Calcolabilità

Indice generale

Cardinalità degli insiemi numerici

In questo paragrafo parlerò della cardinalità degli insiemi e in particolare dell'insieme dei numeri naturali (N), dei numeri interi (Z), dei numeri razionali (Q) e dei numeri reali (R).

Dovete scusarmi se la trattazione potrebbe apparire confusa, contorta e poco rigorosa: non sono abituato a fare dimostrazioni matematiche.

Cosa c'entrano gli insiemi numerici con l'Informatica? Questo lo vedremo più avanti, per ora portate pazienza e seguite il mio ragionamento (che poi non è mio ma di Cantor).

Su Wikipedia leggo:

In teoria degli insiemi per cardinalità (o numerosità o potenza) di un insieme finito si intende il numero dei suoi elementi. La cardinalità di un insieme A è indicata con i simboli |A|, #(A) oppure card(A).

Noi useremo |A| perché è più corto.

In pratica se un insieme contiene tre mele la sua cardinalità è 3, se contiene 2 penne la sua cardinalità è 2 e se contiene un gatto la sua cardinalità è 1. Se un insieme contiene zero elementi la sua cardinalità è zero. L'insieme senza elementi si chiama insieme vuoto e si scrive {} e ovviamente non ha senso chiedersi se siano zero mele, zero penne o zero gatti: l'insieme vuoto è l'insieme vuoto.

Come faccio a sapere se due insiemi hanno lo stesso numero di elementi? Qualcuno dirà: basta contarli e se le cardinalità sono uguali significa che hanno lo stesso numero di elementi. La risposta non è sbagliata ma c'è un modo più diretto per farlo.

Immaginiamo di avere un insieme di penne e un insieme di tappi. Per sapere se hanno la stessa cardinalità basta infilare un tappo in una penna: se avanzano tappi significa che la cardinalità dell'insieme dei tappi è maggiore, se avanzano penne è l'insieme delle penne ad avere una cardinalità maggiore, se non avanzano né penne né tappi i due insiemi hanno la stessa cardinalità. Come vedete non ho dovuto contarli. Nel linguaggio matematico quello che sto cercando di fare è trovare una funzione biunivoca tra i due insiemi.

Più formalmente, dagli appunti sulla calcolabilità del prof. Alberto Marchetti-Spaccamela:

Dati due insiemi (non necessariamente finiti) A e B, abbiamo che

i) |A| ≥ |B| se esiste una funzione suriettiva f : A → B

ii) |A| > |B| se esiste una funzione suriettiva f : A → B ma non esiste una funzione suriettiva

g : B → A

iii) |A| = |B| se esiste una corrispondenza biunivoca fra A e B.

Domanda: quanti sono i numeri naturali?

Partendo dallo zero e applicando ripetutamente il successore, come in questo semplice programmino in Ruby

```
n=0
```

```
while true
  puts n
  n=n.succ
end
```

otteniamo tutti i numeri naturali.

Chiaramente il programma non finirà mai di scrivere nuovi numeri (nella realtà non è possibile per i limiti della memoria dei computer reali).

I numeri naturali sono infiniti perché qualunque numero mi venga in mente io posso sempre trovarne uno più grande: basta applicare l'operazione di successore per trovare quello che segue.

Da notare che, poiché scriviamo i numeri in notazione posizionale (purtroppo in base 10), ci sono delle regole (un algoritmo) per sapere sempre come si scrive il successore di un numero, quindi io non solo so che quel numero esiste, so anche come si scrive.

Per esempio so che dopo il 3 viene il 4, dopo il 4 il 5, dopo il 9 il 10 e dopo il 99 il 100. Qualunque numero io scriva, non avrò problemi a scrivere anche il suo successore.

Dopo essermi convinto che i naturali non hanno mai fine, mi pongo una ulteriore domanda: quanti sono i numeri interi (Z)? Si può dare una definizione formale di numeri interi come classi di equivalenze di coppie di naturali. Restando terra terra come piace a noi, i numeri interi sono i naturali e i loro opposti.

I numeri interi comprendono i numeri naturali o, in altre parole, i naturali sono un sottoinsieme proprio dei numeri interi. Se i due insiemi fossero finiti è evidente che, se il primo contiene il secondo, la cardinalità del primo è maggiore O UGUALE alla cardinalità del secondo.

Questo ragionamento vale per gli insiemi finiti, ma varrà anche tra due insiemi con un numero infinito di elementi, come appunto N e Z?

Come spiegavo prima, per verificare se i due insiemi hanno lo stesso numero di elementi quello che devo fare è trovare, sempre che esista, una funzione biunivoca tra i due insiemi, ossia infilare una penna in ogni tappo.

N={0,1,2,3,4,...}

Z={0,1,2,3,..,-1,-2,-3,...}

Basta scrivere Z con un ordine diverso, per esempio partendo dallo zero e alternando i negativi coi positivi

Z={0,-1,1,-2,2,-3,3,...}

e il gioco è fatto.

```
 N    Z
--------
 0 ↔  0
```

```
 1 ↔ -1
 2 ↔  1
 3 ↔ -2
 4 ↔  2
.. ↔ ..
```

In pratica, se riesco a mettere in fila uno dietro l'altro gli elementi di un insieme ho trovato la funzione biunivoca coi naturali e quell'insieme è detto NUMERABILE.

ESERCIZIO (FACILE): scrivi un programma che elenca uno dopo l'altro i numeri interi senza tralasciarne neanche uno.

A questo punto è utile fare una osservazione: Z contiene N e in più ha tutti i negativi, tuttavia i due insiemi hanno lo stesso numero di elementi. Se i due insiemi fossero finiti questo non sarebbe possibile. All'apparenza ciò è paradossale ma non dobbiamo dimenticare che stiamo parlando di insiemi infiniti.

Andiamo oltre: anche l'insieme dei numeri razionali (Q) ha la stessa cardinalità dei numeri naturali?

Per rispondere a questa domanda dobbiamo, come prima, trovare se esiste una funzione biunivoca tra N e Q. Da notare che l'insieme Q è DENSO, ossia tra due numeri razionali ne esiste sempre un altro, cosa non vera per i naturali e per gli interi. Per esempio tra 0 e 1 c'è 1/2, tra 0 e 1/2 c'è 1/4, tra 0 e 1/4 c'è 1/8 e così via. Insomma, SOLO tra 0 e 1 ci sono INFINITI numeri razionali. Pensare che i razionali siano tanti quanti i naturali a questo punto potrebbe apparire un azzardo. Anche qui, quello che dobbiamo fare è trovare, se esiste, una funzione biunivoca tra i due insiemi.

Lasciamo stare per un attimo i numeri razionali e prendiamo le coppie di interi, per esempio (2,3), (5,4), (1,1), (6,4) etc. In fondo il modo migliore per esprimere un numero razionale (non intero) è mediante una frazione, possibilmente ridotta ai minimi termini, come per esempio 2/3, 5/4, 1/1 (meglio 1 e basta), 6/4 (meglio 3/2) e una frazione altro non è che una coppia numeratore/denominatore.

Immaginiamo di riportare sul piano cartesiano tutti i punti aventi solo coordinate intere.

```
(-1, 1) ( 0, 1) ( 1, 1)

(-1, 0) ( 0, 0) ( 1, 0)

(-1,-1) ( 0,-1) ( 1,-1)
```

come per esempio queste coppie.

Posso anche scriverle in forma più compatta così

```
-1/1   0/1   1/1
```

```
-1/0   0/0   1/0


-1/-1  0/-1  1/-1
```

siete d'accordo?

Il metodo che segue l'ho battezzato "visita a spirale" ed è uno degli infiniti modo per "mettere in fila" o se preferite "numerare" i razionali. L'ho scoperto da solo e per questo ne vado molto fiero ma presumo che prima di me sia stato scoperto da qualche migliaio di altre persone.

```
-1/1 ←  0/1  ←  1/1
  ↓              ↑
-1/0    0/0  →  1/0    ...
  ↓                     ↑
-1/-1 → 0/-1 →  1/-1 → 2/-1
```

Si parte da 0/0 e si procede all'infinito tracciando una spirale. E' importante notare che questa procedura non tralascia nessuna coppia di interi.

Ora è facile ottenere la funzione biunivoca seguente:

```
 N     IxI
-----------
 0 ↔   0/0
 1 ↔   1/0
 2 ↔   1/1
 3 ↔   0/1
 4 ↔  -1/1
 … ↔   …
 9 ↔   -1/0
 … ↔   …
```

Ciò dimostra che i naturali e le coppie di interi hanno la stessa cardinalità ossia |N|=|IxI|.

ESERCIZIO (DIFFICILE): scrivi un programma che elenca una dopo l'altra le coppie di interi senza tralasciarne neanche una.

I razionali e le coppie di interi però non sono esattamente la stessa cosa. E' vero che un modo di esprimere i razionali è usare la coppia di interi numeratore/denominatore però ci sono infiniti modi per esprimere lo stesso razionale, per esempio 1/2, 2/4, 3/6, ... e infinite coppie che non rappresentano un razionale, per esempio 1/0, 2/0, 3/0, ... Nella visita a spirale dovremmo saltare infinite coppie di interi.

ESERCIZIO (ANCORA PIU' DIFFICILE): scrivi un programma che elenca SENZA RIPETIZIONI uno dopo l'altro tutti i numeri razionali espressi mediante frazioni senza tralasciarne neanche uno.

Dobbiamo dimostrare in modo un po' più rigoroso che |N|=|Q|. Intanto dimostriamo che le coppie di interi hanno una cardinalità MAGGIORE O UGUALE a quella dei numeri razionali ossia che sono ALMENO quanti i razionali.

Per farlo basta trovare una funzione SURIETTIVA tra le coppie di interi e i razionali. In pratica associamo a ogni coppia di interi il numero razionale espresso da quella coppia ridotta ai minimi termini (o un solo numero intero, se numeratore e denominatore sono uguali) ed escludiamo tutte quelle coppie nella forma i/0. In questo modo non viene tralasciato alcun razionale.

```
IXI                              Q
--------------------------------
0/0, -1/0, 1/0, -2/0, ...
1/1, -1/-1, -2/-2, 2/2, ... →   1
0/1, 0/-1, 0/2, ...          →   0
...                              ...
3/2, -3/-2, 6/4, -6/-4, ... →  3/2
...                              ...
```

ATTENZIONE: l'esistenza di tale funzione suriettiva garantisce che |IxI|≥|Q|: MAGGIORE O UGUALE, NON solamente maggiore. Anche se ci fossero molti elementi del primo insieme che finiscono nello stesso elemento del secondo ed elementi del primo insieme a cui non è associato alcun elemento del secondo, poichè stiamo parlando di insiemi infiniti questo non significa necessariamente che la cardinalità del primo sia maggiore

Come ultimo passaggio dimostriamo che |Q|≥|N|. Anche qui, attenzione, MAGGIORE O UGUALE. Questo è evidente perché N è contenuto in Q ma se proprio volessimo trovare una funzione suriettiva tra Q e N basta associare ogni elemento di Q che è anche in N a se stesso e gli altri numeri razionali non considerarli.

Ricapitolando abbiamo che:

|N|=|IxI|

|IxI|≥|Q|

|Q|≥|N|

Quindi

|Q|≥|N|=|IxI|≥|Q|

e, dato che N e IxI hanno la stessa cardinalità

|Q|≥|N|≥|Q|

ossia

$|Q| \geq |N|$ e allo stesso tempo $|N| \geq |Q|$

ma questo è possibile solo se

$|Q| = |N|$

Da notare che, a differenza di quanto avviene per gli insiemi infiniti, aggiungere un numero finito di elementi a un insieme infinito non ne aumenta la cardinalità.

Arrivati a questo punto si potrebbe pensare che tutti gli insiemi infiniti abbiano la stessa cardinalità: dopo tutto sempre di infinito si tratta. Vediamo se è è vero.

Come ultimo insieme numerico prendiamo in esame i numeri reali (R). I numeri reali sono un sovrainsieme proprio dei naturali ma, come abbiamo già visto per altri insiemi infiniti, ciò non è sufficiente per dire che cardinalità dei reali è maggiore di quella dei naturali.

I numeri reali si possono esprimere in forma decimale con una parte intera ed eventualmente una virgola (gli anglosassoni e i linguaggi di programmazione usano il punto) seguita dalla parte decimale. La parte intera è composta da un numero finito di cifre, la parte decimale può avere anche infinite cifre.

Per esempio 1/2 si può scrivere 0.5, 1/5 è 0.2, 1/10 è 0.1, 1/3 è $0.\underline{3}$, 1/7 $0.\underline{142857}$ e così via. Da notare che i numeri razionali possono anche avere infinite cifre dopo la virgola ma sempre con un periodo che si ripete. La presenza o l'assenza del periodo dipende unicamente dalla base di numerazione usata per scriverli, ad esempio 1/3 in base 3 si scrive 0.1.

Al contrario dei reali razionali, i reali irrazionali (non razionali, ovvero non esprimibili mediante frazioni) possiedono infinite cifre **non periodiche** dopo la virgola.

Poiché R è un soprainsieme proprio di N, ovviamente $|R| \geq |N|$. Per dimostrare che le due cardinalità sono uguali, dovrei dimostrare mediante una funzione suriettiva tra N e R che vale anche $|N| \geq |R|$

Al contrario, per dimostrare che non lo sono, basterebbe dimostrare che tale funzione **non esiste**. Dimostrare l'inesistenza di qualcosa è sempre piuttosto problematico. In genere si parte dal presupposto che quel qualcosa esiste per poi giungere a un paradosso.

Per ora limitiamoci a considerare l'intervallo [0,1] e vediamo se si riesce a trovare almeno una funzione suriettiva tra i naturali e i reali di questo intervallo. Per esempio fanno parte dell'intervallo lo 0 (intero e naturale), 0.1 e $0.\overline{9}$ (razionali) ma anche 0.4537... seguito da altre infinite cifre senza un periodo (irrazionale). Se non ci riusciamo con questo intervallo, figuriamoci con tutti i reali!

Per semplificare la dimostrazione io scriverei i numeri in forma binaria. Funziona lo stesso anche se si scrivono in forma decimale ma in forma binaria è più facile.

Partiamo dal presupposto che tale funzione esista. Io potrei quindi elencare uno dopo l'altro tutti i numeri reali compresi tra zero e uno, eventualmente anche con ripetizioni (non è importante) in questo modo:

```
N    [0,1]
0 → 0.01010…
1 → 0.11111…
2 → 0.01000…
3 → 0.10101…
```

4 → 0.01100…

… → …

Semplifichiamo ulteriormente scrivendo solo la parte dopo la virgola, in modo da avere una matrice infinita di bit:

0 → 01010…

1 → 11111…

2 → 01000…

3 → 10101…

4 → 01100…

… → …

ed evidenziamo la diagonale della matrice

0 → **0**1010…

1 → 1**1**111…

2 → 01**0**00…

3 → 101**0**1…

4 → 0110**0**…

… → …

che, nel caso dell'esempio equivale a 01000… (ovviamente poi prosegue all'infinito).

Non importante ai fini della dimostrazione ma da notare che anche questa stringa binaria prima o poi dovrà figurare in una riga della matrice perché ci devono essere tutte. Per esempio potrebbe essere la terza stringa in corrispondenza del numero 2.

Se ora nego ogni bit della stringa sulla diagonale (01000...) ottengo la stringa 10111… .

Essendo anch'essa una stringa binaria prima o poi dovrà comparire all'interno della matrice

0 → **0**1010…

1 → 1**1**111…

2 → 01**0**00…

3 → 101**0**1…

4 → 0110**0**…

… → …

n → 10111…

… → …

ma questo è impossibile perché, ovunque si trovi, differirà sempre per un bit dalla stringa diagonale in quanto è stata ottenuta negandola! Esiste quindi un elemento del secondo insieme che non fa parte del codominio della funzione. Questo dimostra non esiste una funzione suriettiva tra i due insiemi e pertanto che la cardinalità di N non può essere maggiore o uguale a quella di [0,1] né tantomeno di tutto R. Ne consegue che

|R|>|N|.

Cardinalità dei programmi e dei problemi

La domanda che ci poniamo ora è: quanti sono i programmi in un certo linguaggio di programmazione (per esempio Ruby)? Sono numerabili?

I programmi sorgente, in qualunque linguaggio, non sono altro che stringhe FINITE di simboli. Per esempio il seguente programma in Ruby per calcolare il fattoriale

```
n=gets.to_i
f=1
while n>0
 f=f*n
 n=n-1
end
puts f
```

è una stringa contenente molte lettere, la cifra 1, alcuni simboli matematici, … e il carattere “a capo”. Ovviamente non tutte le stringhe sono programmi in Ruby, quindi l’insieme dei programmi in Ruby è un sottoinsieme proprio dell’insieme delle stringhe.

I simboli possono essere ordinati: per esempio, nella codifica ASCII, utilizzata comunemente per codificare programmi, l’ordine, da sinistra a destra e dall’alto in basso, è il seguente:

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

tratto da https://it.wikipedia.org/wiki/ASCII

Da notare che alcuni simboli non sono stampabili e che è presente anche “a capo” (nella tabella DLE, posizione 10).

Se provassimo a ordinare le stringhe composte da questi simboli utilizzando l’usuale ordine lessicografico non ci riusciremmo. Per esempio, immaginando di partire dalla lettera a, ci troveremmo intrappolati in una situazione come questa:

```
(stringa vuota)
a
aa
aaa
aaaa
aaaaa
…
```

Nel dizionario l’ordinamento lessicografico (alfabetico) funziona perché l’insieme delle parole è finito. L’insieme di tutte le stringhe possibili invece non è finito, pertanto dobbiamo trovare un altro modo per ordinarle.

Si potrebbe mantenere l’ordine lessicografico TRA STRINGHE DELLA STESSA LUNGHEZZA scrivendo prima la stringa di lunghezza zero (la stringa vuota), poi le stringhe di lunghezza uno (quelle formate da un solo carattere) poi quelle di lunghezza due e via di questo passo.

```
LUNGHEZZA 0

(stringa vuota)

LUNGHEZZA 1

…

0

1

2

…

a

b

c

…

LUNGHEZZA 2

…

00

01

02

…

aa

ab

ac

…

zz

…
```

Dato che è possibile elencare una dopo l’altra tutte le possibili stringhe, esiste una funzione biunivoca tra i naturali e l’insieme delle stringhe che pertanto è numerabile. Dato che i programmi in Ruby sono stringhe, basta elencare solo quelle che rispettano la sintassi del linguaggio saltando le altre per ottenere una corrispondenza biunivoca tra i numeri naturali e i programmi in Ruby che quindi sono numerabili.

Segue la traccia di una dimostrazione più rigorosa, come quella vista precedentemente tra i naturali e i razionali.

Dato che c’è una funzione suriettiva tra i naturali e le stringhe $|N| \geq |S|$.

Poiché S contiene P (i programmi in ruby) $|S| \geq |P|$.

Poiché un numero naturale è un programma in Ruby $|P| \geq |N|$.

Riassumendo:

$|N| \geq |S| \geq |P| \geq |N| \Rightarrow |N| \geq |P| \geq |N| \Rightarrow |N| = |P|$

Domanda (facile): Secondo questo ordinamento, qual è il primo programma (posizione 0) in Ruby?

L’insieme di tutti i possibili problemi è numerabile? Esempi di problemi:

- calcolare il fattoriale di un numero naturale;
- determinare il massimo tra due numeri razionali;
- ordinare un array di interi;
- determinare se (true o false) un array è ordinato;
- determinare se (true o false) un numero intero è maggiore di zero;
- determinare se (true o false) una stringa è palindroma.

È abbastanza facile implementare più di un programma per risolvere ognuno di questi problemi.

Problemi di decisione

Ci sono alcuni problemi, detti di decisione, che danno come risultato vero o falso (true o false) o in alternativa 1 o 0. Gli ultimi tre problemi precedentemente elencati sono di questo tipo.

Ovviamente i problemi di decisione sono un sottoinsieme proprio di tutti i problemi. Tra i problemi di decisione ci interessano in particolare quelli sui numeri naturali: essi sono ovviamente un sottoinsieme proprio di tutti i problemi di decisione e quindi di tutti i problemi in generale. Esempi di problemi di decisione sui naturali potrebbero essere i seguenti:

Dato in ingresso un numero naturale determinare se è:

- pari
- dispari
- minore di quattro
- primo
- …

Esempi di programmi in Ruby che risolvono questi problemi di decisione sui numeri naturali sono i seguenti:

- `puts gets.to_i%2==0`
- `puts gets.to_i%2==1`
- `puts gets.to_i<4`
- `(troppo lungo da scrivere)`
- …

Un qualunque problema di decisione sui numeri naturali in pratica consiste nel determinare se un certo numero naturale appartiene o meno a un certo sottoinsieme dei naturali dove l'insieme è rappresentato da una successione infinita di true o false, o meglio, per semplificare, da una stringa binaria di lunghezza infinita dove 1 in posizione n significa che il numero naturale n soddisfa una certa condizione di appartenenza (true), 0 che non la soddisfa (false).

Per esempio, l'insieme dei numeri primi è identificato dalla stringa binaria 0011010100…

0123456789…

0011010100…

Cerchiamo ora di determinare se la cardinalità dei programmi in Ruby è maggiore o uguale a quella dei problemi di decisione sui numeri naturali trovando una funzione suriettiva dal primo al secondo insieme, come quella nella tabella che segue (la tabella è solo un esempio):

```
Naturali    Problema di decisione sui naturali
            01234…
0        →  01010… (dispari)
1        →  11111… (naturale)
2        →  01000… (uguale a uno)
3        →  10101… (pari)
4        →  00110… (primo)
5        →  01110…
…
```

Vi ricorda niente?

Se applichiamo il metodo della diagonale utilizzato precedentemente per dimostrare la non esistenza di una funzione suriettiva tra i naturali e i reali scritti in forma binaria appartenenti all'intervallo [0,1] dimostriamo analogamente che non esiste una funzione suriettiva tra i programmi e i problemi di decisione sui naturali: da ciò ne consegue che la cardinalità dei problemi di decisione sui numeri naturali è maggiore della cardinalità dei numeri naturali e quindi dei programmi in Ruby (o in qualunque altro linguaggio). Dato che i problemi di decisione sui naturali sono un sottoinsieme proprio dei problemi in generale chiaramente la cardinalità dei programmi è minore della cardinalità dei problemi.

|PROBLEMI|≥|PROBLEMI DI DECISIONE SU N|>|N|=|PROGRAMMI| =>

|PROBLEMI|>|PROGRAMMI|

In pratica ci sono una infinità più che numerabile di problemi non risolvibili mediante programmi perché i problemi sono più dei programmi, che sono una infinità numerabile. Questo risultato è notevole perché pone dei limiti a ciò che è possibile calcolare per via algoritmica.

Il problema della fermata

Il problema della fermata consiste nel determinare se (true o false) un certo programma terminerà o meno con un certo input.

Vi ricordate il programma correttore che assegna in maniera automatica i voti alle verifiche? Per funzionare il programma correttore deve eseguire uno ad uno i programmi implementati dagli studenti. Spesso capitava che, con un certo input, il programma di uno studente entrava in un ciclo infinito e, di conseguenza, il correttore non terminava mai la sua esecuzione (all'epoca dovevo terminare tutto io con Ctrl+c, mettere il programma in una blacklist e ricominciare daccapo). Per ovviare all'inconveniente ho modificato il programma correttore in modo da terminare il programma dello studente se questo non fornisce il risultato entro un certo lasso di tempo prestabilito. Ovviamente c'è la possibilità (molto remota data la semplicità dei programmi) che il programma dello studente, con un po' più di tempo a disposizione, porti a termine la sua esecuzione fornendo un risultato.

Il programma ha solo bisogno di un po' più di tempo o non terminerà mai?

Sarebbe bello se esistesse un programma in grado di determinare a priori se un altro programma si fermerà o meno con un certo input. Purtroppo questo programma non esiste.

Sia L un linguaggio di programmazione. Non esiste un programma Q scritto in L che, con input una coppia (p, i), dove p è la stringa che descrive un programma P scritto in L e i è una stringa di simboli di input per P, termina sempre in tempo finito e decide se P termina o no con input i.

Dagli appunti sulla calcolabilità del prof. Alberto Marchetti-Spaccamela

Dimostriamo la non esistenza di questo programma.

P e Q sono due programmi scritti nello stesso linguaggio L, per esempio Ruby. Q determina se P termina o meno (non termina o termina con un errore) con un certo input i e per farlo prende in ingresso il codice sorgente p del programma P e l'input i.

`P(i)` **`termina`** → `Q(p,i) dà` **`true`**

`P(i)` **`non termina`** → `Q(p,i) dà` **`false`**

Non meravigliatevi dell'esistenza di programmi che hanno come input il codice di altri programmi: è una situazione piuttosto comune. Pensate all'interprete ruby che traduce programmi scritti in Ruby in qualcosa di più vicino alla macchina su cui viene eseguito il programma.

Ora, per semplicità, consideriamo solo il caso in cui l'input i coincide col codice p del programma P, ossia

`P(p)` **`termina`** → `Q(p,p) dà` **`true`**

`P(p)` **`non termina`** → `Q(p,p) dà` **`false`**

Ammetto che è una cosa un po' strana, ma serve per evitare di dipendere da un input particolare. Quando un programma prende in ingresso due stringhe uguali tanto vale "passargliene" solo una. Inoltre, quando l'input di ogni programma P è la stringa col suo stesso codice sorgente, è inutile "passargli" tale stringa. Possiamo pertanto semplificare in questo modo:

`P` **`termina`** → `Q(p) dà` **`true`**

`P` **`non termina`** → `Q(p) dà` **`false`**

In pratica ci limitiamo al caso in cui Q determina se il programma P con input il suo stesso codice sorgente p terminerà fornendo un risultato oppure no.

Ora consideriamo il programma R il cui codice sorgente è la stringa r. **R è uguale a Q con l'aggiunta in fondo di un ciclo infinito se l'output di Q è true**.

`P` **`termina`** → `Q(p) dà` **`true`** → `R(p)` **`non termina`** `(ciclo infinito)`

`P` **`non termina`** → `Q(p) dà` **`false`** → `R(p)` **`termina`** `(dà false)`

In pratica se il programma P termina R non termina, se P non termina R termina (restituendo false).

Ora cerchiamo di determinare se R termina o meno:

`R` **`termina`** → `Q(r) dà` **`true`** → `R` **`non termina`** `(ciclo infinito)`

`R` **`non termina`** → `Q(r) dà` **`false`** → `R` **`termina`** `(dà false)`

il che è chiaramente paradossale, quindi Q, il programma che decide se un qualunque altro programma si fermerà o meno, non può esistere.

DA COMPLETARE! NON STUDIARE

Prendiamo per esempio un insieme S e indichiamo con P(S) l'insieme delle parti di S ossia l'insieme di tutti i sottoinsiemi di S.

Per esempio,

se S={} allora P(S)={{}}

se S={a,b} allora P(S)={{},{a},{b},{a,b}}

se S={0,1,2} allora P(S)={{},{0},{1},{2},{0,1},{0,2},{1,2},{0,1,2}}.

Da notare che l'insieme delle parti di un insieme finito ha ovviamente una cardinalità maggiore dell'insieme di partenza

P(S)|>|S|

e in particolare

|P(S)|=2^|S|

(trovate dimostrazione anche su Wikipedia).

Anche negli INSIEMI INFINITI l'insieme delle parti ha una cardinalità maggiore dell'insieme di partenza, ossia

P(S)|>|S|

vale anche per gli insiemi infiniti?

Intanto bisogna dimostrare che la cardinalità dell'insieme delle parti è MAGGIORE O UGUALE della cardinalità dell'insieme di partenza, ossia che |P(S)|≥|S|

Per farlo basta trovare UNA QUALUNQUE funziona suriettiva da P(S) a S.

Questo è facile, per esempio basta associare agli insiemi che contengono un solo elemento l'elemento che contengono:

P(S) S

{}

{a} → a

{b} → b

{c} → c

... ...

{a,b}

...

e abbiamo dimostrato che

$|P(S)| \geq |S|$

Se S e di conseguenza P(S) fossero finiti, dato il gran numero di elementi di P(S) a cui non corrispondono elementi di S avrei giò dimostrato che la cardinalità dell'insieme delle parti è maggiore in senso stretto.

Per gli insiemi infiniti, come abbiamo visto anche in precedenza, ciò non è sufficiente perché le cardinalità dei due insiemi potrebbero anche coincidere (= invece di ≥).

Per completare la dimostrazione devo anche dimostrare che NON ESISTE una funzione suriettiva da S a P(S).

Poniamo per assurdo che questa funzione esista e che si chiami f, come per esempio nel FALLIMENTARE TENTATIVO che segue.

S P(S)

... → {}

a → {a}

b → {b}

c → {c}

... → {a,b}

... → T

Se x è un qualunque elemento di S e f(x) è un elemento di P(S), costruisco un insieme T che contiene tutti quegli elementi che non appartengono a f(a). T è dunque dicerso da qualunque altro insieme di P(S)