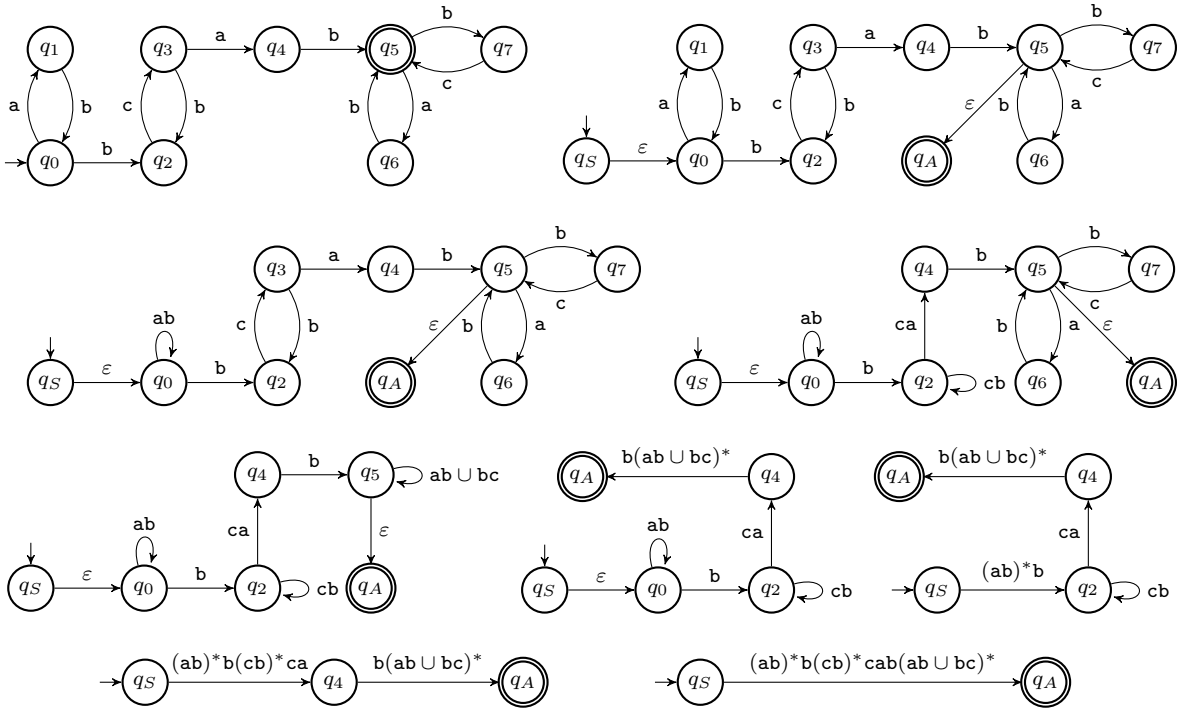
Automi e Linguaggi (M. Cesati)

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 $\{\mathtt{ab}, \mathtt{bc}\}^*$ tali che almeno una '$\mathtt{c}$' precede (anche non immediatamente) una '$\mathtt{a}$'. Ad esempio, $\varepsilon \notin A$, $\mathtt{cba} \notin A$, $\mathtt{bcab} \in A$, $\mathtt{bcba} \notin A$, $\mathtt{abbc} \notin A$. Determinare una espressione regolare per il linguaggio A, ovvero dimostrare che una tale espressione regolare non esiste.

Soluzione: Poiché $A \subseteq \{\mathtt{ab}, \mathtt{bc}\}^*$, A contiene le stringhe costituite dalla concatenazione di un numero finito di sottostringhe "$\mathtt{ab}$" e "$\mathtt{bc}$"; inoltre, in ogni elemento di A deve esistere una sottostringa "$\mathtt{ab}$" che appare dopo una sottostringa "$\mathtt{bc}$". 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:

$$(\mathtt{ab})^*\mathtt{b}(\mathtt{cb})^*\mathtt{cab}(\mathtt{bc} \cup \mathtt{ab})^* \qquad \text{ossia} \qquad (\mathtt{ab})^*(\mathtt{bc})^+\mathtt{ab}(\mathtt{bc} \cup \mathtt{ab})^*.$$

Esercizio 2 [6] Sia $B = \{\mathtt{a}^n\mathtt{b}^m \mid n \neq m\}$; 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 $C = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\} = \{\mathtt{a}^n\mathtt{b}^m \mid n = m\}$, 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 $C \subseteq \{\mathtt{a}, \mathtt{b}\}^*$: esso contiene sia tutte le stringhe con i simboli '$\mathtt{a}$' e '$\mathtt{b}$' fuori ordine, sia le stringhe con '$\mathtt{a}$' e '$\mathtt{b}$' in ordine ma diseguali in numero. In altri termini:

$$\overline{C} = \overline{D} \cup B \qquad \text{ossia} \qquad C = D \cap \overline{B}$$

ove D è l'insieme di stringhe generate dall'espressione regolare $\mathtt{a}^*\mathtt{b}^*$. 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 $\mathtt{a}^p\mathtt{b}^{p+1}$, $s \in B$ 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 $xy^iz = \mathtt{a}^{p+k(i-1)}\mathtt{b}^{p+1}$. 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 '$\mathtt{a}$' e '$\mathtt{b}$' 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 $s' = \mathtt{a}^p\mathtt{b}^{p+p!}$, 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: $j = 1 + \frac{p\cdot(p-1)\cdots 2\cdot 1}{k}$, 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 $A\,B$ dei linguaggi regolari A e B è un linguaggio regolare, ma questo non è sufficiente a stabilire la regolarità di $A \diamond B \subseteq A\,B$: 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

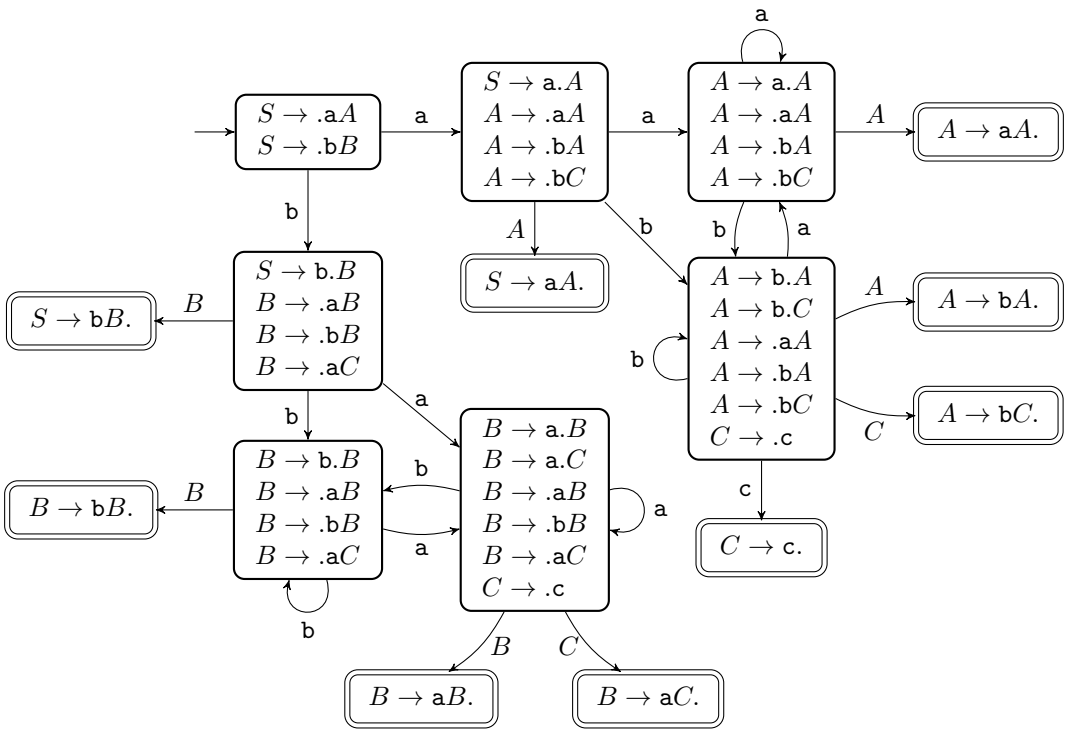
il passo 6 ha portato ad una suddivisione della stringa $w = xy$ in modo tale che la porzione y, analizzata da D_B, è un elemento del linguaggio B. In effetti però il salto nondeterministico del passo 6 può avvenire solo a partire da uno stato di accettazione di D_A; quindi la prima parte della stringa, 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

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

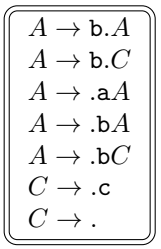
(a) Determinare se G è deterministica (DCFG). (b) Si consideri la grammatica G' ottenuta da G aggiungendo la regola $C \to \varepsilon$. La nuova grammatica G' è deterministica?

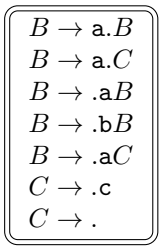
Soluzione: Per determinare se la grammatica G è deterministica costruiamo il corrispondente automa DK:



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 $C \to \varepsilon$, allora i due stati contenenti le espansioni iniziali di C diventano:





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 $L = \{\langle M, w\rangle \mid M$ è una DTM con un nastro semi-infinito che su input w 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. Innanzi tutto 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 $\mathcal{A}_{\text{TM}}$ 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 $\mathcal{A}_{\text{TM}}$ 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 $\mathcal{A}_{\text{TM}}$ 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 $L \in$ coNP, per definizione $L^c \in$ P. L'asserto da dimostrare è dunque che se $K = L^c \in$ NP allora $K^* \in$ 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 $M(w_1)$, $M(w_2)$, ..., $M(w_k)$, 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.