Lezione 9 – l’Halting Problem

Lezione del 05/04/2024

Si fa presto a dire "infinito"

- La dispensa 4 descrive il lavoro di Cantor (vabbé, una piiiccola parte del lavoro di Cantor) sui numeri transfiniti:
  - che dimostra che esistono insiemi infiniti "piccoli" e insiemi infiniti "grandi"
  - dove "piccolo" e "grande" sono basati sul concetto di corrispondenza biunivoca
  - e un numero transfinito è la "grandezza" di un insieme infinito
- Cantor ha dimostrato che non esiste una corrispondenza biunivoca fra l'insieme dei numeri naturali e l'insieme dei numeri reali
  - ossia che comunque si scelga una funzione (totale) $f: \mathbb{N} \rightarrow \mathbb{R}$ esiste $y_f \in \mathbb{R}$ tale che, per ogni $x \in \mathbb{N}$, $f(x) \neq y_f$
- e questo prova che l'insieme dei numeri reali è strettamente "più grande" dell'insieme dei naturali
  - in effetti, Cantor ha dimostrato anche che l'insieme infinito "più piccolo" di tutti è quello dei numeri naturali

Si fa presto a dire “infinito”

- Non studiamo la dispensa 4
  - che è molto bella (non perché l’ho scritta io, sono belli gli argomenti che tratta) e, se qualcuno vuole guardarla, ne possiamo discutere al di fuori delle lezioni
- Vediamo se riesco a darvi un assaggino di quanto è bella...
- In realtà, quel che Cantor ha **dimostrato** è che non esiste una corrispondenza biunivoca fra l’insieme dei numeri naturali e l’intervallo reale [0,1]
- ossia, che nel segmento 

, in questo segmentino tranquillo tranquillo che sappiamo disegnare senza difficoltà
- dentro quel segmentino ci sono infinitissimamente più punti di quanti sono i numeri naturali!
  - E mai noi riusciremmo a disegnarli tutti, i numeri naturali...
- E che lo stesso vale per ogni intervallo reale [0, ε]
  - per quanto vicino allo 0 scegliamo ε
- Non fa un po’ girar la testa?

Problemi irrisolvibili

- La dispensa 5 è dedicata allo studio dell'esistenza di problemi "impossibili" da risolvere (come sappiamo, Turing-irrisolvibili, con tutto quel che segue)
- Nei paragrafi 5.1 e 5.2 si utilizza quanto studiato nella dispensa 4 per dimostrare che ***esiste*** **un problema irrisolvibile**, secondo lo schema seguente:
  - 1) si dimostra che le macchine di Turing sono tante quanti i numeri naturali
  - 2) e, utilizzando questo ''conteggio'', si mostra che esiste almeno un linguaggio che non è deciso da alcuna macchina di Turing
    - dimostrando, cioè, che i problemi sono più dei numeri naturali
- ossia, ***esiste*** **almeno un problema che non può essere risolto con una macchina di Turing** (e, quindi, per la tesi di Church-Turing non può proprio essere risolto)
  - facile facile! Basta contare...
- Per dimostrare il punto 1) dobbiamo tornare un po' indietro…

macchina = parola = numero

- ...Siamo a pag. 11 della dispensa 2: avevamo descritto una macchina di Turing T
  - con alfabeto {0,1},
  - insieme degli stati $Q_T = \{\omega_0,\dots,\omega_{k-1}\}$, con stato iniziale ω_0, stato di accettazione ω_1, e stato di rigetto ω_2 – osservate: $|Q_T| = k$
  - e insieme delle quintuple $P = \{p_1,\dots, p_h\}$, dove la sua *i*-esima quintupla è $p_i = \langle \omega_{i1} , b_{i1} , b_{i2} , \omega_{i2} , m_i \rangle$

- mediante la parolona
  - $\rho_T = \omega_0 - \omega_1 \otimes \omega_{11} - b_{11} - b_{12} - \omega_{12} - m_1 \oplus \omega_{21} - b_{21} - b_{22} - \omega_{22} - m_2 \oplus \dots$

    $\dots \oplus \omega_{h1} - b_{h1} - b_{h2} - \omega_{h2} - m_h \oplus$

macchina = parola = numero

- Poi, a pag. 13 (dispensa 2) avevamo introdotto una codifica binaria b^Q dell'insieme Q_T degli stati di T, che, nella lezione 4 di questa serie, avevamo semplificato come segue:
  - $b^Q : Q_T \rightarrow \{ 0,1 \}^k$, ossia, la codifica b^Q rappresenta uno stato di T mediante una parola di k bit
  - $b^Q(\omega_i)$ è la parola che ha un 1 in posizione i+1 e 0 altrove – esempio: se k=4, $b^Q(\omega_0)$=1000, $b^Q(\omega_1)$=0100, $b^Q(\omega_2)$=0010, $b^Q(\omega_3)$=0001
- a questo punto, avevamo rappresentato T mediante la seguente parolona nell'alfabeto $\Sigma = \{0, 1, \oplus, \otimes, -, f , s, d\}$:
  - $\beta_T = b^Q(\omega_0) - b^Q(\omega_1) \otimes b^Q(\omega_{11}) - b_{11} - b_{12} - b^Q(\omega_{12}) - m_1 \oplus b^Q(\omega_{21}) - b_{21} - b_{22} - b^Q(\omega_{22}) - m_2 \oplus \ldots \oplus b^Q(\omega_{h1}) - b_{h1} - b_{h2} - b^Q(\omega_{h2}) - m_h \oplus$
- Ci siamo quasi…

macchina = parola = numero

- Quello che viene fatto nel paragrafo 5.1 (dispensa 5) è trasformare la parola β_T in un numero: sostituiamo in β_T
  - ogni carattere ‘s’ con il carattere ‘5’, ogni carattere ‘f’ con il carattere ‘6’, e ogni carattere ‘d’ con il carattere ‘7’;
  - ogni carattere ‘-’ con il carattere ‘4’, ogni carattere ‘$\oplus$’ con il carattere ‘3’ e ogni carattere ‘$\otimes$’ con il carattere ‘2’;
  - ogni carattere ‘$\square$’ con il carattere ‘9’;
- Infine, premettiamo il carattere ‘8’ alla parola ottenuta.
- Alla “parola” ottenuta… *In realtà, quello che abbiamo ottenuto è un numero intero*
- Abbiamo associato ad ogni macchina di Turing un numero intero
  - e l’associazione è univoca: a macchine di Turing diverse sono associati interi diversi
  - o, equivalentemente, un intero non può corrispondere a due macchine di Turing
- **Cioè, abbiamo imparato a rappresentare le macchine di Turing mediante numeri naturali!**
  - **Mediante parole nell’alfabeto {0,… ,9} che possono essere lette come numeri naturali**

E ora, ''ordiniamo'' le macchine

- Abbiamo imparato a rappresentare le macchine di Turing mediante numeri naturali
  - se una macchina di Turing T è rappresentata dall'intero h indichiamo quella macchina come T_h;
- A questo punto, possiamo ordinare le macchine:

  scriviamo $T_h < T_k$ se h < k

- Allora, abbiamo

  una ''*prima*'' macchina – quella rappresentata dall'intero più piccolo di tutti –

  una ''seconda'' macchina e così via...

- Indichiamo con T_{h_1} la prima macchina, con T_{h_2} la seconda macchina, ... , e, in generale, con T_{h_i} la i-esima macchina

Ma anche l'input di queste macchine...

- Osserviamo che, se ci limitiamo a considerare macchine di Turing che lavorano sull'alfabeto binario, anche l'input di una TM è rappresentato da un numero intero
  - ESERCIZIO: e come distinguere 101 da 0101 da 0000101???
    (sugg. : possiamo premettere un 1: 101 → 1101 , 0101 → 10101 , 0000101 → 10000101
- Allora, abbiamo
  un ''*primo*'' input, il numero 1 – che rappresenta la parola '□'
  un ''*secondo*'' input, 10 – che rappresenta la parola 0
  un ''*terzo*'' input, 11 – che rappresenta la parola 1
  un ''*quarto*'' input, 100 – che rappresenta la parola 00

  e così via...
- Ossia, una generica parola binaria b è rappresentata dal numero n la cui rappresentazione binaria è ottenuta premettendo un 1 alla parola b
  - ESEMPIO: la parola b=0011 è rappresentata dal numero 19 perché la rappresentazione binaria di 19 è 10011
- E, dunque, possiamo pensare che l'input di una macchina di Turing sia un numero intero

E allora...

- Possiamo costruire una matrice M binaria con infinite righe e infinite colonne tale che

$$M[\,i,k] = \begin{cases} 1 \text{ se la computazione } T_{h_i}(k) \text{ termina in } q_A \\ 0 \text{ se la computazione } T_{h_i}(k) \textbf{ non } \text{termina in } q_A \end{cases}$$

- cioè,

$$\begin{array}{c|ccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \dots \\ T_{h_1} & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & \dots \\ T_{h_2} & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & \dots \\ & & & & & \dots & & & & & & \\ T_{h_i} & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & \dots \\ & & & & & \dots & & & & & & \end{array}$$

- ossia, T_{h_2} accetta gli interi: 2 (che rappresenta la parola 0), 3 (che rappresenta la parola 1), 6 (la parola 10), 8 (la parola 000), 10 (la parola 010), ...

M e linguaggi

- Abbiamo costruito la matrice M

	1	2	3	4	5	6	7	8	9	10	…
T_{h_1}	0	0	1	0	1	0	0	1	1	1	…
T_{h_2}	0	1	1	0	0	1	0	1	0	1	…
					…						
T_{h_i}	1	0	1	0	1	1	1	0	1	0	…
					…						

- M rappresenta <u>*tutti*</u> i linguaggi accettati dalle macchine di Turing
  - perché tutte le macchine di Turing sono rappresentate in M!
- Così, ad esempio, gli 1 sulla riga 1 rappresentano il linguaggio $\mathbf{L_1}$ accettato da $\mathbf{T_{h_1}}$: $\mathbf{L_1}$ = { 3, 5, 8, 9, 10, …}
  - o, equivalentemente, $\mathbf{L_1}$ = { 1, 01, 000, 001, 010, …}

E ora, diagonale!

- Consideriamo la diagonale della matrice M

	1	2	3	4	5	6	7	8	9	10	…
T_{h_1}	**0**	0	1	0	1	0	0	1	1	1	…
T_{h_2}	0	**1**	1	0	0	1	0	1	0	1	…
T_{h_3}	0	0	**1**	1	0	1	0	1	1	0	…
T_{h_4}	1	0	1	**0**	1	1	1	0	1	0	…

…

- e costruiamo un linguaggio L che rappresenta la diagonale ''*complementata*''
  - ossia, contiene tutti e soli i numeri cui corrisponde uno 0 sulla diagonale
  - nell'esempio, L conterrebbe 1 e 4
- Formalmente, L = { k : M[k,k]=0}
- O, anche, L = { k : T_{h_k}(k) **non** accetta}

Ricapitoliamo

- **L = { k : T_{h_k}(k) <u>non</u> accetta }**
- allora:
  - il linguaggio L non è accettato da T_{h_1}, perché 1 ∈ L ma T_{h_1}(1) non accetta
  - il linguaggio L non è accettato da T_{h_2}, perché 2 ∉ L ma T_{h_2}(2) accetta
  - il linguaggio L non è accettato da T_{h_3}, perché 3 ∉ L ma T_{h_2}(3) accetta
  - il linguaggio L non è accettato da T_{h_4}, perché 4 ∈ L ma T_{h_4}(4) non accetta
  - …
- Ossia, il linguaggio L non è accettato da nessuna delle macchine che compaiono come righe della matrice M
- Ma ***<u>tutte</u>*** le macchine di Turing sono rappresentate in M
- Questo significa che

  non esiste alcuna macchina di Turing che accetti L
- Ossia,

  L è un linguaggio non accettabile

Problemi irrisolvibili

- Abbiamo dimostrato che ***esiste*** almeno un problema che non può essere risolto con una macchina di Turing (e, quindi, per la tesi di Church-Turing non può essere risolto)
  - abbiamo usato la tecnica della **diagonalizzazione** – la stessa usata da Cantor per mostrare che [0,1] contiene più punti di quanti sono i numeri naturali
  - in qualche modo, abbiamo mostrato che il numero dei problemi è maggiore del numero di macchine
- Purtroppo, la dimostrazione non ci dà alcuna idea di come possa essere fatto un problema irrisolvibile – *non ci permette di costruirlo!*
  - magari, i problemi irrisolvibili sono strani, astrusi, sono problemi astratti costruiti apposta per essere irrisolvibili… Problemi, insomma, che non incontreremmo mai nella vita reale e che, quindi, che ce ne importa se non li sappiamo risolvere?
- Invece, non è così!
- Turing ha costruito un problema irrisolvibile
  - anzi, la sua macchina l'ha inventata proprio per arrivare a dimostrare che questo problema è irrisolvibile
  - ed è un problema con il quale ogni informatico fa i conti tutti i giorni!

Costruire un problema irrisolvibile

- Ricordiamo che abbiamo imparato a rappresentare una macchina di Turing mediante un numero naturale – abbiamo **codificato** macchine di Turing mediante numeri naturali
  - se vogliamo dirlo bene, abbiamo trovato una corrispondenza biunivoca fra le macchine di Turing e un sottoinsieme dei numeri naturali
  - i numeri naturali che rappresentano macchine di Turing hanno certe proprietà: iniziano tutti per 8, …
  - ESERCIZIO: scrivete tutte le proprietà che un numero deve soddisfare perché sia la codifica di una macchina di Turing. E fatene un algoritmo ☺
- E abbiamo anche visto che, se ci limitiamo a considerare macchine di Turing che lavorano sull'alfabeto binario, anche l'input di una TM è un numero intero

Costruire un problema irrisolvibile

- Turing considerò il seguente linguaggio, sottoinsieme di $\mathbb{N} \times \mathbb{N}$:

  L_H = { (i,x) $\in \mathbb{N} \times \mathbb{N}$: i è la codifica di una macchina di Turing T_i e T_i (x) termina }

- che si chiama **Halting Problem**
- Turing dimostrò che L_H è accettabile
- e dimostrò anche che L_H non è decidibile
  - e questo, come sappiamo bene!, significa che L_H^C non è accettabile
  - e su questo punto torneremo più avanti
- E noi, adesso, studiamo queste dimostrazioni (paragrafo 5.3 della dispensa 5)
- Ma prima cerchiamo di capire che senso ha domandarsi, data (i,x) $\in \mathbb{N} \times \mathbb{N}$, se (i,x) $\in L_H$
  - quale può essere la rilevanza di questa domanda?

A chi importa dell'Halting Problem?

- Sei informatico, ti capiterà, qualche volta nella vita, di scrivere un programma
  - complicatissimo – mesi e mesi di lavoro
- Bene, dopo tutta questa fatica, lanci il tuo programma su un certo input x
  - x è un'istanza del problema risolto dal tuo programma della quale è importantissimissimo calcolare la soluzione!
- e attendi la risposta…
- ... e attendi ...
- ... e attendi ...
- Ti viene un dubbio atroce: e se fosse andato in loop?!
- Certo sarebbe bello se esistesse un **programma** che, se gli do in input un altro programma P e un suo input **x**, quello mi dice se l'esecuzione di P su **x** termina oppure no
  - sarebbe bello se esistesse un **programma** che decide l'Halting Problem!

L_H è accettabile – Teorema 5.4

- Questo è facile: prendete la macchina Universale U e le fate un paio di modifiche – trasformandola nella macchina U'
  - la prima modifica serve a fare in modo che U' verifichi se l'input i scritto sul suo primo nastro è davvero la codifica di una macchina di Turing T_i – e che ve l'ho assegnato a fare l'esercizio, sennò?!
  - se non è così, U'(i,x) rigetta
- La seconda modifica, serve a fare in modo che, se i è la codifica di una macchina di Turing T_i , U' accetti la coppia (i,x) ogni qualvolta T_i (x) termina, ossia, sia nel caso in cui accetta sia nel caso in cui rigetta:
- accertato che i è la codifica di una macchina di Turing T_i, U'(i,x) simula U(i,x) e
  - se U(i,x) accetta allora U'(i,x) accetta
  - se U(i,x) rigetta allora U'(i,x) accetta
- quindi U'(i,x) accetta tutte e sole le coppie (i,x) che appartengono a L_H – ossia, L_H è accettabile
- Ma se (i,x) $\notin L_H$? Ossia, se (i,x) $\in L_H^C$? Nulla possiamo concludere circa l'esito della computazione U'(i,x)…

L_H non è decidibile – Teorema 5.5

- La dimostrazione è per assurdo: supponiamo che L_H sia decidibile
- Se L_H è decidibile, allora esiste una macchina T tale che, per ogni $(i,x) \in \mathbb{N} \times \mathbb{N}$,
  - T(i,x) accetta se $(i,x) \in L_H$
  - T(i,x) rigetta se $(i,x) \notin L_H$
  - NOTA BENE: T termina su ogni input!
- Ebbene, se abbiamo T, possiamo utilizzare T per ***costruire*** una nuova macchina T' tale che
  - T'(i,x) accetta se $(i,x) \notin L_H$
  - T'(i,x) rigetta se $(i,x) \in L_H$
- Come è fatta T'? Beh, prendiamo T, la smontiamo e invertiamo gli stati di accettazione e di rigetto.
- Oppure, se non abbiamo pinze e cacciaviti, usiamo T come se fosse una funzione
  - con la coppia (i,x) sul nastro, T' "invoca" T passandogli (i,x) come parametri, e quando T(i,x) termina T' risponde complementando q_A con q_R

L_H non è decidibile – Teorema 5.5

- Se L_H è decidibile, allora esiste una macchina T che accetta (i,x) se (i,x) $\in L_H$, rigetta (i,x) se (i,x) $\notin L_H$
- E da T, ***costruiamo*** T' che rigetta (i,x) se (i,x) $\in L_H$, accetta (i,x) se (i,x) $\notin L_H$
  - e, come T, anche T' termina su ogni input

- Osservate: più che *simulare* T, T' **usa** T – proprio come nei linguaggi di programmazione si invoca una funzione
  - questo significa che, per *costruire* T', non abbiamo bisogno di sapere come è fatta T
  - la usiamo "chiusa", a *scatola nera*
  - tutto quello che abbiamo bisogno di sapere, per costruire T', è come risponde T sui vari input (i,x)

L_H non è decidibile – Teorema 5.5

- Se L_H è decidibile, allora esiste una macchina T che accetta (i,x) se $(i,x) \in L_H$, rigetta (i,x) se $(i,x) \notin L_H$ – quindi, T termina su ogni input
- E da T, ***costruiamo*** T' che rigetta (i,x) se $(i,x) \in L_H$, accetta (i,x) se $(i,x) \notin L_H$
  - e, come T, anche T' termina su ogni input
- Ora, di nuovo con la "tecnica della scatola nera", a partire da T', ***costruiamo*** una macchina T'', che accetta (i,x) se $(i,x) \notin L_H$, mentre **non termina** se $(i,x) \in L_H$,
  - con la coppia (i,x) sul nastro, T'' invoca T' passandogli (i,x) come parametri: quando T'(i,x) termina, se termina in q_A allora anche T'' termina in q_A, se, invece, T'(i,x) termina in q_R allora T''(i,x) entra in loop
  - è sufficiente aggiungere le due quintuple **$\langle q_R, 0, 0, q_R, F \rangle$ e $\langle q_R, 1, 1, q_R, F \rangle$** e rimuovere q_R dall'insieme degli stati finali di T''
- Repetita iuvant: per ogni $(i,x) \in \mathbb{N} \times \mathbb{N}$,
  - T''(i,x) accetta se $(i,x) \notin L_H$
  - **T''(i,x) non termina se $(i,x) \in L_H$**
  - NOTA BENE: col cavolo che T'' termina su ogni input!

L_H non è decidibile – Teorema 5.5

- Se L_H è decidibile, allora esiste una macchina T che accetta (i,x) se (i,x) $\in L_H$, rigetta (i,x) se (i,x) $\notin L_H$ – quindi, T termina su ogni input
- E da T, ***costruiamo*** T' che rigetta (i,x) se (i,x) $\in L_H$, accetta (i,x) se (i,x) $\notin L_H$,
- poi da T' ***costruiamo*** T'', che accetta (i,x) se (i,x) $\notin L_H$, mentre **non termina** se (i,x) $\in L_H$,
- Penultimo passo: siamo quasi pronti a tirare la stoccata finale
- NOTA BENE: poiché (i,x) $\in \mathbb{N} \times \mathbb{N}$, ossia, l'input di T, di T' e di T'' è costituito da una coppia di interi, **allora (i,i) – che è una coppia di interi – può ben essere dato in input a queste tre macchine**: se i è la codifica di una macchina di Turing, allora
  - **T(i, i) accetta se (i, i)** $\in$ $\mathbf{L_H}$, ossia se T_i (i) termina,
    e **T(i, i) rigetta se (i, i)**$\notin$ $\mathbf{L_H}$, ossia se T_i (i) non termina
  - **T'(i, i) accetta se (i, i)** $\notin$ $\mathbf{L_H}$, ossia se T_i (i) non termina,
    e **T'(i, i) rigetta se (i, i)**$\in$ $\mathbf{L_H}$, ossia se T_i (i) termina
  - **T''(i, i) accetta se (i, i)** $\notin$ $\mathbf{L_H}$, ossia se T_i (i) non termina,
    e **T''(i, i) non termina se (i, i)**$\in$ $\mathbf{L_H}$, ossia se T_i (i) termina

L_H non è decidibile – Teorema 5.5

- Se L_H è decidibile, allora esiste una macchina T che accetta (i,x) se (i,x) $\in L_H$, rigetta (i,x) se (i,x) $\notin L_H$ – quindi, T termina su ogni input
- E da T, *costruiamo* T' che rigetta (i,x) se (i,x) $\in L_H$, accetta (i,x) se (i,x) $\notin L_H$,
- poi da T' *costruiamo* T'', che accetta (i,x) se (i,x) $\notin L_H$, mentre **non termina** se (i,x) $\in L_H$,
- Compreso che come input di T, T' e T''possiamo usare una coppia (i,x) $\in \mathbb{N} \times \mathbb{N}$ tale che x=i, di nuovo con la "tecnica della scatola nera", a partire da T'', ***costruiamo*** un'ultima macchina T* che lavora con un solo input e tale **che l'esito della computazione T*(i) coincide con l'esito della computazione T''(i, i)**
- Ossia, se i è la codifica di una macchina di Turing, allora
  - **T*(i) accetta se (i, i) $\notin L_H$, ossia se T_i (i) non termina,**
  - **T*(i) non termina se (i, i) $\in L_H$, ossia se T_i (i) termina**

  - PS: se i *non* è la codifica di una macchina di Turing, allora (i, i) $\notin L_H$, e quindi T*(i) accetta, ma di questo ci interessa poco

L_H non è decidibile – Teorema 5.5

- Se L_H è decidibile, allora esiste una macchina T che accetta (i,x) se (i,x) ∈ L_H, rigetta (i,x) se (i,x) ∉ L_H – quindi, T termina su ogni input
- E da T, ***costruiamo*** T' che rigetta (i,x) se (i,x) ∈ L_H, accetta (i,x) se (i,x) ∉ L_H ,
- poi da T' ***costruiamo*** T'', che accetta (i,x) se (i,x) ∉ L_H, mentre non termina se (i,x) ∈ L_H,
- Infine, da T'' ***costruiamo*** T* con un solo input: T*(i) accetta se (i, i) ∉ L_H, mentre **non termina** se (i, i) ∈ L_H,
- ALTRA NOTA BENE: **poiché abbiamo supposto che T esiste, allora anche T* esiste**
  - ossia è una macchina vera per davvero – l'abbiamo costruita fisicamente a partire da T!
- E, se T* esiste, allora la posso codificare come intero – lo abbiamo visto all'inizio di questa lezione
- Chiamiamo k il codice di T* ottenuto applicando il procedimento illustrato nelle prime 7 slides di questa lezione
- cioè, $T^* = T_k$

L_H non è decidibile – Teorema 5.5

- Chiamiamo k il codice di T* ottenuto applicando il procedimento illustrato nelle prime 7 slides di questa lezione - cioè, **$T^* = T_k$**
- Ma k è un intero
- allora, k può essere input di T* - ossia, input di T_k
- Ossia, possiamo considerare la computazione **$T_k(k)$**
- Ebbene, siamo al nocciolo della questione:

quale è l'esito della computazione $T^*(k) = T_k(k)$?

- Osservate bene: $T_k(k)$ è la computazione di *una macchina che si interroga su sé stessa* – che cerca di verificare se essa stessa soddisfa una certa proprietà

L_H non è decidibile – Teorema 5.5

- **Quale è l'esito della computazione $T^*(k) = T_k (k)$?**
- Ricapitoliamo:
  - $L_H = \{ (i,x) \in \mathbb{N} \times \mathbb{N}$: i è la codifica di una macchina di Turing T_i e T_i (x) termina }
  - $T^*(k) = T_k (k)$ accetta se $(k, k) \notin L_H$, mentre non termina se $(k, k) \in L_H$,
- Dunque, $T^*(k) = T_k (k)$ o accetta oppure non termina
- $T^*(k) = T_k (k)$ potrebbe forse accettare?
  - $T^*(k) = T_k (k)$ accetta solo se $(k, k) \notin L_H$,
  - poiché k è il codice di una macchina di Turing, $(k, k) \notin L_H$ solo se $T_k (k)$ non termina: dunque, **$T^*(k) = T_k (k)$ accetta solo se $T^*(k) = T_k (k)$ non termina**
  - OPS! Allora, no: non è possibile che $T^*(k) = T_k (k)$ accetti
- Allora, non c'è altra possibilità: $T^*(k) = T_k (k)$ non termina! Siamo sicuri?
  - $T^*(k) = T_k (k)$ non termina solo se $(k, k) \in L_H$, ossia (dalla definizione di L_H), solo se T_k (k) termina: dunque, **$T^*(k) = T_k (k)$ non termina solo se $T^*(k) = T_k (k)$ termina**
  - RI-OPS! Allora, no: non è possibile che $T^*(k) = T_k (k)$ non termini

L_H non è decidibile – Teorema 5.5

- In conclusione,
  - $T^*(k) = T_k (k)$ o accetta oppure non termina – non vi sono altre possibilità!
  - E, però, non è possibile che $T^*(k) = T_k (k)$ accetti e non è possibile nemmeno che $T^*(k) = T_k (k)$ non termini
  - GOSH!
- Qualcosa non torna… Ricapitoliamo:
  - partendo dall'ipotesi "L_H è decidibile" – ossia che esista la macchina T che decide L_H
  - siamo arrivati a **costruire** una computazione, $T^*(k) = T_k (k)$, che non può esistere!
- E, quindi, non c'è verso, abbiamo sbagliato a supporre che L_H è decidibile!
- Abbiamo, così, dimostrato che **L_H è indecidibile!**

L_H è accettabile e L_H non è decidibile!

- Ma cosa significa che L_H è accettabile ma non è decidibile?
  - ricordate quel che abbiamo dimostrato su accettabilità, decidibilità e linguaggi complemento un paio di lezioni fa?
  - "un linguaggio L è decidibile se e solo se L è accettabile e L^C è accettabile"
  - allora, poiché L_H è accettabile e L_H non è decidibile :
  - **$L_H{}^C$ non è accettabile!**
- E questo significa che, quando state lì ad aspettare se l'esecuzione del vostro (sudatissimo) programma termini sull'importantissima istanza che gli avete dato in input,
- la domanda alla quale è difficile rispondere è proprio
- Ma non è che, per caso, è andato in loop????

Un paio di note

- Intanto, c'è una piccolissima differenza fra la dimostrazione dell'indecidibilità dell'Halting Problem che vi ho proposto qui e quella che trovate sulla dispensa 5 (ma proprio piccola piccola):
  - in questa lezione ho fatto un passo intermedio in più: da T, a T', a T'', a T*
  - sulla dispensa si passa da T a T' e poi direttamente a T*
  - aggiungendo il passaggio a T'' mi è sembrato di aiutarvi. Ma non avrete alcuna difficoltà a seguire sulla dispensa dopo aver letto questa lezione!

- Poi, per chi ha intenzione di leggere la dispensa 4: la dimostrazione dell'indecidibilità dell'Halting Problem è una applicazione della tecnica di diagonalizzazione di Cantor. Provate a comprendere in che modo