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

Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 23 giugno 2020

(La prova è stata svolta "a libri aperti", perciò una piccola parte degli esercizi proposti presenta una difficoltà superiore a quella consueta. I testi di alcuni esercizi sono stati modificati rispetto ai compiti assegnati in sede d'esame; la tipologia del'esercizio ed il relativo svolgimento rimangono comunque immutati.)

Esercizio 1. Sia $A \subseteq \{0,1\}^*$, e sia

$$A_1 = \{x \in \{0,1\}^* \mid \exists y \in A : |x| = |y| \text{ e } x \text{ e } y \text{ differiscono per al più un singolo bit}\}.$$

Ad esempio, se $A = \{0, 11, 010\}$ allora $A_1 = A \cup \{1, 01, 10, 110, 000, 011\}$. Dimostrare che se A è un linguaggio regolare allora anche A_1 è regolare.

Soluzione: Per la dimostrazione è sufficiente esibire un DFA oppure un NFA che riconosca il linguaggio A_1, partendo dal presupposto dell'esistenza di un DFA M che riconosce il linguaggio A. In sostanza, A_1 contiene le stringhe di A ed inoltre le stringhe ottenibili invertendo un singolo bit delle stringhe di A. Costruiamo l'automa non deterministico N in modo che simuli il comportamento di M; prima di leggere ogni simbolo dell'input, N può non deterministicamente decidere che quel simbolo è differente da quanto si aspetta M, e quindi entrare in un insieme di stati alternativo che simulano ancora il comportamento di M ma senza più mosse nondeterministiche.

Se dunque $M = (Q, \{0,1\}, \delta, q_0, F)$, sia $N = (Q_1, \{0,1\}, \delta_1, q_0, F_1)$, ove:

- l'insieme degli stati è $Q_1 = Q \cup Q'$, con $Q' = \{q' \mid q \in Q\}$;
- l'insieme degli stati di accettazione è $F_1 = F \cup F'$, ove $F' = \{q' \mid q \in F\}$;
- per ogni transizione $\delta(p, b) = q$ in M, in N esistono tre transizioni: $\delta_1(p, b) = q$ (emula il comportamento di N, ancora nessun bit invertito), $\delta_1(p, \overline{b}) = q'$ (qui si verifica una inversione di bit), e $\delta_1(p', b) = q'$ (emula il comportamento di N, inversione già avvenuta).

Supponiamo che $x \in A_1$. Se $x \in A$, $N(x)$ accetta perché esiste un percorso non deterministico che utilizza solo stati originariamente in Q ed arriva ad uno stato in F. Altrimenti se $x \notin A$, allora deve esistere una stringa $x' \in A$ che differisce da x per un singolo bit, supponiamo quello

in posizione k. Si consideri allora il percorso non deterministico entro $N(x)$ che segue gli stati originariamente in Q per i primi $k-1$ simboli dell'input, poi segue la transizione verso gli stati Q' in corrispondenza del k-esimo simbolo (arrivando allo stato che M avrebbe raggiunto leggendo il complemento del bit presente in input), ed infine segue gli stati Q' emulando il comportamento di M per i restanti simboli di input. Poiché $M(x')$ accetta, $N(x)$ arriva necessariamente in uno stato in F', quindi in uno stato di accettazione.

Supponiamo ora che esista un percorso non deterministico accettante per $N(x)$. Se lo stato di accettazione finale fa parte di F, allora lo stesso percorso esiste nella computazione deterministica $M(x)$, quindi $x \in A$, e pertanto $x \in A_1$. Se invece lo stato di accettazione finale fa parte di F', allora nel percorso non deterministico è stata applicata una transizione dagli stati Q agli stati Q'; supponiamo che sia avvenuta in corrispondenza della lettura del k-esimo simbolo. Allora la stringa x' ottenuta da x invertendo esattamente il k-esimo bit è accettata da $M(x')$, in quanto N replica nelle transizioni tra gli stati Q' il comportamento dell'automa M. Poiché dunque $x' \in A$, per definizione $x \in A_1$.

Esercizio 2. Sia $A \subseteq \Sigma^*$, e sia

$$A_{1/2} = \{x \in \Sigma^* \mid \exists y : |x| = |y| \text{ e } xy \in A\}.$$

Ad esempio, se $A = \{1, 001, 0100, 0110, 1010\}$ allora $A_{1/2} = \{01, 10\}$. Dimostrare che se A è un linguaggio regolare allora $A_{1/2}$ è regolare.

Soluzione: Per la dimostrazione è sufficiente esibire un DFA od un NFA che riconosce le stringhe di $A_{1/2}$, partendo dal presupposto dell'esistenza di un DFA M che riconosce il linguaggio A. In sintesi, $A_{1/2}$ contiene le metà superiori di tutte le stringhe di A che hanno lunghezza pari. L'idea della dimostrazione è quella di costruire un automa che simula il comportamento di N e, contemporaneamente, utilizza il non-determinismo per indovinare (1) lo stato finale di M dopo aver letto la prima metà della stringa e (2) la seconda metà della stringa di A mancante nelle stringhe di $A_{1/2}$.

Se dunque $M = (Q, \{0,1\}, \delta, q_0, F)$, sia $N = (Q', \{0,1\}, \delta', q_0', F')$, ove:

- $Q' = (Q \times Q \times Q) \cup \{q_0'\}$ (ogni stato (p,q,r) di N codifica tre stati di M: p è lo stato ottenuto seguendo la stringa di ingresso x, q è uno stato indovinato non deterministicamente in cui si fermerà $M(x)$, ed r è lo stato ottenuto in M seguendo la seconda metà della stringa a partire da q);
- $F' = \{(p,p,q) \mid p \in Q, q \in F\}$ (gli stati di accettazione di N sono quelli in cui lo stato finale di $M(x)$ è esattamente quello indovinato non deterministicamente, ed inoltre la

simulazione di M sulla seconda parte della stringa indovinata non deterministicamente porta ad uno stato di accettazione di M);

- δ' include le transizioni non deterministiche

$$\delta'(q_0') = \{(q_0, p, p) \mid p \in Q\}$$

(all'inizio $N(x)$ indovina uno stato finale p per $M(x)$ e si prepara a simulare la parte di stringa mancante a partire da esso); inoltre per ogni $p, q, r \in Q$ ed ogni $b \in \{0,1\}$,

$$\delta'((p,q,r),b) = \{(\delta(p,b), q, \delta(r,c) \mid c \in \{0,1\}\}$$

(il primo elemento della terna di stati segue la computazione di $M(x)$, il secondo rimane fisso, ed il terzo simula M su di un simbolo generato non deterministicamente).

Supponiamo che $x \in A_{1/2}$. Sia $\overline{q}$ lo stato in cui termina $M(x)$, e consideriamo in $N(x)$ il percorso non deterministico che inizia con la transizione da q_0' verso $(q_0, \overline{q}, \overline{q})$. Per ogni simbolo letto di x, N emula deterministicamente la computazione di $M(x)$ nel primo elemento della terna di stati, arrivando alla fine in $\overline{q}$. Poiché $x \in A_{1/2}$, deve esistere $y \in \{0,1\}^{|x|}$ tale che $xy \in A$. Consideriamo dunque le transizioni non deterministiche che "indovinano" i bit di y, partendo dallo stato $\overline{q}$: poiché $M(xy)$ accetta, $N(x)$ arriva in uno stato $(\overline{q}, \overline{q}, q_A)$, ove $q_A \in F$ (si noti che $|x| = |y|$, quindi la prima componente "raggiunge" $\overline{q}$ quando la terza componente "raggiunge" q_A). Poiché $(\overline{q}, \overline{q}, q_A) \in F'$, $N(x)$ accetta.

Viceversa, supponiamo che $N(x)$ accetti una stringa x, ossia che esista un percorso non deterministico che porti nello stato $(\overline{q}, \overline{q}, q_A)$, con $q_A \in F$. Allora per costruzione $M(x)$ termina nello stato $\overline{q}$, ed esiste una sequenza di $|x|$ simboli y che porta l'automa M dallo stato $\overline{q}$ allo stato q_A. Ciò significa che la computazione deterministica $M(xy)$ accetta, quindi $xy \in A$. Pertanto per definizione $x \in A_{1/2}$.

Esercizio 3. Sia $A \subseteq \{0,1\}^*$ tale che $x \in A$ se e solo se la somma di tutti i bit di x è minore della metà della lunghezza della stringa x (ossia, se $x = x_1, \ldots, x_n$, allora $\sum_{i=1}^{n} x_i < \frac{n}{2}$). Dimostrare che A non è un linguaggio regolare.

Esercizio 4. Sia $A \subseteq \{0,1\}^*$ tale che $x \in A$ se e solo se il numero di bit 1 in x è minore del numero di bit 0. Dimostrare che A non è un linguaggio regolare.

Soluzione: Questi problemi sono identici, anche se hanno una diversa formulazione. Per dimostrare che A non è un linguaggio regolare utilizziamo il "pumping lemma". Supponiamo

dunque per assurdo che A sia regolare, e sia $p > 0$ la sua "pumping length". Consideriamo la stringa

$$w = \overbrace{1 \cdots 1}^{p} \overbrace{0 \cdots 0}^{p+1} \in A.$$

A causa del "pumping lemma" deve esistere una suddivisione $w = xyz$ ove $|y| > 0$, $|xy| \leq p$, e per ogni intero i vale $xy^i z \in A$. Poiché $|xy| \leq p$ allora $xy \in 1^*$; considerato che $|y| > 0$, ne consegue che nella stringa xy^2z il numero di 1 deve crescere almeno di una unità mentre il numero di 0 rimane costante. Perciò $xy^2z \notin A$. Questo contraddice il "pumping lemma", e dunque si deve concludere che il linguaggio A non è regolare.

Esercizio 5. Siano $A, B \subseteq \Sigma^*$, e sia $A \setminus B = \{x \in A \,|\, x \notin B\}$. Dimostrare che se A è un linguaggio libero dal contesto (CFL) e B è un linguaggio regolare allora $A \setminus B$ è CFL.

Soluzione: Dando per dimostrato il lemma che afferma che se A è CFL e B è regolare allora $A \cap B$ è CFL, l'asserto segue immediatamente considerando l'uguaglianza $A \setminus B = A \cap B^c$ ed osservando che il complemento di un linguaggio regolare è regolare.

D'altra parte una dimostrazione diretta per l'esercizio è essenzialmente identica alla dimostrazione che $A \cap B$ è CFL. Poiché A è CFL, esiste un PDA $P = (Q_P, \Sigma, \Gamma_P, \delta_P, q_0^P, F_P)$ che accetta tutti e solo gli elementi di A. Analogamente, poiché B è regolare, esiste un DFA $D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$ che accetta tutti e solo gli elementi di B. L'idea della dimostrazione è di esibire un altro PDA che si comporta esattamente come P ma contemporaneamente simula l'esecuzione di D, ed accetta se e solo se P accetta mentre D rifiuta. (Nel caso di $A \cap B$, si deve accettare se e solo se sia P che B accettano.)

Consideriamo dunque un altro PDA $P' = (Q, \Sigma, \Gamma, \delta, q_0, F)$ ove $Q = Q_P \times Q_D$, $\Gamma = \Gamma_P$, $q_0 = (q_0^P, q_0^D)$ e $F = F_P \times F_D^c$.

Per ogni stato $q = (q^P, q^D) \in Q_P \times Q_D$ e simboli $x \in \Sigma_\varepsilon$, $y \in \Gamma_\varepsilon$, sia

$$\delta(q, x, y) = \begin{cases} \{(q'^P, q^D), y') \,|\, (q'^P, y') \in \delta_P(q^P, \varepsilon, y)\} & \text{se } x = \varepsilon \\ \{(q'^P, q'^D), y') \,|\, (q'^P, y') \in \delta_P(q^P, x, y) \text{ e } \delta_D(q^D, x) = q'^D\} & \text{se } x \in \Sigma \end{cases}$$

Si osservi che P' può effettuare delle transizioni senza consumare simboli di input, ed in questo caso lo stato di D registrato negli stati di P' non cambia. Altrimenti può essere applicata una qualunque delle transizioni definite da P e contemporaneamente la transizione definita da D.

Supponiamo che $w \in A \setminus B$; allora esiste una sequenza di transizioni di P che porta $P(w)$ in uno stato in F_P, ed esiste una sequenza di transizioni di D che porta $D(w)$ in uno stato di non accettazione, ossia in F_D^c. Pertanto per costruzione esiste una sequenza di transizioni di

P' che porta $P'(w)$ in uno stato di $F = F_P \times F_D^c$. Viceversa, se P' accetta una stringa w, allora per costruzione di P' $P(w)$ accetta mentre $D(w)$ rifiuta, e dunque $w \in A \setminus B$.

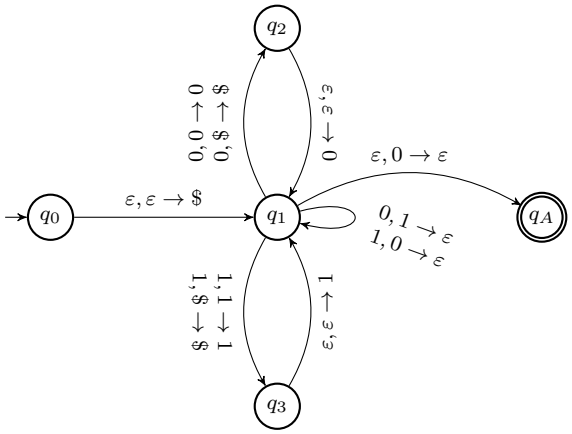
Poiché esiste un PDA che riconosce $A \cap B$, concludiamo che esso è CFL.

Esercizio 6. Sia $A \subseteq \{0,1\}^*$ tale che $x \in A$ se e solo se la somma di tutti i bit di x è minore della metà della lunghezza della stringa x (ossia, se $x = x_1, \ldots, x_n$, allora $\sum_{i=1}^{n} x_i < \frac{n}{2}$). Dimostrare che il linguaggio A è libero dal contesto (CFL).

Soluzione: Per dimostrare che A è CFL possiamo esibire una grammatica libera dal contesto che genera tutte e sole le stringhe di A, oppure un automa a pila che riconosce gli elementi di A. Poiché $x \in A$ se e solo se il numero di bit 1 in x è strettamente inferiore al numero di bit 0, scegliamo di costruire un PDA che tenga traccia dello "sbilanciamento" nelle quantità dei due tipi di simboli.

In pratica, l'automa memorizza nello stack esclusivamente i simboli in eccesso: ad esempio, ipotizzando che ad un certo punto della computazione siano stati letti 10 simboli '0' e 7 simboli '1', nello stack si troverebbero tre simboli '0' ed il simbolo '$' (che serve a marcare il fondo dello stack). Se durante la scansione della stringa il numero di '0' e di '1' finora letti fosse identico, allora lo stack conterrebbe soltanto il simbolo '$'. In nessun caso nello stack potranno essere memorizzati sia simboli '0' che '1'.

Un automa che implementa questo meccanismo è il seguente:

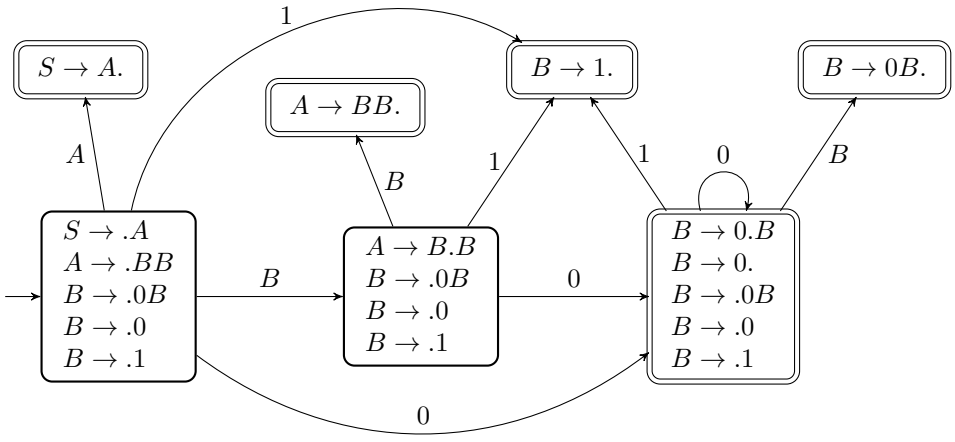


Lo stato iniziale q_0 scrive il simbolo '$' e fa transitare l'automa nello stato principale q_1. In questo stato, quando viene letto un simbolo dell'input diverso dal simbolo sulla cima dello stack, allora viene rimosso il simbolo sulla cima dello stack (o meglio, poiché l'operazione di

lettura lo ha rimosso dallo stack, il simbolo *non* viene re-inserito), e si resta nello stato q_1. Se invece il simbolo letto dall'input è uguale a quello sulla cima dello stack, oppure lo stack è vuoto (ossia '$' è sulla cima), allora viene aggiunto un simbolo in eccesso: per prima cosa si rimette sulla cima dello stack il simbolo tolto, poi se ne aggiunge un altro identico a quello letto dall'input. Per questo scopo vengono utilizzati gli stati q_2 e q_3. Infine, in qualsiasi momento l'automa può transitare nello stato di accettazione q_A, purché però sulla cima dello stack sia presente il simbolo '0', che indica che fino a quel momento si è avuto un eccesso di zero rispetto agli uno. Se il salto a q_A avviene quando l'input non è stato totalmente consumato, l'automa non potrà accettare perchè dallo stato q_A non si può più leggere alcun simbolo di input. Il percorso non deterministico che salta a q_A in corrispondenza della fine dell'input assicura l'esistenza di un percorso deterministico accettante, a condizione ovviamente che nell'input fosse presente una eccedenza di '0'.

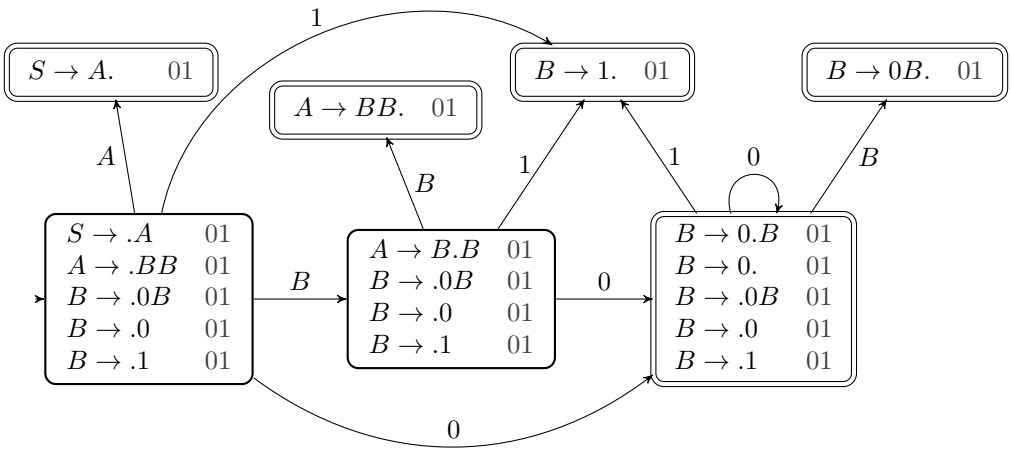
Esercizio 7. Si consideri la grammatica CFG con variabile iniziale S, simboli terminali 0 e 1, e avente le seguenti regole: $S \longrightarrow A$, $A \longrightarrow BB$, $B \longrightarrow 0B$, $B \longrightarrow 0$, $B \longrightarrow 1$. Verificare se la grammatica è LR(0) oppure LR(1).

Soluzione: Iniziamo dimostrando che la grammatica non è LR(0) (ossia DCFG) utilizzando il DK-test:



Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo '0' contiene una regola completata e diverse regole in cui il punto precede un simbolo terminale. Perciò il DK-test è fallito, e di conseguenza la grammatica non è DCFG.

Dimostriamo ora che la grammatica non è LR(1) utilizzando il DK_1-test: aggiungiamo i simboli di "lookahead" al grafo precedente, ottenendo così il seguente grafo:



Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo '0' contiene diverse regole consistenti. Infatti la regola completata ha come simboli di lookahead sia '0' che '1', e nello stesso stato compaiono regole non completate in cui il simbolo '0' od il simbolo '1' seguono il punto. Pertanto il DK_1-test è fallito e la grammatica non è LR(1).

Esercizio 8. Dimostrare che il problema di stabilire se due automi a stati finiti deterministici (DFA) riconoscono lo stesso linguaggio è decidibile esibendo un algoritmo che confronta le decisioni dei due automi su tutte le stringhe fino ad una determinata lunghezza X. Quale è il valore di X superato il quale si può concludere che gli automi riconoscono lo stesso linguaggio?

Soluzione: Consideriamo i DFA $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che operano sullo stesso alfabeto Σ (nel caso in cui gli automi operassero su alfabeti differenti, consideriamo un alfabeto comune costituito dalla loro unione, ed aggiungiamo transizioni a ciascuno dei due automi che portano l'automa in uno stato di rifiuto definitivo ogni volta che legge un simbolo non incluso nel suo alfabeto originale).

Supponiamo di essere in grado di calcolare un valore limite X come richiesto dal testo dell'esercizio. La seguente TM rifiuta se esiste una stringa $w \in \Sigma^*$ con $|w| \leq X$ tale che $D_1(w)$ e $D_2(w)$ forniscono risposte discordanti; se tale stringa w non esiste, la TM accetta.

M= " On input $\langle D_1, D_2 \rangle$, where D_1 and D_2 are DFA's:

1. Compute the value X (see below)
2. For each string w with $|w| \leq X$:
    3. Emulate the DFA D_1 on input w

4. Emulate the DFA D_2 on input w
5. If $D_1(w)$ accepts while $D_2(w)$ rejects, then reject
6. If $D_1(w)$ rejects while $D_2(w)$ accepts, then reject

7. Accept (because D_1 and D_2 agree on all strings)"

Poiché M genera un numero finito di stringhe (limitato superiormente da $|\Sigma|^X$) e per ciascuna di esse simula l'esecuzione di due DFA che concludono ciascuna in al massimo X passi, M termina su ogni input ed è quindi un decisore. Rimane però da stabilire il valore di X che rende M un decisore per il linguaggio $\mathcal{EQ}_{\text{DFA}}$.

Lemma: se esiste una stringa $v \in \Sigma^*$ per la quale D_1 e D_2 non concordano (ossia, $v \in L(D_1) \triangle L(D_2) = (L(D_1) \cap L(D_2)^c) \cup (L(D_2) \cap L(D_1)^c)$), allora esiste una stringa $w \in L(D_1) \triangle L(D_2)$ di lunghezza non superiore a $|Q_1| \cdot |Q_2|$.

Infatti, supponiamo che $|v| > |Q_1| \cdot |Q_2|$, e consideriamo la sequenza di stati $q_1, q_2, \ldots, q_{|v|}$ attraversati dall'automa D_1 leggendo la stringa v, e la analoga sequenza di stati $q'_1, q'_2, \ldots, q'_{|v|}$ attraversati da D_2 leggendo v. Poiché il numero delle coppie di stati $(q, q') \in Q_1 \times Q_2$ è uguale a $|Q_1| \cdot |Q_2|$, devono esistere due simboli v_i e v_j di v (con $i < j$) aventi la stessa coppia di stati (q, q'). La stringa v' ottenuta da v rimuovendo tutti i simboli che seguono v_i fino al simbolo v_j incluso è ancora contenuta in $L(D1) \triangle L(D_2)$ ed ha lunghezza strettamente inferiore a v. Ripetendo il procedimento si giunge ad identificare una stringa w di lunghezza non superiore a $|Q_1| \cdot |Q_2|$ ed inclusa in $L(D1) \triangle L(D_2)$.

Pertanto, impostando $X = |Q_1| \cdot |Q_2|$ si ottiene per M un decisore del linguaggio $\mathcal{EQ}_{\text{DFA}}$.

Esercizio 9. Siano $A, B \subseteq \Sigma^*$ linguaggi ricorsivamente enumerabili. Dimostrare che il linguaggio $A \overline{\circ} B = \{x \in \Sigma^* \mid \exists\, y \in B : xy \in A\}$ è ricorsivamente enumerabile.

Soluzione: L'operatore "$\overline{\circ}$" introdotto nel testo dell'esercizio può essere considerato come l'inverso dell'operatore di concatenazione: esso rimuove dalle stringhe del primo linguaggio A tutte le "code" costituite dalle stringhe del secondo linguaggio B.

I linguaggi ricorsivamente enumerabili possono essere caratterizzati dall'esistenza di un enumeratore, ossia una TM che iterativamente "stampa" su di un nastro speciale tutte le stringhe che fanno parte del linguaggio. In particolare, se una stringa fa parte del linguaggio, l'enumeratore la stamperà certamente dopo un numero finito di passi. Poiché per ipotesi i linguaggi A e B sono ricorsivamente enumerabili, esistono i corrispondenti enumeratori E_A e E_B. Dimostriamo che $A \overline{\circ} B$ è ricorsivamente enumerabile mostrando come è possibile costruire un enumeratore E per tale linguaggio basato su E_A e E_B.

E= " On any input (ignored):

1. For every increasing value $n = 1, 2, \ldots, \infty$
    2. Run the emulator E_A and collect the first n strings of A
    3. For each string v printed by E_A:
        4. Run the emulator E_B and collect the first n strings of B
        5. For each string y printed by E_B:
            6. If y is the trailing substring of v ($v = xy$ for some x):
                7. Print the leading substring x"

Verifichiamo che effettivamente E è un enumeratore per il linguaggio $A \circledcirc B$. Innanzi tutto, se E stampa una stringa x, allora necessariamente deve esistere una stringa $y \in B$ ed una stringa $v \in A$ tale che $v = xy$. Perciò $x \in A \circledcirc B$.

Viceversa, supponiamo che $x \in A \circledcirc B$. Per definizione deve allora esistere una stringa $y \in B$ tale che la concatenazione xy è un elemento di A. L'enumeratore E_A stamperà la stringa xy dopo un numero finito di passi; sia xy la n_A-esima stringa stampata da E_A. Analogamente, l'enumeratore E_B stamperà la stringa y dopo un numero finito di passi; sia y la n_B-esima stringa stampata da E_B. Nell'iterazione dell'enumeratore E con $n = \max(n_A, n_B)$, xy è inclusa nelle stringhe collezionate da E_A e y è inclusa nelle stringhe collezionate da E_B; perciò E potrà verificare che y è la "coda" di xy e stamperà la stringa x.

Esercizio 10. Si consideri il linguaggio $A = \{x \in \{0,1\}^* \mid x \text{ ha un numero di bit 1 uguale al numero di bit 0}\}$. Sia $B = \{\langle M \rangle \mid M \text{ è una TM e } L(M) \subseteq A\}$. Il linguaggio B è decidibile oppure no? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio B non è decidibile, e per dimostrarlo è sufficiente verificare che le ipotesi del Teorema di Rice sono verificate. Si consideri come proprietà P del linguaggio della TM l'essere un sottinsieme del linguaggio A. Tale proprietà è non banale: infatti la TM che accetta tutte le stringhe non soddisfa la proprietà ($\Sigma^* \not\subseteq A$), mentre la TM che rifiuta tutte le stringhe soddisfa la proprietà ($\emptyset \subseteq A$). Inoltre, P è una proprietà del linguaggio riconosciuto dalla TM: infatti se due diverse TM riconoscono lo stesso linguaggio, per entrambe vale che il linguaggio è un sottinsieme di A, e dunque entrambe le TM soddisfano la priorità P. Poiché tutte le ipotesi del Teorema di Rice sono soddisfatte possiamo concludere immediatamente che il linguaggio B contenente codifiche di TM che soddisfano la proprietà P è indecidibile.

Esercizio 11. 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 (suggerimento: considerare il problema NP-completo Vertex Cover).

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.

Esercizio 12. Il problema Graph 3-Coloring è costituito da un grafo non diretto G tale che ogni nodo del grafo può essere "colorato" con uno di tre possibili colori in modo che nodi adiacenti abbiano colori differenti. Dimostrare che il problema è NP-completo (suggerimento: considerare il problema NP-completo `3SAT`).

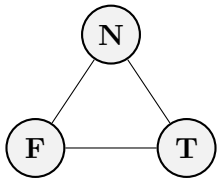
Soluzione: Per dimostrare che il problema GRAPH 3-COLORING (3COL) è verificabile in tempo polinomiale consideriamo la seguente TM:

M= " On input $\langle G, k, C \rangle$, where G is a graph and $k \in \mathbb{N}$:
1. Assign an ordering $v_1, v_2, \ldots, v_n$ to the n nodes of G.
2. Verify that C is a list of elements in $\{c_1, c_2, c_3\}$; if not, reject.
3. Verify that C includes exactly $n = |V(G)|$ elements; if not, reject.
4. For each edge $(v_i, v_j) \in E(G)$:
    6. If the i-th element and the j-th element of C are the same, reject.
7. Accept (all adjacent nodes have different colors)"

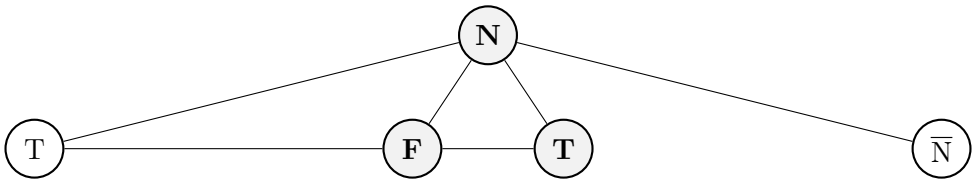
I passi 1, 2 e 3 possono essere completati in tempo lineare nel numero di nodi di G. Il ciclo nel passo 4 esegue $O(|E(G)|)$ iterazioni, ed in ciascuna di esse si controllano due elementi in una lista di n elementi. Pertanto M termina in un numero di passi polinomiale nella dimensione dell'input, e quindi 3COL $\in$ NP.

Per dimostrare che 3COL è NP-hard, consideriamo una riduzione dal problema 3SAT. Sia dunque Φ una formula CNF costituita da m clausole $C_1, \ldots, C_m$ e k variabili $x_1, \ldots, x_k$. Ciascuna clausola è la disgiunzione di tre letterali. Vediamo come costruire da Φ un grafo G che è colorabile con 3 colori se e solo se Φ è soddisfacibile.

La struttura di base della riduzione sono 3 nodi mutuamente connessi, chiamiamo questo tipo di sottografo K_3. È evidente che per colorare una struttura di tipo K_3 è necessario usare tutti e tre i colori a disposizione. Per comodità assegniamo ai 3 colori le etichette T (per 'True'), F (per 'False') e N (per 'Neutral'). Il grafo G include una struttura P di tipo K_3 che svolge il ruolo di tabella dei colori di riferimento:

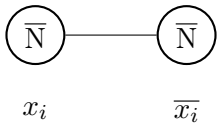


Per forzare un nodo del grafo ad assumere un determinato colore sarà sufficiente collegare il nodo ai due colori complementari nella tabella di riferimento. Un solo arco verso la struttura P impedirà al nodo di assumere il colore corrispondente:

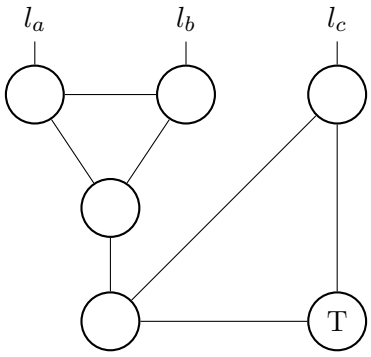


Per semplificare il grafico, una etichetta con un colore entro un nodo implica l'esistenza degli archi verso i nodi corrispondenti ai colori complementari nella struttura P. Un nodo con una etichetta con un colore negato implica l'esistenza di un arco verso il nodo della struttura P con il colore corrispondente.

Per ciascuna variabile x_i della formula Φ, il grafo G conterrà un gadget costituito da due nodi connessi tra loro ed il cui colore è forzato ad essere T oppure F. Uno dei due nodi sarà associato al letterale x_i, l'altro al letterale $\overline{x_i}$.



Per ciascuna clausola $C_i = l_a \vee l_b \vee l_c$ il grafo G include il seguente gadget:



I tre nodi superiori sono collegati da archi ai nodi dei gadget delle variabili corrispondenti ai letterali nella clausola (l'ordine è indifferente). È evidente che se i tre nodi superiori del gadget sono collegati a nodi tutti colorati con F, allora non è possibile colorare il gadget. Infatti in questo caso il nodo collegato al nodo letterale l_c dovrebbe avere colore N (perchè adiacente a due nodi colorati con T e F). Di conseguenza, il nodo in basso a sinistra nel gadget deve avere colore F. Ora ciascun nodo della struttura K_3 sopra quest'ultimo nodo è adiacente ad un nodo con colore F, quindi non è possibile colorarla. D'altra parte non è difficile convincersi che ogni altra configurazione permette di colorare con F almeno uno dei tre nodi superiori del gadget e di conseguenza tutto il gadget risulta colorabile.

Supponiamo che Φ sia una formula soddisfacibile, e consideriamo una assegnazione di verità alle variabili che rende la formula vera. Nel grafo corrispondente coloriamo con T i nodi corrispondenti alle variabili con valore vero, e coloriamo con F le variabili assegnate a falso. Di conseguenza, i colori dei nodi delle variabili negate è assegnato a T o F (poiché per costruzione queste variabili non possono avere il colore N). Si noti che per "colorare con T" o "colorare con F" si deve intendere assegnare lo stesso colore del nodo corrispondente nella struttura P. A questo punto risultano colorati i nodi della struttura P e i nodi dei gadget delle variabili, e dobbiamo verificare se questa colorazione può essere estesa ai nodi dei gadget delle clausole. Poiché Φ è soddisfatta, in ogni clausola esiste un letterale che assume il valore vero; quindi ogni gadget delle clausole ha un nodo nella fila superiore collegato ad un nodo con colore T, e che quindi può essere colorato con F. Perciò il resto del gadget può essere colorato, come sopra evidenziato.

Supponiamo ora che Φ sia una formula non soddisfacibile. Questo significa che ogni assegnazione di valori di verità alle variabili rende falsa almeno una clausola. Consideriamo quindi una qualunque colorazione del grafo: poiché i nodi dei gadget delle variabili devono avere il valore T o F, possiamo considerare l'assegnazione di verità corrispondente. Consideriamo ora un gadget delle clausole corrispodente ad una clasuola non soddisfatta di Φ: tutti i nodi devono essere connessi a nodi con colore F, quindi come sopra evidenziato non è possibile colorare il resto del gadget. Perciò non esiste una colorazione per il grafo.

La costruzione descritta è una riduzione polinomiale dal problema 3SAT al problema 3COL, e quindi 3COL è NP-hard.

Esercizio 13. Dimostrare che il problema 2SAT contenente formule Booleane in forma normale congiuntiva con 2 letterali in ogni clausola appartiene alla classe P.

Soluzione: Benché esistano algoritmi per risolvere efficientemente (anche in tempo lineare) il problema 2SAT, questi sono generalmente sofisticati e poco intuitivi. Possiamo però convincerci che 2SAT $\in$ P trasformandolo in un problema su grafi diretti.

Una formula Φ di 2SAT è la congiunzione di clausole costituite dalla disgiunzione di esattamente due letterali: $(l_1 \vee l_2)$, ove ogni letterale è una variabile diretta o negata. L'idea è che se ad un certo punto abbiamo determinato che il letterale l_1, ad esempio, ha il valore falso, allora necessariamente il letterale l_2 deve assumere il valore vero, altrimenti la formula non potrà essere soddisfatta. Riformulando, la clausola $(l_1 \vee l_2)$ si traduce in due implicazioni: $(\neg l_1 \Rightarrow l_2)$ e $(\neg l_2 \Rightarrow l_1)$.

Data la formula Φ costruiamo il grafo diretto G avente 2 nodi per ciascuna variabile v, il primo associato al letterale v ed il secondo al letterale $\neg v$. Il grafo include inoltre due archi per ogni clausola $(l_1 \vee l_2)$: un arco dal nodo $\neg l_1$ al nodo l_2, ed un altro arco dal nodo $\neg l_2$ al nodo l_1. Gli archi rappresentano le "implicazioni": assegnare ad un letterale l un valore vero implica dover assegnare il valore vero a tutti i letterali corrispondenti ai nodi raggiungibili con un singolo arco dal nodo l.

L'idea centrale è che la soddisfacibilità di Φ è correlata con la struttura delle componenti fortemente connesse (CFC) del grafo. Una CFC è un sottoinsieme di nodi tale che per ciascuna coppia di nodi u, v nel CFC esistono i percorsi diretti da u a v e da v a u utilizzanti solo nodi del sottoinsieme. Se una assegnazione di verità alle variabili rende vero un certo letterale, allora tutti i letterali nella stessa CFC devono essere resi veri, altrimenti la formula non sarebbe soddisfatta.

Un'altra osservazione importante è che se il grafo G contiene il percorso da x a y, allora contiene anche il percorso da $\neg y$ a $\neg x$ (infatti se è stato inserito un arco $(u \to v)$ a causa della clausola $(\neg u \vee v)$ allora è stato inserito anche l'arco $(\neg v \to \neg u)$). Quindi per ogni CFC esiste una CFC "complementare" contenente tutti i nodi complementari della prima. Se CFC e CFC complementare coincidessero, allora in questa CFC troveremmo sia una variabile che la sua negazione. Assegnare un qualsiasi valore a questa variabile porterebbe ad una contraddizione.

Lemma: la formula Φ *non* è soddisfacibile se e solo se esiste una variabile x tale che il nodo x ed il nodo $\neg x$ sono nella stessa CFC.

Infatti, supponiamo che esistano nodi x e $\neg x$ nella stessa CFC. Dunque esiste nel grafo sia il percorso da x a $\neg x$ che il percorso da $\neg x$ a x. Supponiamo per assurdo che Φ sia soddisfacibile, e fissiamo pertanto una assegnazione di verità alle variabili che soddisfa la formula. Se alla variabile x viene assegnato il valore vero, consideriamo il percorso da x a $\neg x$. La catena di implicazioni porta a concludere che anche a $\neg x$ deve essere assegnato il valore vero, ma questo è impossibile perchè non si può assegnare lo stesso valore di verità ad una variabile ed alla sua negazione. Lo stesso discorso si applica al percorso da $\neg x$ a x qualora alla variabile x sia assegnato il valore falso. La contraddizione prova che Φ non può essere soddisfacibile.

Viceversa, supponiamo che non esista alcun variabile x per la quale i nodi x e $\neg x$ sono nella stessa CFC. Ciò significa che esistono almeno due CFC complementari e distinte. Anche se possono esistere archi tra CFC differenti, questi archi non costituiscono mai un ciclo, altrimenti tutte le CFC toccate dal ciclo sarebbero un unica CFC. Di conseguenza esiste una CFC che non ha archi uscenti: impostare al valore vero tutti i letterali in questa CFC non comporta conseguenze verso altre variabili non incluse nella CFC, proprio perché non esistono archi uscenti. Rimuoviamo ora dal grafo questa CFC e la sua CFC complementare (ormai tutti i

letterali in questa CFC complementare avranno assegnato il valore falso). Di nuovo, esisterà una CFC del grafo rimanente senza archi uscenti, ed assegniamo il valore vero a tutti i suoi letterali. Ripetiamo il procedimento fino ad esaurire tutte le CFC. Si consideri ora una generica clasuola $(l_1 \vee l_2)$ di Φ. Questa ha dato luogo a due distinti archi nel grafo: $(\neg l_1 \rightarrow l_2)$ e $(\neg l_2 \rightarrow l_1)$. Se uno dei due archi è contenuto in una CFC selezionata per assegnare il valore vero, l'altro arco è nella CFC complementare che è stata rimossa di conseguenza. La clasuola è soddisfatta dal letterale l_1 o l_2 contenuto nella CFC selezionata per il valore vero. L'altra possibilità è che i due archi colleghino due CFC differenti, ma in questo caso la direzione è unica, altrimenti si avrebbe un ciclo tra CFC. Ora, l'assegnazione di verità deve avere dato il valore vero ai letterali dei nodi l_1 e l_2, in quanto tra le due questa è la CFC sicuramente senza archi uscenti. Perciò la clausola è soddisfatta perchè entrambi i suoi letterali sono veri. Dunque Φ è soddisfacibile.

Il lemma appena dimostrato ci consente di descrivere la seguente TM che decide il problema 2SAT:

M= " On input $\langle \Phi \rangle$, a Boolean CNF formula with clauses of size 2:
1. Build the graph G corresponding to Φ
2. For every node $x \in V(G)$:
    3. Run the TM N deciding the PATH problem on $\langle G, x, \neg x \rangle$
    4. Run the TM N deciding the PATH problem on $\langle G, \neg x, x \rangle$
        5. If both $N(\langle G, x, \neg x \rangle)$ and $N(\langle G, \neg x, x \rangle)$ accept, then reject
6. Accept (no node and its complement in the same CFC)"

Poiché sappiamo che il problema PATH (relativo all'esistenza di un percorso tra due nodi in un grafo diretto) è in P, possiamo utilizzare una TM N che decide PATH in tempo polinomiale come procedura della TM M. Ogni passo della macchina può essere completato in tempo polinomiale nella dimensione dell'input, e quindi 2SAT $\in$ P.

Esercizio 14. Il problema GRAPH 2-COLORING è costituito da un grafo non diretto G tale che ogni nodo del grafo può essere "colorato" con uno di due possibili colori in modo che nodi adiacenti abbiano colori differenti. Dimostrare che il problema è in P.

Soluzione: Mostriamo una TM che decide direttamente le istanze del problema GRAPH 2-COLORING (2COL):

M= " On input $\langle G \rangle$, where $G = (V, E)$ is a undirected graph:
1. Scan every node in V; if all of them are colored, then accept

2. Pick a uncolored node $u \in V$
3. Assign an arbitrary color to u
4. Push u on the stack
5. While the stack is not empty:
    6. Pop a node u from the stack
    7. For every edge $e \in E$:
        8. If u is not one of the ends of e, continue
        9. If the other end v of e is already colored:
            10. If u and v have the same color, then reject
        11. Else (v is not colored):
            12. Assign to v the color opposite to the color of u
            13. Push v on the stack
14. Restart from step 1"

La TM utilizza una porzione del nastro come uno stack. Ogni passo elementare dell'algoritmo richiede un tempo d'esecuzione costante oppure lineare nella dimensione del grafo. Ad ogni iterazione del ciclo più esterno viene colorato almeno un nodo, quindi il numero di iterazioni è al più $n = |V|$. Nello stack sono memorizzati, una sola volta, tutti i nodi appena colorati, quindi al massimo contiene n elementi. Possiamo dunque immediatamente concludere che M esegue in tempo polinomiale nella dimensione del grafo.

È immediato verificare che se $M(\langle G \rangle)$ accetta allora il grafo è 2-colorabile: infatti M accetta solo se tutti i nodi sono colorati, ed ogni assegnazione di un colore comporta la verifica della consistenza dei colori dei nodi adiacenti. Viceversa, supponiamo che il grafo è 2-colorabile. La procedura esegue una nuova iterazione del ciclo esterno per ogni componente connessa del grafo. All'interno di una componente connessa esistono due schemi di colorazione alternativi ed equivalentemente leciti; M sceglie uno dei due schemi assegnando un colore arbitrario al primo nodo non colorato della componente. Successivamente la scelta dei colori è forzata, in quanto ogni nodo adiacente ad uno già colorato deve necessariamente avere il colore opposto. M quindi si limita a completare la colorazione della componente connessa. Poichè in effetti il grafo è 2-colorabile, M alla fine accetterà.

Possiamo dare una dimostrazione alternativa se assumiamo di avere dimostrato che il problema 2SAT sulla soddisfacibilità di formule booleane in CNF con 2 letterali in ogni clausola è risolvibile in tempo polinomiale. In questo caso la dimostrazione dell'esercizio segue facilmente mostrando una riduzione dal problema Graph 2-Coloring (2COL) al problema 2SAT: 2COL $\leq_p$ 2SAT. Sia dunque $G = (V, E)$ un grafo non diretto. Costruiamo la formula Φ avente una variabile x_v per ogni nodo $v \in V$ e due clausole (x_u, x_v) e $(\overline{x_u}, \overline{x_v})$ per ogni arco $(u, v) \in E$. Le due clausole rappresentano le quattro implicazioni $(\overline{x_u} \Rightarrow x_v)$, $(\overline{x_v} \Rightarrow x_u)$, $(x_u \Rightarrow \overline{x_v})$ e

$(x_v \Rightarrow \overline{x_u})$ che possono essere interpretate come "x_u e x_v devono avere assegnazioni di verità differenti". Supponiamo che esista una colorazione dei nodi di G con due colori. Allora l'assegnazione di verità alle variabili di Φ che assegna ad una variabile il valore vero se il nodo ha un determinato colore ed il valore falso se ha l'altro colore soddisfa la formula. Infatti se esistesse una clausola non soddisfatta in Φ questa avrebbe due letterali falsi. Ma ciò è impossibile perché le due variabili corrispondenti avrebbero lo stesso valore, e questo significa che due nodi adiacenti del grafo avrebbero lo stesso colore. Viceversa, se esiste una assegnazione di verità che soddisfa la formula, assegniamo i colori ai nodi in modo corrispondente. Se ora esistesse un arco nel grafo che collega due nodi colorati allo stesso modo, allora una delle due corrispondenti clausole nella formula sarebbe falsa, perchè le variabili avrebbero lo stesso valore.