Automi e Linguaggi (M. Cesati)

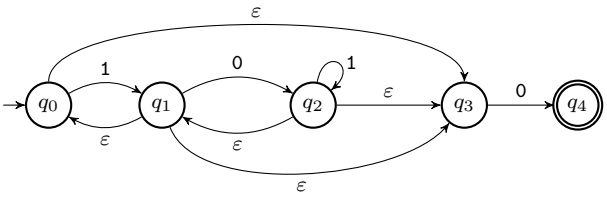
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 16 settembre 2020

(La prova è stata svolta "a libri aperti".)

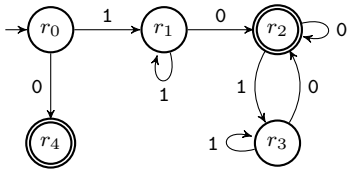
Esercizio 1. Determinare un automa deterministico che riconosca il linguaggio generato dalla espressione regolare $(\mathtt{1}(\mathtt{01}^*)^*)^*\mathtt{0}$.

Soluzione: [Corr. 26.04.2022] 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 allo NFA derivato dalla espressione regolare si ottiene:

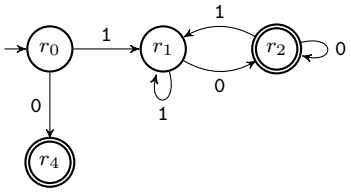


Un automa deterministico equivalente è il seguente:

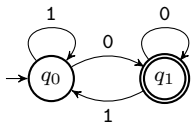
$r_0 = \{q_0, q_3\}$
$r_1 = \{q_0, q_1, q_3\}$
$r_2 = \{q_0, q_1, q_2, q_3, q_4\}$
$r_3 = \{q_0, q_1, q_2, q_3\}$
$r_4 = \{q_4\}$



In effetti gli stati r_1 e r_3 sono identici, e quindi è possibile ottenere un DFA equivalente con quattro stati:



Esercizio 2. Determinare una espressione regolare equivalente all'automa

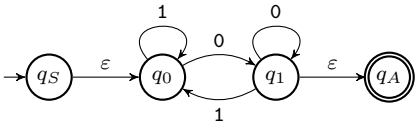


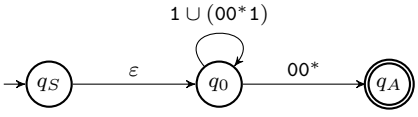
Soluzione: L'automa nel testo è molto semplice, ed è possibile determinare facilmente che il linguaggio riconosciuto dall'automa sono tutte le stringhe binarie che terminano con `0`. Dunque una espressione regolare equivalente è ad esempio $(\texttt{0} \cup \texttt{1})^*\texttt{0}$. Poiché però questa espressione non è stata derivata con un metodo formale, è necessario dimostrare la sua equivalenza con l'automa. In effetti, l'asserto segue dimostrando che:

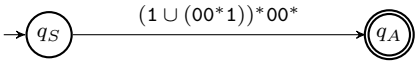
1. Ogni stringa accettata dall'automa deve terminare con `0`. Infatti, l'unico stato di accettazione è q_1, e l'unico simbolo che può portare in q_1 è `0`. Inoltre q_1 non è lo stato iniziale, dunque la stringa vuota (che non termina con `0`) non è accettata.

2. Ogni stringa che termina con `0` è accettata dall'automa. Cominciamo con l'osservare che tutti gli stati hanno transizioni uscenti sia per il simbolo `0` che per il simbolo `1`. Pertanto, tutte le stringhe con simboli in $\{\texttt{0}, \texttt{1}\}$ sono totalmente "consumate" dall'automa. Si consideri ora una stringa s che termina con `0`: lo stato seguente alla lettura dell'ultimo `0` non può essere diverso da q_1, poiché questo è l'unico simbolo a cui si può arrivare leggendo uno `0`. Poiché q_1 è uno stato di accettazione, la stringa s viene accettata.

L'esercizio può essere svolto anche in modo più meccanico utilizzando la procedura di conversione da un GNFA ad una espressione regolare. Si consideri ad esempio la seguente conversione:



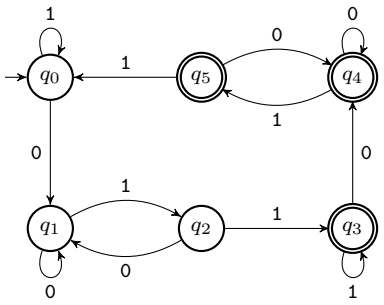




L'espressione regolare $(\mathtt{1} \cup (\mathtt{00}^*\mathtt{1}))^*\mathtt{00}^*$ è una soluzione dell'esercizio. È possibile semplificare l'espressione considerando che $\mathtt{00}^*$ è equivalente a $\mathtt{0}^*\mathtt{0}$, e che $(\mathtt{1} \cup \mathtt{0}^*\mathtt{01})$ è equivalente a $(\mathtt{0}^*\mathtt{1})$. Dunque una espressione regolare equivalente è $(\mathtt{0}^*\mathtt{1})^*\mathtt{0}^*\mathtt{0}$, che a sua volta è equivalente a $(\mathtt{0} \cup \mathtt{1})^*\mathtt{0}$.

Esercizio 3. Si consideri $A = \{x \in \{\mathtt{0},\mathtt{1}\}^* \mid$ il numero di sequenze '011' entro x è dispari$\}$. Ad esempio, $\varepsilon \notin A$, $\mathtt{10111} \in A$ e $\mathtt{101100110} \notin A$. Il linguaggio A è regolare? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio A è regolare. Per dimostrarlo, consideriamo il seguente DFA D:



Dimostriamo che D accetta $x \in \{\mathtt{0},\mathtt{1}\}^*$ se e solo se $x \in A$. La dimostrazione è per induzione sulla lunghezza $|x|$. Se $|x| \leq 2$, allora necessariamente $x \notin A$ perché x non può contenere la sequenza '011' lunga 3; d'altra parte, non è possibile raggiungere uno stato di accettazione in D a partire dallo stato iniziale q_0 utilizzando meno di 3 passi, quindi $D(x)$ rifiuta. Se $|x| = 3$, allora $x \in A$ se e solo se $x = \mathtt{011}$, ed in effetti è facile verificare che questa stringa di 3 bit porta nello stato di accettazione q_3, mentre ogni altra stringa lascia l'automa in uno stato di non accettazione. Supponiamo dunque che l'ipotesi induttiva sia vera per $|x| < n$, e consideriamo una stringa x lunga esattamente $n > 3$ bit. Sia y la stringa costituita dai primi $n-1$ bit di x, e distinguiamo i seguenti casi:

1. y termina con '00', '10' o '11': in questo caso il bit finale di x non può formare una sequenza '011' con gli ultimi due bit di y, pertanto $x \in A$ se e solo se $y \in A$.

   Supponiamo che $x \in A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere q_5, in quanto non esistono transizioni che portano in q_5 leggendo $\mathtt{0}$, e non esistono due transizioni consecutive che possano portare in q_5 leggendo $\mathtt{11}$. Dundue lo stato di $D(y)$ è in $\{q_3, q_4\}$. Se ora l'ultimo bit di x fosse $\mathtt{0}$, l'ultimo stato di $D(x)$ sarebbe necessariamente

q_4, quindi $D(x)$ accetterebbe. Se invece l'ultimo bit di x fosse `1`, l'ultimo stato di $D(x)$ sarebbe necessariamente in $\{q_3, q_5\}$, e dunque $D(x)$ accetterebbe ancora. Pertanto $D(x)$ accetta.

Supponiamo che $x \notin A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto $D(y)$ lascia l'automa in uno stato di non accettazione, che però non potrebbe essere q_2. Infatti non esistono transizioni che portano in q_2 leggendo `0`, e non esistono due transizioni consecutive che possano portare in q_2 leggendo `11`. Dunque lo stato di $D(y)$ è in $\{q_0, q_1\}$. Se ora l'ultimo bit di x fosse `0`, l'ultimo stato di $D(x)$ sarebbe necessariamente q_1, quindi $D(x)$ non accetterebbe. Se invece l'ultimo bit di x fosse `1`, l'ultimo stato di $D(x)$ sarebbe necessariamente in $\{q_0, q_2\}$, e dunque $D(x)$ non accetterebbe ancora. Pertanto $D(x)$ non accetta.

2. y termina con '`01`' e l'ultimo bit di x è `0`: anche in questo caso il bit finale di x non può formare una sequenza '`011`' con gli ultimi due bit di y, pertanto $x \in A$ se e solo se $y \in A$.

   Supponiamo che $x \in A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere né q_3 né q_4, in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo `01`. Dunque lo stato finale di $D(y)$ è q_5, e quindi lo stato finale di $D(x)$ è q_4. Perciò $D(x)$ accetta.

   Supponiamo che $x \notin A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto y lascia l'automa in uno stato di non accettazione, che però non potrebbe essere né q_0 né q_1, in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo `01`. Dunque lo stato finale di $D(y)$ è q_2, e quindi lo stato finale di $D(x)$ è q_1. Perciò $D(x)$ rifiuta.

3. y termina con '`01`' e l'ultimo bit di x è `1`: in questo caso il bit finale di x forma una nuova sequenza '`011`' con gli ultimi due bit di y, pertanto $x \in A$ se e solo se $y \notin A$.

   Supponiamo che $x \in A$, quindi $y \notin A$, dunque per ipotesi induttiva $D(y)$ rifiuta. Pertanto y lascia l'automa in uno stato di non accettazione, che però non potrebbe essere né q_0 né q_1, in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo `01`. Dunque lo stato finale di $D(y)$ è q_2, e quindi lo stato finale di $D(x)$ è q_3. Perciò $D(x)$ accetta.

   Supponiamo che $x \notin A$, quindi $y \in A$, dunque per ipotesi induttiva $D(y)$ accetta. Pertanto y lascia l'automa in uno stato di accettazione, che però non potrebbe essere né q_3 né q_4, in quanto non esistono transizioni consecutive che possano portare in uno di questi stati leggendo `01`. Dunque lo stato finale di $D(y)$ è q_5, e quindi lo stato finale di $D(x)$ è q_0. Perciò $D(x)$ rifiuta.

In sintesi, abbiamo dimostrato per induzione che l'automa D accetta la stringa x se e solo se $x \in A$. Pertanto il linguaggio A è regolare.

Esercizio 4. Sia $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{+}\}$, e sia E il linguaggio su Σ generato dall'espressione regolare $(\mathtt{0} \cup \mathtt{1})^+(\mathtt{+}(\mathtt{0} \cup \mathtt{1})^+)^*$. Sia inoltre $\varphi : E \to \mathbb{N}$ la funzione che associa ad una stringa di E il valore intero corrispondente alla somma dei numeri binari rappresentati. Ad esempio, $\mathtt{0100+11} \in E$ ed inoltre $\varphi(\mathtt{0100+11}) = 7$. Si consideri il linguaggio sull'insieme $\Sigma \cup \{\equiv\}$

$$F = \{x \equiv y \,|\, x, y \in E \text{ e } \varphi(x) = \varphi(y)\}\,.$$

Ad esempio, $\mathtt{0100+11} \equiv \mathtt{101+10} \in F$ mentre $\mathtt{0100+11} \equiv \mathtt{101+01} \notin F$. Il linguaggio F è libero dal contesto (CFL)? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio F non è libero dal contesto. Per dimostrarlo, supponiamo per assurdo che lo sia ed utilizziamo il *pumping lemma* per i CFL. Sia dunque $p > 0$ la *pumping length* di F. Consideriamo la stringa $s = \mathtt{1}^p \equiv \mathtt{1}(\mathtt{+1})^{2^p-1}$; essa appartiene ad F perchè $\varphi(\mathtt{1}^p) = 2^p = \varphi(\mathtt{1}(\mathtt{+1})^{2^p-1})$. Poiché $|s| > p$, deve esistere una suddivisione $s = uvxyz$ tale che $uv^ixy^iz \in F$ per ogni i, $|vy| > 0$ e $|vxy| \leq p$. Consideriamo i seguenti casi:

1. la stringa vy contiene il simbolo '$\equiv$': la stringa pompata verso l'alto o verso il basso contiene un numero di '$\equiv$' diverso da uno, e dunque non può far parte di F.

2. vxy è interamente alla sinistra di '$\equiv$' in s: poiché $|vxy| \leq p$, la stringa vxy è interamente composta da '$\mathtt{0}$'. Pompando verso l'alto il valore φ della sottostringa a sinistra di '$\equiv$' viene moltiplicato per una potenza di due (in quanto $|vy| > 0$ il numero di '$\mathtt{0}$' aumenta almeno di una unità). D'altra parte il valore φ della sottostringa a destra non cambia, e quindi la stringa risultante non può far parte di F.

3. vxy è interamente alla destra di '$\equiv$' in s: pompando verso l'alto la stringa risultante non appartiene a F, in quanto o non è valida (nel caso si producano '$\mathtt{+}$' consecutivi) oppure il valore φ della sottostringa a destra di '$\equiv$' aumenta (poiché vy contiene almeno un '$\mathtt{1}$') mentre il valore di φ per la sottostringa di sinistra non cambia.

4. il simbolo '$\equiv$' è incluso in x: osserviamo che sia v che y devono essere non nulle, altrimenti si ricadrebbe nel caso in cui aumenterebbe il valore φ di una sola delle due sottostringhe. Consideriamo $i = 1$: poiché $|v| > 0$, il valore di φ della sottostringa di sinistra come minimo deve raddoppiare (infatti come minimo si aggiunge un '$\mathtt{0}$' significativo al numero di sinistra), quindi diventa $\geq 2^{p+1}$. D'altra parte, poiché $|vxy| \leq p$, y deve contenere meno di $p/2$ occorrenze della sottostringa '$\mathtt{1+}$' o '$\mathtt{+1}$', quindi il valore di

φ della sottostringa a destra non può essere maggiore di $2^p + p/2$. Poiché $p/2 < 2^p$ per ogni $p > 0$, ne consegue che le due sottostringhe hanno valori φ differenti e la stringa pompata non fa parte di F.

Poiché non esiste modo di suddividere la stringa s in modo da soddisfare il *pumping lemma* si ha una contraddizione. Resta dunque dimostrato che F non è CFL.

Esercizio 5. Date due stringhe non nulle s_1 e s_2 su uno stesso alfabeto Σ, diciamo che le due stringhe sono *totalmente differenti* ($s_1 \not\approx s_2$) se, denotato con $s[i]$ l'i-esimo carattere della stringa s, $s_1[i] = s_2[j]$ implica che $i \neq j$.

Sia $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, e si consideri il linguaggio $B = \left\{ s_1 \# s_2 \,|\, s_1, s_2 \in \Sigma^*, 0 < |s_1| \leq |s_2| \text{ e } s_1 \not\approx s_2^{\mathcal{R}} \right\}$. Ad esempio, $\varepsilon \notin B$, $\# \notin B$, $\mathtt{ca}\#\mathtt{cba} \in B$, e $\mathtt{bc}\#\mathtt{ab} \notin B$. Il linguaggio B è libero dal contesto (CFL)? Si giustifichi la risposta con una dimostrazione.

Soluzione: Il linguaggio B è libero dal contesto. Per dimostrarlo, si consideri la seguente CFG G con variabile iniziale S:

$$
\begin{array}{lcl}
S & \longrightarrow & \mathtt{a}A\mathtt{b} \mid \mathtt{a}A\mathtt{c} \mid \mathtt{b}A\mathtt{a} \mid \mathtt{b}A\mathtt{c} \mid \mathtt{c}A\mathtt{a} \mid \mathtt{c}A\mathtt{b} \\
A & \longrightarrow & \mathtt{a}A\mathtt{b} \mid \mathtt{a}A\mathtt{c} \mid \mathtt{b}A\mathtt{a} \mid \mathtt{b}A\mathtt{c} \mid \mathtt{c}A\mathtt{a} \mid \mathtt{c}A\mathtt{b} \mid B \\
B & \longrightarrow & B\mathtt{a} \mid B\mathtt{b} \mid B\mathtt{c} \mid \#
\end{array}
$$

Dimostriamo che la grammatica genera tutte e sole le stringhe in B. In primo luogo osserviamo che ogni derivazione di G inizia applicando una regola che espande la variabile iniziale S e produce il primo carattere di s_1 (ossia $s_1[1]$) e l'ultimo carattere di s_2 (ossia $s_2^{\mathcal{R}}[1]$).

Cominciamo con la dimostrazione che $L(G) \subseteq B$, procedendo per induzione sulla lunghezza di s_1. Come base per l'induzione consideriamo il caso $|s_1| = 1$. Poiché $s_1[1]$ è generato da una regola che espande la variabile iniziale S seguita immediatamente dalla regola $A \longrightarrow B$, è immediato verificare che i caratteri $s_1[1]$ e $s_2^{\mathcal{R}}[1]$ devono necessariamente essere differenti. L'applicazione delle regole che espandono B può solo aggiungere caratteri alla stringa s_2. Pertanto $s_1 \not\approx s_2$ e $|s_2| \geq 1 = |s_1|$, quindi ogni stringa prodotta da G con $|s_1| = 1$ appartiene a B. Assumiamo ora come ipotesi induttiva che l'asserto sia vero per ogni stringa $u\#v$ generata da G con $|u| < n$, e consideriamo una stringa $s_1\#s_2 \in L(\text{`}G)$ tale che $|s_1| = n$. In ogni derivazione da S ad una stringa terminale deve essere sempre applicata la regola $A \longrightarrow B$, perché B è l'unica variabile che può espandere con una regola senza variabili a destra. Consideriamo la sequenza di derivazioni ottenute da quella che produce la stringa

$s_1\#s_2$ rimuovendo la regola applicata subito prima $A \longrightarrow B$. Poiché per ipotesi $|s_1| = n > 1$, tale regola ha la forma $A \longrightarrow xAy$, con $x, y \in \Sigma$. Pertanto la derivazione ottenuta togliendo questa regola è ancora valida e produce una stringa $u\#wv$ con $u, v, w \in \Sigma^*$, $|u| = |v| = n-1$, $s_1 = ux$ e $s_2 = wyv$. Per ipotesi induttiva si ha che $u\#wv \in B$, dunque $u \not\approx (wv)^{\mathcal{R}} = v^{\mathcal{R}}w^{\mathcal{R}}$. Ciò significa che le stringhe u e $v^{\mathcal{R}}$ sono totalmente differenti. Inoltre la regola rimossa $A \longrightarrow xAy$ garantisce che $x \neq y$. Perciò $ux \not\approx v^{\mathcal{R}}y$, quindi $s_1 = ux \not\approx v^{\mathcal{R}}yw^{\mathcal{R}} = s_2^{\mathcal{R}}$, e finalmente $s_1\#s_2 \in B$.

Dimostriamo ora che $B \subseteq L(G)$, ossia che la grammatica è in grado di generare qualunque stringa appartenente a B. Procediamo per induzione sulla lunghezza della sottostringa a sinistra del carattere '#'. La base dell'induzione è il caso in cui tale sottostringa è lunga un carattere, ossia il caso di una stringa in B con la forma $x\#vy$, con $x, y \in \Sigma$. L'appartenenza in B implica che $x \not\approx (vy)^{\mathcal{R}} = yv^{\mathcal{R}}$, o equivalentemente che $x \neq y$. Pertanto, detta $m = |v|$, la grammatica G può derivare questa stringa utilizzando in sequenza le regole $S \longrightarrow xAy$, $A \longrightarrow B$, $B \longrightarrow B\,v[m]$, $\ldots$, $B \longrightarrow B\,v[1]$, $B \longrightarrow \#$. Quindi $x\#vy \in L(G)$. Assumiamo ora come ipotesi induttiva che l'asserto sia vero per ogni stringa in B avente una sottostringa a sinistra del '#' di lunghezza inferiore a $n > 1$. Consideriamo una stringa $s_1\#s_2 \in B$ con $|s_1| = n$. Sia $x = s_1[n] \in \Sigma$ l'ultimo carattere di s_1, e sia $y = s_2^{\mathcal{R}}[n] \in \Sigma$ (esiste certamente perché $|s_2| \geq |s_1|$); siano inoltre $u, v, w \in \Sigma^*$ tali che $s_1 = ux$ e $s_2 = wyv$, con $|u| = |v| = n-1$. L'appartenenza in B implica che $s_1 \not\approx s_2^{\mathcal{R}}$, ossia che $ux \not\approx (wyv)^{\mathcal{R}} = v^{\mathcal{R}}yw^{\mathcal{R}}$. Di conseguenza, $u \not\approx v^{\mathcal{R}}$, quindi $u \not\approx v^{\mathcal{R}}w^{\mathcal{R}} = (wv)^{\mathcal{R}}$. Perciò $u\#wv \in B$, dunque per ipotesi induttiva $u\#wv \in L(G)$. Consideriamo la derivazione da S alla stringa $u\#wv$: deve necessariamente essere applicata la regola $A \longrightarrow B$, altrimenti non si potrebbe produrre una stringa di soli terminali. Modifichiamo ora la derivazione inserendo subito prima dell'applicazione della regola $A \longrightarrow B$ la regola $A \longrightarrow xAy$: la regola esiste certamente perché $x \neq y$, e si può applicare perchè si applica alla medesima variabile A. La nuova derivazione produce esattamente la stringa $ux\#wyv$, e quindi $s_1\#s_2 \in L(G)$.

Esercizio 6. Si consideri l'insieme $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ e la seguente grammatica G con variabile iniziale S:

$$S \quad \longrightarrow \quad \mathtt{a}S\mathtt{b} \quad | \quad \mathtt{b}S \quad | \quad S\mathtt{a} \quad | \quad \varepsilon$$

Quale è l'insieme di stringhe $L(G)$ generate da G? Giustificare la risposta con una dimostrazione.

Soluzione: Il linguaggio generato dalla grammatica include tutte le stringhe sull'alfabeto di terminali Σ, ossia $L(G) = \Sigma^*$. La dimostrazione è per induzione. Come base dell'induzione,

consideriamo che la stringa vuota ε è generata dalla grammatica grazie alla regola $S \longrightarrow \varepsilon$. Supponiamo dunque come ipotesi induttiva che la grammatica sia in grado di generare tutte le stringhe in Σ^* di lunghezza inferiore a $n \geq 0$, e consideriamo una stringa x di lunghezza n. Possiamo distinguere i seguenti tre casi:

1. il primo carattere di x è '`b`': Si consideri una derivazione da S in cui la prima regola applicata è $S \longrightarrow \mathtt{b}\,S$: la stringa ottenuta è pertanto $\mathtt{b}\,S$. Poiché $x = \mathtt{b}\,y$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y. Perciò la derivazione costituita dalla regola che genera il primo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x. Pertanto $x \in L(G)$.

2. l'ultimo carattere di x è '`a`': Si consideri una derivazione da S in cui la prima regola applicata è $S \longrightarrow S\,\mathtt{a}$: la stringa ottenuta è pertanto $S\,\mathtt{a}$. Poiché $x = y\,\mathtt{a}$ ove $y \in \Sigma^*$ e $|y| = n - 1$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y. Perciò la derivazione costituita dalla regola che genera l'ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x. Pertanto $x \in L(G)$.

3. il primo carattere di x è '`a`' e l'ultimo carattere di x è '`b`': Si consideri una derivazione da S in cui la prima regola applicata è $S \longrightarrow \mathtt{a}\,S\,\mathtt{b}$: la stringa ottenuta è pertanto $\mathtt{a}\,S\,\mathtt{b}$. Poiché $x = \mathtt{a}\,y\,\mathtt{b}$ ove $y \in \Sigma^*$ e $|y| = n - 2$, per l'ipotesi induttiva esiste una derivazione da S che genera la stringa terminale y. Perciò la derivazione costituita dalla regola che genera il primo e l'ultimo carattere di x seguita dalle regole che generano y costituiscono una derivazione della stringa x. Pertanto $x \in L(G)$.

In tutti i tre casi si conclude che $x \in L(G)$. Pertanto resta dimostrato che $x \in L(G)$ per ogni possibile stringa di Σ^*, e quindi $L(G) = \Sigma^*$.

Esercizio 7. Dimostrare che se P=NP allora ogni linguaggio in P diverso da $\emptyset$ e da Σ^* è NP-completo.

Soluzione: La dimostrazione è in effetti molto semplice considerando che una riduzione polinomiale è basata su una macchina di Turing deterministica che ha la capacità di risolvere direttamente i problemi in P, e quindi in NP, poiché assumiamo che P=NP.

Formalmente, consideriamo un qualunque linguaggio $A \in$ P diverso dagli insiemi banali, ossia dall'insieme vuoto $\emptyset$ e dall'insieme contenente tutte le stringhe Σ^*, e dimostriamo che A è

NP-completo. Innanzi tutto dobbiamo provare che $A \in$ NP, ma questo è immediato perché per ipotesi $A \in$ P e P=NP. Mostriamo adesso che A è NP-hard. Sia dunque $B \in$ NP. Poiché P=NP, $B \in$ P, dunque esiste una DTM M che decide B. Trasformiamo M in un'altra DTM N che, per ogni istanza x, simula l'esecuzione di $M(x)$. Se $M(x)$ accetta, N si ferma lasciando sul nastro la codifica di un elemento $I_y \in A$ (esiste certamente perché $A \neq \emptyset$). Se invece $M(x)$ rifiuta, N si ferma lasciando sul nastro la codifica di un elemento $I_n \notin A$ (esiste certamente perché $A \neq \Sigma^*$). La DTM N esegue in tempo polinomiale e calcola una istanza-sì di A per ogni istanza-sì di B, ed una istanza-no di A per ogni istanza-no di B. Dunque N costituisce una riduzione polinomiale da B ad A. Poiché B è un generico problema in NP, A è NP-hard. La conclusione è che A è NP-completo.

Esercizio 8. Siano dati un grafo non diretto $G = (V, E)$, una coppia di nodi $a, b \in G$, ed un intero $k \in \mathbb{N}$. Si dimostri che il problema di determinare se esiste un percorso da a a b in G di lunghezza non superiore a k è in P.

Soluzione: Il problema è noto come SHORTEST PATH, ed è una generalizzazione del problema polinomiale PATH che richiede l'esistenza di un percorso tra una coppia di nodi a prescindere dalla sua lunghezza. In effetti una idea per dimostrare che SHORTEST PATH è in P consiste nel modificare l'algoritmo utilizzato per dimostrare che PATH è in P, in modo da considerare anche la lunghezza dei percorsi. Si osservi inoltre che non esiste una reale differenza tra la versione su grafi non diretti e quella su grafi diretti.

M= " On input $\langle G, a, b, k\rangle$, where G is a graph, $a, b \in V(G)$, $k \in \mathbb{N}$
1. Allocate a vector of lengths having $|V(G)|$ elements (one for each node)
2. Initialize all lengths to ∞
3. Set the length of node a to 0
3. For $i = 0$ to $|E(G)|$:
    4. For each edge $e \in E(G)$:
        5. If e links a node u with length i and a node v with length ∞:
            6. Set the length of v to $i + 1$
7. If b has length $\leq k$, accept; otherwise, reject"

La DTM inizia a marcare il nodo a con la lunghezza 0; poi marca tutti i nodi adiacenti ad a con la lunghezza 1; poi procede a marcare tutti i nodi adiacenti ai nodi di lunghezza 1 che non erano stati già marcati con la lunghezza 2; e così via. Il ciclo esterno è ripetuto per un numero di volte pari al numero di archi nel grafo (la lunghezza del più lungo percorso possibile), e ad ogni iterazione vengono scanditi tutti gli archi del grafo. Pertanto la complessità temporale

dell'algoritmo è $O(|E|^2)$. È inoltre evidente che alla fine il nodo b viene marcato con un valore $\leq k$ se e solo se esiste un percorso entro il grafo da a a b di lunghezza non superiore a k. Pertanto M è un decisore in tempo polinomiale, e quindi SHORTEST PATH è in P.

Si osservi che l'analogo problema LONGEST PATH, che richiede di verificare l'esistenza di un percorso di lunghezza non inferiore ad un valore dato, è NP-completo.

Esercizio 9. Uno stato interno r di una macchina di Turing M è detto *inutile* se non esiste alcuna stringa di input che possa portare la macchina M ad assumere lo stato interno r. Dimostrare che il linguaggio contenente le codifiche di tutte le macchine di Turing con uno o più stati inutili è indecidibile.

Soluzione: Cominciamo con l'osservazione evidente che la proprietà di avere o meno uno stato inutile appartiene alla particolare macchina di Turing, e non è propria del linguaggio riconosciuto dalla macchina. Ad esempio, consideriamo una TM che non ha stati inutili, e modifichiamo la sua descrizione in modo da aggiungere uno stato che non è mai utilizzato in alcuna transizione. Ovviamente la macchina modificata riconosce lo stesso linguaggio della macchina originale ma possiede uno stato inutile. Dunque non possiamo dimostrare quanto richiesto utilizzando il Teorema di Rice.

Per dimostrare l'indecidibilità del linguaggio contenente le codifiche di TM aventi almeno uno stato inutile costruiamo una riduzione dal problema di accettazione delle TM $\mathcal{A}_{\text{TM}}$. Come al solito dunque consideriamo una istanza $\langle M, w\rangle$ di $\mathcal{A}_{\text{TM}}$ e mostriamo come costruire una TM N che ha uno stato inutile se e solo se $M(w)$ accetta. La difficoltà di questa costruzione, però, risiede nella macchina M stessa: infatti, dovendo simulare M, N include nei suoi stati interni anche gli stati interni di M. Se M avesse stati inutili, la macchina N potrebbe avere stati inutili anche se in effetti $M(w)$ non accettasse. Dobbiamo quindi fare attenzione a costruire N in modo da rendere ogni stato di M "utile".

Siano Σ e Γ rispettivamente gli alfabeti di input e di nastro della macchina M; la TM N utilizzerà $\Sigma' = \Sigma \cup \{\#\}$ e $\Gamma' = \Gamma \cup \{\#\}$, ove '#' è un nuovo simbolo non utilizzato in M. Il simbolo '#' serve ad implementare un ciclo in cui si entra in ogni stato interno di M: in pratica viene aggiunta una nuova transizione per ogni stato interno di M in cui leggendo '#' si passa in uno stato interno di N che continua il ciclo.

Nella descrizione di N seguente si deve assumere che non vengono introdotti stati interni propri di N inutili a meno che questo che non sia affermato esplicitamente. In particolare, lo stato interno r di N è utilizzato solo quando esplicitamente menzionato. Inoltre si tenga presente che gli stati di accettazione e rifiuto di M non sono stati finali per N.

N = " On input x, where $x \in \Sigma'^*$:

1. If $x =$ '#':
   2. Execute a loop and enter in every internal state of M
   3. Halt
4. Run M on input w
5. If $M(w)$ accepts, enter in internal state r
6. Halt"

Supponiamo per assurdo che il problema di decidere se una TM ha stati interni inutili sia decidibile, e sia dunque R un decisore per tale problema. Consideriamo allora la seguente TM D:

D = " On input $\langle M, w\rangle$, where M is a TM and $w \in \Sigma^*$:

1. Build from $\langle M, w\rangle$ the encoding of the TM N
2. Run R on $\langle N\rangle$
3. If $R(\langle N\rangle)$ accepts, then reject
4. Otherwise, if $R(\langle N\rangle)$ rejects, then accept"

Se $R(\langle N\rangle)$ accetta, allora N ha almeno uno stato interno inutile. Per costruzione di N questo stato può essere unicamente r. Ciò significa che $M(w)$ non accetta. Se invece $R(\langle N\rangle)$ rifiuta, allora ogni stato interno di N è utile, quindi anche lo stato interno di r deve essere visitato per un certo input x. In effetti l'input x di N viene ignorato se è diverso da '#'. L'unica possibilità è che $M(w)$ accetta e quindi r sia utilizzato nel passo 5 di N. Poiché D complementa la decisione di R, D è in effetti un decisore per $\mathcal{A}_{\text{TM}}$, ma questo è in contraddizione con il fatto che $\mathcal{A}_{\text{TM}}$ è indecidibile. Pertanto resta dimostrato che il problema di decidere se una TM ha stati interni inutili è indecidibile.

Esercizio 10. Si consideri il linguaggio costituito dalle codifiche delle formule booleane che hanno almeno due assegnazioni di verità che soddisfano la formula. Dimostrare che tale linguaggio è NP-completo.

Soluzione: Questo problema è generalmente chiamato DOUBLE SAT, ed è una generalizzazione molto semplice del problema SAT.

Si dimostra facilmente che DOUBLE SAT è in NP. Infatti, data una qualunque istanza $\langle\Phi\rangle$ che codifica una formula booleana, un certificato per l'esistenza di una soluzione è costituito da due liste di valori di verità. Il verificatore controlla che ciascuna lista contenga esattamente

un valore (vero o falso) per ciascuna variabile di Φ, e che entrambe le assegnazioni di verità soddisfino la formula Φ.

Per dimostrare che DOUBLE SAT è NP-hard consideriamo la seguente riduzione dal problema SAT: considerata una generica istanza $\langle\Phi\rangle$ di SAT, sia Φ' la formula booleana ottenuta da Φ aggiungendo una nuova variabile v e ponendo

$$\Phi' = \Phi \wedge (v \vee \overline{v}).$$

È immediato verificare che se la formula Φ è soddisfacibile allora esistono almeno due assegnazioni di verità che soddisfano Φ': la assegnazione che soddisfa Φ estesa con $v = T$ e la stessa assegnazione estesa con $v = F$. Viceversa, se la formula Φ non è soddisfacibile, allora nemmeno Φ' può essere soddisfacibile, perché la variabile v non appare nella formula Φ e quindi non può contribuire a rendere quella parte della formula Φ' soddisfacibile.

Da tutto ciò si può concludere che DOUBLE SAT è NP-completo.

Esercizio 11. Si consideri un grafo non diretto $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Si dimostri che il problema di stabilire se il grafo G contiene un sottoinsieme W di nodi con $|W| \geq \frac{n}{2}$ tale che non esiste alcun arco tra i nodi di W è NP-completo.

Soluzione: Questo problema è generalmente noto con il nome HALF INDEPENDENT SET (HIS). Dimostrare che HIS è in NP è molto semplice. Data una istanza $\langle G\rangle$, un certificato per l'esistenza di una soluzione è una lista di nodi del grafo G. Il verificatore controlla che nella lista non vi siano nodi ripetuti, che la lista contenga almeno $|V(G)|/2$ nodi, e che ciascuna coppia di nodi nella lista sia non adiacente nel grafo G, ossia non esista tra essi un arco. Tutti questi controlli possono naturalmente essere effettuati in tempo polinomiale nella dimensione del grafo.

Per dimostrare che HIS è NP-hard mostriamo una riduzione dall'analogo problema NP-completo INDEPENDENT SET (IS). Si presti attenzione che le istanze di IS e di HIS hanno una differente struttura: le istanze di IS codificano un grafo G ed un numero intero k, mentre le istanze di HIS codificano soltanto un grafo. Dunque la riduzione deve "eliminare" il parametro k dall'istanza di IS.

La riduzione considera il valore del parametro k ed opera differentemente nei casi $k = n/2$, $k < n/2$, e $k > n/2$. Si ricordi che una riduzione è una funzione calcolabile in tempo polinomiale, e confrontare due valori interi è una operazione eseguibile in tempo polinomiale nella lunghezza delle codifiche dei valori.

1. $\underline{k = n/2}$: data l'istanza di IS $\langle G, k\rangle$, la riduzione costruisce l'istanza $\langle G\rangle$ (si elimina semplicemente il valore k dall'istanza). Infatti, una istanza-sì di IS è tale per cui esiste un sottoinsieme $W \subseteq V$ con $|W| \geq k = n/2$ tale che non esiste alcun arco tra i nodi in W; il grafo è dunque anche una istanza-sì di HIS. Ovviamente vale anche il viceversa.

2. $\underline{k < n/2}$: data l'istanza di IS $\langle G, k\rangle$, la riduzione costruisce l'istanza $\langle G'\rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $n-2k$ nodi isolati. Sia $\langle G, k\rangle$ una istanza-sì di IS, dunque esista un insieme indipendente $W \subseteq V$ con $|W| \geq k$. Nel grafo G', W unito ai nuovi nodi costituisce un insieme indipendente di dimensione $k+(n-2k) = n-k$ nodi. D'altra parte, G' contiene $n+(n-2k) = 2(n-k)$ nodi, quindi G' è una istanza-sì di HIS.

   Se invece $\langle G, k\rangle$ è una istanza-no di HIS, non esiste alcun insieme indipendente di dimensione maggiore o uguale a k in G. Nel grafo G' pertanto non esiste alcun insieme indipendente di dimensione maggiore o uguale a $n-k$, perché G' è ottenuto da G aggiungendo solo $n-2k$ nodi e non rimuovendo alcun arco. Dunque G' è una istanza-no di HIS, perché nessun insieme indipendente può essere di dimensione maggiore o uguale alla metà dei $2(n-k)$ nodi di G'.

3. $\underline{k > n/2}$: data l'istanza di IS $\langle G, k\rangle$, la riduzione costruisce l'istanza $\langle G'\rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $2k-n$ nodi ed i seguenti archi: tutti i nuovi nodi sono collegati tra loro (quindi costituiscono un insieme completo), inoltre ciascuno dei nuovi nodi è collegato a ciascuno dei nodi del grafo G. Ovviamente, una istanza-sì di IS è anche una istanza-sì di G'. Infatti G' ha $n+(2k-n) = 2k$ nodi, e l'insieme indipendente W con $|W| \geq k$ è anche un insieme indipendente di G' con almeno la metà dei nodi di G'.

   Consideriamo ora una istanza-no di IS, dunque supponiamo che nel grafo G non esista alcun insieme indipendente di dimensione maggiore o uguale a k. Il grafo G' è ottenuto da G aggiungendo un sottografo completo ed aggiungendo tutti gli archi tra i vecchi ed i nuovi nodi. Pertanto, nessuno dei nuovi nodi può essere aggiunto ad un insieme indipendente di G. Si conclude dunque che non può esistere in G' un insieme indipendente avente la metà dei nodi di G', quindi $\langle G'\rangle$ è una istanza-no di HIS.

È facile verificare che la dimensione dell'istanza $\langle G'\rangle$ è polinomialmente limitata dalla dimensione dell'istanza $\langle G, k\rangle$ (infatti, $k \leq n$, quindi in ogni caso $|V(G')| \leq 2n$). Inoltre la riduzione è calcolabile in tempo polinomiale. Quindi HIS è NP-hard, e di conseguenza NP-completo.

Esercizio 12. Si consideri un grafo non diretto $G = (V, E)$ con $|V| = n$ nodi e $|E| = m$ archi. Si dimostri che il problema di stabilire se il grafo G contiene un sottoinsieme W di nodi con $|W| \leq \frac{n}{2}$ tale che ogni arco di G contiene almeno un nodo in W è NP-completo.

Soluzione: Questo problema è generalmente noto con il nome Half Vertex Cover (HVC). Dimostrare che HVC è in NP è molto semplice. Data una istanza $\langle G \rangle$, un certificato per l'esistenza di una soluzione è una lista di nodi del grafo G. Il verificatore controlla che nella lista non vi siano nodi ripetuti, che la lista contenga non più di $|V(G)|/2$ nodi, e che ciascun arco di G sia collegato ad un nodo della lista. Tutti questi controlli possono naturalmente essere effettuati in tempo polinomiale nella dimensione del grafo.

Assumendo come dimostrato che il problema Half Independent Set (introdotto nell'esercizio precedente) è NP-hard, è immediato verificare che HVC è NP-hard. Infatti, un insieme di nodi W in un grafo G è un insieme indipendente se e solo se l'insieme di nodi $U = V(G) \setminus W$ è un ricoprimento tramite vertici ("vertex cover"). Dunque esiste un insieme indipendente di dimensione almeno $|V(G)|/2$ se e solo se esiste un ricoprimento tramite vertice di dimensione al più $|V(G)|/2$.

In alternativa, mostriamo una riduzione tra Independent Set (IS) e HVC. Si presti attenzione che le istanze di IS e di HVC hanno una differente struttura: le istanze di IS codificano un grafo G ed un numero intero k, mentre le istanze di HVC codificano soltanto un grafo. Dunque la riduzione deve "eliminare" il parametro k dall'istanza di IS.

La riduzione considera il valore del parametro k ed opera differentemente nei casi $k = n/2$, $k < n/2$, e $k > n/2$. Si ricordi che una riduzione è una funzione calcolabile in tempo polinomiale, e confrontare due valori interi è una operazione eseguibile in tempo polinomiale nella lunghezza delle codifiche dei valori.

1. $\underline{k = n/2}$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G \rangle$ (si elimina semplicemente il valore k dall'istanza). Infatti, una istanza-sì di IS è tale per cui esiste un sottoinsieme $W \subseteq V$ con $|W| \geq k = n/2$ tale che non esiste alcun arco tra i nodi in W; l'insieme $U = V \setminus W$ è un ricoprimento tramite vertici con $|U| \leq n - k = n/2$; il grafo è dunque anche una istanza-sì di HVC. Ovviamente vale anche il viceversa.

2. $\underline{k < n/2}$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $n-2k$ nodi isolati. Sia $\langle G, k \rangle$ una istanza-sì di IS, dunque esista un insieme indipendente $W \subseteq V$ con $|W| \geq k$. Nel grafo G', W unito ai nuovi nodi costituisce un insieme indipendente W' di dimensione $k + (n - 2k) = n - k$ nodi. Pertanto $U = V(G') \setminus W' = V \setminus W$ ha $|U| = n - (n - k) = k$ nodi ed è un ricoprimento tramite vertici per G'. D'altra parte, G' contiene $n + (n - 2k) = 2(n - k)$ nodi, e $k < n/2$ implica $k < n - k$, quindi G' è una istanza-sì di HVC.

   Se invece $\langle G, k \rangle$ è una istanza-no di HIS, non esiste alcun insieme indipendente di dimensione maggiore o uguale a k in G. Nel grafo G' pertanto non esiste alcun insieme indipendente di dimensione maggiore o uguale a $n-k$, perché G' è ottenuto da G aggiun-

gendo solo $n-2k$ nodi e non rimuovendo alcun arco. Dunque in G' nessun ricoprimento tramite vertici può essere di dimensione minore o uguale a $2(n-k)-(n-k)=n-k$, e quindi $\langle G' \rangle$ è una istanza-no di HVC.

3. $k > n/2$: data l'istanza di IS $\langle G, k \rangle$, la riduzione costruisce l'istanza $\langle G' \rangle$ in cui il grafo G' è costituito dal grafo G al quale sono aggiunti $2k-n$ nodi ed i seguenti archi: tutti i nuovi nodi sono collegati tra loro (quindi costituiscono un insieme completo), inoltre ciascuno dei nuovi nodi è collegato a ciascuno dei nodi del grafo G. Ovviamente, una istanza-sì di IS è anche una istanza-sì di G'. Infatti G' ha $n+(2k-n)=2k$ nodi, e l'insieme indipendente W con $|W| \geq k$ è anche un insieme indipendente di G' con almeno la metà dei nodi di G'; pertanto il complemento di questo insieme è un ricoprimento tramite vertici con al più la metà dei nodi di G'.

   Consideriamo ora una istanza-no di IS, dunque supponiamo che nel grafo G non esista alcun insieme indipendente di dimensione maggiore o uguale a k. Il grafo G' è ottenuto da G aggiungendo un sottografo completo ed aggiungendo tutti gli archi tra i vecchi ed i nuovi nodi. Pertanto, nessuno dei nuovi nodi può essere aggiunto ad un insieme indipendente di G. Si conclude dunque che non può esistere in G' un insieme indipendente avente la metà dei nodi di G'. Pertanto, ogni ricoprimento tramite vertici in G' deve contenere più della metà dei nodi di G', dunque $\langle G' \rangle$ è una istanza-no di HVC.

È facile verificare che la dimensione dell'istanza $\langle G' \rangle$ è polinomialmente limitata dalla dimensione dell'istanza $\langle G, k \rangle$ (infatti, $k \leq n$, quindi in ogni caso $|V(G')| \leq 2n$). Inoltre la riduzione è calcolabile in tempo polinomiale. Quindi HVC è NP-hard, e di conseguenza NP-completo.