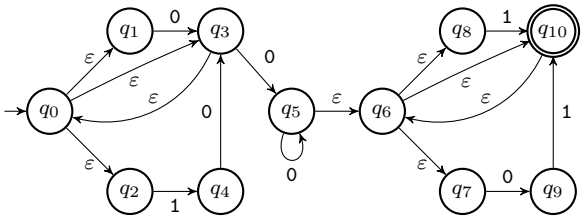
Automi e Linguaggi **(M. Cesati)**

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 22 febbraio 2021

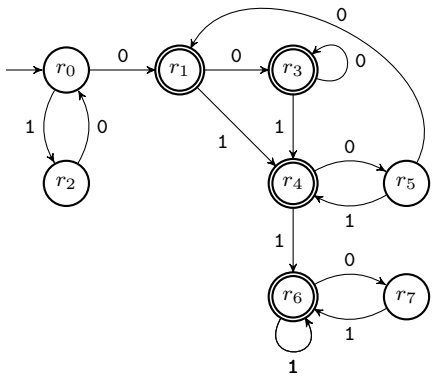
Esercizio 1 [6] Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $(\mathtt{0} \cup \mathtt{10})^*\mathtt{00}^*(\mathtt{1} \cup \mathtt{01})^*$.

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:

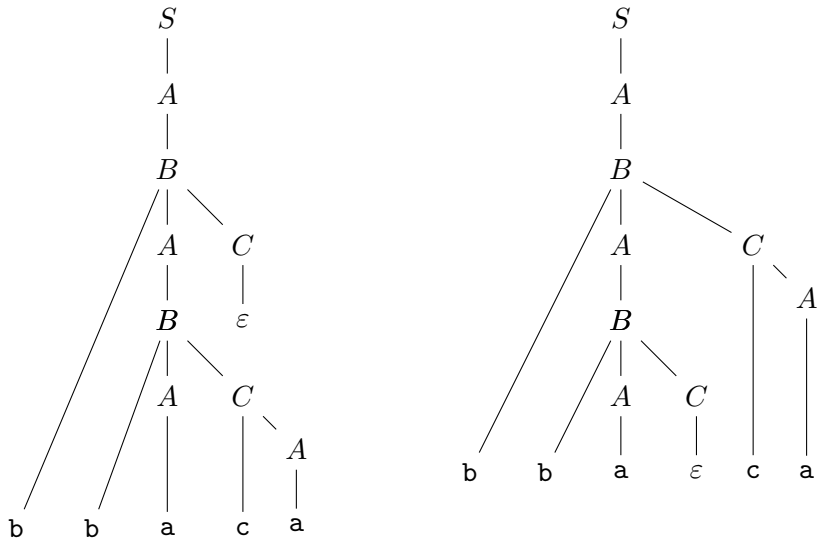
$$S \to A \qquad A \to \mathtt{a}A \mid B \mid \mathtt{a} \qquad B \to \mathtt{b}AC \qquad C \to \mathtt{c}A \mid \varepsilon$$

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":

$$S \to A \to B \to \texttt{b}AC \to \texttt{b}BC \to \texttt{bb}ACC \to \texttt{bba}CC \to \texttt{bbac}AC \to \texttt{bbaca}C \to \texttt{bbaca}\varepsilon$$

$$S \to A \to B \to \texttt{b}AC \to \texttt{b}BC \to \texttt{bb}ACC \to \texttt{bba}CC \to \texttt{bba}\varepsilon C \to \texttt{bbac}A \to \texttt{bbaca}$$

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 $L = \{\texttt{0}^n\,\texttt{1}\,\texttt{0}^m\,\texttt{1}\,\texttt{0}^q \mid n, m > 0, q = \max(n, m)\}$ 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 "pompate" verso l'alto e verso il basso.

Consideriamo la stringa $s = \texttt{0}^p\,\texttt{1}\,\texttt{0}^p\,\texttt{1}\,\texttt{0}^p \in L$, e dimostriamo che non è possibile determinare una suddivisione $s = uvxyz$ con $|vy| > 0$ e $|vxy| \leq p$ tale che $uv^ixy^iz \in L$ 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 $\mathtt{0}^s\,\mathtt{1}\,\mathtt{0}^p\,\mathtt{0}^p$ (ovvero $\mathtt{0}^p\,\mathtt{1}\,\mathtt{0}^s\,\mathtt{1}\,\mathtt{0}^p$) con $s > p$; poiché $p \neq \max(s, p)$, $\mathtt{0}^s\,\mathtt{1}\,\mathtt{0}^p\,\mathtt{1}\,\mathtt{0}^p \notin L$.

3. La stringa vxy include sia zeri nella sequenza più a sinistra che zeri nella sequenza centrale (considerando il caso 1, necessariamente $x = \mathtt{1}$). Pompando verso l'alto o verso il basso la stringa diventa $\mathtt{0}^s\,\mathtt{1}\,\mathtt{0}^t\,\mathtt{1}\,\mathtt{0}^p$, 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 $0^p\,\mathtt{1}\,0^s\,\mathtt{1}\,0^t$ 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 $\mathtt{0}^p\,\mathtt{1}\,\mathtt{0}^p\,\mathtt{1}\,0^t$ 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 $\mathrm{HALT_E}$, 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:

$$\mathrm{HALT} = \mathrm{HALT_E} \cup \mathrm{HALT_O}.$$

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 $\langle T \rangle \in \text{HALT}_\text{O}$. 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 $\langle T \rangle \notin \text{HALT}_\text{O}$. 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 $\mathcal{I} = \{(R_1, R_2) \mid R_1 \text{ e } R_2 \text{ sono espressioni regolari senza "}*\text{" tali che } L(R_1) \neq L(R_2)\}$. 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 $\mathcal{I} \in$ 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 $\Phi = C_i \wedge C_2 \wedge \cdots \wedge C_m$, ove ogni C_i è la disgiunzione di letterali (variabili o negazione di variabili). Sia $\text{Var}(\Phi) = \{x_1, x_2, \ldots, x_n\}$ 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

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

  ove n è il numero di variabili in Φ. Ne consegue che $L(R_1) = \{\mathtt{0}, \mathtt{1}\}^n$, 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

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

S_i genera dunque tutte le stringhe in $\{\mathtt{0}, \mathtt{1}\}^n$ 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 $w \in \{\mathtt{0}, \mathtt{1}\}^n$ 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 $L(R_2) \neq \{\mathtt{0}, \mathtt{1}\}^n = L(R_1)$. 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 $L(R_2) \neq L(R_1) = \{\mathtt{0}, \mathtt{1}\}^n$, esiste una stringa $w \in \{\mathtt{0}, \mathtt{1}\}^n$ 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.