***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 $w = w_1 w_2 \cdots w_{2m} \in A$, considerare la stringa $w^{\#} = w_1 w_3 \cdots w_{2m-1} w_2 w_4 \cdots w_{2m}$. 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: $\{0^n 1^n \mid n \geq 0\}$): 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.

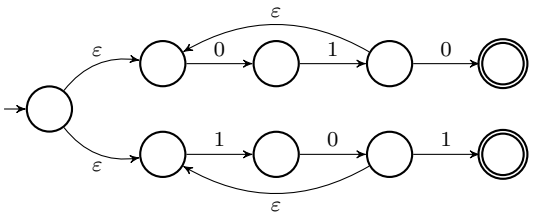
Consideriamo dunque il linguaggio regolare $A = L(R)$, ove R è l'espressione regolare $(01)^*$. Naturalmente $A = \{(01)^n \mid n \geq 0\}$, pertanto è immediato verificare che $A^{\#} = \{0^n 1^n \mid n \geq 0\}$. D'altra parte, $A^{\#}$ non è regolare, come dimostrato durante lo svolgimento del corso.

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 $s = 0^p 1^p$, 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.

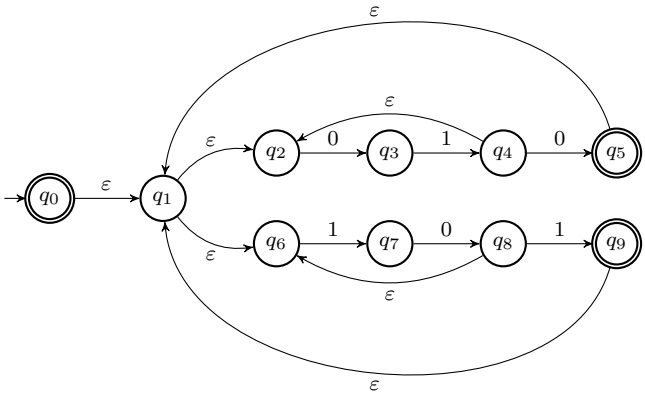
Esercizio 2 [6] Determinare un DFA che riconosce il linguaggio associato alla espressione regolare $((01)^+0 \cup (10)^+1)^*$.

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 $(01)^+0 \cup (10)^+1$:

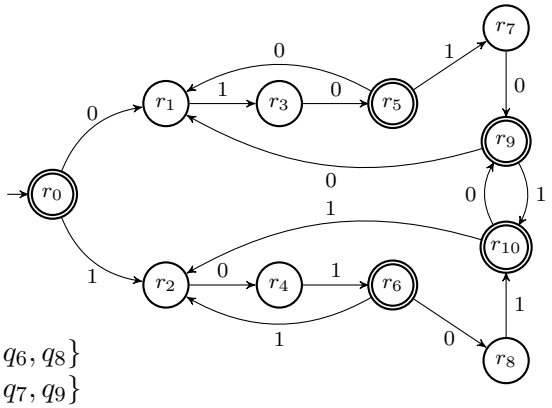


Aggiungiamo infine la trasformazione per l'operatore star ed otteniamo un NFA associato alla REX nel testo:



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 $S \longrightarrow AB \mid BA$, $A \longrightarrow \mathtt{a}C \mid C \mid \varepsilon$, $B \longrightarrow \mathtt{b}C \mid C$, $C \longrightarrow \mathtt{c}A \mid \mathtt{c}B$. 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 DK_1. Tale stato è di accettazione, per la presenza della regola completata '$A \to .$'; 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 DK_1-test fallisce, e la grammatica non è LR(1).

$S \to .AB$	`abc`
$S \to .BA$	`abc`
$A \to .\mathtt{a}C$	`bc`
$A \to .C$	`bc`
$A \to .$	`bc`
$B \to .\mathtt{b}C$	`abc`
$B \to .C$	`abc`
$C \to .\mathtt{c}A$	`abc`
$C \to .\mathtt{c}B$	`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 $S \to .AB$; 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 $\mathtt{b}C$, e dunque `b` è simbolo di lookhead; oppure può generare C, il quale a sua volta genera comunque `c` come primo simbolo terminale. Al contrario, poichè 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 $S \longrightarrow ACB$, $A \longrightarrow 01A \mid \varepsilon$, $B \longrightarrow B10 \mid \varepsilon$, $C \longrightarrow 00C \mid 11C \mid \varepsilon$. 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: "$A \to 01A$" e "$A \to \varepsilon$". 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 $L(C) = \{w_1\, w_2\, \ldots w_n \mid n \geq 0, w_i = 00 \text{ oppure } 11, 1 \leq i \leq n\}$, 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.

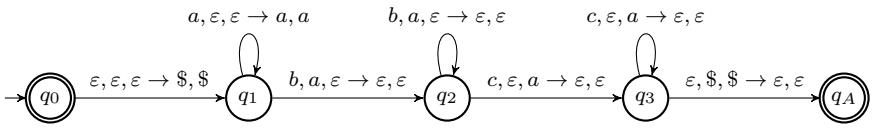
Soluzione: Ovviamente il modello di calcolo PDA rappresentato dagli automi a pila con un solo stack non può essere più potente di quello con automi a pila con due stack PDA_2, poiché qualsiasi automa con un solo stack può essere simulato senza difficoltà da un automa con due stack. Perciò il testo dell'esercizio 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 PDA_2, ove l'etichetta $v, w, x \to y, z$ 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 PDA_2 riconosce l'intera classe dei linguaggi ricorsivamente enumerabili.

Esercizio 6 [9] Una istanza del problema Balanced Partition è costituita da $2\,m$ numeri interi non negativi (non necessariamente distinti tra loro) la cui somma è pari ($2\,s$). 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 $2\,s$ 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 restrigendo 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 $\lambda = 2\,t - \mu$, e da $n + 1$ valori interi nulli (ossia n zeri $z_0 = \cdots = z_n = 0$). 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 $W = T \cup \{z_0, \ldots z_{n-q}\}$ (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 $\{z_{n-q+1}, \ldots z_n\}$ (ovviamente $|Z| = (n - q) + 1 + (n - n + q - 1 + 1) = n + 1$). Si ha:

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

e

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

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 $T = W \setminus \{z_0, \ldots, z_n\}$, 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.