Teoria della computazione

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

6 aprile 2022

Cosa possiamo calcolare?

Quali problemi siamo in grado di risolvere?

- Con un certo tipo di formalismo di calcolo?
- In assoluto?
- La seconda domanda appare molto generale:
    - Cosa si intende per "problema"? Riordinare la stanza? Trovare un file sul disco fisso? Trattenere il respiro per 10 minuti?
    - Quante e quali macchine dobbiamo considerare per rispondere "in assoluto"?
    - Come astraiamo dalla specifica abilità del solutore?
    - E dai mezzi impiegati per risolvere il problema?

Una prima inquadratura

Come formalizzare un "problema informatico"?

- Abbiamo formalizzato un "problema informatico" usando il concetto di linguaggio
- Formuliamo un calcolo come il problema di capire se $x \in L$ o di calcolare $y = \tau(x)$
- Queste due formulazioni possono essere unificate riconducendo una all'altra
  - So calcolare $y = \tau(x)$, voglio risolvere $x \in L$: definisco $\tau(x) = 1 \Leftrightarrow x \in L$ e $\tau(x) = 0 \Leftrightarrow x \notin L$
  - $\mathcal{M}$ risolve $x \in L$: definisco $L_\tau = \{x \ddagger y | y = \tau(x)\}$, poi per tutte le possibili stringhe y chiedo a $\mathcal{M}$ se $x \ddagger y \in L_\tau$. Se $\tau(x)$ è definita, prima o poi la macchina risponderà positivamente (probabilmente più poi che prima, ma per ora non ci interessa l'efficienza).

Con quale macchina calcolare?

Come formalizzare un "problema informatico"?

- Esistono moltissimi formalismi di calcolo, moltissimi altri possono essere inventati: quale scegliere?
  - A seconda della scelta, ottengo risultati come: "$a^n b^n | n > 0$ è riconosciuto da un AP e una MT ma non da un FSA"
- Riflettendo sulla MT si nota come non sia facile costruire un meccanismo con capacità di calcolo maggiori (=che risolva più problemi)
  - Aggiungere nastri, testine, dimensioni al nastro non cambia
  - Essa emula qualunque meccanismo di calcolo usiamo in pratica

Tesi di Church-Turing

La MT è tutto quello che ci serve

- Nel 1933 Gödel e Herbrandt individuano un insieme di funzioni sugli interi che appaiono definire ciò che può essere calcolato "a mano, con carta e penna"
- Nel 1936, Alonso Church definisce un altro sistema basato su funzioni ricorsive, il λ-calcolo, anch'esso in grado di descrivere tutte le funzioni "calcolabili operativamente"
- Nel 1936 Turing definisce quella che è la MT a nastro singolo sempre nell'intento di fornire un formalismo per rappresentare tutto ciò che è "effettivamente calcolabile"
- Turing e Church dimostrano che i tre formalismi citati sono equivalenti: definiscono lo stesso insieme di problemi
- Tesi di Church-Turing: *Tutti i problemi calcolabili operativamente sono descritti da una MT!*

Un inquadramento completo

Dalla tesi di Church-Turing in avanti

- La domanda "quali sono i problemi che possiamo risolvere automaticamente o algoritmicamente?" è ben posta ora
  - n.b.: è vero, la tesi di Church-Turing non è formalmente dimostrata, ma negli ultimi 80 anni non ha avuto controesempi
- Turing chiama "effectively computable" una funzione che può essere calcolata da una procedura eseguita da una macchina, senza necessità di intervento esterno, e che dà risultato in tempo finito nella sua tesi di dottorato[a]
- Esistono problemi che *non* si possono risolvere algoritmicamente?
  - Come è possibile determinare se questo è il caso?

[a] `https://dx.doi.org/10.1112/plms/s2-45.1.161`, pag.6

Enumerazione algoritmica

Enumerare un insieme

- Enumerazione $\mathcal{E}$ di un insieme = corrispondenza biunivoca tra i suoi elementi e quelli di $\mathbb{N}$
- Enumerazione algoritmica: $\mathcal{E}$ è "effectively computable": esiste un algoritmo (o una MT) che la calcola
- Enum. algoritmica di $L = \{a^*b^*\}$, $\mathcal{E} : L \to \mathbb{N}$: "etichetta" le stringhe in ordine crescente di lunghezza. Per stringhe della stessa lunghezza, "etichettale" in ordine lessicografico
  - $\varepsilon \mapsto 0, a \mapsto 1, b \mapsto 2, aa \mapsto 3, ab \mapsto 4, bb \mapsto 5, aaa \mapsto 6, aab \mapsto 7 \ldots$

Primo fatto sulle MT

- Le MT sono *algoritmicamente enumerabili* (dimostriamolo)

Enumerazione algoritmica delle MT

Premesse senza perdita di generalità

- Consideriamo le MT a nastro singolo, con alfabeto $\mathbf{A} = \{0, 1, \text{ƀ}\}$ e $\mathbf{Q} = \{q_0, q_1\}$
- Osserviamo quali sono le possibili δ di queste MT

	0	1
q_0	$\perp$	$\perp$
q_1	$\perp$	$\perp$

MT_1

	0	1
q_0	$\langle q_0, 0, S\rangle$	$\perp$
q_1	$\perp$	$\perp$

MT_2

	0	1
q_0	$\langle q_0, 1, S\rangle$	$\perp$
q_1	$\perp$	$\perp$

MT_3

$\cdots$

- Posso contare il numero di δ possibili e sapere quante MT a 2 stati/2 lettere di alfabeto esistono

Enumerazione algoritmica delle MT

Enumerazione delle macchine

- In generale, ho $|\mathbf{C}|^{|\mathbf{D}|}$ funzioni $f : \mathbf{D} \to \mathbf{C}$
- Facendo i conti con $\delta : \mathbf{Q} \times \mathbf{A} \to (\mathbf{Q} \times \mathbf{A} \times \{R, S, L\}) \cup \{\perp\}$ abbiamo che, con $|\mathbf{Q}| = 2, |\mathbf{A}| = 3$ ci sono $(2 \cdot 3 \cdot 3 + 1)^{2 \cdot 3} = 19^6$ possibili δ per MT a 2 stati/2 lettere
- Scelgo un ordine arbitrario per l'insieme $\{\mathsf{MT}_0, \ldots, \mathsf{MT}_{19^6-1}\}$
- Allo stesso modo ordino le $(3 \cdot 3 \cdot 3 + 1)^{3 \cdot 3} = 28^9$ MT-3-stati
- Numerando gli insiemi uno dopo l'altro ottengo un'enumerazione $\mathcal{E} : \mathsf{MT} \to \mathbb{N}$
- $\mathcal{E}$ è algoritmica: posso scrivere un programma (come è fatto?) che, data δ mi fornisce il suo numero.
- $\mathcal{E}(\mathcal{M})$ è detto numero di Gödel di $\mathcal{M}$, $\mathcal{E}(\cdot)$ è detta gödelizzazione

Convenzioni aggiuntive

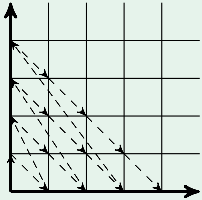
Convenzioni sul calcolo

- Problema = calcolo di una funzione $f : \mathbb{N} \rightarrow \mathbb{N}$
- f_i = funzione calcolata dalla i-esima MT
- N.B.: $f_i(x) = \perp$ se $\mathcal{M}_i$ non si ferma quando riceve in ingresso x
- Convenzione: $f_i(x) = \perp$ se e solo se $\mathcal{M}_i$ non si ferma quando riceve in ingresso x
  - Data una generica $\mathcal{M}_i$ basta fare in modo che proceda all'infinito (e.g. sposti all'infinito la testina verso sx) se non calcola un valore significativo per $f_i(x)$
  - La convenzione consente di non dover trattare gli stati finali separatamente separatamente

Macchina di Turing Universale

Calcolare una generica MT con *un*'altra

- Esiste (almeno) una *Macchina di Turing universale (MTU)*
  - è la MT che calcola $g(i, x) = f_i(x)$
- La MTU non sembra essere dello stesso tipo delle altre $\mathcal{M}_i$ perchè $f_i(\cdot)$ è funzione di una variabile, $g(\cdot, \cdot)$ di due
- Proviamo il contrario ricordando che $\mathbb{N} \times \mathbb{N}$ è enumerabile



Mappa di Cantor

d associa (x, i) a $n \in \mathbb{N}$

$$d(x, i) = \frac{(x+i)(x+i+1)}{2} + x$$

- N.B. d è invertibile: dato n ottengo una e una sola (x, y)

Macchina di Turing Universale

Calcolare una generica MT con *un*'altra

- Posso cambiare la codifica di $g(i,x)$, realizzando una $\hat{g}(n) = g(d^{-1}(n))$ (d e d^{-1} sono facilmente computabili)
- Schema di una MTU che calcola $\hat{g}$
  - Dato n calcolo $d^{-1}(n) = \langle i, x \rangle$
  - Calcolo $\mathcal{E}^{-1}(i)$, memorizzo la δ di M_i sul nastro della MTU, separata da $\ddagger$

ƀ	ƀ	ƀ	$\ddagger$	q_0	0	q_1	0	L	$\ddagger$	q_0	1	q_2	1	R	$\ddagger$	ƀ	ƀ	ƀ

  - Uso un'altra porzione di nastro per simularne la configurazione

ƀ	ƀ	ƀ	$\triangle$	1	0	1	0	1	0	q_1	0	1	1	1	$\triangle$	ƀ	ƀ	ƀ

  - N.B.: I simboli speciali $\ddagger, \triangle$ vengono codificati in binario
- La MTU lascia sul nastro $f_i(x) \Leftrightarrow M_i$ termina la computazione su x

A confronto con la pratica

MT, MTU, ASIC e Calcolatori programmabili

- Abbiamo visto che una MT è un modello molto semplice di calcolatore in qualche senso analoga ad una macchina con programma cablato (un ASIC)
- Una MTU è l'analogo di un calcolatore programmabile
  - Il numero di Göedel i agisce da "codice" del programma, l'ingresso x sono i dati
- Una MTU (e anche la sua implementazione[a]) può essere molto semplice: ne esiste una con 4 stati e 6 simboli
- Nulla vieta che i sia il numero di Göedel di un'altra MTU e che x contenga quindi il numero di Gödel e i dati da far girare nella MTU "emulata": modello di una macchina virtuale

[a] https://en.wikipedia.org/wiki/One_instruction_set_computer

Un richiamo di teoria degli insiemi

Il teorema di Cantor: $|S| < 2^{|S|}$

- Dimostriamo che dato un insieme S, $|S| < |\wp(S)|$
  - Ci saranno utili sia il risultato, sia la tecnica dimostrativa
  - N.B. il teorema è valido anche se $|S|$ non è finito

Dimostrazione.

Dim. che esiste una $f : S \rightarrow \wp(S)$ iniettiva, ma non una suriettiva.

$\exists$ **Iniettiva.** Esempio di f iniettiva: f mappa $x \in S$ in $\{x\} \in \wp(S)$.

$\nexists$ **Suriettiva.** Considero $T = \{x \in S, x \notin f(x)\}$. Per assurdo. Hp: esiste f sur., quindi $T = f(x)$.

$$x \in T \Leftrightarrow x \in f(x) \text{ (per hp } T = f(x)\text{)}$$
$$x \in T \Leftrightarrow x \notin f(x) \text{ (data la definizione di } T\text{)}$$

ma, per qualunque $x \in S$ o è vero $x \in T$ o è vero $x \notin T$ ↯ ■

Quanti e quali problemi sono risolvibili algoritmicamente?

Cominciamo dal "quanti"

- Sappiamo che $f : \mathbb{N} \to \mathbb{N} \supseteq f : \mathbb{N} \to \{0,1\}$ e quindi $|f : \mathbb{N} \to \mathbb{N}| \supseteq |f : \mathbb{N} \to \{0,1\}|$. Sappiamo anche che $|f : \mathbb{N} \to \{0,1\}| = 2^{\aleph_0}$ ($= |\mathbb{R}|$ sotto l'ipotesi del continuo)
- Quante sono le funzioni calcolabili $f_i : \mathbb{N} \to \mathbb{N}, i \in \mathbb{N}$? $|\mathbb{N}| = \aleph_0$ (abbiamo dimostrato che sono enumerabili)
  - N.B. $\mathcal{E} : \{\mathcal{M}_i\} \to \mathcal{N}$ induce una $\hat{\mathcal{E}} : \mathcal{N} \to \{f_y\}$ non biunivoca, ma ci basta per dire quanto sopra
- Esistono quindi almeno $2^{\aleph_0}$ funzioni (sui naturali), ma solo $\aleph_0$ MT $\Rightarrow$ la "stragrande" maggioranza delle funzioni (problemi) non è risolvibile algoritmicamente!

Abbiamo perso molto?

Problemi definibili

- Domanda: "quanti sono i problemi *definibili*?"
- Definiamo un problema con una frase in un qualche linguaggio:
  - $f(x) = x^3 + x + 1$
  - $f(x) = \sum_{i=0}^{100} ix^2$
  - "il numero che moltiplicato per il diametro di una circonferenza dà la sua lunghezza"
- Un linguaggio è sottoinsieme di $\mathbf{A}^*$, che è numerabile
- L'insieme dei problemi definibili è quindi numerabile come quello dei risolvibili!
- Dato che Problemi risolvibili $\subseteq$ Problemi definibili (una MT definisce una funzione oltre a calcolarla)
  - Possiamo sperare che l'inclusione non sia propria...

Quali sono i problemi risolvibili?

Il problema della terminazione del calcolo

- Problema di carattere estremamente pratico:
  - Costruisco un programma
  - Lo eseguo su dei dati in ingresso
  - So che il programma potrebbe non terminare la sua esecuzione (in gergo, "va in loop")
  - Posso determinare, con un metodo generale, se e quando questo accade?
- Rifrasando in termini equivalenti di MT:
  - Consideriamo la funzione $g(i, x) = \begin{cases} 1 & \text{se } f_i(x) \neq \perp \\ 0 & \text{se } f_i(x) = \perp \end{cases}$
  - Esiste una MT che calcola g?

Il problema della terminazione del calcolo

... non può essere risolto

- **NON esiste** una MT che calcola la g appena definita!
- Di conseguenza, nella pratica:
  - Non esiste un compilatore che possa dirci che il nostro programma andrà in loop su un dato input
  - Non possiamo costruire l'antivirus definitivo che sia in grado di capire a priori se un programma è malevolo
  - Non possiamo "creare un programma per tentativi ciechi" controllando solo a posteriori che sia quello corretto
- Per contro, dire se ci siam dimenticati una parentesi in un linguaggio di programmazione (ben progettato) è fattibile: basta un AP per risolvere il problema

Dimostrazione

Note tecniche

- Tecnica diagonale simile a quella usata da Cantor per dimostrare che $|\mathbf{S}| < 2^{|\mathbf{S}|}$ in una dimostrazione per assurdo

Dimostrazione

- Hp (nego la tesi): esiste ed è computabile

$$g(i,x) = \begin{cases} 1 & \text{se } f_i(x) \neq \perp \\ 0 & \text{se } f_i(x) = \perp \end{cases}$$

- É quindi computabile anche $h(x) = \begin{cases} 1 & \text{se } g(x,x) = 0 \\ \perp & \text{altrimenti} \end{cases}$
- Se h è computabile, esiste x_h tale che $f_{x_h} = h$

Dimostrazione

Dimostrazione - seguito

- A questo punto cosa succede se calcolo $h(x_h)$?
  - Caso $h(x_h) = 1$: Dato che $f_{x_h} = h$, si ha che $f_{x_h}(x_h) = 1$. Tuttavia, per definizione di h abbiamo che $g(x_h, x_h) = 0$, ma quindi, per definizione di g, $f_{x_h}(x_h) = \perp$ ↯
  - Caso $h(x_h) = \perp$: Dato che $f_{x_h} = h$, si ha che $f_{x_h}(x_h) = \perp$. Tuttavia, per definizione di h abbiamo che $g(x_h, x_h) = 1$, ma quindi, per definizione di g, $f_{x_h}(x_h) \neq \perp$ ↯
- Otteniamo una contraddizione in entrambi i casi. ■

Una visione intuitiva della dimostrazione

Se ho un programma $g(i, x)$ in grado di dire se un generico altro programma f_i termina, posso usarlo per costruirne un programma h che fa sempre sbagliare $g(i, x)$ nel comprendere se h termina.

Generalizzazioni e specializzazioni

Un lemma del teorema precedente

- $h'(x) = \begin{cases} 1 & \text{se } g(x,x) = 1 \text{ (se } f_x(x) \neq \perp) \\ 0 & \text{se } g(x,x) = 0 \text{ (se } f_x(x) = \perp) \end{cases}$ non è calcolabile
- N.B. non può essere ricavato come conseguenza del teorema precedente (che copre un caso più generale, dove i e x possono esser diversi)
- Se un dato problema P non è calcolabile un suo caso particolare (= stesso problema su una restrizione del dominio) potrebbe esserlo
  - ma una sua generalizzazione (= stesso problema con dominio ampliato) non lo è mai
- Se un dato problema P è calcolabile, una sua generalizzazione potrebbe non esserlo (ma una sua specializzazione è sempre calcolabile)

Un ulteriore importante problema indecidibile

Calcolare se una funzione calcolabile è totale

$$k(i) = \begin{cases} 1 & \text{se } \forall x \in \mathbb{N}, f_i(x) \neq \perp \ (f_i(\cdot) \text{ totale}) \\ 0 & \text{se } \exists x \in \mathbb{N}, f_i(x) = \perp \ (f_i(\cdot) \text{ non totale}) \end{cases} \quad \text{non è calcolabile}$$

- Problema simile *ma diverso* dal precedente: voglio sapere se il calcolo di f_i termina per *tutti* i suoi input
  - Saper dire, per un x fissato, se $f_i(x) \neq \perp$ non mi consente di dire sicuramente se $f_i(\cdot)$ è totale: posso provare un po'di x, ma potrei non trovare quello critico per cui $f_i(x) = \perp$
  - Viceversa, potrei essere in grado di dire che una data f_i non è totale, anche se non so decidere se $f_i(x) \stackrel{?}{=} \perp$ per un x fissato
- In pratica: questo è il problema di determinare se, dato un programma, termina su un qualsiasi dato in ingresso.

Dimostrazione

Ancora per assurdo + diagonale

- Hp assurdo: $k(i) = \begin{cases} 1 & \text{se } \forall x \in \mathbb{N}, f_i(x) \neq \perp \\ 0 & \text{se } \exists x \in \mathbb{N}, f_i(x) = \perp \end{cases}$ è calcolabile
  - Per come è definita, k è anche totale
- Definisco $g(x) = w =$ numero di Gödel della x-esima MT che calcola una funzione totale (in pratica $g(\cdot)$ enumera le MT che calcolano funzioni totali)
- Se k è calcolabile e totale, lo è anche g; posso infatti:
  - Calcolare $k(x)$ al crescere di x. Trovato $x_0 | k(x_0) = 1$ pongo $g(0) = x_0$ e riprendo a calcolare k da $x_0 + 1$ in avanti
  - Trovato $x_1 | k(x_1) = 1$ pongo $g(1) = x_1$, e riprendo ...
  - La procedura è algoritmica, e calcola $g(x)$ per ogni x essendo le funzioni totali (numerabilmente) infinite

Dimostrazione

Continua dalla slide precedente

- g è strettamente monotona: $g(x) = w_x < w_{x+1} = g(x+1)$
- g^{-1} è quindi ancora una funzione strettamente monotona ma non totale ($g^{-1}(w)$ è definita solo se w è il n. di Gödel di una funzione totale)
- Definisco $h(x) = f_{g(x)}(x) + 1 = f_w(x) + 1$. Sappiamo che $f_w(x)$ è calcolabile e totale (per def. di g) $\Rightarrow$ anche h lo è
- Esiste un $\bar{w} \mid f_{\bar{w}}(\cdot) = h(\cdot)$. Dato che h è totale, sicuramente $g^{-1}(\bar{w}) \neq \perp$: poniamo $g^{-1}(\bar{w}) = \bar{x}$
- Che forma ha $h(\bar{x})$?
  Per definizione di h, $h(\bar{x}) = f_{g(\bar{x})}(\bar{x}) + 1 = f_{\bar{w}}(\bar{x}) + 1$,
  ma, siccome $h(\cdot) = f_{\bar{w}}(\cdot)$ abbiamo anche $h(\bar{x}) = f_{\bar{w}}(\bar{x})$ ↯ ■

Problema risolvibile $\neq$ problema risolto

Sapere che la soluzione esiste $\neq$ essere in grado di determinarla

- Spesso è possibile dare una dimostrazione non costruttiva: dimostro che una soluzione esiste, ma non mostro come ricavarla in generale
    - E' il caso tipico di problemi che non hanno soluzione nota in forma chiusa
- Nel contesto della calcolabilità: posso sapere che *esiste* una MT che risolve il problema, anche se non sono in grado di dire *quale* sia tra le possibili
- Alcuni esempi con problemi a risposta binaria:
    - É vero che una partita a scacchi "perfetta" termina in parità?
    - É vero che ogni intero > 2 è la somma di due primi?

Problema risolvibile $\neq$ problema risolto

Un po' di formalizzazione

- In un problema con risposta binaria so a priori che la risposta è "sì" o "no" anche se non so quale sia
- Ricordando che per noi problema = funzione e risolvere un problema = calcolare una funzione, che funzioni associamo ai problemi precedenti?
  - Codificando "sì" $= 1$, "no" $= 0$, i due problemi precedenti sono espressi dalle una tra le due funzioni $f_1(x) = 1$, $f_0(x) = 0$
- Dunque sono risolvibili problemi come:
  - Dire se $g(10, 20) = 1$, ossia se $f_{10}(20) \neq \perp$
  - Dire se $\forall x \in \{10, 11, 12\}, g(x, 20) = 1$
- Sono tutti problemi con risposta "sì" o "no": possiamo non riuscire a determinare quale tra le due sia corretta, ma sicuramente sono calcolabili

Un caso più interessante

Calcolare le cifre di π

- $f(x) = x$-esima cifra dell'espansione decimale di π
  - f è calcolabile, è noto più di un algoritmo (MT) che la calcola
- Date le capacità attuali di calcolare f ci domandiamo se $g(x)$ sia calcolabile: $g(x) = 1$ se nell'espansione di π sono presenti x cifre 5 consecutive, 0 altrimenti
- Calcolando la sequenza $f(0) = 3, f(1) = 1, f(2) = 4,$ $f(3) = 1, f(4) = 5, f(5) = 9$ sappiamo che $g(1) = 1$
- A priori, i valori di $g(x)$ saranno distribuiti tra 0 e 1 in un modo deterministico, ma non predicibile (al meglio delle nostre conoscenze)

$g(x)$ è calcolabile?

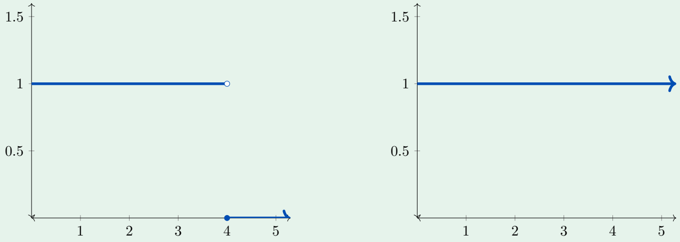
Un approccio enumerativo

- Data x, se $g(x) = 1$ lo scoprirò sempre (basta calcolare abbastanza cifre di π)
- Se $g(x) = 0$ non posso esserne certo semplicemente calcolando un grande numero di cifre di π: la sequenza che cerco potrebbe essere appena dopo!
- Consideriamo la congettura: "Qualsiasi sia x esiste una sequenza di x cifre 5 nell'espansione di π?"
- Se fosse vera, $g(x)$ sarebbe calcolabile banalmente (sarebbe costante)
- In conclusione, date le conoscenze attuali, non sappiamo dimostrare né che g sia calcolabile, né che non lo sia

Un primo esercizio

Una "lieve" modifica a g

- $h(x)$: $h(x) = 1$ se nell'espansione di π ci sono almeno x cifre 5 consecutive, 0 altrimenti. $h(x)$ è computabile?
- Prima osservazione : se $h(x) = 1$ per una data x, allora $\forall y \leq x, h(y) = 1$
- Deduciamo che $h(x)$ può essere fatta in due modi



Un primo esercizio

Analizziamo $h(x)$

- $h(x)$ appartiene quindi sicuramente all'insieme delle funzioni:
$$\{h_{\bar{x}} | \forall x \leq \bar{x},\ h_{\bar{x}}(x) = 1, \forall x > \bar{x},\ h_{\bar{x}}(x) = 0\} \cup \{\bar{h} | \forall x,\ \bar{h}(x) = 1\}$$
- Ogni funzione $h_{\bar{x}}$ di questo insieme è banalmente calcolabile (la MT corrispondente, data $\bar{x}$ deve solo emettere 1 o 0 a seconda che l'ingresso sia minore/uguale o maggiore)
- Quindi h è sicuramente calcolabile, esiste la MT che la calcola
- Siamo in grado di dire quale MT calcola h ? Al momento no: tra le MT non sappiamo quale scegliere!

Decidibilità e semidecidibilità

Insiemi decidibili

- Concentriamoci su problemi con risposta binaria
  - Problema = dato un insieme $\mathbf{S} \subseteq \mathbb{N}, x \stackrel{?}{\in} \mathbf{S}$
  - Alternativamente, calcolare la funzione caratteristica di $\mathbf{S}$
    $\mathbf{1}_S$: $\mathbf{1}_S(x) = 1$ se $x \in S$, $\mathbf{1}_\mathbf{S}(x) = 0$ se $x \notin \mathbf{S}$
- Un insieme $\mathbf{S}$ si dice *ricorsivo* o *decidibile* se e solo se la sua funzione caratteristica è computabile
  - (N.B.: $\mathbf{1}_\mathbf{S}$ è una funzione dunque relazione totale per definizione)

Decidibilità e semidecidibilità

Insiemi semidecidibili

- Un insieme $\mathbf{S}$ è *ricorsivamente enumerabile* (RE) o *semidecidibile* se e solo se
  - $\mathbf{S}$ è l'insieme vuoto
  - $\mathbf{S}$ è l'immagine di una funzione totale e computabile: $\mathbf{S} = \mathbf{I}_{g_\mathbf{S}} = \{x | x = g_\mathbf{S}(y), y \in \mathbb{N}\} \Rightarrow \mathbf{S} = \{g_\mathbf{S}(0), g_\mathbf{S}(1), \ldots\}$ da cui *ricorsivamente (algoritmicamente) enumerabile*
- Il termine *semidecidibile* deriva dal fatto che:
  - Se $x \in \mathbf{S}$ enumerando gli elementi di $\mathbf{S}$ prima o poi lo trovo
  - Se $x \notin \mathbf{S}$ non sono mai certo di poter rispondere "no" enumerando, potrei non aver ancora trovato x

Legami tra decidibilità e semidecidibilità

Teorema (Decidibilità $\Rightarrow$ semidecidibilità)

Se un insieme $\mathbf{S}$ *è ricorsivo, esso è ricorsivamente enumerabile*

Dimostrazione.

- Se $\mathbf{S}$ è vuoto, è RE per definizione.
- Assumiamo $\mathbf{S} \neq \varnothing$, e costruiamo una funzione totale e computabile di cui $\mathbf{S}$ è immagine. $\exists k \in \mathbf{S} \Rightarrow \mathbf{1_S}(k) = 1$ Definiamo $g_\mathbf{S}$ come
  $$g_\mathbf{S} : \begin{cases} g_\mathbf{S}(x) = x \text{ se } \mathbf{1_S}(x) = 1 \\ g_\mathbf{S}(x) = k \text{ se } \mathbf{1_S}(x) = 0 \end{cases}$$
- $g_\mathbf{S}$ è computabile, totale e $\mathbf{I}_{g_\mathbf{S}} = \mathbf{S} \Rightarrow \mathbf{S}$ è RE ■

N.B. Dimostrazione non costruttiva: non sappiamo se $\mathbf{S} \neq \varnothing$ né in generale calcolare $g_\mathbf{S}$

Legami tra decidibilità e semidecidibilità

Teorema (semidecidibilità+semidecidibilità=decidibilità)

$\mathbf{S}$ *è ricorsivo se e solo se sono ricorsivamente enumerabili sia* $\mathbf{S}$ *che il suo complemento* $\bar{\mathbf{S}} = \mathbb{N} \setminus \mathbf{S}$

S ricorsivo $\Rightarrow$ S e $\bar{\text{S}}$ RE.

- $\mathbf{S}$ ricorsivo $\Rightarrow$ $\mathbf{S}$ RE per teorema precedente
- $\mathbf{S}$ ricorsivo $\Rightarrow$ $\mathbf{1}_{\mathbf{S}}(x)$ calcolabile $\Rightarrow$ $\mathbf{1}_{\bar{\mathbf{S}}}(x)$ calcolabile (scambio 0 con 1 come risultato di $\mathbf{1}_{\mathbf{S}}(x)$ calcolabile e totale) $\Rightarrow$ $\bar{\mathbf{S}}$ ricorsivo $\Rightarrow$ $\bar{\mathbf{S}}$ RE $\blacksquare$

Legami tra decidibilità e semidecidibilità

S ricorsivo $\Leftarrow$ S e $\bar{\mathrm{S}}$ RE.

- $\mathbf{S}$ RE $\Rightarrow \mathbf{S} = \{g_{\mathbf{S}}(0), g_{\mathbf{S}}(1), g_{\mathbf{S}}(2), g_{\mathbf{S}}(3), \ldots\}$
  $\bar{\mathbf{S}}$ RE $\Rightarrow \bar{\mathbf{S}} = \{g_{\bar{\mathbf{S}}}(0), g_{\bar{\mathbf{S}}}(1), g_{\bar{\mathbf{S}}}(2), g_{\bar{\mathbf{S}}}(3), \ldots\}$
  Osserviamo che $\mathbf{S} \cup \bar{\mathbf{S}} = \mathbb{N}$ e $\mathbf{S} \cap \bar{\mathbf{S}} = \varnothing$, quindi una qualunque $x \in \mathbb{N}$ appartiene a una e una sola delle due enumerazioni di cui sopra:
  $$(\forall x \in \mathbb{N},\ \exists y | x = g_{\bar{\mathbf{S}}}(y)\ \vee\ x = g_{\mathbf{S}}(y))\ \wedge\ (\neg \exists z | g_{\bar{\mathbf{S}}}(z) = g_{\mathbf{S}}(z))$$
- Sono quindi certo di trovare qualunque x nell'enumerazione $\{g_{\mathbf{S}}(0), g_{\bar{\mathbf{S}}}(0), g_{\mathbf{S}}(1), g_{\bar{\mathbf{S}}}(1), g_{\mathbf{S}}(2), g_{\bar{\mathbf{S}}}(2), g_{\mathbf{S}}(3), \ldots\}$
- Nel momento in cui trovo x in posizione dispari concludo che $x \notin \mathbf{S}$, altrimenti $x \in \mathbf{S}$: so quindi calcolare $\mathbf{1}_{\mathbf{S}}$ $\blacksquare$

Conseguenze pratiche e non

Chiusura per complementazione

- Gli insiemi decidibili sono chiusi rispetto al complemento

Definizione delle f calcolabili e totali

- Dato un insieme $\mathbf{S}$ per cui
  - Se $i \in \mathbf{S}$ allora f_i è calcolabile e totale
  - Se f è totale e computabile allora $\exists i \in \mathbf{S} | f_i = f$
- Allora $\mathbf{S}$ non è ricorsivamente enumerabile
- Dimostrazione per assurdo: $\exists\ \mathbf{1_S}(i)$ computabile. Definisco $h(x) = \{f_x(x) + 1$ se $\mathbf{1_S}(x) = 1,\ 0$ altrimenti$\}$. Ho che $\forall x\ h(x) \neq f_{\mathbf{1_S}(x)}(x)$. $h(x)$ è calcolabile, ma diversa da tutte le calcolabili (in almeno un punto, per definizione) $\lightning$ ■

Conseguenze pratiche e non

Risvolti pratici

- Non è possibile, con un formalismo RE (automi, grammatiche, funzioni ricorsive) *definire l'insieme di tutte e sole le f calcolabili totali*
- Non posso in nessun modo descrivere come è fatto l'insieme di tutti e soli i programmi che terminano sempre:
  - Gli FSA e gli AP D definiscono solo funzioni totali, ma non tutte
  - Le MT definiscono tutte le funzioni calcolabili, ma anche quelle non totali
  - Il C mi permette di scrivere qualunque algoritmo, ma anche quelli che non terminano
  - Esiste un sottoinsieme del C che definisce tutti e soli i programmi che non terminano? **NO.**

Riusciamo ad eliminare le funzioni non totali?

Estendiamo la funzione parziale

- Arricchiamo $\mathbb{N}$ con un nuovo valore $\perp$ oppure assegnamo un valore convenzionale ad f quando non è definita
- Matematicamente, non c'è problema nel farlo (infatti in matematica pura si dà poco attenzione alle funzioni parziali)
- Esaminiamo $g(x) = \begin{cases} f_x(x) + 1 \text{ se } f_x(x) \neq \perp \\ \perp \text{ altrimenti} \end{cases}$
- Non riusciamo a costruire una funzione computabile e totale che estende $g(x)$ (devo risolvere il problema dell'arresto per definirla!)
- Posso provare ad estendere una funzione parziale e renderla totale, ma potrei perdere la computabilità del risultato!

Sugli insiemi ricorsivamente enumerabili

Fatti equivalenti

- É equivalente dire che:
  - $\mathbf{S}$ è ricorsivamente enumerabile
  - $\mathbf{S}$ è il dominio $\mathbf{D}_h$ di una funzione computabile e parziale $\mathbf{S} = \{x|h(x) \neq \perp\}$
  - $\mathbf{S}$ è il codominio $\mathbf{I}_g$ di una funzione computabile e parziale $\mathbf{S} = \{x|x = g(y), y \in \mathbb{N}\}$
- N.B. Non contraddice la definizione di RE: posso sempre ottenere una funzione totale $\bar{g}$ da g tale per cui $\mathbf{S} = \mathbf{I}_{\bar{g}}$.
  Allo stesso modo posso, dalla g totale della definizione, ottenere una $\tilde{g}$ parziale facilmente, e.g. da $g(x) = x$ a $\tilde{g}(0) = \perp, \tilde{g}(x) = x - 1$

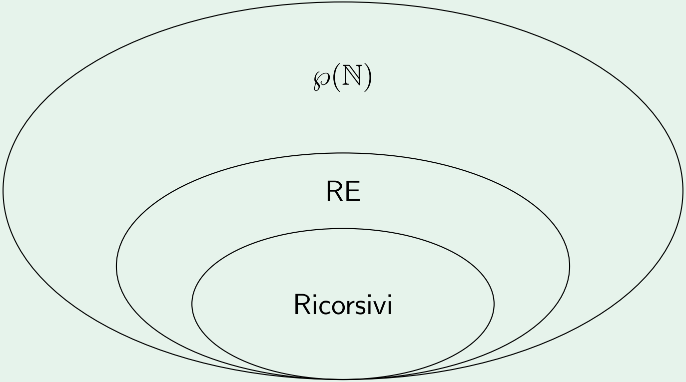
Sugli insiemi ricorsivamente enumerabili

Un ulteriore risultato di semidecidibilità

- Sfruttando le equivalenze appena citate possiamo dimostrare che esistono insiemi semidecidibili che non sono decidibili
- Consideriamo $\mathbf{S} = \{x | f_x(x) \neq \perp\}$: è il dominio $\mathbb{D}_h$ della funzione $h(x) = f_x(x)$ che è computabile e parziale
- Abbiamo quindi che $\mathbf{S}$ è RE
- Sappiamo anche che l'indicatrice $\mathbf{1}_{\mathbf{S}}(x) = 1$ se $f_x(x) \neq \perp, 0$ altrimenti non è computabile totale dunque $\mathbf{S}$ non è decidibile

Riassumendo

Una gerarchia degli insiemi



- I contenimenti sono tutti stretti
- Gli insiemi RE non sono chiusi per complemento

Verso il teorema di Rice

Il teorema di Kleene del punto fisso

- Sia una funzione $t(\cdot)$ totale e computabile. É sempre possibile trovare un $p \in \mathbb{N}$ tale che $f_p = f_{t(p)}$. La funzione f_p è detta punto fisso di $t(\cdot)$.

Il teorema di Kleene del punto fisso

Dimostrazione - 1

- Dato $u \in \mathbb{N}$ definiamo una MT che effettua il seguente calcolo sull'ingresso x:
  1. Calcola $f_u(u) = z$
  2. Se e quando il calcolo precedente termina, calcola $f_z(x)$
- La definizione è effettiva, quindi esiste una MT che la calcola
- Possiamo costruire la MT e cercare (per confronto con le altre MT) il suo numero di Gödel $g(u)$ per una qualsiasi u
- N.B. $g(u)$ è totale e computabile
- Otteniamo $f_{g(u)}(x) = \begin{cases} f_{f_u(u)}(x) \text{ se } f_u(u) \neq \perp \\ \perp \text{ altrimenti} \end{cases}$

Il teorema di Kleene del punto fisso

Dimostrazione - 2

- Sappiamo che, data la $g(\cdot)$ totale e computabile di cui sopra, e data una $t(\cdot)$ totale e computabile anche $t(g(\cdot))$ lo è
- Chiamiamo v il numero di Gödel di $t(g(\cdot))$, cioè $t(g(\cdot)) = f_v(\cdot)$
- Ripetiamo la costruzione della slide precedente usando v al posto di u ottenendo
$$f_{g(v)}(x) = \begin{cases} f_{f_v(v)}(x) \text{ se } f_v(v) \neq \perp \\ \perp \text{ altrimenti} \end{cases}$$
- Ricordando che $t(g(\cdot)) = f_v(\cdot)$ è totale e computabile (ossia $\forall v, f_v(v) \neq \perp$), otteniamo che $f_{g(v)}(x) = f_{f_v(v)}(x)$
- Sostituendo nel secondo membro $f_{f_v(v)}(x) = f_{t(g(v))}$, quindi $g(v)$ è il punto fisso di $t(\cdot)$ ■

Il teorema di Rice

Teorema

$\mathbf{F}$ *insieme di funzioni computabili, l'insieme* $\mathbf{S}$ *degli indici delle MT che calcolano le funzioni di* $\mathbf{F}$, $\mathbf{S} = \{x | f_x \in \mathbf{F}\}$, *è decidibile se e solo se* $\mathbf{F} = \varnothing$ *o* $\mathbf{F}$ *è l'insieme di tutte le funzioni computabili.*

Dimostrazione - 1

- Per assurdo. Supponiamo $\mathbf{S}$ sia decidibile, $\mathbf{F}$ non vuoto e diverso dall'insieme di tutte le funzioni computabili
- Consideriamo $\mathbf{1}_S(x) = \{1 \text{ se } f_x \in \mathbf{F},\ 0 \text{ altrimenti}\}$; essa è calcolabile per l'ipotesi appena fatta
- Possiamo quindi calcolare
  1. il più piccolo $i \in \mathbb{N}$ tale che $f_i \in \mathbf{F}$, ovvero $i \in \mathbf{S}$
  2. il più piccolo $j \in \mathbb{N}$ tale che $f_j \notin \mathbf{F}$, ovvero $j \notin \mathbf{S}$

Il teorema di Rice

Dimostrazione.

- Per quanto detto, $h_{\mathbf{S}}(x) = \{i \text{ se } f_x \notin \mathbf{F}, j \text{ altrimenti}\}$ è a sua volta calcolabile e totale
- Applicando il teorema di Kleene alla funzione totale e computabile $h_{\mathbf{S}}(x)$ otteniamo che esiste un il punto fisso $f_{\bar{x}}$ tale per cui $f_{\bar{x}} = f_{h_{\mathbf{S}}(\bar{x})}$
- Arriviamo alla contraddizione assumendo:
  - $h_{\mathbf{S}}(\bar{x}) = i$: per definizione di $h_{\mathbf{S}}(\cdot)$ abbiamo che $f_{\bar{x}} \notin \mathbf{F}$, ma da quanto appena detto per il t. di Kleene $f_{\bar{x}} = f_{h_{\mathbf{S}}(\bar{x})} = f_i$ da cui, per come definito i, $f_i \in \mathbf{F}$ ↯
  - $h_{\mathbf{S}}(\bar{x}) = j$: per definizione di $h_{\mathbf{S}}(\cdot)$ abbiamo che $f_{\bar{x}} \in \mathbf{F}$, ma da quanto appena detto per il t. di Kleene $f_{\bar{x}} = f_{h_{\mathbf{S}}(\bar{x})} = f_j$ da cui, per come definito j, $f_j \notin \mathbf{F}$ ↯

∎

Il teorema di Rice

Effetti pratici

- In tutti i casi non banali $\mathbf{S}$, l'insieme delle funzioni calcolabili con una data caratteristica desiderata, *non è decidibile*!
  - N.B. è quindi *semidecidibile* o *indecidibile*
- Posso dire se un programma `P` è corretto? Dire se risolve un dato problema? (La macchina $\mathcal{M}_x$ calcola la sola funzione contenuta nell' insieme $\mathbf{S} = \{f\}$?)
- É possibile stabilire l'equivalenza tra due programmi? (La macchina $\mathcal{M}_x$ calcola la sola funzione contenuta nell'insieme $\mathbf{S} = \{f_y\}$?)
- É possibile stabilire se un generico programma gode di una qualsiasi proprietà non banale riferita alla funzione che calcola? (e.g. non produce mai un risultato negativo?)

Stabilire se un problema è (semi)decidibile

Metodi pratici

- Dire se un generico problema è (semi)decidibile o meno è un problema indecidibile
- Dato uno specifico problema:
  - Se troviamo un algoritmo *che termina sempre* $\rightarrow$ decidibile
  - Se troviamo un algoritmo che termina sempre se la risposta è "sì", ma può non terminare se è "no" $\rightarrow$ semidecidibile
  - Se riteniamo che il problema sia indecidibile come fare? Potremmo costruirci una dimostrazione diagonale ogni volta ... fattibile, ma parecchio impegnativo!

Dimostrazioni di non (semi)decidibilità

La riduzione dei problemi

- Il teorema di Rice ci consente di mostrare agevolmente che un problema non è decidibile
  - N.B. potrebbe comunque essere semidecidibile!
- Una tecnica alternativa, molto generale, è quella della *riduzione dei problemi*
- L'abbiamo già usata implicitamente, in maniera informale
- Consente di dimostrare in modo agevole l'indecidibilità di alcuni problemi

La tecnica di riduzione

Una visione operativa

- Se ho un algoritmo per risolvere un problema $\mathtt{P}$, posso riusarlo (modificandolo) per risolvere $\mathtt{P}'$ simile a $\mathtt{P}$
  - Se so risolvere il problema di ricerca di un elemento in un insieme, so calcolare l'intersezione tra due insiemi
  - Se so calcolare unione e complemento di due insiemi, so calcolarne l'intersezione
- In generale, se trovo un algoritmo che, dato un esemplare di $\mathtt{P}'$ ne costruisce la soluzione usando un esemplare di $\mathtt{P}$ che so risolvere, ho *ridotto* $\mathtt{P}'$ a $\mathtt{P}$

La tecnica di riduzione

Formalizzazione

- Sono in grado di risolvere $y \in \mathbf{S}'$ (= calcolare $\mathbf{1}_{\mathbf{S}'}(\cdot)$) e voglio risolvere $x \in \mathbf{S}$
- Se ho una funzione t calcolabile e totale tale per cui $x \in \mathbf{S} \Leftrightarrow t(x) \in \mathbf{S}'$ posso farlo!
  - Dato x, calcolo $\mathbf{1}_{\mathbf{S}'}(t(x))$ che, equivale a calcolare $\mathbf{1}_{\mathbf{S}}(x)$
- Utile anche in direzione opposta
  - So di non saper risolvere $y \in \mathbf{S}'$ (S' non è decidibile)
  - Se trovo una funzione t calcolabile e totale tale per cui $x \in \mathbf{S} \Leftrightarrow t(x) \in \mathbf{S}'$ anche $\mathbf{S}$ è non decidibile

La tecnica di riduzione

Esempi di utilizzo

- Dall'indecidibilità del problema dell'arresto della MT deduciamo l'indecidibilità del problema della terminazione del calcolo in generale
  - Siano dati una MT $\mathcal{M}_i$, un numero $x \in \mathbb{N}$,
  - Costruisco un programma P (e.g., in `C`) che simula $\mathcal{M}_i$ e memorizzo il numero x su un file `f`
  - Il programma `P` termina la computazione su f se e solo se $g(i, x) \neq \perp$
  - Se sapessi decidere se `P` termina, sarei in grado di risolvere il problema dell'arresto della MT
- N.B. avremmo potuto dimostrare in modo diretto l'indecidibilità di un programma `C` enumerando i possibili programmi e applicando la consueta tecnica diagonale... con un po' più di fatica

La tecnica di riduzione

Esempi di utilizzo

- É decidibile dire se, durante l' esecuzione di un generico programma `P` si accede ad una variabile non inizializzata?
  - Assumiamo per assurdo che sia decidibile
  - Riduciamo il problema della terminazione a quello appena citato:
    - Dato un generico programma `Q(n)`, costruisco un programma `P` fatto in questo modo `{int x,y; Q(n); y=x;}`, avendo cura di usare variabili non presenti in `Q`
    - l'accesso `y=x` alla variabile non inizializzata `x` da parte di `P` è fatto se (e solo se) `Q` termina
  - Se fossi in grado di decidere il problema dell' accesso a variabile non inizializzata, potrei decidere il problema della terminazione del calcolo $\lightning$

La tecnica di riduzione

Esempi di utilizzo

- Una grande varietà di proprietà tipiche (e che spesso vorremmo eliminare) possono essere dimostrate non decidibili come visto sopra:
  - Il programma effettua un accesso ad un array fuori dai limiti?
  - Il programma effettua una divisione per zero?
  - Questi tipi saranno compatibili a run-time?
  - Questo programma solleverà un'eccezione?

Considerazioni pratiche

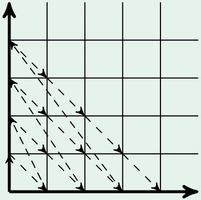
Non decidibili, ma...

- Le proprietà di cui sopra, così come l'arresto della MT non sono decidibili, ma sono semidecidibili:
  - Se la MT si ferma, prima o poi lo scopro, se un programma va in `segfault`, anche ...
- Questo è vero, ma con quale metodo operativo sono in grado di scoprire questo fatto?
  - Se inizio ad eseguire `P` con ingresso y e `P` non si ferma, come faccio a scoprire che `P` con ingresso x effettua una divisione per zero?

Simulazione in diagonale

Dimostrazione di semidecidibilità

- Stabilire se $\exists z | f_x(z) \neq \perp$ è semidecidibile
- Come fare? Se inizio a calcolare $f_x(0)$ e non termina, come scopro se $f_x(1) \neq \perp$?
- Idea: simulo "in diagonale" $f_x()$ eseguendo una sola mossa alla volta per ogni input fin quando una non si arresta



ascisse: valori di ingresso

ordinate: numero della mossa

- Se $\exists z | f_x(z) \neq \perp$ sicuramente lo incontro prima o poi (simulo sempre abbastanza passi di $f_x(z)$)

Riassumendo

Una dimostrazione di indecidibilità

- Scoprire molti dei comportamenti indesiderati a runtime di un programma non è decidibile, ma solamente semidecidibile
- N.B. Attenzione a qual è esattamente il problema semidecidibile: la *presenza* del comportamento indesiderato!
- Ma il complemento di un problema semidecidibile non può essere altro che indecidibile
  - Se fosse semidecidibile, sarebbero entrambi decidibili!
- L'assenza di errori a run-time di un programma è quindi un problema indecidibile!
- Come "consolazione" otteniamo un metodo di dimostrare che un problema è indecidibile: dimostrare che il suo complemento è solamente semidecidibile, ma non decidibile