Lezione 9 – indecidibilità dell'Halting problem e riduzioni

Lezione del 21/03/2025

Costruire un problema irrisolvibile

- Come abbiamo visto, 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 prende il nome di **Halting Problem**

- abbiamo dimostrato che L_H è accettabile
- e ora dimostriamo che L_H non è decidibile
  - e questo, come sappiamo bene!, significa che L_H^C non è accettabile

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) ∈ L_H, rigetta (i,x) se (i,x) ∉ L_H
- E da T, ***costruiamo*** T' che rigetta (i,x) se (i,x) ∈ L_H, accetta (i,x) se (i,x) ∉ 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) ∈ 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
  - 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) ∉ L_H, mentre **non termina** se (i,x) ∈ 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 **⟨ q_R , 0, 0, q_R , F⟩ e ⟨ q_R , 1, 1, q_R , F⟩** e rimuovere q_R dall'insieme degli stati finali di T''
- Repetita iuvant: per ogni (i,x) ∈ $\mathbb{N} \times \mathbb{N}$,
  - T''(i,x) accetta se (i,x) ∉ L_H
  - **T''(i,x) non termina se (i,x) ∈ 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) ∈ 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,
- Penultimo passo: siamo quasi pronti a tirare la stoccata finale
- NOTA BENE: poiché (i,x) ∈ $\mathbb{N} \times \mathbb{N}$, 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) ∈ L_H** , ossia se T_i (i) termina,
    e **T(i, i) rigetta se (i, i)∉ L_H** , ossia se T_i (i) non termina
  - **T'(i, i) accetta se (i, i) ∉ L_H** , ossia se T_i (i) non termina,
    e **T'(i, i) rigetta se (i, i)∈ L_H** , ossia se T_i (i) termina
  - **T''(i, i) accetta se (i, i) ∉ L_H** , ossia se T_i (i) non termina,
    e **T''(i, i) non termina se (i, i)∈ 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) ∈ 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,
- Compreso che come input di T, T' e T''possiamo usare una coppia (i,x) ∈ $\mathbb{N} \times \mathbb{N}$ 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) ∉ L_H , ossia se T_i (i) non termina,**
  - **T*(i) non termina se (i, i) ∈ L_H , ossia se T_i (i) termina**

  - PS: se i *non* è la codifica di una macchina di Turing, allora (i, i) ∉ 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

Ancora sulla simulazione "a scatola chiusa"

- Torniamo alla dimostrazione dell'indecidibilità dell'Halting Problem
  - o, equivalentemente, alla non accettabilità del suo complemento
- Siamo partiti supponendo di avere una macchina T in grado di decidere L_H, e poi:
  - senza sapere come era fatta T (senza "aprirla")
  - abbiamo costruito una serie di altre macchine – T, T', T'', T* - che ci hanno portato dove volevamo
- La macchina T' la abbiamo costruita senza sapere come era fatta T
  - e come avremmo potuto saperlo? T manco esiste…
- Che, poi, è quello che facciamo sempre quando utilizziamo, ad esempio una classe delle API di Java
  - per dire, chi di voi, prima di utilizzare il metodo show di JFrame si andrebbe a vedere come è implementato?!
- Questo utilizzo "a scatola nera" di T corrisponde esattamente al concetto di "invocazione di funzione"

Simulare "a scatola chiusa" – (1)

- Ora, quando T' usava T, T' passava (come "parametro") a T il suo stesso input (i,x)
- In generale, possiamo utilizzare una macchina T_0 all'interno di un'altra macchina T_1 in un modo un po' più complesso
  - in effetti, il linguaggio deciso/accettato da T_0 potrebbe anche essere molto diverso da quello deciso/accettato da T_1
  - allora, potrebbe essere necessario "modificare" l'input di T_1 prima di "darlo in pasto" a T_0
- Esempio (scemo): voglio costruire una macchina che decida il linguaggio L_{P12} che contiene tutte (e sole) le parole palindrome di lunghezza pari costituite dai caratteri '1' e '2'
  - caspiterina, quanto assomiglia a L_{PPAL} questo linguaggio L_{P12}, però!
  - quasi quasi, invece di mettermi lì a ri-progettare ex novo un'altra macchina, proverei a ri-utilizzare T_{PPAL} – che decide L_{PPAL}
  - peccato che T_{PPAL} lavori sull'alfabeto {a,b} invece che sull'alfabeto {1,2}
  - Uhm… quasi quasi, provo a trasformare le parole di L_{P12} in parole di L_{PPAL}…

Simulare "a scatola chiusa"– (1)

- Voglio costruire una macchina che decida il linguaggio L_{P12} che contiene tutte (e sole) le parole palindrome di lunghezza pari costituite dai caratteri '1' e '2'
  - voglio costruire una macchina T_{P12} che utilizzi "a scatola nera" T_{PPAL}
  - peccato che T_{PPAL} lavori sull'alfabeto {a,b} invece che {1,2}
  - devo trasformare le parole di L_{P12} in parole di L_{PPAL}...
- Facile: prendo il mio x ∈{1,2}* e procedo così: assumendo $x = x_1 x_2 \dots x_n$, per ogni h =1, 2, ... , n
  - se x_h = '1' allora poniamo y_h = 'a'
  - se x_h = '2' allora poniamo y_h = 'b'
  - infine, poniamo $y = y_1 y_2 \dots y_n$.
- Quello che ho ottenuto è quindi una parola y ∈ {a,b}* che ha le seguenti caratteristiche
  - **se x ∈ L_{P12} allora y ∈ L_{PPAL}**
  - **se x ∉ L_{P12} allora y ∉ L_{PPAL}**

Riduzioni (many-to-one)

- Quello che abbiamo fatto, in realtà, è qualcosa di più di una semplice trasformazione di una parola in un'altra parola
- Abbiamo progettato una funzione f : {1,2}* → {a,b}* tale che
- 1) **f è totale e calcolabile** – ossia,
  - è definita per ogni parola x ∈ {1,2}* e, inoltre,
  - esiste una macchina di Turing di tipo trasduttore T_f tale che, per ogni parola x ∈ {1,2}*, la computazione T_f (x) termina con la parola f(x) ∈ {a,b}* scritta sul nastro di output
- 2) per ogni x ∈ {1,2}* vale che: x ∈ L_{P12} se e solo se f(x) ∈ L_{PPAL}
  - che in matematichese si scrive : $\forall x \in \{1,2\}^* [x \in L_{P12} \leftrightarrow f(x) \in L_{PPAL}]$
- la funzione f si chiama ***riduzione*** da L_{P12} a L_{PPAL}
- e si dice che L_{P12} è ***riducibile*** a L_{PPAL} e si scrive $L_{P12} \preccurlyeq L_{PPAL}$

Riduzioni (many-to-one)

- Quello che abbiamo detto sino ad ora può essere generalizzato
- Dati due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, diciamo che **L_1 è *riducibile* a L_2** e scriviamo **$L_1 \preccurlyeq L_2$** se
- Esiste una funzione $f : \Sigma_1^* \rightarrow \Sigma_2^*$ tale che
- 1) f è totale e calcolabile – ossia,
  - è definita per ogni parola $x \in \Sigma_1^*$ e, inoltre,
  - esiste una macchina di Turing di tipo trasduttore T_f tale che, per ogni parola $x \in \Sigma_1^*$, la computazione $T_f(x)$ termina con la parola $f(x) \in \Sigma_2^*$ scritta sul nastro di output
- 2) per ogni $x \in \Sigma_1^*$ vale che: $x \in L_1$ se e solo se $f(x) \in L_2$
  - **$\forall x \in \Sigma_1^* [x \in L_1 \leftrightarrow f(x) \in L_2]$**
- Siamo al paragrafo 5.5

Decidibilità, accettabilità e riduzioni

- Il concetto di riduzione si rivela molto utile come strumento per dimostrare che un linguaggio è decidibile/accettabile: dato un linguaggio L_3
  - se dimostro che **$L_3 \preccurlyeq L_4$** , per un qualche altro linguaggio L_4 ,
  - se io so che **L_4 è decidibile**
  - allora, posso concludere che anche **L_3 è decidibile**
- Infatti, sia $L_3 \subseteq \Sigma_3^*$ e $L_4 \subseteq \Sigma_4^*$
  - **L_4 è decidibile** : allora esiste una macchina T_4 tale che, per ogni $x \in \Sigma_4^*$, $T_4(x)$ termina in q_A se $x \in L_4$, $T_4(x)$ termina in q_R se $x \notin L_4$
  - **$L_3 \preccurlyeq L_4$** : allora esiste una un trasduttore T_f tale che, per ogni $x \in \Sigma_3^*$, $T_f(x)$ termina con una parola $y \in \Sigma_4^*$ scritta sul nastro di output tale che $y \in L_4$ se $x \in L_3$, e $y \notin L_4$ se $x \notin L_3$
- Ora, costruiamo una macchina T_3 , a 2 nastri, che, con input $x \in \Sigma_3^*$:
  - prima simula $T_f(x)$ scrivendo l'output y sul secondo nastro
  - poi simula $T_4(y)$: se $T_4(y)$ accetta allora anche T_3 accetta, se $T_4(y)$ rigetta allora anche T_3 rigetta,

Decidibilità, accettabilità e riduzioni

- Abbiamo costruito una macchina T_3 , a 2 nastri, che, con input $x \in \Sigma_3^*$:
  - prima simula $T_f(x)$ scrivendo l'output y sul secondo nastro
  - poi simula $T_4(y)$: se $T_4(y)$ accetta allora anche T_3 accetta, se $T_4(y)$ rigetta allora anche T_3 rigetta,
- E a che ci serve?! Beh,
  - poiché è vero che $y \in L_4$ se $x \in L_3$, e $y \notin L_4$ se $x \notin L_3$, allora:
  - se $x \in L_3$ allora $y \in L_4$ e, quindi, $T_4(y)$ accetta; quindi, T_3 accetta le parole in L_3,
  - se , $x \notin L_3$ allora $y \notin L_4$ e, quindi, $T_4(y)$ rigetta; quindi T_3 rigetta le parole che non sono in L_3.
- In conclusione, T_3 decide L_3. Ossia, **L_3 è decidibile**
- Con una dimostrazione simile (che vi fate per esercizio) si dimostra che dato un linguaggio L_3
  - se dimostro che **$L_3 \preccurlyeq L_4$** , per un qualche altro linguaggio L_4 ,
  - se io so che **L_4 è accettabile**
  - allora, posso concludere che anche **L_3 è accettabile**

Decidibilità, accettabilità e riduzioni

- Il concetto di riduzione si rivela molto utile come strumento per dimostrare che un linguaggio è **non** decidibile/non accettabile: dato un linguaggio L_2
  - se dimostro che $L_1 \preccurlyeq L_2$, per un qualche altro linguaggio L_1 ,
  - se io so che **L_1 è non decidibile**
  - allora, posso concludere che anche **L_2 è non decidibile**
- Infatti, sia $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$
  - **se L_2 fosse decidibile** (per assurdo): allora, poiché $L_1 \preccurlyeq L_2$, per quello che abbiamo appena dimostrato (nelle ultime due slides) anche **L_1 sarebbe decidibile** contraddicendo l'ipotesi che **L_1 è non decidibile**
- Con una dimostrazione simile (che vi fate per esercizio) si dimostra che dato un linguaggio L_2
  - se dimostro che $L_1 \preccurlyeq L_2$, per un qualche altro linguaggio L_1 ,
  - se io so che **L_1 è non accettabile**
  - allora, posso concludere che anche **L_2 è non accettabile**

Decidibilità, accettabilità e riduzioni

- In conclusione, il concetto di riduzione può essere utilizzato in due "direzioni":

- 1) riducendo un linguaggio **"sconosciuto"** ad un linguaggio accettabile/decidibile dimostriamo che il linguaggio **"sconosciuto"** è anch'esso accettabile/decidibile
  - ad esempio, dimostrando che $L_{P12} \preccurlyeq L_{PPAL}$ e sapendo che L_{PPAL} è decidibile, abbiamo dimostrato che L_{P12} è decidibile
  - ad esempio, dimostrando che $L_{H0} \preccurlyeq L_{H}$ e sapendo che L_{H} è accettabile, abbiamo dimostrato che L_{H0} è accettabile

- 2) riducendo un linguaggio non accettabile/non decidibile ad un linguaggio **"sconosciuto"** dimostriamo che il linguaggio **"sconosciuto"** è anch'esso non accettabile/non decidibile
  - ad esempio, dimostrando che $L_{H} \preccurlyeq L_{H0}$ e sapendo che L_{H} è non decidibile, abbiamo dimostrato che L_{H0} è decidibile

Riduzioni (many-to-one): esempio

- Facciamo un esempio di dimostrazione di non decidibilità di un linguaggio mediante riduzioni
  - è una modifica dell'esempio a pag. 7 della dispensa 5 – non tanto facile
- Consideriamo il linguaggio

  L_{H0} = { i ∈ ℕ: i è la codifica di una macchina di Turing T_i e T_i (0) termina}

  - ossia, i ∈ L_{H0} se i è la codifica di una macchina di Turing T_i e la computazione di T_i con input la parola costituita dal solo '0' termina
- caspita quanto assomiglia a L_H questo L_{H0}!
  - In effetti, L_{H0} è un sottoinsieme di L_H : L_{H0} contiene solo coppie (i,0)
  - questo si esprime dicendo che L_{H0} è una *restrizione* di L_H
  - e implica (ovviamente!) che un algoritmo che "si occupa" di L_H può "occuparsi allo stesso modo" anche di L_{H0}
  - e, quindi, la macchina di Turing che accetta L_H accetta anche L_{H0}
  - ma L_{H0} sembra "più facile" di L_H : quindi, magari, L_{H0} è decidibile...
- E, invece, sorprendentemente, non è vero che L_{H0} è "più facile" di L_H !

Riduzioni (many-to-one): esempio

- Non è vero che L_{H0} è "più facile" di L_H !
  - E come si fa a dimostrarlo?
- Riducendo L_H a L_{H0}!
  - ossia, individuando una funzione totale e calcolabile f che trasforma parole di L_H in parole di L_{H0} e parole non in L_H in parole non in L_{H0}
- Ricordiamo:
  L_H = {(i,x) ∈ ℕ×ℕ: i è la codifica di una macchina di Turing T_i e T_i (x) termina}
  e **L_{H0} = { i ∈ ℕ: i è la codifica di una macchina di Turing T_i e T_i (0) termina }**
- Consideriamo una qualsiasi coppia **(i,x)** ∈ ℕ×ℕ , con **x = $x_1x_2...x_n$**, e da essa deriviamo un intero **k = $k_{(i,x)}$** = f(**i,x**) nel modo seguente:
- a) se i non è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $M_{(i,x)}$ che entra in loop qualunque sia il suo input
- b) se i è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $M_{(i,x)}$ che, con input 0, simula <u>la sola computazione T_i (x)</u>
  - vedi prossima pagina
- c) $k_{(i,x)}$ è la codifica di $M_{(i,x)}$, ossia f(**i,x**) = $k_{(i,x)}$

Riduzioni (many-to-one): esempio

- b) se i è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $\mathbf{M_{(i,x)}}$ che
  - 1) nello stato iniziale $\mathbf{q^1}$: se legge 0 sul nastro allora scrive $\boldsymbol{x_1}$ e entra nello stato $\mathbf{q^2}$ muovendosi a destra, altrimenti rigetta
  - 2) nello stato $\mathbf{q^2}$: se legge ▫ sul nastro, scrive $\boldsymbol{x_2}$ e entra nello stato $\mathbf{q^3}$ muovendosi a destra, altrimenti rigetta
  - ... (eccetera eccetera) ...
  - n) nello stato $\mathbf{q^n}$: se legge ▫ sul nastro, scrive $\boldsymbol{x_n}$, riposiziona la testina sul primo carattere dell'input e entra nello stato iniziale $\mathbf{q_0}$ di $\mathbf{T_i}$, altrimenti rigetta
  - a questo punto, $\mathbf{M_{(i,x)}}$ inizia a simulare $\mathbf{T_i(x)}$ a scatola aperta
- Osservazione 1. Il numero degli stati $\mathbf{M_{(i,x)}}$ dipende da **x** e potrebbe sembrare non costante: non è così!
  - Innanzi tutto, **x** non è input per $\mathbf{M_{(i,x)}}$ (che si attende come input un solo carattere 0).
  - Poi, ricordiamo che abbiamo considerato una coppia **(i,x)** e solo dopo averla fissata abbiamo costruito $\mathbf{M_{(i,x)}}$. In altre parole, per ogni **x** abbiamo una $\mathbf{M_{(i,x)}}$, ossia **x** è costante per $\mathbf{M_{(i,x)}}$

Riduzioni (many-to-one): esempio

- Consideriamo una qualsiasi coppia **(i,x)** ∈ $\mathbb{N}\times\mathbb{N}$, con **x =** $x_1x_2...x_n$, e da essa deriviamo un intero $\mathbf{k_{(i,x)}}$ = f(**i**,**x**) nel modo seguente:
  - b) se i è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $\mathbf{M_{(i,x)}}$ che
    - 1) nello stato iniziale $\mathbf{q^1}$: se legge 0 sul nastro allora scrive x_1 e entra nello stato $\mathbf{q^2}$ muovendosi a destra, altrimenti entra in loop
    - 2) nello stato $\mathbf{q^2}$: se legge sul nastro, scrive x_2 e entra nello stato $\mathbf{q^3}$ muovendosi a destra, altrimenti rigetta
    - ... (eccetera eccetera) ...
    - n) nello stato $\mathbf{q^n}$: se legge sul nastro, scrive x_n , riposiziona la testina sul primo carattere dell'input e entra nello stato iniziale $\mathbf{q_0}$ di $\mathbf{T_i}$, altrimenti rigetta
    - a questo punto, $\mathbf{M_{(i,x)}}$ inizia a simulare $\mathbf{T_i(x)}$ a scatola aperta
- Sappiamo ben calcolare $\mathbf{M_{(i,x)}}$:
  - sia nel caso a), in cui l'intero **i** che non è la codifica di una macchina di Turing
  - sia nel caso b) in cui possiamo effettivamente costruire $\mathbf{M_{(i,x)}}$: avendo l'intero **i** che è la codifica (che ben conosciamo!) di $\mathbf{T_i}$, possiamo costruire le quintuple di $\mathbf{M_{(i,x)}}$ utilizzando tale codifica.
- Quindi, f è totale e calcolabile.

Riduzioni (many-to-one): esempio

- Fissando una qualsiasi **(i,x)** ∈ ℕ×ℕ, con **x =** $x_1x_2...x_n$, abbiamo costruito $\mathbf{M_{(i,x)}}$:
  - a) se i non è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $\mathbf{M_{(i,x)}}$ che entra in loop qualunque sia il suo input
  - b) se i è la codifica di una macchina di Turing, allora abbiamo costruito una macchina di Turing $\mathbf{M_{(i,x)}}$ che, con input 0, termina se e solo se $\mathbf{T_i(x)}$ termina
  - c) $\mathbf{k_{(i,x)}}$ è la codifica di $\mathbf{M_{(i,x)}}$
- L_{H0} = { i ∈ ℕ: i è la codifica di una macchina di Turing T_i e T_i (0) termina}
- Per ogni **(i,x)** ∈ ℕ×ℕ , poiché $\mathbf{k_{(i,x)}}$ è la codifica di se $\mathbf{M_{(i,x)}}$,
  - ossia $\mathbf{T_{k_{(i,x)}}} = \mathbf{M_{(i,x)}}$
- allora $\mathbf{k_{(i,x)}}$ ∈ L_{H0} se e solo se $\mathbf{T_{k_{(i,x)}}}$ (0) = $\mathbf{M_{(i,x)}}$(0) termina
- resta da capire in quali casi $\mathbf{M_{(i,x)}}$(0) termina

Riduzioni (many-to-one): esempio

- Fissando una qualsiasi **(i,x)** $\in \mathbb{N}\times\mathbb{N}$, con $\mathbf{x = x_1x_2\ldots x_n}$, abbiamo costruito $\mathbf{M_{(i,x)}}$:
  - a) se i non è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $\mathbf{M_{(i,x)}}$ che entra in loop qualunque sia il suo input
  - b) se i è la codifica di una macchina di Turing, allora abbiamo costruito una macchina di Turing $\mathbf{M_{(i,x)}}$ che, con input 0, termina se e solo se $\mathbf{T_i(x)}$ termina
  - c) $\mathbf{k_{(i,x)}}$ è la codifica di $\mathbf{M_{(i,x)}}$
- Per ogni **(i,x)** $\in \mathbb{N}\times\mathbb{N}$, se $\mathbf{M_{(i,x)}}(0)$ termina allora $\mathbf{M_{(i,x)}}$ è stata costruita eseguendo il passo b)
  - perché la macchina costruita eseguendo il passo a) non termina mai!
  - e quindi **i** è la codifica di una macchina di Turing $\mathbf{T_i}$
- e inoltre, $\mathbf{T_i(x)}$ termina,

- Ricapitolando: se $\mathbf{k_{(i,x)}} \in L_{H0}$ allora $\mathbf{T_{k_{(i,x)}}}(0) = \mathbf{M_{(i,x)}}(0)$ termina
- e se $\mathbf{T_{k_{(i,x)}}}(0) = \mathbf{M_{(i,x)}}(0)$ termina allora **i** è la codifica di una macchina di Turing e $\mathbf{T_i(x)}$ termina
- ossia: se $\mathbf{k_{(i,x)}} \in L_{H0}$ allora **(i,x)** $\in L_H$

Riduzioni (many-to-one): esempio

- Fissando una qualsiasi **(i,x)** ∈ ℕ×ℕ, con **x =** $x_1x_2...x_n$, abbiamo costruito $\mathbf{M_{(i,x)}}$:
  - a) se i non è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $\mathbf{M_{(i,x)}}$ che entra in loop qualunque sia il suo input
  - b) se i è la codifica di una macchina di Turing, allora abbiamo costruito una macchina di Turing $\mathbf{M_{(i,x)}}$ che, con input 0, termina se e solo se $\mathbf{T_i(x)}$ termina
  - c) $\mathbf{k_{(i,x)}}$ è la codifica di $\mathbf{M_{(i,x)}}$

- Per ogni **(i,x)** ∈ ℕ×ℕ , se $\mathbf{M_{(i,x)}}(0)$ non termina allora
  - o $\mathbf{M_{(i,x)}}$ è stata costruita eseguendo il passo a) e quindi **i** non è la codifica di una macchina di Turing $\mathbf{T_i}$
  - oppure $\mathbf{M_{(i,x)}}$ è stata costruita eseguendo il passo b), e quindi **i** è la codifica di una macchina di Turing $\mathbf{T_i}$, ma $\mathbf{T_i(x)}$ non termina,

- Ricapitolando: se $\mathbf{k_{(i,x)}} \notin L_{H0}$ allora $\mathbf{T_{k_{(i,x)}}}(0) = \mathbf{M_{(i,x)}}(0)$ non termina
- e se $\mathbf{T_{k_{(i,x)}}}(0) = \mathbf{M_{(i,x)}}(0)$ non termina allora **(i,x)** $\notin L_H$
- ossia, se $\mathbf{k_{(i,x)}} \notin L_{H0}$ allora allora **(i,x)** $\notin L_H$

Riduzioni (many-to-one): esempio

- Ricapitolando, abbiamo considerato una qualsiasi coppia **(i,x)** ∈ $\mathbb{N}\times\mathbb{N}$, con **x** = $x_1x_2\ldots x_n$, e da essa abbiamo derivato un intero $\mathbf{k}_{(i,x)}$ nel modo seguente:
  - a) se i non è la codifica di una macchina di Turing, allora abbiamo costruito una macchina di Turing $\mathbf{M}_{(i,x)}$ che entra in loop qualunque sia il suo input
  - b) se i è la codifica di una macchina di Turing, allora abbiamo costruito una macchina di Turing $\mathbf{M}_{(i,x)}$ che, con input 0, termina se e solo se $\mathbf{T_i(x)}$ termina
  - c) abbiamo calcolato $\mathbf{k}_{(i,x)}$ come la codifica di $\mathbf{M}_{(i,x)}$ ponendo, $\mathbf{k}_{(i,x)}$= f(**i**,**x**).
- 1) Poiché abbiamo descritto il procedimento che permette di calcolare $\mathbf{k}_{(i,x)}$ a partire da ogni coppia **(i,x)**, allora f è totale e calcolabile
- 2) per ogni **(i,x)** ∈ $\mathbb{N}\times\mathbb{N}$ vale che: **(i,x)** ∈ L_H se e solo se $\mathbf{k}_{(i,x)}$= f(**i**,**x**) ∈ L_{H0}
- Dunque, $\mathbf{L_H \preccurlyeq L_{H0}}$ e ciò dimostra che $\mathbf{L_{H0}}$ non è decidibile!