Automi e Linguaggi (M. Cesati)

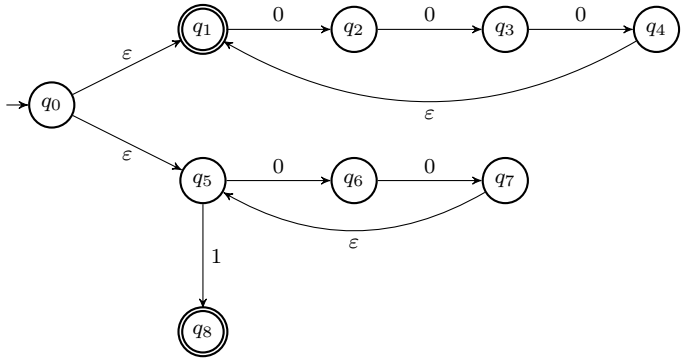
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 14 luglio 2020

(La prova è stata svolta "a libri aperti". I testi di alcuni esercizi sono stati modificati rispetto ai compiti assegnati in sede d'esame; la tipologia dell'esercizio ed il relativo svolgimento rimangono comunque immutati.)

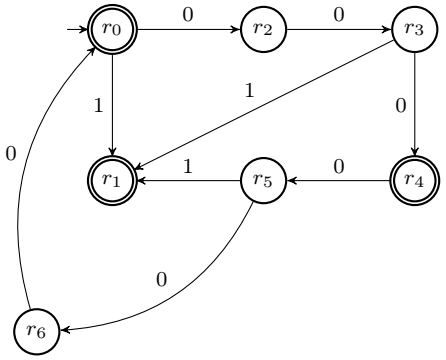
Esercizio 1. Determinare un automa a stati finiti deterministico (DFA) che riconosca il linguaggio generato dall'espressione regolare $(\mathtt{000})^* \cup (\mathtt{00})^*\mathtt{1}$.

Soluzione: L'esercizio si può risolvere in modo totalmente meccanico derivando innanzi tutto un NFA dalla espressione regolare, e successivamente trasformando lo NFA in un DFA. Applicando qualche semplificazione di poco conto allo NFA derivato dalla espressione regolare si ottiene:

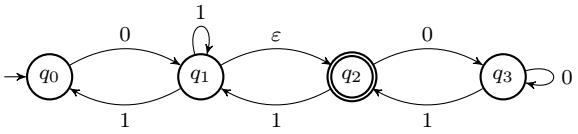


Un automa deterministico equivalente è il seguente:

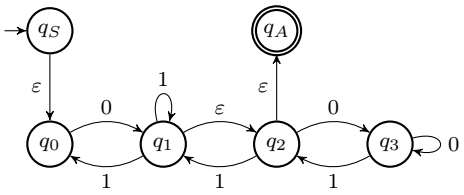
$r_0 = \{q_0, q_1, q_5\}, \{q_1, q_4, q_5, q_7\}$
$r_1 = \{q_8\}$
$r_2 = \{q_2, q_6\}$
$r_3 = \{q_3, q_5, q_7\}$
$r_4 = \{q_1, q_4, q_6\}$
$r_5 = \{q_2, q_5, q_7\}$
$r_6 = \{q_3, q_6\}$

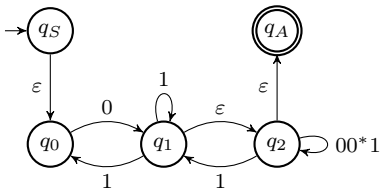


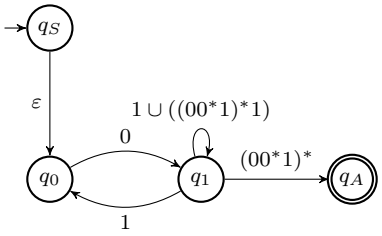
Esercizio 2. Determinare una espressione regolare equivalente al seguente automa a stati finiti non deterministico (NFA):

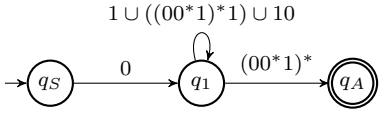


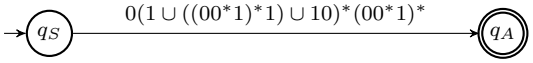
Soluzione: Per derivare l'espressione regolare in modo meccanico trasformiamo l'automa in un GNFA e rimuoviamo uno alla volta gli stati intermedi. Nei diagrammi seguenti omettiamo gli archi con etichetta $\emptyset$.











Una espressione regolare che genera il linguaggio riconosciuto dall'automa è

$$0\,(1 \cup 10 \cup (00^*1)^*1)^*\,(00^*1)^*$$

Esercizio 3. Sia $A = \{u^{\mathcal{R}}\texttt{\#}v \mid u, v \in \{\texttt{0},\texttt{1}\}^*$, u è la codifica binaria di un numero $n \geq 1$, e v è la codifica binaria del numero $n+1\}$. Le codifiche binarie non hanno zeri non significativi a sinistra. Si noti che il linguaggio codifica u invertendo l'ordine dei suoi bit. Ad esempio fanno parte di A le stringhe `01#11` e `11#100`, mentre non fanno parte di A le stringhe `10#10` (perché $\texttt{10}^{\mathcal{R}} = \texttt{01}$ ha uno zero non significativo) e `1#11`. Dimostrare che A è un linguaggio libero dal contesto (CFL).

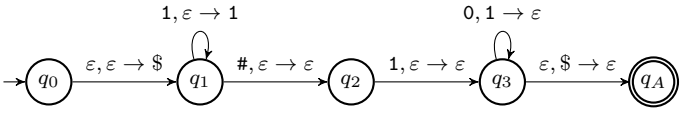
Soluzione:

Una semplice dimostrazione che A è CFL può essere ottenuta osservando che A è in effetti l'unione di due linguaggi:

$$A = B \cup C, B = \{\mathtt{1}^n\mathtt{\#10}^n \mid n \geq 1\}, C = \left\{ u^{\mathcal{R}}\mathtt{\#}v \mid u, v \in \{\mathtt{0},\mathtt{1}\}^*, |u| = |v|, (u)_2 + 1 = (v)_2 \right\},$$

ove $(x)_2$ rappresenta il numero codificato in binario dalla stringa x. In altri termini, B rappresenta le istanze in cui è presente un riporto nella addizione $+1$, mentre C rappresenta le istanze di A in cui non è presente un riporto, e dunque tali che le stringhe u e v hanno la stessa lunghezza.

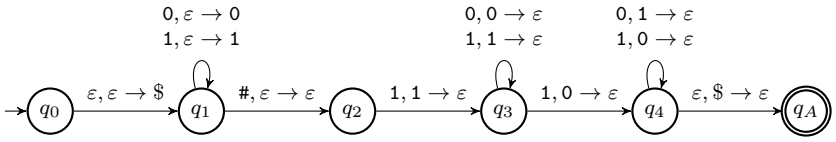
Per dimostrare che A è CFL è sufficiente dimostrare che sia B che C sono CFL (i linguaggi liberi dal contesto sono infatti chiusi rispetto all'operazione di unione). Il linguaggio B è una semplice variante del linguaggio $\mathtt{a}^n\mathtt{b}^n$, ed un PDA che lo riconosce è ad esempio:



Poiché le codifiche dei numeri nelle istanze-sì di A non devono contenenere zeri non significativi, il linguaggio C può essere descritto come:

$$C = \left\{ x\,\mathtt{0}\,w\,\mathtt{1\,\#\,1}\,y\,\mathtt{1}\,z \mid w, x, y, z \in \{\mathtt{0},\mathtt{1}\}^*, w^{\mathcal{R}} = y, x^{\mathcal{R}} = \overline{z} \right\}$$

Un esempio di PDA che riconosce C è:



(Si osservi che la definizione di A richiede che il numero n codificato da u sia maggiore di zero. Se fosse ammesso anche il caso $n = 0$ allora bisognerebbe aggiungere a C la stringa `0#1`. Poiché un linguaggio costituito da una sola stringa è regolare, C continuerebbe comunque ad essere CFL.)

Esercizio 4. Sia $A = \{u\#v \mid u, v \in \{\mathtt{0},\mathtt{1}\}^*$, u è la codifica binaria di un numero $n \geq 1$, e v è la codifica binaria del numero $n+1\}$. Le codifiche binarie non hanno zeri non significativi a sinistra. Ad esempio fanno parte di A le stringhe `10#11` e `11#100`, mentre non fanno parte di A le stringhe `01#10` e `1#11`. Dimostrare che A *non* è un linguaggio libero dal contesto (CFL).

Soluzione: Per assurdo, supponiamo che A sia CFL, e che dunque esista per esso una "pumping length" $p > 0$. Consideriamo allora la stringa $s = \mathtt{1}^p\,\mathtt{0}\,\mathtt{1}^p\,\mathtt{\#}\,\mathtt{1}^{p+1}\,\mathtt{0}^p \in A$. Poiché $|s| > p$, deve esistere una suddivisione $s = uvxyz$ tale che $|vy| > 0$, $|vxy| \leq p$, e $uv^ixy^iz \in A$ per ogni $i \geq 0$. Possono verificarsi solo i seguenti casi per la suddivisione di s:

1. La sottostringa vxy non include '`#`' ed è quindi tutta alla sinistra (o destra) di '`#`': pompando verso il basso o verso l'alto si modifica la parte a sinistra (o destra) del simbolo '`#`', quindi evidentemente la stringa risultante non può essere inclusa in A in quanto soltanto uno dei due numeri codificati nella stringa sarebbe variato rispetto a quelli codificati in s.

2. Il simbolo '`#`' è incluso in v oppure y: in questo caso pompando verso l'alto si ottiene una stringa che include due o più simboli '`#`', e che dunque non può far parte del linguaggio A.

3. Il simbolo '`#`' è incluso in x, pertanto v include simboli a sinistra di '`#`' e y include simboli a destra di '`#`'. Poiché $|vxy| \leq p$, nessun simbolo '`0`' può apparire in v od in y. Poiché inoltre $|vy| > 0$, almeno uno dei seguenti due sottocasi deve verificarsi:

   (a) $|v| > 0$: pompando verso l'alto la stringa assume la forma $\mathtt{1}^p\,\mathtt{0}\,\mathtt{1}^q\,\mathtt{\#}\,\mathtt{1}^r\,\mathtt{0}^p$ con $q > p$ e $r \geq p + 1$. Questa stringa non può far parte del linguaggio A perché sommando uno ad una sequenza di $q > p$ bit '`1`' si deve ottenere una sequenza di $q > p$ bit '`0`'.

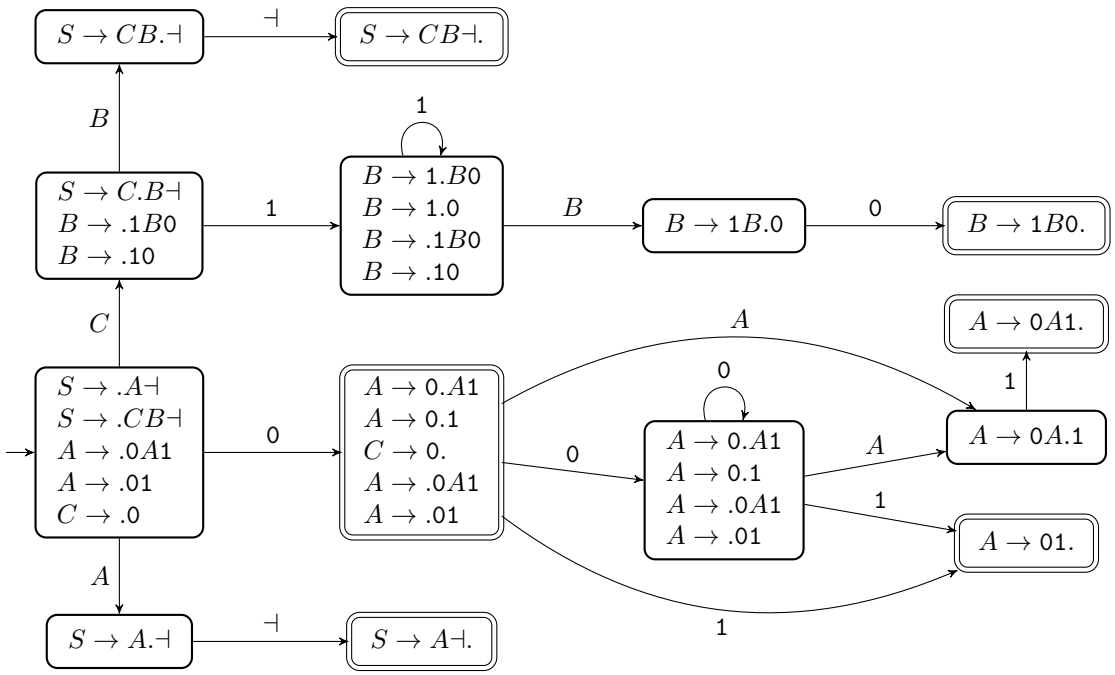
   (b) $|y| > 0$: pompando verso il basso la stringa assume la forma $\mathtt{1}^p\,\mathtt{0}\,\mathtt{1}^q\,\mathtt{\#}\,\mathtt{1}^r\,\mathtt{0}^p$ con $q \leq p$ e $r < p + 1$. Se ora fosse $p + r < p + q + 1$ allora la stringa non potrebbe far parte del linguaggio A perché il numero a sinistra sarebbe maggiore del numero a destra. Dunque deve valere $p \geq r \geq q + 1 > q$, ossia $q < p$, e quindi $|v| > 0$. Si ricade pertanto nel caso precedente.

Poiché non è possibile trovare una suddivisione per la stringa s che soddisfa il pumping lemma, A non può essere un linguaggio libero dal contesto.

Esercizio 5. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$
\begin{array}{lcl}
S \to & A\dashv & \mid\ CB\dashv \\
A \to & \mathtt{0}A\mathtt{1} & \mid\ \mathtt{01} \\
B \to & \mathtt{1}B\mathtt{0} & \mid\ \mathtt{10} \\
C \to & \mathtt{0} &
\end{array}
$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:

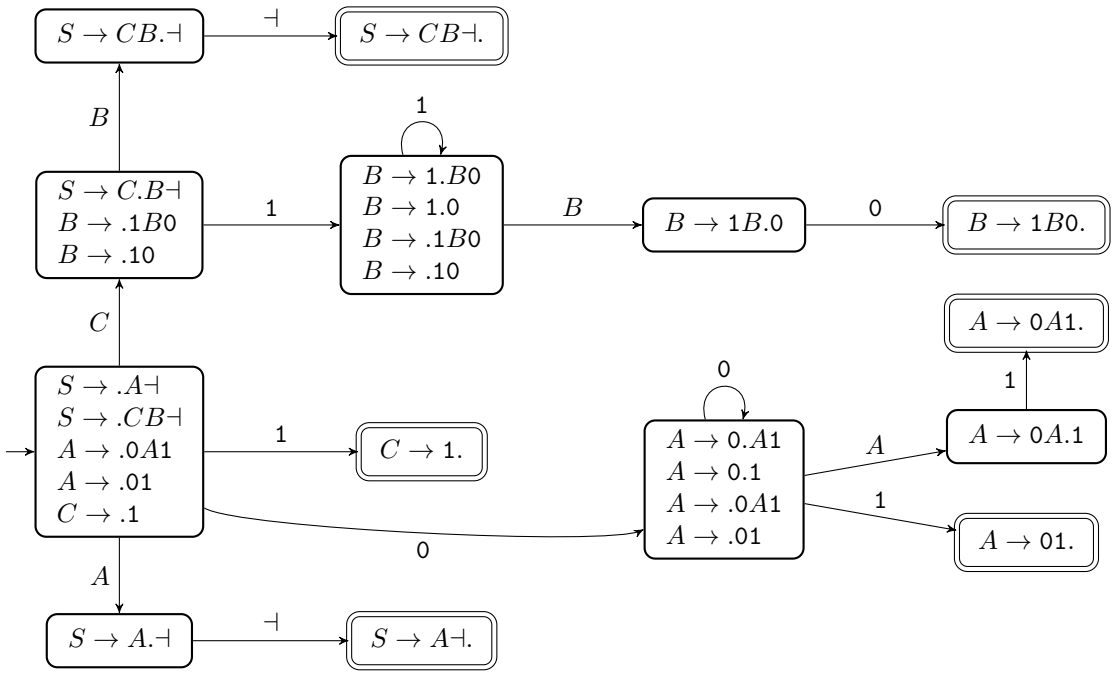


Lo stato accettante raggiungibile dallo stato iniziale seguendo il simbolo '0' contiene una regola completata e diverse regole in cui il punto precede un simbolo terminale. Perciò il DK-test è fallito, e di conseguenza la grammatica non è DCFG.

Esercizio 6. Determinare se la seguente grammatica CFG con variabile iniziale S è deterministica:

$$\begin{aligned} S &\to A\dashv \mid CB\dashv \\ A &\to 0A1 \mid 01 \\ B &\to 1B0 \mid 10 \\ C &\to 1 \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica eseguiamo il DK-test, ottenendo così il seguente diagramma:



Poiché ogni stato finale ha una sola regola, quella completata, il DK-test risulta soddisfatto, pertanto la grammatica analizzata è deterministica.

Esercizio 7. Sia $A \circledcirc B = \{x \mid \exists y \in B : xy \in A\}$, con $A, B \subseteq \Sigma^*$. Dimostrare che se A è un linguaggio libero dal contesto (CFL) e B è un linguaggio regolare allora $A \circledcirc B$ è libero dal contesto.

Soluzione: Poiché A è CFL, esiste un PDA $P = (Q_P, \Sigma, \Gamma, \delta_P, q_0^P, F_P)$ che lo riconosce. Analogamente, poiché B è un linguaggio regolare, esiste un DFA $D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$ che lo riconosce. Mostriamo come derivare da P e D un PDA P' che riconosce il linguaggio $A \circledcirc B$. L'idea è quella di simulare il comportamento di P fino alla fine della stringa (indovinata in modo non deterministico); successivamente P' simula il comportamento sia di P che di D su di una stringa y generata in modo non deterministico.

Costruiamo il PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ nel seguente modo. Gli stati interni di P sono $Q = Q_P \times (\{\hat{q}\} \cup Q_D)$, ove $\hat{q} \notin Q_D$. In sostanza, ciascuno stato di P' codifica uno stato di P (quando accoppiato con $\hat{q}$) oppure sia uno stato di P che uno stato di D. L'alfabeto di ingresso di P' coincide con il corrispondente alfabeto di P e D. L'alfabeto di stack di P' coincide con quello di P. Lo stato iniziale di P' è $q_0 = (q_0^P, \hat{q})$. Gli stati di accettazione di P' sono $F = F_P \times F_D$, ossia quelli che codificano stati di accettazione sia per P che per D.

La tabella di transizioni di P' è la seguente:

1. Per ogni stato $(q^P, \hat{q}) \in Q$ e simboli $a \in \Sigma_\varepsilon$, $b \in \Gamma_\varepsilon$, δ contiene tutte le transizioni equivalenti in P:

$$\delta((q^P, \hat{q}), a, b) = \left\{((q'^P, \hat{q}), b') \mid (q'^P, b') \in \delta_P(q^P, a, b)\right\}$$

2. Per ogni simbolo di ingresso $a \in \Sigma$, e per ogni coppia di transizioni $(q'^P, b') \in \delta_P(q_P, a, b)$ e $\delta_D(q^D, a) = q'^D$ che leggono quel simbolo a, δ contiene la transizione

$$((q'^P, q'^D), b') \in \delta((q^P, q^D), \varepsilon, b)$$

   Si osservi che la transizione in P *non* legge il simbolo di ingresso a.

3. Per ogni transizione $(q'^P, b') \in \delta_P(q^P, \varepsilon, b)$ di P che non legge simboli di ingresso e per ogni stato q^D, δ contiene la transizione

$$((q'^P, q^D), b') \in \delta((q^P, q^D), \varepsilon, b)$$

4. Per ogni stato $q^P \in Q_P$, δ contiene la transizione

$$((q^P, q_0^D), \varepsilon) \in \delta((q^P, \hat{q}), \varepsilon, \varepsilon)$$

Sia $x \in A \,\overline{\circ}\, B$, e consideriamo la computazione $P'(x)$. Poiché per definizione esiste un $y \in B$ tale che $xy \in A$, esiste una sequenza di transizioni in P' di tipo (1) che leggono l'intera stringa x e portano P' nello stato $(q_x^P, \hat{q})$ e con il contenuto dello stack S_x, ove q_x^P è esattamente lo stato raggiunto da P e S_x è il contenuto dello stack in una computazione accettante di $P(xy)$ dopo aver letto la stringa iniziale x. È possibile ora applicare una transizione di tipo (4) e transitare nello stato (q_x^P, q_0^D). Poiché $P(xy)$ accetta, esiste una sequenza di transizioni di P dallo stato q_x che leggono i simboli della stringa y e portano in uno stato di accettazione di P. Se una di queste transizioni legge un simbolo di y si può applicare una transizione di tipo (2): l'automa P' dunque modifica lo stack ed entra nello stato interno corrispondente alla transizione applicata da P; inoltre P' simula la corrispondente transizione di D applicata al simbolo di y. Se invece una di queste transizioni non legge un simbolo di y, P' può applicare una transizione di tipo (3) per modificare in modo appropriato stato interno di P e configurazione dello stack, lasciando invariato lo stato interno simulato di D. Poiché $y \in B$, quando la sequenza di transizioni corrispondente a quella di $P(xy)$ è terminata, lo stato interno di P' è uno stato di accettazione. Dunque $P'(x)$ accetta, in quanto l'intera stringa in ingresso è stata letta.

Viceversa, supponiamo che $P'(x)$ accetti una certa stringa $x \in \Sigma^*$, e consideriamo una sequenza di transizioni che porta in uno stato di accettazione. Notiamo innanzi tutto che per poter accettare la stringa, questa deve essere stata letta completamente dall'automa. Inoltre

nessuno stato di accettazione può contenere lo stato $\hat{q}$ come secondo componente della coppia. Si osservi anche che nessuna transizione in P' con lo stato $\hat{q}$ come secondo componente può leggere simboli di ingresso. Infine, non esiste alcuna transizione in P' che possa portare da uno stato interno (q^P, q^D) ad uno stato interno con $\hat{q}$ come secondo componente. Pertanto, i simboli della stringa x debbono essere letti da P' utilizzando esclusivamente transizioni di tipo (1). Poiché P' accetta, nella sequenza accettante deve esistere una transizione di tipo (4), sia questa $((q_x^P, q_0^D), \varepsilon) \in \delta((q_x^P, \hat{q}), \varepsilon, \varepsilon)$. Sia inoltre S_x la configurazione dello stack in questo punto della sequenza. Le transizioni di tipo (1) che portano da $(q_0^P, \hat{q})$ fino a $(q_x^P, \hat{q})$ corrispondono a transizioni di P; pertanto esiste una computazione di $P(x)$ che termina nello stato q_x^P con configurazione dello stack S_x. Consideriamo ora le transizioni di tipo (2) e (3) seguenti: esse portano P' in uno stato di accettazione senza leggere alcun altro simbolo di ingresso. Per ogni transizione di tipo (2) che appare nella sequenza, esiste (almeno) un simbolo $a \in \Sigma$ che ha permesso di introdurre la transizione in δ. Sia y una stringa costituita dalla sequenza di simboli 'a' definita dalla sequenza di transizioni di tipo (2). Le transizioni di tipo (2) corrispondono a transizioni in D che portano l'automa D da q_0^D ad uno stato di accettazione in F_D leggendo la stringa y in ingresso; pertanto, $y \in B$. Le transizioni di tipo (2) e (3) invece corrispondono ad una sequenza di transizioni che portano l'automa P da q_x^P e configurazione dello stack S_x ad uno stato di accettazione in F_P; pertanto $xy \in A$. Per definizione dunque $x \in A \,\overline{\circ}\, B$.

Esercizio 8. Siano A e B linguaggi Turing-riconoscibili (o in altri termini, ricorsivamente enumerabili). Dimostrare che l'intersezione $A \cap B$ è Turing-riconoscibile.

Soluzione: Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B, rispettivamente.

Consideriamo la seguente TM M:

M= " On input x:
1. Emulate the TM M_A on input x
2. Emulate the TM M_B on input x
3. If both $M_A(x)$ and $M_B(x)$ accepted, then accept
4. Otherwise, reject"

Supponiamo che $M(x)$ accetti; di conseguenza sia l'emulazione di $M_A(x)$ che quella di $M_B(x)$ devono terminare, ed inoltre entrambe le computazioni devono accettare. Perciò $x \in A$ e $x \in B$, e quindi $x \in A \cap B$. D'altra parte, se $x \notin A$, allora o $M_A(x)$ non termina, e dunque

$M(x)$ non termina, oppure $M_A(x)$ termina e rifiuta, e quindi $M(x)$ non potrà accettare. Stesso ragionamento vale se $x \notin B$. Dunque se $x \notin A \cap B$, allora $M(x)$ non accetta (in quanto non termina oppure rifiuta).

Esercizio 9. Siano A e B linguaggi Turing-riconoscibili (o in altri termini, ricorsivamente enumerabili). Dimostrare che $A\,B = \{xy \mid x \in A, y \in B\}$ è Turing-riconoscibile.

Soluzione: Poiché sia A che B sono ricorsivamente enumerabili, esistono due TM M_A e M_B che riconoscono le stringhe di A e B, rispettivamente.

Consideriamo la seguente NTM M:

M= " On input w:
1. Guess a subdivision x, y of w: $w = xy$
2. Emulate the TM M_A on input x
3. Emulate the TM M_B on input y
4. If both $M_A(x)$ and $M_B(x)$ accepted, then accept
5. Otherwise, reject"

Supponiamo che $M(w)$ accetti; di conseguenza esiste una computazione accettante che "indovina" una opportuna suddivisione $xy = w$, ed emula $M_A(x)$ e $M_B(y)$. Poiché $M(w)$ termina accettando, le due emulazioni devono terminare ed entrambe le computazioni devono accettare. Pertanto $x \in A$ e $y \in B$, e quindi $w \in A\,B$. D'altra parte, supponiamo che $w \in A\,B$; dunque esiste una suddivisione $w = xy$ tale che $x \in A$ e $y \in B$. Questa suddivisione corrisponde ad un ramo dell'albero di computazioni di $M(w)$; inoltre poiché M_A riconosce A, $M_A(x)$ termina accettando; analogamente $M_B(y)$ termina accettando. Perciò $M(w)$ accetta.

Si osservi che il non-determinismo di M non costituisce un problema, perché per ogni TM non deterministica esiste una TM deterministica equivalente. Perciò il linguaggio $A\,B$ è ricorsivamente enumerabile.

Esercizio 10. Si consideri il problema codificato dal linguaggio $\{\langle\Phi\rangle \mid \Phi$ è una formula 3-CNF per la quale esiste una assegnazione di verità che rende un letterale vero ed un letterale falso in ciascuna clausola$\}$. Dimostrare che tale linguaggio è NP-completo esibendo una riduzione da 3SAT.

Soluzione: Questo problema è noto con diversi nomi, ad esempio $\neq$-SAT oppure "Heterogeneous SAT". Essendo molto simile al problema NAESAT (Not-All-Equal SAT) introdotto in un precedente esercizio d'esame, ci riferiremo ad esso con il nome NAE3SAT.

Per dimostrare che NAE3SAT è in NP osserviamo che esso è polinomialmente verificabile: infatti ogni istanza che fa parte del linguaggio ha come certificato l'assegnazione di verità alle variabili che garantisce la presenza in ciascuna clausola di un letterale falso e di un letterale vero. Tale certificato è certamente di dimensione non superiore alla dimensione dell'istanza, e verificare che l'assegnazione soddisfa i requisiti del problema è implementabile in modo immediato, in modo del tutto analogo alla verifica usata per SAT che una assegnazione di verità renda almeno un letterale vero in ogni clausola.

La riduzione polinomiale dal problema 3SAT è simile a quella utilizzata per NAESAT: introduciamo una nuova variabile s che non appare in alcuna clausola della formula originale e la cui assegnazione di verità assicurerà l'esistenza di un letterale falso in ogni clausola. A differenza però della riduzione per NAESAT non possiamo semplicemente aggiungere s ad ogni clausola perché il numero di letterali in ciascuna clausola diventerebbe uguale a quattro, mentre in NAE3SAT deve essere esattamente tre. Possiamo però sfruttare la stessa idea descritta nella riduzione da SAT a 3SAT: spezziamo ogni clausola di quattro letterali in due clasuole di tre letterali ciascuna aggiungendo una nuova variabile per ciascuna clausola originale.

In pratica, sia Φ una formula 3-CNF costituita da m clausole: $\Phi = \wedge_{i=1}^{m} C_i$. Per ciascuna clausola $C_i = (l_a \vee l_b \vee l_c)$, definiamo due clausole: $(l_a \vee l_b \vee z_i)$ e $(l_c \vee \overline{z_i} \vee s)$. La variabile s è una nuova variabile comune a tutte le clausole, mentre z_i è una nuova variabile specifica per la clausola C_i. La formula 3-CNF Φ' costituita dalla congiunzione delle $2m$ clausole derivate da quelle di Φ è l'istanza del problema NAE3SAT. È evidente che la dimensione di Φ' è polinomialmente limitata dalla dimensione di Φ e che la sua costruzione è meccanica.

Supponiamo dunque che Φ sia soddisfacibile. Esiste perciò una assegnazione di verità alle variabili che rende vero un letterale in ciascuna clausola C_i. Consideriamo dunque un generico letterale $C_i = (l_a \vee l_b \vee l_c)$. La seguente tabella riassume le possibili assegnazioni di verità ai letterali, e le impostazioni delle variabili z_i e s che assicurano in ciascuna delle due clausoli corrispondenti di Φ' l'esistenza di un letterale vero e di un letterale falso.

l_a	l_b	l_c	z_i	s
F	F	F	escluso	
F	F	T	T	*
F	T	F	F	*
F	T	T	T	*
T	F	F	F	*
T	F	T	T	*
T	T	F	F	*
T	T	T	F	F

Si osservi che non è possibile che l_a, l_b e l_c siano tutti falsi, in quanto l'assegnazione di verità deve soddisfare la clausola originale. In tutti gli altri casi tranne l'ultimo il valore della variabile comune s non è importante, perché è sufficiente impostare opportunamente il valore di z_i per avere un letterale vero ed uno falso in ogni clausola. Solo nel caso in cui tutti e tre i letterali originali hanno il valore vero è necessario assegnare a s il valore falso.

Assegnando dunque alle variabili z_i il valore opportuno secondo la corrispondente tabella ed assegnando alla variabile s il valore falso si ottiene una estensione della assegnazione di verità per Φ' che soddisfa le condizioni del problema NAE3SAT.

Viceversa, consideriamo una formula Φ' che possiede una assegnazione di verità che assegna a ciascuna clausola un letterale vero ed uno falso. Si consideri il valore assunto dalla variabile s: se nella assegnazione s è vera, consideriamo l'assegnazione di verità complementare a quella data: essa necessariamente continuerà ad assegnare a ciascuna clausola un letterale vero ed un falso. In questa assegnazione il valore di s sarà falso. Assumiamo dunque che ad s sia assegnato il valore falso. Ciò significa che in ogni clausola con letterali $\overline{z_i}$, almeno uno tra l_c e $\overline{z_i}$ deve assumere il valore vero. Se l_c assume il valore vero allora la corrispondente clausola $(l_a \vee l_b \vee l_c)$ è soddisfatta. Altrimenti $\overline{z_i}$ deve essere vero, dunque z_i è falsa, perciò nella clausola $(l_a \vee l_b \vee z_i)$ almeno una tra l_a e l_b deve essere vera. Pertanto $(l_a \vee l_b \vee l_c)$ è ancora soddisfatta. Dunque la formula Φ è soddisfacibile.

Si può dunque concludere che 3SAT $\leq_P$ NAE3SAT, e dunque NAE3SAT è NP-completo.

Esercizio 11. Si consideri il problema Feedback Vertex Set: dato un grafo diretto $G = (V, A)$ ed un numero $k \in \mathbb{N}$, esiste un sottoinsieme $V' \subseteq V$ con $|V'| \leq k$ tale che ogni *ciclo* diretto entro G include almeno un nodo in V'? Dimostrare che il problema è NP-completo.

Soluzione: Il problema Feedback Vertex Set (o FVS) è verificabile polinomialmente: un certificato è banalmente la lista di nodi che costituisce l'insieme V'. Si consideri infatti il seguente algoritmo.

M= " On input $\langle G, k, V' \rangle$:

1. Verify that V' is a subset of k or less nodes of G
2. Build the graph $G' = G \setminus V'$

3. For every pairs of nodes s, t in G':
    4. Run PATH(G', s, t) to determine if there is a path from s to t
    5. If the path exists:
        6. Let I be the nodes of the path except s and t
        6. Build the graph $G'' = G \setminus I$
        7. Run PATH(G'', s, t)
            8. If the path exists, reject (cycle found)
9. There is no cycle in G', hence accept"

La notazione $G \setminus V$ indica il grafo ottenuto da G rimuovendo tutti i nodi dell'insieme V e tutti gli archi incidenti su questi nodi. Poiché l'algoritmo PATH è polinomiale, il verificatore esegue in tempo polinomiale nel numero di nodi del grafo G in istanza.

Per dimostrare che FVS è NP-hard possiamo considerare una semplicissima riduzione dal problema NP-completo Vertex Cover. Sia infatti (G, k) una istanza del problema VC, ove G è un grafo non diretto e $k \in \mathbb{N}$. Consideriamo il grafo diretto G' ottenuto sostituendo ad ogni arco non diretto di G una coppia di archi in direzione opposta tra gli stessi nodi in G'. Sia dunque (G', k) l'istanza ridotta di FVS, che ha ovviamente dimensione polinomiale ed è meccanicamente costruibile.

Supponiamo che (G, k) sia una istanza-sì di VC. Dunque esiste un sottoinsieme di al più k nodi che copre ogni arco di VC. Lo stesso sottoinsieme di nodi in G' copre ogni arco diretto, quindi la rimozione dei nodi di V' rende G' un grafo senza archi, e quindi senza cicli. Perciò (G', k) è una istanza-sì di FVS. Viceversa, supponiamo che (G', k) sia una istanza-sì di FVS, pertanto esiste un sottoinsieme di al più k nodi che copre ogni ciclo del grafo G'. Si consideri ora che ciascun arco del grafo G ha dato origine ad un ciclo di dimensione 2 in G', e anche tali cicli devono essere coperti da V'. Pertanto l'insieme di nodi V' copre ogni arco del grafo G, e dunque (G, k) è una istanza-sì di VC.

Concludendo, VC $\leq_p$ FVS, pertanto FVS è NP-hard, e dunque NP-completo.

Esercizio 12. Si consideri il problema Longest Path: dato un grafo non diretto $G = (V, E)$ ed un numero $k \in \mathbb{N}$, esiste un percorso semplice (senza nodi ripetuti) che utilizza almeno k archi? Dimostrare che il problema è NP-completo.

Soluzione: Il problema Longest Path (LP) è verificabile polinomialmente. Infatti un certificato consiste in una lista di nodi del grafo. Il verificatore deve controllare che la lista contenga k nodi, che non contenga nodi ripetuti, e che tra ciascun nodo ed il successivo esista

un arco nel grafo. È immediato osservare come ciascuna di queste operazioni può essere svolta in tempo polinomiale, dunque LP è in NP.

Per dimostrare che il problema è NP-hard consideriamo il problema UNDIRECTED HAMILTONIAN PATH (UHP), che già sappiamo essere NP-completo. Una istanza di UHP è costituita da un grafo non diretto G ed una coppia di nodi s e t; la questione è se esiste un percorso semplice tra s e t che attraversa tutti i nodi del grafo. Per ridurre questo problema a LP possiamo osservare che un percorso semplice che attraversa n nodi deve avere $n-1$ archi. Dunque richiedendo nell'istanza di LP che il percorso sia lungo almeno il numero di nodi del grafo meno uno restringe in pratica la ricerca ad un percorso hamiltoniano. Si ha però una complicazione: a differenza di UHP, una istanza di LP non contiene una coppia di nodi selezionati come termini del percorso.

Dato un grafo G ed una coppia di nodi s e t, consideriamo il grafo G' ottenuto da G aggiungendo due nodi s' e t' e due archi (s, s') e (t, t'). Consideriamo l'istanza di LP costituita da G' e dal numero $n+1$, ove n è il numero di nodi del grafo G. Ovviamente la riduzione è calcolabile in tempo polinomiale.

Supponiamo che esista un percorso halmitoniano in G tra s e t. Dunque nel grafo G' esiste un percorso semplice tra i nodi s e t di lunghezza $n-1$. Considerando i due archi aggiuntivi possiamo costruire un percorso semplice entro G' di lunghezza $n+1$. Dunque $(G', n+1)$ è una istanza-sì di LP. Viceversa, supponiamo che esista un percorso semplice di lunghezza $n+1$ entro G': poiché G ha esattamente $n+2$ nodi, il percorso deve attraversare tutti i nodi del grafo. Poiché inoltre i nodi s' e t' hanno un solo arco incidente, essi devono necessariamente essere i terminali del percorso entro G'. Considerando dunque i successivi nodi s e t adiacenti a s' e t', esiste in G' un percorso di lunghezza $n-1$ tra s e t. Questo percorso esiste anche in G tra i nodi s e t, e dunque (G, s, t) è una istanza-sì di UHP.

Pertanto UHP $\leq_P$ LP, e dunque LP è NP-completo.