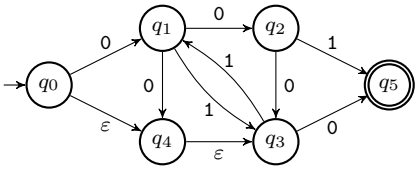
Automi e Linguaggi **(M. Cesati)**

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

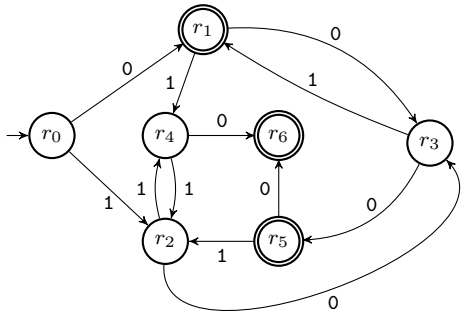
Compito scritto del 24 gennaio 2024

Esercizio 1 [6] Determinare un automa a stati finiti deterministico (DFA) equivalente al seguente automa non deterministico:

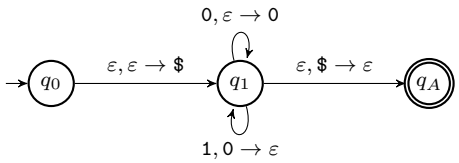


Soluzione: È conveniente determinare preventivamente la chiusura degli stati del NFA rispetto alle ε-transizioni: per tutti gli stati la chiusura coincide con lo stato stesso, tranne che per $E(q_0) = \{q_0, q_4, q_3\}$ e $E(q_4) = \{q_4, q_3\}$. Si ottiene alla fine del procedimento:

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



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 $s = \varepsilon$);
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, $\texttt{001101} \in A$. Invece, la stringa $\overline{s} = \texttt{001110}$ non fa parte del linguaggio A in quanto la sottostringa $s_5 = 00111$ ha una eccedenza di simboli '1'. La stringa $\overline{s}$ 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.

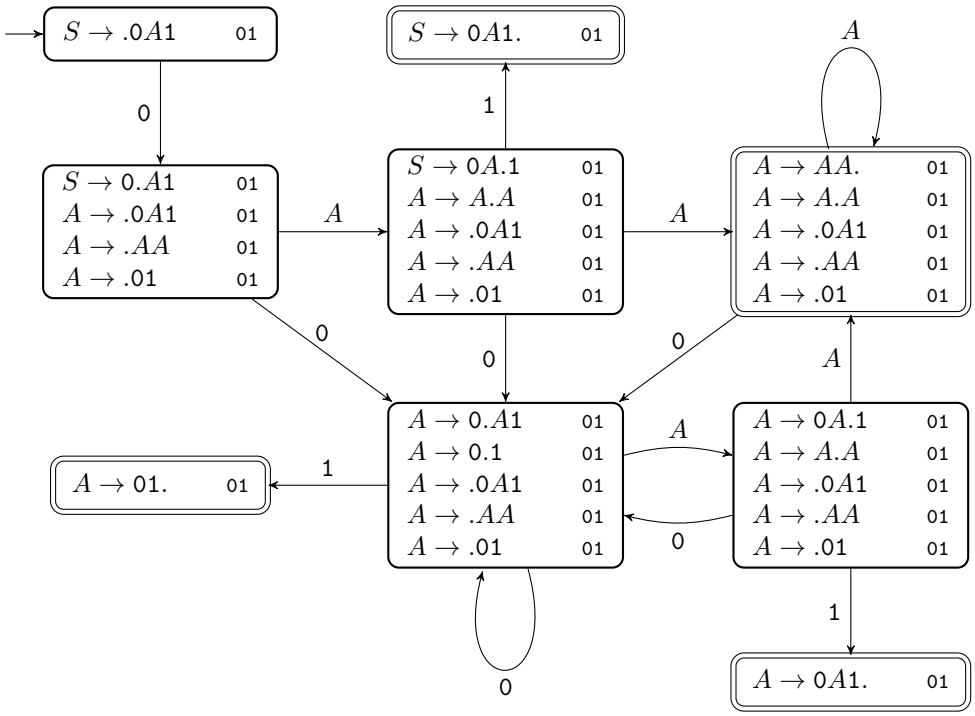
Dimostriamo che A non è regolare. Supponiamo per assurdo che lo sia, e dunque che si possa applicare il Pumping Lemma. Sia $p > 0$ la "pumping length" per A, e consideriamo la stringa $s = 0^p1^p$. È immediato verificare che $s \in A$ e che $|s| \geq p$. Deve quindi esistere una suddivisione $s = xyz$ con $|y| > 0$, $|xy| \leq p$ e $xy^iz \in A$ 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 $xy^0z = xz$ 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:

$$S \to \texttt{0}A\texttt{1} \quad A \to \texttt{0}A\texttt{1} \mid AA \mid \texttt{01}.$$

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

Soluzione: Per verificare se la grammatica G è LR(1) utilizziamo il DK_1-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 $S \to \texttt{0}.A\texttt{1}$ 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 $A \to .AA$, 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 $A \setminus B = \{x \in A \,|\, x \notin B\}$. 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 $\mathcal{A}_{\text{TM}}$, ossia coincide con $\mathcal{A}^c_{\text{TM}}$. Se $A \setminus B$ fosse Turing-riconoscibile, allora sia $\mathcal{A}_{\text{TM}}$ che il suo complemento sarebbero Turing-riconoscibili, e quindi $\mathcal{A}_{\text{TM}}$ 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 $k = 0 \to \infty$ 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 $\mathcal{P}_{\text{TM}} = \{\langle M \rangle \mid M$ è una macchina di Turing deterministica che si ferma su input x entro $|x|^{|x|}$ passi $\}$. Il linguaggio $\mathcal{P}_{\text{TM}}$ 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 $\mathcal{P}_{\text{TM}}$ 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 $\mathcal{P}_{\text{TM}}$.

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 $\mathcal{P}_{\text{TM}}$ 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 $\mathcal{P}_{\text{TM}}$ non sia un problema decidibile.

Supponiamo per assurdo che $\mathcal{P}_{\text{TM}}$ 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 $K^K > c\,K + h$.

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 $\mathcal{P}_{\text{TM}}$ ha rifiutato $\langle T\rangle$.

In ogni caso si arriva dunque ad una contraddizione; ne consegue che non può esistere un decisore D per $\mathcal{P}_{\text{TM}}$.

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 $\Phi = C_i \wedge C_2 \wedge \cdots \wedge C_m$, 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 J (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 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 $w \in \{\mathtt{0}, \mathtt{1}\}^n$ 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 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 $L(R_2) \neq L(R_1) = \{\mathtt{0},\mathtt{1}\}^n$, 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 J. Dunque J è NP-hard.