituisce il dominating set: è certamente di dimensione non superiore al numero di nodi del grafo, e verificare che ogni nodo del grafo fa parte od è adiacente al sottoinsieme può essere facilmente realizzato da un algoritmo che esegue in tempo polinomiale nella dimensione dell'istanza del problema:

M= “On input $\langle G = (V, E), k, V' \rangle$, where G is a graph, $V' \subseteq V$:

1. if $|V'| > k$: reject
2. for each node v in V :
 3. if $v \in V'$ then continue with next node in step 1
 4. for each edge $e \in E$:
 5. if e links v to a node in V' , continue with next node in step 1
 6. Reject, because node v is not dominated by V'
 7. Accept, because all nodes in V are dominated by V' ”

Il numero totale di passi eseguiti dall'algoritmo è $O(n m n) = O(n^4)$, ove $n = |V(G)|$ e $m = |E(G)| = O(n^2)$.

Consideriamo ora una riduzione polinomiale da VERTEX COVER (VC) a DS. Sia $(G = (V, E), k)$ una istanza di VC. Costruiamo un nuovo grafo $G' = (V', E')$ in questo modo: $V' = V \cup V_E$ è costituito dai nodi di G e da un nodo v_e per ciascun arco $e \in E$; in totale quindi $|V'| = n + m = |V| + |E|$. $E' = E \cup E_V$ è costituito dagli archi di G e, per ciascun nodo $v_e \in V_E$, da due archi (v_e, v) e (v_e, w) ove $e = (v, w)$; in totale quindi $|E'| = 3m = 3|E|$. Possiamo dimostrare che $(G, k) \in \text{VC}$ se e solo se $(G', k + s) \in \text{DS}$, ove s è il numero di nodi $S \subseteq V$ “isolati” (senza archi incidenti).



Supponiamo che $(G, k) \in \text{VC}$; dunque esiste $U \subseteq V$ tale che $|U| = k$ ed ogni arco di G è incidente ad almeno un nodo di U . Consideriamo il sottoinsieme di nodi $U' = U \cup S$ in G' (esiste perché tutti i nodi di G sono anche nodi di G'), e dimostriamo che è un dominating set di dimensione $k + s$. Infatti, sia $x \in V(G') = V \cup V_E$: se $x \in V$, allora o $x \in S$, e dunque $x \in U'$, oppure esiste un arco $e \in E$ incidente su x . Poiché U è un ricoprimento degli archi in G , esiste un nodo $y \in U$ tale che $e = (x, y)$; pertanto, il nodo x è dominato dal nodo $y \in U'$. Se invece $x \in V_E$, allora $x = v_e$ per un certo arco $e \in E$: dunque, U deve contenere un nodo y che ricopre e ; allora, per costruzione di E_V , $(y, v_e) \in E_V$, e quindi x è dominato da $y \in U'$.

Per la direzione opposta, supponiamo che $(G', k + s) \in \text{DS}$, e quindi esiste un sottoinsieme U' , con $|U'| = k + s$, che domina ogni nodo di G' . Risulta evidente che $S \subseteq U'$, perché l'unico

modo per dominare nodi isolati è inserirli nel sottoinsieme dominante. Un'altra osservazione è che se U' contiene un qualunque nodo $v_e \in V_E$, è possibile sostituire in U' il nodo v_e con uno qualunque dei due nodi v, w tali che $e = (v, w)$. Infatti per costruzione di G' il nodo v_e può dominare solo se stesso, v e w ; ma v (o equivalentemente w) domina almeno se stesso, w e v_e , quindi sostituendo v_e con v si ottiene un dominating set di dimensione pari od inferiore a quella di U' che domina almeno lo stesso insieme di nodi di G' . Consideriamo dunque il dominating set U'' in cui ogni nodo v_e è stato sostituito da un nodo in V come appena descritto, e sia $U = U'' \setminus S$. Il sottoinsieme U ha cardinalità $\leq k$, è un sottoinsieme di nodi di G , ed è un ricoprimento degli archi in G . Infatti, sia $e \in E$, con $e = (v, w)$. Poiché il nodo v_e deve essere dominato da qualche nodo in U'' , per costruzione v, w , od entrambi appartengono al sottoinsieme dominante U'' . Naturalmente $v, w \notin S$, quindi v, w oppure entrambi, appartengono ad U . Pertanto, l'arco e risulta ricoperto da un elemento di U .

Abbiamo dunque dimostrato che la trasformazione da (G, k) a $(G', k + s)$ è una riduzione tra problemi. È inoltre evidente che tale trasformazione può essere costruita in tempo polinomiale. Pertanto, $\text{VC} \leq_m \text{DS}$, e quindi DS è NP-hard in quanto VC è NP-completo. Ciò conclude la dimostrazione di NP-completezza di DS.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

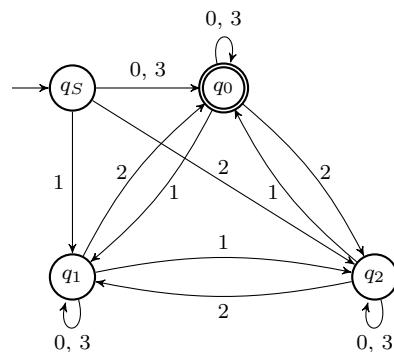
Compito scritto del 2 febbraio 2022

Esercizio 1 [5] Determinare un automa a stati finiti deterministico (DFA) per il linguaggio contenente tutti e soli i numeri in base 4 multipli di 3.

Soluzione: L'automa a stati finiti richiesto può essere costruito considerando che l'unica informazione necessaria da memorizzare è il valore modulo 3 del numero rappresentato dalle cifre lette man mano. In altri termini, se v è il valore del numero rappresentato dalle cifre in base 4 lette fino ad un certo punto, e la successiva cifra letta è $b \in \{0, \dots, 3\}$, allora il nuovo valore rappresentato dalle cifre lette sarà $v' = v \times 4 + b$; tuttavia, è sufficiente memorizzare negli stati dell'automa soltanto il resto della divisione per tre, in quanto $v' \bmod 3 = ((v \bmod 3) \times 4 + b) \bmod 3$. Si ha dunque la seguente tabella:

| $v \bmod 3$ | $b = 0$ | $b = 1$ | $b = 2$ | $b = 3$ |
|-------------|---------|---------|---------|---------|
| 0 | 0 | 1 | 2 | 0 |
| 1 | 1 | 2 | 0 | 1 |
| 2 | 2 | 0 | 1 | 2 |

È ora immediato derivare dalla tabella il seguente DFA:



Lo stato iniziale q_S ha il solo scopo di rifiutare la stringa vuota ε .

Esercizio 2 [6] Si consideri l'insieme $\Sigma = \{a, b\}$ e la seguente grammatica G con variabile iniziale S :

$$S \rightarrow aSb \mid bS \mid Sa \mid \varepsilon$$

Quale è l'insieme di stringhe $L(G)$ generate da G ? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio generato dalla grammatica include tutte le stringhe sull'alfabeto di terminali Σ , ossia $L(G) = \Sigma^*$. La dimostrazione è per induzione. Come base dell'induzione, consideriamo che la stringa vuota ε è generata dalla grammatica grazie alla regola $S \rightarrow \varepsilon$. Supponiamo dunque come ipotesi induttiva che la grammatica sia in grado di generare tutte le stringhe in Σ^* di lunghezza inferiore a $n \geq 0$, e consideriamo una stringa x di lunghezza n . Possiamo distinguere i seguenti tre casi:

1. il primo carattere di x è ‘b’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow bS$: la stringa ottenuta è pertanto bS . Poiché $x = by$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera il primo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.
2. l'ultimo carattere di x è ‘a’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow Sa$: la stringa ottenuta è pertanto Sa . Poiché $x = ya$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera l'ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.
3. il primo carattere di x è ‘a’ e l'ultimo carattere di x è ‘b’: Si consideri una derivazione da S in cui la prima regola applicata è $S \rightarrow aSb$: la stringa ottenuta è pertanto aSb . Poiché $x = ayb$ ove $y \in \Sigma^*$ e $|y| = n - 2$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y . Perciò la derivazione costituita dalla regola che genera il primo e l'ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x . Pertanto $x \in L(G)$.

In tutti i tre casi si conclude che $x \in L(G)$. Pertanto resta dimostrato che $x \in L(G)$ per ogni possibile stringa di Σ^* , e quindi $L(G) = \Sigma^*$.

Esercizio 3 [8] Sia $B = \{w\bar{w} \mid w \in \{0, 1\}^*\}$, ove \bar{w} è la stringa ottenuta da w complementando ogni bit. Dimostrare che B non è un CFL.

Soluzione: Per assurdo, supponiamo che B sia CFL e che quindi valga per esso il “Pumping lemma”. Sia dunque p la “pumping length” per B . In questo esercizio la scelta della stringa

che contraddice le conclusioni del lemma deve essere fatta con attenzione. Ad esempio, la stringa $0^p 1^p 1^p 0^p$ non andrebbe bene, perché può essere pompata ponendo $u = 0^{p-1}$, $v = 0$, $x = \epsilon$, $y = 1$, $z = 1^{2p-1} 0^p$: infatti, $uv^i xy^i z = 0^{p-1} 1^i 1^{2p-1} 0^p = 0^{p+i-1} 1^p 1^{p+i-1} 0^p$.

Consideriamo invece la stringa $s = 0^p 1^p 0 1^p 0^p 1$: evidentemente $s \in B$ (in quanto $\overline{0^p 1^p 0} = 1^p 0^p 1$), quindi deve essere possibile determinare una suddivisione $s = uvxyz$ tale che per ogni $i \geq 0$, $uv^i xy^i z \in B$, $|vy| > 0$ e $|vxy| \leq p$. Distinguiamo i seguenti casi:

1. Se vy non contiene esattamente lo stesso numero di zero ed uno, pompando verso il basso o verso l'alto si otterrebbe una stringa con una eccedenza di zero od uno, che dunque non potrebbe far parte di B . Ciò implica anche che $|vy| > 1$.
2. Se la sottostringa vxy fosse interamente nella prima metà di s , allora pompando verso il basso si otterrebbe una stringa uxz in cui uno dei bit della seconda occorrenza di 1^p in s finirebbe nell'ultima posizione della prima metà di uxz (in quanto $|vxy| \leq p$ e $|vy| > 1$). Pertanto uxz non può essere in B perché il bit in ultima posizione della prima metà deve essere 0.
3. Analogamente, la sottostringa vxy non può essere interamente nella seconda metà di s , altrimenti pompando verso il basso uno dei bit della prima occorrenza di 1^p in s finirebbe nell'ultima posizione della prima metà di uxz , e ciò significa che tale stringa non potrebbe far parte di B perché essa termina con 1.
4. Pertanto, vxy deve contenere sia un bit della prima metà di s che un bit della seconda metà. Poiché però vy deve contenere un ugual numero di zero ed uno, l'unica possibilità è che vy sia o la stringa 01 oppure la stringa 10, ove lo 0 è quello in ultima posizione della prima metà di s . Pompando verso il basso si ottiene una stringa in cui nell'ultima posizione della prima metà vi è un 1, dunque tale stringa non può far parte di B .

Poiché non è possibile suddividere la stringa s in accordo al pumping lemma, concludiamo che il pumping lemma non può essere applicato, e pertanto che l'ipotesi che B fosse un CFL è falsa.

Esercizio 4 [6] Uno stato interno r di una macchina di Turing M è detto *inutile* se non esiste alcuna stringa di input che possa portare la macchina M ad assumere lo stato interno r . Dimostrare che il linguaggio contenente le codifiche di tutte le macchine di Turing con uno o più stati inutili è indecidibile.

Soluzione: Cominciamo con l'osservazione evidente che la proprietà di avere o meno uno stato inutile appartiene alla particolare macchina di Turing, e non è propria del linguaggio riconosciuto dalla macchina. Ad esempio, consideriamo una TM che non ha stati inutili, e modifichiamo la sua descrizione in modo da aggiungere uno stato che non è mai utilizzato in alcuna transizione. Ovviamente la macchina modificata riconosce lo stesso linguaggio della

macchina originale ma possiede uno stato inutile. Dunque non possiamo dimostrare quanto richiesto utilizzando il Teorema di Rice.

Per dimostrare l'indecidibilità del linguaggio contenente le codifiche di TM aventi almeno uno stato inutile costruiamo una riduzione dal problema di accettazione delle TM \mathcal{A}_{TM} . Come al solito dunque consideriamo una istanza $\langle M, w \rangle$ di \mathcal{A}_{TM} e mostriamo come costruire una TM N che ha uno stato inutile se e solo se $M(w)$ accetta. La difficoltà di questa costruzione, però, risiede nella macchina M stessa: infatti, dovendo simulare M , N include nei suoi stati interni anche gli stati interni di M . Se M avesse stati inutili, la macchina N potrebbe avere stati inutili anche se in effetti $M(w)$ non accettasse. Dobbiamo quindi fare attenzione a costruire N in modo da rendere ogni stato di M “utile”.

Siano Σ e Γ rispettivamente gli alfabeti di input e di nastro della macchina M ; la TM N utilizzerà $\Sigma' = \Sigma \cup \{\#\}$ e $\Gamma' = \Gamma \cup \{\#\}$, ove ‘ $\#$ ’ è un nuovo simbolo non utilizzato in M . Il simbolo ‘ $\#$ ’ serve ad implementare un ciclo in cui si entra in ogni stato interno di M : in pratica viene aggiunta una nuova transizione per ogni stato interno di M in cui leggendo ‘ $\#$ ’ si passa in uno stato interno di N che continua il ciclo.

Nella descrizione di N seguente si deve assumere che non vengono introdotti stati interni propri di N inutili a meno che questo che non sia affermato esplicitamente. In particolare, lo stato interno r di N è utilizzato solo quando esplicitamente menzionato. Inoltre si tenga presente che gli stati di accettazione e rifiuto di M non sono stati finali per N .

N = “On input x , where $x \in \Sigma'^*$:

1. If $x = \#\#$:
2. Execute a loop and enter in every internal state of M
3. Halt
4. Run M on input w
5. If $M(w)$ accepts, enter in internal state r
6. Halt”

Supponiamo per assurdo che il problema di decidere se una TM ha stati interni inutili sia decidibile, e sia dunque R un decisore per tale problema. Consideriamo allora la seguente TM D :

D = “On input $\langle M, w \rangle$, where M is a TM and $w \in \Sigma^*$:

1. Build from $\langle M, w \rangle$ the encoding of the TM N
2. Run R on $\langle N \rangle$
3. If $R(\langle N \rangle)$ accepts, then reject
4. Otherwise, if $R(\langle N \rangle)$ rejects, then accept”

Se $R(\langle N \rangle)$ accetta, allora N ha almeno uno stato interno inutile. Per costruzione di N questo stato può essere unicamente r . Ciò significa che $M(w)$ non accetta. Se invece $R(\langle N \rangle)$ rifiuta,

allora ogni stato interno di N è utile, quindi anche lo stato interno di r deve essere visitato per un certo input x . In effetti l'input x di N viene ignorato se è diverso da '#'. L'unica possibilità è che $M(w)$ accetta e quindi r sia utilizzato nel passo 5 di N . Poiché D complementa la decisione di R , D è in effetti un decisore per \mathcal{A}_{TM} , ma questo è in contraddizione con il fatto che \mathcal{A}_{TM} è indecidibile. Pertanto resta dimostrato che il problema di decidere se una TM ha stati interni inutili è indecidibile.

Esercizio 5 [6] 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.

Esercizio 6 [9] Si consideri il linguaggio costituito dalle codifiche delle formule booleane che hanno almeno due assegnazioni di verità che soddisfano la formula. Dimostrare che tale linguaggio è NP-completo.

Soluzione: Questo problema è generalmente chiamato DOUBLE SAT, ed è una generalizzazione molto semplice del problema SAT.

Si dimostra facilmente che DOUBLE SAT è in NP . Infatti, data una qualunque istanza $\langle \Phi \rangle$ che codifica una formula booleana, un certificato per l'esistenza di una soluzione è costituito da due liste di valori di verità. Il verificatore controlla che ciascuna lista contenga esattamente un valore (vero o falso) per ciascuna variabile di Φ , e che entrambe le assegnazioni di verità soddisfino la formula Φ .

Per dimostrare che DOUBLE SAT è NP-hard consideriamo la seguente riduzione dal problema SAT: considerata una generica istanza $\langle \Phi \rangle$ di SAT, sia Φ' la formula booleana ottenuta da

Φ aggiungendo una nuova variabile v e ponendo

$$\Phi' = \Phi \wedge (v \vee \bar{v}).$$

È immediato verificare che se la formula Φ è soddisfacibile allora esistono almeno due assegnazioni di verità che soddisfano Φ' : la assegnazione che soddisfa Φ estesa con $v = T$ e la stessa assegnazione estesa con $v = F$. Viceversa, se la formula Φ non è soddisfacibile, allora nemmeno Φ' può essere soddisfacibile, perché la variabile v non appare nella formula Φ e quindi non può contribuire a rendere quella parte della formula Φ' soddisfacibile.

Da tutto ciò si può concludere che DOUBLE SAT è NP-completo.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 23 giugno 2022

Esercizio 1 [5] Sia A un linguaggio regolare contenente stringhe di lunghezza pari. Per ciascuna stringa $w = w_1w_2 \cdots w_{2m} \in A$, considerare la stringa $w^\# = w_1w_3 \cdots w_{2m-1}w_2w_4 \cdots w_{2m}$. Dimostrare che il linguaggio $A^\# = \{w^\# \mid w \in A\}$ non è necessariamente regolare.

Soluzione: L'operatore “ $\#$ ” definito nel testo dell'esercizio riordina i caratteri di una stringa con un numero pari di elementi in modo da posizionare prima i caratteri originariamente con indice dispari, e poi i caratteri con indice pari.

Si osservi che il testo dell'esercizio non può in alcun modo essere travisato. Ad esempio, è manifestamente sbagliato assumere che la definizione di A sia esclusivamente che A contiene stringhe di lunghezza pari, in quanto questo non implica che A sia regolare (ad esempio: $\{0^n1^n \mid n \geq 0\}$): l'ipotesi di regolarità è quindi essenziale, oltre a quella di contenere stringhe di lunghezza pari. Od ancora, non è possibile interpretare A come il linguaggio contenente tutte le stringhe di lunghezza pari (fissato un certo alfabeto), perché l'operatore ‘ $\#$ ’ non modifica la lunghezza delle stringhe a cui è applicato, quindi $A^\#$ sarebbe identico ad A , e non sarebbe possibile dimostrare che esso è non regolare.

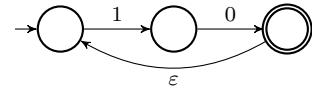
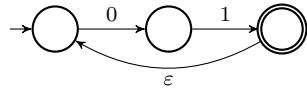
Il testo richiede di dimostrare che dato un qualunque linguaggio regolare A contenente solo stringhe di lunghezza pari non è necessariamente vero che $A^\#$ è regolare. Per svolgere l'esercizio è dunque sufficiente esibire un particolare linguaggio regolare A tale che $A^\#$ non è regolare. L'esistenza del contro-esempio dimostra l'asserto dell'esercizio.

Consideriamo dunque il linguaggio regolare $A = L(R)$, ove R è l'espressione regolare $(01)^*$. Naturalmente $A = \{(01)^n \mid n \geq 0\}$, pertanto è immediato verificare che $A^\# = \{0^n1^n \mid n \geq 0\}$. D'altra parte, $A^\#$ non è regolare, come dimostrato durante lo svolgimento del corso.

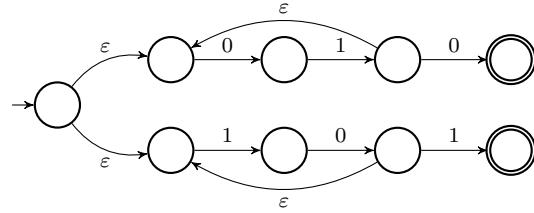
Per completezza dimostriamo che $A^\#$ non è regolare applicando il Pumping Lemma. Se infatti $A^\#$ fosse regolare, esisterebbe un valore $p > 0$ tale che ogni stringa di lunghezza maggiore di p potrebbe essere “pompata”. Considerando però la stringa $s = 0^p1^p$, ogni suddivisione di $s = xyz$ con $|xy| \leq p$ e $|y| > 0$ comporta una variazione nel numero di 0's mentre il numero di 1's rimane costante. Pertanto la stringa “pompata” non potrebbe fare parte del linguaggio. Poiché il Pumping Lemma non vale, $A^\#$ non è regolare.

Esercizio 2 [6] Determinare un DFA che riconosce il linguaggio associato alla espressione regolare $((01)^+0 \cup (10)^+1)^*$.

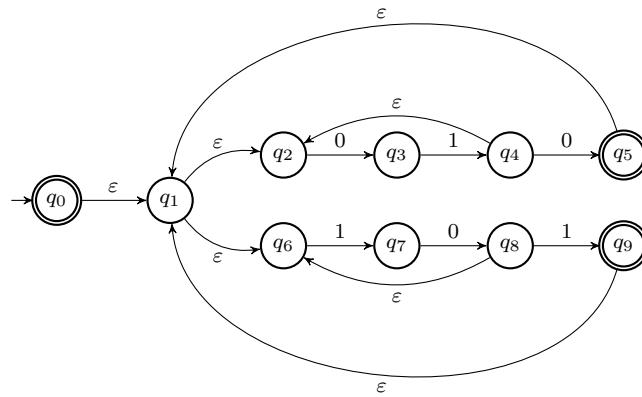
Soluzione: Deriviamo il DFA richiesto a partire da un NFA associato alla espressione regolare nel testo. La procedura meccanica di conversione da REX a NFA produce un automa con un numero considerevole di stati. È conveniente quindi cominciare a considerare gli NFA elementari ed ovviamente corretti corrispondenti alle due espressioni regolari $(01)^+$ e $(10)^+$:



Come passo successivo deriviamo lo NFA per la REX $(01)^+0 \cup (10)^+1$:

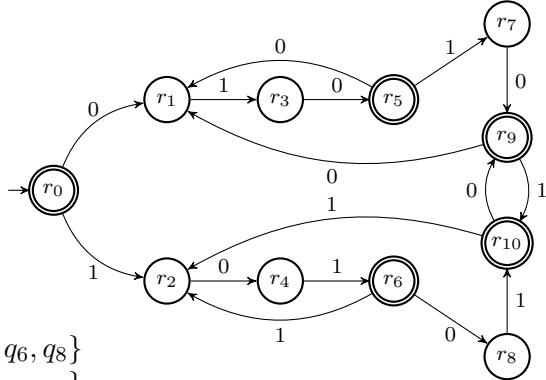


Aggiungiamo infine la trasformazione per l'operatore star ed otteniamo un NFA associato alla REX nel testo:



Infine tramite la procedura di trasformazione da NFA a DFA otteniamo:

$$\begin{aligned}
r_0 &= E(q_0) = \{q_0, q_1, q_2, q_6\} \\
r_1 &= E(q_3) = \{q_3\} \\
r_2 &= E(q_7) = \{q_7\} \\
r_3 &= E(q_4) = \{q_2, q_4\} \\
r_4 &= E(q_8) = \{q_6, q_8\} \\
r_5 &= E(q_3) \cup E(q_5) = \{q_1, q_2, q_3, q_5, q_6\} \\
r_6 &= E(q_7) \cup E(q_9) = \{q_1, q_2, q_6, q_7, q_9\} \\
r_7 &= E(q_4) \cup E(q_7) = \{q_2, q_4, q_7\} \\
r_8 &= E(q_3) \cup E(q_8) = \{q_3, q_6, q_8\} \\
r_9 &= E(q_3) \cup E(q_5) \cup E(q_8) = \{q_1, q_2, q_3, q_5, q_6, q_8\} \\
r_{10} &= E(q_4) \cup E(q_7) \cup E(q_9) = \{q_1, q_2, q_4, q_6, q_7, q_9\}
\end{aligned}$$



Esercizio 3 [6] Si consideri la grammatica $S \rightarrow AB \mid BA$, $A \rightarrow aC \mid C \mid \varepsilon$, $B \rightarrow bC \mid C$, $C \rightarrow cA \mid cB$. Determinare se la grammatica è LR(1).

Soluzione:

La grammatica non è LR(1), e per dimostrarlo in effetti è sufficiente generare lo stato iniziale dell'automa DK₁. Tale stato è di accettazione, per la presenza della regola completa ‘ $A \rightarrow :$; d’altra parte, lo stesso stato include regole in cui il punto è seguito da un simbolo incluso tra quelli di lookahead della regola completata (‘ b ’ e ‘ c ’). Perciò il DK₁-test fallisce, e la grammatica non è LR(1).

| | |
|---------------------|-----|
| $S \rightarrow .AB$ | abc |
| $S \rightarrow .BA$ | abc |
| $A \rightarrow .aC$ | bc |
| $A \rightarrow .C$ | bc |
| $A \rightarrow .$ | bc |
| $B \rightarrow .bC$ | abc |
| $B \rightarrow .C$ | abc |
| $C \rightarrow .cA$ | abc |
| $C \rightarrow .cB$ | abc |

Si osservi che a non è tra i simboli di lookahead associati alle regole che espandono A . Infatti, tali regole sono state inserite nello stato a causa della regola $S \rightarrow .AB$; i simboli di lookahead sono esclusivamente il primo simbolo terminale di ogni stringa generabile da ciò che segue A nella regola, ossia il primo simbolo di ogni stringa che può essere generata a partire da B . Ora B può generare bC , e dunque b è simbolo di lookahead; oppure può generare C , il quale a sua volta genera comunque c come primo simbolo terminale. Al contrario, poiché A può generare ε , i simboli di lookahead delle regole che espandono B coincidono con quelli delle regole che espandono S .

Esercizio 4 [6] Si consideri la grammatica $S \rightarrow ACB$, $A \rightarrow 01A \mid \varepsilon$, $B \rightarrow B10 \mid \varepsilon$, $C \rightarrow 00C \mid 11C \mid \varepsilon$. Determinare se il linguaggio generato dalla grammatica è regolare.

Soluzione: È facile osservare che la grammatica ha una forma particolare: S espande nella concatenazione delle tre variabili A , C e B . A propria volta, ciascuna di queste variabili genera una stringa terminale concatenata alla variabile stessa, ovvero la stringa vuota. In altre parole, la stringa generata inizialmente da A dovrà essere espansa utilizzando esclusivamente

la variabile A ; la stessa cosa si verifica per le stringhe generate da B e da C . Perciò se $L(X)$ è il linguaggio contenente le stringhe generate a partire dalla variabile X e “ \circ ” rappresenta la concatenazione di linguaggi, il linguaggio generato dalla grammatica è

$$L(S) = L(A) \circ L(C) \circ L(B).$$

Consideriamo dunque le regole che espandono la variabile A : “ $A \rightarrow 01A$ ” e “ $A \rightarrow \varepsilon$ ”. È immediato dimostrare che $L(A) = \{(01)^n \mid n \geq 0\}$, ossia è il linguaggio associato alla espressione regolare “ $(01)^*$ ”. Analogamente le regole che espandono la variabile B generano le stringhe del linguaggio $L(B) = \{(10)^n \mid n \geq 0\}$, ossia il linguaggio associato alla espressione regolare “ $(10)^*$ ”. Infine le tre regole che espandono la variabile C generano il linguaggio $L(C) = \{w_1 w_2 \dots w_n \mid n \geq 0, w_i = 00 \text{ oppure } 11, 1 \leq i \leq n\}$, ossia il linguaggio associato alla espressione regolare “ $(00 \cup 11)^*$ ”.

Poiché $L(A)$, $L(B)$ e $L(C)$ sono linguaggi regolari e la concatenazione di linguaggi regolari è un linguaggio regolare, ne consegue che il linguaggio generato dalla grammatica è regolare. In effetti una REX associata a tale linguaggio è “ $(01)^* (00 \cup 11)^* (10)^*$ ”.

Esercizio 5 [8] Si consideri il modello di calcolo PDA_2 analogo agli automi a pila (PDA) ma aventi due stack invece di uno. Si dimostri che tale modello di calcolo non è equivalente a quello dei PDA.

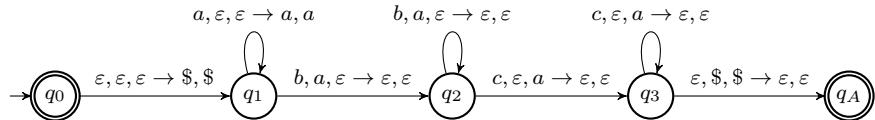
Soluzione: Ovviamente il modello di calcolo PDA rappresentato dagli automi a pila con un solo stack non può essere più potente di quello con automi a pila con due stack PDA_2 , poiché qualsiasi automa con un solo stack può essere simulato senza difficoltà da un automa con due stack. Perciò il testo dell'esercizio richiede di dimostrare che il modello di calcolo PDA_2 è strettamente più potente del modello PDA. La dimostrazione più semplice consiste nell'esibire un linguaggio che non può essere deciso da un PDA e può essere deciso da un PDA_2 .

Consideriamo dunque il linguaggio $C = \{a^n b^n c^n \mid n \geq 0\}$. Sappiamo che esso non è CFL, dunque non esiste alcun PDA che possa riconoscere i suoi elementi. Per completezza, dimostriamo che C non è CFL. Supponiamo per assurdo che lo sia, e dunque che valga per esso il Pumping Lemma per una determinata lunghezza $p > 0$. Sia dunque $s = a^p b^p c^p \in C$. Poiché $|s| \geq p$, il Pumping Lemma afferma che deve esistere una suddivisione $s = uvxyz$ tale che $|vxy| \leq p$, $|vy| > 0$ e $uv^i xy^i z \in C$ per ogni $i \geq 0$. Possono darsi solo due casi:

1. v e y contengono entrambi un solo tipo di simboli: la stringa $uv^i xy^i z$ con $i \neq 0$ non può contenere lo stesso numero di a , b e c , quindi $uv^i xy^i z \notin C$;
2. v oppure y (oppure entrambi) contengono più di un tipo di simboli: nella stringa $uv^i xy^i z$ con $i \neq 0$ esiste una b che precede una a oppure una c che precede una b , quindi $uv^i xy^i z \notin C$.

Poiché non è possibile suddividere la stringa s in modo da soddisfare il Pumping Lemma, C non può essere CFL.

D'altra parte, un automa con due stack può facilmente riconoscere gli elementi di C . Si consideri ad esempio il seguente PDA₂, ove l'etichetta $v, w, x \rightarrow y, z$ indica che l'automa legge il simbolo v dall'input, w dal primo stack e x dal secondo stack, poi scrive y sul primo stack e z sul secondo stack. Come sempre, ε indica che l'automa non legge o scrive nulla.



L'automa comincia a leggere i simboli di tipo a e li copia su entrambi gli stack. Successivamente legge i simboli di tipo b e li confronta numericamente con i simboli scritti sul primo stack, senza modificare il secondo stack. Infine legge i simboli di tipo c e li confronta numericamente con i simboli scritti sul secondo stack. L'automa accetta se l'input può essere letto interamente ed entrambi gli stack sono vuoti.

In effetti è possibile dimostrare che un automa con due stack è in grado di simulare l'esecuzione di una generica macchina di Turing, quindi il modello di calcolo PDA₂ riconosce l'intera classe dei linguaggi ricorsivamente enumerabili.

Esercizio 6 [9] Una istanza del problema BALANCED PARTITION è costituita da $2m$ numeri interi non negativi (non necessariamente distinti tra loro) la cui somma è pari ($2s$). Il problema richiede di decidere se è possibile suddividere i numeri in due sottoinsiemi ciascuno con m elementi e tali che la somma degli elementi in ciascun sottoinsieme sia uguale a s . Dimostrare che BALANCED PARTITION è NP-completo.

Soluzione: BALANCED PARTITION è un problema polinomialmente verificabile. Infatti, sia U un multi-insieme di numeri interi non negativi con somma $2s$ che ammette una partizione in due multi-insiemi I e J ($I \cup J = U$, $I \cap J = \emptyset$) tale che $\sum_{x \in I} x = \sum_{x \in J} x = s$. Un certificato per tale istanza-sì di BALANCED PARTITION è costituito semplicemente da uno dei due sottoinsiemi. Esiste dunque un verificatore che opera in tempo polinomiale nella dimensione dell'istanza:

V= “On input $\langle U, I \rangle$, where U is a multi-set of non-negative integers:

1. Verify that $I \subset U$ and $2|I| = |U|$
2. Compute $J = U \setminus I$
3. Compute $v = \sum_{x \in I} x$
4. Compute $w = \sum_{x \in J} x$
5. Accept if $v = w$, reject otherwise”

Pertanto, BALANCED PARTITION è incluso in NP. Per dimostrare che è anche NP-hard, possiamo esibire una riduzione polinomiale da un altro problema NP-hard, ad esempio SUBSET SUM.

Sia dunque (S, t) una istanza di SUBSET SUM, ossia un multi-insieme di numeri interi S ed un intero t . Dalla dimostrazione di NP-hardness fatta a lezione è evidente che questo problema è NP-hard anche restringendo le istanze agli interi non negativi. Consideriamo la riduzione polinomiale che trasforma (S, t) nel multi-insieme U come segue. Sia $n = |S|$ e sia $\mu = \sum_{x \in S} x$. Se $\mu < t$, allora (S, t) è certamente una istanza-no, dunque la riduzione polinomiale si limita a costruire una istanza-no elementare di BALANCED PARTITION. Altimenti, il multi-insieme U è costituito da S , dall'intero non negativo $\lambda = 2t - \mu$, e da $n + 1$ valori interi nulli (ossia n zeri $z_0 = \dots = z_n = 0$). Ovviamente $|U| = |S| + 1 + n + 1 = 2(n + 1)$.

Supponiamo che (S, t) sia una istanza-sì di SUBSET SUM, e dunque che esista $T \subseteq S$ tale che $\sum_{x \in T} x = t$. Sia $q = |T|$, e consideriamo il multi-insieme $W = T \cup \{z_0, \dots, z_{n-q}\}$ (si osservi che se $q = n$ allora W include il solo elemento z_0). Quindi $|W| = n + 1$. Inoltre il sottoinsieme $Z = U \setminus W$ è costituito da $n - q$ elementi di $S \setminus T$, da λ , e da q zeri $\{z_{n-q+1}, \dots, z_n\}$ (ovviamente $|Z| = (n - q) + 1 + (n - n + q - 1 + 1) = n + 1$). Si ha:

$$\sum_{x \in W} x = \sum_{x \in T} x + z_0 + \dots + z_{n-q} = t$$

e

$$\sum_{x \in Z} x = \sum_{x \in S \setminus T} x + \lambda + z_{n-q+1} + \dots + z_n = (\mu - t) + (2t - \mu) = t.$$

Pertanto U è una istanza-sì di BALANCED PARTITION.

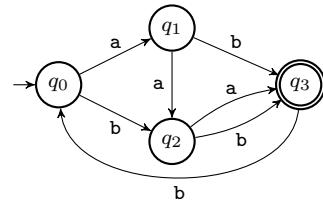
Supponiamo al contrario che U sia una istanza-sì di BALANCED PARTITION derivata dalla riduzione di una istanza (S, t) , e siano W e Z tali che $W \cap Z = \emptyset$, $|W| = |Z| = n + 1$, e $\sum_{x \in W} x = \sum_{x \in Z} x = t$. Senza perdita di generalità supponiamo che $\lambda \notin W$, e sia $T = W \setminus \{z_0, \dots, z_n\}$, ossia T è il multi-insieme W a cui sono stati rimossi gli zeri z_i eventualmente presenti. Pertanto $T \subseteq S$, ed inoltre $\sum_{x \in T} x = t$. Perciò (S, t) è una istanza-sì di SUBSET SUM.

Automi e Linguaggi (M. Cesati)

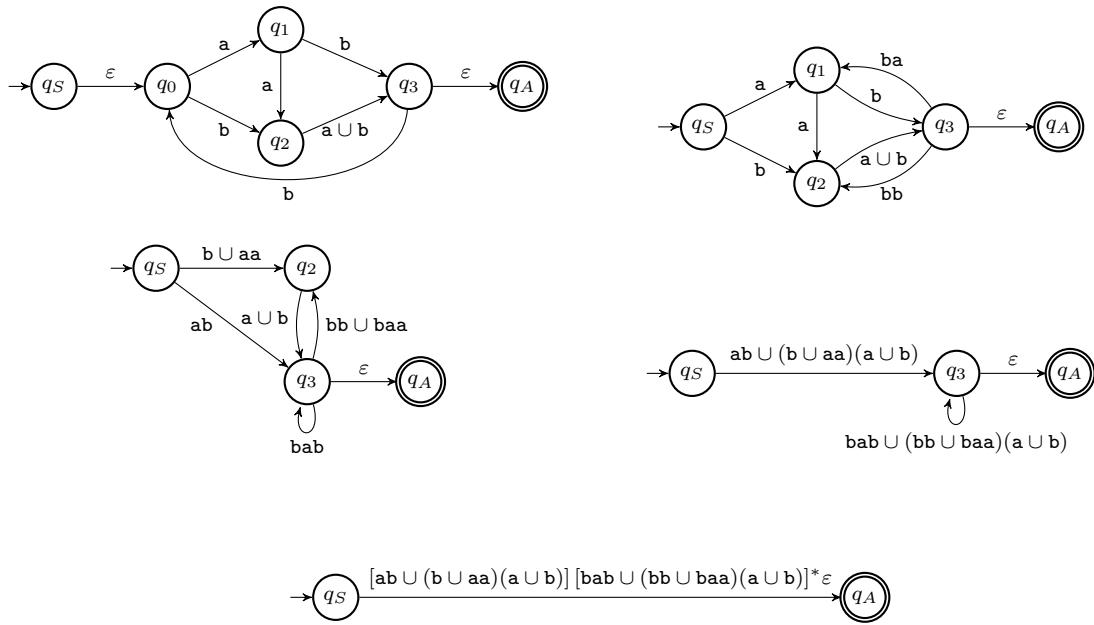
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 13 luglio 2022

Esercizio 1 [6] Determinare una espressione regolare per il linguaggio riconosciuto dal DFA:



Soluzione: Convertiamo il DFA in un GNFA e rimuoviamo nell'ordine i nodi q_0 , q_1 , q_2 e q_3 . Si ottiene:

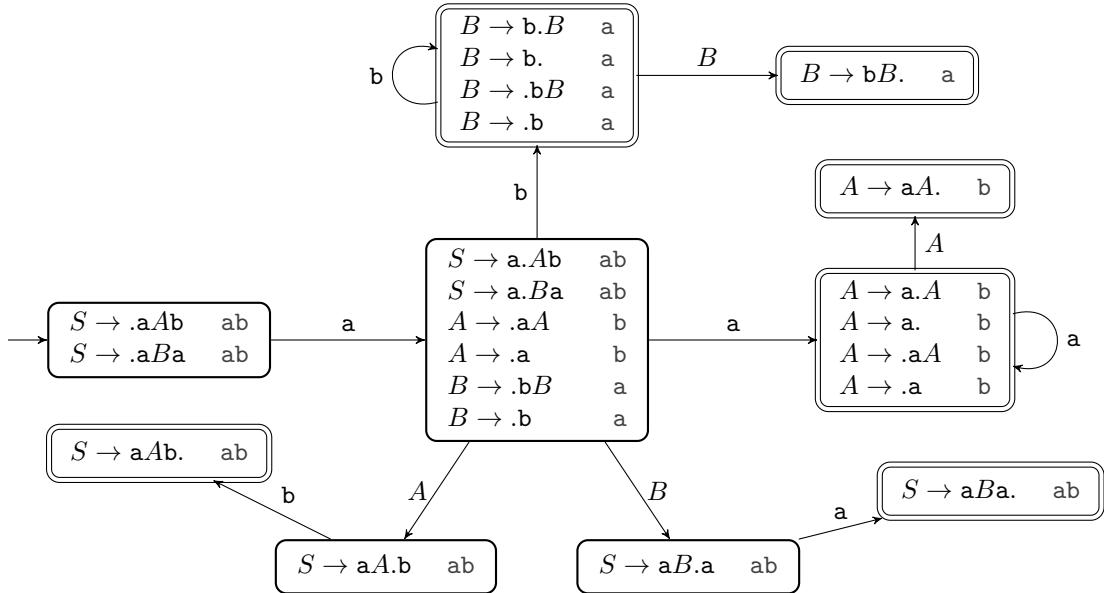


Semplificando l'espressione regolare, poiché $(bb \cup baa) = b(b \cup aa)$ e $bab \cup b \dots = b(ab \cup \dots)$, si ottiene:

$$[ab \cup (b \cup aa)(a \cup b)] \{b [ab \cup (b \cup aa)(a \cup b)]\}^*$$

Esercizio 2 [6] Si consideri la grammatica G con variabile iniziale S e regole $S \rightarrow aA b \mid aB a$, $A \rightarrow aA \mid a$, $B \rightarrow bB \mid b$. Determinare se la grammatica G è LR(1) e se G è deterministica.

Soluzione: Per determinare se la grammatica G è LR(1) costruiamo il DFA DK_1 ; questo automa, scartando i simboli di lookahead, serve anche a determinare se G è LR(0) (deterministica).



La grammatica G è LR(1), in quanto nessuno degli stati di accettazione contiene due regole tra loro consistenti. Infatti, tutti gli stati di accettazione hanno una unica regola completata; inoltre, in ogni regola non completata in cui il punto è seguito da un simbolo terminale, questo simbolo non è incluso tra i simboli di lookhead della regola completata.

D'altra parte lo stesso automa testimonia che G non è deterministica, in quanto esistono stati di accettazione contenenti regole non completate in cui il punto è seguito da un simbolo terminale.

Esercizio 3 [6] Dimostrare che se A e B sono linguaggi liberi dal contesto (CFL), allora il linguaggio $C = \{x^n y^n \mid n \geq 0, x \in A, y \in B\}$ è libero dal contesto.

Soluzione: Non è possibile esibire la dimostrazione richiesta in quanto l'asserto è falso. Infatti si consideri il seguente contro-esempio. Sia $A = \{0^h 1^h \mid h \geq 0\}$; sappiamo che A è CFL. Sia $B = \{2\}$, ossia l'insieme costituito dall'unico simbolo 2. Poiché B è finito, B è regolare e quindi anche CFL. Il linguaggio C definito come nel testo per questi A e B particolari è $C = \{(0^h 1^h)^n 2^n \mid n, h \geq 0\}$.

Se per assurdo C fosse CFL, allora dovrebbe valere per esso il pumping lemma. Sia dunque $p > 0$ la lunghezza associata a C , e sia $s = (0^p 1^p)^3 2^3 = 0^p 1^p 0^p 1^p 0^p 1^p 222$. Ovviamente $|s| = 6p + 3 > p$ e $s \in C$ ($n = 3$, $h = p$). Per il pumping lemma deve esistere una suddivisione $s = uvxyz$ con $|vxy| \leq p$, $|vy| > 0$ e $uv^i xy^i z \in C$ per ogni $i \geq 0$. Si considerino ora le seguenti due alternative. Se la suddivisione è tale che pompando non si modificano il numero di 2 nella stringa, allora per poter far parte del linguaggio la stringa pompata deve modificare tutte e tre le copie di $0^p 1^p$ in testa alla stringa. D'altra parte, la condizione $|vxy| \leq p$ consente di modificare soltanto una o due di tali copie, e quindi la stringa pompata non può far parte di C .

L’alternativa è che la suddivisione consenta di modificare il numero di 2 alla fine della stringa; in tal caso però la condizione $|vxy| \leq p$ consente di modificare solo l’ultima occorrenza di 1^p nella stringa, perciò la stringa pompata avrà la forma $0^p 1^p 0^p 1^p 0^p 1^q 2^r$, e qualunque siano i valori di q e r tale stringa non può far parte di C . Perciò il pumping lemma non vale e C non è CFL.

Esercizio 4 [7] Sia A un qualsiasi linguaggio libero dal contesto (CFL) contenente stringhe di lunghezza pari. Per ciascuna stringa $w = c_1 c_2 \cdots c_{2m} \in A$, considerare la stringa $w^\# = c_1 c_3 \cdots c_{2m-1} c_2 c_4 \cdots c_{2m}$. Dimostrare che il linguaggio $A^\# = \{w^\# \mid w \in A\}$ non è necessariamente libero dal contesto.

Soluzione: La dimostrazione consiste semplicemente nell’esibire un contro-esempio. Consideriamo il linguaggio $D = \{(\mathbf{ab})^n \mathbf{c}^{2n} \mid n \geq 0\}$. È immediato verificare che D è CFL, in quanto ad esempio omomorfo al linguaggio CFL $\{0^n 1^n \mid n \geq 0\}$, con $h(0) = \mathbf{ab}$ e $h(1) = \mathbf{cc}$.

Il linguaggio $D^\#$ è quindi $\{\mathbf{a}^n \mathbf{c}^n \mathbf{b}^n \mathbf{c}^n \mid n \geq 0\}$, che è non CFL. Supponiamo per assurdo che lo sia, dunque valga per esso il pumping lemma. Sia $p > 0$ tale che ogni stringa s di $D^\#$ di dimensione $\geq p$ è suddivisibile in modo da essere “pompata”. Consideriamo quindi $s = \mathbf{a}^p \mathbf{c}^p \mathbf{b}^p \mathbf{c}^p \in D^\#$ con $|s| = 4p > p$. Per qualunque suddivisione $s = uvxyz$ con $|vxy| \leq p$:

1. Se ciascuna stringa v e y contiene al più solo tipo di simbolo, qualunque stringa $uv^i xy^i z$ con $i \neq 0$ non può far parte di $D^\#$ in quanto due sottosequenze di simboli uguali conservano certamente lunghezza p mentre almeno un’altra sottosequenza assume lunghezza diversa da p ($|vy| > 0$)
2. Se almeno una delle stringhe v e y contiene più di un tipo di simbolo, la stringa $uv^i xy^i z$ con $i \neq 0$ non può far parte di $D^\#$ in quanto la stringa non farebbe parte di $\mathbf{a}^* \mathbf{c}^* \mathbf{b}^* \mathbf{c}^*$.

Poiché s non può essere suddivisa in modo conforme al pumping lemma, il pumping lemma non è valido, e quindi $D^\#$ non può essere CFL.

Esercizio 5 [7] Sia A un linguaggio decidibile prefissato, con $A \neq \Sigma^*$ e $A \neq \emptyset$. Dimostrare che un qualsiasi linguaggio L è decidibile se e solo se L riduce tramite Turing ad A (ossia se $L \leq_T A$).

Soluzione: Sia A un qualunque linguaggio decidibile prefissato. È richiesto di dimostrare che (1) ogni linguaggio decidibile riduce tramite Turing ad A , e che (2) se un linguaggio riduce tramite Turing ad A allora esso è decidibile.

Per la prima parte, sia L un linguaggio decidibile, e sia quindi M una TM che decide L . È possibile considerare M come una TM M^A con oracolo A che non fa alcuna domanda

all'oracolo (ovvero ignora le risposte date dall'oracolo). Pertanto banalmente M^A decide L , e quindi $L \leq_T A$.

Per la seconda parte, consideriamo un linguaggio L che riduce tramite Turing ad A , ossia $L \leq_T A$. Pertanto, esiste una TM con oracolo M^A che decide L . Poiché A è decidibile, esiste anche una TM T che decide A . Consideriamo ora la TM N ottenuta da M^A sostituendo ogni interrogazione dell'oracolo per A con la simulazione della TM T che decide A . Le risposte fornite dalla simulazione di T sono esattamente le stesse di quelle fornite dall'oracolo per A , dunque su ogni istanza x , $M^A(x)$ fornisce lo stesso risultato di $N(x)$. Possiamo dunque concludere che N decide il linguaggio L .

Esercizio 6 [8] Si consideri un grafo non diretto $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Si dimostri che il problema di stabilire se il grafo G contiene un sottoinsieme W di nodi con $|W| \leq \frac{n}{2}$ tale che ogni arco di G contiene almeno un nodo in W è NP-completo.

Soluzione: Questo problema è generalmente noto con il nome HALF VERTEX COVER (HVC). Dimostrare che HVC è in NP è molto semplice. Data una istanza $\langle G \rangle$, un certificato per l'esistenza di una soluzione è una lista di nodi del grafo G . Il verificatore controlla che nella lista non vi siano nodi ripetuti, che la lista contenga non più di $|V(G)|/2$ nodi, e che ciascun arco di G sia collegato ad un nodo della lista. Tutti questi controlli possono naturalmente essere effettuati in tempo polinomiale nella dimensione del grafo.

Esibiamo ora una riduzione tra INDEPENDENT SET (IS) e HVC. Si presti attenzione che le istanze di IS e di HVC hanno una differente struttura: le istanze di IS codificano un grafo G ed un numero intero k , mentre le istanze di HVC codificano soltanto un grafo. Dunque la riduzione deve “eliminare” il parametro k dall'istanza di IS.

La riduzione considera il valore del parametro k ed opera differentemente nei casi $k = n/2$, $k < n/2$, e $k > n/2$. Si ricordi che una riduzione è una funzione calcolabile in tempo polinomiale, e confrontare due valori interi è una operazione eseguibile in tempo polinomiale nella lunghezza delle codifiche dei valori.

1. $k = n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G \rangle$ (si elimina semplicemente il valore k dall'istanza). Infatti, una istanza-sì di IS è tale per cui esiste un sottoinsieme $W \subseteq V$ con $|W| \geq k = n/2$ tale che non esiste alcun arco tra i nodi in W ; l'insieme $U = V \setminus W$ è un ricoprimento vertici con $|U| \leq n - k = n/2$; il grafo è dunque anche una istanza-sì di HVC. Ovviamente vale anche il viceversa.
2. $k < n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $n - 2k$ nodi isolati. Sia $\langle G, k \rangle$ una istanza-sì di IS, dunque esiste un insieme indipendente $W \subseteq V$ con $|W| \geq k$. Nel grafo G' , W unito ai nuovi nodi costituisce un insieme indipendente W' di dimensione $k + (n - 2k) = n - k$ nodi. Pertanto $U = V(G') \setminus W' = V \setminus W$ ha $|U| = n - k$ nodi ed è un

ricoprimento tramite vertici per G' . D'altra parte, G' contiene $n + (n - 2k) = 2(n - k)$ nodi, quindi G' è una istanza-sì di HVC.

Se invece $\langle G, k \rangle$ è una istanza-no di IS, non esiste alcun insieme indipendente di dimensione maggiore o uguale a k in G . Nel grafo G' pertanto non esiste alcun insieme indipendente di dimensione maggiore o uguale a $n - k$, perché G' è ottenuto da G aggiungendo solo $n - 2k$ nodi e non rimuovendo alcun arco. Dunque in G' nessun ricoprimento tramite vertici può essere di dimensione minore o uguale a $2(n - k) - (n - k) = n - k$, e quindi $\langle G' \rangle$ è una istanza-no di HVC.

3. $k > n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $2k - n$ nodi ed i seguenti archi: tutti i nuovi nodi sono collegati tra loro (quindi costituiscono un insieme completo), inoltre ciascuno dei nuovi nodi è collegato a ciascuno dei nodi del grafo G . Ovviamente, una istanza-sì di IS è anche una istanza-sì di G' . Infatti G' ha $n + (2k - n) = 2k$ nodi, e l'insieme indipendente W con $|W| \geq k$ è anche un insieme indipendente di G' con almeno la metà dei nodi di G' ; pertanto il complemento di questo insieme è un ricoprimento tramite vertici con al più la metà dei nodi di G' .

Consideriamo ora una istanza-no di IS, dunque supponiamo che nel grafo G non esista alcun insieme indipendente di dimensione maggiore o uguale a k . Il grafo G' è ottenuto da G aggiungendo un sottografo completo ed aggiungendo tutti gli archi tra i vecchi ed i nuovi nodi. Pertanto, nessuno dei nuovi nodi può essere aggiunto ad un insieme indipendente di G . Si conclude dunque che non può esistere in G' un insieme indipendente avente la metà dei nodi di G' . Pertanto, ogni ricoprimento tramite vertici in G' deve contenere più della metà dei nodi di G' , dunque $\langle G' \rangle$ è una istanza-no di HVC.

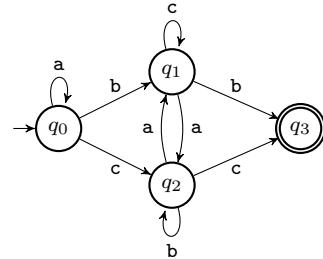
È facile verificare che la dimensione dell'istanza $\langle G' \rangle$ è polinomialmente limitata dalla dimensione dell'istanza $\langle G, k \rangle$ (infatti, $k \leq n$, quindi in ogni caso $|V(G')| \leq 2n$). Inoltre la riduzione è calcolabile in tempo polinomiale. Quindi HVC è NP-hard, e di conseguenza NP-completo.

Automi e Linguaggi (M. Cesati)

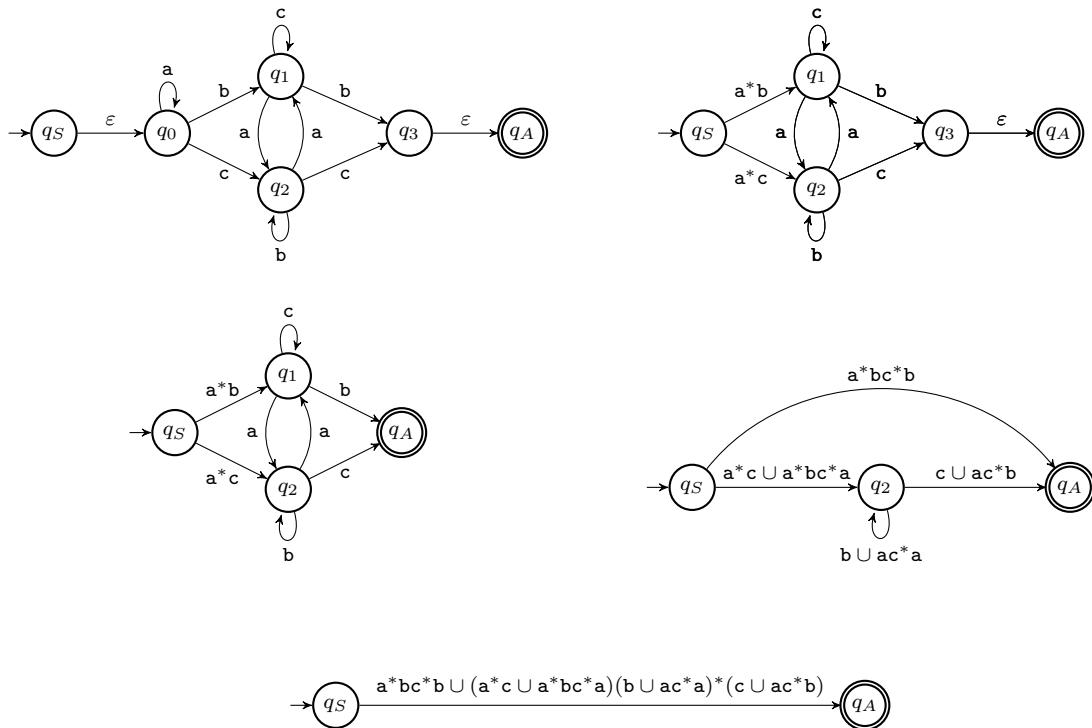
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 26 agosto 2022

Esercizio 1 [6] Determinare una espressione regolare per il linguaggio riconosciuto dal DFA:



Soluzione: Convertiamo il DFA in un GNFA e rimuoviamo nell'ordine i nodi q_0 , q_3 , q_1 e q_2 . Si ottiene:



In conclusione, una espressione regolare è $a^*[bc^*b \cup (c \cup bc^*a)(b \cup ac^*a)^*(c \cup ac^*b)]$,

Esercizio 2 [6] Si consideri il linguaggio $A = \{w z \bar{w}^R \mid w \in \{0,1\}^*, z = 0^n 1^n, n \geq 0\}$, ove \bar{w}^R è la stringa ottenuta complementando i bit in w ed invertendone l'ordine. Il linguaggio A è regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A non è regolare. Tuttavia questa dimostrazione non può essere basata sul fatto che A contiene, come sottoinsieme, un linguaggio non regolare (se bastasse questo, potremmo anche dimostrare che $\{0,1\}^*$ è non regolare!). Invece, assumiamo per assurdo che l'intero linguaggio A sia regolare, e dunque che per esso valga il pumping lemma

con lunghezza p . Consideriamo la stringa $s = 0^p 1^p$, che fa certamente parte di A considerando, ad esempio, $w = \varepsilon$ e $n = p$. Il pumping lemma afferma che esiste una suddivisione $s = xyz$ con $|xy| \leq p$, $|y| > 0$ e $xy^i z \in A$ per ogni $i \geq 0$. Consideriamo quindi il caso $i = 2$, e quindi la stringa $xyyz$. Poiché $|xy| \leq p$, y contiene solo zeri, quindi $xyyz = 0^q 1^p$ con $q > p$. Tuttavia, il linguaggio A non contiene alcuna stringa della forma $0^q 1^p$ con $q > p$.

Infatti, supponiamo per assurdo che $0^q 1^p \in A$. Quindi $0^q 1^p = w z \bar{w}^R$. Se $w = \varepsilon$, allora $0^q 1^p = z = 0^n 1^n$, per qualche n , quindi varrebbe $q = p$, contraddicendo che $q > p$. Se invece $w \neq \varepsilon$, allora

$$0^q 1^p = \underbrace{0^l 1^m}_{w} \underbrace{0^n 1^n}_{z} \underbrace{0^m 1^l}_{\bar{w}^R}$$

con $m+l > 0$ e $n \geq 0$. Quindi necessariamente, per evitare l'occorrenza di un 1 seguito da uno 0, $l > 0$ e $m = 0$. Perciò $0^q 1^p = 0^{l+n} 1^{l+n}$, e quindi $q = p$, contraddicendo la condizione $q > p$. La contraddizione deriva dall'aver supposto che $0^q 1^p \in A$ con $q > p$. Un altro ragionamento che porta alla stessa conclusione si basa sull'osservazione che il numero di bit 0 ed il numero di bit 1 in ogni stringa appartenente ad A deve coincidere: infatti per definizione nella porzione z della stringa vi sono lo stesso numero di 0 e 1, e lo stesso vale nella stringa $w \bar{w}^R$, in cui ad ogni bit in w corrisponde un bit complementato in \bar{w}^R . Pertanto, $0^q 1^p \notin A$ se $q \neq p$.

Poiché la stringa pompata non può appartenere ad A , il pumping lemma non vale, e dunque A non può essere regolare.

Esercizio 3 [6] Si consideri la grammatica G con variabile iniziale S :

$$S \rightarrow A \quad A \rightarrow aAb \mid bAa \mid aBb \quad B \rightarrow aBb \mid \varepsilon.$$

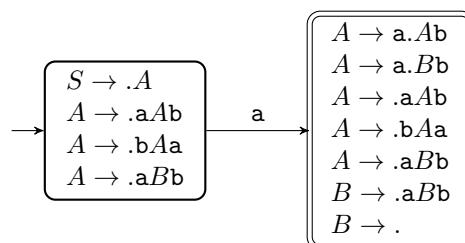
La grammatica G è deterministica? Giustificare la risposta con una dimostrazione.

Soluzione: La grammatica G non può essere deterministica in quanto essa è ambigua. È sufficiente infatti considerare le seguenti due differenti derivazioni “a sinistra” della stringa aabb:

$$S \Rightarrow A \Rightarrow aAb \Rightarrow aaBbb \Rightarrow aa\varepsilon bb = aabb$$

$$S \Rightarrow A \Rightarrow aBb \Rightarrow aaBbb \Rightarrow aa\varepsilon bb = aabb$$

Come dimostrazione alternativa è sufficiente esibire due stati dell'automa DK:



Poiché lo stato di accettazione contiene una regola in cui il punto è seguito da un simbolo terminale, G non è deterministica.

Esercizio 4 [6] Sia $B = \{ \langle M \rangle \mid M \text{ è una macchina di Turing tale che per ogni stringa } x \text{ accettata da } M, M \text{ accetta anche la stringa rovesciata } x^R \}$. Il linguaggio B è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Poiché il linguaggio è costituito da codifiche di macchine di Turing, è plausibile che ad esso possa applicarsi il teorema di Rice, che afferma che qualunque proprietà non banale relativa ai linguaggi riconosciuti da TM è indecidibile. Per verificare se le ipotesi del teorema di Rice sono soddisfatte, consideriamo se la proprietà caratterizzante l'insieme B è banale o meno, ovvero se B è diverso dall'insieme vuoto e dall'insieme di tutte le codifiche delle TM. Innanzi tutto, sia M' una TM che accetta esclusivamente la stringa 00 e rifiuta tutte le altre stringhe; poiché $(00)^R = 00$, $\langle M' \rangle \in B$, dunque $B \neq \emptyset$. D'altra parte, sia M'' una TM che accetta esclusivamente la stringa 01 e rifiuta tutte le altre stringhe: ovviamente $(01)^R = 10 \notin L(M'')$, dunque $\langle M'' \rangle \notin B$.

La seconda ipotesi del teorema di Rice è che la proprietà caratterizzante B deve riferirsi al linguaggio riconosciuto dalle macchine di Turing, e non alle TM stesse. Ciò è evidente dalla definizione stessa di B : se $\langle M \rangle \in B$ e $L(N) = L(M)$, allora necessariamente N deve accettare il rovescio di qualunque stringa $x \in L(N)$ perché $x^R \in L(M) = L(N)$; dunque $\langle N \rangle \in B$.

Avendo quindi verificato le ipotesi del teorema di Rice, possiamo concludere direttamente con il suo asserto, ossia che il linguaggio B non è decidibile.

È possibile anche dimostrare che B è non decidibile tramite una riduzione diretta da un altro problema non decidibile, quale ad esempio \mathcal{A}_{TM} ($\mathcal{A}_{\text{TM}} \leq_m B$). Si deve prestare attenzione però a non confondere le istanze dei due problemi. Ad esempio, una TM T che rifiuta ogni input (ossia tale che $L(T) = \emptyset$) in effetti appartiene a B , perché se T accetta una stringa accetta anche il suo rovescio (banalmente è vero perché T non accetta alcuna stringa); quindi $\langle T \rangle \in B$. D'altra parte, $\langle T, x \rangle \notin \mathcal{A}_{\text{TM}}$ per ogni possibile input x .

Supponiamo quindi che esista, per assurdo, un decisore D per il linguaggio B , e consideriamo la seguente TM:

P= “On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. From $\langle M, w \rangle$, build the encoding of the following DTM R :

R=“On any input x :

- a. If $x = 01$, then accepts
- b. Run $M(w)$
- c. If $M(w)$ accepts, then accept
- d. If $M(w)$ rejects, then reject”

2. Run D on input $\langle R \rangle$
3. If $D(\langle R \rangle)$ accepts, then accept, else reject"

Supponiamo che $M(w)$ accetti; dunque $R(x)$ accetta sia se $x = 01$ sia se $x \neq 01$; perciò R accetta ogni stringa, ossia $L(R) = \Sigma^*$. Di conseguenza, $\langle R \rangle \in B$. Se invece $M(w)$ non accetta (sia perché rifiuta oppure perché non si ferma), allora $R(x)$ accetta soltanto se $x = 01$; dunque $L(R) = \{01\}$, e $\langle R \rangle \notin B$, in quanto $10 \notin L(R)$. Il decisore D può determinare se $\langle R \rangle \in B$, e di conseguenza P può decidere la generica istanza $\langle M, w \rangle$ di \mathcal{A}_{TM} . Questa è ovviamente una contraddizione perché \mathcal{A}_{TM} non è decidibile.

Esercizio 5 [7] Siano A e B linguaggi Turing-riconoscibili (ossia ricorsivamente enumerabili). La differenza simmetrica $A \Delta B$ di A e B (gli elementi che stanno in A o in B ma non in entrambi) è necessariamente Turing-riconoscibile? Giustificare la risposta con una dimostrazione.

Soluzione: $A \Delta B$ non è necessariamente Turing-riconoscibile; per dimostrarlo è sufficiente esibire un contro-esempio. Sia dunque $A = \mathcal{A}_{\text{TM}}$, il linguaggio contenente le codifiche delle macchine di Turing e delle stringhe da esse accettate. Sia inoltre $B = \Sigma^*$, ove Σ è l'alfabeto sul quale sono costruite le codifiche delle TM in \mathcal{A}_{TM} . Si dimostra facilmente che $A \Delta B = (A \setminus B) \cup (B \setminus A)$; inoltre nel nostro caso $\mathcal{A}_{\text{TM}} \setminus \Sigma^* = \emptyset$ e $\Sigma^* \setminus \mathcal{A}_{\text{TM}} = \mathcal{A}_{\text{TM}}^c$. Perciò $\mathcal{A}_{\text{TM}} \Delta \Sigma^* = \emptyset \cup \mathcal{A}_{\text{TM}}^c = \mathcal{A}_{\text{TM}}^c$. Sappiamo che Σ^* è regolare e quindi Turing-riconoscibile; anche \mathcal{A}_{TM} è Turing-riconoscibile, ma non decidibile. Perciò $\Sigma^* \Delta \mathcal{A}_{\text{TM}} = \mathcal{A}_{\text{TM}}^c$ non può essere Turing-riconoscibile; se infatti lo fosse, poiché sia \mathcal{A}_{TM} che $\mathcal{A}_{\text{TM}}^c$ sarebbero Turing-riconoscibili, allora sarebbero anche entrambi decidibili, il che è manifestamente falso.

Esercizio 6 [9] Si consideri un problema in cui l'istanza è costituita da $2m$ numeri interi non negativi (non necessariamente distinti tra loro) la cui somma è pari ($2s$). Il problema richiede di decidere se è possibile suddividere i numeri in due sottoinsiemi ciascuno con m elementi e tali che la somma degli elementi in ciascun sottoinsieme sia uguale a s . Dimostrare che tale problema è NP-completo.

Soluzione: Questo problema è conosciuto col nome di **BALANCED PARTITION**. È un problema polinomialmente verificabile: infatti, sia U un multi-insieme di numeri interi non negativi con somma $2s$ che ammette una partizione in due multi-insiemi I e J ($I \cup J = U$, $I \cap J = \emptyset$) tale che $\sum_{x \in I} x = \sum_{x \in J} x = s$. Un certificato per tale istanza-sì di **BALANCED PARTITION** è costituito semplicemente da uno dei due sottoinsiemi. Esiste dunque un verificatore che opera in tempo polinomiale nella dimensione dell'istanza:

V= “On input $\langle U, I \rangle$, where U is a multi-set of non-negative integers:

1. Verify that $I \subset U$ and $2|I| = |U|$

2. Compute $J = U \setminus I$
3. Compute $v = \sum_{x \in I} x$
4. Compute $w = \sum_{x \in J} x$
5. Accept if $v = w$, reject otherwise"

Pertanto, BALANCED PARTITION è incluso in NP. Per dimostrare che è anche NP-hard, possiamo esibire una riduzione polinomiale da un altro problema NP-hard, ad esempio SUBSET SUM.

Sia dunque (S, t) una istanza di SUBSET SUM, ossia un multi-insieme di numeri interi S ed un intero t . Dalla dimostrazione di NP-hardness fatta a lezione è evidente che questo problema è NP-hard anche restringendo le istanze agli interi non negativi. Consideriamo la riduzione polinomiale che trasforma (S, t) nel multi-insieme U come segue. Sia $n = |S|$ e sia $\mu = \sum_{x \in S} x$. Se $\mu < t$, allora (S, t) è certamente una istanza-no, dunque la riduzione polinomiale si limita a costruire una istanza-no elementare di BALANCED PARTITION. Altimenti, il multi-insieme U è costituito da S , dall'intero non negativo $\lambda = 2t - \mu$, e da $n + 1$ valori interi nulli (ossia n zeri $z_0 = \dots = z_n = 0$). Ovviamente $|U| = |S| + 1 + n + 1 = 2(n + 1)$.

Supponiamo che (S, t) sia una istanza-sì di SUBSET SUM, e dunque che esista $T \subseteq S$ tale che $\sum_{x \in T} x = t$. Sia $q = |T|$, e consideriamo il multi-insieme $W = T \cup \{z_0, \dots, z_{n-q}\}$ (si osservi che se $q = n$ allora W include il solo elemento z_0). Quindi $|W| = n + 1$. Inoltre il sottoinsieme $Z = U \setminus W$ è costituito da $n - q$ elementi di $S \setminus T$, da λ , e da q zeri $\{z_{n-q+1}, \dots, z_n\}$ (ovviamente $|Z| = (n - q) + 1 + (n - n + q - 1 + 1) = n + 1$). Si ha:

$$\sum_{x \in W} x = \sum_{x \in T} x + z_0 + \dots + z_{n-q} = t$$

e

$$\sum_{x \in Z} x = \sum_{x \in S \setminus T} x + \lambda + z_{n-q+1} + \dots + z_n = (\mu - t) + (2t - \mu) = t.$$

Pertanto U è una istanza-sì di BALANCED PARTITION.

Supponiamo al contrario che U sia una istanza-sì di BALANCED PARTITION derivata dalla riduzione di una istanza (S, t) , e siano W e Z tali che $W \cap Z = \emptyset$, $|W| = |Z| = n + 1$, e $\sum_{x \in W} x = \sum_{x \in Z} x = t$. Senza perdita di generalità supponiamo che $\lambda \notin W$, e sia $T = W \setminus \{z_0, \dots, z_n\}$, ossia T è il multi-insieme W a cui sono stati rimossi gli zeri z_i eventualmente presenti. Pertanto $T \subseteq S$, ed inoltre $\sum_{x \in T} x = t$. Perciò (S, t) è una istanza-sì di SUBSET SUM.

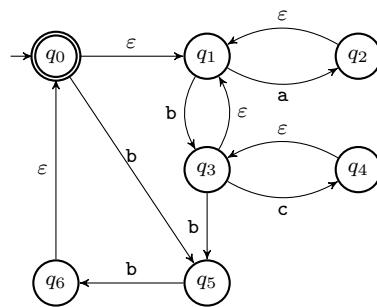
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 13 settembre 2022

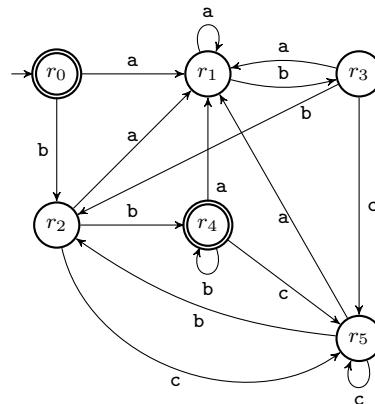
Esercizio 1 [6] Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((a^*bc^*)^*bb)^*$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione allo NFA derivato dalla espressione regolare si ottiene:



Un automa deterministico equivalente è il seguente:

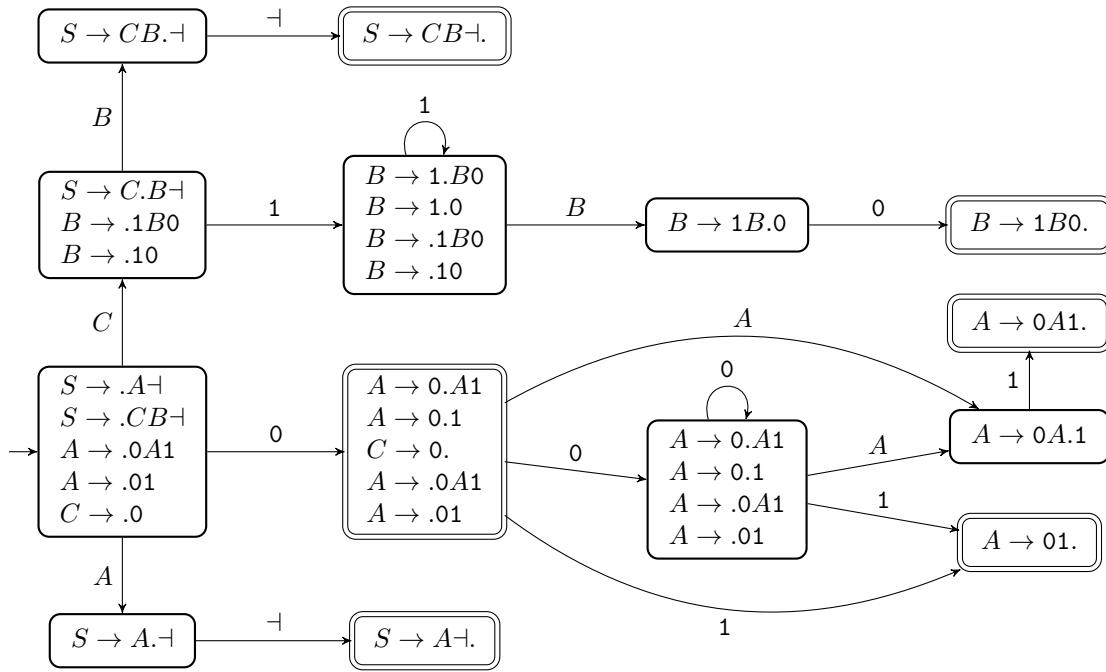
$$\begin{aligned}
 r_0 &= \{q_0, q_1\} \\
 r_1 &= \{q_1, q_2\} \\
 r_2 &= \{q_1, q_3, q_5\} \\
 r_3 &= \{q_1, q_3\} \\
 r_4 &= \{q_0, q_1, q_3, q_5, q_6\} \\
 r_5 &= \{q_1, q_3, q_4\}
 \end{aligned}$$



Esercizio 2 [6] Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{aligned}
 S &\rightarrow A\vdash \quad | \quad CB\vdash \\
 A &\rightarrow 0A1 \quad | \quad 01 \\
 B &\rightarrow 1B0 \quad | \quad 10 \\
 C &\rightarrow 0
 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:



Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo ‘0’ contiene una regola completata e diverse regole in cui il punto precede un simbolo terminale. Perciò il DK-test è fallito, e di conseguenza la grammatica non è DCFG.

Esercizio 3 [7] Sia $A = \{u^R \# v \mid u, v \in \{0, 1\}^*\}$, u è la codifica binaria di un numero $n \geq 1$, e v è la codifica binaria del numero $n + 1$. Le codifiche binarie non hanno zeri non significativi a sinistra. Si noti che il linguaggio codifica u invertendo l’ordine dei suoi bit. Ad esempio fanno parte di A le stringhe $01\#11$ e $11\#100$, mentre non fanno parte di A le stringhe $10\#10$ (perché $10^R = 01$ ha uno zero non significativo) e $1\#11$. Dimostrare che A è un linguaggio libero dal contesto (CFL).

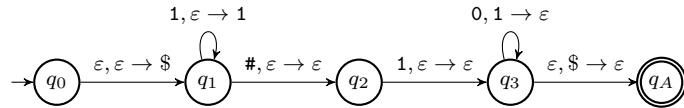
Soluzione:

Una semplice dimostrazione che A è CFL può essere ottenuta osservando che A è in effetti l’unione di due linguaggi:

$$A = B \cup C, B = \{1^n \# 10^n \mid n \geq 1\}, C = \{u^R \# v \mid u, v \in \{0, 1\}^*, |u| = |v|, (u)_2 + 1 = (v)_2\},$$

ove $(x)_2$ rappresenta il numero codificato in binario dalla stringa x . In altri termini, B rappresenta le istanze in cui è presente un riporto nella addizione $+1$, mentre C rappresenta le istanze di A in cui non è presente un riporto, e dunque tali che le stringhe u e v hanno la stessa lunghezza.

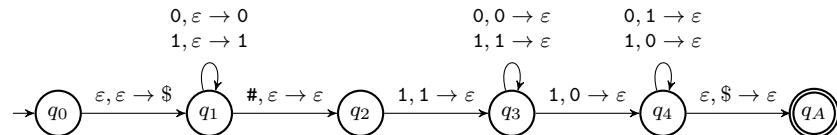
Per dimostrare che A è CFL è sufficiente dimostrare che sia B che C sono CFL (i linguaggi liberi dal contesto sono infatti chiusi rispetto all’operazione di unione). Il linguaggio B è una semplice variante del linguaggio $a^n b^n$, ed un PDA che lo riconosce è ad esempio:



Poiché le codifiche dei numeri nelle istanze-sì di A non devono contenere zeri non significativi, il linguaggio C può essere descritto come:

$$C = \{x 0 w 1 \# 1 y 1 z \mid w, x, y, z \in \{0, 1\}^*, w^R = y, x^R = z\}$$

Un esempio di PDA che riconosce C è:



(Si osservi che la definizione di A richiede che il numero n codificato da u sia maggiore di zero. Se fosse ammesso anche il caso $n = 0$ allora bisognerebbe aggiungere a C la stringa $0\#1$. Poiché un linguaggio costituito da una sola stringa è regolare, C continuerebbe comunque ad essere CFL.)

Esercizio 4 [6] Si consideri il linguaggio $A = \{x \in \{0, 1\}^* \mid x \text{ ha un numero di bit 1 uguale al numero di bit 0}\}$. Sia $B = \{\langle M \rangle \mid M \text{ è una TM e } L(M) \subseteq A\}$. Il linguaggio B è decidibile oppure no? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio B non è decidibile, e per dimostrarlo è sufficiente verificare che le ipotesi del Teorema di Rice sono verificate. Si consideri come proprietà P del linguaggio della TM l'essere un sottinsieme del linguaggio A . Tale proprietà è non banale: infatti la TM che accetta tutte le stringhe non soddisfa la proprietà $(\Sigma^* \not\subseteq A)$, mentre la TM che rifiuta tutte le stringhe soddisfa la proprietà $(\emptyset \subseteq A)$. Inoltre, P è una proprietà del linguaggio riconosciuto dalla TM: infatti se due diverse TM riconoscono lo stesso linguaggio, per entrambe vale che il linguaggio è un sottinsieme di A , e dunque entrambe le TM soddisfano la priorità P . Poiché tutte le ipotesi del Teorema di Rice sono soddisfatte possiamo concludere immediatamente che il linguaggio B contenente codifiche di TM che soddisfano la proprietà P è indecidibile.

Esercizio 5 [7] Siano A e B linguaggi Turing-riconoscibili (ossia ricorsivamente enumerabili). La differenza simmetrica $A \Delta B$ di A e B (gli elementi che stanno in A o in B ma non in entrambi) è necessariamente Turing-riconoscibile? Giustificare la risposta con una dimostrazione.

Soluzione: $A \Delta B$ non è necessariamente Turing-riconoscibile; per dimostrarlo è sufficiente esibire un contro-esempio. Sia dunque $A = \mathcal{A}_{\text{TM}}$, il linguaggio contenente le codifiche delle macchine di Turing e delle stringhe da esse accettate. Sia inoltre $B = \Sigma^*$, ove Σ è l'alfabeto sul quale sono costruite le istanze in \mathcal{A}_{TM} . Si dimostra facilmente che $A \Delta B = (A \setminus B) \cup (B \setminus A)$; inoltre nel nostro caso $\mathcal{A}_{\text{TM}} \setminus \Sigma^* = \emptyset$ e $\Sigma^* \setminus \mathcal{A}_{\text{TM}} = \mathcal{A}_{\text{TM}}^c$. Perciò $\mathcal{A}_{\text{TM}} \Delta \Sigma^* = \emptyset \cup \mathcal{A}_{\text{TM}}^c = \mathcal{A}_{\text{TM}}^c$.

Sappiamo che Σ^* è regolare e quindi Turing-riconoscibile; anche \mathcal{A}_{TM} è Turing-riconoscibile, ma non decidibile. Perciò $\Sigma^* \Delta \mathcal{A}_{\text{TM}} = \mathcal{A}_{\text{TM}}^c$ non può essere Turing-riconoscibile; se infatti lo fosse, poiché sia \mathcal{A}_{TM} che $\mathcal{A}_{\text{TM}}^c$ sarebbero Turing-riconoscibili, allora sarebbero anche entrambi decidibili, il che è manifestamente falso.

Esercizio 6 [8] Si consideri il problema FEEDBACK VERTEX SET: dato un grafo diretto $G = (V, A)$ ed un numero $k \in \mathbb{N}$, esiste un sottoinsieme $V' \subseteq V$ con $|V'| \leq k$ tale che ogni *ciclo* diretto entro G include almeno un nodo in V' ? Dimostrare che il problema è NP-completo.

Soluzione: Il problema FEEDBACK VERTEX SET (o FVS) è verificabile polinomialmente: un certificato è banalmente la lista di nodi che costituisce l'insieme V' . Si consideri infatti il seguente algoritmo.

M= “On input $\langle G, k, V' \rangle$:

1. Verify that V' is a subset of k or less nodes of G
2. Build the graph $G' = G \setminus V'$
3. For every pairs of nodes s, t in G' :
 4. Run $\text{PATH}(G', s, t)$ to determine if there is a path from s to t
 5. If the path exists:
 6. Let I be the nodes of the path except s and t
 7. Build the graph $G'' = G \setminus I$
 8. If the path exists, reject (cycle found)
 9. There is no cycle in G' , hence accept”

La notazione $G \setminus V$ indica il grafo ottenuto da G rimuovendo tutti i nodi dell'insieme V e tutti gli archi incidenti su questi nodi. Poiché l'algoritmo PATH è polinomiale, il verificatore esegue in tempo polinomiale nel numero di nodi del grafo G in istanza.

Per dimostrare che FVS è NP-hard possiamo considerare una semplicissima riduzione dal problema NP-completo VERTEX COVER. Sia infatti (G, k) una istanza del problema VC, ove G è un grafo non diretto e $k \in \mathbb{N}$. Consideriamo il grafo diretto G' ottenuto sostituendo ad ogni arco non diretto di G una coppia di archi in direzione opposta tra gli stessi nodi in G' . Sia dunque (G', k) l'istanza ridotta di FVS, che ha ovviamente dimensione polinomiale ed è meccanicamente costruibile.

Supponiamo che (G, k) sia una istanza-sì di VC. Dunque esiste un sottoinsieme di al più k nodi che copre ogni arco di VC. Lo stesso sottoinsieme di nodi in G' copre ogni arco diretto, quindi la rimozione dei nodi di V' rende G' un grafo senza archi, e quindi senza cicli. Perciò (G', k) è una istanza-sì di FVS. Viceversa, supponiamo che (G', k) sia una istanza-sì di FVS, pertanto esiste un sottoinsieme di al più k nodi che copre ogni ciclo del grafo G' . Si consideri ora che ciascun arco del grafo G ha dato origine ad un ciclo di dimensione 2 in G' , e anche tali cicli devono essere coperti da V' . Pertanto l'insieme di nodi V' copre ogni arco del grafo G , e dunque (G, k) è una istanza-sì di VC.

Concludendo, $\text{VC} \leq_p \text{FVS}$, pertanto FVS è NP-hard, e dunque NP-completo.

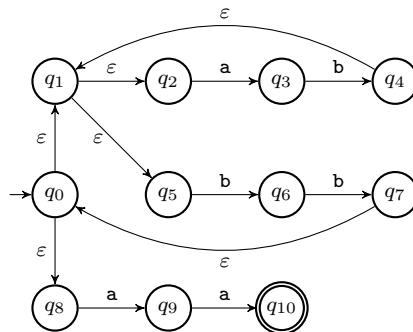
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 24 gennaio 2023

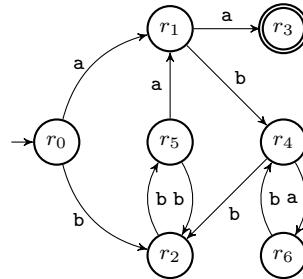
Esercizio 1 [6] Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $((ab)^*bb)^*aa$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando poche semplificazioni allo NFA derivato meccanicamente dalla espressione regolare si ottiene:

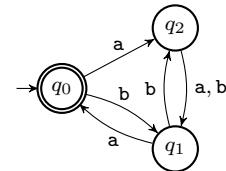


Un automa deterministico equivalente è il seguente:

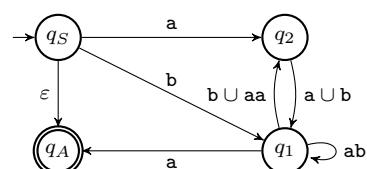
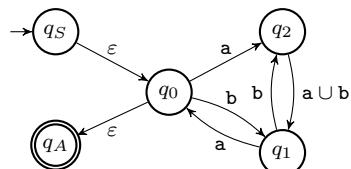
$$\begin{aligned} r_0 &= \{q_0, q_1, q_2, q_5, q_8\} \\ r_1 &= \{q_3, q_9\} \\ r_2 &= \{q_6\} \\ r_3 &= \{q_{10}\} \\ r_4 &= \{q_1, q_2, q_4, q_5\} \\ r_5 &= \{q_0, q_1, q_2, q_5, q_7, q_8\} \\ r_6 &= \{q_3\} \end{aligned}$$

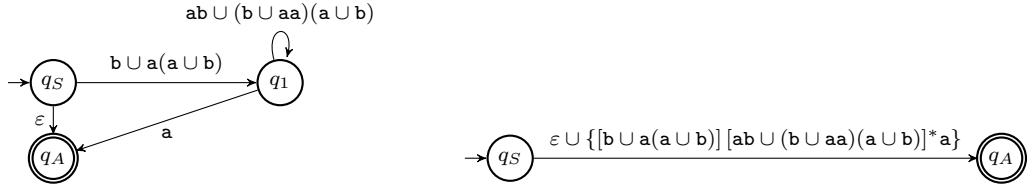


Esercizio 2 [6] Determinare una espressione regolare per il linguaggio riconosciuto dal seguente automa deterministico:



Soluzione: Trasformiamo l'automa in un GNFA aggiungendo gli stati q_S e q_A , e rimuoviamo nell'ordine i nodi q_0 , q_2 e q_1 :





Una espressione regolare che genera il linguaggio riconosciuto dall'automa è quindi:

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

Cambiando l'ordine di eliminazione dei nodi si ottengono espressioni regolari diverse ma equivalenti. Ad esempio:

$$\begin{array}{ll} \{[b \cup a(a \cup b)][b(a \cup b)]^*a\}^* & (\text{ordine: } q_2, q_1, q_0) \\ \{ba \cup (a \cup bb)[(a \cup b)b]^*(a \cup b)a\}^* & (\text{ordine: } q_1, q_2, q_0) \end{array}$$

Esercizio 3 [6.5] Si consideri il linguaggio $A = \{w \in \{0,1\}^+ \mid w = w_1w_2, |w_1| = |w_2|\}$, e il numero di zero in w_1 è uguale al numero di uno in $w_2\}$. Ad esempio, $01 \in A$, $011 \notin A$, $0110 \in A$, $1100 \in A$, $0010 \notin A$, $0011 \in A$. Il linguaggio A è regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A non è regolare; per dimostrarlo, supponiamo per assurdo che lo sia, e che quindi valga per esso il pumping lemma con un opportuno valore $p > 0$.

Consideriamo la stringa $s = 0^p101^p \in A$, di lunghezza $2p + 2$. Il pumping lemma afferma che esiste una suddivisione $s = xyz$ con $|xy| \leq p$ e $|y| > 0$ tale che $xy^i z \in A$ per qualsiasi $i \geq 0$. Si osservi ora che A non contiene la stringa nulla e non contiene stringhe di lunghezza dispari. Pertanto la lunghezza $|y|$ deve essere un numero pari, altrimenti la lunghezza $|xz|$ sarebbe dispari, e quindi $xy^0 z$ non potrebbe far parte di A . Quindi possiamo scrivere $|y| = 2k$ con $k > 0$. Di conseguenza, $|xy^i z| = 2p + 2 + 2k(i - 1)$.

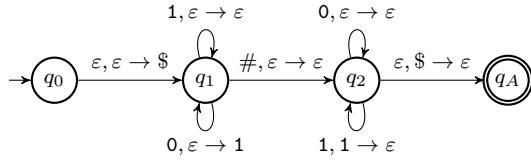
Consideriamo ora il valore particolare $i = 0$: la corrispondente stringa $xy^0 z$ ha lunghezza $|xz| = 2p + 2 - 2k$. Se $xz = w_1w_2$, con $|w_1| = |w_2|$, allora $|w_1| = |w_2| = p + 1 - k$. Perciò necessariamente $w_2 = 1^{p+1-k}$, in quanto $k \geq 1$. D'altra parte, w_1 contiene esattamente $p - 2k + 1$ zero, e dunque dovremmo imporre

$$p - 2k + 1 = p + 1 - k,$$

che implica $k = 0$. Ma ciò è impossibile in quanto $|y| = 2k > 0$. La contraddizione deriva dall'aver supposto che per A valga il pumping lemma, e dunque A non può essere regolare.

Esercizio 4 [6.5] Si consideri il linguaggio $B = \{w_1 \# w_2 \mid w_1, w_2 \in \{0,1\}^*\}$, ed il numero di zero in w_1 è uguale al numero di uno in $w_2\}$. Si osservi che w_1 e w_2 possono avere differente lunghezza. Ad esempio, $0\#1 \in B$, $0\#11 \notin B$, $11\#00 \in B$, $00\#10 \notin B$, $0\#011 \notin B$. Il linguaggio B è libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Per dimostrare che B è CFL è sufficiente esibire un PDA che riconosca tutti e soli gli elementi di B :



Solo la lettura del simbolo ‘#’ fa transitare l’automa dallo stato q_1 allo stato q_2 . Lo stato q_1 memorizza sullo stack un simbolo ‘1’ per ogni simbolo ‘0’ letto in ingresso, e corrispondentemente lo stato q_2 rimuove un simbolo dallo stack per ogni simbolo ‘1’ letto in ingresso. È possibile transitare da q_2 allo stato di accettazione q_A solo se lo stack è vuoto, e l’automa accetterà la stringa solo se si entra in q_A avendo letto tutti i caratteri della stringa in ingresso.

Esercizio 5 [6] Sia $C = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) \text{ è decidibile}\}$. C è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio C non è decidibile, e per dimostrarlo è sufficiente verificare che le ipotesi del Teorema di Rice sono verificate. Si consideri come proprietà P della TM il riconoscere un linguaggio decidibile. Tale proprietà è non banale: infatti una TM M_1 che accetta tutte le stringhe riconosce Σ^* , che è decidibile; d’altra parte, sappiamo che \mathcal{A}_{TM} è ricorsivamente enumerabile ma non decidibile; dunque una TM M_0 che riconosce tale linguaggio non soddisfa la proprietà P . Inoltre, P è in effetti una proprietà del linguaggio riconosciuto dalla TM: infatti se due diverse TM riconoscono lo stesso linguaggio, per entrambe vale che il linguaggio è decidibile oppure no, dunque entrambe le TM soddisfano la priorità P oppure no. Poiché tutte le ipotesi del Teorema di Rice sono soddisfatte possiamo concludere immediatamente che il linguaggio C contenente codifiche di TM che soddisfano la proprietà P è indecidibile.

Esercizio 6 [9] Si consideri il linguaggio costituito dalle codifiche delle formule booleane che hanno almeno due assegnazioni di verità che soddisfano la formula. Dimostrare che tale linguaggio è NP-completo.

Soluzione: Questo problema è generalmente chiamato DOUBLE SAT, ed è una generalizzazione molto semplice del problema SAT.

Si dimostra facilmente che DOUBLE SAT è in NP. Infatti, data una qualunque istanza $\langle \Phi \rangle$ che codifica una formula booleana, un certificato per l’esistenza di una soluzione è costituito da due liste di valori di verità. Il verificatore controlla che ciascuna lista contenga esattamente un valore (vero o falso) per ciascuna variabile di Φ , e che entrambe le assegnazioni di verità soddisfino la formula Φ .

Per dimostrare che DOUBLE SAT è NP-hard consideriamo la seguente riduzione dal problema SAT: considerata una generica istanza $\langle \Phi \rangle$ di SAT, sia Φ' la formula booleana ottenuta da Φ aggiungendo una nuova variabile v e ponendo

$$\Phi' = \Phi \wedge (v \vee \bar{v}).$$

È immediato verificare che se la formula Φ è soddisfacibile allora esistono almeno due assegnazioni di verità che soddisfano Φ' : la assegnazione che soddisfa Φ estesa con $v = T$ e la

stessa assegnazione estesa con $v = F$. Viceversa, se la formula Φ non è soddisfacibile, allora nemmeno Φ' può essere soddisfacibile, perché la variabile v non appare nella formula Φ e quindi non può contribuire a rendere quella parte della formula Φ' soddisfacibile.

Da tutto ciò si può concludere che DOUBLE SAT è NP-completo.

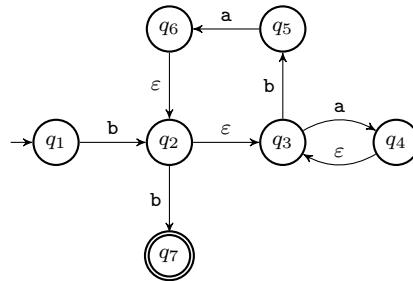
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 8 febbraio 2023

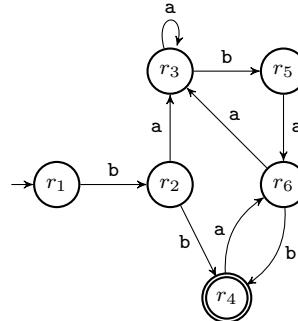
Esercizio 1 [6] Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $b(a^*ba)^*b$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando poche semplificazioni allo NFA derivato meccanicamente dalla espressione regolare si ottiene:

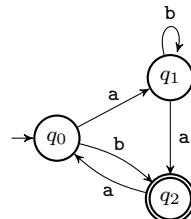


Un automa deterministico equivalente è il seguente:

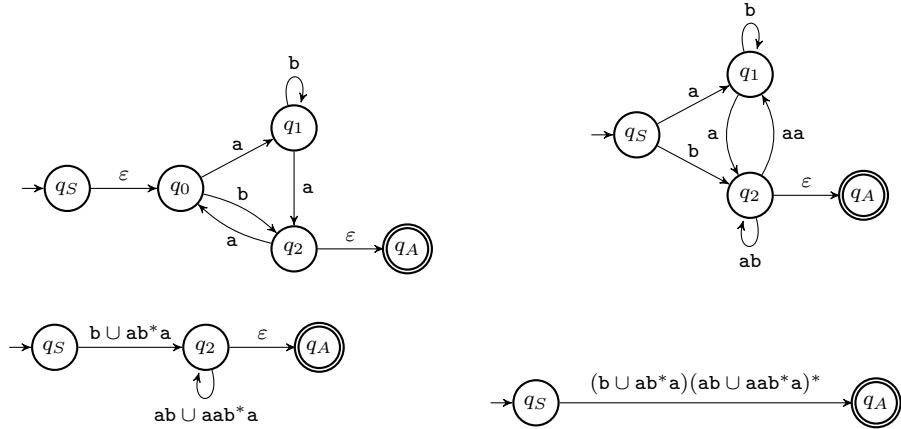
$$\begin{aligned}
 r_1 &= \{q_1\} \\
 r_2 &= \{q_2, q_3\} \\
 r_3 &= \{q_3, q_4\} \\
 r_4 &= \{q_5, q_7\} \\
 r_5 &= \{q_5\} \\
 r_6 &= \{q_2, q_3, q_6\}
 \end{aligned}$$



Esercizio 2 [6] Determinare una espressione regolare per il linguaggio riconosciuto dal seguente automa deterministico:



Soluzione: Trasformiamo l'automa in un GNFA aggiungendo gli stati q_S e q_A , e rimuoviamo nell'ordine i nodi q_0 , q_1 e q_2 :



Una espressione regolare che genera il linguaggio riconosciuto dall'automa è quindi:

$$(b \cup ab^*a)(ab \cup aab^*a)^*$$

Cambiando l'ordine di eliminazione dei nodi si ottengono espressioni regolari diverse ma equivalenti. Ad esempio:

$$\begin{aligned} &(ba \cup ab^*aa)^*(b \cup ab^*a) \quad (\text{ordine: } q_2, q_1, q_0) \\ &(b \cup ab^*a)[a(b \cup ab^*a)]^* \quad (\text{ordine: } q_1, q_0, q_2) \end{aligned}$$

Esercizio 3 [7] Sia C un linguaggio regolare, e sia M un DFA tale che $L(M) = C$. Dimostrare che C è l'insieme vuoto se e solo se C non include alcuna stringa di lunghezza inferiore al numero di stati interni di M .

Soluzione: Denotiamo con p il numero di stati interni dell'automa M . La dimostrazione della condizione necessaria è banale. Infatti se $C = \emptyset$, allora non esiste alcuna stringa $w \in C$, ed in particolare non esiste alcuna stringa $w \in C$ con $|w| < p$. Per dimostrare la condizione sufficiente, assumiamo che C non includa alcuna stringa di lunghezza inferiore a p , e supponiamo per assurdo che C non sia vuoto. Ciò implica che esiste un elemento $w \in C$ avente lunghezza minima tra quelli in C con necessariamente $|w| = n \geq p$. Poiché $w \in C$, la computazione $M(w)$ è accettante: sia q_0, \dots, q_n la sequenza di stati interni assunti dall'automa man mano che vengono letti i simboli di $w = w_1 w_2 \dots w_n$. Poiché $n \geq p$, per il principio “pigeonhole” deve esistere almeno uno stato \bar{q} che si ripete nei primi $p+1$ stati nella sequenza:

$$q_0, q_1, \dots, q_i = \bar{q}, \dots, q_j = \bar{q}, q_{j+1}, \dots, q_n$$

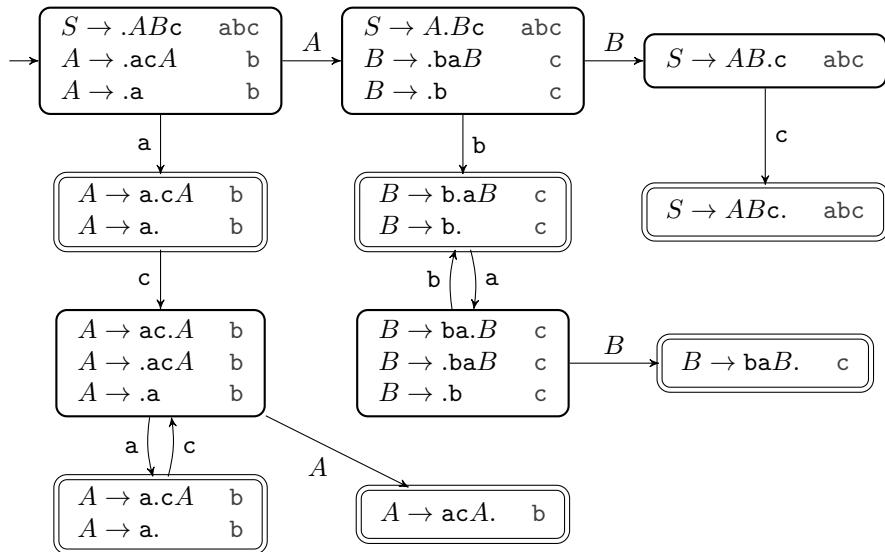
con $0 \leq i < j \leq p$. Consideriamo dunque la stringa $w' = w_1 \dots w_i w_{j+1} \dots w_n$: nella computazione $M(w')$ l'automa deterministico entra nello stato q_i dopo aver letto w_i , quindi legge w_{j+1} e transita in q_{j+1} ; da questo momento in poi l'automa deterministico segue le stesse transizioni utilizzate per la parte finale della stringa w , quindi termina nello stato di accettazione q_n . Perciò $w' \in C$ e $|w'| < |w|$. Ma questa è una contraddizione, perché abbiamo supposto che w sia un elemento di C di lunghezza minima, e tale contraddizione può derivare unicamente dall'aver supposto che $C \neq \emptyset$. Resta dunque dimostrato che $C = \emptyset$.

Esercizio 4 [6] Si consideri la grammatica libera dal contesto con variabile iniziale S e regole

$$S \rightarrow ABc \quad A \rightarrow acA \mid a \quad B \rightarrow baB \mid b$$

La grammatica è deterministica? È LR(1)? Giustificare le risposte.

Soluzione: Per determinare se la grammatica è deterministica o LR(1) eseguiamo il DK₁-test:



Poiché in nessun stato accettante sono incluse regole consistenti tra loro, il DK₁-test indica che la grammatica è LR(1). D'altra parte, se consideriamo l'automa senza i simboli di lookahead, è evidente che sono presenti stati accettanti che includono regole con il dot che precede un simbolo terminale; pertanto la grammatica analizzata non è deterministica.

Esercizio 5 [6] Sia $B = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) \text{ è regolare}\}$. B è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio B non è decidibile, e per dimostrarlo è sufficiente controllare che le ipotesi del Teorema di Rice siano vere. Si consideri come proprietà P della TM il riconoscere un linguaggio regolare. Tale proprietà è non banale: infatti una TM M_1 che accetta tutte le stringhe riconosce Σ^* , che è regolare; d'altra parte, è banale costruire una TM M_0 che riconosce il linguaggio $\{a^n b^n \mid n \geq 0\}$, che sappiamo essere non regolare; quindi M_0 non soddisfa la proprietà P . Inoltre, P è in effetti una proprietà del linguaggio riconosciuto dalla TM: infatti se due diverse TM riconoscono lo stesso linguaggio, per entrambe vale che il linguaggio è regolare oppure no, dunque entrambe le TM soddisfano la priorità P oppure no. Poiché tutte le ipotesi del Teorema di Rice sono soddisfatte possiamo concludere immediatamente che il linguaggio B contenente codifiche di TM che soddisfano la proprietà P è indecidibile.

Esercizio 6 [9] Il problema NOT-ALL-EQUAL SAT (NAESAT) è costituito dalle istanze di SAT in cui esiste una assegnazione di verità alle variabili tale per cui in ogni clausola esiste

almeno un letterale vero ed un letterale falso (ossia, nessuna clausola ha i valori dei letterali tutti veri o tutti falsi). Dimostrare che NAESAT è NP-completo.

Soluzione: Per prima cosa dimostriamo l'appartenza di NAESAT in NP. Il problema è polinomialmente verificabile perché ogni istanza che fa parte del linguaggio ha come certificato l'assegnazione di verità alle variabili che garantisce la presenza in ciascuna clausola di un letterale falso e di un letterale vero. Tale certificato è certamente di dimensione non superiore alla dimensione dell'istanza, e verificare che l'assegnazione soddisfa i requisiti del problema è implementabile in modo immediato, in modo del tutto analogo alla verifica usata per SAT che una assegnazione di verità renda almeno un letterale vero in ogni clausola.

Per dimostrare la NP-hardness di NAESAT costruiamo una riduzione dal problema NP-completo 3SAT. Data una istanza di 3SAT

$$\Phi = \bigwedge_i \bigvee_{j=1}^3 l_{i,j},$$

la riduzione polinomiale costruisce la formula CNF

$$\Psi = \bigwedge_i \left(s \vee \bigvee_{j=1}^3 l_{i,j} \right),$$

ove s è una variabile che non appare in alcuno dei letterali $l_{i,j}$.

Supponiamo che Φ sia una “istanza sì” di 3SAT, quindi che esista un assegnazione di valori di verità alle variabili di Φ che rende almeno un letterale entro ogni clausola vero. La stessa assegnazione di valori, estesa con l'assegnazione del valore “falso” alla variable s , assicura che nella formula Ψ esista almeno un letterale vero ed un letterale falso in ogni clausola. Pertanto, Ψ derivata da Φ è una “istanza sì” di NAESAT.

Viceversa, supponiamo che una formula Ψ costruita con lo stesso procedimento sia una “istanza sì” di NAESAT, pertanto che esista una assegnazione di verità alle variabili di Ψ tale che ogni clausola di Ψ include almeno un letterale vero ed un letterale falso. Distinguiamo due casi: se l'assegnazione di verità ha posto la variabile s a “falso”, allora la stessa assegnazione di verità (senza s) soddisfa la formula Φ da cui Ψ è stata derivata, poiché le rimanenti variabili garantiscono la presenza di almeno un letterale vero in ogni clausola di Φ . Se invece l'assegnazione di verità ha posto la variabile s a “vero”, allora si consideri l'assegnazione di verità complementare, in cui tutte le variabili ricevono la negazione del valore precedentemente assegnato. È immediato verificare che la nuova assegnazione di verità continua a garantire la presenza in ogni clausola di Ψ di un letterale vero e di un letterale falso, perché il valore di ciascun letterale viene negato rispetto alla precedente assegnazione. Quindi anche la assegnazione complementare certifica che Ψ appartiene a NAESAT. Nella assegnazione complementare la variabile s ha il valore “falso”, dunque si ricade nel caso precedente: la formula Φ è una “istanza sì” di 3SAT.

In conclusione, ogni “istanza sì” di 3SAT corrisponde ad una “istanza sì” di NAESAT, ed ogni “istanza no” di 3SAT corrisponde ad una “istanza no” di NAESAT. La trasformazione dalle istanze di 3SAT a quelle di NAESAT è evidentemente costruibile in tempo polinomiale e costituisce una riduzione tra problemi. Quindi NAESAT è NP-hard, e pertanto è NP-completo, come volevasi dimostrare.

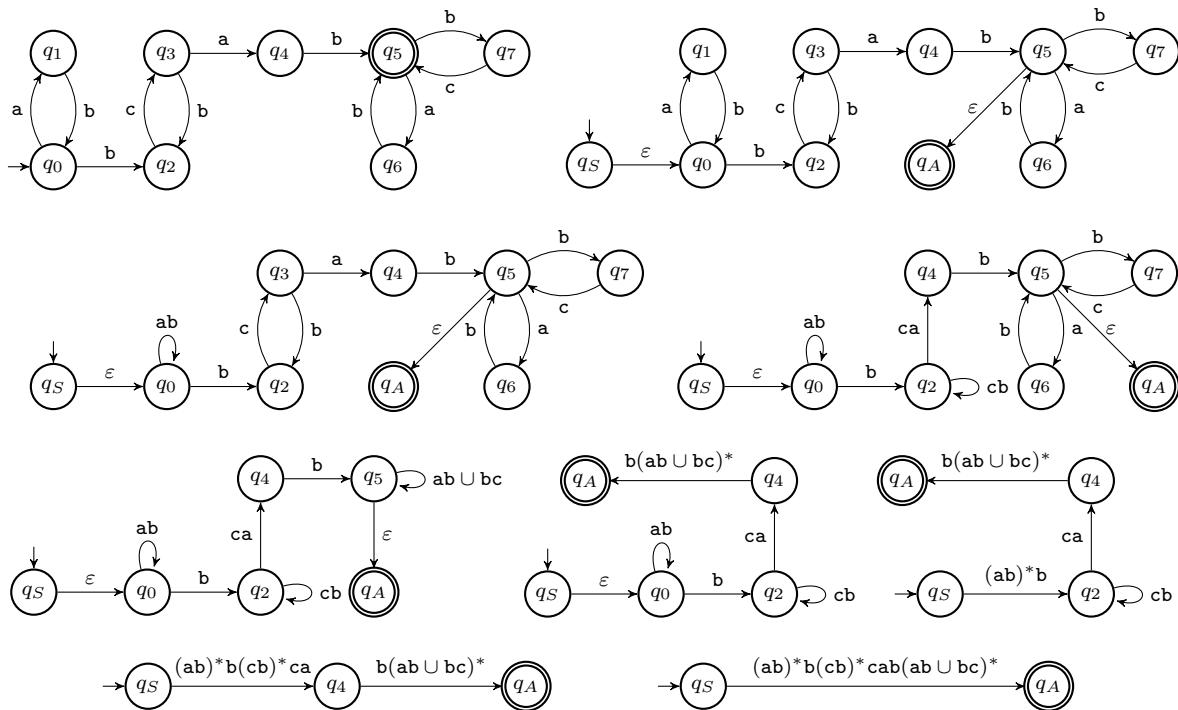
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 11 luglio 2023

Esercizio 1 [6] Sia A il linguaggio contenente le stringhe in $\{ab, bc\}^*$ tali che almeno una ‘c’ precede (anche non immediatamente) una ‘a’. Ad esempio, $\varepsilon \notin A$, $cba \notin A$, $bcab \in A$, $bcba \notin A$, $abbc \notin A$. Determinare una espressione regolare per il linguaggio A , ovvero dimostrare che una tale espressione regolare non esiste.

Soluzione: Poiché $A \subseteq \{ab, bc\}^*$, A contiene le stringhe costituite dalla concatenazione di un numero finito di sottostringhe “ab” e “bc”; inoltre, in ogni elemento di A deve esistere una sottostringa “ab” che appare dopo una sottostringa “bc”. Come primo passo costruiamo un DFA che riconosce le stringhe appartenenti ad A , trasformiamolo in GNFA, e rimuoviamo nell’ordine i nodi q_1 , q_3 , q_6 e q_7 , q_5 , q_0 , q_2 , ed infine q_4 .



Quindi una possibile soluzione è:

$$(ab)^*b(cb)^*cab(bc \cup ab)^* \quad \text{ossia} \quad (ab)^*(bc)^+ab(bc \cup ab)^*.$$

Esercizio 2 [6] Sia $B = \{\mathbf{a}^n\mathbf{b}^m \mid n \neq m\}$; dimostrare che il linguaggio B non è regolare.

Soluzione: Una facile dimostrazione della non regolarità di B può essere trovata osservando che B è strettamente legato al linguaggio $C = \{\mathbf{a}^n\mathbf{b}^n \mid n \geq 0\} = \{\mathbf{a}^n\mathbf{b}^m \mid n = m\}$, che sappiamo essere non regolare. Se infatti $w \in C$ allora $w \notin B$, e vice versa. Consideriamo quindi il complemento \overline{C} del linguaggio $C \subseteq \{\mathbf{a}, \mathbf{b}\}^*$: esso contiene sia tutte le stringhe con i simboli ‘ \mathbf{a} ’ e ‘ \mathbf{b} ’ fuori ordine, sia le stringhe con ‘ \mathbf{a} ’ e ‘ \mathbf{b} ’ in ordine ma diseguali in numero. In altri termini:

$$\overline{C} = \overline{D} \cup B \quad \text{ossia} \quad C = D \cap \overline{B}$$

ove D è l’insieme di stringhe generate dall’espressione regolare $\mathbf{a}^*\mathbf{b}^*$. Sappiamo che i linguaggi regolari sono chiusi rispetto all’intersezione ed al complemento, e che D è regolare in quanto generato da una espressione regolare. Se per assurdo B fosse un linguaggio regolare, allora lo sarebbe anche \overline{B} e quindi $D \cap \overline{B}$, ossia C . Ma questo contraddice il fatto che C non è regolare. Resta perciò dimostrato che B non è un linguaggio regolare.

È possibile anche dimostrare che il linguaggio B non è regolare tramite il pumping lemma, ma la scelta della stringa s non è banale. Supponiamo per assurdo che B sia regolare, e dunque che esista per esso $p > 0$ tale che ogni stringa di lunghezza $\geq p$ ammetta una suddivisione che può essere pompata conservando l’appartenenza a B . Se scegliamo come s la stringa $\mathbf{a}^p\mathbf{b}^{p+1}$, $s \in B$ e $|s| \geq p$, ma in effetti s ammette una suddivisione che può essere pompata. Infatti, sia $s = xyz$ con $|xy| \leq p$ e $|y| = k > 0$. Risulta evidente che le possibili suddivisioni di s in cui la lunghezza k di y è fissata sono tutte equivalenti, e per la generica stringa pompata si ottiene $xy^iz = \mathbf{a}^{p+k(i-1)}\mathbf{b}^{p+1}$. Ora dovremmo dimostrare che per ogni suddivisione di s , ossia per ogni k intero compreso tra 1 e p , esiste un indice j tale che $xy^jz \notin B$, il che significa che $p + k(j - 1) = p + 1$, ossia $k(j - 1) = 1$; infatti, per la non appartenenza a B è necessario che il numero di ‘ \mathbf{a} ’ e ‘ \mathbf{b} ’ sia uguale. Ora però non possiamo concludere che per ogni intero k compreso tra 1 e p il valore $j = 1 + 1/k$ è certamente intero. Quindi la stringa s non funziona perché può effettivamente essere pompata. Questa analisi ci suggerisce però un modo per costruire una stringa s' che non può essere pompata: sia $s' = \mathbf{a}^p\mathbf{b}^{p+p!}$, ove $p! = \prod_{h=2}^p h$. Ripetendo l’analisi precedente, ogni suddivisione di s' dipende dalla lunghezza k di y , e per ciascun valore di k tra 1 e p deve esistere un intero j tale che $p + k(j - 1) = p + p!$, ossia $k(j - 1) = p!$. Tale intero però effettivamente esiste: $j = 1 + \frac{p(p-1)\dots2\cdot1}{k}$, perché k è un intero compreso tra 1 e p . Quindi la stringa s' non ammette una suddivisione che può essere pompata, quindi il pumping lemma non vale. Resta perciò dimostrato che B non è regolare.

Esercizio 3 [7] Siano A e B linguaggi regolari, e sia $A \diamond B = \{xy \mid x \in A, y \in B, |x| = |y|\}$. Dimostrare che $A \diamond B$ è un linguaggio libero dal contesto (CFL).

Soluzione: Sappiamo che la concatenazione $A \diamond B$ dei linguaggi regolari A e B è un linguaggio regolare, ma questo non è sufficiente a stabilire la regolarità di $A \diamond B \subseteq A \diamond B$: infatti $xy \in A \diamond B$ se e solo se $x \in A$, $y \in B$, ed inoltre $|x| = |y|$. È esattamente quest’ultima condizione che

non può in generale essere controllata da un automa a stati finiti. D'altra parte l'esercizio richiede di dimostrare che $A \diamond B$ è CFL, e questo effettivamente si può fare esibendo un PDA che è in grado di confrontare la lunghezza di due stringhe.

Sia dato dunque un generico linguaggio regolare A , e sia D_A un automa a stati finiti che riconosce i suoi elementi; analogamente sia dato un generico linguaggio regolare B ed un corrispondente automa a stati finiti D_B . Consideriamo il seguente automa a pila (PDA):

- $P =$ “On input w , where w is a string:
1. Push the “end of stack” symbol ‘\$’ on the stack
 2. Start simulating D_A on w , while at the same time:
 3. For every character read from the input:
 4. Push one symbol ‘#’ on the stack
 5. If D_A reaches an accepting state:
 6. Nondeterministically jumps to step 7
 7. Start simulating D_B on the remaining portion of w , while at the same time:
 8. For every character read from the input:
 9. Pop one symbol from the stack
 10. When the input string has been fully read:
 11. Check if D_B is in an accepting state; if not, reject
 12. Check if stack top contains the ‘\$’ pushed in step 1: if not, reject
 13. Accept”

È facile derivare P da D_A e D_B : ad esempio, per simulare D_A , per ogni transizione $\delta_A(q, \alpha) = q'$ di D_A includiamo in P una transizione $\delta_P(q, \alpha, \varepsilon) = \{(q', \#)\}$. Analogamente, per simulare D_B , per ogni transizione $\delta_B(q, \alpha) = q'$ di D_B includiamo in P una transizione $\delta_P(q, \alpha, \#) = \{(q', \varepsilon)\}$.

Dimostriamo che P riconosce esattamente il linguaggio $A \diamond B$, e dunque che $A \diamond B$ è CFL. Supponiamo che $w \in A \diamond B$; dunque per definizione $w = xy$, con $x \in A$, $y \in B$ e $|x| = |y|$. Il PDA simula D_A sull'input w , e quindi esiste un ramo di computazione deterministica che raggiunge uno stato di accettazione di D_A dopo aver letto l'ultimo simbolo di x . Dunque esiste un ramo di computazione deterministica che, nel passo 6, termina la simulazione di D_A e comincia la simulazione di D_B sulla restante parte dell'input, ossia su y . Quando l'input è stato interamente consumato lo stato interno di D_B raggiunto è accettante. Inoltre il numero di simboli ‘#’ aggiunti allo stack nel passo 4 coincide con il numero di simboli rimossi nel passo 9. Pertanto il ramo di computazione deterministica giunge ad accettare nel passo 13.

Viceversa, supponiamo che P accetti una stringa w . Per definizione, deve esistere un ramo di computazione deterministica che giunge al passo 13 (l'unico che accetta la stringa). Pertanto al passo 12 lo stack doveva essere vuoto (ossia con ‘\$’ sulla cima), ed al passo 11 la simulazione di D_B doveva essersi conclusa in uno stato di accettazione. In questo ramo di computazione

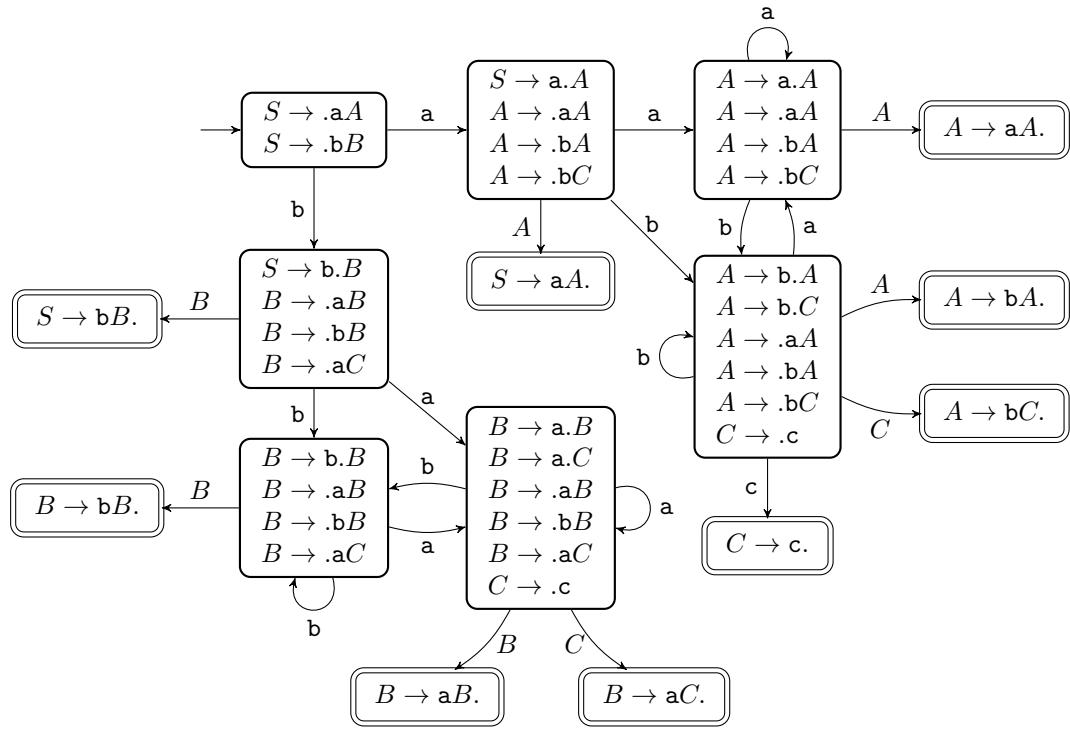
il passo 6 ha portato ad una suddivisione della stringa $w = xy$ in modo tale che la porzione y , analizzata da D_B , è un elemento del linguaggio B . In effetti però il salto nondeterministico del passo 6 può avvenire solo a partire da uno stato di accettazione di D_A ; quindi la prima parte della stringa, x , deve essere un elemento di A . Infine, poiché nel passo 12 lo stack è stato trovato vuoto, necessariamente $|x| = |y|$. Pertanto $w = xy$ è un elemento di $A \diamond B$.

Esercizio 4 [7] Si consideri la grammatica G libera dal contesto con variabile iniziale S

$$S \rightarrow aA \mid bB \quad A \rightarrow aA \mid bA \mid bC \quad B \rightarrow aB \mid bB \mid aC \quad C \rightarrow c$$

- (a) Determinare se G è deterministica (DCFG). (b) Si consideri la grammatica G' ottenuta da G aggiungendo la regola $C \rightarrow \varepsilon$. La nuova grammatica G' è deterministica?

Soluzione: Per determinare se la grammatica G è deterministica costruiamo il corrispondente automa DK:



Poiché tutti gli stati accettanti contengono una unica regola, nessuno di essi contiene due regole compatibili; pertanto, la grammatica G è deterministica.

Se però aggiungiamo alla grammatica anche la regola $C \rightarrow \varepsilon$, allora i due stati contenenti le espansioni iniziali di C diventano:

| | |
|---------------------|---------------------|
| $A \rightarrow b.A$ | $B \rightarrow a.B$ |
| $A \rightarrow b.C$ | $B \rightarrow a.C$ |
| $A \rightarrow .aA$ | $B \rightarrow .aB$ |
| $A \rightarrow .bA$ | $B \rightarrow .bB$ |
| $A \rightarrow .bC$ | $B \rightarrow .aC$ |
| $C \rightarrow .c$ | $C \rightarrow .c$ |
| $C \rightarrow .$ | $C \rightarrow .$ |

A causa della regola completata i due stati sono di accettazione, ma in entrambi esiste anche una regola in cui il dot è seguito da un simbolo terminale. Pertanto, la grammatica con la nuova regola non è più deterministica.

Esercizio 5 [7] Si consideri il linguaggio $L = \{\langle M, w \rangle \mid M \text{ è una DTM con un nastro semi-infinito che su input } w \text{ tenta di muovere la testina a sinistra della prima cella occupata dall'input}\}$. Dimostrare che il linguaggio L non è decidibile.

Soluzione: Osserviamo subito che non è possibile applicare il teorema di Rice, per due differenti motivi. Innanzitutto L non contiene semplicemente codifiche di macchine di Turing, ma coppie costituite da codifiche di macchine di Turing e stringhe di input. In secondo luogo, la proprietà che caratterizza il linguaggio L dipende dalle caratteristiche interne della macchina di Turing in istanza, e non dal linguaggio da essa riconosciuto. Supponiamo infatti che M processi la stringa w senza tentare di muovere la testina a sinistra dell'input; allora possiamo sempre derivare da M una TM M' che simula $M(w)$ e muove la testina a sinistra dell'input dopo aver terminato la simulazione, per poi accettare e rifiutare come $M(w)$.

La precedente considerazione suggerisce una dimostrazione dell'asserto dell'esercizio: dimostriamo cioè che \mathcal{A}_{TM} riduce tramite funzione al linguaggio L , ossia $\mathcal{A}_{\text{TM}} \leq_m L$. Dobbiamo dunque mostrare come sia possibile derivare da una istanza $\langle N, x \rangle$ di \mathcal{A}_{TM} una istanza $\langle M, w \rangle$ di L tale che $\langle N, x \rangle \in \mathcal{A}_{\text{TM}}$ se e solo se $\langle M, w \rangle \in L$. In effetti, poniamo $w = x$ e consideriamo la seguente TM:

M = “On input w , where w is a string;

1. In a single transition:
 2. Read the symbol σ under the head
 3. Write the special symbol ‘\$’
 4. Remember the symbol σ using an internal state
 5. Move the head to the right
6. Repeat until a blank symbol is read:
 7. In a single transition:
 8. Read the symbol σ under the head
 9. Write the symbol stored in the internal state
 10. Remember the symbol σ using an internal state

11. Move the head to the right
12. Write the symbol stored in the internal state
13. Move the head to the left up to the symbol ‘\$’
14. Move the head to the right on the first symbol of w
15. Simulate the TM N on input w
16. If N moves the head to the left:
 17. If the symbol under the head is ‘\$’:
 18. Move back the head to the right
 19. If $N(w)$ accepts:
 20. Move the head to the left up to ‘\$’
 21. Move the head to the left once more
 22. Halt”

I passi 1–14 eseguono una copia dell’input w spostandolo di una cella verso destra; nella prima cella occupata dall’input originario viene scritto il carattere speciale ‘\$’ (che non appare nell’alfabeto di nastro di N). Nei passi 14–18 viene simulata l’esecuzione della TM N sull’input w . Durante la simulazione viene effettuato un controllo dopo ogni mossa della testina verso sinistra: se si è arrivati a leggere il carattere ‘\$’, la testina viene mossa verso destra prima di continuare la simulazione, in modo da implementare il meccanismo di “rimbalzo della testina” nella computazione $N(w)$. I passi 19–21, nel caso in cui $N(w)$ abbia accettato, muovono la testina a sinistra del simbolo speciale ‘\$’, ossia a sinistra della posizione iniziale dell’input originale della TM M .

Supponiamo che $\langle N, w \rangle \in \mathcal{A}_{\text{TM}}$; allora $M(w)$ simula $N(w)$, e poi effettua lo spostamento a sinistra nei passi 19–21. Pertanto $\langle M, w \rangle \in L$. Se invece $\langle N, w \rangle \notin \mathcal{A}_{\text{TM}}$, la simulazione di $N(w)$ dei passi 14–18 garantisce che la testina non si muova mai a sinistra dell’input w originale. Se $N(w)$ dovesse terminare rifiutando, i passi 20–21 non vengono eseguiti e quindi $M(w)$ termina senza tentare di spostare la testina a sinistra dell’input originale. Se invece $N(w)$ non dovesse terminare, M non arriva mai ad eseguire il passo 19 e non termina mai. In entrambi i casi, risulta che $\langle M, w \rangle \notin L$.

È evidente che la trasformazione dall’istanza $\langle N, w \rangle$ all’istanza $\langle M, w \rangle$ può essere effettuata da una TM in modo meccanico e deterministico, e dunque $\mathcal{A}_{\text{TM}} \leq_m L$. Poiché \mathcal{A}_{TM} non è decidibile, anche L non è decidibile, come volevasi dimostrare.

Esercizio 6 [7] Dimostrare che se L è un linguaggio in coNP, allora il linguaggio ottenuto applicando l’operatore “star” di Kleene al complemento di L (ossia $(L^c)^*$) è un linguaggio in NP.

Soluzione: Poiché $L \in \text{coNP}$, per definizione $L^c \in \text{P}$. L’asserto da dimostrare è dunque che se $K = L^c \in \text{NP}$ allora $K^* \in \text{NP}$.

Per la definizione dell'operatore di Kleene, un elemento di K^* è costituito dalla concatenazione di un numero finito di elementi di K . Poiché K è in NP, esiste una NTM M che decide ogni istanza di K in tempo nondeterministico polinomiale. Possiamo quindi derivare da M la seguente NTM:

- $N =$ “On input w , where w is a string:
1. If $w = \varepsilon$, then accept
 2. Nondeterministically guess a subdivision $w_1 w_2 \cdots w_k$ of w :
 3. For every substring w_i :
 4. Run M on w_i
 5. If M accepted all substrings w_i , then accept; otherwise reject”

Osserviamo che la NTM N esegue in tempo nondeterministico polinomiale. Infatti N genera tramite il nondeterminismo tutte le possibili suddivisioni della stringa w ; per ciascuna suddivisione, N esegue al più $|w|$ volte la NTM M (il numero di sottostringhe per ciascuna suddivisione non può essere superiore alla lunghezza di w), e ciascuna esecuzione di M completa in tempo nondeterministico polinomiale.

Supponiamo che $w \in K^*$; pertanto o w è la stringa vuota ε (ed in tale caso N accetta al passo 1) oppure w è la concatenazione di un numero finito k di elementi K : $w = w_1 w_2 \cdots w_k$. Nel ramo di computazione deterministica corrispondente a $w_1 w_2 \cdots w_k$, N esegue $M(w_1)$, $M(w_2), \dots, M(w_k)$, e tutte queste esecuzioni sono accettanti perché ciascuna sottostringa è un elemento di K . Dunque N accetta al passo 5.

Viceversa, supponiamo che $N(w)$ accetti una determinata stringa w . Se lo ha fatto al passo 1, allora $w = \varepsilon$, ma $\varepsilon \in K^*$ per definizione dell'operatore di Kleene. Se invece lo ha fatto al passo 5, allora deve esistere un ramo di computazione deterministica in cui tutte le esecuzioni delle NTM M hanno accettato; pertanto deve esistere una suddivisione di w in un certo numero di sottostringhe k ($w = w_1 w_2 \cdots w_k$) tale che $w_i \in K$ per ciascun i ; di conseguenza, per la definizione dell'operatore di Kleene, $w \in K^*$.

Una dimostrazione alternativa si basa sulla esistenza di un verificatore polinomiale V' per K^* derivato dal verificatore polinomiale V per K . Infatti, sappiamo che per ciascuna istanza-sì w_i di K esiste un certificato c_i che può essere verificato da V in tempo polinomiale in $|w_i|$. Consideriamo dunque una istanza-sì w di K^* ; per definizione dell'operatore di Kleene, o $w = \varepsilon$, oppure $w = w_1 w_2 \cdots w_k$ con $w_i \in K$. Un certificato per w consiste nella concatenazione $c_1 \# c_2 \# \cdots \# c_k$ dei certificati di ciascun w_i . Il verificatore V' genera deterministicamente tutte le possibili suddivisioni della sottostringa w (sono al più $O(n^2)$ se $n = |w|$); per ciascuna suddivisione $w = w_1 w_2 \cdots w_k$, V' esegue V per verificare se il certificato c_i conferma che $w_i \in K$. Al termine V' accetta se ha trovato una suddivisione per w verificata dai certificati c_i . Poiché V esegue in tempo polinomiale in $|w_i| \leq n$, anche V' completa il lavoro in tempo polinomiale in n .

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 21 settembre 2023

Esercizio 1 [7] Determinare un automa a stati finiti deterministico (DFA) per il linguaggio contenente tutti i numeri in base 3 multipli di 4, ovvero dimostrare che tale automa non esiste.

Soluzione: L'alfabeto di ingresso dell'automa richiesto è costituito dalle cifre ‘0’, ‘1’ e ‘2’. L'automa deve essenzialmente “memorizzare” tramite i propri stati interni quale sia il resto della divisione tra il numero parzialmente letto in input ed il valore quattro. Quindi sia q_j , con j compreso tra 0 e 3, lo stato interno che memorizza il resto j nella divisione per quattro. Se i simboli in ingresso terminano avendo l'automa nello stato interno q_0 , allora il valore ternario letto in ingresso è un multiplo di quattro; pertanto, q_0 è l'unico stato di accettazione.

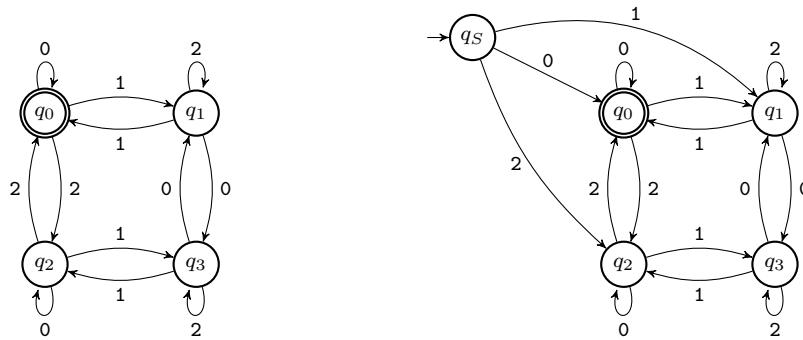
Le transizioni da uno stato all'altro sono condizionate dal simbolo letto in ingresso. Per ogni successiva cifra letta il valore ottenuto precedentemente deve essere moltiplicato per tre (essendo appunto i numeri codificati in base 3), e poi al valore ottenuto va sommata la nuova cifra. Quindi le transizioni si ottengono nel seguente modo:

1. Nello stato q_0 il valore finora letto dall'input è un multiplo di quattro, ossia $4k$ per un generico $k \in \mathbb{N}$.
 - ‘0’ Si ottiene il valore $3 \times (4k) = 4 \times (3k)$, quindi ancora un multiplo di quattro: si rimane in q_0 .
 - ‘1’ Si ottiene il valore $3 \times (4k) + 1 = 4 \times (3k) + 1$, quindi si transita in q_1 .
 - ‘2’ Si ottiene il valore $3 \times (4k) + 2 = 4 \times (3k) + 2$, quindi si transita in q_2 .
2. Nello stato q_1 il valore finora letto dall'input è a distanza uno da un multiplo di quattro, ossia $4k + 1$ per un generico $k \in \mathbb{N}$.
 - ‘0’ Si ottiene il valore $3 \times (4k + 1) = 4 \times (3k) + 3$, quindi si transita in q_3 .
 - ‘1’ Si ottiene il valore $3 \times (4k + 1) + 1 = 4 \times (3k + 1)$, quindi si transita in q_0 .
 - ‘2’ Si ottiene il valore $3 \times (4k + 1) + 2 = 4 \times (3k + 1) + 1$, quindi si rimane in q_1 .
3. Nello stato q_2 il valore finora letto dall'input è a distanza due da un multiplo di quattro, ossia $4k + 2$ per un generico $k \in \mathbb{N}$.
 - ‘0’ Si ottiene il valore $3 \times (4k + 2) = 4 \times (3k + 1) + 2$, quindi si rimane in q_2 .
 - ‘1’ Si ottiene il valore $3 \times (4k + 2) + 1 = 4 \times (3k + 1) + 3$, quindi si transita in q_3 .
 - ‘2’ Si ottiene il valore $3 \times (4k + 2) + 2 = 4 \times (3k + 2)$, quindi si transita in q_0 .

4. Nello stato q_3 il valore finora letto dall'input è a distanza tre da un multiplo di quattro, ossia $4k + 3$ per un generico $k \in \mathbb{N}$.

- ‘0’ Si ottiene il valore $3 \times (4k + 3) = 4 \times (3k + 2) + 1$, quindi si transita in q_1 .
- ‘1’ Si ottiene il valore $3 \times (4k + 3) + 1 = 4 \times (3k + 2) + 2$, quindi si transita in q_2 .
- ‘2’ Si ottiene il valore $3 \times (4k + 3) + 2 = 4 \times (3k + 2) + 3$, quindi si rimane in q_3 .

In conclusione si ottiene il seguente automa (a sinistra)



Lo stato iniziale non può però essere q_0 , in quanto la stringa vuota ε non dovrebbe essere accettata: essa non rappresenta un numero in base 3. Pertanto, aggiungiamo all'automa uno stato iniziale q_S il cui scopo è leggere il primo simbolo del numero in ingresso. Si ottiene perciò il DFA a destra qui sopra, che rappresenta la soluzione dell'esercizio.

Esercizio 2 [6] Si consideri il linguaggio $A = \{x=y-z \mid x, y, z \in \{0, 1\}^*\}$ e il numero binario x è la differenza dei numeri binari y e $z\}$. Ad esempio, $10=11-1 \in A$, $10=11-0 \notin A$. Il linguaggio A è regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A non è regolare, in quanto intuitivamente per decidere se una stringa appartiene o meno al linguaggio l'automa dovrebbe saper “contare”, ossia valutare il valore numerico corrispondente alle sottostringhe x , y e z . Per dimostrare formalmente che A non è regolare utilizziamo il pumping lemma. Si assuma per assurdo che A sia regolare. Esiste allora una lunghezza $p > 0$ tale che ogni stringa s di lunghezza maggiore o uguale a p ammette una suddivisione “pompabile”. Sia $s = 1^p=1^p-0$. La stringa di lunghezza $2p+3$ appartiene evidentemente al linguaggio, perché il numero codificato da 1^p (ossia $2^p - 1$) è presente sia a sinistra che a destra del simbolo ‘=’, e a destra gli viene sottratto il valore zero. Il pumping lemma afferma dunque che deve esistere una suddivisione $s = uvw$ tale che $|uv| \leq p$, $|v| > 0$, e $uv^iw \in A$ per ogni $i \geq 0$. La condizione $|uv| \leq p$ implica che la sottostringa v deve essere interamente compresa nella porzione di simboli ‘1’ a sinistra del simbolo ‘=’; inoltre la condizione $|v| > 0$ implica che v deve includere almeno un simbolo ‘1’. Consideriamo dunque il caso $i = 0$: la stringa risultante è necessariamente nella forma

$1^q=1^p-0$, con $q < p$. Pertanto la uguaglianza numerica ($2^q - 1 = 2^p - 1 - 0$) non può essere vera, e quindi $uv^0w = uw \notin A$. Ciò contraddice l'asserto del pumping lemma. Pertanto A non è un linguaggio regolare.

Esercizio 3 [7] Derivare un automa a pila (PDA) per il linguaggio $A = \{x\#y \mid x, y \in \{0, 1\}^*\}$ e x è la codifica binaria $\langle|y|\rangle$ della lunghezza di y , ovvero dimostrare che non è possibile derivare un tale PDA.

Soluzione: In effetti non è possibile costruire un PDA per il linguaggio A . È tuttavia estremamente difficile dimostrare questo fatto ragionando direttamente sul PDA, perché si dovrebbe dimostrare che *qualsiasi* algoritmo concepibile (tra gli infiniti possibili) fallirebbe nel riconoscere gli elementi di A . Possiamo invece ottenere il risultato ricordando che ogni linguaggio riconosciuto da un PDA è CFL e riconoscendo che A non è CFL. Per dimostrare l'ultimo asserto utilizziamo il “pumping lemma” per i CFL.

Supponiamo per assurdo che A sia CFL. A causa del “pumping lemma”, deve pertanto esistere una lunghezza $p > 0$ tale che, per ogni elemento s di lunghezza maggiore o uguale a p , s può essere suddiviso come $s = uvxyz$ in modo tale che $|vy| > 0$, $|vxy| \leq p$, e per ogni $i \geq 0$, $uv^i xy^i z \in A$. Consideriamo la stringa $s = 1^p \# 0^{2^p-1}$: è immediato verificare che $|s| > p$ e $s \in A$. Dimostriamo che per ogni possibile suddivisione di s esiste un valore di $i \geq 0$ per il quale la stringa “pompata” *non* fa parte di A . In effetti una dimostrazione abbastanza semplice può essere basata sul fatto che ogni simbolo 1 aggiunto a sinistra comporta almeno il raddoppio (in effetti la triplicazione) della lunghezza della stringa a destra; dunque il numero di simboli da aggiungere dovrebbe essere esponenziale in p , mentre la lunghezza della porzione y non può superare p . La stessa idea fondamentale viene sfruttata nella seguente dimostrazione formale in cui si utilizza esclusivamente il “pompaggio verso il basso”.

Cominciamo con l'osservare che ogni suddivisione in cui $'\#'$ $\notin x$ non può soddisfare il lemma. Infatti se $'\#'$ $\notin x$, e considerando che $|vy| > 0$, allora o vxy deve essere tutto a sinistra del simbolo $\#$ oppure deve essere tutto alla sua destra (altrimenti il simbolo $\#$ sarebbe rimosso o duplicato, e la stringa non potrebbe far parte di A in quanto malformata). In entrambi i casi pompare verso il basso produce una stringa in cui la codifica binaria a sinistra non coincide con il numero di simboli a destra.

Assumiamo dunque che $'\#'$ $\in x$, e pertanto $v \in 1^*$ e $y \in 0^*$. Analizziamo i seguenti casi basati sulla lunghezza della sottostringa v :

- $|v| = 0$: in questo caso, poiché $|vy| > 0$, deve valere $|y| > 0$. Pertanto pompando verso il basso, ossia considerando $i = 0$, si ottiene una stringa uxz in cui la codifica binaria a sinistra è identica ma il numero di elementi a destra è diminuito; tale stringa non può appartenere ad A .

- $|v| = 1$: consideriamo il valore $i = 0$, ossia la stringa $1^{p-1}\#0^l$ ottenuta pompando verso il basso, ove il valore l dipende dalla dimensione di y . Perché tale stringa possa far parte di A deve valere $l = 2^{p-1} - 1$, e quindi $|y| = 2^p - 1 - (2^{p-1} - 1) = 2^{p-1}$. Ma allora $|vxy| = 1 + 2^{p-1} + |x| \geq 2 + 2^{p-1}$; ricordando che $|vxy| \leq p$, si ottiene $2^{p-1} \leq p - 2$. In effetti non esiste alcun valore positivo per p che possa soddisfare tale disegualanza, e quindi non esiste suddivisione che possa conservare l'appartenza ad A pompando verso il basso.
- $|v| > 1$: si applica lo stesso ragionamento del caso precedente. Sia $j = |v|$, e consideriamo il valore $i = 0$. Necessariamente la lunghezza di y deve essere pari a $l = 2^p - 1 - (2^{p-j} - 1) = (1 - 2^{-j}) 2^p$. Pertanto: $p \geq |vxy| \geq j + 1 + (1 - 2^{-j}) 2^p$, e poiché $(1 - 2^{-j}) > 1/2$ per $j > 1$:

$$2^{p-1} < 2^p (1 - 2^{-j}) \leq p - j - 1 < p - 1.$$

Poiché questa disegualanza non ha alcuna soluzione per $p > 0$, non esiste alcuna suddivisione che possa conservare l'appartenza ad A pompando verso il basso.

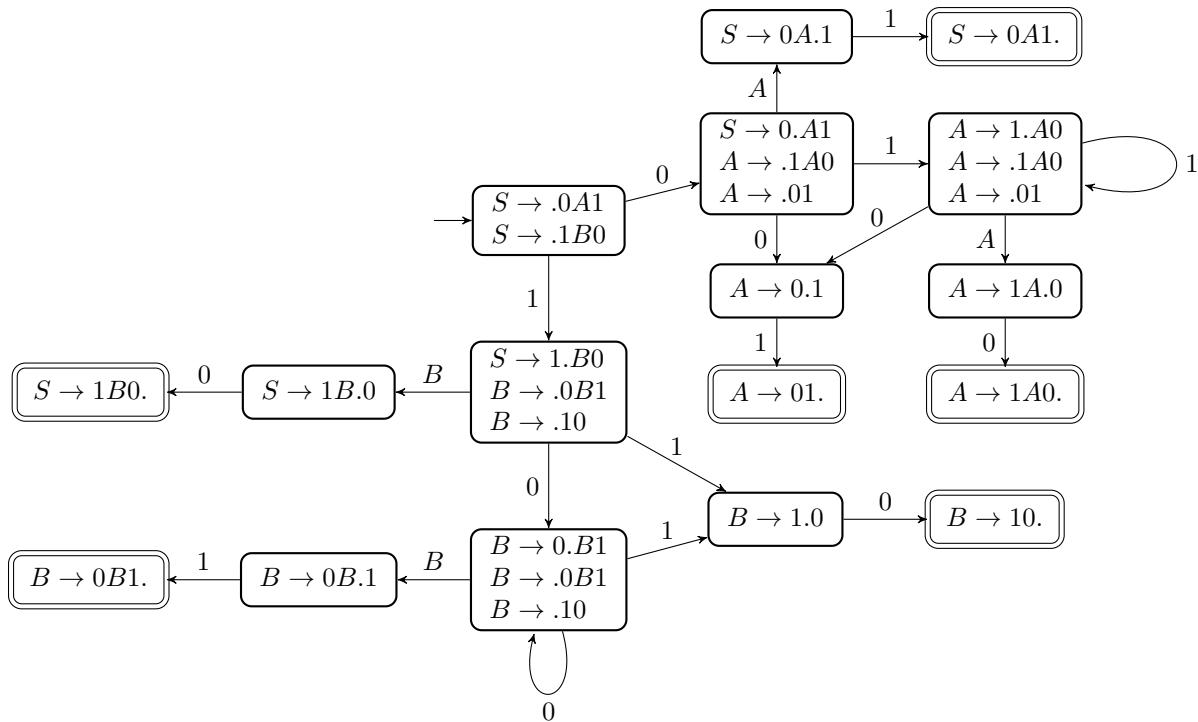
Poiché ogni possibile suddivisione di s risulta non appartente ad A quando pompata verso il basso, le conclusioni del “pumping lemma” non valgono. La contraddizione è dovuta alla supposizione che A sia CFL. Concludiamo pertanto che non esiste alcun PDA che possa riconoscere gli elementi del linguaggio A .

Esercizio 4 [6] Si consideri la grammatica G con variabile iniziale S :

$$S \rightarrow 0A1 \mid 1B0 \quad A \rightarrow 1A0 \mid 01 \quad B \rightarrow 0B1 \mid 10.$$

La grammatica è deterministica? Giustificare la risposta con una dimostrazione.

Soluzione: Per verificare se la grammatica G è deterministica utilizziamo il DK-test.



Poiché tutti gli stati finali dell'automa DK contengono una sola regola completata, la grammatica è deterministica: infatti ovviamente negli stati accettanti non esistono né diverse regole completate né regole non completate in cui il dot precede un simbolo terminale.

Esercizio 5 [7] Dimostrare che la concatenazione di linguaggi Turing-riconoscibili (ossia ricorsivamente enumerabili) è un linguaggio Turing-riconoscibile.

Soluzione: Siano A e B linguaggi ricorsivamente enumerabili; dobbiamo dimostrare che $AB = \{xy \mid x \in A, y \in B\}$ è ricorsivamente enumerabile. Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B , rispettivamente.

Consideriamo la seguente NTM M :

M = “On input w :

1. Guess a subdivision x, y of w : $w = xy$
 2. Emulate the TM M_A on input x
 3. Emulate the TM M_B on input y
 4. If both $M_A(x)$ and $M_B(y)$ accepted, then accept
 5. Otherwise, reject”

Supponiamo che $M(w)$ accetti; di conseguenza esiste una computazione accettante che “indovina” una opportuna suddivisione $xy = w$, ed emula $M_A(x)$ e $M_B(y)$. Poiché $M(w)$ termina accettando, le due emulazioni devono terminare ed entrambe le computazioni devono accettare. Pertanto $x \in A$ e $y \in B$, e quindi $w \in AB$. D’altra parte, supponiamo che $w \in AB$; dunque esiste una suddivisione $w = xy$ tale che $x \in A$ e $y \in B$. Questa suddivisione corrisponde ad un ramo dell’albero di computazioni di $M(w)$; inoltre poiché M_A riconosce A , $M_A(x)$ termina accettando; analogamente $M_B(y)$ termina accettando. Perciò $M(w)$ accetta.

Si osservi che il non-determinismo di M non costituisce un problema, perché per ogni TM non deterministica esiste una TM deterministica equivalente. Perciò il linguaggio AB è ricorsivamente enumerabile.

Esercizio 6 [7] Sia $\mathcal{L} = \{\langle M \rangle \mid M \text{ è una macchina di Turing deterministica tale che esiste almeno una stringa di input } x \text{ per la quale } M(x) \text{ non termina}\}$. Dimostrare che \mathcal{L} non è decidibile.

Soluzione: Osserviamo innanzi tutto che non è possibile ricorrere al Teorema di Rice per risolvere l’esercizio, poiché la proprietà che caratterizza il linguaggio \mathcal{L} non è propria del linguaggio delle macchine di Turing che appartengono a \mathcal{L} . Infatti, sia M una macchina di Turing che termina su ogni input e rifiuta almeno una stringa \bar{x} . È facile derivare da M una TM M' che per ogni input x dapprima controlla se $x = \bar{x}$, ed in tal caso entra in un ciclo senza fine; altrimenti, se $x \neq \bar{x}$, $M'(x)$ simula $M(x)$. Pertanto $L(M) = L(M')$, ma $\langle M \rangle \notin \mathcal{L}$ mentre $\langle M' \rangle \in \mathcal{L}$.

Per dimostrare che \mathcal{L} è non decidibile è sufficiente mostrare una riduzione da un problema indecidibile; è naturale prendere in considerazione \mathcal{A}_{TM} . Possiamo in effetti mostrare che $\mathcal{A}_{\text{TM}} \leq_T \mathcal{L}$ tramite la seguente TM con oracolo \mathcal{L} che decide \mathcal{A}_{TM} :

$M^{\mathcal{L}} =$ “On input $\langle T, x \rangle$, where T is a TM and x is a string:

1. Build from $\langle T, x \rangle$ the following TM:

$M' =$ “On input y , where y is a string:

- (a) If $y \neq x$, then halt
- (b) Run T on input x ”
- (c) Halt”

2. Ask the oracle whether $\langle M' \rangle \in \mathcal{L}$
3. If the answer is YES, reject
4. Otherwise, if the answer is NO, run T on input x
5. If $T(x)$ accepts, then accept; otherwise, reject”

Sia $\langle T, x \rangle \in \mathcal{A}_{\text{TM}}$; dunque $M'(y)$ termina sempre (al passo (a) se $y \neq x$, al passo (c) se $y = x$). Perciò $\langle M' \rangle \notin \mathcal{L}$, e quando la computazione $M^{\mathcal{L}}(\langle T, x \rangle)$ interroga l’oracolo, questo risponderà

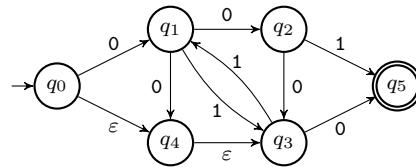
NO. Dunque $M^{\mathcal{L}}(\langle T, x \rangle)$ simula $T(x)$ ed accetta poiché $T(x)$ accetta. Supponiamo invece che $\langle T, x \rangle \notin \mathcal{A}_{\text{TM}}$. Esistono due casi: $T(x)$ rifiuta e $T(x)$ entra in un ciclo senza fine senza terminare. Nel caso in cui $T(x)$ rifiuta, l'oracolo risponde NO perché $T(x)$ termina, quindi non esiste alcun input sul quale M' non termina. $M^{\mathcal{L}}(\langle T, x \rangle)$ arriva dunque a simulare $T(x)$ e, quando quest'ultima computazione termina rifiutando, rifiuta. Se infine $T(x)$ non termina, $M'(x)$ non termina, dunque $\langle M' \rangle \in \mathcal{L}$ e l'oracolo risponde YES; perciò $M^{\mathcal{L}}(\langle T, x \rangle)$ rifiuta al passo 3. Riassumendo, $M^{\mathcal{L}}$ decide \mathcal{A}_{TM} , e quindi $\mathcal{A}_{\text{TM}} \leq_T \mathcal{L}$; poiché \mathcal{A}_{TM} è indecidibile, resta dimostrato che anche \mathcal{L} è indecidibile.

Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

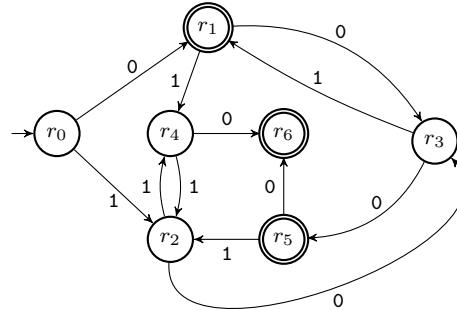
Compito scritto del 24 gennaio 2024

Esercizio 1 [6] Determinare un automa a stati finiti deterministico (DFA) equivalente al seguente automa non deterministico:

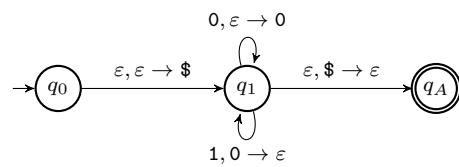


Soluzione: È conveniente determinare preventivamente la chiusura degli stati del NFA rispetto alle ε -transizioni: per tutti gli stati la chiusura coincide con lo stato stesso, tranne che per $E(q_0) = \{q_0, q_4, q_3\}$ e $E(q_4) = \{q_4, q_3\}$. Si ottiene alla fine del procedimento:

$$\begin{aligned}
 r_0 &= \{q_0, q_3, q_4\} \\
 r_1 &= \{q_1, q_5\} \\
 r_2 &= \{q_1\} \\
 r_3 &= \{q_2, q_3, q_4\} \\
 r_4 &= \{q_3\} \\
 r_5 &= \{q_3, q_5\} \\
 r_6 &= \{q_5\}
 \end{aligned}$$



Esercizio 2 [7] Si consideri il linguaggio A riconosciuto dal seguente PDA:



Dimostrare che il linguaggio A non è regolare.

Soluzione: L'automa a pila P è molto semplice, tuttavia la caratterizzazione formale del linguaggio $A = L(P)$ richiede attenzione. Osserviamo che:

- P accetta soltanto se lo stack è vuoto;
- ogni simbolo ‘0’ letto dall’input viene copiato sullo stack;
- ogni simbolo ‘1’ letto dall’input deve corrispondere ad un simbolo ‘0’ sullo stack, che viene conseguentemente rimosso.

Perciò ogni stringa $s \in A$ deve necessariamente:

1. contenere lo stesso numero di simboli ‘0’ e ‘1’ (possibilmente anche 0 occorrenze, ossia $s = \epsilon$);
2. per ciascuna sottostringa s_k di s cominciante dal primo simbolo di s e di lunghezza k ($1 \leq k \leq |s|$), il numero di simboli ‘1’ deve essere minore o uguale del numero di simboli ‘0’.

Ad esempio, $001101 \in A$. Invece, la stringa $\bar{s} = 001110$ non fa parte del linguaggio A in quanto la sottostringa $s_5 = 00111$ ha una eccedenza di simboli ‘1’. La stringa \bar{s} non è in effetti accettata da P poiché nel momento in cui si deve leggere l’ultimo bit ‘1’ di s_5 la cima dello stack contiene il simbolo ‘\$’; pertanto l’unica transizione applicabile porta nello stato finale q_A senza aver completato la lettura dell’input.

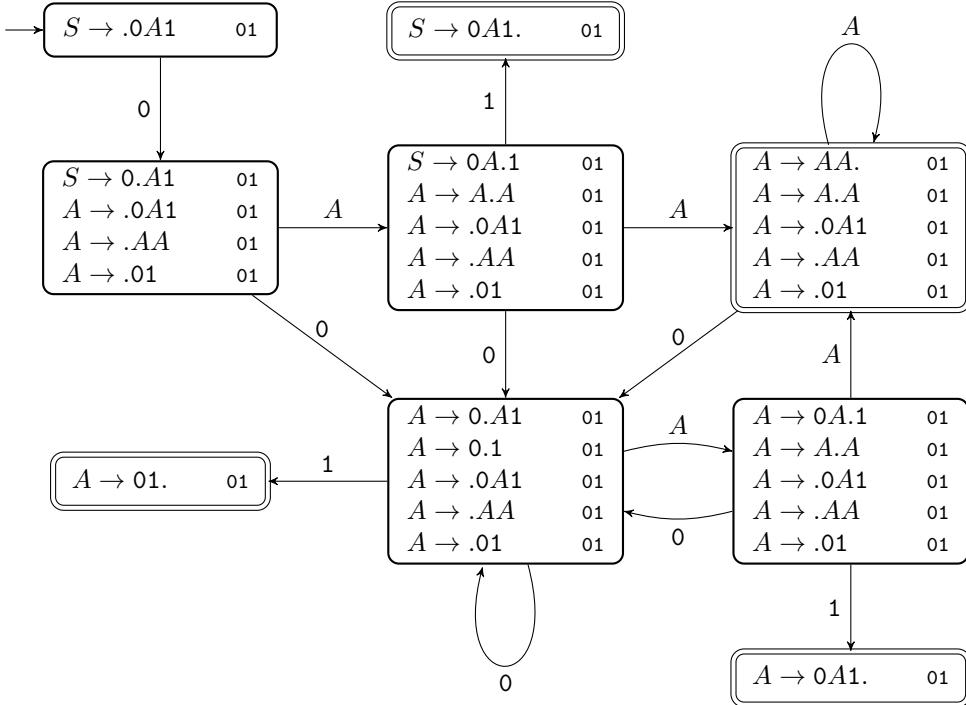
Dimostriamo che A non è regolare. Supponiamo per assurdo che lo sia, e dunque che si possa applicare il Pumping Lemma. Sia $p > 0$ la “pumping length” per A , e consideriamo la stringa $s = 0^p 1^p$. È immediato verificare che $s \in A$ e che $|s| \geq p$. Deve quindi esistere una suddivisione $s = xyz$ con $|y| > 0$, $|xy| \leq p$ e $xy^i z \in A$ per ogni $i \geq 0$. Qualunque sia tale suddivisione, y contiene solo zeri (poiché $|xy| \leq p$) ed almeno uno zero (poiché $|y| > 0$). Consideriamo quindi il caso $i = 0$: la stringa $xy^0 z = xz$ contiene meno zeri di s mentre il numero di ‘1’ è invariato: pertanto $xz \notin A$. Perciò assumere che A sia regolare porta ad una contraddizione.

Esercizio 3 [7] Si consideri la grammatica G con variabile iniziale S :

$$S \rightarrow 0A1 \quad A \rightarrow 0A1 \mid AA \mid 01.$$

La grammatica G è LR(1)? Giustificare la risposta con una dimostrazione.

Soluzione: Per verificare se la grammatica G è LR(1) utilizziamo il DK₁-test.



A titolo di spiegazione, consideriamo come sono determinati i simboli di lookahead per lo stato a cui si arriva dallo stato iniziale leggendo 0. I simboli di lookahead della regola $S \rightarrow 0.A1$ sono 01 perché identici a quelli della regola nello stato iniziale. Dobbiamo introdurre nello stato anche le regole iniziali per le espansioni della variabile A ; tra i simboli di lookahead dobbiamo includere ‘1’, perché è il simbolo seguente A nella regola che ha determinato la reintroduzione delle regole di A . Però, tra le regole introdotte ritroviamo anche $A \rightarrow .AA$, e quindi dovremmo reintrodurre nuovamente le regole iniziali per A , questa volta con i simboli di lookahead relativi al primo simbolo terminale generabile dalla stringa che segue la A che segue il dot, ossia la variabile A stessa. Esaminando le regole di A si evidenzia che il primo simbolo generabile da A è sempre ‘0’; dunque dobbiamo introdurre anche ‘0’ nei simboli di lookahead delle regole iniziali per A . Discorso analogo deve essere fatto in tutti gli stati in cui sono reintrodotte le regole iniziali di A .

Alla fine, uno stato finale dell'automa DK_1 contiene regole consistenti. Infatti, la regola completata ha come simboli di lookahead entrambi i simboli terminali, ed esistono regole in cui il dot precede il simbolo terminale 0. Pertanto la grammatica non è LR(1).

Esercizio 4 [7] Siano A e B due linguaggi Turing-riconoscibili (ossia ricorsivamente enumerabili). Sia $A \setminus B = \{x \in A \mid x \notin B\}$. Quali tra i quattro linguaggi $A \setminus B$, $A \setminus B^c$, $A^c \setminus B$, $A^c \setminus B^c$ è Turing riconoscibile? Giustificare le risposte con dimostrazioni o controesempi.

Soluzione: Siano M_A e M_B le DTM che riconoscono, rispettivamente, i linguaggi A e B . Supponiamo, senza perdita di generalità, che A e B siano linguaggi definiti su di un comune

insieme alfabeto Σ .

(a) $A \setminus B$: questo linguaggio non è necessariamente Turing-riconoscibile. Per dimostrarlo, è sufficiente considerare $A = \Sigma^*$ (un linguaggio regolare, quindi decidibile, e di conseguenza Turing-riconoscibile), e $B = \mathcal{A}_{\text{TM}}$ (il linguaggio Turing-riconoscibile ma non decidibile di accettazione delle TM). L'insieme $A \setminus B = \Sigma^* \setminus \mathcal{A}_{\text{TM}}$ contiene tutte le stringhe che non fanno parte di \mathcal{A}_{TM} , ossia coincide con $\mathcal{A}_{\text{TM}}^c$. Se $A \setminus B$ fosse Turing-riconoscibile, allora sia \mathcal{A}_{TM} che il suo complemento sarebbero Turing-riconoscibili, e quindi \mathcal{A}_{TM} sarebbe decidibile.

(b) $A \setminus B^c$: è necessariamente Turing-riconoscibile. Infatti, è facile verificare che $A \setminus B^c = A \cap B$ e che la seguente TM derivata da M_A e M_B riconosce il linguaggio $A \cap B$:

M_{\cap} = “On input x , where x is a string in Σ^* . ”

1. for $k = 0 \rightarrow \infty$ do:
 2. simulate for at most k steps the computation of M_A on x
 3. simulate for at most k steps the computation of M_B on x
 4. if both $M_A(x)$ and $M_B(x)$ accepted, accept
 5. if either $M_A(x)$ or $M_B(x)$ rejected, reject”

Naturalmente M_{\cap} non decide $A \cap B$ perché se la stringa in input non appartiene ad A o a B la corrispondente DTM potrebbe non terminare affatto. Se però $x \in A \cap B$, sia $M_A(x)$ che $M_B(x)$ terminano in un numero finito di passi, e quindi dopo un numero finito di passi $M_{\cap}(x)$ eseguirà le simulazioni di $M_A(x)$ e $M_B(x)$ per un numero di passi k sufficiente a farle terminare in accettazione.

(c) $A^c \setminus B$: non è necessariamente Turing-riconoscibile. La dimostrazione è analoga a quella del caso (a) ponendo $A = \emptyset$ (quindi $A^c = \Sigma^*$) e $B = \mathcal{A}_{\text{TM}}$.

(d) $A^c \setminus B^c$: non è necessariamente Turing-riconoscibile. Infatti, $x \in A^c \setminus B^c$ se e solo se $(x \in A^c) \wedge (x \notin B^c)$, ossia se e solo se $(x \notin A) \wedge (x \in B)$, ossia se e solo se $x \in B \setminus A$. Quindi, questo caso è del tutto analogo al caso (a).

Esercizio 5 [7] Sia $\mathcal{P}_{\text{TM}} = \{\langle M \rangle \mid M \text{ è una macchina di Turing deterministica che si ferma su input } x \text{ entro } |x|^{|x|} \text{ passi}\}$. Il linguaggio \mathcal{P}_{TM} è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Osserviamo innanzi tutto che il Teorema di Rice non può essere applicato, perché la proprietà che caratterizza \mathcal{P}_{TM} non è specifica del linguaggio riconosciuto dalle DTM. Infatti è facile pensare ad una DTM che su input di dimensione 1 termina sempre accettando in un solo passo, ed un'altra DTM equivalente alla prima che però termina con un passo in più. Ora entrambe le DTM riconoscono lo stesso linguaggio ma solo la codifica della prima appartiene a \mathcal{P}_{TM} .

Usando una macchina di Turing universale è possibile decidere se una DTM si ferma entro un numero prefissato di passi, anche dipendente dalla lunghezza dell'input. Tuttavia, per l'appartenenza a \mathcal{P}_{TM} si dovrebbe determinare se una data DTM M si ferma entro il limite di passi dipendente dall'input per tutti i possibili input di lunghezza arbitraria. Questo ci porta a ipotizzare che \mathcal{P}_{TM} non sia un problema decidibile.

Supponiamo per assurdo che \mathcal{P}_{TM} sia decidibile; dunque esiste un decisore D che, per ciascuna DTM M decide in tempo $t(|\langle M \rangle|)$ se $\langle M \rangle \in \mathcal{P}_{\text{TM}}$. Consideriamo allora la seguente DTM T costruita modificando D :

$T =$ “On input $\langle x \rangle$, where x is a string:

1. Compute the length $n = |x|$ of the input string x
2. If $n \leq K$ (see below), then halts
3. Get, via Recursion Theorem, its own description $\langle T \rangle$
4. Run D on $\langle T \rangle$
5. If $D(\langle T \rangle)$ accepts, then
6. enter an endless loop.”

È immediato verificare l'esistenza della DTM T se si ammette l'esistenza della DTM D . Si osservi che il numero di step elementari eseguiti da T dal passo 3 in poi è in alternativa infinito (se viene eseguito il passo 6) oppure pari ad una costante determinata dal numero di step elementari $t(|\langle T \rangle|)$ del decisore D e dal numero di step dei passi 3 e 5. Inoltre, il calcolo della lunghezza della stringa x ed la successiva istruzione condizionale possono essere eseguiti banalmente in tempo $O(|x|)$. Pertanto, se il passo 6 non viene eseguito, l'esecuzione di $T(x)$ termina in un numero di passi non superiore a $c|x| + h$, per opportune costanti c e h . La costante K nell'algoritmo è un qualunque valore tale che $K^K > cK + h$.

Consideriamo dunque l'esecuzione di T su una qualunque stringa x . Se $|x| \leq K$, allora $T(x)$ termina in meno di $c|x| + h < |x|^{|x|}$ step al passo 2. Se invece $|x| > K$, allora $T(x)$ in alternativa termina in meno di $c|x| + h < |x|^{|x|}$ step, oppure non termina affatto.

Più precisamente, sia $|x| > K$, e supponiamo che il passo 5 verifichi che $D(\langle T \rangle)$ ha accettato. Di conseguenza viene eseguito il ciclo senza fine del passo 6. Pertanto esiste un input x per il quale T non termina entro $|x|^{|x|}$ passi, e quindi $T \notin \mathcal{P}_{\text{TM}}$. Ma ciò è contraddetto dal decisore $D(\langle T \rangle)$ che invece ha accettato.

Supponiamo al contrario che il passo 5 verifichi che $D(\langle T \rangle)$ ha rifiutato: in questo caso l'esecuzione di $T(x)$ termina con un numero totale di passi inferiore a $|x|^{|x|}$. Pertanto, qualunque sia la stringa di input x , $T(x)$ termina con un numero di step inferiore a $|x|^{|x|}$, e quindi per definizione $T \in \mathcal{P}_{\text{TM}}$. Questo però è una contraddizione, perché il decisore D per \mathcal{P}_{TM} ha rifiutato $\langle T \rangle$.

In ogni caso si arriva dunque ad una contraddizione; ne consegue che non può esistere un decisore D per \mathcal{P}_{TM} .

Esercizio 6 [7.5] Si consideri il linguaggio $J = \{(R_1, R_2) \mid R_1 \text{ e } R_2 \text{ sono espressioni regolari tali che } L(R_1) \neq L(R_2)\}$. In altri termini, le istanze-sì del linguaggio sono coppie di espressioni regolari che generano linguaggi differenti. Dimostrare che il linguaggio J è NP-hard.

Soluzione: Dimostriamo che J è NP-hard esibendo una riduzione polinomiale dal problema SAT. Consideriamo come istanza una formula CNF $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, ove ogni C_i è la disgiunzione di letterali (variabili o negazione di variabili). Sia $\text{Var}(\Phi) = \{x_1, x_2, \dots, x_n\}$ l'insieme di tutte le variabili booleane incluse in Φ . La riduzione polinomiale trasforma Φ in una istanza di J (R_1, R_2), ove:

- L'espressione regolare R_1 è

$$R_1 = \underbrace{(0 \cup 1)(0 \cup 1) \cdots (0 \cup 1)}_{n \text{ volte}},$$

ove n è il numero di variabili in Φ . Ne consegue che $L(R_1) = \{0, 1\}^n$, ossia l'insieme di tutte le possibili assegnazioni di verità alle variabili di Φ .

- L'espressione regolare R_2 è

$$R_2 = S_1 \cup S_2 \cup \cdots \cup S_m,$$

ove S_i è derivato dalla clausola C_i di Φ ($1 \leq i \leq m$) in base alle variabili occorrenti in C_i . In particolare, $S_i = (S_i^1 \ S_i^2 \ \cdots \ S_i^n)$ con

$$S_i^k = \begin{cases} \emptyset & \text{se sia } x_k \text{ che } \overline{x_k} \text{ appaiono in } C_i \\ 0 & \text{se } x_k \text{ appare in } C_i \\ 1 & \text{se } \overline{x_k} \text{ appare in } C_i \\ (0 \cup 1) & \text{se né } x_k \text{ né } \overline{x_k} \text{ appaiono in } C_i. \end{cases}$$

S_i genera dunque tutte le stringhe in $\{0, 1\}^n$ che corrispondono ad assegnazioni di verità alle n variabili che rendono *falsa* la disgiunzione di letterali della clausola C_i . Infatti, se la clausola contiene sia una variabile x_k che la sua negazione, allora sarà sempre vera, dunque $S_i = \emptyset$ (poiché $S_i^k = \emptyset$). Se invece una variabile non appare nella clausola, il suo valore non influisce sul valore della clausola, quindi entrambe le assegnazioni 0 e 1 sono permesse per falsificare la clausola. Se infine la variabile appare una volta sola nella clausola, il corrispondente bit nella sequenza generata da S_i rende il corrispondente letterale falso.

Supponiamo che Φ sia soddisfacibile, e dunque esista una assegnazione di verità che renda vera tutte le clausole C_i di Φ . La corrispondente stringa di bit $w \in \{0, 1\}^n$ non può far parte di $L(S_i)$, perché $L(S_i)$ include tutte e sole le stringhe che rendono falsa la clausola C_i , per ogni i da 1 a m . Poiché $L(R_2) = \bigcup_{i=1}^m L(S_i)$, $w \notin L(R_2)$, e dunque $L(R_2) \neq \{0, 1\}^n = L(R_1)$. Pertanto, (R_1, R_2) è una istanza-sì di J .

Al contrario, supponiamo che $(R_1, R_2) \in J$, ove R_1 e R_2 derivano da una formula CNF Φ come sopra descritto. Poiché $L(R_2) \neq L(R_1) = \{0, 1\}^n$, esiste una stringa $w \in \{0, 1\}^n$ tale che $w \notin L(R_2)$. Poiché $L(R_2) = \bigcup_{i=1}^m L(S_i)$, $w \notin L(S_i)$ per ogni i da 1 a m . Pertanto l'assegnazione di verità corrispondente a w rende vere tutte le clausole C_i , per $1 \leq i \leq m$, e dunque rende vera Φ . Pertanto Φ è soddisfacibile.

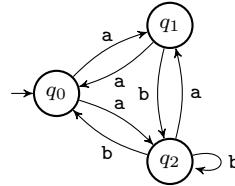
È immediato verificare che la trasformazione da Φ a (R_1, R_2) è eseguibile in tempo polinomiale in $|\Phi|$, e dunque costituisce una riduzione polinomiale tra SAT ed il linguaggio J . Dunque J è NP-hard.

Automi e Linguaggi (M. Cesati)

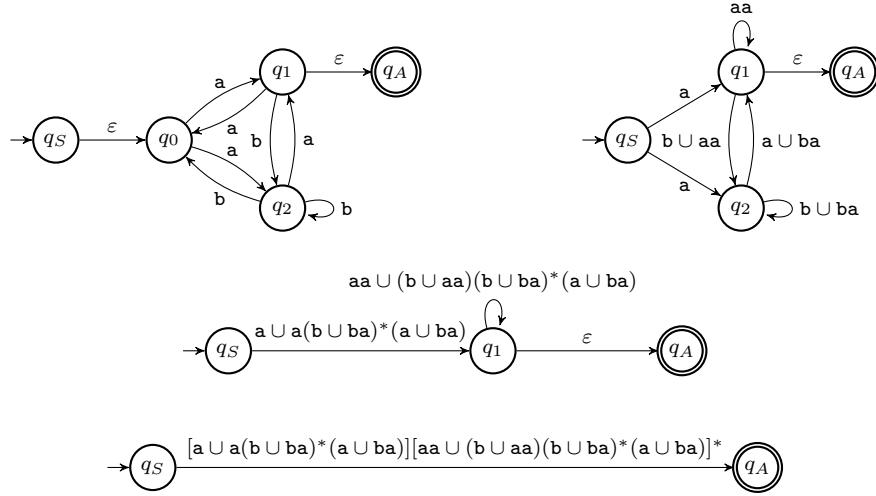
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 13 febbraio 2024

Esercizio 1 [6.5] Determinare una espressione regolare (REX) per il linguaggio riconosciuto dal seguente NFA:



Soluzione: Determiniamo un GNFA equivalente al NFA dato e rimuoviamo, nell'ordine, i nodi q_0 , q_2 e q_1 :



Una REX che genera il linguaggio riconosciuto dal NFA è dunque

$$[a \cup a(b \cup ba)^*(a \cup ba)][aa \cup (b \cup aa)(b \cup ba)^*(a \cup ba)]^*.$$

Esercizio 2 [6] Sia A il linguaggio riconosciuto dalla REX $1(0 \cup 1)^*1$. Sia

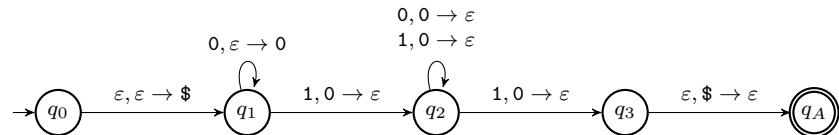
$$B = \{w \in \{0, 1\}^* \mid w = 0^n v, v \in A, |v| = n, n > 0\}.$$

Dimostrare che il linguaggio B non è regolare.

Soluzione: Dimostriamo che B non è regolare. Supponiamo per assurdo che lo sia, e dunque che si possa applicare il Pumping Lemma. Sia $p > 0$ la “pumping length” per B , e consideriamo la stringa $s = 0^p 1^p$. È immediato verificare che $s \in B$ e che $|s| \geq p$. Deve quindi esistere una suddivisione $s = xyz$ con $|y| > 0$, $|xy| \leq p$ e $xy^i z \in B$ per ogni $i \geq 0$. Qualunque sia tale suddivisione, y contiene solo zeri (poiché $|xy| \leq p$) ed almeno uno zero (poiché $|y| > 0$). Consideriamo quindi il caso $i = 0$: la stringa $xy^0 z = xz$ contiene meno zeri di s mentre il numero di ‘1’ è invariato, quindi ha la forma $0^q 1^p$ con $q < p$: pertanto $xz \notin B$. Perciò assumere che B sia regolare porta ad una contraddizione.

Esercizio 3 [6.5] Si consideri il linguaggio B come nell’esercizio precedente. Si determini un automa a pila (PDA) P tale che $L(P) = B$, oppure dimostrare che tale PDA non esiste.

Soluzione: La struttura degli elementi del linguaggio B è molto semplice: le stringhe sono costituite da un numero $n > 0$ di ‘0’, seguiti da una stringa che inizia e termina con ‘1’ e lunga esattamente n simboli. Il progetto del PDA è quindi consequenziale: si comincia scrivendo il simbolo di “fine stack”, poi si accumulano sullo stack gli ‘0’ iniziali. Appena si legge ‘1’ in input si comincia a leggere la seconda parte della stringa rimuovendo per ogni carattere letto un simbolo dallo stack. Per terminare, si indovina non deterministicamente la posizione dell’ultimo ‘1’ della stringa, e si accetta se lo stack non contiene più alcun ‘0’.



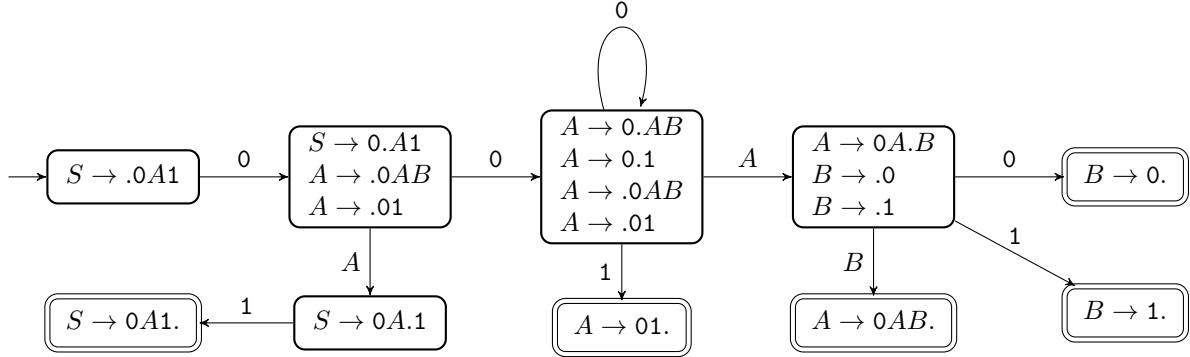
Se la scelta non deterministica effettuata nello stato q_2 risulta sbagliata, il ramo di computazione deterministica non può accettare in quanto non può essere letto alcun altro simbolo della stringa di ingresso. Per lo stesso motivo, se la stringa non termina con ‘1’ l’automa non può accettare. Se gli zeri iniziali sono in numero maggiore della restante parte della stringa, non è possibile rimuovere tutti gli ‘0’ accumulati sullo stack e quindi non si può arrivare allo stato di accettazione. Se infine il numero di zeri iniziali è inferiore alla restante parte della stringa, non è possibile consumare tutti i simboli in ingresso perché non esiste transizione da q_2 che legge un simbolo in ingresso senza contemporaneamente leggere uno ‘0’ dallo stack. Pertanto questo PDA riconosce esattamente il linguaggio B .

Esercizio 4 [6] Si consideri la grammatica G con variabile iniziale S avente le regole:

$$S \rightarrow 0A1 \quad A \rightarrow 0AB \mid 01 \quad B \rightarrow 0 \mid 1$$

Determinare se G è una grammatica deterministica.

Soluzione: Per verificare se la grammatica G è deterministica utilizziamo il DK-test.



Poiché nessuno stato di accettazione include due regole tra loro consistenti (due regole terminate oppure una regola terminata ed un'altra regola non terminata in cui il dot precede un simbolo terminale), il DK-test ha successo e quindi la grammatica G è deterministica.

Esercizio 5 [7] Sia A un linguaggio Turing-riconoscibile (ossia ricorsivamente enumerabile) e B un linguaggio decidibile. Sia $X \setminus Y = \{x \in X \mid x \notin Y\}$. Quali tra i quattro linguaggi $A \setminus B$, $A \setminus B^c$, $A^c \setminus B$, $A^c \setminus B^c$ è Turing riconoscibile? Giustificare le risposte con dimostrazioni o controesempi.

Soluzione: Supponiamo, senza perdita di generalità, che A e B siano linguaggi definiti su di un comune insieme alfabeto Σ . Osserviamo subito che, poiché B è per ipotesi decidibile, allora anche il suo complemento B^c deve essere decidibile. Pertanto, i quattro casi del testo dell'esercizio si riducono ai seguenti due, in quanto la risposta per $A \setminus B$ vale anche a $A \setminus B^c$, e la risposta per $A^c \setminus B$ vale anche per $A^c \setminus B^c$.

(a) $A \setminus B$: questo linguaggio è Turing-riconoscibile. Infatti, sia M_A la DTM che riconosce il linguaggio A , e sia M_B la DTM che decide il linguaggio B . Da esse si può derivare la seguente DTM:

M = “On input x , where x is a string in Σ^* :

1. run M_A on input x
2. if $M_A(x)$ rejected, then reject
3. run M_B on input x
4. if $M_B(x)$ rejected, then accept
5. reject”

Supponiamo che $x \in A \setminus B$; poiché M_A riconosce A , $M_A(x)$ termina in un numero finito di passi accettando. Pertanto M arriva ad eseguire il passo 3. Poiché M_B decide B , $M_B(x)$ termina in un numero finito di passi e rifiuta, in quanto $x \notin B$. Perciò $M(x)$ accetta al passo 4. D'altra parte, se $x \notin A \setminus B$, allora $M(x)$ potrebbe non terminare, in quanto non si

ha alcuna garanzia che il passo 1 termini in un numero finito di passi. Quindi M riconosce (ma non decide) il linguaggio $A \setminus B$.

(b) $A^c \setminus B$: questo linguaggio non è necessariamente Turing-riconoscibile. Per dimostrarlo è sufficiente considerare $A = \mathcal{A}_{\text{TM}}$ (il linguaggio Turing-riconoscibile ma non decidibile di accettazione delle TM) e $B = \emptyset$. Il linguaggio $A^c \setminus B = \mathcal{A}_{\text{TM}}^c \setminus \emptyset = \mathcal{A}_{\text{TM}}^c$ non è Turing-riconoscibile. Infatti, se un linguaggio ed il suo complemento sono Turing-riconoscibili, allora il linguaggio stesso è decidibile; però sappiamo che \mathcal{A}_{TM} non è decidibile.

Esercizio 6 [8] Il problema PCP (Post Correspondence Problem) non è decidibile. Si consideri la restrizione F-PCP di PCP: le sue istanze-sì sono le istanze-sì di PCP che hanno un match con una sequenza di tessere tutte differenti tra loro. F-PCP è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Una generica istanza del problema PCP è una collezione di tessere

$$P = \left\{ \left[\begin{matrix} t_1 \\ b_1 \end{matrix} \right], \left[\begin{matrix} t_2 \\ b_2 \end{matrix} \right], \dots, \left[\begin{matrix} t_n \\ b_n \end{matrix} \right] \right\}$$

con $t_i, b_i \in \Sigma^*$ per ogni i . Una istanza-sì è una collezione P tale che esiste un “match”, ossia una sequenza finita di tessere di P (eventualmente ripetute) per cui la concatenazione delle stringhe superiori delle tessere produce la stessa stringa della concatenazione delle stringhe inferiori. Sappiamo che il problema PCP è indecidibile, sostanzialmente poiché è possibile costruire una istanza di PCP che simula esattamente la computazione di una macchina di Turing.

Nel problema PCP è cruciale che per la costruzione del match si possano utilizzare tessere identiche (ossia ripetute). La riduzione tra \mathcal{A}_{TM} e PCP usa in effetti in modo essenziale tessere identiche nella costruzione del match. Se consideriamo la restrizione F-PCP di PCP in cui le istanze-sì sono caratterizzate da match in cui non appaiono tessere ripetute, la riduzione canonica da \mathcal{A}_{TM} non funziona più; in effetti, è facile mostrare che F-PCP è un problema decidibile.

Consideriamo la seguente DTM T :

T = “On input P , where P is a collection of PCP’s dominos:

1. for $k = 1$ to n :
2. for each disposition π of class k of the elements in P :
3. if π is a PCP match, then accept
4. reject”

La decidibilità di F-PCP risiede nel fatto che il numero di possibili match costituiti da tessere tutte differenti è finito e dipende ovviamente dalla cardinalità n dell’insieme di tessere P nella

istanza del problema. Più precisamente, poiché esistono $D_{n,k} = \frac{n!}{(n-k)!}$ modi di permutare k elementi tratti da un insieme di n elementi (senza ripetizioni), il numero di esecuzioni del passo 3 è $\sum_{k=1}^n D_{n,k}$, che ovviamente è un numero finito. A propria volta, il passo 3 consiste nel verificare se una data permutazione π è un match valido per PCP, ossia se la concatenazione delle k stringhe superiori dei domino in π coincide con la concatenazione delle k stringhe inferiori.

Pertanto T decide ogni istanza di F-PCP, e quindi questo problema è decidibile.

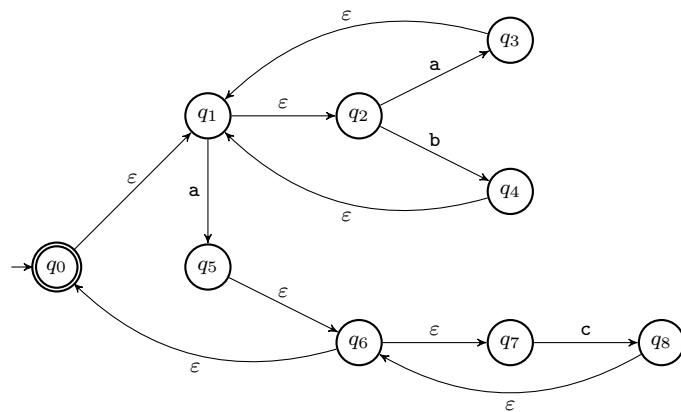
Automi e Linguaggi (M. Cesati)

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 24 giugno 2024

Esercizio 1 [6] Determinare un automa a stati finiti deterministico (DFA) per il linguaggio generato dalla REX $((a \cup b)^*ac^*)^*$.

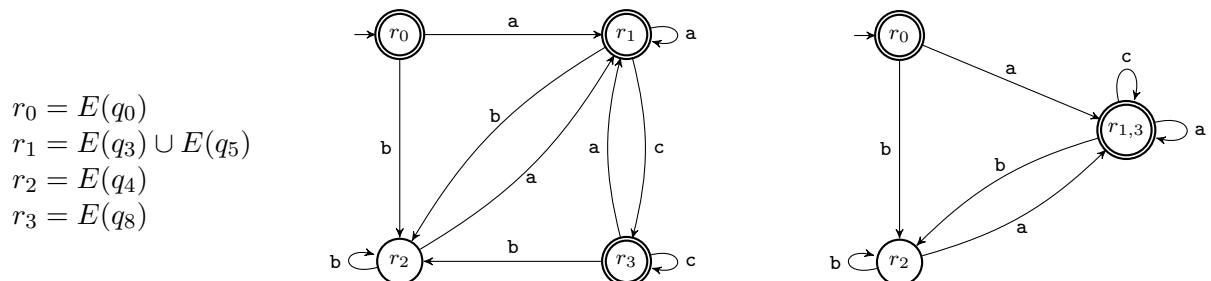
Soluzione: Innanzi tutto determiniamo un automa non deterministico equivalente alla REX data. Utilizzando la costruzione canonica si ottiene:



Le chiusure rispetto alle ϵ -transizioni degli stati di questo automa sono:

$$\begin{aligned}
 E(q_0) &= \{q_0, q_1, q_2\} & E(q_1) &= \{q_1, q_2\} & E(q_2) &= \{q_2\} \\
 E(q_3) &= \{q_1, q_2, q_3\} & E(q_4) &= \{q_1, q_2, q_4\} & E(q_5) &= \{q_0, q_1, q_2, q_5, q_6, q_7\} \\
 E(q_6) &= \{q_0, q_1, q_2, q_6, q_7\} & E(q_7) &= \{q_7\} & E(q_8) &= \{q_0, q_1, q_2, q_6, q_7, q_8\}
 \end{aligned}$$

Adottando la procedura di conversione dall'automa non deterministico a quello deterministico si ottiene l'automa a sinistra qui sotto:



Si può semplificare questo automa osservando che gli stati r_1 e r_3 hanno le stesse transizioni e lo stesso stato di accettazione e quindi possono essere fusi, ottenendo così l'automa a destra. Si osservi anche che r_0 e r_2 , pur avendo le stesse transizioni, non possono essere fusi perché r_0 è di accettazione mentre r_2 non lo è.

Esercizio 2 [6] Si consideri il linguaggio $A = \{w0^h1^k \mid w \in \{\text{a}, \text{b}\}^*, |w| = h+k, \text{ e } |w|_{\text{a}} = h\}$, ove $|w|_{\text{a}}$ indica il numero di simboli ‘a’ contenuti in w . Ad esempio, $\text{bab011} \in A$, $0\text{bab11} \notin A$, $\text{ab00} \notin A$. Determinare se A è un linguaggio libero dal contesto (CFL), giustificando la risposta con una dimostrazione.

Soluzione: Intuitivamente il linguaggio A non è CFL in quanto per decidere se $x \in A$ è necessario contare sia le occorrenze di ‘a’ che di ‘b’ nella prima parte della stringa, e confrontarle rispettivamente con il numero di occorrenze di ‘0’ e di ‘1’ nella seconda parte. Per dimostrarlo formalmente supponiamo per assurdo che A sia CFL, e dunque che per esso valga il Pumping Lemma. Sia dunque $p > 0$ la pumping length, e consideriamo la stringa $s = \text{a}^p\text{b}^p0^p1^p$. Ovviamente $s \in A$ e $|s| = 4p$, dunque deve esistere una suddivisione $s = uvxyz$ tale che $|vy| > 0$, $|vxy| \leq p$ e $uv^i xy^i z \in A$ per ogni $i \geq 0$.

Considerando la stringa vy possono verificarsi soltanto questi tre casi:

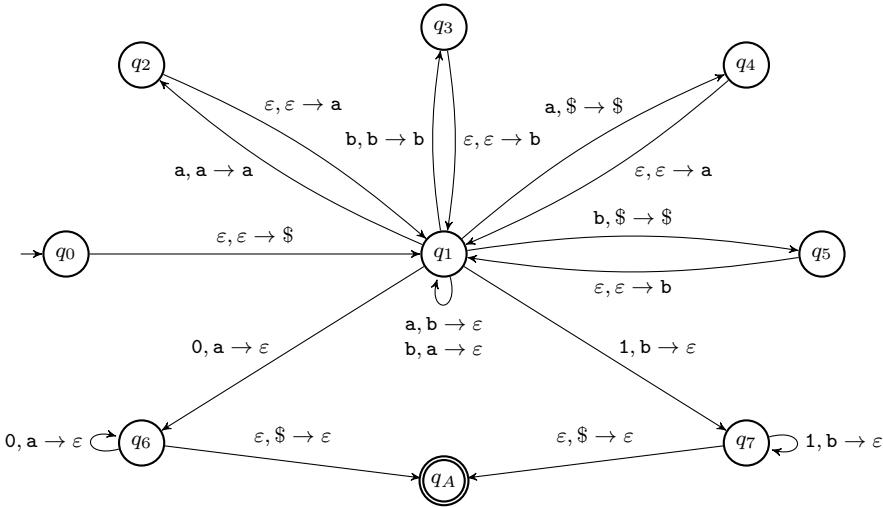
1. In v non sono presenti simboli ‘a’ e ‘b’, ossia tutti i simboli ‘b’ (e quindi ‘a’) cadono entro u . Consideriamo dunque la stringa $uv^i xy^i z$ per qualunque $i \neq 1$: poiché $|vy| > 0$, il numero di ‘0’ e/o ‘1’ deve variare, ma il numero di ‘a’ e ‘b’ rimane lo stesso, pertanto la stringa pompata non può far parte di A .
2. In y non sono presenti simboli ‘0’ e ‘1’, ossia tutti i simboli ‘0’ (e quindi ‘1’) cadono entro z . Consideriamo dunque la stringa $uv^i xy^i z$ per qualunque $i \neq 1$: poiché $|vy| > 0$, il numero di ‘a’ e/o di ‘b’ deve variare, ma il numero di ‘0’ e ‘1’ rimane lo stesso, pertanto la stringa pompata non può far parte di A .
3. In v sono presenti simboli ‘a’ e/o ‘b’ ed in y sono presenti simboli ‘0’ e/o ‘1’. In effetti però, poiché $|vxy| \leq p$, v non può contenere il simbolo ‘a’ e y non può contenere il simbolo ‘1’, ossia tutte le ‘a’ sono in u e tutti gli ‘1’ sono in z . Pertanto la stringa $uv^i xy^i z$ per qualunque $i \neq 1$ modifica il numero di ‘b’ ma non modifica il numero di ‘1’, e modifica il numero di ‘0’ ma non modifica il numero di ‘a’; dunque la stringa pompata non può appartenere ad A .

Poiché ogni possibile suddivisione della stringa s non è pompabile, il Pumping Lemma non vale. Ciò implica che A non può essere CFL.

Esercizio 3 [7] Si considerino i linguaggi $B = \{w0^h \mid w \in \{a, b\}^* \text{ e } |w|_a - |w|_b = h > 0\}$ e $C = \{w1^h \mid w \in \{a, b\}^* \text{ e } |w|_b - |w|_a = h > 0\}$, ove $|w|_x$ indica il numero di occorrenze del carattere x nella stringa w . Determinare un automa a pila deterministico (DPDA) che riconosca il linguaggio $B \cup C$, od in alternativa dimostrare che tale automa non esiste.

Soluzione: Il linguaggio $B \cup C$ contiene elementi costituiti dalla concatenazione di stringhe in $\{a, b\}^*$ e da un numero in unario che rappresenta la differenza del numero di occorrenze dei due simboli nella prima parte. In particolare, se il numero di ‘a’ è maggiore del numero di ‘b’ allora la differenza è codificata in unario utilizzando il simbolo ‘0’; al contrario se il numero di ‘b’ è maggiore del numero di ‘a’ allora la differenza è codificata in unario utilizzando il simbolo ‘1’. Un automa a pila che riconosca $B \cup C$ dovrebbe innanzitutto calcolare la differenza nel numero di simboli nella prima parte della stringa, e poi confrontare tale differenza con il numero di ‘0’ o ‘1’, come appropriato, nella seconda parte. Si osservi che $B \cup C$ non contiene alcuna stringa avente lo stesso numero di ‘a’ e di ‘b’.

Consideriamo dunque il seguente automa a pila P (le transizioni omesse si intendono sempre entranti in uno stato di non accettazione con ogni transizione uscente che rientra sullo stesso stato):



Dimostriamo che $L(P) = B \cup C$. L'automa inizializza lo stack con il simbolo ‘\$’, poi nello stato interno q_1 legge la prima parte della stringa in input, contenente solo caratteri ‘a’ e ‘b’. Supponiamo di leggere dall'input x il carattere ‘a’: vi sono tre possibili casi:

1. Sulla cima dello stack vi è il carattere ‘a’: ciò significa che prima della lettura corrente valeva $|x|_a > |x|_b$; dunque P rimette sullo stack il carattere ‘a’ letto in q_1 e aggiunge tramite q_2 un ulteriore carattere ‘a’: la differenza (ossia il numero di ‘a’ sullo stack) aumenta di una unità.

2. Sulla cima dello stack vi è il carattere ‘\$’: ciò significa che prima della lettura corrente valeva $|x|_a = |x|_b$; dunque P rimette sullo stack il carattere ‘\$’ letto in q_1 e aggiunge tramite q_4 un carattere ‘a’ sullo stack (è stata letta in totale una ‘a’ in più delle ‘b’).
3. Sulla cima dello stack vi è il carattere ‘b’: ciò significa che prima della lettura corrente valeva $|x|_a < |x|_b$; dunque P non aggiunge nuovamente sullo stack il carattere ‘b’ letto in q_1 , in modo da ridurre la differenza tra le ‘b’ e le ‘a’ di una unità.

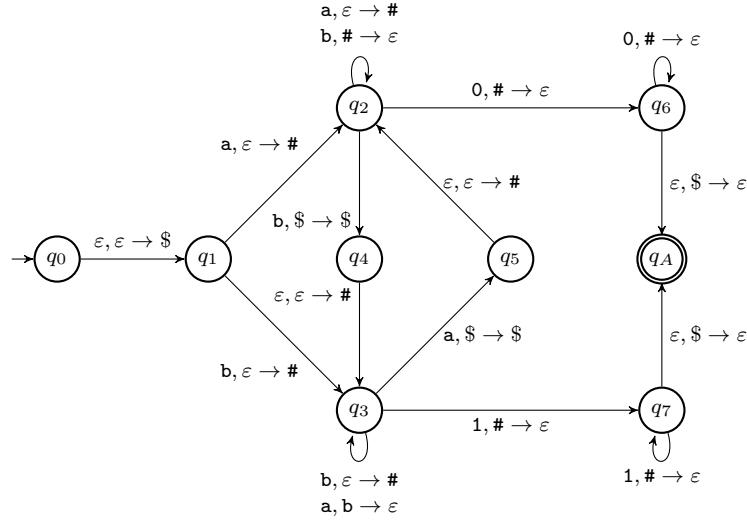
Analogo meccanismo è implementato quando si legge un carattere ‘b’ dall’input, per mezzo degli stati interni q_1 , q_3 e q_5 .

Dal momento in cui P legge un carattere diverso ‘a’ e ‘b’ si lascia lo stato interno q_1 per non rientrarvi più. Di conseguenza, ogni stringa malformata in cui un carattere ‘a’ o ‘b’ segue uno ‘0’ od un ‘1’ non può essere accettata. Similmente, si osservi che qualunque stringa malformata iniziante con ‘0’ o ‘1’ non può essere accettata perché sulla cima dello stack si troverebbe ‘\$’, ma non esistono transizioni da q_1 verso q_6 e q_7 definite quando la cima dello stack contiene ‘\$’; per il medesimo motivo, nessuna stringa che contenga un ugual numero di ‘a’ e ‘b’ può essere accettata.

Se il primo carattere diverso da ‘a’ e ‘b’ è ‘0’, allora la cima dello stack deve contenere ‘a’ (quindi ci si aspetta una eccedenza di ‘a’ in quanto $x \in B$): lo stato q_6 continua a leggere gli ‘0’ nella parte finale di x rimuovendo man mano le ‘a’ dallo stack. Se lo stack contiene ‘\$’ si può entrare nello stato di accettazione q_A , ma la stringa x viene accettata soltanto se P è riuscito a leggere tutti gli ‘0’ nella parte finale, ossia se questi codificano in unario il valore $|x|_a - |x|_b$. Analogo discorso si applica se il primo carattere diverso da ‘a’ e ‘b’ è ‘1’: ci si aspetta che $x \in C$ e che nella parte finale di x vi siano tanti ‘1’ quanti sono i caratteri ‘b’ sullo stack, ossia in quantità pari a $|x|_b - |x|_a$.

Osservando l’automa P è immediato verificare che esso è deterministico. Infatti gli unici stati con più di una transizione uscente sono q_1 , q_6 e q_7 . Tutte le transizioni uscenti da q_1 forzano la lettura di un simbolo in input ed una “pop” dallo stack, quindi simbolo in input e simbolo sulla cima dello stack definiscono deterministicamente la transizione da applicare. Lo stato q_6 ha solo due transizioni uscenti, ma l’applicazione di queste è deterministica perché dipende dal simbolo sulla cima dello stack (se ‘a’ oppure ‘\$’). Analogo discorso vale per lo stato q_7 . Dunque, P è in effetti un DPDA.

Una soluzione alternativa dell’esercizio è costituita dal seguente DPDA. La differenza sostanziale rispetto alla soluzione precedente è che l’informazione su quale simbolo sia in ogni momento eccedente non è memorizzata entro lo stack (il simbolo sulla cima indicante il carattere attualmente eccedente) ma dagli stati interni (q_2 e q_6 per le eccedenze di ‘a’, q_3 e q_7 per le eccedenze di ‘b’).



Esercizio 4 [7] Sia L un linguaggio Turing-riconoscibile (ossia ricorsivamente enumerabile) e sia $L^s = \{x \mid \exists y \in L \text{ tale che } x \text{ è una sottostringa di } y\}$. Dimostrare che L^s è Turing-riconoscibile.

Soluzione: Il linguaggio L^s è costituito da tutte le sottostringhe degli elementi di L . Se ad esempio $abcd \in L$, allora certamente a , b e bc fanno parte di L^s .

Per dimostrare che L^s 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 substring y of x :
4. print y ”

Sia $y \in L^s$; dunque y è una sottostringa di un elemento $x \in L$, ossia $x = wyz$, con w e z eventualmente uguali a ϵ . Poiché E enumera L , dopo un tempo finito genererà l'elemento x di L . Pertanto E' stamperà tutte le sottostringhe di x , e dunque anche y . Viceversa, se E' stampa una stringa y , allora necessariamente tale stringa è una sottostringa di una stringa x stampata da E , e poiché E è un enumeratore per L , $x \in L$. Pertanto $y \in L^s$. Concludiamo che E' è un enumeratore per L^s , e dunque L^s è ricorsivamente enumerabile.

Esercizio 5 [7] Sia $\mathcal{D} = \{\langle M \rangle \mid M \text{ è una macchina di Turing deterministica per la quale esiste una stringa } x \text{ tale che } M(x) \text{ accetta in meno di } |\langle M \rangle| \text{ passi}\}$. In altri termini, $\langle M \rangle \in \mathcal{D}$

se e solo se esiste una stringa x tale che $M(x)$ accetta in un numero di passi inferiore alla lunghezza della codifica di M . \mathcal{D} è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Innanzitutto osserviamo che non è consentito utilizzare il Teorema di Rice, in quanto la proprietà che caratterizza l'appartenenza al linguaggio \mathcal{D} dipende strettamente da una caratteristica della macchina di Turing particolare (la lunghezza della sua codifica) e non è una proprietà del linguaggio associato alla TM. Infatti, potremmo considerare una TM M il cui linguaggio associato è costituito da una singola stringa w e tale che il numero di passi della computazione $M(w)$ è superiore alla lunghezza della codifica $|\langle M \rangle|$. D'altra parte, è sempre possibile ottenere una TM M' aggiungendo alla TM M stati e transizioni inutili in modo da aumentare la dimensione della codifica fino a renderla superiore al numero di passi di $M(w)$. Pertanto avremmo $L(M) = L(M')$, $\langle M \rangle \in \mathcal{D}$ e $\langle M' \rangle \notin \mathcal{D}$.

In effetti il linguaggio \mathcal{D} è decidibile poiché le computazioni associate alla proprietà che definisce il linguaggio sono di lunghezza limitata. Per dimostrarlo formalmente, consideriamo $\langle M \rangle \in \mathcal{D}$. Per definizione esiste dunque una stringa w che è accettata da M con un numero di passi inferiore a $|\langle M \rangle|$. È immediato osservare che se w avesse lunghezza maggiore o uguale a $|\langle M \rangle|$, allora esisterebbe un'altra stringa w' di lunghezza inferiore a $|\langle M \rangle|$ che è accettata da M in meno di $|\langle M \rangle|$ passi. Infatti, solo $|\langle M \rangle| - 1$ simboli di w possono essere letti da M prima di accettare la stringa, quindi qualunque prefisso di w di lunghezza $|\langle M \rangle| - 1$ deve ugualmente essere accettato.

Possiamo dunque considerare un decisore T per \mathcal{D} basato su una TM universale che esegue M per meno di $|\langle M \rangle|$ passi su tutte le stringhe di lunghezza inferiore a $|\langle M \rangle|$. Formalmente, sia S l'insieme di tutte le stringhe sull'alfabeto di input Σ di M di lunghezza inferiore a $|\langle M \rangle|$. Definiamo T come segue:

$T =$ “On input $\langle M \rangle$, where M is a Turing machine:

1. For every string $w \in S$:
2. Run $M(w)$ for at most $|\langle M \rangle| - 1$ steps
3. If $M(w)$ accepts, then accept
4. Reject”

L'insieme S contiene $\sum_{k=0}^{|\langle M \rangle|-1} |\Sigma|^k$ stringhe, quindi il ciclo esterno viene eseguito un numero finito di volte. In ogni iterazione la simulazione di $M(w)$ viene interrotta dopo $|\langle M \rangle| - 1$ passi, quindi $T(\langle M \rangle)$ termina sempre. Se $T(\langle M \rangle)$ accetta, allora esiste una stringa w che è accettata da M in un numero di passi inferiore a $|\langle M \rangle|$, e quindi per definizione $\langle M \rangle \in \mathcal{D}$. Viceversa, se $T(\langle M \rangle)$ rifiuta, allora per tutte le stringhe $w \in S$ la computazione $M(w)$ non accetta entro $|\langle M \rangle| - 1$ passi. Per quanto osservato precedentemente non può esistere alcuna stringa $w \in \Sigma^*$ che è accettata da M in meno di $|\langle M \rangle|$ passi, pertanto $\langle M \rangle \notin \mathcal{D}$. Dunque, la TM T è un decisore per il linguaggio \mathcal{D} .

Esercizio 6 [7] Sia $\mathcal{L} = \{\langle M \rangle \mid M \text{ è una macchina di Turing deterministica tale che esiste almeno una stringa di input } x \text{ per la quale } M(x) \text{ non termina}\}$. \mathcal{L} è decidibile? Giustificare la risposta con una dimostrazione.

Soluzione: Osserviamo innanzi tutto che non è possibile ricorrere al Teorema di Rice per risolvere l'esercizio, poiché la proprietà che caratterizza il linguaggio \mathcal{L} non è propria del linguaggio delle macchine di Turing che appartengono a \mathcal{L} . Infatti, sia M una macchina di Turing che termina su ogni input e rifiuta almeno una stringa \bar{x} . È facile derivare da M una TM M' che per ogni input x dapprima controlla se $x = \bar{x}$, ed in tal caso entra in un ciclo senza fine; altrimenti, se $x \neq \bar{x}$, $M'(x)$ simula $M(x)$. Pertanto $L(M) = L(M')$, ma $\langle M \rangle \notin \mathcal{L}$ mentre $\langle M' \rangle \in \mathcal{L}$.

Per dimostrare che \mathcal{L} è non decidibile è sufficiente mostrare una riduzione da un problema indecidibile; è naturale prendere in considerazione \mathcal{A}_{TM} . Possiamo in effetti mostrare che $\mathcal{A}_{\text{TM}} \leq_T \mathcal{L}$ tramite la seguente TM con oracolo \mathcal{L} che decide \mathcal{A}_{TM} :

$M^{\mathcal{L}} = \text{"On input } \langle T, x \rangle, \text{ where } T \text{ is a TM and } x \text{ is a string:}$

1. Build from $\langle T, x \rangle$ the following TM:

$M' = \text{"On input } y, \text{ where } y \text{ is a string:}$

- (a) If $y \neq x$, then halt
- (b) Run T on input x "
- (c) Halt"

2. Ask the oracle whether $\langle M' \rangle \in \mathcal{L}$
3. If the answer is YES, reject
4. Otherwise, if the answer is NO, run T on input x
5. If $T(x)$ accepts, then accept; otherwise, reject"

Sia $\langle T, x \rangle \in \mathcal{A}_{\text{TM}}$; dunque $M'(y)$ termina sempre (al passo (a) se $y \neq x$, al passo (c) se $y = x$). Perciò $\langle M' \rangle \notin \mathcal{L}$, e quando la computazione $M^{\mathcal{L}}(\langle T, x \rangle)$ interroga l'oracolo, questo risponderà NO. Dunque $M^{\mathcal{L}}(\langle T, x \rangle)$ simula $T(x)$ ed accetta poiché $T(x)$ accetta. Supponiamo invece che $\langle T, x \rangle \notin \mathcal{A}_{\text{TM}}$. Esistono due casi: $T(x)$ rifiuta e $T(x)$ entra in un ciclo senza fine senza terminare. Nel caso in cui $T(x)$ rifiuta, l'oracolo risponde NO perché $T(x)$ termina, quindi non esiste alcun input sul quale M' non termina. $M^{\mathcal{L}}(\langle T, x \rangle)$ arriva dunque a simulare $T(x)$ e, quando quest'ultima computazione termina rifiutando, rifiuta. Se infine $T(x)$ non termina, $M'(x)$ non termina, dunque $\langle M' \rangle \in \mathcal{L}$ e l'oracolo risponde YES; perciò $M^{\mathcal{L}}(\langle T, x \rangle)$ rifiuta al passo 3. Riassumendo, $M^{\mathcal{L}}$ decide \mathcal{A}_{TM} , e quindi $\mathcal{A}_{\text{TM}} \leq_T \mathcal{L}$; poiché \mathcal{A}_{TM} è indecidibile, resta dimostrato che anche \mathcal{L} è indecidibile.

Automi e Linguaggi (M. Cesati)

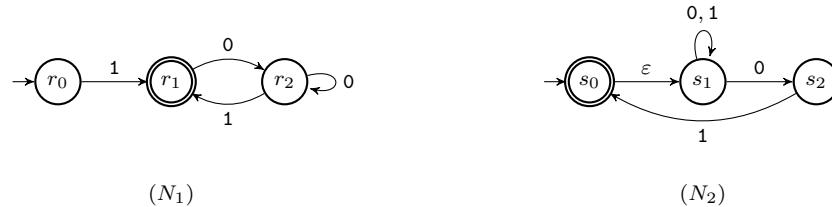
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 16 luglio 2024

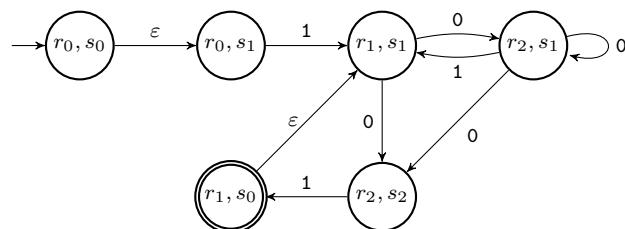
Esercizio 1 [7] Si considerino le REX $R_1 = 1(0^*01)^*$ e $R_2 = [(0 \cup 1)^*01]^*$. Determinare una REX per il linguaggio $L(R_1) \cap L(R_2)$.

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 .

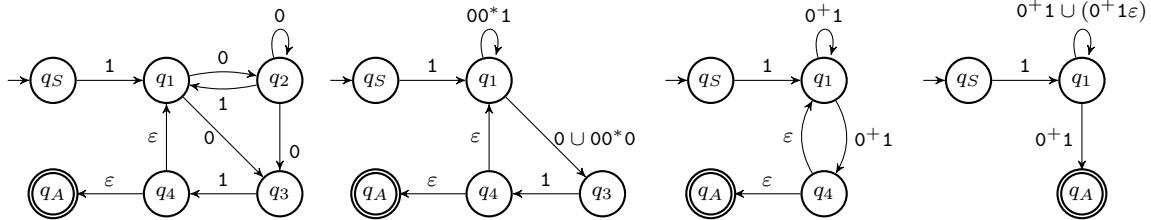
Determiniamo innanzi tutto gli NFA N_1 e N_2 per i linguaggi $L(R_1)$ e $L(R_2)$. Semplificando gli automi derivati meccanicamente si ottengono:



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 $1(0^+1)^+$.

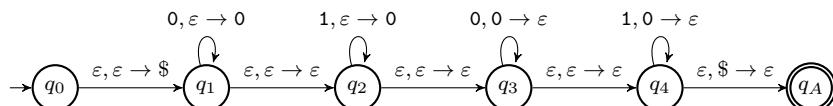
Una REX diversa, ma ovviamente equivalente, si ottiene trasformando gli NFA iniziali in DFA prima di determinare l'automa intersezione. In particolare, un DFA equivalente a N_2 ed il DFA intersezione risultante dall'intersezione con N_1 (già un DFA) sono i seguenti:



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

Esercizio 2 [6.5] Si consideri il linguaggio $A = \{0^h 1^k 0^l 1^m \mid h, k, l, m \geq 0 \text{ e } h + k = l + m\}$. 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 $w = 0^h 1^k 0^l 1^m$ 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 $w = 0^h \dots$. 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 $w = 0^h 1^k \dots$; contemporaneamente 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 $w = 0^h 1^k 0^l \dots$; 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 $w = 0^h 1^k 0^l 1^m \dots$; 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, $w \in A$.

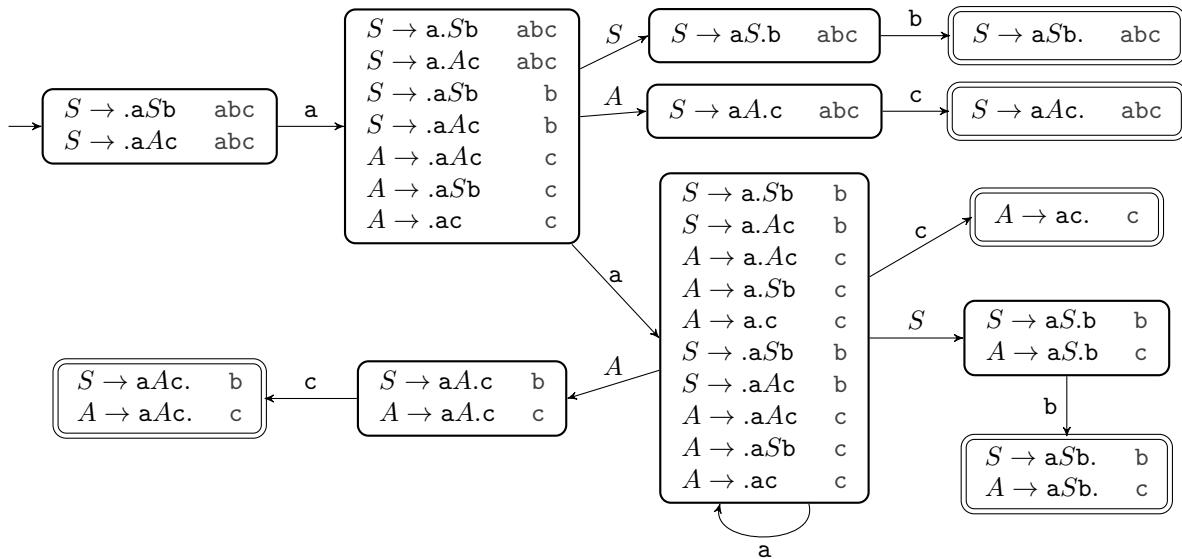
Avendo dimostrato che $A \subseteq L(P) \subseteq A$, ne consegue che $A = L(P)$. Pertanto A è un linguaggio libero dal contesto.

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

$$S \rightarrow aSb \mid aAc, \quad A \rightarrow aAc \mid aSb \mid ac$$

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

Soluzione: Per verificare se G è LR(1) eseguiamo il DK₁-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 DK₁-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 $A \setminus B = \{x \in A \mid x \notin B\}$. 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'appartenenza 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}_{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}_{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}_{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.