***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 $A = \{xx^{\mathcal{R}} \mid x \in \{0,1\}^*\}$; $A$ 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 $s = \mathtt{0}^p\mathtt{1}\mathtt{1}\mathtt{0}^p$, certamente appartenente ad $A$ ($(\mathtt{1}\mathtt{0}^p)^{\mathcal{R}} = \mathtt{0}^p\mathtt{1}$) e di lunghezza $> p$. Per essa deve valere il pumping lemma, quindi deve esistere una suddivisione $s = xyz$ tale che (1) $xy^iz \in A$ per ogni $i \geq 0$, (2) $|y| > 0$ e (3) $|xy| \leq p$. Dalle ultime due disuguaglianze possiamo dedurre che $y \in \mathtt{0}^+$, ossia $y$ è costituito da uno o più "0". Pertanto per ogni $i \neq 1$ la stringa pompata $xy^iz$ avrebbe la forma $\mathtt{0}^q\mathtt{1}\mathtt{1}\mathtt{0}^p$ 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.

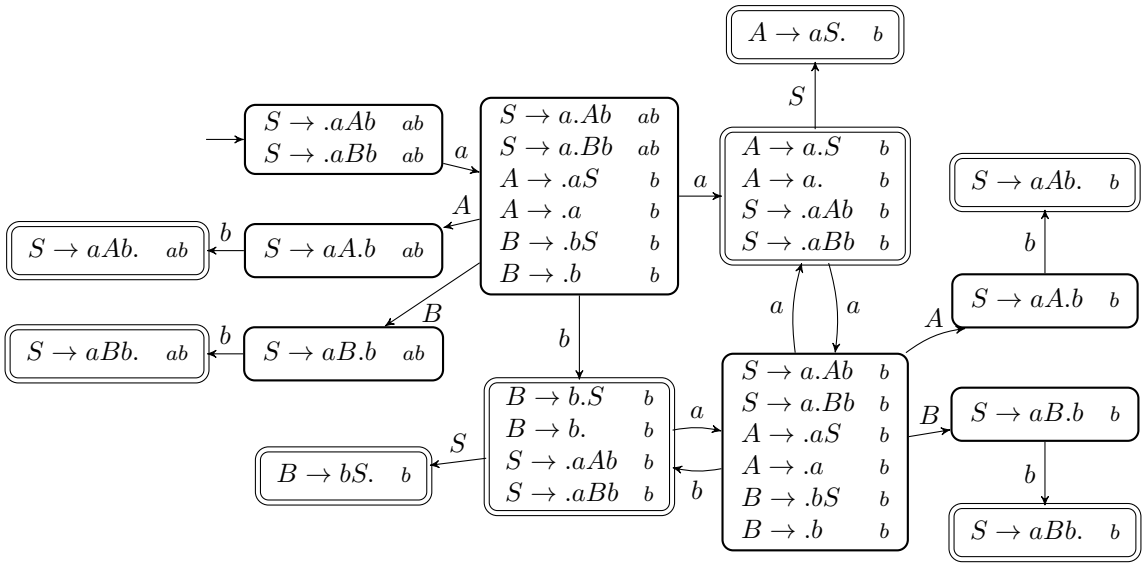
**Esercizio 2** [5] Sia $B = \{xyx^{\mathcal{R}} \mid x, y \in \{\mathtt{0},\mathtt{1}\}^*\}$; $B$ è un linguaggio regolare? Giustificare la risposta con una dimostrazione.

**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 $x^{\mathcal{R}}$ della testa; in effetti però $x$ e $y$ possono essere qualunque elemento di $\{\mathtt{0},\mathtt{1}\}^*$. Possiamo quindi considerare $x = \varepsilon \in \{\mathtt{0},\mathtt{1}\}^*$: poiché $\varepsilon^{\mathcal{R}} = \varepsilon$ (il rovescio della stringa vuota è ancora una stringa vuota), $B$ contiene come sottoinsieme $\{\varepsilon y\varepsilon \mid y \in \{\mathtt{0},\mathtt{1}\}^*\}$, ossia contiene $\{\mathtt{0},\mathtt{1}\}^*$. Quindi $\{\mathtt{0},\mathtt{1}\}^* \subseteq B \subseteq \{\mathtt{0},\mathtt{1}\}^*$, ossia $B = \{\mathtt{0},\mathtt{1}\}^*$. Naturalmente $B$ è regolare, ad esempio perché generato dalla espressione regolare $(\mathtt{0} \cup \mathtt{1})^*$.

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

$$S \to aAb \,|\, aBb \quad A \to aS \,|\, a \quad B \to bS \,|\, b$$

**Soluzione:** La grammatica è LR(1) ma non LR(0) (ossia non è DCFG). Per dimostrare ciò costruiamo l'automa per il $DK_1$-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

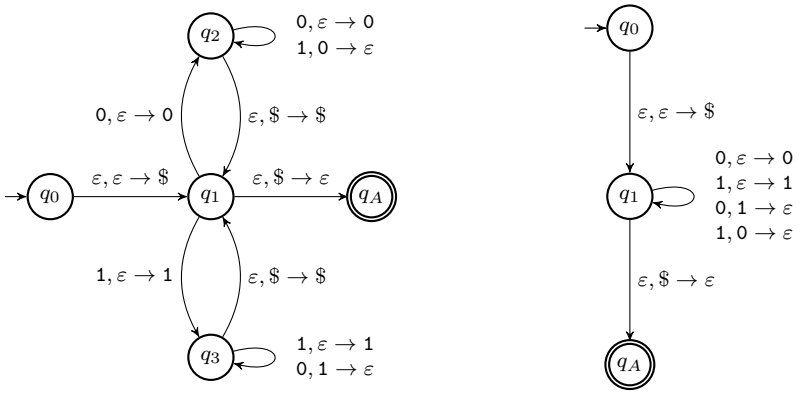
$$C = \{x \in \{\mathtt{0}, \mathtt{1}\}^* \,|\, x \text{ ha lo stesso numero di } \mathtt{0} \text{ e } \mathtt{1}\},$$

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

**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

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

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.