Problemi che i calcolatori non possono risolvere

È importante sapere se un programma è corretto, cioè fa quello che ci aspettiamo.

È facile vedere che il programma

```
main()
{
        printf(‘‘hello, world\n’’);
}
```

stampa `hello, world`.

Ma il programma

```
main()
{
 int n, total, x, y, z;
 scanf(''%d'', &n);
 total = 3;
 while(1) {
   for (x=1; x<= total-2; x++)
     for (y=1; y<= total-x-1; y++) {
         z = total - x - y;
         if(exp(x,n) + exp(y,n) == exp(z,n))
                 printf(``hello, world\n'');
     }
 total++;
 }
}
```

Stampa `hello, world`, dato un input n se e solo se l'equazione

$$x^n + y^n = z^n$$

ha una soluzione dove x, y e z sono interi.

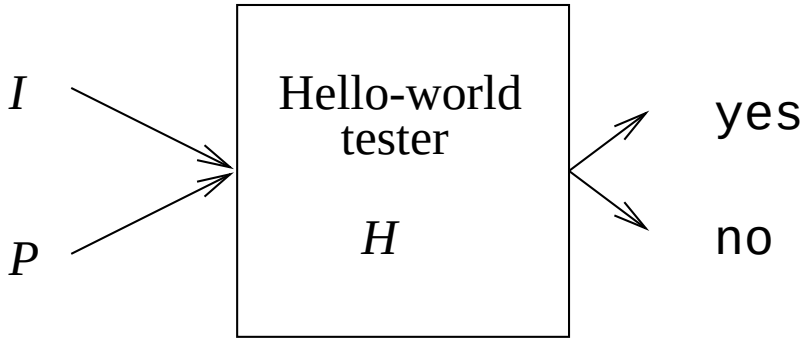
Sappiamo ora che stampa `hello, world` con l'input $n = 2$, e cicla per sempre su input $n > 2$ (Fermat's Last Theorem).

Ci sono voluti 300 anni per provarlo.

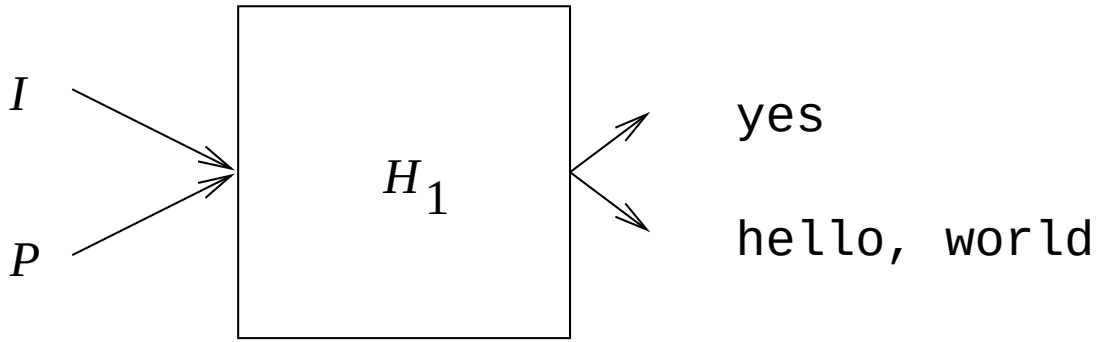
Possiamo sperare di avere un programma che prova la correttezza di programmi?

L'ipotetico programma H che testa "hello, world"

Supponiamo che H esista.

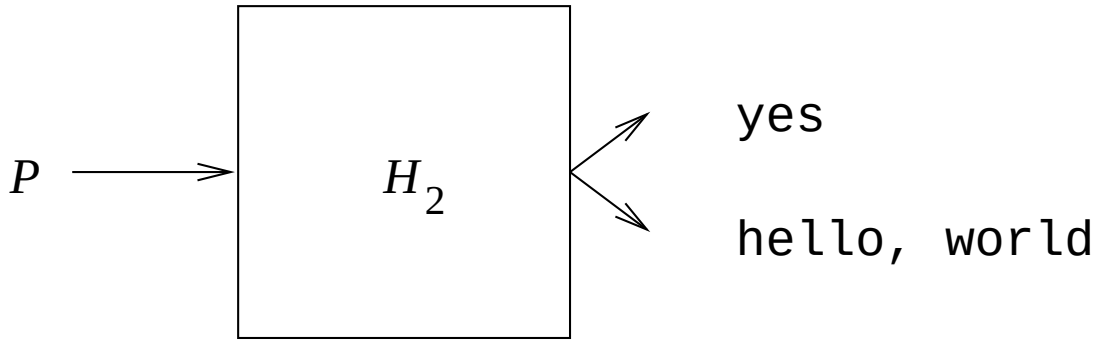


Modifichiamo il comando di stampa di `no` di H in `hello, world`. Otteniamo il programma H_1

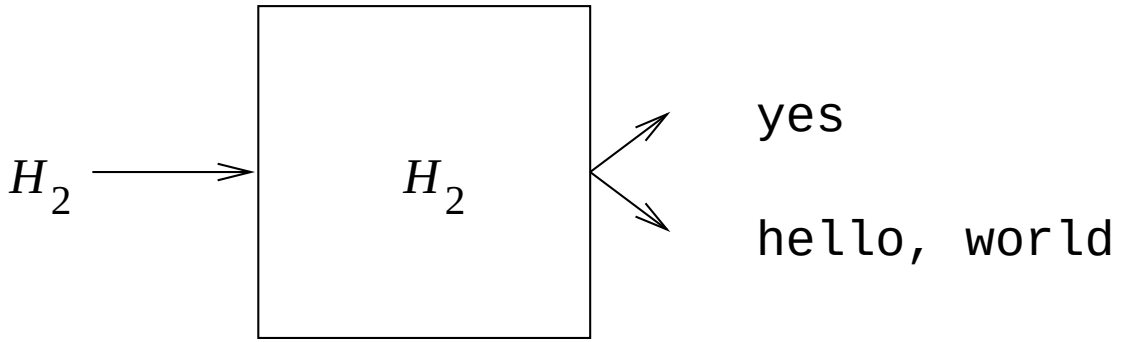


Modifichiamo H_1 in modo che prenda P ed I come un singolo input. Questo significa che P prende se stesso come input.

Otteniamo il programma H_2:



Diamo H_2 come input ad H_2.



Se H_2 stampa `yes`, avrebbe dovuto stampare `hello, world`.

Se H_2 stampa `hello, world`, avrebbe dovuto stampare `yes`.

Quindi H_2 non può esistere.

Quindi neanche H può esistere.

Quindi il problema affrontato è *indecidibile*.

Problemi indecidibili

Problemi per cui non c'è nessun programma che li possa risolvere.

Problema: appartenenza di una stringa ad un linguaggio.

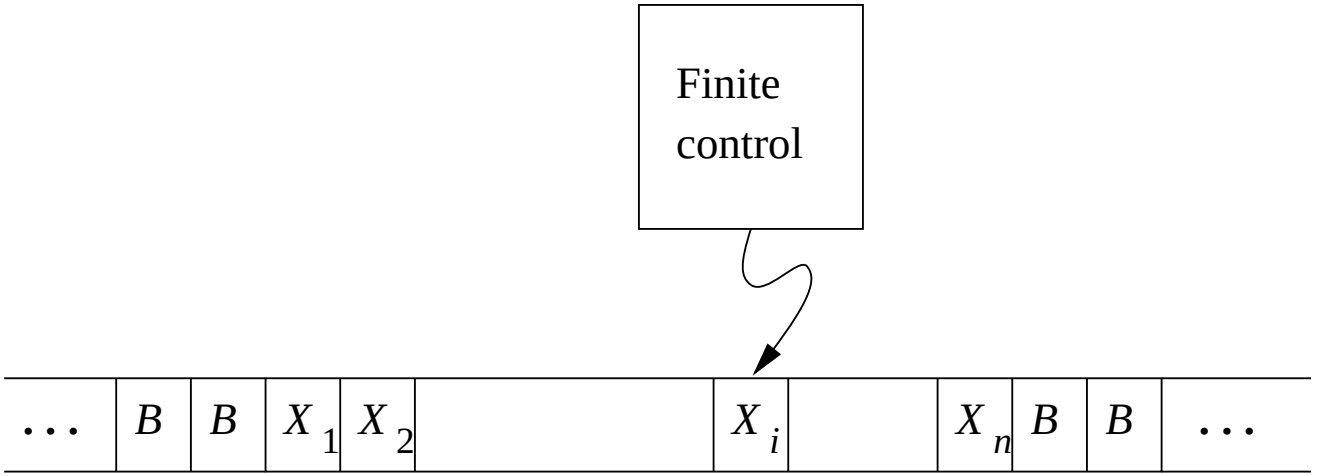
Il numero di linguaggi diversi su un alfabeto non è numerabile.

I programmi (stringhe finite su un alfabeto) sono numerabili: li ordino per lunghezza, e poi lessicograficamente $\Rightarrow$ primo programma, secondo programma, ecc.

Quindi esistono infinitamente più linguaggi che programmi.

Quindi devono esistere problemi indecidibili (Godel 1931).

La macchina di Turing (Turing, 1936)



Una TM fa una *mossa* in funzione del suo stato, e del simbolo sotto la testina di lettura del nastro.

In una mossa, una TM

1. cambia stato
2. scrive un simbolo del nastro nella cella sotto la testina
3. muove la testina di una cella verso destra o verso sinistra

Una *macchina di Turing* deterministica è una 7-tupla

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

dove

- Q è un insieme finito di *stati*,
- Σ è un insieme finito di *simboli di input*,
- Γ è un insieme finito di *simboli di nastro*,
- δ è una *funzione di transizione* da $Q \times \Gamma$ a $Q \times \Gamma \times \{L, R\}$,
- q_0 è lo *stato iniziale*,
- $B \in \Gamma$ è il *simbolo blank*, e
- $F \subseteq Q$ è l'isieme di stati finali.

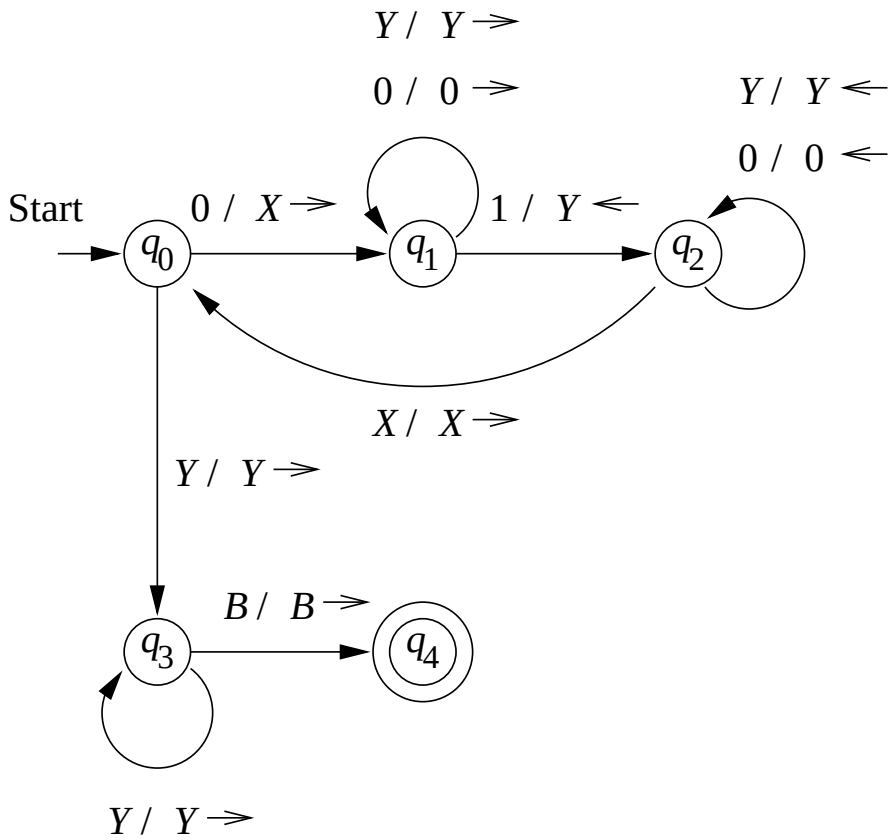
Una TM per $\{0^n 1^n : n \geq 1\}$

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

dove δ è data dalla tabella seguente

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
$\star q_4$					

Possiamo rappresentare M con il seguente *diagramma di transizione*



Accettazione per arresto

Una TM *si arresta* se entra in uno stato q guardando un simbolo di nastro X e non ci sono mosse possibili, cioè $\delta(q, X)$ non è definita.

Se una TM accetta una stringa, possiamo assumere che si arresti (basta rendere indefinito $\delta(q, X)$ per ogni q accettante).

Se non accetta, non possiamo fare in modo che si arresti.

Linguaggi *ricorsivi*: esiste una TM che si arresta su ogni stringa (sia accettata che no).

Linguaggi *ricorsivamente enumerabili*: esiste una TM che si arresta se la stringa è accettata.

Problema *decidibile*: esiste una TM che si arresta sempre.

Altri modelli di TM

Estensioni: più nastri, non-determinismo.

Restrizioni: nastro illimitato solo a destra, divieto di sostituire un simbolo del nastro con B, nastro come stack.

Tutti i modelli sono equivalenti: accettano i linguaggi *ricorsivamente enumerabili* (tesi di Church, 1936).

TM non deterministiche

Dati uno stato e un simbolo di nastro, la funzione di transizione dà un insieme di triple: (nuovo stato, simbolo da scrivere, direzione dello spostamento). La TM ne sceglie una.

Linguaggio accettato: insieme di stringhe per cui c'è una sequenza di mosse che porta ad uno stato finale.

TM non deterministiche e TM deterministiche accettano gli stessi linguaggi.

Linguaggi ricorsivamente enumerabili

D'ora in poi: calcolatore = macchina di Turing

L è *ricorsivamente enumerabile* se L = L(M) per una TM M. M si ferma se accetta una stringa, ma potrebbe non fermarsi se non la accetta.

Esistono linguaggi indecidibili: consideriamo il linguaggio formato dalle coppie (M, w) tali che:

- M è una TM (cofidicata in binario) con alfabeto {0,1}
- w è una stringa di 0 e 1
- M accetta w

Se questo problema è indecidibile, allora lo è anche il problema in cui una TM può avere qualunque alfabeto.

Supponiamo di codificare una TM come una stringa di 0 e 1. Così possiamo parlare della i-esima TM M_i.

Il linguaggio di diagonalizzazione

Il *linguaggio di diagonalizzazione* L_d è l'insieme delle stringhe w_i tali che $w_i \notin L(M_i)$.

Tutte le stringhe w tali che M con codice w non accetta w.

Matrice con TM sulle righe e stringhe sulle colonne $\Rightarrow$ la diagonale corrisponde a stringhe w_i e TM M_i. Le stringhe di L_d corrispondono agli 0 della diagonale.

È possibile che la diagonale complementata sia una riga? No, perchè la diagonale complementata è in disaccordo con ogni riga in almeno una posizione.

$\Rightarrow$ L_d non può essere accettato da nessuna TM.

Linguaggi ricorsivi

L è ricorsivo se $L = L(M)$ per una TM M tale che:

- se $w \in L$, allora M la accetta (e si arresta)
- se $w \notin L$, allora M non la accetta ma si arresta.

Problema (dell'accettazione di L): *decidibile* se L è ricorsivo, altrimenti *indecidibile*.

Classi di linguaggi

- ricorsivi = decidibili = M si arresta sempre
- ricorsivamente enumerabili = M si arresta se accetta
- non ricorsivamente enumerabili. Esempio: L_d.

Il linguaggio universale

Il linguaggio universale L_u è l'insieme delle stringhe binarie che codificano una coppia (M, w) dove $w \in L(M)$.

Esiste una TM U, detta TM universale, tale che $L_u = L(U)$.

U ha tre nastri: uno per il codice di M e w, uno per il nastro di M, e uno per la codifica dello stato di M. Così U simula M su w, e accetta (M, w) se e solo se M accetta w.

Dato che alcune TM M hanno linguaggi non ricorsivi ma ricorsivamente enumerabili, non si fermano quando la stringa w in input non è del linguaggio. Anche U avrà lo stesso comportamento su (M, w). Quindi L_u è RE e non ricorsivo.

Il problema dell'arresto

Data una TM M, definiamo $H(M)$ l'insieme delle stringhe w tali che M si arresta con input w.

Consideriamo il linguaggio che contiene le coppie (M, w) tali che $w \in H(M)$.

Anche per questo linguaggio esiste una TM che prende in input le coppie (M, w) tali che $w \in H(M)$, e simula il comportamento di M su w. Quindi l'insieme di tali coppie è un linguaggio RE ma non ricorsivo.

Quindi, non esiste nessun algoritmo che possa dire se un programma termina o no. Esiste però un algoritmo che, se il programma in input termina, si ferma, e se non termina cicla.

Proprietà dei linguaggi

Tulle le proprietà non banali di una TM (cioè dei linguaggi RE) sono indecibili.

Esempio: essere un linguaggio libero dal contesto. Controllare se una TM accetta un linguaggio libero dal contesto è indecidibile.

Proprietà banale: se viene soddisfatta da tutti i linguaggi o da nessuno.

Teorema di Rice:

Ogni proprietà non banale dei linguaggi RE è indecidibile.

Esempi di problemi indecidibili

Tutti i problemi sulle TM che riguardano i linguaggi accettati sono indecidibili.

Esempi:

Il linguaggio accettato da una TM è vuoto?

Il linguaggio accettato da una TM è finito?

Il linguaggio accettato da una TM è regolare?

Il linguaggio accettato da una TM è libero dal contesto?

Il linguaggio accettato da una TM contiene la stringa ab?

Il linguaggio accettato da una TM contiene tutti i numeri pari?

Non tutto è indecidibile

Problemi che riguardano gli stati di una TM, e non il linguaggio accettato, possono essere decidibili.

Esempi:

Se una TM ha cinque stati.

Se esiste un input tale che una TM faccia almeno cinque passi prima di fermarsi.

Se una TM esegue una certa transizione.

Se una TM passa per lo stato p.

Se una TM non passa mai per lo stato p.

Se una TM passa dallo stato p allo stato q in al più tre passi.

Problemi intrattabili

Ci occuperemo solo di problemi decidibili, cioè ricorsivi.

Tra loro, alcuni sono trattabili, cioè risolvibili in tempo polinomiale, e altri no.

Tempo polinomiale ed esponenziale

Una TM M ha *complessità in tempo T(n)* se, dato un input w di lunghezza n, M si ferma dopo al massimo $T(n)$ passi.

Esempi: $T(n) = 5n^2 + 3n$, o $T(n) = 4^n + 3n^2$

Un linguaggio L è nella classe $\mathcal{P}$ se esiste un polinomio $T(n)$ tale che $L = L(M)$ per una TM deterministica M con complessità in tempo $T(n)$.

Se la complessità non è polinomiale, si dice che è esponenziale, anche se $T(n)$ non è un esponenziale.

Esempio: $T(n) = n^{log_2 n}$. Cresce più velocemente di qualunque polinomio, ma più lentamente di qualunque esponenziale 2^{cn}

Tempo polinomiale non deterministico

Un linguaggio L è nella classe NP (non polinomiale deterministica) se esiste una TM non deterministica M tale che $L = L(M)$ e, quando M ha un input lungo n, al massimo fa $T(n)$ mosse con $T(n)$ polinomio.

Dato che ogni TM deterministica è una TM non-deterministica, allora $\mathcal{P} \subseteq \mathcal{NP}$.

Una TM non deterministica polinomiale può esaminare un numero esponenziale di strade "in parallelo". Quindi sembra che sia impossibile che $\mathcal{P} = \mathcal{NP}$. Ma nessuno lo ha mai provato.

Riduzioni polinomiali

Per dimostrare che un problema P_2 non può essere risolto in tempo polinomiale, riduciamo un altro problema P_1, non in P, a P_2.

Riduzione: da una istanza (stringa accettata) di P_1 ad una di P_2

Vincolo in più: in tempo polinomiale

Se P_1 è riducibile a P_2: se P_2 è in P, allora lo è anche P_1. Dato che P_1 non è in P, neanche P_2 lo è.

P_2 è difficile almeno tanto quanto P_1.

Problemi NP-completi

Un linguaggio L è NP-completo se è in NP e se per ogni altro linguaggio L' in NP esiste una riduzione polinomiale di L' a L.

Esempio: problema del commesso viaggiatore.

Sono i problemi più difficili tra quelli in NP.

Teorema 10.4: Se P_1 è NP-completo e P_2 è in NP, e posso ridurre polinomialmente P_1 a P_2, allora P_2 è NP-completo.

Prova: la riduzione polinomiale è transitiva.

Teorema 10.5: Se un problema NP-completo è in P, allora P=NP.

Problemi NP-ardui

Un problema è NP-arduo se possiamo dimostrare che ogni problema in NP è riducibile a lui, ma non che è in NP.

Quindi potrebbe anche non avere un algoritmo esponenziale (se non è in NP).