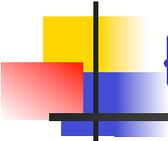


# Teoria della Calcolabilità

---

- Si occupa delle questioni fondamentali circa la **potenza** e le **limitazioni** dei sistemi di calcolo.
- L'origine risale alla prima metà del ventesimo secolo, quando i logici matematici iniziarono ad esplorare i concetti di **computazione**  
**algoritmo**  
**problema risolvibile per via algoritmica**  
e dimostrarono l'esistenza di **problemi non risolvibili**, ossia **problemi che non ammettono un algoritmo di risoluzione**.  
**⇒ Problemi non decidibili**

1



## Problemi computazionali

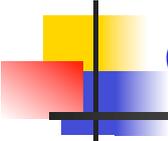
---

Problemi formulati matematicamente di cui cerchiamo una soluzione algoritmica.

### Classificazione:

- **problemi non decidibili**
- **problemi decidibili**
  - **problemi trattabili** (costo polinomiale)
  - **problemi intrattabili** (costo esponenziale)

2



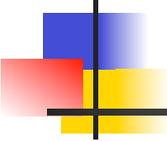
# Calcolabilità e complessità

---

- **Calcolabilità:** nozioni di algoritmo e di problema non decidibile.
- **Complessità:** nozione di algoritmo efficiente e di problema intrattabile.
- La calcolabilità ha lo scopo di classificare i problemi in *risolvibili* e *non resolvibili*, mentre la complessità in "*facili*" e "*difficili*".

3

## ESISTENZA DI PROBLEMI INDECIDIBILI



---

4

## Insiemi numerabili

- Due insiemi A e B hanno lo stesso numero di elementi



si può stabilire una **corrispondenza biunivoca** tra i loro elementi.

- Un insieme è **numerabile** (possiede una infinità numerabile di elementi)



i suoi elementi possono essere messi in **corrispondenza biunivoca con i numeri naturali**.

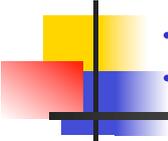
5

## Insiemi numerabili

- Un insieme numerabile è un insieme i cui elementi possono essere enumerati, ossia descritti da una sequenza del tipo

$$a_1, a_2, \dots, a_n, \dots$$

6



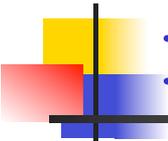
## Insiemi numerabili: esempi

---

- Insieme dei numeri naturali  $\mathbb{N}$
- Insieme dei numeri interi  $\mathbb{Z}$ :
  - $n \leftrightarrow 2n + 1 \quad n \geq 0$
  - $n \leftrightarrow 2|n| \quad n < 0$

0, -1, 1, -2, 2, -3, 3, -4, 4, ...
- Insieme dei numeri naturali pari:
  - $2n \leftrightarrow n \quad 0, 2, 4, 6, 8, \dots$
- Insieme delle stringhe finite di simboli presi da un alfabeto finito.

7



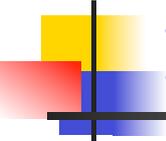
## Insiemi numerabili: esempi

---

- Insieme delle stringhe finite di simboli di un alfabeto finito.

a, b, ..., z, aa, ab, ..., az, ba, bb, ..., bz, .....,  
za, ..., zz, aaa, ....

8



## Insiemi non numerabili

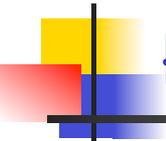
---

Sono tutti gli insiemi non equivalenti a  $\mathbb{N}$ .

### Esempi:

- insieme dei numeri reali
- insieme dei numeri reali compresi nell'intervallo aperto  $(0,1)$
- insieme dei numeri reali compresi nell'intervallo chiuso  $[0,1]$
- insieme di tutte le linee nel piano
- insieme delle funzioni in una o più variabili.

9



## Problemi computazionali

---

L'insieme dei problemi computazionali  
NON è numerabile.

10

## Problemi e funzioni

- Un problema computazionale può essere visto come una funzione matematica che associa ad ogni insieme di dati, espressi da  $k$  numeri interi, il corrispondente risultato, espresso da  $j$  numeri interi

$$f: N^k \rightarrow N^j$$

- L'insieme delle funzioni  $f: N^k \rightarrow N^j$  NON è numerabile.

11

## Diagonalizzazione

$$F = \{ \text{funzioni } f \mid f: N \rightarrow \{0,1\} \}$$

ogni  $f \in F$  può essere rappresentata da una sequenza infinita:

$$\begin{array}{cccccccc} x & 0 & 1 & 2 & 3 & 4 & \dots & n & \dots \\ f(x) & 0 & 1 & 0 & 1 & 0 & \dots & 0 & \dots \end{array}$$

o, se possibile, da una regola finita di costruzione:

$$f(x) = \begin{cases} 0 & x \text{ pari} \\ 1 & x \text{ dispari} \end{cases}$$

12

# Diagonalizzazione

## Teorema

L'insieme  $F$  non è numerabile.

Dim.

- Per assurdo,  $F$  sia numerabile.
- Possiamo enumerare ogni funzione:  
assegnare ad ogni  $f \in F$  un numero progressivo nella numerazione, e costruire una tabella (infinita) di tutte le funzioni

13

# Diagonalizzazione

$x$	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	1	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	0	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	0	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	0	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	1	0	0	0	1	...
...				...						...

14

# Diagonalizzazione

Consideriamo la funzione  $g \in F$

$$g(x) = \begin{cases} 0 & f_x(x) = 1 \\ 1 & f_x(x) = 0 \end{cases}$$

$g$  non corrisponde ad alcuna delle  $f_i$  della tabella poiché differisce da tutte nei valori posti sulla diagonale principale.

15

# Diagonalizzazione

$x$	0	1	2	3	4	5	6	7	8	...
$f_0(x)$	<b>1</b>	0	1	0	1	0	0	0	1	...
$f_1(x)$	0	<b>0</b>	1	1	0	0	1	1	0	...
$f_2(x)$	1	1	<b>0</b>	1	0	1	0	0	1	...
$f_3(x)$	0	1	1	<b>0</b>	1	0	1	1	1	...
$f_4(x)$	1	1	0	0	<b>1</b>	0	0	0	1	...
...										
$g(x)$	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	.	.	.		

16

## Diagonalizzazione

- Per assurdo:  $\exists j$  t.c.  $g(x) = f_j(x)$
- allora  $g(j) = f_j(j)$ , ma per definizione

$$g(j) = \begin{cases} 0 & f_j(j) = 1 \\ 1 & f_j(j) = 0 \end{cases}$$

- cioè  $g(j) \neq f_j(j)$ .

**$\Rightarrow$  contraddizione!!!**

17

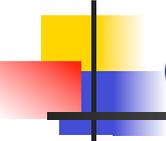
## Diagonalizzazione

Per qualunque numerazione scelta, esiste sempre almeno una funzione esclusa:  $F$  non è numerabile.

Si possono considerare linee arbitrarie che attraversano la tabella toccando tutte le righe e tutte le colonne esattamente una volta, e definire funzioni che assumono in ogni punto un valore opposto a quello incontrato sulla linea.

**Esistono infinite funzioni di  $F$  escluse da qualsiasi numerazione.**

18



## Conclusione

---

- ***F non è numerabile***, e a maggior ragione, non sono numerabili gli insiemi delle funzioni:

$$f: N \rightarrow N$$

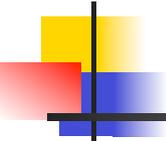
$$f: N \rightarrow R$$

$$f: R \rightarrow R$$

$$f: N^k \rightarrow N^j$$

*L'insieme dei problemi computazionali non è numerabile.*

19

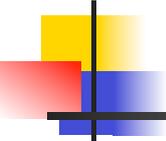


## Il problema della rappresentazione

---

L'informatica rappresenta tutte le sue entità (quindi anche gli algoritmi) in forma digitale, come ***sequenze finite di simboli di alfabeti finiti*** (e.g.,  $\{0,1\}$ );  
descrive dunque un ***mondo numerabile***.

20

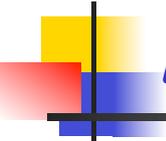


# Il concetto di algoritmo

---

- Il concetto di **algoritmo** è l'elemento centrale della teoria della calcolabilità.
- Un **algoritmo** è un procedimento di calcolo che consente di pervenire alla soluzione di un problema, numerico o simbolico, mediante una sequenza finita di operazioni, completamente e univocamente determinate.

21



# Algoritmi

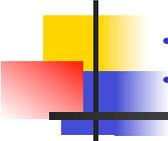
---

Possiamo descrivere gli algoritmi come programmi per un calcolatore. Questa rappresentazione è costituita da sequenze finite di simboli:

**Gli algoritmi sono un'infinità numerabile!**

Le funzioni matematiche (e quindi i problemi computazionali) non sono numerabili.

22



## Il problema della rappresentazione

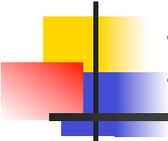
---

$|\{\text{Problemi}\}| \gg |\{\text{Algoritmi}\}|$



**Esistono funzioni (problemi) per cui non esiste un algoritmo di calcolo!**

23



## Il problema dell'arresto

---

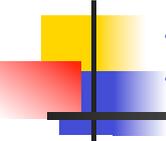
Abbiamo dimostrato l'esistenza di funzioni/problemi non calcolabili.

I problemi che si presentano spontaneamente sono tutti calcolabili.

Non è stato facile individuare un problema che non lo fosse.

Turing (1930): **Problema dell'arresto.**

24

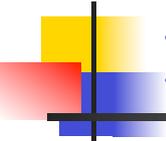


## Il problema dell'arresto

---

- Considera un algoritmo che indaga sulle proprietà di altri algoritmi, che sono trattati come dati.
- È legittimo: gli algoritmi sono rappresentabili con sequenze di simboli, che possono essere presi dallo stesso alfabeto usato per codificare i dati di input.
- Una stessa sequenza di simboli può essere quindi interpretata sia come un programma, sia come un dato di ingresso di un altro programma.

25

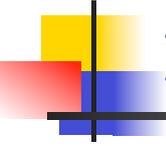


## Il problema dell'arresto

---

- Un algoritmo **A**, comunque formulato, può operare sulla rappresentazione di un altro algoritmo **B**.
- Possiamo calcolare **A(B)**.
- In particolare può avere senso calcolare **A(A)**.

26

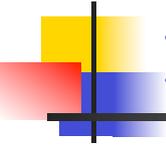


## Il problema dell'arresto

---

*Presi ad arbitrio un **algoritmo**  $A$  e i suoi **dati di input**  $D$ , decidere in tempo finito se la computazione di  $A$  su  $D$  termina o no.*

27



## Il problema dell'arresto

---

Consiste nel chiedersi se un generico programma termina la sua esecuzione, oppure “va in ciclo”, ovvero continua a ripetere la stessa sequenza di istruzioni all'infinito (supponendo di non avere limiti di tempo e memoria).

28

## ESEMPIO:

Stabilire se un intero  $p > 1$  è primo.

---

Primo(p)

```
fattore = 2;
```

```
while (p % fattore != 0)
```

```
    fattore++;
```

```
return (fattore == p);
```

Termina sicuramente (la guardia del **while** diventa falsa quando  $\text{fattore} = p$ ).

29

## ESEMPIO

- Programma che trova il più piccolo numero intero pari (maggiore o uguale a 4) che **NON** sia la somma di due numeri primi.
- Il programma si **arresta** quando trova  $n \geq 4$  che **NON** è la somma di due primi.

30

## ESEMPIO

**Goldbach()**

```
n = 2;
```

```
do {
```

```
    n = n + 2;
```

```
    controesempio = true;
```

```
    for (p = 2; p ≤ n - 2; p++) {
```

```
        q = n - p;
```

```
        if (Primo(p) && Primo(q))
```

```
            controesempio = false;
```

```
    }
```

```
} while (!controesempio);
```

```
return n;
```

31

## Conggettura di Goldbach.

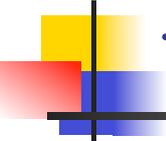
XVIII secolo

“ogni numero intero pari  $n \geq 4$  è la somma di due numeri primi”

Conggettura falsa → Goldbach() si arresta

Conggettura vera → Goldbach() NON si arresta

32



## TEOREMA

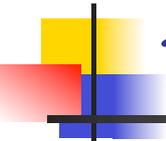
---

Turing ha dimostrato che riuscire a dimostrare se un programma arbitrario si arresta e termina la sua esecuzione non è solo un'impresa ardua, ma in generale è IMPOSSIBILE!

### TEOREMA

Il problema dell'arresto è INDECIDIBILE.

33



## DIMOSTRAZIONE

---

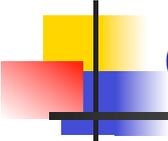
Se il problema dell'arresto fosse decidibile, allora esisterebbe un **algoritmo ARRESTO** che:

- presi  $A$  e  $D$  come dati di input
- determina in tempo finito le risposte:

$ARRESTO(A,D) = 1$  se  $A(D)$  termina

$ARRESTO(A,D) = 0$  se  $A(D)$  non termina

34



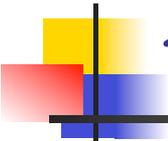
## Osservazione

---

*L' algoritmo ARRESTO non può consistere in un algoritmo che simuli la computazione  $A(D)$ :*

se  $A$  non si arresta su  $D$ , ARRESTO non sarebbe in grado di rispondere NO (0) in tempo finito.

35



## DIMOSTRAZIONE

---

In particolare possiamo scegliere  $D = A$ , cioè considerare la computazione  $A(A)$ :

$ARRESTO(A,A) = 1$



$A(A)$  termina

36

## DIMOSTRAZIONE

Se esistesse l' algoritmo ARRESTO,  
esisterebbe anche il seguente algoritmo:

PARADOSSO(A)

```
while (ARRESTO(A,A)) {  
    ;  
}
```

37

## DIMOSTRAZIONE

L' ispezione dell' algoritmo PARADOSSO  
mostra che:

PARADOSSO(A) termina

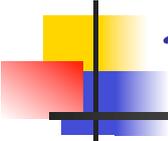


$x = \text{ARRESTO}(A,A) = 0$



A(A) non termina

38



## DIMOSTRAZIONE

---

Cosa succede calcolando PARADOSSO(PARADOSSO)?

PARADOSSO(PARADOSSO) termina



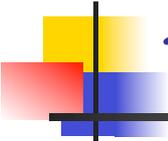
$x = \text{ARRESTO}(\text{PARADOSSO}, \text{PARADOSSO}) = 0$



PARADOSSO(PARADOSSO) non termina

contraddizione!

39



## DIMOSTRAZIONE

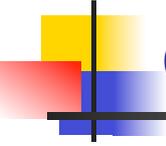
---

L'unico modo di risolvere la contraddizione è che l'algoritmo PARADOSSO non possa esistere.

Dunque non può esistere nemmeno l'algoritmo ARRESTO.

In conclusione, *il problema dell'arresto è indecidibile!*

40



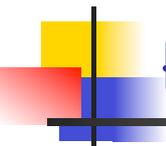
## Osservazione

---

Come già osservato, l' algoritmo ARRESTO costituirebbe uno strumento estremamente potente:

permetterebbe infatti di dimostrare congetture ancora aperte sugli interi (esempio: la congettura di Goldbach).

41



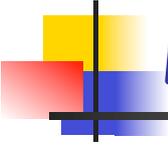
## Problemi indecidibili

---

- Altri problemi lo sono:
  - Ad esempio, è indecidibile stabilire l'**equivalenza** tra due programmi (se per ogni possibile input, producono lo stesso output)
- **"Lezione di Turing"**:

*non esistono algoritmi che decidono il comportamento di altri algoritmi esaminandoli dall'esterno, cioè senza passare dalla loro simulazione.*

42



## Modelli di calcolo

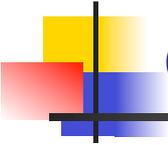
---

La teoria della calcolabilità dipende dal linguaggio/calcolatore/modello di calcolo?

oppure ...

la decidibilità è una proprietà del problema?

43



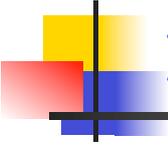
## Osservazione

---

- L'uso di uno specifico linguaggio non influisce sui risultati di indecidibilità:

sono validi per qualunque modello di calcolo che possa formalizzare il comportamento di un calcolatore.

44

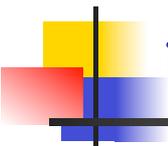


## Il decimo problema di Hilbert

---

- Esistono risultati di non calcolabilità relativi ad altre aree della matematica, tra cui la teoria dei numeri e l'algebra.
- Tra questi, occupa un posto di rilievo il ben noto **decimo problema di Hilbert**.

45



## Equazioni diofantee

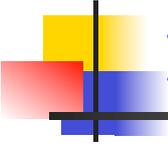
---

Un'**equazione diofantea** è un'equazione della forma

$$p(x_1, x_2, \dots, x_m) = 0$$

dove  $p$  è un polinomio a coefficienti interi.

46



## Il decimo problema di Hilbert

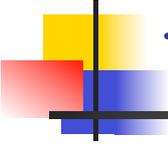
---

Data un' arbitraria equazione diofantea, di grado arbitrario e con un numero arbitrario di incognite

$$p(x_1, x_2, \dots, x_m) = 0$$

stabilire se  $p$  ammette soluzioni intere.

47

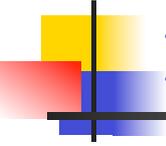


## Teorema

---

Il decimo problema di Hilbert non è calcolabile.

48



## Il decimo problema di Hilbert

---

La questione circa la calcolabilità di questo problema è rimasta aperta per moltissimi anni,

ha attratto l'attenzione di illustri matematici,

ed è stata risolta nel 1970 da un matematico russo allora poco più che ventenne, Yuri Matiyasevich.