Dispense per il corso di informatica teorica

Sebastiano Vigna

4 febbraio 2009

Indice

1 Notazione e definizioni di base

Il prodotto cartesiano degli insiemi X e Y è l'insieme $X \times Y = \{\langle x, y\rangle \mid x \in X \wedge y \in Y\}$ delle coppie ordinate degli elementi di X e Y. La definizione si estende per ricorsione a n insiemi. Al prodotto cartesiano $X_1 \times X_2 \times \cdots \times X_n$ sono naturalmente[1] associate le *proiezioni* $\pi_1, \pi_2, \ldots, \pi_n$ definite da

$$\pi_i(\langle x_1, x_2, \ldots, x_n\rangle) = x_i.$$

Poniamo

$$X^n = \overbrace{X \times X \times \cdots \times X}^{n \text{ volte}}$$

e $X^0 = \{*\}$ (qualunque insieme con un solo elemento). La *somma disgiunta* degli insiemi X e Y è, intuitivamente, un'unione di X e Y che però tiene separati gli elementi comuni ("evita i conflitti"). Formalmente,

$$X + Y = X \times \{0\} \cup Y \times \{1\}.$$

Di solito ometteremo, con un piccolo abuso di notazione, la seconda coordinata (0 o 1).

Una *relazione* tra gli insiemi $X_1, X_2, \ldots, X_n$ è un sottoinsieme R del prodotto cartesiano $X_1 \times X_2 \times \cdots \times X_n$. Se $n = 2$ è uso scrivere $x\ R\ y$ per $\langle x, y\rangle \in R$. Una relazione tra due insiemi è detta *binaria*. Se R è una relazione binaria tra X e Y, X è detto il *dominio* di R, ed è denotato da $\operatorname{dom} R$, mentre Y è detto il *codominio* di R, ed è denotato da $\operatorname{cod} R$. Il *rango* o *insieme di definizione* di R è l'insieme $\operatorname{ran} R = \{x \in X \mid \exists y \in Y\ x\ R\ y\}$, e in generale può non coincidere con il dominio di R. L'*immagine* di R è l'insieme $\operatorname{imm} R = \{y \in Y \mid \exists x \in X\ x\ R\ y\}$, e in generale può non coincidere con il codominio di R. Una relazione binaria R tra X e Y è *monodroma* se per ogni $x \in X$ esiste al più un $y \in Y$ tale che $x\ R\ y$. È *totale* se per ogni $x \in X$ esiste un $y \in Y$ tale che $x\ R\ y$, cioè se $\operatorname{ran} R = \operatorname{dom} R$. È *iniettiva* se per ogni $y \in Y$ esiste al più un $x \in X$ tale che $x\ R\ y$. È *suriettiva* se per ogni $y \in Y$ esiste un $x \in X$ tale che $x\ R\ y$, cioè se $\operatorname{imm} R = \operatorname{cod} R$. È *biiettiva* se è sia iniettiva che suriettiva.

Una *funzione* da X a Y è una relazione monodroma e totale tra X e Y (notate che l'ordine è rilevante); in tal caso scriviamo $f : X \to Y$ per dire che f "va da X a Y". Se f è una funzione da X a Y è uso scrivere $f(x)$ per l'unico $y \in Y$ tale che $x\ f\ y$. Diremo che f *mappa* x in $f(x)$, o, in simboli, $x \mapsto f(x)$. Le nozioni di dominio, codominio, iniettività, suriettività e biiettività vengono ereditate dalle relazioni. Se una funzione $f : X \to Y$ è biiettiva, è facile verificare che esiste una funzione *inversa* f^{-1}, che soddisfa le equazioni $f(f^{-1}(y)) = y$ e $f^{-1}(f(x)) = x$ per ogni $x \in X$ e $y \in Y$. Una *funzione parziale*[2] da X a Y è una relazione monodroma tra X e Y; una funzione parziale può non essere definita su elementi del suo dominio, fatto che

[1] L'uso dell'aggettivo "naturale", qui e altrove, è tecnico, e relativo alla teoria delle categorie. *Dixi, et salvavi animam meam*.

[2] È una sfortunata evenienza che una funzione parziale *non* sia una funzione. Ragioni storiche hanno consolidato ormai da tempo questa disgraziata nomenclatura.

denotiamo con la scrittura $f(x) = \perp$ ("$f(x)$ è indefinito" o "f è indefinita su x"), che significa $x \notin \operatorname{ran} f$. Date funzioni (parziali) $f : X \to Y$ e $g : Y \to Z$, la *composizione* $g \circ f$ di f con g è la funzione definita da $(g \circ f)(x) = g(f(x))$. Si noti che, per convenzione, $f(\perp) = \perp$ per ogni funzione (parziale) f. Dati gli insiemi X e Y, denotiamo con $Y^X = \{f \mid f : X \to Y\}$ l'insieme delle funzioni da X a Y. Si noti che per insiemi finiti[3] $\left|Y^X\right| = |Y|^{|X|}$.

Denoteremo con $[n]$, o anche semplicemente[4] con n, l'insieme $\{0, 1, 2, \ldots, n-1\}$.

Dato un sottoinsieme A di X, possiamo associargli la sua *funzione caratteristica* $\chi_A : X \to 2$, definita da

$$\chi_A(x) = \begin{cases} 0 & \text{se } x \notin A \\ 1 & \text{se } x \in A. \end{cases}$$

Per contro, a ogni funzione $f : X \to 2$ possiamo associare il sottoinsieme di X dato dagli elementi mappati da f in 1, cioè l'insieme $\{x \in X \mid f(x) = 1\}$; tale corrispondenza è inversa alla precedente, ed è quindi naturalmente equivalente considerare sottoinsiemi di X o funzioni da X in 2.

Un *alfabeto* è un insieme finito e non vuoto Σ di *simboli*. L'insieme Σ^* delle *stringhe* (cioè sequenze finite di simboli) su Σ è definito da[5]

$$\Sigma^* = \bigcup_{n \in \mathbf{N}} \Sigma^n.$$

Se $w \in \Sigma^*$ scriveremo $|w|$ per la lunghezza di w. La sola stringa di lunghezza zero è detta *stringa vuota* e indicata con ε. Un *linguaggio* su Σ è un insieme $L \subseteq \Sigma^*$. Spesso utilizzeremo l'alfabeto *binario* 2.

Date funzioni $f, g : \mathbf{N} \to \mathbf{R}$, diremo che f è *di ordine non superiore a* g, e scriveremo $f \in O(g)$ ("f è O-grande di g") se esiste una costante $\alpha \in \mathbf{R}$ tale che $|f(n)| \leq |\alpha g(n)|$ definitivamente. Diremo che f è *di ordine non inferiore a* g, e scriveremo $f \in \Omega(g)$ se $g \in O(f)$. Diremo che f è *dello stesso ordine di* g, e scriveremo $f \in \Theta(g)$, se $f \in O(g)$ e $g \in O(f)$.

Un *grafo (semplice)* G è dato da un insieme finito di *vertici* V_G e da un insieme di *lati* $E_G \subseteq \{\{x, y\} \mid x, y \in V_G \wedge x \neq y\}$; ogni lato è cioè una coppia non ordinata di vertici distinti. Se $\{x, y\} \in E_G$, diremo che x e y sono *adiacenti* in G. Un grafo può essere rappresentato graficamente disegnando i suoi vertici come punti sul piano, e rappresentando i lati come segmenti che congiungono vertici adiacenti. Per esempio, il grafo con insieme di vertici 4 e insieme di lati $\{\{0, 1\}, \{1, 2\}, \{2, 0\}, \{2, 3\}\}$ si può rappresentare così:

[3] L'uguaglianza è vera in generale, utilizzando i cardinali cantoriani.

[4] Questa notazione è in perfetto accordo con l'abbreviazione X^n per la potenza cartesiana: infatti, le ennuple ordinate sono naturalmente equivalenti all'insieme delle funzioni da n in X.

[5] Si noti che Σ^* possiede una naturale struttura di monoide, dove la composizione è data dalla concatenazione, e l'elemento neutro è la stringa vuota ε. Rispetto a queste operazioni, Σ^* è libero su Σ.

L'*ordine* di G è il numero naturale $|V_G|$. Una *cricca (clique)* di G è un insieme di vertici $C \subseteq V_G$ mutuamente adiacenti (nell'esempio in figura, $\{0, 1, 2\}$ è una cricca). Dualmente, un *insieme indipendente* di G è un insieme di vertici $I \subseteq V_G$ mutuamente non adiacenti. Un *cammino* (di lunghezza n) in G è una sequenza di vertici $x_0, x_1, \ldots, x_n$ tale che x_i è adiacente a x_{i+1} ($0 \leq i < n$). Nell'esempio in figura, $1, 0, 2$ è un cammino, mentre $1, 3$ non lo è.

Un'*accoppiamento (matching)* è un sottoinsieme A di lati tali che ogni vertice del grafo fa parte al più di un lato in A. Un accoppiamento A è detto *massimo* se non esiste un accoppiamento B con $|B| > |A|$.

Una *colorazione* di G è una funzione dai vertici di G in una insieme finito C di *colori*. È detta *corretta* se assegna colori diversi a ogni coppia di vertici adiacenti.

2 Alcune questioni di cardinalità

Un insieme X ha *la stessa cardinalità* di un insieme Y se esiste una funzione biiettiva $f : X \to Y$; in tal caso, scriviamo $|X| = |Y|$. Scriveremo $|X| \leq |Y|$ se $|X|$ ha la stessa cardinalità di un sottoinsieme di Y (equivalentemente, se esiste una funzione iniettiva da X a Y). I teoremi di Zermelo e di Cantor–Schröder–Bernstein garantiscono che due cardinalità sono sempre confrontabili, e che se $|X| \leq |Y|$ e $|X| \geq |Y|$ allora $|X| = |Y|$. Un insieme X è detto *numerabile* se $|X| = |\mathbf{N}|$. Un insieme X tale che $|X| \leq |\mathbf{N}|$ è detto *al più numerabile*. Dimostriamo subito il

Teorema 1 (Cantor [Can74]) $|X| < \left|2^X\right|$.

Dimostrazione. Chiaramente $|X| \leq \left|2^X\right|$ (poiché $\{x\} \in 2^X$ per ogni $x \in X$, e quindi l'assegnamento $x \mapsto \{x\}$ definisce una funzione iniettiva da X a 2^X). Assumiamo per assurdo l'esistenza di una biiezione $f : X \to 2^X$, e consideriamo l'insieme

$$Y = \{x \in X \mid x \notin f(x)\}.$$

Ci chiediamo ora se $z = f^{-1}(Y) \in Y$. Se $z \in Y$, allora z soddisfa la formula che definisce Y, e quindi $z \notin f(z) = Y$. D'altra parte, se $z \notin Y$, allora z *non* soddisfa la formula che definisce Y, e quindi $z \in f(z) = Y$. ∎

Per contro, costruiamo esplicitamente alcune biiezioni tra insiemi apparentemente di dimensione diversa. La prima biiezione (detta *coppia di Cantor*) va da $\mathbf{N}^2$ a $\mathbf{N}$, ed è ottenuta nel seguente

modo: consideriamo la matrice infinita a due indici in $\mathbf{N}$. Possiamo percorrere l'intera matrice come segue: iniziamo da $\langle 0, 0\rangle$, poi continuiamo con $\langle 1, 0\rangle$, $\langle 0, 1\rangle$, poi $\langle 2, 0\rangle$, $\langle 1, 1\rangle$, $\langle 0, 2\rangle$ e così via. Formalmente, al passo k scandiamo le k coppie $\langle x, y\rangle$ tali che $x + y = k$. Esiste anche una formula esplicita per la codifica della coppia $\langle x, y\rangle$:

$$\langle\langle x, y\rangle\rangle = \binom{x + y + 1}{2} + y,$$

che può essere invertita in maniera elementare utilizzando l'estrazione di radici.

Leggermente più sofisticata è la costruzione di una biiezione tra $\mathbf{N}^*$ (le sequenze finite di elementi di $\mathbf{N}$) e $\mathbf{N}$. In tal caso, denotiamo con P_i l'i-esimo numero primo ($P_0 = 2$, $P_1 = 3, \ldots$), e codifichiamo la stringa $x_1 x_2 \cdots x_n$ con

$$P_0^{x_1} P_1^{x_2} P_2^{x_3} \cdots P_{n-1}^{x_n+1} - 1.$$

Il teorema di fattorizzazione unica ci dice che l'assegnazione qui descritta è proprio una biiezione tra $\mathbf{N}$ e $\mathbf{N}^*$. Infatti, ogni numero intero x maggiore di zero è prodotto di primi in maniera essenzialmente unica, e quindi gli possiamo associare la lista infinita degli esponenti $e_0, e_1, e_2, \ldots$ (tale lista è definitivamente nulla). Se la lista è costituita interamente da zeri, la codifica inversa restituisce la stringa vuota. Altrimenti, esiste un massimo indice m tale che e_m è diverso da zero, e la codifica inversa restituisce la stringa $e_0 e_1 \cdots (e_m - 1)$.

In generale, qualunque insieme di enti specificati tramite una quantità finita di informazione è al più numerabile. Informalmente, basta elencare, per ogni intero k, tutti gli enti che richiedono meno di k bit per la loro definizione (tale insieme è certamente finito), e rimuovere via via i duplicati. L'assegnamento di un numero naturale distinto a ogni ente è detta *gödelizzazione* dell'insieme di enti, dato che la prima apparizione di questa tecnica è nella dimostrazione originale del teorema di incompletezza di Gödel [Göd31].

3 Le macchine RAM

La macchina RAM è un modello di calcolo vicino all'architettura reale degli elaboratori. Una macchina RAM possiede un numero infinito di registri $R_1, R_2, \ldots$, ciascuno dei quali contiene un numero intero (ma un programma può utilizzare solo un numero finito di tali registri), e un contatore di programma che serve a individuare la riga in esecuzione.

Definizione 1 Una macchina RAM M è definita da un *programma*, cioè da una sequenza finita di *istruzioni* prese dal seguente insieme:

1. **inc** R_i,
2. **dec** R_i,

3. **if** $R_i \neq 0$ **goto** j,

dove $i, j \in \mathbf{N}$ e $i > 0$. Denoteremo con $M[p]$ l'istruzione di indice $p < |M|$. Una *configurazione* di una macchina RAM è un elemento $\boldsymbol{c} = \langle c_0, c_1, c_2, \ldots \rangle$ di $\mathbf{N}^{\mathbf{N}}$, dove la prima componente è da intendersi come un contatore di programma, mentre c_i ($i > 0$) è il contenuto del registro R_i. Una configurazione $\boldsymbol{c}$ è detta *finale* se $c_0 \geq |M|$ (vale a dire, se il contatore di programma punta oltre la fine della sequenza di istruzioni). Data una configurazione $\boldsymbol{c}$ e una macchina RAM M, la *configurazione successiva* $\delta_M(\boldsymbol{c})$ si ottiene applicando nel modo ovvio $M[c_0]$. Più precisamente:

1. se $c_0 \geq |M|$, $\delta_M(\boldsymbol{c}) = \boldsymbol{c}$;
2. se $M[c_0] =$ "**inc** R_i", $\delta_M(\boldsymbol{c}) = \langle c_0 + 1, c_1, \ldots c_{i-1}, c_i + 1, c_{i+1}, \ldots \rangle$;
3. se $M[c_0] =$ "**dec** R_i", $\delta_M(\boldsymbol{c}) = \langle c_0 + 1, c_1, \ldots c_{i-1}, \max\{0, c_i - 1\}, c_{i+1}, \ldots \rangle$;
4. se $M[c_0] =$ "**if** $R_i \neq 0$ **goto** j",

$$\delta_M(\boldsymbol{c}) = \begin{cases} \langle j, c_1, c_2, \ldots \rangle & \text{se } c_i \neq 0 \\ \langle c_0 + 1, c_1, c_2, \ldots \rangle & \text{se } c_i = 0. \end{cases}$$

L'esecuzione della macchina M su input $\langle i_1, \ldots, i_n \rangle \in \mathbf{N}^n$ è la sequenza infinita di configurazioni

$$\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^t, \ldots$$

definita come segue:

- $\boldsymbol{c}^0 = \langle 0, i_1, \ldots, i_n, 0, 0, \ldots \rangle$;
- $\boldsymbol{c}^{t+1} = \delta_M(\boldsymbol{c}^t)$.

L'esecuzione è detta *convergente* o *terminante* se per qualche t la configurazione $\boldsymbol{c}^t$ è finale (cioè $c_0^t \geq |M|$), *divergente* o *non terminante* altrimenti. Il contenuto di R_1 al termine di un'esecuzione convergente è considerato l'*output* di M. Possiamo quindi definire una funzione parziale $\varphi_M^{(n)}$ calcolata da M come segue:

$$\varphi_M^{(n)}(x_1, \ldots, x_n) = \begin{cases} c_1^t & \text{Se } M \text{ converge su input } \langle x_1, \ldots, x_n \rangle \text{ e } \boldsymbol{c}^t \text{ è finale} \\ \perp & \text{altrimenti.} \end{cases}$$

L'insieme di istruzioni di una macchina RAM è volutamente scarno: in questo modo, infatti, diventa più semplice dimostrare teoremi *sulle* macchine RAM. Per semplificare la dimostrazione di teoremi *tramite* macchine RAM, vale la pena di descrivere alcune macroistruzioni facilmente implementabili tramite una tediosa ma banale rinumerazione delle variabili e dei salti.

Una sequenza di macroistruzioni è detta *macroprogramma* (ovviamente ogni programma è anche un macroprogramma). L'espansione di un macroprogramma in un programma si ottiene ricorsivamente, espandendo una macroistruzione scelta arbitrariamente e trasformando il macroprogramma risultante, fino a ottenere un programma (il lettore può facilmente convincersi che la procedura ha termine, purché le macroistruzioni non possano chiamarsi ricorsivamente). Si noti che l'espansione delle macroistruzioni verrà definita in maniera dipendente dal macroprogramma in cui compaiono, e che tale espansione a volte può richiedere modifiche sintattiche al macroprogramma stesso.

1. $R_i \leftarrow 0$. Azzera R_i (non utilizza nessun registro aggiuntivo).
2. **goto** k. Salta a k (non modifica alcun registro, ma utilizza R_1).
3. **if** $R_i = 0$ **goto** k. Salta a k se il contenuto di R_i è uguale a zero (non utilizza nessun registro aggiuntivo).
4. $R_i \leftarrow R_j$. Copia R_j in R_i, utilizzando un registro aggiuntivo che non compare nel macroprogramma corrente.
5. $R_i \leftarrow_M R_{i_1}, R_{i_2}, \ldots, R_{i_n}$. Questa macroistruzione è di facile realizzazione, ma molto potente: inserisce nel macroprogramma corrente il programma di M, rinumerando in modo opportuno i salti e sommando agli indici dei registri il massimo indice p di un registro nel macroprogramma corrente, in modo da evitare conflitti; il programma di M è preceduto dalla copia dei registri $R_{i_1}, R_{i_2}, \ldots, R_{i_n}$ nei registri $R_{p+1}, R_{p+2}, \ldots, R_{p+n}$, e seguito dalla copia di R_{p+1} in R_i.

Utilizzeremo inoltre liberamente etichette per designare simbolicamente i numeri di linea. Per esempio, $R_i \leftarrow 0$ si può implementare come segue:

loop:
  dec R_i
  if $R_i \neq 0$ **goto** loop

mentre **goto** k è

  inc R_1
  if $R_1 \neq 0$ **goto** k
  dec R_1
    ...
  inc R_1
k **dec** R_1,

dove ovviamente vanno modificati gli altri salti presenti nel programma in maniera che non vengano eseguite istruzioni di incremento o decremento indesiderate. La macroistruzione **if** $R_i = 0$ **goto** k si può scrivere come segue:

if $R_1 \neq 0$ **goto** cont
goto k
cont:

Un esempio di macroistruzione che utilizza registri aggiuntivi è $R_i \leftarrow R_j$ (assumiamo t maggiore di qualunque indice di registro del macroprogramma):

$R_i \leftarrow 0$
$R_t \leftarrow 0$
loop:
if $R_j = 0$ **goto** loop2
dec R_j
inc R_i
inc R_t
goto loop
loop2:
if $R_t = 0$ **goto** done
dec R_t
inc R_j
goto loop2:
done:

Chiaramente se $i = j$ la macroistruzione è semplicemente ε. Infine, mostriamo come realizzare $R_i \leftarrow_M R_{i_1}, R_{i_2}, \ldots, R_{i_n}$: assumiamo che p il massimo indice di registro nel macroprogramma, e che il massimo indice di registro che compare in M sia q:

$R_{p+1} \leftarrow R_{i_1}$
$R_{p+2} \leftarrow R_{i_2}$
$\ldots$
$R_{p+n} \leftarrow R_{i_n}$
$R_{p+n+1} \leftarrow 0$
$R_{p+n+2} \leftarrow 0$
$\ldots$
$R_{p+q} \leftarrow 0$
[M con i salti rinumerati e gli indici dei registri incrementati di p]
$R_i \leftarrow R_{p+1}$

Con queste macroistruzioni non è difficile, per esempio, scrivere il programma che calcola la somma di due interi:

loop:
  if $R_2 = 0$ **goto** end
  dec R_2
  inc R_1
  goto loop
end:

4 Le funzioni parziali ricorsive

Definizione 2 La classe delle funzioni parziali ricorsive, denotata da $\mathscr{R}$, è la più piccola classe di funzioni parziali da $\mathbf{N}^n$ in $\mathbf{N}$ (n arbitrario) tale che

1. la funzione costante 0 appartiene a $\mathscr{R}$;
2. la funzione successore s appartiene a $\mathscr{R}$;
3. le proiezioni i-esime $\pi_i^{(n)} : \mathbf{N}^n \to \mathbf{N}$ appartengono a $\mathscr{R}$;
4. se $f : \mathbf{N}^m \to \mathbf{N}$ e $g_i : \mathbf{N}^n \to \mathbf{N}$, per $1 \leq i \leq m$, appartengono a $\mathscr{R}$, allora la funzione parziale $h : \mathbf{N}^n \to \mathbf{N}$ ottenuta per *sostituzione*

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

   appartiene a $\mathscr{R}$;
5. se $g : \mathbf{N}^{n-1} \to \mathbf{N}$ e $h : \mathbf{N}^{n+1} \to \mathbf{N}$ appartengono a $\mathscr{R}$, allora la funzione parziale f definita per *ricorsione primitiva*

$$\begin{aligned} f(0, x_2, x_3, \ldots, x_n) &= g(x_2, x_3, \ldots, x_n) \\ f(y+1, x_2, x_3, \ldots, x_n) &= h(f(y, x_2, x_3, \ldots, x_n), x_2, x_3, \ldots, x_n, y) \end{aligned}$$

   appartiene a $\mathscr{R}$;
6. se $g : \mathbf{N}^{n+1} \to \mathbf{N}$ appartiene a $\mathscr{R}$ ed è totale, allora la funzione parziale f ottenuta per *minimalizzazione*

$$f(x_1, \ldots, x_n) = \mu g(y, x_1, \ldots, x_n) = \min\{y \in \mathbf{N} \mid g(y, x_1, \ldots, x_n) = 0\}$$

   appartiene a $\mathscr{R}$.

Le funzioni (parziali) ricorsive sono importanti perché i molteplici formalismi sviluppati in questo secolo per individuare la classe di funzioni (parziali) calcolabili algoritmicamente hanno sempre portato (modulo ovvie codifiche) all'individuazione della classe $\mathscr{R}$. Questo ha indotto Church alla formulazione della sua famosa tesi:

> Le funzioni calcolabili algoritmicamente sono le funzioni ricorsive.

O, in forma più generale,

> Le funzioni semicalcolabili algoritmicamente sono le funzioni parziali ricorsive.

Si noti che la nozione di "funzione (semi)calcolabile algoritmicamente" è necessariamente di carattere intuitivo. La tesi di Church è fondamentale come strumento euristico, dato che ci permette di costruire dimostrazioni in maniera antropica, anziché formale.

Mostriamo ora qualche semplice esempio di definizione ricorsiva. Le funzioni costanti sono composizioni iterate della funzione successore. La funzione somma $f : \mathbf{N}^2 \to \mathbf{N}$ può essere definita per ricorsione primitiva come segue:

$$\begin{aligned} f(0, x) &= \pi_1^{(1)}(x) = x \\ f(y+1, x) &= s(f(y, x)). \end{aligned}$$

Analogamente, la funzione caratteristica dei numeri pari $\chi_{2\mathbf{N}}$ è data da

$$\begin{aligned} \chi_{2\mathbf{N}}(0) &= 1 \\ \chi_{2\mathbf{N}}(y+1) &= 1 - \chi_{2\mathbf{N}}(y). \end{aligned}$$

In generale, le funzioni definite senza utilizzare la minimalizzazione sono dette *ricorsive primitive*, e sono sempre totali.

Vediamo ora un esempio più completo: vogliamo definire una funzione che restituisce il minimo divisore primo di x. Sarà a questo scopo necessario definire una serie di funzioni intermedie, che svolgono lo stesso ruolo delle subroutine nei linguaggi di programmazione. Innanzitutto, notiamo che mediante ricorsione possiamo definire il decremento

$$\begin{aligned} \mathrm{dec}(0) &= 0 \\ \mathrm{dec}(y+1) &= y, \end{aligned}$$

e che tramite il decremento possiamo definire un'operazione

$$\begin{aligned} \ominus(0, x) &= x \\ \ominus(y+1, x) &= \mathrm{dec}(\ominus(y, x)), \end{aligned}$$

(si noti che $(z+y)\ominus x = z\ominus(y\ominus x)))$ e definire così la differenza troncata $x \dot{-} y = \ominus(y, x)$. In modo completamente analogo alla definizione della somma (utilizzando il fatto che $(y+1)x = yx + x$) è possibile anche definire il prodotto.

Abbiamo a questo punto bisogno di alcune funzioni che rappresentano relazioni sugli interi. In particolare, per quanto riguarda il confronto possiamo dare la seguente definizione:

$$\chi_{\{0\}}(0) = 1$$
$$\chi_{\{0\}}(y+1) = 0,$$

la cui correttezza si dimostra immediatamente per induzione, che ci permette di stabilire se un numero è uguale a zero. A questo punto è facile ottenere una rappresentazione della relazione d'ordine:

$$\geq(x, y) = \chi_{\{0\}}(y \dot{-} x)$$

e dell'uguaglianza:

$$\equiv(x, y) = \geq(x, y) \cdot \geq(y, x).$$

Date le precedenti definizioni, possiamo finalmente scrivere

$$f(z, x, y) = x(y \dot{-} zx),$$

sicché $\mu f(x, y)$ fornisce, su input x, y, il minimo z tale che $zx \geq y$, o 0 se $x = 0$ (la moltiplicazione per x serve per evitare la nonterminazione quando $x = 0$). Chiaramente x divide y se e solo se $\mu f(x, y) \cdot y = x$, e quindi la funzione $d(x, y)$ che vale 1 se e solo se x divide y è in $\mathscr{R}$ (in notazione prefissa, $d(x, y) = \equiv(\cdot(\mu f(x, y), x), y)$).

Non ci rimane che trovare un modo di determinare la primalità: anche in questo caso possiamo utilizzare la minimizzazione sulla funzione

$$g(y, x) = (1 \dot{-} (x \equiv y)) \cdot (1 \dot{-} d(x \dot{-} 1 \dot{-} y, x)),$$

ottenendo una funzione $\mu g(x)$ tale che $x \dot{-} 1 \dot{-} \mu g(x)$ è il *massimo* intero minore di x che divide x, o 0 se $x < 2$ (si noti che x compare con segno negativo nella definizione di g, e quindi l'operatore di minimizzazione funziona "a rovescio", e che $g(x, x) = 0$, e quindi la minimizzazione è sempre definita). Tale intero è uguale a 1 se e solo se x è primo. Poniamo quindi

$$p(x) = (x \dot{-} 1 \dot{-} \mu g(x) \equiv 1).$$

Non ci rimane che combinare d e p nella funzione

$$h(x, y) = 1 \dot{-} d(x, y) \cdot p(x),$$

che vale 0 se e solo se x è primo e divide y, e finalmente minimizzare h per avere la funzione che ci eravamo prefissati di ottenere.

Un altro esempio interessante che ci verrà utile nella dimostrazione del teorema di equivalenza tra le macchine RAM e le funzioni parziali ricorsive è la *definizione per casi*. Supponiamo di avere funzioni parziali ricorsive $f : \mathbf{N}^n \to \mathbf{N}$, $g_i : \mathbf{N}^n \to \mathbf{N}$ per $0 \leq i \leq k$, e di voler definire $h : \mathbf{N}^n \to \mathbf{N}$ come segue:

$$h(x_1, \ldots, x_n) = \begin{cases} g_0(x_1, \ldots, x_n) & \text{se } f(x_1, \ldots, x_n) = 0 \\ g_1(x_1, \ldots, x_n) & \text{se } f(x_1, \ldots, x_n) = 1 \\ \ldots & \\ g_{k-1}(x_1, \ldots, x_n) & \text{se } f(x_1, \ldots, x_n) = k-1 \\ g_k(x_1, \ldots, x_n) & \text{altrimenti.} \end{cases}$$

La costruzione descritta corrisponde intuitivamente al costrutto sintattico di *selezione* dei linguaggi imperativi (`switch` in C). Per ottenere h è sufficiente notare che

$$\begin{aligned} h(x_1, \ldots, x_n) \quad = \quad & g_0(x_1, \ldots, x_n) \cdot (f(x_1, \ldots, x_n) \equiv 0) + \\ & + g_1(x_1, \ldots, x_n) \cdot (f(x_1, \ldots, x_n) \equiv 1) + \cdots + \\ & + g_{k-1}(x_1, \ldots, x_n) \cdot (f(x_1, \ldots, x_n) \equiv k-1) + \\ & + g_k(x_1, \ldots, x_n) \cdot \prod_{i=0}^{k-1} [1 - (f(x_1, \ldots, x_n) \equiv i)]. \end{aligned}$$

Infatti il primo addendo è zero tranne quando $f(x_1, \ldots, x_n) = 0$, nel qual caso vale $g_0(x_1, \ldots, x_n)$, il secondo addendo è zero tranne quando $f(x_1, \ldots, x_n) = 1$, nel qual caso vale $g_1(x_1, \ldots, x_n)$, e così via fino all'ultimo addendo, che vale 0 quando $0 \leq f(x_1, \ldots, x_n) < k$, e $g_k(x_1, \ldots, x_n)$ altrimenti.

4.1 Equivalenza tra macchine RAM e funzioni parziali ricorsive

A supporto della tesi di Church, dimostriamo ora il seguente

Teorema 2 Le funzioni parziali ricorsive coincidono con le funzioni parziali calcolabili da macchine RAM.

Dimostrazione. La dimostrazione procede per induzione strutturale. Le funzioni base non pongono alcun problema. Per la sostituzione, possiamo assumere l'esistenza di macchine $M_1, \ldots, M_m$ che calcolano $g_1, \ldots, g_m$, rispettivamente, e di una macchina M_f che calcola f; in tal caso, il seguente macroprogramma calcola $f(g_1, \ldots, g_m)$:

$R_{n+1} \leftarrow_{M_1} R_1, R_2, \ldots, R_n$
$R_{n+2} \leftarrow_{M_2} R_1, R_2, \ldots, R_n$

$R_{n+3} \leftarrow_{M_3} R_1, R_2, \ldots, R_n$
$\ldots$
$R_{n+m} \leftarrow_{M_m} R_1, R_2, \ldots, R_n$
$R_1 \leftarrow_{M_f} R_{n+1}, R_{n+2}, \ldots, R_{n+m}$

Per quanto riguarda la ricorsione primitiva, siano M_g e M_h le macchine che calcolano $g : \mathbf{N}^{n-1} \to \mathbf{N}$ e $h : \mathbf{N}^{n+1} \to \mathbf{N}$. Utilizzeremo R_{n+1} come contatore, mentre R_{n+2} conterrà il numero di iterazioni da effettuare:

$R_{n+2} \leftarrow R_1$
$R_{n+1} \leftarrow 0$
$R_1 \leftarrow_{M_g} R_2, R_3, \ldots, R_n$
loop:
if $R_{n+2} = 0$ **goto** end
$R_1 \leftarrow_{M_h} R_1, R_2, \ldots, R_{n+1}$
dec R_{n+2}
inc R_{n+1}
goto loop
end:

Mostriamo ora che, per ogni input $x_1, x_2, \ldots, x_n$, all'$(i+1)$-esimo passaggio ($i \leq x_1$) dall'etichetta "loop" i registri $R_1, \ldots, R_{n+2}$ contengono $f(i, x_2, \ldots, x_n), x_2, \ldots, x_n, i, x_1 - i$. Questo significa che il programma terminerà all'(x_1+1)-esimo passaggio, lasciando in R_1 il valore $f(x_1, x_2, \ldots, x_n)$, come desiderato.

La base dell'induzione è immediata, dato che la prima volta ($i = 0$) che arriviamo a "loop", il registro R_1 contiene $g(x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n)$, i registri $R_2, \ldots, R_n$ sono inalterati, R_{n+1} contiene 0 e R_{n+2} contiene x_1. D'altra parte, se assumiamo il teorema vero all'$(i+1)$-esimo passaggio, le tre istruzioni all'interno del ciclo fanno sì che all'$(i+2)$-esimo passaggio R_1 contenga $h(f(i, x_2, \ldots, x_n), x_2, \ldots, x_n, i) = f(i+1, x_2, \ldots, x_n)$, $R_2, \ldots, R_n$ siano inalterati, R_{n+1} contenga $i+1$ ed R_{n+2} contenga $x_1 - i - 1$; il programma è quindi corretto.

Infine, data una macchina M_g per $g : \mathbf{N}^{n+1} \to \mathbf{N}$, μg può essere calcolata come segue:

$R_{n+1} \leftarrow 0$
loop:
$R_{n+2} \leftarrow_{M_g} R_{n+1}, R_1, R_2, \ldots, R_n$
if $R_{n+2} = 0$ **goto** end
inc R_{n+1}
goto loop
end:
$R_1 \leftarrow R_{n+1}$

La dimostrazione di correttezza è lasciata al lettore.

Dobbiamo ora mostrare che le funzioni parziali calcolate dalle macchine RAM sono ricorsive. Sia M una macchina RAM; certamente M utilizza solo una quantità finita di registri, diciamo $R_1, \ldots, R_k$. È chiaro che una configurazione di M è determinata da un elemento di $\mathbf{N}^{k+1}$, cioè da una $(k+1)$-pla di naturali, dato che tutti i registri rimanenti hanno sempre contenuto uguale a zero; utilizzando induttivamente la coppia di Cantor, possiamo costruire una biiezione $\mathbf{N}^{k+1} \to \mathbf{N}$ (stiamo cioè gödelizzando $\mathbf{N}^{k+1}$), il che ci permette di considerare una funzione da $\mathbf{N}$ in $\mathbf{N}$ come una funzione da $\mathbf{N}^{k+1}$ in $\mathbf{N}^{k+1}$. Denoteremo con p_i $(1 \leq i \leq k+1)$ le proiezioni cantoriane, che estraggono da un naturale, pensato come $(k+1)$-upla di naturali, l'i-esimo elemento. È facile verificare che la funzione di codifica e le proiezioni sono ricorsive primitive. Tramite ricorsione primitiva, che permette la definizione per casi, possiamo codificare la funzione di transizione di M in una funzione ricorsiva $f : \mathbf{N} \to \mathbf{N}$: vale a dire,

$$f(\langle\!\langle c_0, \ldots, c_k \rangle\!\rangle)$$

rappresenta (dopo avere "spacchettato" la gödelizzazione) la configurazione

$$\delta_M(\langle c_0, c_1, \ldots, c_k, 0, 0, \ldots \rangle).$$

Formalmente,

$$f(\langle\!\langle c_0, \ldots, c_k \rangle\!\rangle) = \begin{cases} \langle\!\langle c_0, \ldots, c_k \rangle\!\rangle & \text{se } c_0 \geq |M| \\ \langle\!\langle c_0 + 1, \ldots, c_i + 1, \ldots, c_k \rangle\!\rangle & \text{se } M[c_0] = \text{“}\textbf{inc } R_i\text{”} \\ \langle\!\langle c_0 + 1, \ldots, c_i - 1, \ldots, c_k \rangle\!\rangle & \text{se } M[c_0] = \text{“}\textbf{dec } R_i\text{”} \\ \langle\!\langle j, c_1, \ldots, c_k \rangle\!\rangle & \text{se } M[c_0] = \text{“}\textbf{if } R_i \neq 0 \textbf{ goto } j\text{”} \text{ e } c_i \neq 0 \\ \langle\!\langle c_0 + 1, c_1, \ldots, c_k \rangle\!\rangle & \text{altrimenti.} \end{cases}$$

Definiamo ora per ricorsione primitiva la funzione $g : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ come segue:

$$\begin{aligned} g(0, z) &= z \\ g(t+1, z) &= f(g(t, z)). \end{aligned}$$

In questo modo abbiamo che

$$g(t, \langle\!\langle c_0, \ldots, c_k \rangle\!\rangle) = f^t(\langle\!\langle c_0, \ldots, c_k \rangle\!\rangle),$$

cioè g *itera* la funzione di transizione di M codificata in f. Chiaramente,

$$h(\langle\!\langle 0, i_1, \ldots, i_n, 0, 0, \ldots, 0 \rangle\!\rangle) = \mu\,(|M| - p_1(g(t, \langle\!\langle 0, i_1, \ldots, i_n, 0, 0, \ldots, 0 \rangle\!\rangle)))$$

è il numero di passi dopo cui M si arresta su input $i_1, \ldots, i_n$, o $\perp$ se M diverge su tale input. Ma allora

$$\varphi_M^{(n)}(i_1, \ldots, i_n) = p_2(g(h(\langle\!\langle 0, i_1, \ldots, i_n, 0, 0, \ldots, 0 \rangle\!\rangle), \langle\!\langle 0, i_1, \ldots, i_n, 0, 0, \ldots, 0 \rangle\!\rangle)). \blacksquare$$

4.2 Gödelizzazione delle macchine RAM e indecidibilità algoritmica

Una macchina RAM è definita tramite una quantità finita di informazione (la sequenza di istruzioni). Possiamo quindi associare a ogni macchina RAM M un numero di codice[6] z, detto il *numero di Gödel* di M e definire la funzione parziale ricorsiva $\varphi_z^{(n)}$ come la funzione parziale (con n argomenti; l'apice viene omesso quando $n = 1$) calcolata dalla macchina con numero di Gödel z. Il teorema 2 ci dice che *ogni* funzione parziale ricorsiva ha (almeno un) numero di Gödel.

Possiamo finalmente dimostrare il primo risultato di indecidibilità:

Teorema 3 (Post 1922, Gödel [Göd31], Kleene [Kle36]) La funzione parziale

$$f(x) = \begin{cases} 0 & \text{se } \varphi_x(x) = \perp \\ \perp & \text{altrimenti.} \end{cases}$$

non è ricorsiva.

Dimostrazione. Assumiamo per assurdo che f sia ricorsiva, e sia z il numero di Gödel di una macchina RAM che calcola f. L'assurdo è immediato, dato che $f(z) = 0$ sse $\varphi_z(z) = \perp$ sse $f(z) = \perp$. ∎

La funzione parziale f definita sopra non sembra molto significativa, ma *contiene il nucleo combinatoriale di tutti i risultati di indecidibilità*. Procediamo quindi con il seguente

Corollario 1 (Turing [Tur36]) La *funzione d'arresto*

$$g(x, y) = \begin{cases} 0 & \text{se } \varphi_x(y) = \perp \\ 1 & \text{altrimenti} \end{cases}$$

non è ricorsiva.

Dimostrazione. Basta notare che

$$f(x) = \begin{cases} 0 & \text{se } g(x, x) = 0 \\ \perp & \text{altrimenti.} \end{cases}$$

e quindi la ricorsività di g implicherebbe quella di f. ∎

Quindi, *la rilevazione della terminazione di un programma non è ricorsiva*. Non è quindi possibile scrivere un programma che, analizzando staticamente un codice sorgente, dica (in modo valido e completo) se durante l'esecuzione si verificheranno dei loop infiniti.

[6] Il lettore interessato alla codifica effettiva può procedere come segue: associamo prima di tutto un numero naturale a ogni istruzione, per esempio assegnamo a **inc** R_i il numero $3i - 3$, a **dec** R_i il numero $3i - 2$ e a **if** $R_i = 0$ **goto** j il numero $3\langle\langle i - 1, j\rangle\rangle - 1$. In questo modo abbiamo stabilito una biiezione tra $\mathbf{N}$ e le istruzioni. Basta a questo punto utilizzare la biiezione $\mathbf{N}^* \to \mathbf{N}$ descritta nel paragrafo 2.

4.3 La macchina universale

La fondamentale scoperta di Turing è l'esistenza di macchine (e quindi di indici) *universali*:

Teorema 4 (Post 1922, Turing [Tur36]) Per ogni n esiste una macchina RAM che su input $z, x_1, \ldots, x_n$ converge se e solo se la macchina RAM di codice z converge su input $x_1, \ldots, x_n$, e nel caso terminante ha lo stesso output. In particolare, esiste $u \in \mathbf{N}$ tale che $\varphi_u^{(2)}(x, y) = \varphi_x(y)$.

Dimostrazione. L'idea è di creare una macchina RAM M_n che legga la definizione contenuta in z ed emuli passo a passo la macchina corrispondente (l'indice u richiesto è il numero di Gödel di M_1). ∎

La funzione parziale ricorsiva $\varphi_u^{(2)}$ fornita dal teorema precedente è detta *funzione parziale universale*[7], dato che è in grado di simulare il comportamento di qualunque altra funzione parziale ricorsiva con un argomento (ovviamente, è possibile fare lo stesso per funzioni parziali a più argomenti).

4.4 Insiemi ricorsivi e ricorsivamente enumerabili

Definizione 3 Un insieme $A \subseteq \mathbf{N}$ è *ricorsivo* o *decidibile* se χ_A è ricorsiva, *non ricorsivo* o *indecidibile* altrimenti. Un insieme $A \subseteq \mathbf{N}$ è *ricorsivamente enumerabile* o *semidecidibile* se è il rango di una funzione parziale ricorsiva[8].

Gli insiemi ricorsivi sono quelli di cui è possibile stabilire algoritmicamente l'appartenenza. Esempi banali di insiemi ricorsivi sono gli insiemi finiti, i numeri pari, i numeri primi, o i numeri di Gödel di macchine RAM che contengono almeno un'istruzione condizionale. Gli insiemi ricorsivamente enumerabili sono invece quelli di cui è possibile in tempo finito stabilire l'appartenenza, ma di cui non è in generale possibile stabilire la non appartenenza. Per chiarire le idee, vale la pena di dimostrare subito alcuni teoremi. In particolare, il teorema seguente chiarisce il termine "enumerabile":

Teorema 5 (Kleene [Kle36]) Un insieme $A \subseteq \mathbf{N}$ è ricorsivamente enumerabile sse è vuoto o è l'immagine di una funzione ricorsiva.

Dimostrazione. Sia $a \in A = \operatorname{ran} \varphi_z$, e si consideri la macchina RAM M che su input x, t emula la macchina RAM con numero di Gödel z su input x per t passi. Se l'emulazione termina entro t passi, M termina e restituisce x, altrimenti restituisce a. Chiaramente, $\varphi_M^{(2)}$ è totale e ha come immagine A.

[7] Si noti che la funzione parziale universale e il suo indice variano al variare dell'enumerazione scelta per le macchine RAM.

[8] Cioè se esiste una funzione parziale ricorsiva f tale che $A = \{x \in \mathbf{N} \mid f(x) \neq \perp\}$.

Viceversa, se $A = \text{imm}\, f$ con f totale, si consideri la macchina RAM che su input x calcola in sequenza $f(0)$, $f(1)$, ... e così via, fino a incontrare eventualmente x. Se $x \in A$, la macchina si arresta (l'output è irrilevante). Se $x \notin A$, la macchina non terminerà mai l'enumerazione. Dunque, A è il rango della funzione parziale calcolata dalla macchina. ∎

Teorema 6 (Kleene [Kle36]) Un insieme $A \subseteq \mathbf{N}$ è ricorsivo sse A e $\mathbf{N} \setminus A$ sono ricorsivamente enumerabili.

Dimostrazione. L'implicazione da sinistra a destra è banale (basta comporre χ_A con la funzione parziale che mappa 1 in sé e 0 in $\perp$ o viceversa). Se, d'altra parte, esistono due funzioni ricorsive[9] $f, g : \mathbf{N} \to \mathbf{N}$ tali che $\text{imm}\, f = A$ e $\text{imm}\, g = \mathbf{N} \setminus A$, consideriamo la macchina RAM che su input x calcola in sequenza $f(0)$, $g(0)$, $f(1)$, $g(1)$, ... e aspetta di incontrare x. Prima o poi x deve apparire nell'enumerazione (dato che $\text{imm}\, f \cup \text{imm}\, g = \mathbf{N}$); se compare nell'immagine di f, la macchina restituisce 1, altrimenti restituisce 0. Chiaramente tale macchina calcola la funzione caratteristica di A, che risulta quindi essere ricorsivo. ∎

Corollario 2 Un insieme è ricorsivo sse lo è il suo complemento.

Le dimostrazioni precedenti di non ricorsività suggeriscono che il seguente insieme di indici sia importante:

$$K = \{x \in \mathbf{N} \mid \varphi_u^{(2)}(x, x) \neq \perp\}.$$

L'insieme K contiene gli indici delle macchine RAM che si arrestano quando sono lanciate utilizzando il proprio numero di Gödel come input. Dimostriamo subito un risultato fondamentale, che è la controparte insiemistica del teorema 3:

Teorema 7 L'insieme K è ricorsivamente enumerabile, ma non è ricorsivo.

Dimostrazione. Consideriamo la macchina RAM M che su input x emula la macchina RAM di indice x su input x. Chiaramente K è l'insieme di definizione della funzione parziale calcolata da M, ed è quindi ricorsivamente enumerabile. D'altra parte, $\mathbf{N} \setminus K$ è esattamente il rango della funzione parziale f del teorema 3, che abbiamo dimostrato non essere ricorsiva. Se esistesse una funzione parziale ricorsiva g di cui $\mathbf{N} \setminus K$ fosse il rango, avremmo $0 \circ g = f$, il che implicherebbe la ricorsività di f. ∎

Corollario 3 L'insieme $\mathbf{N} \setminus K$ non è ricorsivamente enumerabile.

Analogamente, è immediato dare la controparte insiemistica del corollario 1:

[9] In generale f e g possono avere un qualunque numero di argomenti, ma la precomposizione con una gödelizzazione inversa del loro dominio ci riporta senza perdita di generalità al caso considerato.

Corollario 4 L'insieme

$$\text{Halt} = \{\langle\langle x, y\rangle\rangle \in \mathbf{N} \mid \varphi_u^{(2)}(x, y) \neq \perp\},$$

(da *halting problem*), detto *insieme di arresto*,[10] è ricorsivamente enumerabile, ma non ricorsivo.

La seguente proposizione stabilisce alcune proprietà di chiusura degli insiemi ricorsivamente enumerabili.

Proposizione 1 Se $A, B \subseteq \mathbf{N}$ sono ricorsivamente enumerabili, lo sono anche $A \cup B$ e $A \cap B$.

Dimostrazione. Siano f e g le funzioni ricorsive che enumerano A e B. Allora, la funzione h definita da

$$h(x) = \begin{cases} f(x/2) & \text{per } x \text{ pari} \\ g((x-1)/2) & \text{per } x \text{ dispari} \end{cases}$$

enumera $A \cup B$. D'altra parte, se f e g sono funzioni parziali ricorsive tali che $\operatorname{ran} f = A$ e $\operatorname{ran} g = B$, $\operatorname{ran}(f + g) = A \cap B$. ∎

Ora alziamo un po' il tiro: essenzialmente, mostriamo che tutte le proprietà non banali dei programmi sono indecidibili.

Definizione 4 Un insieme $A \subseteq \mathbf{N}$ è *chiuso funzionalmente* se per ogni $x \in \mathbf{N}$ e $y \in A$ tali che $\varphi_x = \varphi_y$ si ha $x \in A$.

Gli insiemi chiusi funzionalmente definiscono le proprietà *semantiche* (o *estensionali*) dei programmi: sono le proprietà che non dipendono dal testo del programma (cioè dalla sua *intensione*), ma solo dalla funzione parziale che il programma calcola (cioè dalla sua *estensione*). Esempi di insiemi chiusi funzionalmente sono Halt, l'insieme degli indici di funzioni o l'insieme degli indici di funzioni costanti.

Teorema 8 (Rice [Ric53]) Gli insiemi ricorsivi chiusi funzionalmente sono solo $\varnothing$ e $\mathbf{N}$.

Dimostrazione. Si assuma per assurdo che esista un insieme ricorsivo A chiuso funzionalmente tale che $\varnothing \subset A \subset \mathbf{N}$; sia inoltre $a \in A$. Assumiamo senza perdita di generalità che la funzione parziale completamente indefinita ($\perp$) non abbia indice in A (altrimenti scambiamo A e $\mathbf{N} \setminus A$), e consideriamo la funzione ricorsiva primitiva g definita come segue: $g(x)$ è il numero di Gödel della macchina RAM che controlla se $x \in K$, e, in caso di risposta positiva, simula la macchina di indice a. Chiaramente,

$$\varphi_{g(x)} = \begin{cases} \varphi_a & \text{se } x \in K \\ \perp & \text{altrimenti.} \end{cases}$$

[10] Si noti che K è semplicemente la diagonalizzazione dell'insieme di arresto.

Ma allora $x \in K$ sse $\varphi_{g(x)} = \varphi_a$ sse $g(x) \in A$. Se quindi A fosse ricorsivo, lo sarebbe anche K. ∎

Un altro esempio importante di non ricorsività è dato dall'enumerazione delle funzioni ricorsive. La domanda che ci poniamo è la seguente: è possibile elencare *solo* macchine RAM che terminano sempre in maniera da rappresentare *tutte* le funzioni ricorsive?[11] Come conseguenza, otterremmo una funzione ricorsiva g che enumera le funzioni ricorsive, cioè tale che $\varphi_{g(x)}$ è una funzione ricorsiva, e ogni funzione ricorsiva è uguale a $\varphi_{g(x)}$ per qualche x.[12]

Teorema 9 (Kleene [Kle36], Turing [Tur36]) Non esiste alcuna enumerazione delle funzioni ricorsive.

Dimostrazione. Se esistesse una funzione ricorsiva g come sopra, la funzione f definita da

$$f(x) = \varphi_u^{(2)}(g(x), x) + 1 = \varphi_{g(x)}(x) + 1$$

sarebbe ricorsiva, ma non potrebbe apparire nell'enumerazione, dato che differirebbe su almeno un input da ogni funzione nell'enumerazione. ∎

Si noti che il risultato di Kleene *non* è conseguenza del teorema di Rice. Il teorema di Rice, infatti, implica che l'insieme di *tutti* gli indici di funzioni ricorsive non è ricorsivo. Il teorema di Kleene, per contro, ha come conseguenza che *tutti* gli insiemi di indici che contengono almeno un indice di ogni funzione ricorsiva non sono neppure ricorsivamente enumerabili.

Per concludere, mostriamo come la definizione che abbiamo dato di insieme ricorsivo e ricorsivamente enumerabile possa a sua volta essere utilizzata per ridefinire le funzioni (parziali) ricorsive. Data una funzione parziale $f : \mathbf{N}^n \to \mathbf{N}$, la *gödelizzazione di* f, denotata con $\bar{f}$, è il sottoinsieme di $\mathbf{N}$ che codifica f (ricordiamo che f è un sottoinsieme di $\mathbf{N}^{n+1}$).

Proposizione 2 Una funzione è ricorsiva sse lo è la sua gödelizzazione. Una funzione parziale è ricorsiva sse la sua gödelizzazione è ricorsivamente enumerabile.

Dimostrazione. Sia $f : \mathbf{N}^n \to \mathbf{N}$ una funzione ricorsiva. Per decidere se $x \in \bar{f}$, dove $x = \langle\langle x_1, \ldots, x_n, y\rangle\rangle$, basta controllare se $f(x_1, \ldots, x_n) = y$. D'altra parte, se $\bar{f}$ è ricorsivo, per calcolare $f(x_1, \ldots, x_n)$ basta elencare gli elementi di $\bar{f}$ fino a trovarne uno della forma $\langle\langle x_1, \ldots, x_n, y\rangle\rangle$ (tale elemento esiste certamente a causa della totalità di f). La stessa dimostrazione, *mutatis mutandis*, prova anche il secondo asserto. ∎

[11] Si noti che non stiamo chiedendo di *decidere* se una macchina termina, ma solo di *elencare* gli indici di macchine che terminano.

[12] Il termine "funzione ricorsiva", qui e nel prossimo teorema, va inteso con precisione, e cioè in maniera distinta da "funzione *parziale* ricorsiva".

4.5 Cenni sull'indecidibilità dell'aritmetica

Senza entrare nei dettagli, mostriamo come i risultati precedenti portino a conseguenze importanti per la dimostrazione automatica di teoremi:

Teorema 10 (Church [Chu36]) Se l'aritmetica è coerente, l'insieme dei suoi teoremi è indecidibile.

Dimostrazione. Il punto chiave della dimostrazione è la possibilità di definire una formula predicativa del prim'ordine $P(v, w)$, con due variabili libere v e w, che è vera se e solo se la macchina RAM universale M termina su input v, w. Informalmente, la formula dice semplicemente che "esiste un t tale che al passo t la macchina universale si arresta su input v, w". La definizione esatta di P è laboriosa, ma non difficilissima: utilizzando la notazione che abbiamo fissato nella seconda parte della dimostrazione del teorema 2 (specializzata alla macchina universale), si potrebbe scrivere un po' meno informalmente come

$$\exists t \; p_1(g(t, \langle\langle 0, v, w, 0, \ldots, 0\rangle\rangle)) \geq |M|.$$

La parte laboriosa da dimostrare è la possibilità di definire una formula che *rappresenti* g nell'aritmetica (si veda per esempio [Odi89]).

Supponiamo ora per assurdo che esista un algoritmo che, data una formula chiusa dell'aritmetica, dica se si tratta di un teorema o no. Potremmo allora decidere se $x \in K$ semplicemente controllando se[13] $P(\bar{x}, \bar{x})$ è un teorema. ∎

Corollario 5 (Primo teorema di Gödel [Göd31]) Se l'aritmetica è coerente, è incompleta.

Dimostrazione. La dimostrazione è immediata sulla base del teorema di Church: se l'aritmetica fosse completa, potremmo decidere se una formula chiusa F è un teorema o no semplicemente enumerando tutti i teoremi, e aspettando di incontrare F o $\neg F$. ∎

Le dimostrazioni descritte sono ovviamente valide per qualunque sistema formale abbastanza potente da formalizzare le funzioni parziali ricorsive (o l'aritmetica) e il cui insieme di assiomi sia ricorsivamente enumerabile. Si noti però che non siamo stati in grado di *costruire* una formula vera ma non dimostrabile dell'aritmetica. Per questo è necessaria la piena potenza della dimostrazione originale di Gödel, che è decisamente al di là degli scopi del corso. Possiamo però qui mostrare il tipo di diagonalizzazione utilizzato da Gödel: il punto chiave della dimostrazione sta nel mostrare che esiste una formula $\mathrm{Dim}(y, x)$ con due variabili libere che dice che "y è la gödelizzazione di una dimostrazione della formula chiusa il cui numero di Gödel è x".[14] Da qui

[13] Con $P(\bar{x}, \bar{x})$ si intende la formula ottenuta sostituendo il numerale $\bar{x}$ di x al posto di v e di w.

[14] Per costruire una tale formula basta la potenza delle funzioni ricorsive primitive.

possiamo costruire una formula $\mathrm{Teo}(x) = \exists y\, \mathrm{Dim}(y, x)$ che dice che x è il numero di Gödel di un teorema dell'aritmetica. A questo punto gödelizziamo le formule con una variabile libera: sia $\psi_n(x)$ la formula con numero di Gödel n, e consideriamo la formula $\neg\mathrm{Teo}(\psi_x(\bar{x}))$ ("non si può dimostrare che il numero x soddisfa la formula con numero di Gödel x"), che, avendo una variabile libera, deve avere un numero di Gödel, diciamo g (si noti che x compare una volta come indice, e una volta tramite il suo numerale; occorre quindi un po' di lavoro per far vedere che la scrittura abbreviata $\psi_x(\bar{x})$ è descrivibile nell'aritmetica del prim'ordine). Ci chiediamo se

$$\neg\mathrm{Teo}(\psi_g(\bar{g}))$$

sia dimostrabile o no. Se lo fosse, allora sarebbe vera (per la validità del calcolo predicativo), e dunque sarebbe vero che non si può dimostrare $\psi_g(\bar{g})$. Ma $\psi_g(x)$ è $\neg\mathrm{Teo}(\varphi_x(\bar{x}))$, e quindi $\psi_g(\bar{g})$ è $\neg\mathrm{Teo}(\varphi_g(\bar{g}))$, il che è assurdo perché abbiamo assunto quest'ultima dimostrabile. Rimane solo il caso in cui $\neg\mathrm{Teo}(\psi_g(g))$ non sia dimostrabile: in questo caso, *la formula risulta vera*, dato che asserisce precisamente la propria indimostrabilità.

4.6 Alcuni risultati di teoria della ricorsione

Cominciamo con un risultato tecnico, ma fondamentale, il cosiddetto *teorema s-m-n*, o *teorema del parametro*:

Teorema 11 (Kleene [Kle38]) Per ogni m ed n esiste una funzione ricorsiva primitiva $s^n_m : \mathbf{N}^{n+1} \to \mathbf{N}$ tale che

$$\varphi^{(m+n)}_z(x_1, \ldots, x_m, y_1, \ldots, y_n) = \varphi^{(m)}_{s^n_m(z, y_1, \ldots, y_n)}(x_1, \ldots, x_m).$$

Dimostrazione. Facciamo la dimostrazione nel caso $m = n = 1$ (il caso generale viene dimostrato per induzione). Sia M la macchina che ha numero di Gödel z. Consideriamo la macchina M_{y_1} così definita

inc R_2
inc R_2
...
inc R_2
[M con i salti rinumerati]

dove ci sono esattamente y_1 istruzioni aggiuntive di incremento. Il codice della nuova macchina si può calcolare in modo ricorsivo primitivo a partire da z e da y_1, e tale numero è, ovviamente, $s^1_1(z, y_1)$. ∎

Il teorema *s-m-n*, dall'apparenza innocua, ci dice che è possibile incorporare costanti in un programma, riducendone il numero di argomenti, in maniera ricorsiva primitiva. Si noti che però

il teorema dice anche (dato che ogni numero è una macchina RAM) che è possibile incorporare *codice*.

Dimostriamo ora il *teorema di ricorsione*:

Teorema 12 (Kleene [Kle38]) Sia $g : \mathbf{N}^{n+1} \to \mathbf{N}$ una funzione parziale ricorsiva. Allora esiste $e \in \mathbf{N}$ tale che

$$\varphi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n).$$

Dimostrazione. Definiamo

$$f(x_1, \ldots, x_n, y) = g(s_n^1(y, y), x_1, \ldots, x_n).$$

Poiché f è ricorsiva, esiste $z \in \mathbf{N}$ tale che $\varphi_z^{(n+1)} = f$. Sia $e = s_n^1(z, z)$. Allora, per ogni $x_1, \ldots, x_n \in \mathbf{N}$

$$\begin{aligned} g(e, x_1, \ldots, x_n) &= f(x_1, \ldots, x_n, z) \\ &= \varphi_z^{(n+1)}(x_1, \ldots, x_n, z) \\ &= \varphi_{s_n^1(z,z)}^{(n)}(x_1, \ldots, x_n) \\ &= \varphi_e^{(n)}(x_1, \ldots, x_n). \blacksquare \end{aligned}$$

Corollario 6 (Teorema del punto fisso) Per ogni funzione ricorsiva $t : \mathbf{N} \to \mathbf{N}$ e per ogni $n \in \mathbf{N}$ esiste $e \in \mathbf{N}$ tale che

$$\varphi_{t(e)}^{(n)} = \varphi_e^{(n)}.$$

Dimostrazione. Si consideri $g(z, x_1, \ldots, x_n) = \varphi_{t(z)}^{(n)}(x_1, \ldots, x_n)$. Poiché g è ricorsiva, esiste $e \in \mathbf{N}$ tale che $\varphi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n) = \varphi_{t(e)}^{(n)}(x_1, \ldots, x_n)$. $\blacksquare$

Corollario 7 Qualunque compilatore compila correttamente almeno un programma.

Corollario 8 Esiste un $e \in \mathbf{N}$ tale che φ_e è la funzione costante e (esiste un programma che stampa se stesso).

Dimostrazione. Sia $t : \mathbf{N} \to \mathbf{N}$ una funzione ricorsiva tale che $\varphi_{t(x)}$ sia la funzione costante x. Ad esempio, tale funzione può associare a x il numero di Gödel della macchina che esegue x incrementi del registro R_1 dopo averlo azzerato. Il teorema del punto fisso dice allora che esiste un e tale che

$$e = \varphi_{t(e)}^{(n)} = \varphi_e^{(n)}. \blacksquare$$

5 Le macchine di Turing

In questo paragrafo andiamo a introdurre un nuovo modello di calcolo, che pur essendo molto lontano dall'architettura degli elaboratori permette, paradossalmente, una definizione precisa delle risorse tempo e spazio.

Definizione 5 Una macchina di Turing M a $k+1$ nastri è una settupla $\langle Q, \Sigma, I, q_0, q_S, q_N, \delta \rangle$, dove

1. Q è un insieme finito di *stati*;
2. Σ è l'alfabeto *di lavoro* della macchina;
3. I è l'alfabeto *di input* della macchina;
4. $q_0 \in Q$ è lo stato *iniziale*;
5. $q_S \in Q$ è lo stato *finale accettante*;
6. $q_N \in Q$ è lo stato *finale non accettante*;
7. $\delta_M : Q \times (I + \{\_\}) \times (\Sigma + \{\_\})^k \to Q \times \{D, S, F\} \times (\Sigma \times \{D, S, F\})^k$ è la *funzione di transizione* della macchina. Per ogni stato, simbolo sul nastro di input e k-upla di simboli sui nastri di lavoro, la funzione di transizione restituisce un nuovo stato, le istruzioni di movimento per le $k+1$ testine (destra, sinistra, ferma) e il simbolo da scrivere sui k nastri di lavoro.

Il contenuto dei nastri, la posizione delle testine e lo stato di una macchina di Turing ne definiscono la *configurazione*, o *stato complessivo*. Si assume per convenzione che la funzione di transizione non alteri lo stato complessivo della macchina se questa è nello stato finale.

Utilizzando la funzione di transizione, una macchina di Turing passa in maniera deterministica da una configurazione all'altra. Più precisamente

Definizione 6 Una *configurazione* di una macchina di Turing M è una $(k+1)$-upla di stringhe $\langle w_0, w_1, \ldots, w_k \rangle$, dove $w_0 \in (Q + I + \{\_\})^*$ e $w_i \in (Q + \Sigma + \{\_\})^*$ per $i > 0$. Ogni stringa è della forma $u_i q s_i v_i$, dove $q \in Q$, $u_i, v_i \in \Sigma^*$ (o I^* se $i = 0$) e $s_i \in \Sigma + \{\_\}$ (o $I + \{\_\}$ se $i = 0$). Il contenuto del nastro i-esimo è dato da w_i da cui sia stato cancellato q, e la posizione di q all'interno di w_i determina la posizione della testina all'interno del nastro i-esimo; in particolare, s_i è il simbolo correntemente sotto la testina. Una configurazione è detta finale se lo stato che compare in tutte le sue stringhe è finale.

È immediato calcolare, a partire da una configurazione C, la configurazione che si ottiene applicando la funzione di transizione δ_M nella configurazione C.

Definizione 7 Una *computazione* di una macchina di Turing M su input $z \in I^*$ è una sequenza infinita di configurazioni

$$C_0, C_1, C_2, \ldots, C_t, \ldots$$

dove C_0 è la configurazione con tutti i nastri di lavoro vuoti, stato uguale a q_0 e nastro di input contenente z, vale a dire, la configurazione $\langle q_0 z, q_{0\_}, \ldots, q_{0\_}\rangle$, e C_{t+1} è ottenuta da C_t applicando δ_M. La computazione è *convergente* o *terminante* se per qualche t la configurazione C_t è finale, *divergente* o *nonterminante* altrimenti.

Le macchine di Turing possono essere utilizzate come *riconoscitori di linguaggi* nel seguente modo:

Definizione 8 Se M è una macchina di Turing con alfabeto di input I, il *linguaggio riconosciuto da* M, denotato da L_M, è l'insieme dei $w \in I^*$ tali che su input w la macchina M si arresta nello stato accettante.

Ogni linguaggio, essendo l'alfabeto finito, è gödelizzabile. Più precisamente, assumendo senza perdita di generalità che I sia l'insieme $\{1, 2, \ldots, n\}$, abbiamo la seguente gödelizzazione di I^*:

$$s_k s_{k-1} \cdots s_1 s_0 \;\mapsto\; s_k n^k + s_{k-1} n^{k-1} + \cdots + s_1 n + s_0,$$

che, oltre a essere biiettiva, ha l'utile proprietà di mappare l'ordine lessicografico di stringhe nel comune ordine tra numeri interi. Possiamo quindi identificare i linguaggi su I^* con sottoinsiemi di $\mathbf{N}$, e dare l'equivalenza computazionale tra macchine di Turing e $\mathscr{R}$:

Teorema 13 (Turing [Tur36]) I linguaggi riconosciuti da macchine di Turing sono i linguaggi ricorsivamente enumerabili.

La dimostrazione è estremamente tecnica, ma consiste semplicemente nel mostrare che è possibile emulare una qualunque macchina RAM con una macchina di Turing e viceversa.

È chiaramente possibile anche definire una *funzione parziale calcolata da una macchina di Turing* : tale funzione parziale va da I^* ad Σ^*, ed è indefinita quando la macchina non termina, mentre ha come valore il contenuto dell'ultimo nastro di lavoro in caso di terminazione. Anche in questo caso (modulo gödelizzazione) si può mostrare che le macchine di Turing calcolano le funzioni parziali ricorsive.

Una macchina di Turing è definita tramite una quantità finita di informazione. In effetti, per descrivere una macchina è sufficiente fornire $|Q|$, $|I|$, $|\Sigma|$, il numero corrispondente agli stati iniziali e finali, e una tabulazione (finita) di δ, cioè una tabella che ha tante righe quanti gli elementi del dominio di δ, e contiene su ogni riga l'output corrispondente. Analogamente a quanto fatto per le macchine RAM, possiamo quindi associare a ogni macchina di Turing M il suo numero di Gödel, e costruire una macchina di Turing universale.

6 Complessità strutturale

La complessità strutturale studia la difficoltà intrinseca di soluzione di un problema, e in particolare le risorse necessarie per risolverlo, indipendentemente dal modello di calcolo e dall'algoritmo scelti. Studieremo in particolare *problemi di decisione*, e cioè problemi consistenti nel dire se una certa *istanza* (per esempio, un grafo o un numero) soddisfa una certa proprietà (per esempio, possedere una cricca di dimensione specificata, o essere primo). Le istanze dei problemi di decisione saranno codificate tramite stringhe di un alfabeto.

6.1 Problemi di decisione

Definizione 9 Un *problema di decisione* sull'alfabeto Σ è un linguaggio $L \subseteq \Sigma^*$.

In genere utilizzeremo l'alfabeto binario 2. Un problema computazionale specifica quali stringhe corrispondono a domande la cui risposta corretta è "sì". Ad esempio, si consideri il problema "p è primo?", dove p è un numero naturale. Utilizzando la rappresentazione binaria L sarà il linguaggio delle stringhe che cominciano con 1 e che rappresentano un numero primo: $\{10, 11, 101, 111, 1011, \ldots\}$. Si noti che assimiliamo le stringhe che non corrispondono del tutto a rappresentazioni di numeri (cioè quelle che cominciano con 0) e quelle che rappresentano numeri composti.

Chiaramente, per passare da un problema di decisione in senso intuitivo alla sua formalizzazione è necessaria una *codifica* degli enti che entrano in gioco. Per esempio, abbiamo codificato nell'esempio precedente i numeri tramite l'alfabeto binario, ma avremmo potuto anche utilizzare l'alfabeto unario 1 e codificare p con

$$\overbrace{0000\cdots00}^{p \text{ volte}},$$

ottenendo una codifica estremamente ridondante. È inoltre importante che sia possibile riconoscere effettivamente L (cioè che sia ricorsivo). Le stringhe che codificano un ente sono dette *istanze* del problema. Le istanze che fanno parte di L sono le istanze *positive* (cioè quelle per cui la risposta è "sì"), mentre quelle fuori da L sono le istanze *negative*. Si assume che tutte le stringhe in L siano istanze.

Il modello che utilizzeremo per analizzare problemi di decisione è la macchina di Turing. Anche se tale modello sembra irrealistico, è possibile dimostrare che il tempo impiegato da una macchina di Turing (misurato come numero di applicazioni della funzione di transizione) è di ordine non superiore al cubo del tempo impiegato da una macchina RAM (dove il tempo necessario per l'accesso o la modifica di un registro è il logaritmo del contenuto del registro) per lo stesso problema. In generale, la *tesi del calcolo sequenziale* afferma che

Tutti i modelli ragionevoli di calcolo sequenziale sono in relazione polinomiale di complessità con le macchine di Turing.

Finalmente possiamo definire il costo in tempo e spazio:

Definizione 10 Una macchina di Turing M ha *costo in tempo* $T_M : \mathbf{N} \to \mathbf{N}$ se $T_M(n)$ è il massimo tempo di arresto di una computazione terminante su input di lunghezza n, e un *costo in spazio* $S_M : \mathbf{N} \to \mathbf{N}$ se $S_M(n)$ è il massimo numero di celle di nastro di lavoro distinte utilizzate da una computazione terminante su input di lunghezza n.

In generale, assoceremo a una macchina di Turing l'*andamento asintotico* delle sue funzioni costo, anziché le funzioni esatte. La ragione sta nel famoso

Teorema 14 (di *speed-up*, Hartmanis e Stearns [HS65]) Sia L un linguaggio riconoscibile da una macchina di Turing M con costo in tempo f; allora, per ogni $\varepsilon > 0$ esiste una macchina di Turing che riconosce L in tempo $\varepsilon f(n) + n + 2$ su input di lunghezza n.

La dimostrazione del teorema precedente richiede una sofisticata simulazione di M tramite una macchina che esegue, in un solo passo, più (circa $\lceil 1/\varepsilon \rceil$) passi di M. Anche lo spazio può essere moltiplicato per una costante:

Teorema 15 (di compressione, Hartmanis e Stearns [HS65]) Sia L un linguaggio riconoscibile da una macchina di Turing M con costo in spazio f; allora, per ogni $\varepsilon > 0$ esiste una macchina di Turing che riconosce L con costo in spazio εf.

La dimostrazione costruisce una macchina di Turing che rappresenta con un solo simbolo diversi simboli di M, comprimendo così i nastri, a scapito della dimensione dell'alfabeto di lavoro e del numero di stati.

6.2 Classi di complessità

Definizione 11 La classe TEMPO (f) è la classe dei linguaggi riconoscibili dalle macchine di Turing M tali che $T_M \leq f$. La classe SPAZIO (f) è la classe dei linguaggi riconoscibili dalle macchine di Turing M tali che $S_M \leq f$.[15]

Vale la pena di notare immediatamente le relazioni fra tempo e spazio:

[15] La funzione f non deve essere arbitraria, o si può cadere in situazioni patologiche (si veda [BC98]). Tutte le funzioni analitiche non presentano questo problema, e saranno le sole utilizzate in questo corso. Più precisamente, assumeremo che le funzioni coinvolte siano *costruibili in tempo e spazio*, e cioè che esista una macchina di Turing che si arresta in esattamente $f(n)$ passi su input di lunghezza n, e una macchina di Turing che si arresta dopo aver utilizzato esattamente $f(n)$ celle di nastro di lavoro su input di lunghezza n. Non è difficile, per esempio, immaginare due macchine per il caso $f(n) = n$. Si noti però che log *non* è costruibile in tempo.

Teorema 16 Se $M = \langle Q, \Sigma, I, q_0, q_S, q_N, \delta \rangle$ ha $k+1$ nastri,

$$S_M \leq kT_M.$$

Dimostrazione. Affinché una cella del nastro di lavoro sia utilizzata, la testina deve essere posizionata sopra di essa. Ma in una macchina di Turing con $k+1$ nastri, a ogni passo di esecuzione si possono visitare al più k nuove celle dei nastri di lavoro. ∎

Teorema 17 Se $M = \langle Q, \Sigma, I, q_0, q_S, q_N, \delta \rangle$ ha $k+1$ nastri,

$$T_M(n) \leq n \left[(|\Sigma| + 1)^{S_M(n)} S_M(n) \right]^k |Q|.$$

Dimostrazione. Le possibili configurazioni di una macchina di Turing con $k+1$ nastri che lavora in spazio S_M sono tante quante i possibili contenuti dei nastri di lavoro, posizioni delle testine e stati. Dato che nessuna computazione su input di lunghezza n usa più di $S_M(n)$ celle, le possibili configurazioni sono al più $n \left[(|\Sigma| + 1)^{S_M(n)} S_M(n) \right]^k |Q|$. Basta a questo punto notare che una macchina di Turing non può visitare due volte la stessa configurazione, o entrerebbe in loop infinito. ∎

Si noti che non abbiamo mai richiesto che le macchine di Turing che utilizziamo terminino su ogni input, e il problema di stabilire se una macchina di Turing si arresta o no su ogni input in tempo f non è neppure ricorsivamente enumerabile. Data però una macchina di Turing che lavora in tempo f, è facile costruire una macchina che si ferma sempre, riconosce lo stesso linguaggio e lavora in tempo $O(f)$. Basta considerare una macchina M' che si arresta esattamente[16] dopo $f(n)$ passi su input n, e metterla in parallelo con M. Se M' si ferma prima di M rifiutiamo l'input, altrimenti diamo il responso di M.

Meno ovvio è che lo stesso procedimento è applicabile allo *spazio*: in questo caso, modifichiamo M (che assumiamo lavorare in spazio $f \geq \log$) in modo che su input di lunghezza n tenga un contatore utilizzando $cf(n)$ celle di nastro, per un'opportuna costante c tale che $2^{cf(n)}$ sia maggiore del numero di configurazioni di M su input n. Il contatore viene incrementato a ogni transizione di M. Chiaramente, non appena il contatore riempie tutte le $cf(n)$ celle sappiamo che M è in un ciclo infinito, e interrompiamo la computazione.

Possiamo ora introdurre uno dei protagonisti di questa parte del corso:

Definizione 12 La classe P è la classe dei linguaggi riconosciuti in tempo polinomiale nella lunghezza dell'input. Vale a dire,[17]

$$\mathrm{P} = \bigcup_{k \in \mathbf{N}} \mathrm{TEMPO}\left(n^k\right).$$

[16] L'esistenza di questa macchina è dovuta al fatto che f è costruibile in tempo.

[17] La scrittura esatta della formula richiederebbe $\lambda n.n^k$.

La classe P è *robusta*: se l'avessimo definita tramite le macchine RAM avremmo ottenuto la stessa classe di linguaggi. Se accettiamo la tesi del calcolo sequenziale, P *è indipendente dal modello di calcolo*. I problemi in P sono quelli considerati *trattabili*, in quanto possiamo trovare le loro soluzioni in tempo ragionevole. In effetti, la *tesi di Church estesa* dice che

> I problemi di decisione trattabili sono quelli risolubili in tempo polinomiale nella lunghezza dell'input.

Fino a ora abbiamo utilizzato le macchine di Turing per *riconoscere* linguaggi. Mostriamo ora come utilizzarle per *verificare* l'appartenenza a un linguaggio. Un nastro è detto *a senso unico* (*one way*) se la sua testina può muoversi solo verso destra.

Definizione 13 Diciamo che una macchina di Turing M con un nastro addizionale di input (detto *nastro dei certificati*) a senso unico *verifica* il linguaggio L se

1. qualunque sia il contenuto del nastro dei certificati, se $w \notin L$ allora M non accetta w;
2. per ogni $w \in L$ esiste una stringa $w' \in I^*$ (detta *certificato o prova d'appartenenza di* w) tale che M accetta w se trova w' sul nastro dei certificati.[18]

La classe NTEMPO (f) è la classe di linguaggi verificabili da una macchina di Turing con costo in tempo f: vale a dire, per ogni input accettato w di lunghezza n esiste un certificato w' tale che su input w, w' la macchina termina in al più $f(n)$ passi.[19] La classe NSPAZIO (f) è la classe di linguaggi verificabili da una macchina di Turing con costo in spazio f: vale a dire, per ogni input accettato w di lunghezza n esiste un certificato w' tale che su input w, w' la macchina utilizza al più $f(n)$ celle di nastro di lavoro. In particolare, la classe

$$\mathrm{NP} = \bigcup_{k \in \mathbf{N}} \mathrm{NTEMPO}\left(n^k\right)$$

è la classe di linguaggi verificabili in tempo polinomiale.

Cerchiamo di dare intuitivamente un'idea della classe NP. I problemi di NP sono linguaggi per le cui stringhe è possibile produrre un *certificato* rapidamente verificabile. Come tipico esempio, consideriamo il problema k-CRICCA: dato in ingresso un grafo G e un intero k, dobbiamo determinare se esiste in G una cricca di almeno k elementi. In questo caso, il certificato è la lista dei vertici della cricca, se tale cricca esiste. Il verificatore controlla (in tempo quadratico) che la lista fornita abbia almeno k vertici, e che siano tutti mutuamente adiacenti, e solo nell'ultimo

[18] Si noti che, alternativamente, possiamo dire che il linguaggio L_M verificato da M è l'insieme delle stringhe $w \in I^*$ tali che esiste un certificato $w' \in I^*$ per cui M accetta su input w, w'.

[19] Si noti che il costo di un verificatore è misurato *solo sulle istanze positive*.

caso accetta. Chiaramente, se il grafo in input non ha una cricca di k elementi, non è possibile fornire un certificato che convinca il verificatore. Quindi, k-CRICCA è in NP.

La classe NP è di grande importanza perché moltissimi problemi di interesse pratico considerati "difficili" stanno in NP. Dato che il verificatore può semplicemente fare a meno di considerare il certificato e procedere al riconoscimento per conto suo, e che lo spazio utilizzato è limitato dal tempo (teorema 16), abbiamo la seguente

Proposizione 3 Valgono le seguenti inclusioni:

1. TEMPO $(f) \subseteq$ NTEMPO (f);
2. SPAZIO $(f) \subseteq$ NSPAZIO (f);
3. TEMPO $(f) \subseteq$ SPAZIO (f);
4. NTEMPO $(f) \subseteq$ NSPAZIO (f).

In particolare, P $\subseteq$ NP.

Il problema fondamentale della complessità strutturale[20] è se l'inclusione del teorema precedente è propria, cioè se

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}.$$

La classe P contiene i problemi che possiamo *risolvere* in tempo polinomiale. La classe NP contiene i problemi di cui possiamo *verificare* le soluzioni in tempo polinomiale.[21] Tornando all'esempio precedente, non conosciamo a tutt'oggi nessun algoritmo che possa neppure *approssimare* il problema k-CRICCA. Essenzialmente, l'unico modo noto per avere una risposta è enumerare tutti i possibili sottoinsiemi di vertici, e controllare se uno di essi è una cricca di dimensione k o più. Essendo il numero di tali sottoinsiemi esponenziale nella dimensione dell'input (perlomeno per alcuni valori di k), un tale algoritmo non è polinomiale. Il teorema seguente mostra che possiamo agire in questo modo anche nel caso generale.

Teorema 18 NTEMPO $(f) \subseteq \bigcup_{c \in \mathbf{N}}$ TEMPO $\left(2^{cf}\right)$.

Dimostrazione. Consideriamo un linguaggio L in NTEMPO (f) e una macchina di Turing M che lo verifica. Chiaramente, su input di lunghezza n il verificatore non può leggere più di $f(n)$

[20] E un problema fondamentale della matematica: recentemente, durante una conferenza europea promossa dall'American Mathematical Society, Steve Smale ha affermato che il problema P $\stackrel{?}{=}$ NP, la congettura di Poincaré e la congettura di Riemann sono i tre problemi aperti più importanti della matematica contemporanea.

[21] La dicotomia tra P e NP come dicotomia tra trovare e verificare risale a molto tempo fa: nel 1666 Leibniz confrontava l'*ars inveniendi* e l'*ars iudicandi*, cioè la capacità di *trovare* dimostrazioni come contrapposta a quella di *controllare* la loro correttezza.

caratteri da un certificato. Chiamiamo M' la macchina di Turing che enumera tutte le $|I|^{f(n)}$ stringhe su I di lunghezza $f(n)$ e simula l'esecuzione di M per ognuna di tali stringhe.[22] Se almeno una simulazione termina nello stato accettante entro $f(n)$ passi, M' accetta, altrimenti rifiuta. Chiaramente M' riconosce L, e numero di passi richiesto dall'emulazione su input di lunghezza n è al più $f(n)|I|^{f(n)} \leq 2^{2\log|I| f(n)}$. ∎

Corollario 9 NP $\subseteq$ EXP, ove

$$\text{EXP} = \bigcup_{k \in \mathbf{N}} \text{TEMPO}\left(2^{n^k}\right)$$

è la classe dei linguaggi riconoscibili in tempo esponenziale.

L'appartenenza a NP non è sempre discernibile a prima vista. Ad esempio, il problema COMPOSTO (stabilire se un dato numero è composto) è chiaramente in NP, dato che su input $m = pq$ possiamo offrire p e q come prova di appartenenza. L'appartenza a NP del suo complemento, PRIMO, non è però affatto immediata: come possiamo dimostrare "in breve" che un numero è primo? In questo caso solo un risultato significativo di teoria dei numeri e un'applicazione astuta dello stesso hanno dimostrato il *teorema di Pratt*, che asserisce che PRIMO$\in$ NP (per una dimostrazione più dettagliata, si veda [BC98]).

Macchine di Turing non deterministiche. La "N" nella definizione di NP (e più in generale nella definizione 13) significa *non deterministico*, e si riferisce al modello originale di definizione di NP. Tale modello, la *macchina di Turing non deterministica*, è una macchina di Turing che a ogni passo di esecuzione può fare due transizioni diverse. Un computazione non è più quindi una sequenza infinita di configurazioni, ma un albero binario infinito di configurazioni, che ha alla radice la configurazione iniziale, e dove i figli di ogni configurazione sono le due configurazioni successive possibili. Una macchina di Turing non deterministica accetta un input se esiste almeno un cammino nell'albero che conduce a una configurazione finale accettante, rifiuta altrimenti. Tali macchine sono equivalenti ai verificatori: infatti, se un problema è risolubile su una macchina di Turing non deterministica con una certa restrizione sulle risorse (per esempio, tempo polinomiale), si può costruire un verificatore con la stessa restrizione e che riconosce lo stesso linguaggio. Il verificatore si aspetta un certificato che descriva il cammino accettante, e controlla che il cammino conduca in effetti a una configurazione accettante. Viceversa, una macchina di Turing non deterministica può emulare un verificatore nel seguente modo: ogni volta che il verificatore legge un simbolo del nastro dei certificati, la macchina di Turing non deterministica biforca la computazione seguendo i due possibili risultati (0 o 1). Chiaramente esiste un cammino accettante sse esiste un certificato.

[22] È chiaro che sarebbe possibile migliorare le prestazioni di questo algoritmo utilizzando, per esempio, tecniche di enumerazione implicita, ma non esiste alcun modo noto di mostrare che l'algoritmo risultante è subesponenziale.

6.3 I teoremi di gerarchia

Dimostriamo ora il nostro primo risultato di separazione. Per farlo, utilizzeremo il seguente lemma, che mostra come ridurre i nastri di lavoro di una macchina di Turing:

Lemma 1 Se L è accettato da una macchina di Turing in tempo f, allora è accettato da una macchina di Turing con un solo nastro di lavoro in tempo $O(f \cdot \log f)$. Se L è accettato da una macchina di Turing in spazio f, allora è accettato da una macchina di Turing con un solo nastro di lavoro in spazio $O(f)$.

Teorema 19 (della gerarchia di spazio, Hartmanis e Stearns [HS65]) Se $g \in \Omega(f) \setminus O(f)$ e $f \in \Omega(\log)$, $\text{SPAZIO}(g) \supset \text{SPAZIO}(f)$.

Dimostrazione. Costruiamo per diagonalizzazione un linguaggio che sta in $\text{SPAZIO}(g)$ ma non in $\text{SPAZIO}(f)$. Per farlo, consideriamo la macchina di Turing M (con alfabeto di input 2) che su input w di lunghezza n esegue il seguente programma: prima di tutto scrive $g(n)$ celle su un nastro di lavoro, poi salta il prefisso della forma $111\cdots10$ dell'input w, controlla che il resto dell'input codifichi una macchina di Turing M_w a due nastri (in caso contrario rifiuta) e la simula su input w (si noti che se Σ è l'alfabeto di lavoro di M_w, M utilizza al più $\lceil \log |\Sigma| \rceil$ celle per simulare una cella di M_w). Nell'effettuare la simulazione, M cura di non utilizzare più di $g(n)$ celle, e per farlo marca una cella delle $g(n)$ predisposte ogni volta che la simulazione utilizza una nuova cella. Se la simulazione può essere completata e termina in uno stato accettante, M rifiuta, altrimenti accetta. È chiaro che M lavora in spazio $O(g)$, e quindi, grazie al teorema di compressione, possiamo assumere senza perdita di generalità che lavori in spazio g.

Sia L_M il linguaggio accettato da M, e assumiamo per assurdo che esista una macchina di Turing M' con alfabeto di lavoro Σ e codifica $u \in 2^*$ che accetta L_M in spazio f; assumiamo inoltre senza perdita di generalità che M' si arresti su ogni input (possiamo farlo perché $f \in \Omega(\log)$) e che abbia un solo nastro di lavoro (utilizzando il lemma 1 e il teorema di compressione se necessario). Poiché $g \notin O(f)$, esiste un N tale che $g(n) > \log |\Sigma| f(n)$ per tutti gli $n \geq N$. Sull'input $1^j 0u$, in cui j è scelto in modo da ottenere una stringa di lunghezza maggiore o uguale a N, M ha abbastanza spazio per completare la simulazione, e quindi $1^j 0u \in L_M$ sse M accetta $1^j 0u$ sse M' rifiuta $1^j 0u$ sse $1^j 0u \notin L_M$. ∎

Quindi, se g cresce più in fretta di f (per quanto di poco), i linguaggi riconoscibili in spazio g contengono strettamente quelli riconoscibili in spazio f. In modo analogo è possibile dimostrare il

Teorema 20 (della gerarchia di tempo) Se $g \in \Omega(f \cdot \log f) \setminus O(f \cdot \log f)$, $\text{TEMPO}(g) \supset \text{TEMPO}(f)$.

Si noti che per dimostrare la separazione in tempo abbiamo bisogno di un *gap* leggermente più grande tra f e g, reso necessario dal fatto che dobbiamo simulare una macchina con un

numero arbitrario di nastri (vedi enunciato del lemma 1). Conseguenza immediata dei teoremi precedenti è che

Corollario 10 P $\subset$ EXP.

Dimostrazione. Basta osservare che $P \subseteq \text{TEMPO}\,(2^n) \subset \text{TEMPO}\left(2^{n^2}\right) \subseteq \text{EXP}$. ∎

6.4 NP-completezza

Il problema P $\stackrel{?}{=}$ NP, pur rimanendo irrisolto, ha spinto gli studiosi a classificare i problemi in NP sulla base della loro difficoltà. Questo ha portato a identificare una classe di problemi particolarmente difficili, ognuno dei quali rappresenta, in un senso che andiamo a rendere preciso, "tutta la difficoltà di NP".

Definizione 14 Il problema $A \subseteq \Sigma^*$ è funzionalmente (o molti-a-uno, o Karp) riducibile in tempo polinomiale al problema $B \subseteq \Theta^*$, e si scrive $A \preceq_m^p B$, se esiste una funzione $f : \Sigma^* \to \Theta^*$, detta *trasformazione (o riduzione) polinomiale*, calcolabile in tempo polinomiale tale che $x \in A$ sse $f(x) \in B$.

La definizione di $A \preceq_m^p B$ formalizza l'idea che B *sia più difficile di* A (si noti che m sta per "molti-a-uno" e p per "polinomiale"). Infatti, se avessimo un algoritmo polinomiale per risolvere B, potremmo risolvere in tempo polinomiale anche A, semplicemente applicando f all'input e risolvendo B. Formalmente,

Proposizione 4 Se $B \in$ P e $A \preceq_m^p B$, allora $A \in$ P.

Costruiamo subito una riduzione di questo tipo tra SODD (stabilire se una formula proposizionale in forma normale congiunta sia soddisfacibile), e k-INDIPENDENTE (stabilire se un grafo ha un insieme indipendente con almeno k vertici).

Teorema 21 SODD $\preceq_m^p$ k-INDIPENDENTE.

Dimostrazione. Data una formula F in forma normale congiunta formata da k clausole sulle variabili proposizionali $x_1, \ldots, x_m$ (il lettore dovrebbe immaginare una possibile codifica in 2^* per tale clausola, e notare che la lunghezza di tale codifica è una limitazione superiore sia per k che per m), costruiamo un grafo G nel seguente modo: per ogni clausola di C composta da l letterali aggiungiamo l vertici, ciascuno etichettato dal corrispondente letterale, uniti in una cricca. Infine, rendiamo adiacenti (cioè poniamo un lato tra) due vertici sse sono etichettati da letterali opposti.

Notiamo ora che l'esistenza in G di un insieme indipendente I con k o più vertici corrisponde a un assegnamento che rende F vera. Infatti, dato che i vertici corrispondenti a una clausola formano una cricca, non possiamo scegliere più di un vertice per clausola; dobbiamo quindi prendere esattamente un vertice per clausola. Inoltre, dato che vertici corrispondenti a letterali opposti sono adiacenti, possiamo costruire un assegnamento che renda veri tutti i letterali che etichettano vertici in I.

D'altra parte, un assegnamento che rende F vera deve rendere vero almeno un letterale per clausola. Se quindi prendiamo per ogni clausola uno dei vertici etichettati da letterali veri, otteniamo un insieme indipendente di cardinalità k. ∎

Notiamo una facile ma importante proprietà di $\preceq_m^p$ (che motiva la decisione di un simbolo simile al "minore o uguale"):

Proposizione 5 Se $A \preceq_m^p B$ e $B \preceq_m^p C$, allora $A \preceq_m^p C$.

Definizione 15 Un problema C è detto *NP-completo* se appartiene a NP, e per ogni problema $B \in \text{NP}$ si ha $B \preceq_m^p C$. La classe dei problemi NP-completi è denotata da NPC.

Un algoritmo polinomiale per un problema NP-completo consentirebbe di risolvere *qualunque* problema in NP in tempo polinomiale. Quindi,

Proposizione 6 Se esiste un problema NP-completo C tale che $C \in \text{P}$ allora $\text{P} = \text{NP}$.

La classe dei problemi NP-completi è perciò di grande importanza, perché ci permette di classificare problemi come presumibilmente intrattabili. In particolare, vale la seguente

Proposizione 7 Se C è NP-completo, A è in NP e $C \preceq_m^p A$ allora A è NP-completo.

Dimostrazione. Sia $B \in \text{NP}$. Componendo la riduzione che mappa B in C (tale riduzione esiste in quanto C è NP-completo) con la riduzione che mappa C in A (che esiste per ipotesi) otteniamo immediatamente il risultato. ∎

Il problema fondamentale, a questo punto, è dimostrare che *almeno un problema è NP-completo*. Il meccanismo delle riduzioni ci permetterà poi di classificare altri problemi come NP-completi, ma senza un primo problema in NPC non abbiamo modo di cominciare. Cominciamo quindi con il dimostrare che almeno un problema (artificioso) è NP-completo:

Teorema 22 Il linguaggio[23]

$$C = \{Mw0^t \mid M \text{ verifica } w \text{ in al più } t \text{ passi}\}$$

è in NPC.

[23] La scrittura $Mw0^t$ denota una stringa di 2^* formata da una descrizione di un verificatore M, da una stringa w (pensata come input per il verificatore) e da t zeri, cioè dal numero t descritto in unario.

Dimostrazione. Prima di tutto si noti che $C \in$ NP. Il certificato dell'input $Mw0^t$ è esattamente il certificato per M su input w, e il verificatore di C emula il comportamento di M, accettando se M accetta entro t passi.

Sia ora A un qualunque problema in NP, M il suo verificatore e p il polinomio che limita il tempo di calcolo di M. Consideriamo la funzione f che mappa w su $Mw0^{p(|w|)}$. Chiaramente f è calcolabile in tempo polinomiale, dato che M è indipendente da w, w va semplicemente copiato e possiamo scrivere $p(|w|)$ zeri in $p(|w|)$ passi. Inoltre, se $w \in A$ esiste un certificato w' tale che M termina su input w, w' in al più $p(|w|)$ passi, e questo implica $Mw0^{p(|w|)} \in C$. D'altra parte, se $Mw0^{p(|w|)} \in C$ per definizione M verifica w in al più $p(|w|)$ passi. ∎

Il problema precedente, pur illustrando l'esistenza di problemi NP-completi, non è di grande utilità, perché non appare evidente come ridurre ulteriormente C ad altri problemi in NP. Passiamo quindi al famoso

Teorema 23 (Cook [Coo71], Levin [Lev73]) SODD $\in$ NPC.

Dimostrazione (cenni). Il fatto che SODD $\in$ NP è immediato, dato che un certificato per la soddisfacibilità di una formula è l'assegnamento che la soddisfa.

Dobbiamo ora mostrare che, dato un arbitrario problema $A \in$ NP, questo si riduce a SODD. Consideriamo un'istanza w di A e sia p il polinomio che limita il tempo di calcolo del verificatore M che riconosce A. È possibile costruire una formula proposizionale di dimensione polinomiale in $|w|$ che dice che M, partendo dallo stato iniziale con w sul nastro di input e una certa stringa sul nastro dei certificati, dopo $p(|w|)$ passi si trova nello stato accettante (le variabili di tale formula codificano, per ogni istante di tempo, una configurazione di M e la stringa sul nastro dei certificati; si noti che possiamo assumere che un nastro non contenga mai più di $p(|w|)$ simboli diversi da _). Una tale formula è soddisfacibile sse $w \in A$, e quindi $A \preceq_m^p$ SODD. ∎

Applichiamo immediatamente il teorema di Cook–Levin:

Teorema 24 k-INDIPENDENTE è NP-completo. k-CRICCA è NP-completo.

Dimostrazione. Il primo asserto è il teorema 21. Per dimostrare il secondo, basta notare che un insieme indipendente di un grafo è una cricca del *grafo complemento*, cioè il grafo che ha gli stessi vertici, ma adiacenza invertita (due vertici sono cioè adiacenti nel complemento sse *non* sono adiacenti nel grafo di partenza), e quindi k-INDIPENDENTE $\preceq_m^p$ k-CRICCA. ∎

Il problema k-COLORABILITÀ chiede, su input $\langle G, k\rangle$, se il grafo G è k-colorabile, cioè se ha una colorazione corretta che usa k colori.

Teorema 25 SODD $\preceq_m^p$ k-COLORABILITÀ. Quindi, k-COLORABILITÀ è NP-completo.

Dimostrazione. Sia $F = \{C_1, \ldots, C_t\}$ una formula in forma clausale congiunta sulle variabili $x_1, \ldots, x_m$. Vogliamo costruire in tempo polinomiale un grafo G che sia $(m+1)$-colorabile sse F è soddisfacibile.

Poniamo

$$V_G = \{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m, v_1, \ldots, v_m, C_1, \ldots, C_t\}.$$

Inoltre:

1. Se $i \neq j$, v_i è adiacente a v_j, a x_j e a $\neg x_j$;
2. x_i è adiacente a $\neg x_i$;
3. un letterale (x_i o $\neg x_i$) è adiacente a C_j sse non vi occorre.

Mostriamo che se F è soddisfacibile con assegnamento α, allora G è $(m+1)$-colorabile. Dato che possiamo eliminare le clausole che contengono sia un letterale che il suo opposto (sono sempre vere), possiamo assumere senza perdita di generalità che le clausole di F contengano meno di m letterali: in caso contrario, aggiungiamo una nuova variabile che non è contenuta in nessuna clausola. Notiamo innanzitutto che dato che G contiene una m-cricca (l'insieme dei v_i) dobbiamo usare almeno m colori, e coloriamo v_i con il colore i. Mostriamo che possiamo colorare i nodi restanti con un solo colore aggiuntivo utilizzando α:

- Se $\alpha(l) = 0$, dove l è un letterale, coloriamo l con il colore $m+1$. I letterali restanti (quelli valutati veri) vengono colorati con il colore dell'unico v_i a cui non sono adiacenti per via di (1).[24]
- Ogni C_i contiene almeno un letterale valutato vero: coloriamo C_i con il colore di quel letterale (nel caso i letterali fossero più di uno la scelta è irrilevante).

È facile verificare che la colorazione così definita è corretta.

Mostriamo ora che se G è $(m+1)$-colorabile, allora F è soddisfacibile. Notiamo nuovamente che l'esistenza della cricca dei v_i implica che tutti i v_i abbiano un colore diverso (diciamo, senza perdita di generalità, che v_i è colorato con il colore i); inoltre, per via di (1) i colori di x_i e $\neg x_i$ sono necessariamente i e $m+1$ (ma non sappiamo chi tra i due letterali opposti abbia colore $m+1$). Definiamo un assegnamento α ponendo $\alpha(x_i) = 0$ sse x_i ha colore $m+1$, e notiamo che per ogni clausola C_j di F esiste un i tale che la clausola non contiene né x_i né $\neg x_i$, ed è quindi adiacente sia a x_i che a $\neg x_i$. Allora C_j non può avere colore $m+1$ (perché almeno uno dei letterali a cui è collegata ha lo stesso colore), e quindi deve avere il colore di uno dei v_k, e cioè il colore di esattamente uno tra x_k e $\neg x_k$. Questo significa che x_k o $\neg x_k$ occorre nella clausola,

[24] Vale a dire, se $\alpha(x_i) = 1$ coloriamo x_i con il colore i, e analogamente per $\neg x_i$.

e quello dei due che occorre è vero. Quindi C_j è vera nell'assegnamento α, e dato che F è la congiunzione delle C_j, è anch'essa vera. ∎

Un problema interessante, e di natura diversa da quelli considerati, è il problema AFFRANCATURA. Dati numeri naturali[25] $a_1, a_2, \ldots, a_k, b$, esistono indici distinti $i_1, i_2, \ldots, i_s$ tali che $b = a_{i_1} + a_{i_2} + \cdots + a_{i_s}$? Detto altrimenti: se devo affrancare una busta con un francobollo di valore b, ce la posso fare (in maniera esatta) con un sottoinsieme dei francobolli $a_1, a_2, \ldots, a_k$?

Teorema 26 SODD $\preceq_m^p$ AFFRANCATURA.

Dimostrazione. L'appartenenza di AFFRANCATURA a NP è banale (il certificato è la lista di indici che specifica la soluzione). Sia ora $F = \{C_1, \ldots, C_t\}$ una formula in forma clausale congiunta sulle variabili $x_1, \ldots, x_m$. Vogliamo costruire in tempo polinomiale un'istanza di AFFRANCATURA che sia risolubile sse F è soddisfacibile.

Per costruire i numeri che rappresentano l'istanza di AFFRANCATURA lavoreremo in base 2β, ove $\beta = \max_i\{|C_i|\}$. Poniamo innanzitutto

$$b = |C_1||C_2|\cdots|C_t|\overbrace{111\cdots11}^{m},$$

ove $|C_i|$ denota la cifra in base 2β corrispondente alla cardinalità di C_i.

La lista dei "francobolli" comprende per ogni letterale l un numero L della forma

$$\overbrace{0110010\cdots10}^{t}\overbrace{000\cdots010\cdots\cdots00}^{m},$$

ove gli uni della parte sinistra corrispondono alle clausole in cui compare l, mentre la posizione dell'unico uno nella parte destra è data dall'indice della variabile che compare in l. Per finire, abbiamo per ogni clausola C_i una lista di $|C_i| - 1$ copie del seguente numero:

$$P_i = \overbrace{000\cdots010\cdots\cdots00}^{t}\overbrace{000\cdots00}^{m},$$

ove la posizione dell'unico uno è data dall'indice della clausola. In totale abbiamo quindi una lista di $2m + \sum_i |C_i| - t$ numeri di lunghezza al più $t + m$ (in base 2β). La lunghezza totale dell'istanza (in base 2) è quindi $O(n^3)$, e la costruzione chiaramente non pone problemi di polinomialità. Si noti che nessuna somma tra i numeri della lista può causare riporti (in base 2β); possiamo quindi verificare l'uguaglianza di una somma con b cifra per cifra.

Costruiamo ora da un assegnamento α tale che $\alpha(F) = 1$ una soluzione per l'istanza associata di AFFRANCATURA. Prendiamo dalla lista i numeri L corrispondenti ai letterali valutati veri, e

[25] Si noti che affiché AFFRANCATURA sia NP-completo è fondamentale che i numeri coinvolti non siano espressi in unario.

per ogni clausola C_i prendiamo tante copie di P_i quanti sono i letterali di C_i valutati falsi; sia c la somma dei numeri così selezionati. È evidente che le ultime m cifre di c saranno esattamente una sequenza di m uni (dato che per ogni coppia di letterali opposti ne scegliamo esattamente uno). Le prime t cifre corrispondono invece alle clausole, e per costruzione abbiamo aggiunto uno alla cifra di posto i per ogni letterale valutato vero (tramite gli L) e per ogni letterale valutato falso (tramite i P_i). Quindi anche le prime t cifre corrispondono a quelle di b, e $b = c$.

Consideriamo ora una soluzione dell'istanza di AFFRANCATURA. Dato che le ultime m cifre di b sono uni, dobbiamo avere scelto esattamente un numero tra i due associati ad ogni coppia di letterali opposti. La scelta dei letterali induce un assegnamento α (l'assegnamento che li rende veri). Notiamo ora che, qualunque sia la scelta tra i P_i, ogni clausola deve contenere un letterale vero, o sarebbe impossibile ottenere $|C_i|$ come cifra i-esima di b. Ne consegue che $\alpha(C_i) = 1$ per ogni i, e quindi $\alpha(F) = 1$. ∎

Il problema AFFRANCATURA è sottilmente (ma profondamente) diverso dai problemi NP-completi incontrati fin qui. Per evidenziare la differenza, consideriamo il seguente "algoritmo polinomiale" per AFFRANCATURA: data l'istanza $a_1, a_2, \ldots, a_k, b$, costruiamo una matrice booleana S di dimensione $k \times b$. Il valore di $S[i, c]$ è il valore di verità della proposizione "l'istanza $a_1, a_2, \ldots, a_i, c$ è positiva".

Chiaramente la riga 1 di S può essere riempita ponendo $S[1, a_1]$ vero. D'altra parte, se abbiamo riempito la riga i possiamo riempire la riga $i + 1$ come segue: per ogni c, se $S[i, c]$ è vero allora poniamo $S[i + 1, c]$ e (se $c + a_{i+1} \leq b$) $S[i + 1, c + a_{i+1}]$ veri. Il riempimento di S richiederà $O(bkn)$ passi (ove n è la lunghezza dell'istanza).

Si noti che la correttezza dell'algoritmo è facilmente dimostrabile per induzione: infatti, se esiste una soluzione per $a_1, a_2, \ldots, a_{i+1}, c$, allora necessariamente ne esiste una per $a_1, a_2, \ldots, a_i, c$ oppure una per $a_1, a_2, \ldots, a_i, c - a_{i+1}$.

Il problema è che mentre la *lunghezza* di b è limitata dalla lunghezza dell'istanza, il suo valore non lo è. L'algoritmo che abbiamo descritto non è dunque polinomiale, anche se, a tutti gli effetti pratici, può essere utilizzato per risolvere il problema, purché b non sia troppo grande.

Algoritmi di questo tipo vengono definiti *pseudopolinomiali*: sono polinomiali nel *valore* degli argomenti numerici dell'istanza, ma non nella loro lunghezza.[26]

Concludiamo dimostrando l'NP-completezza di una generalizzazione classica di AFFRANCATURA, il problema ZAINO. Dato un insieme U di elementi, due funzioni $p, v : U \to \mathbf{N}$ (la funzione *peso* e la funzione *valore*) e due interi P e V, esiste un sottoinsieme $S \subseteq U$ tale che

$$\sum_{u \in S} p(u) \leq P \quad \text{e} \quad \sum_{u \in S} v(u) \geq V?$$

Detto altrimenti, se devo riempire uno zaino con gli elementi di U, è possibile sceglierli in modo che il loro peso non ecceda P, ma il loro valore sia almeno V?

[26] I problemi NP-completi che ammettono algoritmi pseudopolinomiali sono detti NP-completi *in senso debole*.

Teorema 27 AFFRANCATURA $\preceq_m^p$ ZAINO.

Dimostrazione. La dimostrazione è banale, dato che è possibile trasformare un'istanza di AFFRANCATURA in un'istanza di ZAINO per *specializzazione*. Dati infatti $a_1, a_2, \ldots, a_k, b$, poniamo $U = \{1, 2, \ldots, k\}$, $p(i) = v(i) = a_i$ e $P = V = b$. Dato che le limitazioni su P e V si trasformano in uguaglianze, l'istanza di AFFRANCATURA ha soluzione se e solo se l'ha quella corrispondente di ZAINO. ∎

6.5 Le classi di spazio

Definizione 16 La classe PSPAZIO è la classe dei linguaggi riconosciuti in spazio polinomiale nella lunghezza dell'input. Vale a dire,

$$\text{PSPAZIO} = \bigcup_{k \in \mathbf{N}} \text{SPAZIO}\left(n^k\right).$$

Il teorema 16 mostra immediatamente che

Proposizione 8 P $\subseteq$ PSPAZIO.

Per il problema

$$\text{P} \stackrel{?}{=} \text{PSPAZIO},$$

che rappresenta la dicotomia tra le risorse tempo e spazio, non esiste ancora nessun risultato di separazione o di coincidenza.

Definizione 17 La classe LOGSPAZIO è la classe dei linguaggi riconosciuti in spazio logaritmico nella lunghezza dell'input. Vale a dire,

$$\text{LOGSPAZIO} = \text{SPAZIO}\left(\log\right).$$

Si noti che l'uguaglianza $\log n^k = k \log n$ (unitamente al teorema di compressione) mostra che la definizione di LOGSPAZIO è invariante rispetto a variazioni della codifica dell'input, purché le lunghezze siano correlate polinomialmente. Il teorema 17 mostra immediatamente che

Proposizione 9 LOGSPAZIO $\subseteq$ P.

Dato che il teorema della gerarchia di spazio implica che LOGSPAZIO $\subset$ PSPAZIO, abbiamo

Corollario 11 LOGSPAZIO $\subset$ P $\vee$ P $\subset$ PSPAZIO.

Mostriamo un'ulteriore relazione tra tempo e spazio:

Teorema 28 NTEMPO $(f) \subseteq$ SPAZIO (f).

Dimostrazione. Sia M un verificatore che lavora in tempo f. Possiamo simulare il suo comportamento su input di lunghezza n enumerando i certificati di lunghezza $f(n)$ e simulando, per ogni certificato, $f(n)$ passi di M (accettiamo solo se troviamo una computazione di M che termina in uno stato accettante). Ogni simulazione richiede lo stesso spazio, che è al più $f(n)$, e quindi $O(f)$ celle sono sufficienti per completare la simulazione. ∎

Proposizione 10 NP $\subseteq$ PSPAZIO.

Mentre la questione P $\stackrel{?}{=}$ NP rimane per il momento insoluta, nel caso della risorsa spazio si è ottenuto un risultato fondamentale:

Teorema 29 (di Savitch [Sav70]) Se $f \geq \log$ e $L \in$ NSPAZIO (f), allora $L \in$ SPAZIO $(f \cdot f)$

Dimostrazione. Sia M il verificatore per L. Costruiamo una macchina di Turing M' che riconosce L nel seguente modo: M' enumera le configurazioni finali di M (l'enumerazione non tiene conto del contenuto del nastro dei certificati), e per ogni configurazione finale controlla se è possibile raggiungerla da quella iniziale per qualche certificato. Il punto chiave della dimostrazione consiste nel programmare M' in modo che il controllo di raggiungibilità sia effettuato in spazio $f \cdot f$.

Consideriamo una funzione ragg(I_1, I_2, k) che restituisce 1 sse esiste una computazione di al più 2^k passi che porta M dalla configurazione I_1 alla configurazione I_2. Se $k = 0$ possiamo calcolare il risultato controllando la funzione di transizione di M (chiaramente I_1 ha diverse configurazioni successive possibili, a seconda del contenuto del nastro dei certificati). Se invece $k > 0$, enumeriamo le configurazioni I che usano al più spazio f e controlliamo ricorsivamente se ragg$(I_1, I, k-1)$ e ragg$(I, I_2, k-1)$. Si noti (e questa è l'idea fondamentale della dimostrazione) che la seconda chiamata a ragg() *può riutilizzare lo spazio di lavoro della prima*. Chiaramente M accetta L controllando per ogni configurazione finale se è possibile raggiungerla da quella iniziale in al più $2^{cf(|w|)}$ passi, dove c è una costante opportuna scelta in modo che $2^{cf(n)}$ sia maggiore del numero delle configurazioni di M su input di lunghezza n.

Rimane da mostrare che M' lavora entro spazio $f \cdot f$. Per simulare la ricorsione, M' utilizza un nastro di lavoro come pila, memorizzando le informazioni relative alle varie chiamate a ragg(). Poiché ogni chiamata riduce di uno il valore dell'ultimo argomento, la profondità della ricorsione è $O(f)$, e quindi non più di $O(f)$ chiamate sono attive contemporaneamente. Poiché per ogni chiamata dobbiamo memorizzare I_1, I_2 e k, di dimensione $O(f)$, $O(f)$ e $O(\log f)$, rispettivamente, e un numero constante di celle di servizio, $O(f \cdot f)$ celle sono sufficienti. ∎

Corollario 12 PSPAZIO = NPSPAZIO.

La catena di inclusioni a cui giungiamo è quindi la seguente:

$$\text{LOGSPAZIO} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPAZIO} = \text{NPSPAZIO} \subseteq \text{EXP}.$$

7 Classi probabilistiche

I modelli di calcolo considerati fin qui danno sempre risposte, e le danno sempre corrette. È giunto il momento di prendere in considerazione le classi di complessità relative agli algoritmi che sfruttano sorgenti di bit casuali. Intuitivamente, considereremo macchine che hanno la possibilità, a ogni passo, di lanciare una moneta perfettamente bilanciata, e di decidere lo stato successivo sulla base dell'esito del lancio. In realtà, utilizzeremo una definizione basata sui verificatori:

Definizione 18 Una macchina di Turing probabilistica è un verificatore M che dispone di un terzo stato finale $q_?$ detto di *incertezza*.

L'idea che utilizzeremo nell'introdurre classi probabilistiche è che i certificati contengono l'esito del lancio della moneta. Quindi, nel quantificare con quanta probabilità una macchina di Turing probabilistica si arresta con un certo risultato sarà sufficiente misurare il numero di certificati che danno l'esito richiesto.

Definizione 19 La classe RP è la classe dei linguaggi verificati da macchine di Turing probabilistiche che lavorano in tempo polinomiale, che terminano in esattamente $p(n)$ passi su ogni input di lunghezza n, e tali che per ogni istanza positiva almeno la metà delle stringhe di lunghezza $p(n)$ sono certificati.

Intuitivamente, un linguaggio è in RP se esiste un algoritmo probabilistico che lo riconosce in tempo polinomiale, ma l'algoritmo può sbagliare (o dare esito incerto) con probabilità al più $\frac{1}{2}$ nel caso l'istanza sia positiva. Si noti che nel caso l'istanza sia negativa *l'algoritmo non sbaglia mai*: diciamo che RP è definita tramite errore probabilistico *unilaterale (one-sided)*. Questo fatto ha due importanti conseguenze: innanzitutto, la definizione di RP non è dipendente dalla costante $\frac{1}{2}$ (per iterazione, possiamo infatti ottenere un errore arbitrariamente vicino a 0 partendo da un algoritmo che ha una probabilità di errore arbitrariamente grande ma minore di 1); inoltre, $\text{P} \subseteq \text{RP} \subseteq \text{NP}$, dato che RP è definito ponendo restrizioni sulla densità dei certificati.

Mostriamo ora l'appartenenza a RP del problema COMPOSTO, che è di grande utilità pratica nella ricerca dei numeri primi di grande dimensione. Abbiamo già notato come COMPOSTO possegga certificati verificabili in tempo polinomiale (per esempio, un divisore non banale). Ciononostante, questi certificati sono esponenzialmente sparsi, e quindi non dimostrano l'appartenenza di COMPOSTO a RP. Diamo ora la seguente

Definizione 20 Sia n un numero dispari, e $n-1 = 2^a b$, con b dispari. Dato $t \in \mathbf{Z}_n \setminus \{0,1\}$, consideriamo la sequenza

$$t^b, t^{2b}, t^{2^2 b}, t^{2^3 b}, \ldots, t^{n-1}$$

(cioè gli a quadrati successivi a partire da t^b). Diciamo che t è un *testimone di composizione di* n se $t \not\perp n$, se $t^{n-1} \neq 1$ o se esiste un i per cui $t^{2^i b} \neq \pm 1$ e $t^{2^{i+1} b} = 1$.

In sostanza, un testimone è un elemento di $\mathbf{Z}_n$ che ci mostra che n ha un divisore proprio, che $n-1$ non è un multiplo dell'ordine di $\mathbf{U}_n$ o che esistono radici quadrate dell'unità diverse da ± 1. Come vedremo, in tutti e tre i casi n non può essere primo. Dobbiamo ora mostrare che ci sono abbastanza testimoni, e a questo scopo utilizzeremo un risultato fondamentale di teoria dei numeri:

Teorema 30 (cinese del resto) Se $c \perp d$, allora

$$\mathbf{Z}_{cd} \cong \mathbf{Z}_c \times \mathbf{Z}_d.$$

Utilizzeremo anche alcune nozioni relative agli anelli di resti[27], e in particolare il seguente risultato:

Lemma 2 Sia n un numero composto che non è una potenza di primo, e sia $m = 2^a b$, con b dispari e $a > 0$. Allora, per almeno la metà degli elementi x di $\mathbf{U}_n$ tali che $x^m = 1$, la sequenza

$$x^b, x^{2b}, x^{2^2 b}, x^{2^3 b}, \ldots, x^{2^{a-1} b}, x^m = 1$$

contiene una radice quadrata dell'unità diversa da ± 1.

Dimostrazione. Sia $H = \{x \in \mathbf{U}_n \mid x^m = 1\}$. Vale a dire, H è il *sottogruppo delle radici m-esime dell'unità*. Notiamo innanzitutto che l'esponenziazione (per qualunque esponente fissato) è un endomorfismo[28] di H. Quindi, esiste un $j \in \mathbf{N}$ tale che l'endomorfismo di esponenziazione alla $2^{j+1} b$ ha come immagine l'unità, mentre l'endomorfismo α di esponenziazione alla $2^j b$ ha come immagine un sottogruppo non banale di H interamente costituito da radici quadrate dell'unità. In H esistono certamente altre radici oltre a 1 e -1, perché dato che n è prodotto di almeno due interi coprimi, il teorema cinese del resto garantisce l'esistenza in $\mathbf{U}_n$ di almeno quattro radici quadrate associate agli elementi $\langle 1,1\rangle$, $\langle -1,1\rangle$,$\langle 1,-1\rangle$ e $\langle -1,-1\rangle$ di $\mathbf{Z}_c \times \mathbf{Z}_d$, ove $n = cd$ e $c \perp d$, ed essendo m pari, H contiene tutte le radici quadrate dell'unità. Abbiamo quindi due casi:

[27] In quanto segue, quando x è un elemento di un ben specificato anello di resti $\mathbf{Z}_n$ utilizzeremo le normali notazioni di somma, prodotto e esponenziazione per indicare le corrispondenti operazioni in $\mathbf{Z}_n$, risparmiando la tediosa ripetizione della scrittura "mod n". Questa scelta può provocare qualche ambiguità se utilizziamo x anche come un particolare resto (per esempio, scelto in $\{0,1,\ldots,n-1\}$). La confusione non provoca però danni se nel fare ciò è indifferente quale specifico resto scegliere; per esempio, la scrittura $x \perp n$ ha senso, perché $x \perp n \iff x+kn \perp n$.

[28] Un *endomorfismo* di X è un morfismo da X in X.

1. L'immagine di α contiene solo 1 e una radice $r \neq \pm 1$. In questo caso, tutti gli elementi mappati in r da α generano sequenze contenenti r; essi formano inoltre il solo laterale non banale del nucleo, e quindi hanno cardinalità $|H|/2$, come volevasi dimostrare.

2. L'immagine di α contiene 1 e almeno due radici diverse da ± 1. Anche in questo caso, è immediato mostrare che almeno metà degli elementi di H generano le sequenze desiderate.

Gli altri casi possono essere scartati come segue: se l'immagine di α contiene -1, dato che quest'ultimo è associato (nell'isomorfismo del teorema cinese del resto) alla coppia $\langle -1, -1\rangle$, esistono $x \in \mathbf{Z}_c$ e $y \in \mathbf{Z}_d$ tali che $x^{2^j b} = -1$ (in $\mathbf{Z}_c$) e $y^{2^j b} = -1$ (in $\mathbf{Z}_d$). In tal caso, $\langle x, 1\rangle^{2^j b} = \langle -1, 1\rangle$ e $\langle 1, y\rangle^{2^j b} = \langle 1, -1\rangle$, e quindi almeno due radici diverse da ± 1 devono appartenere all'immagine di α (sono potenze $2^j b$-esime degli elementi corrispondenti alle coppie $\langle x, 1\rangle$ e $\langle 1, y\rangle$). ∎

Teorema 31 (Miller [Mil76]) Se n è primo, non ha testimoni. Se n è un numero dispari composto che non è una potenza di primo, almeno metà dei $t \in \mathbf{Z}_n \setminus \{0\}$ sono testimoni.[29]

Dimostrazione. Si noti innanzitutto che se n è primo tutti i numeri sono coprimi con n, e $n - 1$ è l'ordine del gruppo moltiplicativo di $\mathbf{Z}_n$; quindi, $x^{n-1} = 1$ per ogni $x \in \mathbf{Z}_n \setminus \{0\}$. Inoltre l'equazione $x^2 \equiv 1 \mod n$ ha esattamente le due soluzioni 1 e -1, dato che implica $n \mid (x-1)(x+1)$; quindi non esistono testimoni di n quando n è primo.

È chiaro che per concludere la dimostrazione è sufficiente mostrare che almeno metà degli elementi di $\mathbf{U}_n$ sono testimoni, dato che tutti gli elementi in $\mathbf{Z}_n \setminus (\mathbf{U}_n \cup \{0\})$, non essendo coprimi con n, sono testimoni. Inoltre, poiché gli elementi di $\mathbf{U}_n$ che non sono radici $(n-1)$-esime dell'unità sono pure testimoni, è sufficiente dimostrare che almeno metà delle radici stesse sono testimoni. Ma questo è il contenuto del lemma 2. ∎

Abbiamo a questo punto un semplice algoritmo probabilistico polinomiale con errore unilaterale (detto algoritmo di Miller–Rabin [Rab80]) per stabilire se un numero dato n è composto[30]:

1. se n è pari è composto;

2. altrimenti, per bisezione controlliamo se n è una potenza (basta controllare gli esponenti da 2 a $\log n$; dobbiamo quindi operare $\log n$ ricerche da $\log n$ passi, e ogni passo richiede solo un'esponenziazione), e in caso positivo concludiamo che n è composto;

[29] È in realtà possibile mostrare (al prezzo di una dimostrazione del lemma 2 più complessa e cablata su $n - 1$ anziché su un pari qualunque) che almeno *tre quarti* degli elementi di $\mathbf{Z}_n \setminus \{0\}$ sono testimoni.

[30] Si noti che questo algoritmo è in effetti utilizzato in pratica per *produrre* numeri primi. La sua utilità è legata al fatto che il teorema fondamentale dell'aritmetica stabilisce che il numero di primi minori di n è asintotico a $n/\ln n$, e quindi i primi sono abbastanza densi da rendere computazionalmente ragionevole una ricerca esaustiva o aleatoria.

3. estraiamo in maniera uniforme un numero casuale $t \in \mathbf{Z}_n \setminus \{0\}$;
4. se $t \not\perp n$ concludiamo che n è composto;
5. se $t^{n-1} \neq 1$, concludiamo che n è composto;
6. infine, calcoliamo a e b tali che $n-1 = 2^a b$ e costruiamo la sequenza $t^a, t^{2a}, t^{2^2 a}, \ldots, t^{n-1}$. Se nella sequenza compare una radice dell'unità diversa da ± 1 concludiamo che n è composto;
7. concludiamo che n è primo.

Tutti i conti coinvolti sono banalmente eseguibili in tempo polinomiale (peraltro, con esponente basso). Per il teorema precedente, possiamo arrivare all'ultimo passo con n composto con probabilità al più $\frac{1}{2}$. Quindi ripetendo l'algoritmo k volte con esito negativo la possibilità di errore nello stabilire che n è primo è al più 2^{-k}.

Miller ha anche dimostrato in [Mil76] che se vale l'ipotesi generalizzata di Riemann esistono certamente certificati piccoli, e in tal caso è possibile trovarli in tempo quartico (nella lunghezza di n). Esiste quindi una versione deterministica polinomiale dell'algoritmo di Miller–Rabin, ma non siamo al momento in grado di dimostrare che dia risultati corretti.

8 Cenni di crittografia

Lo scenario crittografico più comune consiste in due agenti, Alice e Bubi, che vogliono scambiarsi un messaggio M lungo un canale di comunicazione insicuro. Mario è un malevolo *voyeur* che ha la possibilità di leggere ed eventualmente modificare le comunicazioni tra Alice e Bubi. Ci sono diverse cose che Alice e Bubi vogliono fare:

- Alice vuole trasmettere dei messaggi a Bubi in maniera che Mario non possa comprenderne il contenuto (*riservatezza*);
- Alice vuole trasmettere dei messaggi a Bubi in maniera che Bubi sia sicuro che Alice sia effettivamente il mittente (*autenticità*);
- Alice vuole trasmettere dei messaggi a Bubi in maniera che Bubi sia sicuro che il messaggio sia pervenuto completo e non modificato (*integrità*);
- Bubi vuole avere garanzie sul fatto che Alice non possa negare di avere inviato un dato messaggio (*non ripudiabilità*).

Il compito affrontato dalla crittografia classica è il primo: Alice e Bubi si mettono d'accordo prima della comunicazione su due funzioni calcolabili in tempo polinomiale C e D tali che per ogni messaggio M

$$D(C(M)) = M.$$

La funzione di *crittazione* C trasforma un elemento dello spazio dei *testi in chiaro* in un elemento dello spazio dei *testi crittati*, mentre la funzione di *decrittazione* D opera in senso inverso. Un sistema crittografico è dato dalla coppia di trasformazioni suddette: quando Alice vuole inviare un messaggio a Bubi, invia $C(M)$ al posto di M. Ovviamente, C va scelta in modo che sia (in un qualche senso che andiamo a rendere formale) difficile recuperare M a partire da $C(M)$.

Sebbene in generale non siamo in grado di dimostrare la sicurezza di un sistema crittografico, ne esiste uno la cui sicurezza è stata dimostrata da Shannon utilizzando strumenti di teoria dell'informazione [Sha49]. Supponiamo che Alice, che deve mandare un messaggio di n bit a Bubi, si sia prima accordata con Bubi su una stringa di n bit scelta a caso uniformemente. Al momento di trasmettere il messaggio, Alice effettua l'"o" esclusivo bit a bit del messaggio e della stringa (con lo stesso meccanismo Bubi può recuperare il messaggio originale). Se la chiave è scelta uniformemente, il messaggio crittato (qualunque sia la distribuzione dei testi in chiaro) risulta avere distribuzione uniforme, e quindi Mario non ha alcun modo di recuperare informazioni (in effetti, neppure un bit) del messaggio originale. Formalmente,

$$\Pr\{M = x \mid C(M) = y\} = \Pr\{M = x\},$$

e quindi, in parole povere, conoscere o meno $C(M)$ non è di alcun aiuto nel derivare M.

Questo metodo è però di difficile utilizzo, perché presume lo scambio di una grande quantità di informazioni in maniera sicura prima della comunicazione (la chiave, infatti, può venire usata *una sola volta*). Inoltre, la generazione di chiavi lunghe veramente casuali è un problema non banale.

Definiamo quindi una nozione formale di trasformazione difficilmente invertibile:

Definizione 21 Una funzione $f : 2^* \to 2^*$ è detta *a senso unico (one-way)* se mappa stringhe di lunghezza n in stringhe di lunghezza $l(n)$ per qualche funzione $l : \mathbf{N} \to \mathbf{N}$, è calcolabile in tempo polinomiale, e per qualunque macchina di Turing probabilistica che lavori in tempo polinomiale si ha che $\Pr\{f(w) = f(M(f(w)))\}$ è *trascurabile*, vale a dire di ordine di grandezza inferiore a qualunque funzione della forma $1/|w|^k$.

Informalmente, una funzione è a senso unico se è calcolabile in tempo polinomiale, ma la sua inversa non è quasi mai calcolabile in tempo polinomiale. Si noti che la probabilità è presa sia rispetto ai certificati (e cioè alla sorgente di bit casuali utilizzata dalla macchina), sia rispetto all'input, scelto con probabilità uniforme.

Le funzioni a senso unico sono fondamentali per la generazione di sistemi crittografici. Tuttavia, dimostrare la loro esistenza (e ancor più costruirne qualcuna esplicitamente) rimane per il momento al di fuori della nostra portata. Possiamo però dimostrare quanto segue:

Teorema 32 Se RP = NP non esistono funzioni a senso unico iniettive.

Dimostrazione. Sia f una funzione calcolabile in tempo polinomiale, e consideriamo il seguente problema di decisione: data una stringa $z \in 2^*$ e un intero $k \in \mathbf{N}$, esiste $w \in 2^*$ con bit k uguale a 1 e tale che $f(w) = z$? In altre parole: l'unico $w \in 2^*$ tale che $f(w) = z$ ha bit k uguale a 1? Tale problema è chiaramente in NP: il verificatore si aspetta w come certificato, controlla che $f(w) = z$ e che il bit k di w sia 1, e solo in tal caso accetta.

Se questo problema fosse in RP, potremmo invertire f addirittura con probabilità di successo *costante*. Infatti, dato z possiamo indovinare i bit dell'unico x tale che $f(x) = z$ usando l'algoritmo probabilistico che dimostra l'appartenenza a RP; se per ogni bit effettuiamo $|z| \geq n$ iterazioni, e a ogni strazione controlliamo che la stringa ottenuta sia proprio x, la probabilità di ottenere un bit errato, e *a fortiori* quella di ottenere un input errato, è minore di $n/2^{|z|} \leq n/2^n \leq 1/2$. ∎

Corollario 13 Se P = NP non esistono funzioni a senso unico iniettive.

Le tecniche attualmente utilizzate per costruire funzioni sperabilmente a senso unico sono in parte legate alle ricerche iniziali di Shannon sulla crittografia [Sha49], e fanno uso di *diffusione* (un singolo bit dell'input deve influenzare tutti i bit dell'output) e *confusione* (la distribuzione statistica dei testi in chiaro trasmessi non deve trasparire nei testi crittati), come nel caso del DES (Data Encryption Standard) e di IDEA (International Data Encryption Algorithm). Recentemente, specie per la crittografia a chiave pubblica, ci si è affidati a opportune funzioni esponenziali su $\mathbf{Z}_n$.

Nel nostro caso, la funzione f mappa la coppia data da $\langle C, M\rangle$ (dove per esempio C è descritta algoritmicamente) su $C(M)$. Dato che Bubi sa in partenza che uno degli input è C, può utilizzare D per ottenere M, mentre Mario deve necessariamente invertire f. Si dice in questo caso che Bubi dispone di una *scorciatoia (trapdoor)* per il calcolo di f^{-1}, cioè di un frammento di informazione non disponibile pubblicamente (vale a dire D) che permette di invertire f con facilità (ovviamente, solo se la funzione di crittazione è C!). Chiaramente siamo interessati solo a funzioni a senso unico di questo tipo, altrimenti Bubi avrebbe ben poche possibilità di recuperare M.

In genere, è poco pratico predisporre funzioni C ed D per ogni coppia di agenti comunicanti. Le funzoni C e D vengono invece parametrizzate da una *chiave* K, e vengono rese pubbliche. La nostra funzione a senso unico mappa ora $\langle K, M\rangle$ su $C_K(M)$. La chiave K è naturalmente nota solo ad Alice e Bubi.

Occorrono invece strumenti più sofisticati per ottenere sistemi crittografici a *chiave pubblica*, proposti da Diffie e Hellman [DH76], in cui esistono un algoritmo di crittazione e uno di decrittazione noti, una chiave privata di decrittazione, e *una chiave pubblica di crittazione nota a tutti*. Ovviamente, è necessario che sia difficile inferire la chiave privata a partire da quella pubblica.

8.1 Il sistema a chiave pubblica RSA

Il sistema di crittografia a chiave pubblica RSA (dal nome degli autori Ronald Rivest, Adi Shamir e Leonard Adleman [RSA78]) utilizza alcune semplici proprietà dell'aritmetica di $\mathbf{Z}_n$ per offrire servizi di crittografia, di firma digitale, di autenticazione e di denaro digitale anonimo.

Consideriamo due numeri primi p e q, un numero[31] $e \perp \varphi(pq)$ e l'inverso d di e modulo $\varphi(pq)$ (vale a dire, $ed \equiv 1 \mod \varphi(pq)$).[32] La chiave pubblica è data dalla coppia $\langle e, pq\rangle$, mentre la chiave privata è data dalla coppia $\langle d, pq\rangle$.

Crittografia. Lo spazio dei messaggi in chiaro e quello dei messaggi cifrati è lo stesso, e cioè $\mathbf{U}_{pq}$. Quando Alice vuole inviare un messaggio $x \in \mathbf{U}_{pq}$ a Bubi, lo trasforma in x^e. Quando Bubi riceve x^e, lo eleva alla d, ottenendo così $(x^e)^d = x^{ed} = x^{1+k\varphi(pq)} = x$.

La sicurezza del sistema RSA si basa sull'assunzione che sia difficile ricavare x da x^e, e che sia difficile ricavare d da e e pq. Entrambe le operazioni sarebbero semplici se fosse possibile ricavare p e q da pq (o anche semplicemente $\varphi(pq)$ da pq), ma non conosciamo alcun algoritmo polinomiale in grado di fare ciò.

Notiamo che la funzione che assumiamo essere a senso unico è la seguente:

$$\langle x, e, p, c(p), q, c(q)\rangle \mapsto \langle x^e, e, pq\rangle,$$

Ove $c(-)$ è un certificato di primalità verificabile in tempo polinomiale (per esempio, quelli del teorema di Pratt [BC98]). Se p o q non sono certificati correttamente, la funzione è indefinita[33].

La funzione descritta qui sopra è "storicamente" a senso unico, in quanto vent'anni di crittoanalisi non hanno mostrato come violarla. Ciononostante, negli ultimi anni sono stati ottenuti molti risultati importanti sugli algoritmi di fattorizzazione, e non è detto che non si possa giungere a un algoritmo polinomiale per fattorizzare pq.

Firma digitale. La possibilità di effetture firme digitali è insita nel fatto che l'algoritmo di crittazione e di decrittazione dell'RSA è lo stesso. La chiave pubblica e quella privata svolgono un ruolo perfettamente simmetrico, e quindi Alice, per firmare un messaggio M, trasmette $\langle M, M^{d_A}\rangle$. Bubi riceve il messaggio $\langle M, N\rangle$ e controlla se $N^{e_A} = M$, il che è banalmente vero

[31] Sarebbe più naturale utilizzare c per la chiave pubblica, che viene usata principalmente per crittare, ma in tutta la letteratura sull'argomento viene utilizzata costantemente la lettera e, in quanto iniziale di *encrypting*; peraltro le due chiavi dell'RSA sono perfettamente simmetriche e, come vedremo, nel caso della firma digitale il loro ruolo viene addirittura scambiato.

[32] Ricordiamo che il numero $\varphi(n)$ (detto *indicatore di Eulero)*, l'ordine del gruppo $\mathbf{U}_n$ delle unità di $\mathbf{Z}_n$, è un multiplo dell'ordine di qualunque elemento x di $\mathbf{U}_n$; vale cioè $x^{\varphi(n)} = 1$.

[33] Questo apparentemente bizzarro espediente tecnico serve a evitare che la funzione sia facilmente invertibile nel caso in cui p o q non siano primi, e quindi la probabilità di inversione sia alta a causa di input a cui non siamo interessati.

se $N = M^{d_A}$. In generale non vi è alcun modo con cui Mario possa creare M^{d_A} senza conoscere d_A.

Va notato che se sia Alice che Bubi dispongono delle rispettive chiavi pubbliche e_B e e_A, Alice può addirittura inviare $M^{d_A e_B}$, che è contemporaneamente crittato e firmato. Bubi eleva il messaggio in arrivo all'esponente $e_A d_B$, e ottiene il messaggio originale (o un messaggio improbabile nel caso il mittente non conosca d_A).

Ci sono però due problemi distinti con tale schema. Il primo è che dobbiamo raddoppiare la lunghezza di ogni messaggio. Il secondo, più sottile, è dovuto a una proprietà dell'RSA che può essere utilizzata positivamente per implementare il denaro digitale anonimo (cfr. prossimo paragrafo), ma che in questo caso risulta pericolosa.

Supponiamo che Bubi voglia fare firmare ad Alice un messaggio pericoloso M. Dato che Alice si rifiuta di firmare M, Bubi può cercare di trovare un opportuno $r \in \mathbf{U}_{pq}$ e sottoporre ad Alice un messaggio Mr^{e_A} dall'apparenza innocua. Alice firma il messaggio, ma così facendo fornisce a Bubi $(Mr^{e_A})^{d_A} = M^{d_A} r$, da cui Bubi può facilmente ricavare una versione firmata da Alice del messaggio M.

Per ovviare a entrambi gli inconvenienti, possiamo utilizzare delle funzioni facilmente calcolabili che riassumono un messaggio lungo in uno più corto, e firmare il riassunto anziché il messaggio. È però fondamentale garantire che sia difficile trovare due messaggi con lo stesso riassunto, e per fare ciò basta chiedere che la funzione riassunto sia a senso unico. Infatti la condizione descritta nella definizione 21 dice esattamente, nel caso che f non sia suriettiva, che è difficile in generale trovare *un qualunque input* che dia un output prefissato.

Costruire funzioni riassunto a senso unico (dette anche funzioni *hash*) è chiaramente ancora più difficile che costruire funzioni a senso unico biiettive. Esempi di funzioni utilizzate per riassumere i messaggi sono MD5 (Manipulation Detection 5) e SHA (Secure Hash Code).

Denaro digitale anonimo. Consideriamo un messaggio M contenente una transazione commerciale tra Alice e Bubi. Alice vuole fare firmare digitalmente alla propria banca M prima di inviarlo a Bubi. La chiave pubblica della banca è $\langle e, pq \rangle$, mentre quella privata è $\langle d, pq \rangle$. Se M può essere rivelato alla banca, Alice deve solo offrire M, ottenere M^d dalla banca e inviarlo unitamente a M a Bubi. Bubi può a questo punto fare valere (ad esempio in sede giudiziaria) la validità della transazione in M, dato che ha la firma M^d della banca.

Il denaro fisico ha però una caratteristica fondamentale (specie per chi fa acquisti imbarazzanti): è *anonimo*. Una volta che viene ritirato dalla banca, non c'è modo di sapere come viene speso. Una caratteristica aritmetica dell'RSA può essere utilizzata per gestire denaro digitale in maniera anonima.

Alice, prima di chiedere la firma della banca, *oscura* M tramite un numero casuale $r \in \mathbf{U}_{pq}$: alla banca non viene inviato M, bensì Mr^e (e, data la scelta casuale di r, ci aspettiamo che la distribuzione di Mr^e sia approssimativamente uniforme). La banca firma Mr^e, restituendo ad

Alice $(Mr^e)^d = M^d r$. Alice divide il risultato per r, e ottiene M^d (cioè la firma della banca su M) senza che la banca conosca nessun bit di M.

Dimostrazioni a conoscenza zero. Supponiamo che la chiave pubblica $\langle e, pq\rangle$ di Alice sia nota a Bubi. Descriviamo un protocollo interattivo tramite il quale Alice può dimostrare la propria identità a Bubi, mostrando di conoscere la chiave privata $\langle d, pq\rangle$, senza però produrre neppure un bit di d.

Il protocollo si svolge in due fasi: nella prima, Bubi estrae un numero casuale $r \in \mathbf{U}_{pq}$ e sfida Alice a decrittare r^e. Nella seconda, Alice, che possiede la chiave d, calcola $(r^e)^d = r$ e lo invia a Bubi, che controlla che il numero inviato da Alice sia uguale a quello estratto. Mario non ha la possibilità di impersonare Alice, perché non ha modo di trovare d a partire dalla chiave pubblica. Nella comunicazione tra Alice e Bubi sono stati trasmessi r, un numero casuale, e la sua potenza e-esima. Essendo e pubblico, nessun bit di informazione su d è stato rivelato.[34]

Questo tipo di protocollo interattivo è detto *dimostrazione a conoscenza zero*. In generale, in una dimostrazione a conoscenza zero il dimostratore (in questo caso Alice, in generale una macchina di Turing che lavora in tempo esponenziale) cerca di convincere un verificatore (in questo caso Bubi, in generale una macchina di Turing probabilistica che lavora in tempo polinomiale) del fatto che un input x visibile a entrambi sia in un linguaggio L. Il protocollo soggiace alle seguenti condizioni: se $x \in L$, esiste un dimostratore riesce a convincere il verificatore con probabilità almeno $1 - 2^{-|x|}$; altrimenti, *nessun* dimostratore può convincere il verificatore con probabilità maggiore di $2^{-|x|}$. Uno dei risultati più interessanti ottenuti tramite assunzioni crittografiche (l'esistenza di funzioni a senso unico) è che esistono dimostrazioni a conoscenza zero per tutti i problemi in NP. Ad esempio, è possibile convincere un verificatore del fatto che una formula proposizionale è soddisfacibile senza fare filtrare neppure il valore di una variabile in un assegnamento che soddisfa la formula.

Come violare l'RSA in presenza di modulo fisso. Consideriamo il seguente scenario. Un'organizzazione governativa decide di fornire un sistema nazionale di crittografia basato sull'RSA. Vengono scelti due numeri primi p e q, e l'intero sistema utilizza sempre il modulo pq. A diverse persone, però, vengono assegnati diverse chiavi d ed e. Ci sono diversi vantaggi in questo schema: innanzitutto, è possibile creare dei microprocessori (con pq cablato) che consentono di effettuare operazioni crittografiche ad alta velocità, per cui è possibile scegliere p e q molto grandi. In secondo luogo, la polizia ha la possibilità, in caso di necessità, di intercettare messaggi crittati: infatti, l'organizzazione possiede $\varphi(pq)$, e può quindi sempre calcolare la chiave privata corrispondente a una pubblica.

Vedremo però che questo scenario è completamente illusorio. La ragione fondamentale è che

[34] Questo protocollo è quello effettivamente utilizzato per le connessioni remote con `ssh`.

conoscendo d, e e pq e disponendo di una sorgente di bit casuali *è possibile fattorizzare pq in tempo polinomiale con probabilità di insuccesso limitata*.

Ricordiamo che $de \equiv 1 \mod \varphi(pq)$. Quindi, possiamo ottenere un multiplo $m = ed + 1$ di $\varphi(pq)$ a partire da e e d. Scriviamo $m = 2^a b$ con b dispari. Per il lemma 2, in almeno metà dei casi la sequenza

$$t^b, t^{2b}, t^{2^2 b}, \ldots, t^m,$$

con $t \in \mathbf{U}_{pq}$, contiene una radice dell'unità diversa da ± 1 (si noti che $t^m = t^{\varphi(pq)} = 1$). Una tale radice ci consente di fattorizzare pq, dato che se $x^2 \equiv 1 \mod pq$, allora $(x-1)(x+1)$ è un multiplo di pq. Quindi $p \mid x-1$ e $q \mid x+1$ o viceversa (dato che $x \neq \pm 1$), e per fattorizzare pq è sufficiente calcolare $\mathrm{MCD}(pq, x-1)$ o $\mathrm{MCD}(pq, x+1)$. La possibilità che t non abbia questa proprietà alla k-esima estrazione va a zero come 2^{-k}. Possiamo quindi estrarre a caso elementi di $\mathbf{U}_{pq}$ e costruire la sequenza suddetta fino a ottenere la fattorizzazione desiderata (il numero atteso di estrazioni prima di avere successo è al più 2).

Autenticazione e scambio di chiavi mediante terze parti. Negli scenari che abbiamo prefigurato, le parti in causa posseggono chiavi pubbliche che assumono essere corrette. Nel caso però Alice e Bubi non siano stati in grado di comunicare precedentemente, e che Mario abbia la possibilità di modificare i messaggi, non c'è modo di distribuire le chiavi pubbliche in modo sicuro.

Infatti, un approccio *naif* al problema sarebbe il seguente: Alice manda a Bubi $\langle e_A, p_A q_A \rangle$ in chiaro, e Bubi fa altrettanto. Ora Alice può inviare con sicurezza messaggi a Bubi, e viceversa. Dato che le chiavi sono pubbliche, il fatto di trasmetterle in chiaro non causa problemi.

La sicurezza, in questo caso, è completamente illusoria: Mario può intercettare la chiave di Alice e fornire a Bubi una chiave $\langle e'_A, p_A q_A \rangle$ di cui possiede la corrispondente d'_A. Analogamente avviene per la chiave di Bubi. Se Bubi invia un messaggio $M^{e'_A}$ ad Alice, Mario lo intercetta, lo decritta tramite d'_A e lo riinvia ad Alice crittato con e_A. Alice e Bubi, quindi, stanno apparentemente comunicando messaggi crittati, ma la comunicazione, per Mario, è in chiaro.

Per evitare questo tipo di attacchi, è necessario che perlomeno Alice possa ottenere la chiave di Bubi da una terza parte T di cui già conosce la chiave pubblica $\langle e_T, p_T q_T \rangle$. Quando Alice cerca di comunicare con Bubi, Bubi indica in T la terza parte che conosce la sua chiave pubblica. T fornisce ad Alice la chiave pubblica di Bubi *firmata con la propria chiave* $\langle d_T, p_T q_T \rangle$, e quindi Alice è in grado di verificare la veridicità dell'informazione. A questo punto Alice invia la propria chiave pubblica in forma crittata a Bubi.

Anche in questo caso stiamo assumendo una conoscenza *a priori* di Alice (la chiave pubblica di T). Ciononostante, questa conoscenza è *fissa e indipendente da Bubi*, purché Bubi abbia fornito a T in maniera sicura la propria chiave pubblica. Nel caso Alice e Bubi siano "simmetrici" questo non porta a grandi vantaggi, ma nel caso del commercio elettronico è ragionevole pensare che ogni rivenditore si registri, una volta per tutte all'inizio della sua attività, con un'autorità

che provvederà a distribuire una versione firmata della sua chiave pubblica. Le autorità possono essere mantenute in numero limitato, e quindi Alice parte con una conoscenza *a priori* di gran lunga inferiore all'insieme delle chiavi pubbliche di tutti i rivenditori.

8.2 Lo schema di ElGamal

Per completezza, mostramo in breve un altro sistema di crittografia a chiave pubblica, proposto da ElGamal [ElG85], che utilizza un primo p e un elemento primitivo (cioè un generatore) a di $\mathbf{U}_p$. Alice sceglie un intero x e pubblica la chiave a^x (p e a sono comuni). Se Bubi vuole mandare un messaggio $M \in \mathbf{U}_p$ ad Alice, sceglie un intero casuale $r < p$ e invia $\langle a^r, Ma^{xr}\rangle$. Alice calcola $(a^r)^x = a^{rx}$ elevando a x il primo elemento, e può a questo punto calcolare M con una semplice divisione. Per fare ciò, è necessario conoscere x, e ricavare x da a^x (il *problema del logaritmo discreto*) è reputato un problema difficile (in effetti, più difficile della fattorizzazione). Lo schema di ElGamal consente anche la firma digitale (ma con una procedura più complessa dell'RSA), e in effetti lo standard statunitense per la firma digitale ne prevede l'uso.

8.3 Il poker mentale

Il poker mentale è un classico "problema impossibile"[35] risolubile con tecniche crittografiche. Alice e Bubi vogliono giocare a poker per telefono. Per farlo, devono interagire in modo da scegliere cinque carte a testa da un mazzo di 52 carte. Le condizioni sono ovvie: le carte devono essere scelte in modo equiprobabile, le carte di Alice devono essere diverse da quelle di Bubi e le carte di un giocatore non devono essere note all'altro fino a che il primo non le annunci. Chiaramente, sia Alice che Bubi devono poter controllare che l'altro non bari.

Alice e Bubi si accordano su un primo p e ciascuno dei due genera privatamente due elementi $e, d \in \mathbf{Z}_{p-1}$ tali che $ed = 1$. Quindi si accordano su quali elementi $x_1, x_2, \ldots, x_{52}$ di $\mathbf{U}_p$ rappresentino le carte. Alice distribuisce le carte fornendo a Bubi una permutazione casuale di $x_1^{e_A}, x_2^{e_A}, \ldots, x_{52}^{e_A}$. Bubi sceglie cinque carte per Alice e gliele indica, ma chiaramente non sa quali carte ha scelto (la difficoltà è essenzialmente la stessa che troviamo nell'RSA). Alice può decrittarle, e quindi conosce le proprie carte. Ora Bubi permuta casualmente le carte crittate rimaste, le critta a sua volta e le invia ad Alice, che riceve quindi 47 numeri della forma $x_i^{e_A e_B}$. Alice decritta le carte con la propra chiave (ma le carte rimangono crittate dalla chiave di Bubi, dato che $(x_i^{e_A e_B})^{d_A} = x_i^{e_A d_A e_B} = x_i^{e_B}$), ottenendo così numeri della forma $x_i^{e_B}$ permutati casualmente, sceglie cinque carte e le invia a Bubi, che le decritta con la propria chiave. Alla fine della partita, Bubi e Alice annunciano le proprie chiavi, e quindi ciascuno dei due può control-

[35] In effetti, la crittografia consiste essenzialmente nel fare cose impossibili utilizzando strumenti di cui nessuno ha mai dimostrato l'esistenza.

lare che le carte utilizzate nella partita siano quelle estratte. I criteri menzionati all'inizio sono ovviamente soddisfatti.

Notazione

$X \subseteq Y$ L'insieme X è contenuto nell'insieme Y, ovvero per ogni $x \in X$ abbiamo anche $x \in Y$.

$X \subset Y$ L'insieme X è contenuto *propriamente* nell'insieme Y, ovvero $X \subseteq Y$ e $X \neq Y$.

$\mathbf{N}$ L'insieme del numeri naturali $\{0, 1, 2, \ldots\}$.

n Il numero naturale n, ovvero l'insieme $\{0, 1, \ldots, n-1\}$.

$\mathbf{Z}$ L'insieme del numeri interi.

$\mathbf{R}$ L'insieme del numeri reali.

$\perp$ Simbolo convenzionale per indicare la divergenza: $f(x) = \perp$ significa che f non è definita su x; si assume che $f(\perp) = \perp$.

X^n Potenza n-esima di X, vale a dire l'insieme delle n-uple ordinate di elementi di X.

X^* Chiusura di Kleene di X, vale a dire $\bigcup_{n \in \mathbf{N}} X^n$; equivalentemente, il monoide libero su X. Gli elementi di X^* sono detti *stringhe su* X.

ε La stringa vuota.

(*formula*) Notazione di Iverson: ha valore 0 se la formula è falsa, 1 se è vera [GKP94].

χ_A La funzione caratteristica dell'insieme A, vale a dire $\chi_A(x) = (x \in A)$.

Y^X L'insieme delle funzioni da X in Y.

2^X L'insieme delle parti di X (equivalentemente, l'insieme delle funzioni da X nell'insieme $2 = \{0, 1\}$).

$\operatorname{dom} R$ Dominio di R.

$\operatorname{cod} R$ Codominio di R.

$\operatorname{ran} R$ Rango di R, cioè $\{x \in \operatorname{dom} R \mid \exists y \in \operatorname{cod} R \; x \; R \; y\}$.

$\operatorname{imm} R$ Immagine di R, cioè $\{y \in \operatorname{cod} R \mid \exists x \in \operatorname{dom} R \; x \; R \; y\}$.

$\pi_i^{(n)}$ Proiezione i-esima a n argomenti: $\pi_i^{(n)}(x_1, x_2, \ldots, x_n) = x_i$.

$\varphi_z^{(n)}$ Funzione parziale ricorsiva con n argomenti e di indice z.

φ_z Funzione parziale ricorsiva con un argomento e di indice z.

$\varphi_u^{(2)}$ La funzione parziale universale: $\varphi_u^{(2)}(x, y) = \varphi_x(y)$.

Halt L'insieme di arresto $\{\langle\langle x, y\rangle\rangle \in \mathbf{N} \mid \varphi_u^{(2)}(x, y) \neq \perp\}$.

K La diagonalizzazione dell'insieme d'arresto $\{\langle x, x\rangle \in \mathbf{N} \mid \varphi_u^{(2)}(x, x) \neq \perp\}$.

$f \leq g$ Ordinamento per punti delle funzioni: $f \leq g$ sse per ogni $x \in \operatorname{dom} f = \operatorname{dom} g$ abbiamo $f(x) \leq g(x)$ (ovviamente, $\operatorname{cod} f = \operatorname{cod} g$ deve essere ordinato perché la cosa abbia senso).

$O(f)$ L'insieme $\{g \mid \exists\alpha\ g(x) \leq \alpha f(n)$ definitivamente$\}$.

$\Omega(f)$ L'insieme $\{g \mid \exists\alpha\ g(x) \geq \alpha f(n)$ definitivamente$\}$.

$\Theta(f)$ L'insieme $\{g \mid \exists\alpha, \beta\ \alpha f(x) \leq g(x) \leq \beta f(n)$ definitivamente$\}$.

TEMPO (f) La classe dei problemi risolubili in tempo f.

SPAZIO (f) La classe dei problemi risolubili in spazio f.

NTEMPO (f) La classe dei problemi con certificati verificabili in tempo f.

NSPAZIO (f) La classe dei problemi con certificati verificabili in spazio f.

P La classe dei problemi risolubili in tempo polinomiale.

NP La classe dei problemi con certificati verificabili in tempo polinomiale.

PSPAZIO La classe dei problemi risolubili in spazio polinomiale.

LOGSPAZIO La classe dei problemi risolubili in spazio logaritmico.

EXP La classe dei problemi risolubili in tempo esponenziale.

RP La classe dei problemi risolubili probabilisticamente in tempo polinomiale con errore limitato unilaterale.

$k \mid n$ k divide n (cioè $n = ak$ per qualche intero a).

$m \perp n$ m e n sono coprimi [GKP94].

$\mathbf{Z}_n$ Il gruppo dei resti modulo n (cioè classi di equivalenza di interi rispetto alla relazione di avere lo stesso resto nella divisione per n).

$\mathbf{U}_n$ Il gruppo delle unità di $\mathbf{Z}_n$ (cioè le classi associate a interi coprimi con n).

$\varphi(n)$ L'indicatore di Eulero (la cardinalità di $\mathbf{U}_n$).

Riferimenti bibliografici

[BC98] Anna Bernasconi e Bruno Codenotti. *Introduzione alla complessità computazionale*. Springer–Verlag, 1998.

[Can74] Georg Cantor. Über eine Eigenschaft des Inbegriffs aller reelen algebraischen Zahlen. *J. Math.*, 77:258–262, 1874.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *Amer. J. Math.*, 58:345–363, 1936.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pag. 151–158, 1971.

[DH76] Whitfield Diffie e Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, November 1976.

[ElG85] Taher ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, IT-31(4):469–472, 1985.

[GKP94] Ronald L. Graham, Donald E. Knuth, e Oren Patashnik. *Concrete Mathematics*. Addison–Wesley, second edizione, 1994.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandten Systeme I. *Monatsch. Math. und Phys.*, 38:173–198, 1931.

[HS65] Juri Hartmanis e Richard E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.

[Kle36] Stephen C. Kleene. General recursive functions of natural numbers. *Math. Ann.*, 112:727–742, 1936.

[Kle38] Stephen C. Kleene. On notations for ordinal numbers. *J. Symb. Log.*, 3:150–155, 1938.

[Lev73] Leonid A. Levin. Universal'nyĭe perebornyĭe zadach (Universal search problems). *Problemy Peredachi Informatsii*, 3(9):265–266, 1973.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. System Sci.*, 13:300–317, 1976.

[Odi89] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 della collana *Studies in Logic and the Foundations of Mathematics*. North–Holland, 1989.

[Rab80] Michael O. Rabin. A probabilistic algorithm for testing primality. *J. Number Theory*, 12, 1980.

[Ric53] H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Am. Math. Soc.*, 74:358–366, 1953.

[RSA78] Ronald L. Rivest, Avi Shamir, e Leonard Adleman. A method for obtaining digital signatures and public-key cryptosytems. *CACM*, 21(2):120–126, 1978.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.*, 4(2):177–192, 1970.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Tech. J.*, 28(4), 1949.

[Tur36] Alan M. Turing. On computable numbers, with an application to the *Entscheidungsproblem*. *Proc. London Math. Soc. (3)*, 42:230–265, 1936.

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material,

which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This Licence

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http:///www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.