Esercizi di calcolabilità

Esercizio 1 (nessun limite al tempo). Data un'enumerazione effettiva, esiste una funzione calcolabile totale $t(i,n)$ che maggiora il tempo (>0) di calcolo di $M_i(n)$?

Svolgimento. Una tale funzione non esiste.[1]

Si definisca la funzione

$$t(i,n) = \begin{cases} k & \text{se } M_i(n)\downarrow \text{ in meno di } k \text{ passi} \\ 0 & \text{altrimenti} \end{cases}$$

e si supponga per assurdo $t(i,n)$ calcolabile totale.

Sia allora T_i la misura *esatta* del tempo di calcolo di M_i. Per ogni n, la funzione test $T_i(n) \stackrel{?}{\leq} t(i,n)$ è calcolabile totale poiché basta far girare $M_i(n)$ per al più $t(i,n)$ passi: se l'algoritmo si è arrestato allora il test ha risultato affermativo, altrimenti il test ha risultato negativo. Si definisca la funzione ausiliaria

$$\psi(x) = \begin{cases} \varphi_x(x)+1 & \text{se } T_x(x) \leq t(x,x) \\ 0 & \text{altrimenti} \end{cases}$$

e, per il ragionamento appena svolto, si osservi che $\psi(x)$ è calcolabile totale. Quindi per la tesi di Church-Turing esiste un indice i tale che $\psi = \varphi_i$. Ma da

$$\varphi_i(i) = \psi(i) = \begin{cases} \varphi_i(i)+1 & \text{se } T_i(i) \leq t(i,i) \\ 0 & \text{altrimenti} \end{cases}$$

e dalla osservazione $\varphi_i(i) \neq \varphi_i(i)+1$, segue che $\varphi_i(i)=0$ e dunque $T_i(i) > t(i,i)$. Ciò contraddice l'ipotesi $T_i(i) \leq t(i,i)$ e quindi $t(i,n)$ non è calcolabile totale.

Esercizio 2 (nessun limite allo spazio). Esiste una funzione calcolabile totale che determina se un dato programma M_i calcolato su uno specifico dato x all'interno di un calcolatore C con memoria finita è in ciclo?

Svolgimento. Si definisca

$$h(i,x,C) = \begin{cases} 1 & \text{se } M_i(x)\uparrow \text{ sul calcolatore } C \\ 0 & \text{altrimenti} \end{cases}$$

La funzione totale h è calcolabile? Aver limitata la memoria di C ci permette di stimare il numero massimo di configurazioni di C, e dunque di avere una

[1] Si pensi alle *barre di avanzamento* mostrate da molti programmi: il loro movimento procede a scatti. Se esistesse $t(i,n)$ calcolabile totale, dal confronto del tempo già passato con $t(i,n)$ sarebbe possibile far scorrere una barra di avanzamento in modo continuo.

procedura che in un numero finito di passi stabilisce se $M_i(x)$ è in ciclo o meno.[2] Si rappresenti infatti il calcolatore C come una MdT e siano $n = \#\Sigma_C, m-1 = \#Q_C$ e k le celle del nastro di C. Si possono avere al più n^k stringhe diverse sul nastro di C, su cui la testina si può trovare in al più k posizioni diverse, in al più m stati diversi. Dunque il numero massimo di configurazioni differenti che C può assumere è $l = k \cdot n^k \cdot m$. Sia H la MdT a 4 nastri che al principio contengono sul primo $\langle M_i \rangle$, codifica della MdT M_i, sul secondo $\langle x \rangle$, codifica del dato x, sul terzo la codifica dello stato iniziale di M_i e sul quarto la codifica in unario di l. La MdT H simula il comportamento di $M_i(x)$ lavorando come la MdT universale descritta nel corso, e in più decrementa di una tacca il quarto nastro ogni volta che simula un passo della computazione di $M_i(x)$. Se sul terzo nastro compare lo stato di arresto di M_i e sul quarto nastro vi sono ancora tacche allora H si ferma e restituisce $h(i, x, C) = 0$, altrimenti se sul quarto nastro non vi sono più tacche ed M_i non è nello stato di arresto allora H si ferma e restituisce $h(i, x, C) = 1$ poiché M_i è destinata a ritornare in una delle configurazioni già incontrate.

È necessario notare come l'aver definito i nastri delle MdT infiniti a destra renda possibile la definizione della macchina H. Se imponessimo un limite di spazio, in particolare al quarto nastro di H, sarebbe possibile trovare un calcolatore C con un l maggiore del limite, e dunque la funzione $h(i, x, C)$ diverrebbe non calcolabile.

Detto in altre parole, se non avessimo il vincolo della memoria finita di C, la funzione $h(i, x)$ (a due posti invece che tre) non sarebbe calcolabile, infatti risulterebbe $h(i, i) = 1 - \chi_K(i)$, che sappiamo non essere calcolabile.

Esercizio 3. Si dimostri che un insieme infinito A è ricorsivo se e solo se può essere generato in modo strettamente crescente, cioè esiste una funzione strettamente crescente g tale che $A = Imm(g)$.

Svolgimento. Se A è ricorsivo allora la sua funzione caratteristica è ricorsiva, dunque possiamo definire (con un po' di libertà) la funzione g come

$$\begin{cases} g(0) & = \mu y[\chi_A(y) = 1] \\ g(n+1) & = \mu y[\chi_A(y) = 1 \wedge g(n) < y] \end{cases}$$

La funzione che verfica $<$ è primitiva ricorsiva, e la costruenda funzione g è strettamente crescente. Per mostrare che $A = Imm(g)$ basta ordinare in modo crescente gli elementi di A, cioè riordinare $A = \{a_i\}_i$ in modo che $a_{i_{n+1}}$ sia l'elemento di A immediatamente successivo ad a_{i_n}, e osservare che per ogni n

[2] Nei primi calcolatori, l'operatore sedeva alla consolle, che era munita di una fila di lampadine che mostravano lo stato di avanzamento del programma; uno dei compiti dell'operatore era proprio quello di interrompere l'esecuzione, avendo rilevato un ciclo dal fatto che che la stessa configurazione si ripeteva.

vale $a_{i_n} = g(n)$. In simboli

$$\begin{aligned} A &= \{a_{i_0}, a_{i_1}, \ldots, a_{i_n}, \ldots\} \\ &= \{g(0), g(1), \ldots, g(n), \ldots\} \end{aligned}$$

Se $A = Imm(g)$, con g calcolabile totale strettamente crescente, per dimostrare che A è ricorsivo bisogna far vedere che χ_A è una funzione calcolabile totale. Si definisca allora la seguente funzione che lo è banalmente (B è un insieme finito):

$$\varphi_A(n) = \begin{cases} 1 & \text{se } n \in B = \{g(0), \ldots, g(n)\} \subset A \\ 0 & \text{altrimenti} \end{cases}$$

Verifichiamo che $\varphi_A = \chi_A$, cioè che la funzione appena definita è davvero la funzione caratteristica di A:

- se $n \in B$ allora $n \in A$ per definizione
- se $n \notin B$ allora $n < g(n) < g(n+1) < \ldots$ e quindi $n \notin A$

Esercizio 4. Si dimostri che ogni insieme ricorsivamente enumerabile non ricorsivo (quindi non vuoto) ha almeno un sottoinsieme ricorsivo infinito.

Svolgimento. Sia f la funzione calcolabile totale che enumera l'insieme r.e. dato. A partire da f si costruisca la seguente funzione:

$$\begin{cases} g(0) &= f(0) \\ g(n+1) &= f(\mu\, k[f(k) > g(n)]) \end{cases}$$

L'immagine di g è un insieme infinito. Inoltre tale funzione è *strettamente crescente*, e quindi, per l'esercizio 3, la sua immagine è un insieme ricorsivo.

Esercizio 5. Si definisca l'insieme

$$W_i = \{n \mid \varphi_i(n) \downarrow\} = dom(\varphi_i)$$

e si considerino due funzioni calcolabili φ_i e φ_j tali che $\emptyset \neq W_i \subsetneq W_j$. Si discuta dell'esistenza di una funzione calcolabile totale f tale che per ogni x vale

$$W_{f(x)} = \begin{cases} W_i & \text{se } \varphi_x(x) \uparrow, \text{ cioè se } x \notin K \\ W_j & \text{se } \varphi_x(x) \downarrow, \text{ cioè se } x \in K \end{cases}$$

Svolgimento. Contrariamente a quanto suggerisce l'intuizione, una tale funzione calcolabile totale esiste. Sappiamo che W_i è r.e. e non vuoto, dunque esiste una funzione calcolabile totale h_i tale che $W_i = Imm(h_i)$, e similmente esiste h_j tale che $W_j = Imm(h_j)$. La seguente procedura genera gli (infiniti) elementi di $W_{f(x)}$ e poiché è definita per ogni x, la f è totale:

```
n:=0;
W_{f(x)} := ∅;
while (φ_x(x) non converge in n passi) do
    W_{f(x)} := W_{f(x)} ∪ h_i(n);
    n:=n+1;
n:=0;
while true do
    W_{f(x)} := W_{f(x)} ∪ h_j(n);
    n:=n+1
```

A parole: essendo $W_i \subsetneq W_j$, posso iniziare a costruire $W_{f(x)}$ generando alcuni elementi di W_i con la funzione h_i e inserendoli in $W_{f(x)}$; non appena mi accorgo che la $\varphi_x(x)$ termina inizio ad emettere gli elementi di W_j (alcuni magari già calcolati) in $W_{f(x)}$ con la funzione h_j.

Si noti che se W_i e W_j fossero disgiunti, la funzione f sarebbe la caratteristica di K e quindi *non* sarebbe calcolabile totale.

Esercizio 6 (William E. Dowling). Si consideri il contesto in cui un programma P viene eseguito sul dato x all'interno di un *ambiente*, creato e gestito da un sistema operativo OS fissato. Un programma è detto *virus* se altera il codice di OS durante la sua esecuzione, ed è ragionevole immaginare che esistano virus per OS.[3]

Diciamo che il programma P lanciato sul dato x, all'interno dell'ambiente generato da OS, *diffonde un virus* se altera OS. Altrimenti diciamo che P è *sicuro sul dato* x ed è *sicuro* se lo è su ogni dato.

Fissato il sistema operativo OS, esiste un programma IS-SAFE che, presi in input un qualsiasi programma P e un dato x, decide se P è sicuro su x?

Svolgimento.
Il programma IS-SAFE con le caratteristiche indicate non esiste. La dimostrazione procede per assurdo. Supporre che il programma

$$\text{IS-SAFE}(P,x) = \begin{cases} \text{SI} & \text{se } P \text{ è sicuro su } x \\ \text{NO} & \text{altrimenti} \end{cases}$$

sia sicuro porta ad un assurdo. Si consideri il programma

$$D(P) = \begin{cases} \text{CIAO} & \text{se IS-SAFE}(P,P)=\text{NO} \\ \text{altera OS} & \text{altrimenti} \end{cases}$$

e si valuti $D(D)$. Se D fosse sicuro sul dato D allora non può essere stata scelta l'opzione "altrimenti" nella definizione di D, dunque IS-SAFE(D,D)=NO, e dunque una contraddizione. Se D alterasse OS allora o IS-SAFE(D,D)=SI, e

[3] Questa definizione di virus si concentra sull'interazione fra un programma ed il sistema operativo ignorando la possibile interazione fra diversi programmi.

dunque una contraddizione, o IS-SAFE(D,D)=NO, e quindi l'unica azione compiuta da D è stata la scrittura di CIAO, operazione innocua, e dunque una contraddizione. Di conseguenza D non può essere calcolabile e sicuro.

Esercizio 7. È decidibile se $\forall i, j.\ \varphi_i = \varphi_j$?

Svolgimento. La risposta è negativa. Infatti, sia per assurdo calcolabile la seguente funzione

$$g(i,j) = \begin{cases} 1 & \text{se } \varphi_i = \varphi_j \\ 0 & \text{altrimenti} \end{cases}$$

Inoltre sia

$$\psi(i,j) = \begin{cases} 1 & \text{se } i \in K \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

La ψ è intuitivamente calcolabile (perché?) dunque per la tesi di Church-Turing esiste k per cui $\psi = \varphi_k$. Possiamo quindi applicare il teorema del parametro

$$\psi(i,j) = \varphi_k(i,j) = \varphi_{s(k,i)}(j).$$

Poiché tra le vari macchine che calcolano la ψ ne possiamo scegliere una, prendiamo la k-esima e scriviamo $f(i) = \lambda i.s(k,i)$, ottenendo

$$\psi(i,j) = \varphi_k(i,j) = \varphi_{s(k,i)}(j) = \varphi_{f(i)}(j).$$

(Questo modo di individuare l'indice $f(i)$ è piuttosto comune e in seguito non verrà descritto con questo livello di dettaglio, e scriveremo solo $\psi(i,j) = \varphi_{f(i)}(j)$.) Ora sia n_0 uno degli indici della macchina che restituisce sempre 1, cioè

$$\varphi_{n_0} = \lambda x.1$$

Definiamo adesso

$$g(f(i), n_0) = \begin{cases} 1 & \text{se } \varphi_{f(i)} = \varphi_{n_0} \Rightarrow \psi(f(i), n_0) = 1 \Rightarrow i \in K \\ 0 & \text{se } \varphi_{f(i)} \neq \varphi_{n_0} \Rightarrow \psi(f(i), n_0)\uparrow \Rightarrow i \notin K \end{cases}$$

la quale, poiché $f(x)$ è calcolabile totale per il teorema del parametro, deciderebbe il problema della fermata: assurdo.

NB. Non posso dire che $\{(i,j) \mid \varphi_i \neq \varphi_j\}$ è r.e. mandando le due macchine a coda di rondine in parallelo (p.e. se $\varphi_i = \lambda x.\ 1$ e $\varphi_j = \lambda x.\mathit{indefinita}$ mai potrò concludere che $\varphi_i \neq \varphi_j$).

Condideriamo adesso l'uguaglianza debole: $\varphi_i \sim \varphi_j$ se e solamente se $\forall x \in \mathit{dom}(\varphi_i) \cap \mathit{dom}(\varphi_j).\ \varphi_i(x) = \varphi_j(x)$ Di nuovo $\varphi_i \sim \varphi_j$ non è r.e. mentre il suo complemento è r.e. ($\sim$ non è una relazione di equivalenza: è riflessiva e simmetrica, ma non transitiva).

Esercizio 8. Si dimostri che CONST $= \{x \mid \varphi_x$ totale e costante$\}$ non è ricorsivo.

Svolgimento. Basta ridurre K a CONST. Per farlo, definiamo la funzione:

$$\psi(x,y) = \begin{cases} 1 & \text{se } \exists z > y \text{ tale che } \varphi_x(x) \downarrow \text{ in meno di z passi} \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile parziale, essendo la funzione semi-caratteristica di K.

Allora, per il teorema *s-m-n* esiste f calcolabile totale iniettiva tale che $\varphi_{f(x)}(y) = \psi(x,y)$ (anche qui usiamo il solito trucchetto di scegliere uno degli indici che calcolano la ψ per poi ignorarlo). Adesso

$$x \in K \Rightarrow \varphi_{f(x)} = \psi(x,y) = \lambda y.\ 1 \Rightarrow f(x) \in CONST$$

$$x \notin K \Rightarrow \varphi_{f(x)} = \lambda y.\ \text{indefinito} \Rightarrow f(x) \notin CONST.$$

Quindi CONST non è r.

In alternativa si verifichi che CONST è un i.i.r.f. e che non è vuoto. Di conseguenza non è r.

Inoltre, CONST non è nemmeno r.e. e per dimostrarlo riduciamo $\overline{K}$ a CONST mediante la seguente funzione

$$\varphi_{f(x)}(y) = \psi(x,y) = \begin{cases} 1 & \text{se } \varphi_x(x) \text{ non converge in y passi} \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile parziale (si noti che $\psi(x,0)$ è indefinita, perché $q_0 \in Q \not\ni h$ quindi un passo di calcolo va fatto sempre); allora, per il teorema *s-m-n* esiste f calcolabile totale iniettiva tale che $\varphi_{f(x)}(y) = \psi(x,y)$. Adesso

$$x \in \overline{K} \Rightarrow \varphi_{f(x)} = \psi(x,y) = \lambda y.\ 1 \Rightarrow f(x) \in CONST$$

$$x \notin \overline{K} \Rightarrow \exists y.\, \varphi_{f(x)}(y) \text{ è indefinito} \Rightarrow f(x) \notin CONST.$$

Quindi CONST non è r.e.

Esercizio 9. Si dimostri che
INF $= \{x \mid \text{dominio}(\varphi_x)$ è infinito$\} \leq_{rec}$ CONST $= \{x \mid \varphi_x$ totale e costante$\}$

Svolgimento. Definiamo la funzione:

$$\psi(x,y) = \begin{cases} 1 & \text{se } \exists z > y \text{ tale che } \varphi_x(z) \downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile parziale: basta usare opportunamente la tecnica della coda di colomba su φ_x. Allora, per il teorema *s-m-n* esiste f calcolabile totale iniettiva tale che $\varphi_{f(x)}(y) = \psi(x,y)$ (anche qui usiamo il solito trucchetto di scegliere uno degli indici che calcolano la ψ per poi ignorarlo). Adesso

$$x \in INF \Rightarrow \varphi_{f(x)} = \psi(x,y) = \lambda y.\ 1 \Rightarrow f(x) \in CONST$$

$$x \notin INF \Rightarrow \exists y.\, \varphi_{f(x)}(y) \text{ è indefinito} \Rightarrow f(x) \notin CONST.$$

Esercizio 10. Si dimostri che
$\text{TOT} = \{x \mid \text{dominio}(\varphi_x) = I\!N\} \leq_{rec} \text{INF} = \{x \mid \text{dominio}(\varphi_x) \text{ è infinito}\}$

Svolgimento. Definiamo la seguente funzione, dove abbiamo applicato il teorema *s-m-n*, dato che la ψ è calcolabile parziale (basta usare opportunamente la tecnica della coda di colomba su φ_x su un numero *finito* di argomenti):

$$\varphi_{f(x)}(y) = \psi(x,y) = \begin{cases} 1 & \text{se } \forall z < y.\ \varphi_x(z)\downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Adesso

$$x \in TOT \Rightarrow \varphi_{f(x)} = \psi(x,y) = \lambda y.\ 1 \Rightarrow f(x) \in INF$$
$$x \notin TOT \Rightarrow \exists \overline{y}.\forall y > \overline{y}.\ \varphi_{f(x)}(y) \text{ è indefinita} \Rightarrow f(x) \notin INF.$$

Si noti che la funzione identità non riduce TOT a INF: si consideri la funzione *mezzo* che non è totale ma ha dominio infinito.

Esercizio 11. Si dimostri che

1. $\text{FIN} = \{x \mid dom(\varphi_x) \text{ è finito}\}$, cioè l'insieme degli indici delle funzioni calcolabili con dominio finito, non è ricorsivamente enumerabile.
2. $\text{INF} = \{x \mid dom(\varphi_x) \text{ è infinito}\} = \overline{\text{FIN}}$ non è ricorsivamente enumerabile.

Svolgimento. Per (1) dimostriamo che $\overline{K} \leq_{rec} FIN$, da cui la tesi. Definiamo

$$\psi(x,y) = \begin{cases} 1 & \text{se } x \in K \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile parziale, quindi per il teorema *s-m-n* esiste f calcolabile totale iniettiva, definita con le solite avvertenze, tale che $\varphi_{f(x)}(y) = \psi(x,y)$. Si ha che

$$x \in \overline{K} \text{ sse } x \notin K \Rightarrow \varphi_{f(x)} = \lambda y.\ \text{indefinita} \Rightarrow dom(\varphi_{f(x)}) = \emptyset \Rightarrow f(x) \in FIN$$
$$x \notin \overline{K} \text{ sse } x \in K \Rightarrow \varphi_{f(x)} = \lambda y.1 \Rightarrow dom(\varphi_{f(x)}) = I\!N \Rightarrow f(x) \notin FIN.$$

Per (2) si consideri la funzione (ottenuta come la $\varphi_{f(x)}$ di sopra da un'opportuna ψ')

$$\varphi_{g(x)}(y) = \begin{cases} 1 & \text{se } \varphi_x(x) \text{ non converge entro } y \text{ passi} \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Si ha che

$$x \in \overline{K} \Rightarrow \varphi_{g(x)}(x) \text{ non converge entro } y \text{ passi } \Rightarrow$$
$$\varphi_{g(x)} = \lambda y.1 \Rightarrow dom(\varphi_{g(x)}) \text{ è infinito} \Rightarrow g(x) \in INF$$

$$x \notin \overline{K} \text{ sse } x \in K \Rightarrow \exists z.\ \varphi_{g(x)}(y) = \begin{cases} 1 & \text{se } y \in [0..z] \\ \text{indefinita} & \text{altrimenti} \end{cases}$$
$$\Rightarrow dom(\varphi_{f(x)}) \text{ è finito} \Rightarrow f(x) \in FIN.$$

Nota: è immediato vedere che $\emptyset \neq_{\text{FIN}} I\!N$, da cui, per il lemma usato per dimostrare il teorema di Rice, segue che non è r.

Esercizio 12. L'insieme $I = \{i \mid dom(\varphi_i) = \{3\}\}$ è ricorsivo? ricorsivamente enumerabile? nemmeno ricorsivamente enumerabile? e se $dom(\varphi_i)$ fosse l'insieme dei numeri pari?

Svolgimento Banalmente $\emptyset \neq I \neq \mathbb{N}$ e inoltre I è un insieme di indici che rispettano le funzioni, pertanto I non è r.

Inoltre $K \leq_{rec} I$. Infatti, si consideri

$$\varphi_{f(x)}(y) = \psi(x,y) = \begin{cases} 1 & \text{se } x \in K \ \wedge \ y = 3 \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile parziale. Adesso

$$x \in K \Rightarrow \varphi_{f(x)}(y) = \begin{cases} 1 & \text{se } y = 3 \\ \text{indefinita} & \text{se } y \neq 3 \end{cases} \Rightarrow dom(\varphi_{f(x)}) = \{3\} \Rightarrow f(x) \in I$$

$x \notin K \Rightarrow \varphi_{f(x)} = \lambda y.$ indefinito $\Rightarrow f(x) \notin I$.

Infine, possiamo dimostrare che I non è nemmeno r.e., perché $\overline{K} \leq_{rec} I$; infatti $K \leq_{rec} \overline{I}$, attraverso la seguente funzione di riduzione che modifica leggermente quella usata sopra:

$$\varphi_{g(x)}(y) = \psi(x,y) = \begin{cases} 1 & \text{se } x \in K \ \vee \ y = 3 \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Abbiamo che

$x \in K \Rightarrow \varphi_{g(x)}(y) = 1 \Rightarrow dom(\varphi_{g(x)}) = \mathbb{N} \Rightarrow g(x) \notin I$

$x \notin K \Rightarrow \varphi_{g(x)}(y) = \begin{cases} 1 & \text{se } y = 3 \\ \text{indefinita} & \text{altrimenti} \end{cases} \Rightarrow dom(\varphi_{g(x)}) = \{3\}$
$\Rightarrow g(x) \in I$

Esercizio 13. Esistono due insiemi A e B non ricorsivi la cui intersezione è un insieme ricorsivo infinito?

Svolgimento Sí: si consideri l'insieme $I = \{i \mid \varphi_i(i) \downarrow \text{ in 3 passi}\}$ che è ricorsivo infinito ed è l'intersezione di $A = K$ e $B = \overline{K} \cup I$.

Esercizio 14. La classe degli insiemi di indici che rappresentano le funzioni forma un'algebra booleana? Si giustifichi la risposta.

Svolgimento Basta far vedere, ed è banale, che

- $\emptyset$ è un i.i.r.f. (nessuna)
- se I è un i.i.r.f. lo è anche il suo complemento
- se I e J sono due i.i.r.f., lo è anche $I \cap J$

Esercizio 15. Si dica se esiste una funzione calcolabile totale f tale che

$$\forall x.\ dom(\varphi_{f(x)}) = I\!N$$

Svolgimento Sia i uno degli indici della funzione $\lambda\, x.\, 0$ e si ponga $f = \lambda\, x.\, i$. Chiaramente $dom(\varphi_{f(x)}) = dom(\varphi_i) = I\!N$.

Esercizio 16. Si dimostri che per ogni enumerazione effettiva esiste $i \in K$ tale che

$$\varphi_i(x) = \begin{cases} 0 & \text{se } x = i \\ \text{indef} & \text{altrimenti} \end{cases}$$

Svolgimento Si definisca la seguente funzione $\psi(x, y)$ che è intuitivamente calcolabile, quindi possiamo applicare il teorema *s-n-m* al suo indice e a x, e poi ottenere con il solito metodo, la funzione calcolabile totale f che ne calcola l'indice:

$$\varphi_{f(x)} = \psi(x, y) = \begin{cases} 0 & \text{se } x = y \\ \text{indef} & \text{altrimenti} \end{cases}$$

Per il teorema di ricorsione, $\exists n$ tale che $\varphi_n = \varphi_{f(n)}$ e quindi $\forall y$ si ha che

$$\varphi_n(y) = \varphi_{f(n)}(y) = \psi(n, y) = \begin{cases} 0 & \text{se } y = n \\ \text{indef} & \text{altrimenti} \end{cases}$$

Quindi esiste una funzione calcolabile che converge solamente sul suo indice, indipendentemente dall'enumerazione scelta.

Esercizio 17. Si dimostri che K non è un i.i.r.f.

Svolgimento Dall'esercizio precedente sappiamo che esiste i tale che

$$\varphi_i(x) = \begin{cases} 0 & \text{se } x = i \\ \text{indef} & \text{altrimenti} \end{cases}$$

e dal padding lemma che esiste $j \neq i$ tale che $\varphi_j = \varphi_i$. Si ha che $i \in K$, ma $j \notin K$ perché, essendo $j \neq i$, $\varphi_j(j)\uparrow$.

Esercizio 18. Sia A_i l'insieme di indici costruito nel padding lemma e si dimostri che $A_i \leq_{rec} K$

Svolgimento. L'insieme A_i è generato da una funzione ricorsiva primitiva, per cui è ricorsivamente enumerabile. La tesi segue dalla completezza di K.

Esercizio 19. Dato i, si dica se l'insieme $K@i = \{j \mid j \in K \wedge j \leq i\}$ è r., r.e. o non r.e. e si giustifichi la risposta.

Svolgimento. Poiché $K \downarrow i$ è finito è ricorsivo.

Esercizio 20. Si dimostri che $A = \{x \mid \forall y.\, \varphi_x(y) \uparrow\}$ non è r.e.

Svolgimento. Per farlo basta dimostrare che $\overline{K} \leq A$ ovvero che $K \leq \overline{A} = K_1$. Si definisca allora la seguente funzione $\psi(x, y)$ che è intuitivamente calcolabile, quindi possiamo applicare il teorema del parametro al suo indice e a x, e poi ottenere con il solito metodo, la seguente funzione calcolabile totale f:

$$\varphi_{f(x)} = \psi(x, y) = \begin{cases} 1 & \text{se } x \in K \wedge \exists z.\varphi_x(z) \downarrow \\ \text{indef} & \text{altrimenti} \end{cases}$$

Adesso abbiamo che

$$x \in K \implies \varphi_{f(x)} = \lambda y.1 \implies f(x) \in \overline{A} \quad \text{(basta prendere } z = x\text{)}$$

$$x \notin K \implies \varphi_{f(x)} = \lambda y.\text{indef} \implies f(x) \in A$$

(Nota: la condizione $\exists z.\varphi_x(z) \downarrow$ in effetti è inutile.)

Esercizio 21. Un insieme di indici che rappresenta una singola funzione può essere sempre costruito usando il padding lemma?

Svolgimento. No, perché ogni insieme di indici A_x costruito come nel padding lemma a partire dall'indice x è r.e., in quanto immagine di una funzione primitiva ricorsiva (e quindi calcolabile totale) mentre ci sono insiemi di indici che rappresentano le funzioni, p.e. quello che rappresenta la funzione ovunque indefinita, che non sono r.e. (v. esercizio 20).

Esercizio 22. Sia REC $= \{x \mid dom(\varphi_x)$ è ricorsivo$\}$ e si dimostri che

$$K \leq_{rec} REC$$

Inoltre si potrebbe anche far vedere che REC non è nemmeno r.e.

Svolgimento Per dimostrare che $K \leq_{rec} REC$, definiamo la seguente funzione ψ che è calcolabile, per cui ha un indice e da cui ottengo con il teorema del parametro la seguente $\varphi_{f(x)}$:

$$\varphi_{f(x)}(y) = \psi(x, y) = \begin{cases} 1 & \text{se } x \in K \vee y \in K \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Si ha che

$$x \in K \Rightarrow \varphi_{f(x)} = \lambda y.1 \Rightarrow dom(\varphi_{f(x)}) = I\!N \Rightarrow f(x) \in \mathit{REC}$$

$x \notin K \Rightarrow \varphi_{f(x)} = \lambda y.1$ sse $y \in K \Rightarrow \varphi_{f(x)}$ è la semicaratteristica di $K \Rightarrow f(x) \notin \mathit{REC}$ visto che la funzione semicaratteristica di K non è totale.

Per vedere che $\overline{K} \leq_{rec} \mathit{REC}$, procediamo in modo analogo:

$$\varphi_{f(x)} = \psi(x,y) = \begin{cases} 1 & \text{se } x \in K \wedge y \in K \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Adesso si ha che

$$x \in \overline{K} \text{ sse } x \notin K \Rightarrow \varphi_{f(x)} = \lambda y. \text{ indefinita} \Rightarrow dom(\varphi_{f(x)}) = \emptyset \Rightarrow f(x) \in \mathit{REC}$$

$$\begin{aligned} x \notin \overline{K} \text{ sse } x \in K &\Rightarrow \varphi_{f(x)} = \lambda y.1 \text{ sse } y \in K \\ &\Rightarrow \varphi_{f(x)} \text{ è la semicaratteristica di } K \Rightarrow f(x) \notin \mathit{REC}. \end{aligned}$$

Esercizio 23. Sia A un i.i.r.f. e si dimostri che

$$A \not\leq_{rec} \overline{A}$$

Svolgimento Sia per assurdo f la riduzione cercata, allora per il teorema di ricorsione esiste n tale che $\varphi_n = \varphi_{f(n)}$. Ora, se $n \in A$ si ha che $f(n) \in \overline{A}$ e quindi A non sarebbe un i.i.r.f. contro l'ipotesi.

Esercizio 24. Si dimostri che
FIN $= \{x \mid \text{dominio}(\varphi_x) \text{ è finito}\} \leq_{rec}$ REC $= \{x \mid \text{dominio}(\varphi_x) \text{ è ricorsivo}\}$

Svolgimento. Sia i_0 tale che $dom(\varphi_{i_0}) = K$ e definiamo la funzione:

$$\psi(x,y) = \begin{cases} \varphi_{i_0}(y) & \text{se } \exists z > y \text{ tale che } \varphi_x(z)\downarrow \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

che è calcolabile e allora, per il teorema *s-m-n* esiste f calcolabile totale iniettiva tale che $\varphi_{f(x)}(y) = \psi(x,y)$ (anche qui usiamo il solito trucchetto di scegliere uno degli indici che calcolano la ψ per poi ignorarlo). Adesso

$$x \in FIN \Rightarrow \exists \bar{y} \text{ t.c. } \varphi_{f(x)}(y) = \begin{cases} \varphi_{i_0}(y) & \text{se } y < \bar{y} \\ \text{indefinita} & \text{altrimenti} \end{cases}$$
$$\Rightarrow \text{dominio}(\varphi_{f(x)}) = K \cap \{y < \bar{y} \mid \varphi_x(y)\downarrow\} \text{ finito } \Rightarrow f(x) \in REC$$

$$x \notin FIN \Rightarrow \forall y\, \exists z > y \text{ t.c. } \varphi_x(z)\downarrow \Rightarrow \varphi_{f(x)} = \varphi_{i_0} \Rightarrow f(x) \notin REC$$

Esercizio 25. Gli insiemi di indici costruiti grazie al padding lemma rappresentano le funzioni?

Svolgimento. No e il modo usato per costruirne uno nella dimostrazione del padding lemma fornisce un controesempio. Infatti si prenda la macchina x-ma e si aggiunga al suo alfabeto un simbolo σ che non vi compare e per eccesso di zelo la quintupla $(q_0, \sigma, q_0, \sigma, -)$. Si ottiene una macchina che calcola proprio ϕ_x ma il cui numero non appartiene ad A_x.

Esercizio 26. Si dimostri la non ricorsiva enumerabilità di

$$\text{EXT } = \{x \mid \exists y \text{ tale che } \varphi_x(z) = n \Rightarrow \varphi_y(z) = n \ \wedge \ dom(\varphi_y) = I\!N\}$$

Svolgimento. La seguente funzione è calcolabile e sia i_1 l'indice di uno degli algoritmi che calcolano, mentre sia i_0 uno di quelli della funzione ovunque indefinita.

$$\varphi_{i_1}(x) = \begin{cases} n & \text{se } M_x(x)\downarrow \text{ in } n \text{ passi} \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

Si ha che $i_1 \notin$ EXT, perché altrimenti K sarebbe ricorsivo, mentre φ_{i_0} è estendibile, p.e. dalla funzione costante 0.

Per dimostrare che $\overline{K} \leq_{rec}$ EXT definiamo la seguente funzione calcolabile

$$\varphi_{f(x)}(y) = \varphi_i(x,y) = \psi(x,y) = \begin{cases} \varphi_{i_1}(y) & \text{se } x \in K \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

dove abbiamo usato la tesi di Church-Turing e il teorema del parametro. Adesso

$$x \in \overline{K} \Rightarrow \varphi_{f(x)} = \varphi_{i_0} \Rightarrow f(x) \in \text{EXT}$$
$$x \notin \overline{K} \Rightarrow \varphi_{f(x)} = \varphi_{i_1} \Rightarrow f(x) \notin \text{EXT}$$

Esercizi di complessità

Esercizio 1. [Tseitin 1966] Si dimostri che un'espressione booleana B può essere trasformata in una espressione B' in forma congiuntiva che è soddisfacibile se e solo se B lo è, usando spazio logaritmico. (Si veda anche la dimostrazione che CIRCUIT VALUE $\leqslant_{logspace}$ CIRCUIT SAT.)

Svolgimento Si noti innanzitutto che B e B' *non* devono essere necessariamente equivalenti, ma *solo* entrambe soddisfacibili o non soddifacibili.

Ad ogni sottoespressione B_i di B, che non sia un letterale, associamo una variabile x_i fresca, cioè che non appare in B. Per comodità, data una sottoespressione B_i indichiamo con x_i la variabile ad essa associata se B_i non è un letterale; inoltre scriviamo l_j sia per la variabile fresca x_j che per il letterale l_j che appare in B.

Per ogni sottoespressione B_i non letterale e B_j, B_h sottoespressioni qualunque (anche letterali) si generino le seguenti clausole:

se $B_i = \neg B_j$ **si generano** $(x_i \vee \neg l_j)$ e $(\neg l_j \vee x_i)$ — ovvero $B_i \Leftrightarrow \neg B_j$;

se $B_i = B_j \vee B_h$ **si generano** $(\neg l_j \vee x_i)$, $(\neg l_h \vee x_i)$ e $(\neg x_i \vee l_j \vee l_h)$ — ovvero $B_i \Leftrightarrow B_j \vee B_h$

se $B_i = B_j \wedge B_h$ **si generano** $(\neg x_i \vee l_j)$, $(\neg x_i \vee l_h)$ e $(x_i \neg l_j \vee \neg l_h)$ — ovvero $B_i \Leftrightarrow B_j \wedge B_h$

L'espressione booleana B' è la congiunzione delle clausole così generate e del letterale corrispondente alla variabile fresca, sia x, associata a B, sottoespressione non propria di sé stessa. Si noti che la dimensione di B' è molto piú elevata di quella di B, ma pur sempre lineare rispetto ad essa, e non esponenziale come sarebbe la forma normale congiuntiva di B — la procedura guidava la costruzione di circuiti "compatti" ed equi-soddisfacibili e adesso viene usata nei cosí detti SAT-solver.

La procedura descritta richiede spazio logaritmico: è sufficiente tenere nel nasto di lavoro un contatore in binario alla prossima sottoespressione ancora da valutare.

Definizione: k-SAT Stabilire se esiste un assegnamento che soddisfa un'espressione booleana in forma congiuntiva, dove ogni congiunto contiene *esattamente* k letterali, ovvero se è nella forma

$$B = \bigwedge_{i=1}^{n} C_i \quad \text{dove} \quad C_i = (\bigvee_{j=1}^{k} l_{i,j})$$

Esercizio 2.

a) Dimostrare che SAT si riduce a 3-SAT

b) 3-SAT è $\mathcal{NP}$-completo o solo $\mathcal{NP}$-arduo?

Svolgimento. Per il punto (**a**) prendiamo una generica espressione B in forma congiuntiva e la traduciamo in una che rispetti i vincoli sintattici di 3-SAT e che sia *equisoddisfacibile*, ovvero che sia soddisfacibile se e solo se B lo è — si noti che non è necessario che le due formule siano *equivalenti* (v. esercizio 1)

Si opera su ciascun congiunto $C_i = z_1 \vee z_2 \cdots \vee z_n$.
Se $n < 3$ basta aggiungere a C_i il numero opportuno di disgiunti ff, mentre se $n = 3$ non c'è niente da fare.
Se $n \geq 4$, si spezza il congiunto in piú congiunti e si usano variabili fresche $x_1, x_2, \ldots, x_n$ come segue

$$\begin{aligned} C_i^1 &= (\mathit{ff} \vee z_1 \vee x_1) \\ C_i^j &= (\neg x_{j-1} \vee z_j \vee x_j) \qquad 2 \leq j \leq n-1 \\ C_i^n &= (\neg x_{n-1} \vee z_n \vee \mathit{ff}) \end{aligned}$$

La nuova formula è la congiunzione delle clausole cosí ottenute ed è equisoddisfacibile rispetto alla precedente. Infatti, se esiste un assegnamento che verifica $z_1 \vee z_2 \cdots \vee z_n$ e consideriamo prima i congiunti originati quando $n \geq 4$. Prendiamo allora lo stesso assegnamento per le z_i e lo estendiamo alle x_i in modo che ciascuna C_i^j sia soddisfatta il che è sempre possibile: se $C_i^j = (\neg x_{j-1} \vee z_j \vee x_j)$ e z_j vale ff basta assegnare tt a x_j. Se invece B non è soddisfacibile, per ogni assegnamento deve esistere un congiunto $C_i = z_1 \vee z_2 \cdots \vee z_n$ i cui i letterali z_i ricevono tutti valore ff, e in questo caso la congiunzione dei C_i^j si riduce a $x_1 \wedge (\neg x_1 \vee x_2) \cdots \wedge (\neg x_{n-2} \vee x_{n-1}) \wedge (\neg x_{n-1} \vee \mathit{ff})$ che è chiaramente insoddisfacibile. Nel caso dei congiunti che non siano nella forma C_i^j non c'è bisogno di far niente.

La riduzione è chiaramente logaritmica in spazio, in quanto basta tradurre congiunto per congiunto usando un nastro di lavoro per contare in binario le occorrenze e decrementarlo via via che si generano le nuove clausole e variabili fresche (possiamo ad esempio prendere tre nastri che contengano ognuno un contatore fino a n, rappresentabile in binario in spazio $log(n)$).

Per il punto **(b)** possiamo notare come la riduzione ci garantisce che 3-SAT sia almeno difficile quanto SAT, quindi è difficile almeno quanto i problemi in $\mathcal{NP}$ ($\mathcal{NP}$-arduo). Per stabilire che sia $\mathcal{NP}$-completo dobbiamo dimostrare che 3-SAT sia in $\mathcal{NP}$: ovvio perché 3-SAT $\leq_{id}$ SAT, dove id è la funzione identità.

Definizione: Isomorfismo fra Grafi. Due grafi $G = (N, A)$ e $G' = (N', A')$ sono *isomorfi* se e solo se esiste una funzione biunivoca $f : N \to N'$ che conserva gli archi, cioè tale che $(n, m) \in A \Leftrightarrow (f(n), f(m)) \in A'$.

Esercizio 3. Si dimostri che il problema di stabilire se due grafi sono isomorfi appartiene a $\mathcal{NP}$.

Svolgimento Basta dare una MdT non deterministica che risolve il problema in tempo polinomiale. Tale MdT non deterministica esegue i seguenti passi:

- controlla se $\#N = \#N'$ (senza contare i nodi isolati) — in tempo lineare
- genera a caso un isomorfismo f — in tempo lineare
- verifica che $\forall(n, m) \in A$ sia $(f(n), f(m)) \in A'$ e che $\forall(n, m) \notin A$ sia $(f(n), f(m)) \notin A'$ — in tempo quadratico.

C'è un algoritmo dovuto a Lazlo Babai, che risolve il problema in tempo quasipolinomiale $n^{(logn)^k}$, un po' peggiore del tempo polinomiale, ma assai migliore di quello esponenziale che al momento ci aspettiamo per i problemi in $\mathcal{NP}$. Si tratta di un notevole passo in avanti nella comprensione della frontiera tra $\mathcal{P}$ e $\mathcal{NP}$ (per chi lo capisce :-). Una prima versione conteneva un errore, successivamente corretto da Babai stesso nel 2017, la cui complicatissima dimostrazione dispiega una poderosa conoscenza di come combinare tecniche e risultati di combinatorica e teoria dei gruppi.

Esercizio 4. Si dimostri che 2-SAT è in $\mathcal{P}$.

Svolgimento Basta dare un algoritmo che decide la soddisfacibilità in tempo polinomiale.

Data l'espressione booleana B, si costruisce il grafo diretto $G = (N, A)$ dove i nodi $\alpha, \neg\alpha, \beta, \cdots \in N$ sono in corrispondenza biunivoca con le variabili in B e con la loro negazione, e gli archi $(\neg\alpha, \beta)$ e $(\neg\beta, \alpha)$ appartengono ad A se e solo se B contiene la clausola $\alpha \vee \beta$. Intuitivamente gli archi rappresentano la disgiunzione vista come un'implicazione, secondo le note equivalenze $(\alpha \vee \beta) \equiv (\neg\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \alpha)$.

La seguente osservazione è cruciale: la formula

$$(l_1 \Rightarrow l_2) \wedge (l_2 \Rightarrow l_3) \cdots \wedge (l_n \Rightarrow \neg l_1)$$

può essere soddisfatta solo assegnando l_1 a *ff*, mentre

$$(\neg l_1 \Rightarrow l_2) \wedge (l_2 \Rightarrow l_3) \cdots \wedge (l_n \Rightarrow l_1)$$

può essere soddisfatta solo assegnando l_1 a *tt*.

Poiché nessun assegnamento può soddisfare entrambe le formule, l'algoritmo che decide 2-SAT controlla se in G esiste un nodo (corrispondente a) x tale che c'è un ciclo da x a sé stesso che attraversa il nodo $\neg x$. Se tale ciclo esiste allora si restituisce NO perché B non è soddisfacibile, altrimenti si restituisce SI.

A questo punto occorre dimostrare la correttezza dell'algoritmo proposto. I seguenti due risultati ausiliari hanno una dimostrazione immediata: $(\alpha, \beta) \in A$

se e solo se $(\neg\beta, \neg\alpha) \in A$. Inoltre, se esiste un cammino da α a β in G (in formule $\alpha \rightsquigarrow \beta$), allora in G c'è anche un cammino $\neg\beta \rightsquigarrow \neg\alpha$.

Supponiamo ora che un ciclo $x \rightsquigarrow \neg x \rightsquigarrow x$ esista e dimostriamo che B non è soddisfacibile. Dato che esiste il ciclo, esistono in B delle clausole della seguente forma, in cui per chiarezza abbiamo scritto $a \Rightarrow b$ invece che $\neg a \vee b$

$$(x \Rightarrow y_1) \wedge (y_1 \Rightarrow y_2) \cdots \wedge (y_n \Rightarrow \neg x) \qquad (\neg x \Rightarrow z_1) \wedge (z_1 \Rightarrow z_2) \cdots \wedge (z_n \Rightarrow x)$$

Il congiunto di sinistra è soddifatto solo se x è assegnato a *ff*, quello di destra solo se è assegnato a *tt*, quindi l'espressione non è soddisfacibile.

Se un tale ciclo non esiste, allora costruiamo un'assegnamento che soddisfa B iterando i seguenti passi:

1. si prende un nodo α per cui non c'è ancora un assegnamento e tale che $\alpha \not\rightsquigarrow \neg\alpha$ e gli si assegna *tt*;
2. si pongono a *tt* tutti i nodi raggiungibili a partire da α;
3. si pongono a *ff* tutti i nodi contenenti la negazione delle variabili poste a *tt* nelle fasi 1. e 2.

Dimostriamo che l'assegnamento soddisfa B. Il passo 1. garantisce che ogni variabile x riceva un valore booleano. L'unico caso in cui questo potrebbe non accadere è quando da x (o da $\neg x$) parta un cammino che porta alla loro negazione, ma in questo caso avremmo un ciclo. Dimostriamo adesso che questo valore è unico. Se da α si raggiungono sia (il nodo corrispondente a) una variabile x ($\alpha \rightsquigarrow x$) che (quello al)la sua negazione $\neg x$ ($\alpha \rightsquigarrow \neg x$), per la proprietà ausiliaria esisterebbe il cammino $\alpha \rightsquigarrow x \rightsquigarrow \neg\alpha$ e quindi α non sarebbe stata scelta al passo 1. Se invece a una variabile x raggiunta da α fosse già stato assegnato il valore *ff* in un'iterazione precedente, allora esisterebbe un α' tale che $\alpha' \rightsquigarrow \neg x$. Ma per la proprietà ausiliaria abbiamo sia $\alpha' \rightsquigarrow \neg x$ che $\neg x \rightsquigarrow \neg\alpha$ e α non sarebbe stata scelta al passo 1.

Il costo dell'algoritmo delineato sopra è polinomiale: per ciascuno degli n nodi occorre visitare al massimo ogni arco e questi sono n^2.

Esercizio 5. Si considerino le espressioni booleane in forma *disgiuntiva* e si calcoli la complessità di verificarne la soddisfacibilità.

Si discuta inoltre l'impatto che ha la soluzione proposta sul problema SAT.

Svolgimento La complessità è lineare: basta scorrere l'espressione e verificare che esiste una clausola che non contiene né *ff* né x e $\neg x$. Infatti, una formula disgiuntiva è soddisfacibile se e solo se almeno una sua clausola è soddisfacibile, ed è sempre possibile soddisfare una clausola tranne nei casi di sopra.

Il risultato non inficia l'appartenenza di SAT a $\mathcal{NP}$, né la sua completezza: data una espressione booleana con n simboli, la sua forma normale disgiuntiva ne ha $\mathcal{O}(2^n)$.

Esercizio 6. Si dimostri che 3_3-SAT è $\mathcal{NP}$-completo, dove 3_3-SAT consiste nel problema di decidere la soddisfacibilità di espressioni booleane B in forma congiuntiva dove ogni congiunto contiene *al massimo* tre letterali e dove ogni letterale non compare piú di tre volte in B.

Si usino le riduzioni polinomiali in tempo (anziché quelle logaritmiche in spazio come consueto nel corso).

E se ogni congiunto deve contenere *esattamente* tre letterali?

Svolgimento 3_3-SAT è chiaramente in $\mathcal{NP}$ dato che 3_3-SAT si riduce a SAT con l'identità. Basta quindi mostrare che è arduo per $\mathcal{NP}$, e lo facciamo riducendo 3-SAT (che è completo, v. esercizio 2) a 3_3-SAT.

Sia B un'espressione booleana in forma congiuntiva in cui ogni clausola contiene esattamente tre letterali. Procediamo a rimuovere le occorrenze in eccesso come segue. Data una qualunque x che compare $n > 3$ volte in B:

1. sostituiamo tutte le occorrenze di x con n variabili fresche $x_1, x_2, \ldots x_n$;
2. alla formula cosí ottenuta aggiungiamo i seguenti i seguenti n disgiunti:

$$\bigwedge_{1 \leqslant i \leqslant k-1} (x_i \vee \neg x_{i+1}) \wedge (x_k \vee \neg x_1)$$

La congiunzione delle k disgiunzioni nel punto 2 ha una natura ciclica che obbliga tutte le x_i ad avere lo stesso valore, che sarà proprio quello assegnato a x nell'espressione originale B; la correttezza della trasformazione è quindi immediata. Inoltre ciascuna nuova variabile x_i appare solo due volte.

La complessità della traduzione è lineare: per ogni variabile basta scorrere l'espressione, ricavare le variabili e il numero di occorrenze che eccedono 3 e poi generare un numero lineare di clausole.

Se i congiunti dovessere contenere *esattamente* tre letterali, basterebbe aggiungere a ogni disgiunzione il letterale ff ove necessario.

Esercizio 7. Sia co-$\mathcal{X} = \{I \mid \overline{I} \in \mathcal{X}\}$. Si discutano le seguenti eguaglianze e le loro possibili conseguenze sul problema $\mathcal{P} \subseteq \mathcal{NP}$

$$\mathcal{P} = \text{co-}\mathcal{P} \qquad\qquad \mathcal{NP} = \text{co-}\mathcal{NP}$$

Svolgimento La prima eguaglianza è facilmente verificata: si scambino gli stati di accettazione e di rifiuto della MdT che decide I.

Il valore della seconda è ignoto al momento; si noti solamente che se fosse $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ allora sarebbe $\mathcal{P} \neq \mathcal{NP}$. Assumendo $\mathcal{P} = \mathcal{NP}$, possiamo infatti derivare co-$\mathcal{P}$ = co-$\mathcal{NP}$ per definizione di co-$\mathcal{X}$.

Se l'eguaglianza valesse, nulla si potrebbe dire sull'inclusione stretta o meno delle due classi.