Calcolabilità

Fondamenti di Informatica I
Corso di laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

Domenico Lembo, Paolo Liberatore,
Alberto Marchetti Spaccamela, Marco Schaerf

La teoria della calcolabilità

La teoria della calcolabilità si occupa dei fondamenti del calcolo, inteso nel senso più ampio possibile

Ha due scopi principali:

- scoprire quali sono i limiti del calcolo, cioè caratterizzare quali sono le funzioni che è possibile calcolare in un dato formalismo (modello di calcolo) e quelle che non sono calcolabili
- Confrontare i modelli di calcolo via via proposti per formalizzare il concetto di calcolo, con particolare attenzione alla potenza espressiva

Nota: alcuni dei principali risultati della teoria della calcolabilità sono precedenti all'introduzione dei calcolatori (quindi hanno una importanza teorica dal punto di vista logico-matematico, oltre che pratica)!

Problemi e Programmi

problemi facili

- trovare la media di due numeri
- stampare le linee di un file che contengono una parola

…

problemi difficili

- trovare il circuito minimo data una tabella
- determinare la migliore mossa in una partita a scacchi

…

per questi problemi menzionati esistono algoritmi e quindi programmi che li risolvono

(almeno in teoria)

Cosa succede in generale? Per ogni problema esiste un algoritmo che lo risolve?

(questa domanda ha a che fare con lo studio dei limiti del calcolo)

Problemi e funzioni

Alcune considerazioni preliminari:

- Ogni problema che intendiamo automatizzare può essere ricondotto al calcolo di una funzione, in quanto stabilisce una corrispondenza fra un insieme di dati in ingresso ed un insieme di dati in uscita.
- Consideriamo nel seguito solo funzioni sui numeri naturali: infatti è sempre possibile codificare con un numero naturale qualunque dato e ricondurre il problema originario al calcolo di una funzione matematica sui naturali.

Modelli di calcolo

L'insieme di regole non ambigue ed eseguibili utilizzate per definire un algoritmo (per il calcolo di una funzione) definisce un modello di calcolo

Molti modelli di calcolo sono stati proposti, al fine di formalizzare la nozione di calcolabilità. Ad es.: le equazioni funzionali di Kleene, il lambda calcolo di Church, i sistemi di Post, le funzioni ricorsive, la macchina di Turing.

Fra i principali risultati raggiunti vi è quello che stabilisce che tutti i modelli di calcolo più espressivi proposti sono fra loro equivalenti (calcolano tutti lo stesso insieme di funzioni, che coincide con l'insieme delle funzioni ricorsive).

Tesi di Church-Turing: una funzione è umanamente calcolabile se e solo se è calcolabile da una macchina di Turing.

Anche i più comuni linguaggi di programmazione (come il Python) sono Turing equivalenti.

Nel seguito non facciamo quindi riferimento ad uno specifico modello di calcolo (possiamo comunque pensare di adottare il linguaggio Python)

problemi non calcolabili

Per certi problemi non esiste un programma che li risolve!

Questa affermazione si dimostra in due modi:

1. ci sono più problemi che programmi
2. esiste un problema che non si può risolvere con un programma

problemi di decisione

per semplicità consideriamo solo problemi con soluzione 0/1 (problemi di decisione).

Ogni altro problema ha una sua versione di questo tipo:

- dati a, b, c,
  verificare se c è la media di a e b
- data una posizione degli scacchi e una mossa possibile,
  verificare se è quella migliore

problemi di decisione sui naturali

- dato del problema: naturale
- risultato: sì o no

Per alcuni naturali la soluzione è 1, per tutti gli altri è 0

esempio:

- decidere se un naturale è pari
  soluzione 1 per 2, 4, 6, ecc.
- decidere se un naturale è primo
  soluzione 1 per 2, 3, 5, 7, ecc.

rappresentazione dei problemi

Problema (di decisione) sui naturali = verificare l'appartenenza di un numero naturale ad un certo insieme di numeri naturali

Quest'ultimo è infatti l'insieme di naturali per cui la soluzione è 1

- decidere se un numero naturale è pari
  insieme {2,4,6,8,10,12,…}
- decidere se un numero naturale è primo
  insieme {1,2,3,5,7,11,…}

Cardinalità e numero dei programmi

problema=insiemi di naturali

Quindi:

|problemi| = |insiemi di naturali|

In altri termini, il numero dei problemi (cioè delle funzioni sui naturali) è pari alla cardinalità dell'insieme delle parti di **N**. Dato che questo insieme non è numerabile (o contabile), ne segue che l'insieme dei problemi non è contabile (cioè non può essere messo in corrispondenza biunivoca con l'insieme dei naturali)

D'altro canto,

programma = sequenza di caratteri

quindi:

|programmi| = |interi|

Il che vuol dire che l'insieme dei programmi è contabile

Non calcolabilità

|problemi| > |programmi|

→ ci sono problemi per i quali non c'è un programma

sì ma...la dimostrazione si basa sulla uguaglianza
problema=insieme di interi
che problema è {1,5,29,31,52,100,102,109}?

alcuni insiemi non hanno un senso
nessuna utilità pratica
non è che quelli non risolvibili sono tutti così?

Premessa

- un programma Python sta su un file
- può leggere un file qualsiasi:
- `s=open('nomefile').read()`
- anche se stesso? Sì!
- stesso.py:
  - `s=open('stesso.py').read()`
  - `print(s)`
- apre e legge il file, poi lo stampa

il problema della fermata

```
while i!=10:
   print(i)
   i=i+1
```

- se `i≤10` il programma termina
- altrimenti va avanti all'infinito

Se non termina?

sarebbe utile saperlo prima

es: Word non risponde

ci sta solo mettendo del tempo o è entrato in un ciclo infinito?

secondo caso: errore nel programma

idea: scrivere un programma che verifica se questo può succedere

il problema della fermata

dato

un programma

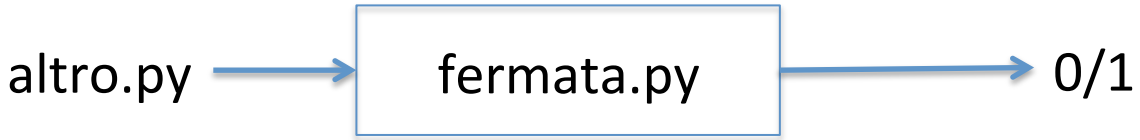
soluzione

1 se il programma termina (con un qualunque insieme di dati in ingresso)

0 se il programma non termina

scrivere un programma che risolve questo problema

il programma del problema della fermata



fermata.py legge un file

in quel file c'è un altro programma (altro.py)

Dopo aver letto altro.py, fermata.py :

stampa 1 se altro.py contiene un programma che termina

altrimenti stampa 0

Complicato?

fermata.py è un programma che legge un file
solo che nel file c'è un altro programma
esempio: vediamo se ci sono `while` in altro.py

```
s=open('altro.py').read()
m=s.find('while')
if m!=-1:
   print('ci sono while')
```

non basta…

problema della fermata, in generale

se non ci sono cicli o chiamate a funzione termina sempre

ma anche se ci sono cicli può terminare:

```
while x<10:
   print(x)
   x=x+1
```

programma con ciclo che termina sempre

Non c'è modo di eseguirlo?

fermata.py esegue il programma in esame altro.py
se questo termina, stampa 1

se non termina?

anche dopo due ore:
come faccio a sapere che non finirà fra 2 secondi?

Analisi del programma

```
while i<10:
   print(i)
   i=i+1
```

termina sempre

```
while i!=10:
   print(i)
   i=i+1
```

termina solo se i≤10, necessaria un'analisi del programma per capirlo

Problema indecidibile

non esiste un programma che risolve il problema della fermata

non c'è modo di realizzare fermata.py

dimostrazione per assurdo: assumiamo che esista

Dimostrazione per assurdo

diciamo che fermata.py esiste

lo modificchiamo così:

	fermata.py		modificato.py
1	f=open('altro.py')		f=open('altro.py')
	…		…
2	print('0')	⇒	quit()
	…		…
3	print('1')	⇒	while 1:
	…		pass
			…

1. legge altro.py
2. se altro.py non termina:
   fermata.py stamperebbe 0
   modificato.py termina senza stampare niente
3. se altro.py termina:
   fermata.py stamperebbe 1
   modificato.py invece entra in un ciclo infinito

Comportamento di modificato.py

se altro.py non termina

    modificato.py termina

se altro.py termina

    modificato.py entra in un ciclo infinito

Ulteriore modifica

se altro.py non termina

modificato.py termina

se altro.py termina

modificato.py entra in un ciclo infinito

si cambia la riga di lettura:

f=open('altro.py') ⇒ f=open('modificato.py')

comportamento: uguale, ma con modificato.py al posto di altro.py…

Quando modificato.py legge se stesso

se modificato.py termina

    modificato.py entra in un ciclo infinito

se modificato.py non termina

    modificato.py termina

Conclusione

modificato.py termina se non termina e viceversa

contraddizione

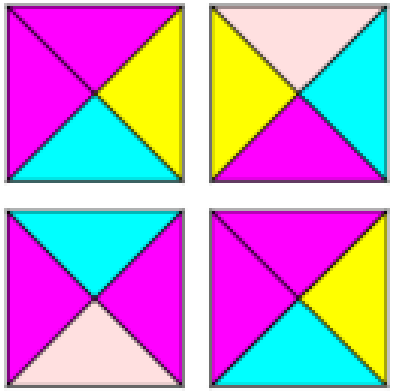
fermata.py non esiste

Poiché Python è Turing-equivalente, possiamo concludere che un programma del genere non è umanamente calcolabile (in nessun modello di calcolo, anche assumendo risorse di calcolo infinite)

Il problema della fermata (halting problem) è indecidibile!

Altri problemi indecidibili

- determinare se delle matrici si possono moltiplicare in qualche modo (anche più volte) per ottenere la matrice di tutti zeri
- determinare se una stringa è generata da una grammatica a forma di frase (il tipo più generale di grammatica)
- determinare se un certo insieme di piastrelle dai bordi colorati possono riempire un piano infinito collimando per colore ai bordi (tiling problem)
- …



si dimostra per riduzione dal problema della fermata a loro, se fossero decidibili lo sarebbe anche il problema della fermata

Linea di dimostrazione

si considera un programma che legge un altro programma

si fa in modo che faccia il contrario di quello che dovrebbe verificare

gli si fa leggere se stesso

Autoreferenzialità:

- per dimostrare che esiste un problema non risolvibile con un programma (problema indecidibile) abbiamo usato un problema relativo ai programmi (problema della fermata)

- poi abbiamo assunto che esista un programma in grado di risolvere il problema, e abbiamo fatto leggere a questo programma il programma stesso!

Paradosso

Quello che oggi chiamiamo paradosso, *παρα* (contro) e *δόξα* *(*opinione), nacque con una nota affermazione di Epimenide di Cnosso (VI secolo a.C.), il quale, cretese egli stesso, ebbe a dire che «*tutti i Cretesi mentono sempre*»

Il significato letterale è “affermazione contro l’intuizione” o “affermazione contro il senso comune”

sempre basati sulla autoreferenzialità

L'antinomia, dal greco αντι (contro) e νομος (legge) è un particolare tipo di paradosso tale che sia la sua affermazione, sia la sua negazione comportano necessariamente un risultato contraddittorio.

Alcuni paradossi

sempre basati sulla autoreferenzialità

- questa frase è falsa
- un coccodrillo ha catturato un bambino, ma ora promette di lasciarlo andare se il padre indovinerà cosa farà il coccodrillo; il padre risponde che non lo lascerà andare
- Se ogni regola ha un'eccezione, allora anche questa regola (“ogni regola ha una eccezione”) deve avere un'eccezione; quindi non è vero che ogni regola ha un'eccezione
- all’unico barbiere di un villaggio è stato ordinato di radere tutti e solo quelli che non radono se stessi. Chi rade il barbiere?”

Paradosso del mentitore

"questa frase è falsa"

- se è vera, allora deve valere "questa frase è falsa"
- se è falsa, allora "questa frase è falsa" è vero

non si può dire che è vera

non si può dire che è falsa

Infatti in entrambi i casi ottengo una contraddizione!

similmente non si riesce a dimostrare né a confutare l'affermazione "tutti i cretesi mentono sempre" di Epimenide di Cnosso!

Paradosso del coccodrillo

"un coccodrillo ha catturato un bambino, ma ora promette di lasciarlo andare se il padre indovinerà cosa farà il coccodrillo.

Il padre ha detto che non lo lascerà andare"

il coccodrillo lo lascia andare → il padre ha sbagliato → il coccodrillo non deve lasciarlo andare

il coccodrillo non lo lascia → il padre ha indovinato → il coccodrillo lo deve lasciare andare

Paradosso del Barbiere

"all'unico barbiere di un villaggio è stato ordinato di radere tutti e solo coloro che non si radono da soli. Chi rade il barbiere?"

Assumiamo che il barbiere si rada da solo. Secondo l'ordine ricevuto, deve radere solo quelli che non si radono da soli. Contraddizione!

Assumiamo che il barbiere non si rada da solo. L'ordinanza però gli impone di radere tutti coloro che non si radono da soli. Quindi deve radersi. Contraddizione!

Questo paradosso (o meglio antinomia), proposto nel 1918 da Bertrand Russell (1872-1970), è una riformulazione del cosiddetto Paradosso di Russell (1902) che è invece formulato sugli insiemi.

Insiemi di insiemi

Ovviamente esistono insiemi che contengono altri insiemi.

famiglia = insieme di persone

insieme di famiglie = insieme di insiemi di persone

Ci sono poi insiemi che contengono se stessi (ad esempio, l'insieme dei concetti astratti è esso stesso un concetto astratto) ed insiemi che non contengono se stessi (ad esempio, l'insieme dei naturali).

Paradosso di Russell

T = { S | S è un insieme e S $\notin$ S }, cioè T è l'insieme degli insiemi che non contengono se stessi

domanda: T $\in$ T?

Assumiamo che sia vero:

T è in se stesso, cioè è un insieme che contiene se stesso; ma, per definizione, T non contiene nessun insieme che contiene se stesso; quindi T non contiene se stesso → contraddizione!

Assumiamo che sia falso

T non è in T, quindi è un insieme che non contiene se stesso; ma, per definizione di T, si trova in T → contraddizione!

Soluzione al paradosso di Russell

Adottare una differente teoria degli insiemi (rispetto a quella “ingenua” proposta da Cantor) dove si impedisca l'autoreferenzialità

Nel sistema di assiomi di Ernst Zermelo si considera un individuo come qualsiasi cosa che non è un insieme. Si costruisce quindi la seguente gerarchia:

Stadio 0 ... insiemi di individui

Stadio 1 ... insiemi i cui elementi sono individui o insiemi dello Stadio 0

Stadio 2 ... insiemi i cui elementi sono individui o insiemi dello Stadio 0 o insiemi dello Stadio 1

…

Stadio n ... insiemi i cui elementi sono individui o insiemi dello Stadio 0 o insiemi dello Stadio 1 ... o insiemi dello Stadio n-1

Così facendo non si otterrà mai un insieme che contenga se stesso, e quindi mai l'insieme di tutti gli insiemi e mai l'insieme di tutti gli insiemi che non sono elementi di se stesso.

Affermazioni vere e affermazioni false

in matematica:

2+4=6

vero

2+2=5

falso

ne esistono di più complicate!

Affermazioni più complesse

- non esistono tre numeri a, b e c tali che $a^n+b^n=c^n$ per qualche n>2

  ultimo teorema di Fermat: vero

  (formulato nel 1637 da Pierre de Fermat, fu dimostrato nel 1994 da Andrew Wiles, dopo 357 anni)

- per ogni numero intero, almeno la metà dei numeri minori ha un numero dispari di fattori primi

  congettura di Pólya: falso

  *(*formulata da George Pólya nel 1919, fu confutata nel 1958 da C.B Haselgrove, dopo 39 anni).

- Una famosissima congettura nel campo della teoria della complessità computazionale:

  P=NP: non si sa, da oltre 40 anni (formulato nel 1971)

  premio per la soluzione: $1.000.000

Il programma di Hilbert e l'incompletezza dimostrata da Gödel

(semplificando molto)

Nel suo "programma", David Hilbert (1862-1943) affrontava, fra le altre cose, la seguente questione:

ogni affermazione matematica si può dimostrare essere vera o essere falsa?

Rispondere affermativamente a questa domanda vuol dire dimostrare che in matematica non ci possono essere paradossi.

Il programma fu proposto da Hilbert intorno al 1920. Tuttavia, nel 1931, Kurt Gödel (1906-1978), dimostrò i cosiddetti teoremi di incompletezza di Gödel. In particolare, il primo teorema dimostra che

certe affermazioni matematiche non possono essere né dimostrate né confutate

Possiamo interpretare il teorema di Gödel come la costruzione effettiva di una funzione non calcolabile, dove, in questo caso, il calcolo riguarda stabilire la verità o falsità di una affermazione.