Halting Theorem and Reducibility

Halting theorem

Questo è un problema fondamentale, che abbiamo trattato anche in Fondamenti teorica#Halting problem, ma qui lo ritrattiamo, perché così lo rifacciamo per bene. In parte è stato trattato anche al corso di Logica.

Enunciato Halting theorem Questo è molto simile a quanto presente sul (Sipser 2012). Ossia consideriamo il linguaggio

$$HALT = \{\langle x, y\rangle \in \Sigma^* \times \Sigma^* : x = code(M), M \text{ si ferma su } x\}$$

Dimostrazione Halting theorem La parte del sì è facile perché basta eseguirlo e vedere che si ferma (quindi abbiamo una La macchina di Turing#La macchina di Turing universale. Se si ferma appartiene al linguaggio, altrimenti è la parte in cui diverge.

Dimostrazione non decidibilità Supponiamo sia decidibile e dimostriamo l'assurdo. Se esiste una macchina f tale per cui decida quel linguaggio,

$$\begin{cases} f(g, y) = 1, g(y) \downarrow \\ \\ f(g, y) = 0, g(y) \uparrow \end{cases}$$

allora possiamo usare questa macchina per costruire un h tale che per cui

$$\begin{cases} h(g) = 1, f(g, g) = 0 \\ \\ h(g) = \uparrow, f(g, g) = 1 \end{cases}$$

Allora la computazione della funzione $h(h)$ genera un assurdo.

Il motivo è che $h(h) = 1 \iff f(h, h) = 0 \iff h(h) = \uparrow$ Questa cosa dovrebbe essere riscritta in modo *code* per essere comprensibile da macchine di Turing.

Opposto di Halting theorem Ossia vogliamo riconoscere il linguaggio

$$HALT^- = \{\langle x, y\rangle : x \neq code(M) \cup x = code(M) \cap M \text{ non si ferma su } x\}$$

Si può dimostrare che questo problema **non è nemmeno riconoscibile** da nessuno! #### Dimostrazione complemento di halting theorem Si ragiona anche qui per assurdo, se fosse riconoscibile, avremmo che Halting theorem principale sarebbe riconoscibile.

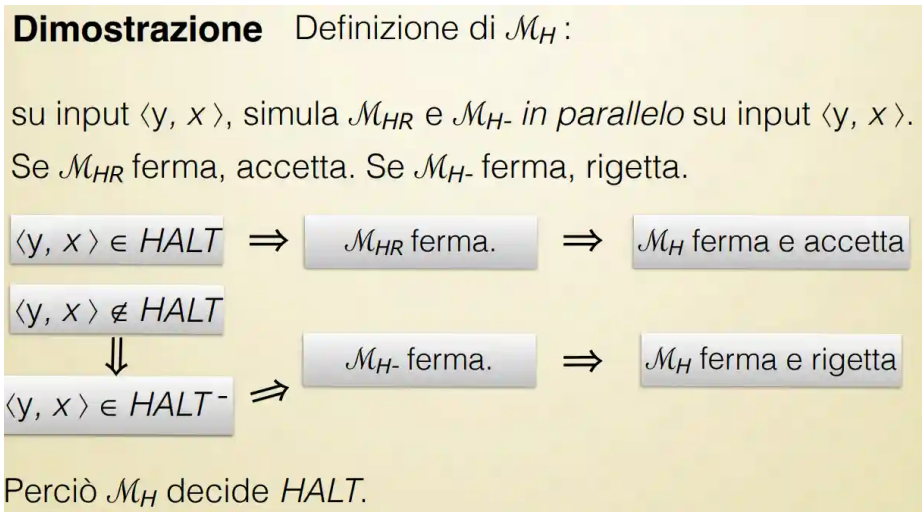

Figure 1: Halting Theorem and Reducibility-20240306120459845

Mapping reducibility

Definizione mapping reducibility Un linguaggio L' è riducibile a un altro linguaggio L se esiste una **funzione computabile totale** $f : \Sigma^* \to \Sigma^*$ tale che valga

$$x \in L' \iff f(x) \in L$$

E si scrive che $L' \leq L$ Ossia posso mappare qualunque parola in L' in una stringa in L. È molto importante che sia computabile, perché è un modo di dire che non stiamo barando nella dimostrazione, e mi appoggio soltanto all'espressività dei due linguaggi.

Proprietà di decidibilità basilari

- Se L è decidibile, allora $\forall L' : L' \leq L$ è decidibile.
- Se L' è indecidibile allora lo è anche $\forall L : L' \leq L$ perché altrimenti si avrebbe un assurdo
- Se L è decidibile e L' no allora $L' \not\leq L$ altrimenti assurdo per il primo punto.

Ogni linguaggio decidibile è semplice + Ossia se L' è decidibile allora per ogni L tale che $L \neq \emptyset$ e $L \neq \Sigma^*$, ossia è un linguaggio finito, si ha che

$$L' \leq L$$

In altre parole, possiamo dire che i linguaggi decidibili sono mapping reducibili a qualunque altro linguaggio non banale, quindi sono le più semplici esistenti in altre parole.

Dimostrazione: Dato che L' è decidibile, esiste $g(x), \forall x \in \Sigma^*$ tale che decide se x appartiene o meno a quel linguaggio. Dato che L è finito, esiste un ω che non appartiene e un altro v che appartiene. Allora costruisco la funzione f così: 1. Runno g, se appartiene, mappo a v 2. Se non appartiene mappo a ω. Ez.

Indecidibilità su nastro vuoto **NOTA:** in ogni caso devo vedere se il codice della macchina è valido.

Definendo il linguaggio

$$ETH = \{x \in \Sigma^* : x = code(\mathcal{M}) \text{ e } \mathcal{M} \text{ si ferma su } \varepsilon\}$$

Per dimostrare ciò basta dimostrare che $HALT \leq ETH$ Ossia dobbiamo costruire una funzione che mappa ogni stringa di $HALT$ in una di ETH. Questo è molto semplice, solo definire qualche dettaglio (non banale)

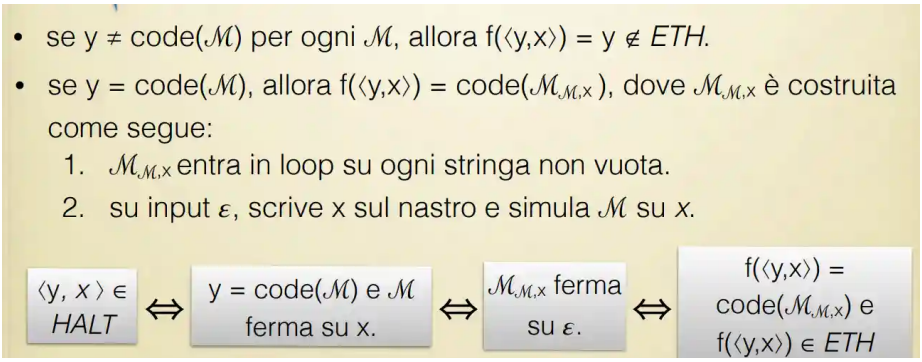

Figure 2: Halting Theorem and Reducibility-20240306125418004

Indicibilità ogni input Definiamo il linguaggio

$$FL = \{x \in \Sigma^* : x = code(\mathcal{M}) \text{ e } \mathcal{M} \text{ ferma su ogni input}\}$$

Anche questo si può dimostrare in maniera simile al precedente, con una mapping reduction. Questo è anche più semplice: Se $\mathcal{M}$ non è il codice di nessuna macchina ritorno quello. Altrimenti: Per ogni input $\langle \mathcal{M}, x \rangle$ costruisco la seguente macchina 1. La macchina nuova prende un input y, la ignora per il

momento, e simula $\langle \mathcal{M}, x \rangle$. 2. Se termina (e quindi appartiene a $HALT$) allora termino anche io ignorando l'input. 3. Altrimenti divergo, e quindi non appartengo. Questa nuova macchina è bona. Quindi funziona l'indicibilità

Quindi Se $\mathcal{M}$ non è codice ho già la risposta. Altrimenti

$\langle y, x \rangle \in HALT \iff$ macchina si ferma su $x \iff \mathcal{M}_{\mathcal{M},x}$ si ferma sempre $\iff f(\langle y, x \rangle) = code(\mathcal{M}_{\mathcal{M},x}) \in FL$

Equivalence problem decidability

$EQ = \{\langle y, x \rangle \in \Sigma^* \times \Sigma^* : x = code(\mathcal{M}), y = code(\mathcal{M}')$ e $\mathcal{M}, \mathcal{M}'$ hanno la stessa funzione parziale$\}$

Anche in questo caso proviamo a ridurci al caso FL. Sempre come prima, per input $\langle \mathcal{M} \rangle$ mi costruisco questa macchina

Prova. E' sufficiente costruire una funzione computabile f tale che:

$$z \in FL \leftrightarrow f(z) \in EQ$$

La definizione di f è come segue. Su argomento z:

- se z ≠ code($\mathcal{M}$) per ogni $\mathcal{M}$, allora f(z) = ⟨z,z⟩ ∉ EQ.
- se z = code($\mathcal{M}$), allora f(z) = ⟨code($\mathcal{M}_1$),code($\mathcal{M}_2$)⟩, dove $\mathcal{M}_1$ e $\mathcal{M}_2$ sono definite come segue:
  - $\mathcal{M}_1$ esegue $\mathcal{M}$ sul suo input e restituisce 1 se $\mathcal{M}$ si ferma, e va in loop altrimenti.
  - $\mathcal{M}_2$ restituisce 1 per ogni input.

Figure 3: Halting Theorem and Reducibility-20240306132502507

La correttezza di questo è un po' più fine, però è giusto. Vedere qui.

Equivalence problem recognizability + Il linguaggio del problema di equivalenza non è nemmeno riconoscibile. Per fare ciò devo ridurre $HALT^-$ a questo EQ. Dimostro la riduzione $HALT \implies EQ^-$

Nota sulla gerarchia Cosa curiosa è che $HALT$ è il suo opposto non sono comparabili. Mentre ci aspetteremmo che HALT sia più semplice. Una altra cosa curiosa è che EQ non è riconoscibile, e nemmeno il suo opposto lo è. Quindi EQ non è nemmeno semi-decidibile.

Dimostrazione. Su ⟨y,x⟩ definiamo f come segue:

- se y ≠ code($\mathcal{M}$) per ogni $\mathcal{M}$, allora prendiamo $\mathcal{M}'$ qualsiasi e consideriamo f(⟨y,x⟩) = ⟨code($\mathcal{M}'$), code($\mathcal{M}'$)⟩.

  Allora f(⟨y,x⟩) ∈ *EQ* e quindi f(⟨y,x⟩) ∉ EQ^-.
- altrimenti y = code($\mathcal{M}$) e consideriamo f(⟨y,x⟩) = ⟨code($\mathcal{M}_1$),code($\mathcal{M}_2$)⟩, dove $\mathcal{M}_1$ e $\mathcal{M}_2$ sono definite come:
  - $\mathcal{M}_1$ esegue $\mathcal{M}$ su x e si ferma se $\mathcal{M}$ si ferma, altrimenti entra in un ciclo.
  - $\mathcal{M}_2$ entra in un ciclo su ogni input.

Figure 4: Halting Theorem and Reducibility-20240306133625876

Figure 5: Halting Theorem and Reducibility-20240517122359749

Turing riducibilità

Definizione di oracolo Dato un linguaggio L e una stringa x, l'oracolo mi dice in tempo finito se $x \in L$.

Definizione Turing-riducibilità

> Dato un L' , questo è Turing riducibile a L, quindi $L' \leq_{TM} L$, se dato un oracolo per L possiamo decidere L'

Mapping reducibility => Turing-riducibilità Possiamo dimostrare in modo semplice che con Turing-riducibilità $HALT$ è riducibile a $HALT^{-}$. Senza problemi. Mentre non posso farlo con Mapping reducibility. Questo mi dice che Turing riducibilità è una proprietà più forte della mapping reducibility, anche se non è propriamente una dimostrazione di questa proprietà.

Baker-Gill-Soloway (1975) Questa sezione è solamente una nota filosofica, ma di poco conto per l'esame.

Prendiamo una macchina di Turing con oracolo, ossia possiamo chiedere se una stringa è parte dell'oracolo in tempo costante (come se fosse un dizionario con accesso diretto).

Esistono oracoli O, O' tali per cui $P = NP$ con una macchina di Turing che usi il primo oracolo, e anche che $P \neq NP$ usando il secondo oracolo. Questo teorema dice che la tecnica di diagonalizzazione di Cantor non può essere usata per risolvere NP = P. Questo dato è inutile per la maggior parte delle cose attuali credo. Oppure TM non sono buoni per risolvere questo problema (ha fatto nascere la branca con i circuiti booleani, non fatta qui).

References

[1] Sipser "Introduction to the Theory of Computation" Cengage Learning 2012