Teoria della calcolabilità

Lezioni di Antonino Salibra

Abstract

Appunti dalle lezioni a cura di Paola Urbani e Marco Lavazza Seranto

Teoria della Calcolabilità, paradossi e linguaggi formali. Tesi di Church, esempi di funzioni non calcolabili: Halting Problem e Problema della Correttezza, dimostrazione della non calcolabilità dell'Halting Problem. Linguaggi: macchine di Turing, paradigma funzionale alpha (PFA). PFA: costruttori, operatori, connettori. PFA: funzioni elementari, funzione esponenziale (iteratore), funzioni segno e segno negato, Principio di Induzione e ricorsione primitiva, esempi ed esercizi.

Part I

lezioni dal 20 settembre al 04 ottobre 2010

Contents

1 Introduzione

La teoria della calcolabilità, della computabilità, e della ricorsione cerca di comprendere cosa può essere effettivamente computato, ovvero quali funzioni ammettono un procedimento di calcolo automatico per ricavarne i valori. L'obiettivo principale è quello di rendere rigorosa (matematicamente formale) l'idea intuitiva di "funzione calcolabile" e scoprire quali siano le possibilità ed i limiti delle cose che ci proponiamo di calcolare. Questa disciplina è comune sia alla matematica che alla computer science. Da una parte l'approccio è quello di approfondire il concetto di calcolabile cercando di individuare le categorie di problemi teoricamente risolvibili e dall'altro mappare questo concetto su cosa i computer sono in grado di elaborare in linea di principio, ovvero senza restrizioni fisiche come, per esempio, costi, tempo, spazio di memoria. Per tali ragioni la teoria della calcolabilità è strettamente legata con la teoria della complessità il cui scopo è quello di vincolare l'idea di calcolabile ai suddetti limiti con particolare attenzione al tempo e allo spazio. Un altro importante aspetto è quello di definire matematicamente il concetto di algoritmo in modo che i programmi possano essere concretamente pensati in termini di oggetti matematici e più precisamente come funzioni che, preso un determinato input, ritornato un risultato.

Nell'agosto del 1900, il matematico tedesco David Hilbert[1] partecipò al Congresso internazionale di matematica di Parigi e, nel suo intervento, presentò una lista di problemi che la matematica era chiamata a risolvere. La lista contava ventitré problemi che apparivano, allo stato della matematica di allora, irrisolti. Il secondo di questi problemi era l'Entscheidungsproblem (letteralmente "problema della decisione"): dimostrare cioè, come si era fatto per gli assiomi della geometria euclidea, che gli assiomi dell'aritmetica dei numeri reali erano anch'essi coerenti. Questa questione coinvolgeva direttamente i fondamenti stessi della matematica. Fino ad allora le prove di non-contraddittorietà erano sempre state prove di coerenza relativa, cioè avevano semplicemente ridotto la coerenza di un certo sistema di assiomi a quella di un altro. Hilbert si rese conto che con l'aritmetica non si poteva più fare riferimento a un altro sistema di assiomi, si era giunti cioè al fondamento logico della matematica e a quel punto bisognava affrontare il problema in termini del tutto generali, non più relativi.

Nello stesso periodo, alcuni logici matematici (Frege[2], Cantor[3]) avevano iniziato a ipotizzare e a costruire una teoria, il logicismo, secondo la quale

[1] David Hilbert (Königsberg, 23 gennaio 1862 – Gottinga, 14 febbraio 1943) è stato un matematico tedesco. È stato uno dei più eminenti ed influenti matematici del periodo a cavallo tra il XIX secolo e il XX secolo.

[2] Friedrich Ludwig Gottlob Frege (Wismar, 8 novembre 1848 – Bad Kleinen 26 luglio 1925) è stato un matematico, logico e filosofo tedesco, padre della logica matematica moderna e della filosofia analitica, nonché studioso di epistemologia, di filosofia della matematica e di filosofia del linguaggio.

[3] Georg Ferdinand Ludwig Philipp Cantor (San Pietroburgo, 3 marzo 1845 – Halle, 6 gennaio 1918) è stato un matematico tedesco, padre della moderna teoria degli insiemi. Cantor ha allargato la teoria degli insiemi fino a comprendere al suo interno i concetti di numeri transfiniti, numeri cardinali e ordinali.

l'aritmetica, in quanto costituita da proposizioni analitiche, doveva essere riducibile alla sola logica. Russel[4], con il suo famoso paradosso destabilizzò questi tentativi, minando alle fondamenta la teoria degli insiemi "ingenua" fino ad allora utilizzata e su cui si basava tutta la matematica ed anche il logicismo.

Brevemente il paradosso (metonimia) di Russel introduce il concetto di autorefernza nella definizione degli assiomi.

Sia R un insieme così definito: $R = \{X : X \notin X\}$ ovvero R sia l'insieme degli insiemi che non contengono se stessi.

Questa definizione porta in sè una evidente contraddizione: se R non contiene se stesso, allora fa parte degli insiemi che non contengono se stessi e quindi si deve appartenere, e viceversa se contiene se stesso allora non fa parte degli insiemi che non contengono sè stessi e quindi non dovrebbe appartenersi. In formule: se $R \in R \iff R \notin R$.

La contraddizione nasce dalla possibilità offerta dalla teoria insiemistica csidetta "ingenua", che consente l'autorefernza.

Per eliminare le contraddizione si dovettero introdurre delle limitazioni nella definizione degli insiemi.

Ernest Zermelo, Abraham Fraenkel e Thoralf Skolem, in tempi successivi diedero vita al sistema assiomatico degli insiemi che porta il nome Zermelo-Fraenkel su cui si basa tutta la matematica ordinaria secondo formulazioni moderne.

Il risultato fondamentale che apre definitivamente la strada alla nascita dell'informatica è dovuto a Goedel[5] (1931). In questo fondamentale lavoro, Goedel dimostra che esistono teoremi dell'aritmetica che non sono decidibili nell'aritmetica stessa. In particolare la verità di una formula non risulta dimostrabile in modo effettivo (ovvero decisa finitamente) nell'aritmetica.

L'importanza di questo risultato sta nel fatto che esso è del tutto indipendente dal particolare sistema formale (linguaggio) scelto, purchè sufficientemente espressivo da rappresentare l'aritmetica stessa. In questo senso i risultati di incompletezza di Goedel risultano validi per tutti i sistemi formali derivanti da una assiomatizzazione degli insiemi, aggiungendo un qualsiasi numero finito di assiomi, purchè questi non generino inconsistenze.

In questo contesto nasce quindi la necessità di approfondire la natura stessa del calcolo logico-formale e di come da questo sia possibile derivare in modo sistematico enunciati la cui verità sia decidibile. Il fallimento quindi dell'idea di ridurre l'intera matematica ad un mero calcolo automatico a partire da assiomi e regole di inferenza, costituisce il terreno ideale per lo studio di ciò che

[4]Bertrand Arthur William Russell, terzo conte Russell (Trellech, 18 maggio 1872 – Penrhyndeudraeth, 2 febbraio 1970), è stato un filosofo, logico e matematico gallese. Fu anche un autorevole esponente del movimento pacifista e un divulgatore della filosofia. In molti hanno guardato a Russell come a una sorta di profeta della vita creativa e razionale; al tempo stesso la sua posizione su molte questioni fu estremamente controversa.

[5]Kurt Gödel (Brno, 28 aprile 1906 – Princeton, 14 gennaio 1978) è stato un matematico, logico e filosofo statunitense di origine austro-ungarica, noto soprattutto per i suoi lavori sull'incompletezza delle teorie matematiche. Gödel è ritenuto uno dei più grandi logici di tutti i tempi insieme a Frege e Aristotele; le sue ricerche ebbero un significativo impatto, oltre che sul pensiero matematico e informatico, anche sul pensiero filosofico del XX secolo.

effettivamente è calcolabile in questo senso.

La necessità di formalizzare il processo di calcolo, sia mediante la defnizione di una macchina calcolatrice che attraverso sistemi formali di calcolo ha portato all'analisi della calcolabilità di Turing[6] e Church[7] nel 1936.

Secondo la tesi di Church:

Una funzione *è calcolabile se e solo se è risolvibile secondo uno qualsiasi dei linguaggi formali introdotti.*

2 Terminologia, formalismi e definizioni

Un alfabeto $A = \{a_1, a_2, ..., a_n\}$ è un insieme finito di n elementi detti simboli dell'alfabeto. Ad un alfabeto appartengono anche gli eventuali segni di interpunzione e/o di collegamento e lo spazio vuoto.. Con un alfabeto di questo tipo è possibile costruire delle stringhe di simboli, mettendoli in sequenza. L'insieme di tutte le stringhe generabili a partire da A costituisce un insieme infinito che chiameremo: A^*.

Che si tratti di un insieme di cadinalità infinita è facilmente verificabile anche prendendo un vocabolario di due sole lettere, diciamo $A = \{a, b\}$ Le stringhe che possiamo generare prendendo solo la lettera a sono infinite. Iniziamo dalla parola che contiene una sola a, poi due, poi tre etc. :

a
aa
aaa
$aaaa$
.....

Possiamo costruire così parole di qualsiasi lunghezza senza mai utilizzare la seconda lettera.

Risulta meno immediato il fatto che l'insieme A^*sia numerabile, ovvero si possa associare a ciascuna stringa un numero naturale in maniera biunivoca. Ciò corrisponde a creare una funzione bijettiva tra l'insieme dei numeri naturali e le stringhe generabili con l'alfabeto disponibile. Cominciamo a costruire le stringhe con un solo simbolo, e allineiamole in ordine lessicografico; associamo ad ogni stringa un numero naturale, poi creiamo le stringhe con due simboli, ordiniamole e associamo ad ogni stringa un numero naturale, cominciando dal successivo dell'ultimo numero usato. Ricordiamo che le possibili stringhe di n

[6] Alan Mathison Turing (1912-1954) matematico e logico inglese. I suoi metodi consentirono agli alleati, duarante al seconda guerra mondiale, di decrittare i messaggi segreti tedeschi, generati con una macchina, l'Enigma, reputata capace di generare codici indecifrabili. Morì suicida per aver addentato una mela intrisa di cianuro a seguito di una condanna per omosessualità. Oggi è considerato uno dei più importanti scienziati del XX secolo.

[7] Alonzo Church (Washington, 14 giugno 1903 – Hudson, 11 agosto 1995) è stato un matematico e logico statunitense che fu in parte responsabile della fondazione dell'informatica (o computer science). Inventore dell lambda calcolo, un sistema di riscrittura formale sviluppato per analizzare formalmente le definizioni di funzioni, le loro applicazioni ed è uno strumento interessante per studiare anche fenomeni di ricorsione.

simboli, costruite a partire da un alfabeto di l simboli, sono esattamente l^n. Continuando così, otterremo un insieme infinito di stringhe, ciascuna associata in maniera biunivoca ad un numero naturale.

nessun simbolo	λ									
numero associato	0									
parole lunghe 1 simbolo	a	b								
numero associato	1	2								
parole lunghe 2 simboli	aa	ab	ba	bb						
numero associato	3	4	5	6						
parole lunghe 3 simboli	aaa	aab	aba	abb	baa	bba	bbb			
numero associato	7	8	9	10	11	12	13			
.....	....	....	....	....	....	....	....	....	....	....

Un programma in generale, può essere visto come una procedura che calcola una funzione $f : \mathbb{N} \to \mathbb{N}$ dove l'input e l'output del programma sono dei sottoinsiemi di N.

Chiamiamo F^* l'insieme di tutte le funzioni possibili.

Una funzione che può essere calcolata da un programma si dice calcolabile**.**

Un programma scritto in un qualsiasi linguaggio, non è altro che una stringa di simboli di quel particolare alfabeto scelto per scrivere il programma. Naturalmente non tutte le stringhe possono rappresentare un programma, perchè prive di significato rispetto al linguaggio, qualsiasi esso sia, che codifica quella particolare procedura-funzione. Quindi l'insieme dei programmi sintatticamente corretti rappresenta un sottoinseme dell'insieme delle stringhe.

Tramite la corrispondenza numeri-stringhe descritta, possiamo pensare di individuare il programma con il numero associato a quella particolare stringa . Possiamo indicare sinteticamente che P_i è il programma individuato dalla stringa i-esima e con P^* l'insieme dei programmi possibili ovvero sintatticamente corretti. Poichè i programmi sono soluzioni algoritmiche di problemi, ciò significa che questi ultimi sono enumerabili, come i programmi che li risolvono.

Chiamiamo F_r^* l'insieme delle funzioni risolubili tramite un programma P.

Il fatto che esistono funzioni $f : N \to N$ che non godono della proprietà di essere calcolabili, e questo lo vedremo presto con qualche esempio, è una dimostrazione implicita del fatto che la cardinalità di F^* di un ordine di grandezza superiore alla cardinalità di P^*.

Per comodità d'ora in avanti lavoreremo con i numeri naturali come alfabeto.

Per dimostrare che la cardinalità di F^* è maggiore di quella di F_r^* , insieme delle funzioni calcolabili, consideriamo intanto come esempio la seguente funzione ($f : \mathbb{N} \to \mathbb{N}$):

- $f(x) = x^2$

  Tale funzione è caratterizzata, e può quindi venir rappresentata, dalla sequenza dei suoi valori di output: *0, 1, 4, 9, 16, 25, 36, ...*

Contando anche lo zero, il risultato dell'applicazione della funzione ad un numero k consiste nell'individuare nella sequenza il numero di posizione *k-esima+1.* Per esempio per *x=5* dovremmo considerare il sesto elemento *(25).*

Consideriamo quindi l'esempio successivo:

- $f : \mathbb{N} \to \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ tale che, per esempio, sia: $f(1) = 1, \cdots f(9) = 9, f(10) = 0, \cdots f(19) = 9$.

  Come nell'esempio precedente anche questa funzione può venir rappresentata tramite la sequenza dei suoi output.

  In questo caso la sequenza sarà: *0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,.....*

  Risulta immediato concludere che qualsiasi funzione $f : \mathbb{N} \to \{0 \dots 9\}$ può venir rappresentata da infinite ripetizioni del codomino, non necessariamente nello stesso ordine.

Un numero reale nell'intervallo aperto $(0, 1)$ può quindi venir correlato ad una funzione $f : \mathbb{N} \to \{0 \dots 9\}$, semplicemente aggiungendo uno 0 seguito da una virgola in testa alla sequenza dei valori, e togliendo tutte le virgole nella sequenza.

La funzione del secondo esempio si correla al numero reale 0,1234567890123... che è proprio un numero nell'intervallo aperto $(0, 1)$.

Le funzioni $f : \mathbb{N} \to \{0 \dots 9\}$ possono essere quindi messe in corrispondenza con i numeri reali dell'intervallo aperto $(0, 1)$, perchè ogni numero reale può essere rappresentato da una sequenza di numeri da *0* a *9* , e quindi dalla funzione che genera tale sequenza.

Sappiamo già che l'insieme dei reali in un qualsiasi intervallo, e quindi anche in quello che stiamo considerando non è numerabile, in quanto la sua cardinalità è maggiore di quella di N . Per convincersi basti considerare che supponendo, per assurdo, di compilare un elenco dei numeri reali, posso sempre costruirne uno non compreso in quell'elenco. Di conseguenza a ciò, anche l'insieme delle funzioni $f : \mathbb{N} \to \{0 \dots 9\}$ non è numerabile e quindi ha cardinalità maggiore di $\mathbb{N}$. per estensione quindi anche F^* , che comprende anche tutte le funzioni del tipo appena definito, ha cardinalità maggiore di di $\mathbb{N}$.

La conclusione viene quindi immediata. Poichè le funzioni risolubili F_r^* sono in corrispondenza biunivoca con i programmi che le risolvono, e questi sono numerabili, anche l'insieme F_r^*è numerabile.

La totalità delle funzioni F^*, l'abbiamo appena dimostrato, è non numerabile, in quanto ha la cardinalità dei reali. Di conseguenza le funzioni calcolabili sono molto meno delle funzioni possibili. La conseguenza di ciò è che i problemi che ammettono una soluzione algoritmica sono molto meno di quelli che non la ammettono. La maggior parte dei problemi non ammette una soluzione algoritmica. Fortuna vuole che la maggioranza dei problemi di nostro interesse fanno parte della minoranza dei problemi calcolabili.

I due problemi seguenti, l'halting problem, o problema della fermata, e il problema della correttezza, fanno parte del grande insieme di problemi non calcolabili.

3 Formalismi e paradigmi

Adottando la tesi di Church una funzione si dice calcolabile se esiste un programma che la calcola. Un programma è la trascrizione dell'algoritmo risolutivo tramite un linguaggio formale (formalismo) qualsiasi. In questo senso si può dire che tutti i formalismi sono equivalenti, cioè hanno la stessa potenza risolutiva.

Vedremo in seguito due paradigmi di programmazione, uno imperativo (macchine di Turing ed a registri) ed uno funzionale.

Entrambi hanno la stessa potenza risolutiva perché il concetto di funzione calcolabile è assoluto.

Sia P_n un programma qualsiasi. Possiamo affermare che: P_n calcola $\Phi_n : \mathbb{N} \to \mathbb{N}$, dove $\Phi_n$rappresenta la funzione calcolabile (parziale[8]) risolta dal programma P_n.

Il programma P_n prenderà in input un numero e restituirà in output un numero.

Ma il numero in input, individua a sua volta un programma, così come il numero in output. Ciò significa che, sotto questo punto di vista, un programma trasforma programmi in input in programmi in output.

Sarà possibile dare in input ad un programma la stringa che rappresenta se stesso.

Consideriamo un programma P_n ;

Se il programma P_n , ricevendo in input la stringa k, produce come risultato la stringa k', diremo che il programma P_n termina su k e scriveremo: $P_n \downarrow k$,

Viceversa diremo che il programma P_n non termina e scriveremo: $P_n \uparrow k$.

4 Halting problem

L'halting problem, o problema della fermata si definisce come segue:

Dato un programma qualsiasi P_n e una qualsiasi stringa di input k, esiste un programma H che, preso in input il programma P_n e la stringa di input k, sia in grado di decidere se il programma termina o meno su k?

Il programmma H dovrebbe comportarsi formalmente come segue:

$$H = \begin{cases} true\ (1) & se\, P_n \downarrow k \\ false\ (0) & se\, P_n \uparrow k \end{cases}$$

ricordiamo che in termini informatici l'asserzione viene spesso rappresentata con l'1 e la negazione con lo 0.

L'halting problem ammette una risposta negativa, cioè non è possibile realizzare il programma H.

La dimostrazione di ciò procede per assurdo.

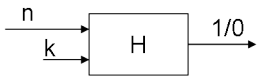
[8] La funzione è parziale perchè il set di numeri disponibili in una macchina fisica è finito

Supponiamo che esista il programma H. In tal caso esisterà una funzione $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ calcolabile definita come segue:

$$f(n,k) = \begin{cases} 1 & se\, P_n \downarrow k \\ 0 & se\, P_n \uparrow k \end{cases}$$

di cui il programma H costituisce l'algoritmo risolutivo.

Possiamo rappresentare graficamente il programma H che calcola f come segue:

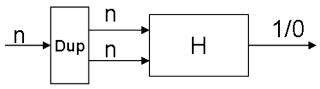


Supponiamo che alla funzione *f*, vengano dati in ingresso, la stringa che corrisponde al programma P_n e si voglia sapere se P_n termina quando, come dato in input, gli viene passata la stringa n stessa che lo definisce.

Al posto di k dovremo passare la stringa n. Si ottiene così una funzione $g(n) : \mathbb{N} \to \mathbb{N}$, calcolabile perchè lo è *f*, definita come segue:

$$f(n,n) = g(n) = \begin{cases} 1 & se\, P_n \downarrow n \\ 0 & se\, P_n \uparrow n \end{cases}$$

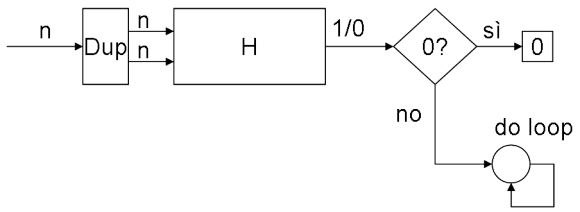
Il programma *H'* che calcola *g(n)* puo venir rappresentato graficamente come segue:



Possiamo costruire allora una nuova funzione *t(n)*, anche questa calcolabile, derivata da *g(n)*, definita come segue:

$$t(n) = \begin{cases} \uparrow & se\, P_n \downarrow n \\ 0 & se\, P_n \uparrow n \end{cases}$$

la quale, presa in ingresso la stringa n, che definisce il programma P_n, dà come risultato 0 se P_n non termina su n, e diverge altrimenti. La funzione *t(n)* è una specie di flip-flop logico. Il programma *H"* che realizza $t\,(n)$può essere disegnato come segue:



Poichè *t(n)* è calcolabile, allora esisterà una stringa, che chiamiamo *r,* che definisce il programma $P_r = H''$ che calcola *t(n).*

Poiche il programma H si applica a qualsiasi coppia di stringhe n e k, il programma *H"* si può applicare a qualsiasi stringa n, e quindi anche alla stringa r che lo definisce.

Cosa succede se diamo in pasto ad *H"* la stringa r che lo definisce? Avviene una specie di corto circuito: i casi sono due o il programma $P_r = H''$ termina o non termina. Se termina, allora H, contenuto in H", restituisce un 1 e questo fa sì che H" vada in un loop infinito e non termini.

Se non termina, allora H, contenuto in H", restituisce uno 0 e questo fa sì che H" termini restituendo uno 0.

In altre parole: se $P_r \downarrow r \iff P_r \uparrow r$ e questa è una contraddizione, da cui deriva l'impossibilità dell'esistenza del programma H che decide se un altro programma termini o meno.

5 Macchine di Turing

La programmazione imperativa è un paradigma di programmazione secondo cui un programma viene inteso come un insieme di istruzioni (dette anche direttive o comandi), ciascuna delle quali può essere pensata come un "ordine" che viene impartito alla macchina virtuale del linguaggio di programmazione utilizzato.

Le macchine di Turing, ideate da Turing intorno al 1937, sono un esempio di programmazione imperativa.

Per capire come funziona una macchina di Turing (TM), pensiamo di dover calcolare la somma di due numeri naturali, per esempio $57 + 48$. Possiamo pensare di enunciare un algoritmo che esegua la somma semplicemente replicando le tecnica imparata alle elementari. Con questa tecnica mettermmo in colonna i due numeri, ci posizioneremmo sul *7* del *57*, leggremmo la cifra, la memorizzeremo, poi ci sposteremmo sull'8 del 48, memorizzandola, poi ci sposteremmo sotto ulteriormente, ci accorgeremmo che non ci sono altre cifre, quindi eseguiremmo l'operazione tra le due cifre memorizzate, scrivendo il risultato di $(7 + 8)\, mod 10$ nella posizione vuota, e riporteremmo un *1*, per le operazioni successive; poi potremmo procedere cone le due cifre successive ricordando di sommare anche il riporto.

Una TM può implementare il semplice alogritmo descritto. Una TM consiste in una memoria infinita (teoricamente) ed una testina che può compiere solamente tre operazioni:

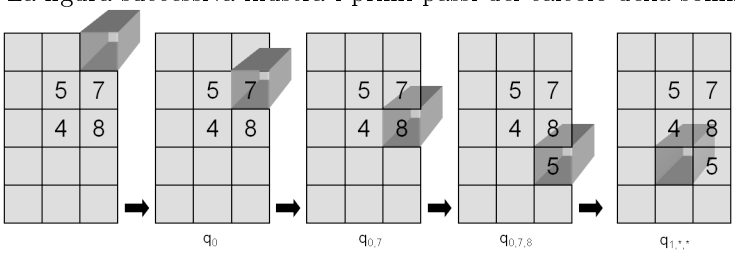
- leggere il contenuto di una cella di memoria
- scrivere in una cella di memoria
- spostarsi da una cella di memoria ad una contigua secondo gli assi della memoria .

La memoria può espandersi in una sola direzione equivalente ad un nastro di telescrivente, in due dimensioni, equivalente ad un foglio quadrettato ed in generale in più dimensioni.

Si può dimostrare che una macchina di questo tipo, se opportunamente programmata, può calcolare tutto ciò che è calcolabile, quindi può implementare qualsiasi algoritmo.

5.1 Alcuni esempi

Chiamiamo q lo stato in cui si trova la macchina. Lo stato iniziale lo chiameremo q_0. La figura successiva illustra i primi passi del calcolo della somma $57 + 48$.



Il funzionamento è il seguente:

- se la macchina è nello stato iniziale e legge una cifra (7), riscrive la stessa cifra, cambia stato e si sposta in direzione sud: $q_0 \rightarrow q_{0,7}$ sud
- spostandosi a sud, la testina legge una cifra (8), riscrive la stessa cifra, cambia stato e si sposta in direzione sud: $q_{0,7} \rightarrow q_{0,7,8}$ sud
- l'ultimo spostamento porta la testina su una casella vuota (blank = b), a questo punto scrive il risultato$(7 + 8)\, mod 10 = 5$ memorizza il risultato nella cella vuota, registra il riporto (1), cambia di stato e si sposta a ovest:

  $q_{0,7,8} \rightarrow q_{1,*,*}$ ovest dove i due asterischi indicano che la testina deve spostarsi di due caselle a vuoto, senza leggere nè scrivere, verso nord in modo da ricominciare le operazioni.

La sequenza degli stati è quindi la seguente:

Configurazione di partenza		Azioni e nuovo stato		
Stato di provenienza	Cosa legge	Cosa scrive	Nuovo stato	Direzione spostamento
q_0	cifra (c)	c	$q_{0,c}$	S (sud)
$q_{0,c}$	cifra (d)	d	$q_{0,c,d}$	S
$q_{0,c,d}$	blank	$(c + d)\, mod 10$	$q_{r,*,*}$	Ovest (O)

Con r si intende il riporto che nella somma di due cifre può essere 1 o 0.

5.2 Descrizione formale

Una TM è un oggetto $M = (Q, q_0, A, P)$dove:

Q: è l'insieme degli stati che può assumere

q_0: è lo stato iniziale

q_f: è lo stato finale in cui la testina si ferma

A: è l'alfabeto dei caratteri che la testina può leggere o scrivere e che comprende sempre anche il blank

P: è il programma che a sua volta è un insieme di quintuple del tipo $(q, a, b, q', direzione)$dove:

- q: è lo stato di partenza
- a: è ciò che legge la testina
- b: è ciò che la testina scrive
- q': è lo stato in cui passo (nuovo stato) prima di spostarmi
- *direzione*: è la direzione in cui la testina deve spostarsi dopo aver eseguito un'istruzione: avanti, indietro nel caso di una macchina bidimensionale; N,S,E,O nel caso di una macchina bidimensionale, come quella descritta nell'esempio.

I primi due elementi delle quintuple sono fondamentali per il funzionamento di una TM. Infatti dato uno stato q e una lettura a è univocamente definita l'azione successiva: la scrittura, il nuovo stato e la direzionee dello spostamento. Per evitare ambiguità la coppia (q, a) non deve duplicarsi lungo il programma. Un linguaggio che mantiene questa proprietà si dice deterministico.

Esempio: Per completezza trascriviamo qui il programma completo di una TM bidimensionale relativo all'esempio della somma dei due numeri 57 e 48.

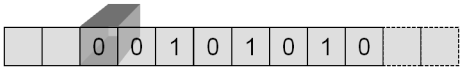
$P(57 + 48) =$

1. $q_0, 7, 7, q_{0,7}, S$ la testina legge il 7, riscrive il 7 e si sposta a sud
2. $q_{0,7}, 8, 8, q_{0,7,8}, S$ la testina legge l'8, riscrive l'8 e si sposta a sud
3. $q_{0,7,8}, b, 5, q_{1,O}, O$ la testina trova una casella vuota e scrive il risultato 5 e riporta l'1, poi si sposta a ovest
4. $q_{1,O}, b, b, q_{1,N}, N$ la testina trova una casella vuota e si sposta a nord
5. $q_{1,N}, 4, 4, q_{1,N,N}, N$ la testina legge 4, riscrive 4 e si sposta ancora a nord
6. $q_{1,N,N}, 5, 5, q_{1,5}, S$ la testina legge il 5, riscrive il 5 e si sposta a sud
7. $q_{1,5}, 4, 4, q_{1,5,4}, S$ la testina legge il 4, riscrive il 4 e si sposta a sud
8. $q_{1,5,4}, b, 0, q_{1,O}, O$ la testina legge blank, scrive il risultato 0 tenendo conto del riporto, poi si sposta a ovest

9. $q_{1,O}, b, b, q_{1,N}, N$ la testina trova una casella vuota e si sposta a nord
10. $q_{1,N}, b, b, q_{1,N,N}, N$ la testina trova una casella vuota e si sposta ancora a nord
11. $q_{1,N,N}, b, b, q_{1,S}, S$ la testina trova ancora una casella vuota, e inverte la direzione
12. $q_{1,S}, b, b, q_{1,S,S}, S$ la testina ritrova la casella vuota e si sposta in basso
13. $q_{1,S,S}, b, 1, q_f, S$ la testina scrive il riporto si porta nello stato finale e si ferma dopo lo spostamento a sud

Esempio: Si vuole scrivere un programma che data una stringa di zeri ed uni, si voglia cancellere le prime due occorrenze del carattere 1.

Pensiamo ad una macchina bidimensionale come ad un nastro composto di tante caselle e che in ogni casella, ci sia un carattere oppure un balnk.



1. $q_0, 0, 0, q_0, D$ la testina legge uno zero, scrive uno zero, e si sposta a destra
2. $q_0, 0, 0, q_0, D$ la testina legge uno zero, scrive uno zero, e si sposta a destra
3. $q_0, 1, 0, q_1, D$ la testina legge uno uno, scrive uno zero, e si sposta a destra
4. $q_1, 0, 0, q_1, D$ la testina legge uno zero, scrive uno zero, e si sposta a destra
5. $q_1, 1, 0, q_f, D$ la testina legge uno zero, scrive uno zero, e si sposta a destra

Il programma dovrà prevedere una qualche strategia se non riesece a trovare i due caratteri da sostituire; per esempio fare in modo che la testina si fermi dopo un certo numero di zeri trovati, oppure che si fermi al primo blank.

6 Paradigma funzionale alpha (PFA)

Un paradigma di programmazione è uno stile fondamentale di programmazione, ovvero un insieme di strumenti concettuali forniti da un linguaggio di programmazione per la stesura di programmi, e definisce/determina il modo in cui il programmatore concepisce e percepisce il programma.

Un paradigma funzionale è un linguaggio di programmazione che vede la memoria come un "blocco unico": ogni operazione trasforma lo stato (il contenuto) della memoria. La memoria può essere vista come un insieme finito di numeri naturali. Le istruzioni sono quindi delle funzioni che operano trasformazioni sulla memoria.

Il linguaggio che descriveremo è un linguaggio funzionale che opera con semplici costrutti sui numeri naturali.

6.1 Costruttore *0* (zero)

Il costruttore *0* (zero) è la funzione base. La funzione *0* (zero) ha la seguente definizione:

$0 : \mathbb{N}^0 \to \mathbb{N}$ con $\mathbb{N}^0 = \{\lambda\}$ ovvero il simbolo nullo.

La funzione *0* (zero) prende quindi in input una sequenza lunga 0 elementi e restituisce una sequenza lunga 1 elemento, rappresentato dal numero naturale 0.

6.2 Successore *S*

L'operatore Successore S ha la seguente definizione:

$S : \mathbb{N} \to \mathbb{N}$ con $S(x) = x + 1$

L'operatore S prende quindi una sequenza lunga un elemento x e restituisce una sequenza lunga ancora un elemento ma di valore pari al successore di x.

6.3 Proiezione $P_{i,j}^n$

L'operatore Proiezione $P_{i,j}^n$ ha la seguente definizione:

$P_{i,j}^n : \mathbb{N}^n \to \mathbb{N}^{(j-i+1)}$ con $0 \leq i \leq j \leq n$

L'operatore $P_{i,j}^n$ prende in ingresso una sequenza di lunghezza n e restituisce gli elementi che hanno posizione compresa tra i valori (parametri) *i,j* estremi compresi. Se $i \geq j$ l'operatore Proiezione restituirà la sequenza di lunghezza nulla λ.

Per esempio:$(5, 37, 45, 2) \xrightarrow{P_{2,3}^4} (37, 45)$

oppure $(5, 37, 45, 2) \xrightarrow{P_{1,1}^4} (5)$

oppure $(5, 37, 45, 2) \xrightarrow{P_{4,3}^4} (\lambda)$

6.4 Concatenazione " ; "

Il connettore Concatenazione **";"** consente di applicare al risultato di una funzione un'altra funzione. Equivale al costrutto algebrico 'composizione di funzioni'. Il connettore Concatenazione **";"** ha la seguente definizione:

date $f : \mathbb{N}^n \to \mathbb{N}^k$ e $g : \mathbb{N}^k \to \mathbb{N}^r$

allora: $f; g : \mathbb{N}^n \to \mathbb{N}^r$ con $(f; g)(x_1, x_2, \ldots, x_n) = g(f(x_1, x_2, \ldots, x_n))$

Per esempio per costruire il numero 1, applichiamo la funzione successore alla funzione 0 ottenendo quindi il valore 1:

$1 \equiv 0; S$.

In pratica: $\mapsto 0 \xrightarrow{S} 1$.

Per costruire il numero due dovremo concatenare alla funzione 1 appena definita, con un'altra funzione successore, quindi: $2 \equiv 0; S; S \equiv 1; S$.

6.5 Parallelo " || "

Il connettore parallelo consente di avere due funzioni che lavorano su due memorie diverse e di unificarne il risultato in un'unica memoria che è grande come la somma dei codomini delle due funzioni date. Il connettore Parallelo "||" ha la seguente definizione:

date $f : \mathbb{N}^n \to \mathbb{N}^k$ e $g : \mathbb{N}^m \to \mathbb{N}^r$

allora$(f \,||\, g)\,(x_1, x_2, \ldots, x_{n,} y_1, y_2, \ldots, y_m) = f\,(x_1, x_2, \ldots, x_{n,})\,, g\,(y_1, y_2, \ldots y_m)$

Per esempio:

$(0; S; S)\,||\,(0; S; S; S) = 2, 3$.

6.6 Giustapposizione " ^ "

Il connettore Giustapposizione consente che due funzioni operino successivamente sulla stessa area di memoria, accodando i risultati per ottenere una unica memoria grande come la somma dei codomini delle due funzioni date. Il connettore Giustapposizione "^" ha la seguente definizione:

date $f : \mathbb{N}^n \to \mathbb{N}^k$ e $g : \mathbb{N}^n \to \mathbb{N}^r$

allora $(f \wedge g)\,(x_1, x_2 \ldots x_n) = f(x_1, x_2 \ldots x_n), g(x_1, x_2 \ldots x_n)$

Per esempio:

$S||S : \mathbb{N}^2 \to \mathbb{N}^2$ quindi: $3, 5 \xrightarrow{S||S} 4, 6$

mentre:

$S \wedge S : \mathbb{N} \to \mathbb{N}^2$ quindi: $3 \xrightarrow{S \wedge S} 4, 4$

6.7 Sequenziazione *exp*

L'operatore Sequenziazione è l'equivalente del costrutto for dei linguaggi imperativi.

L'operatore Sequenziazione *exp* ha la seguente definizione:

data $f : \mathbb{N}^n \to \mathbb{N}^n$

$exp(f) : \mathbb{N}^{n+1} \to \mathbb{N}^n$ dove $exp\,(f)\,(x_1, x_2, \ldots x_n, y) = f^y\,(x_1, x_2, \ldots x_n)$

definendo in maniera ricorsiva : $f^0 = x_1, x_2, \ldots x_n$, ... $f^{y+1} = f\,(f^y\,(x_1, x_2, \ldots x_n))$

La variabile y rappresenta il numero di volte che deve essere ripetuta l'applicazione della funzione *f*.

6.8 Alcune funzioni

Funzione somma: $+ : \mathbb{N}^2 \to \mathbb{N} \quad + \equiv exp\,(S)$

cioè: $x, y \xrightarrow{exp(S)} S^y(x)$ ovvero viene replicato y volte l'operatore Successore S sulla variabile x.

Funzione prodotto: $* : \mathbb{N}^2 \to \mathbb{N} \quad * \equiv 0||P^2_{1,2}; exp\left(+ \wedge P^2_{2,2}\right); P^2_{1,1}$

cioè: $x, y \xrightarrow{0; P^2_{1,2}} 0, x, y \xrightarrow{exp\left(+ \wedge P^2_{2,2}\right)} x * y, y \xrightarrow{P^2_{1,2}} x * y$ ovvero viene aggiunto uno 0 che serve come accumulatore delle somme parziali, si applica

quindi la funzione somma esattamente y volte partendo dalla coppia di valori *0,x*, infine si restituisce il valore x^*y.

Funzione duplicazione: $D : \mathbb{N} \to \mathbb{N}^2 \quad D \equiv \left(0||0||P^1_{1,1}\right); exp(S||S)$

cioè: $x \xrightarrow{0||0||P^1_{1,1}} 0,0,x \xrightarrow{exp(S||S)} x,x$ ovvero vengono aggiunti due zeri, poi applicata la sequenziazione x volte sul successore in parallelo con un'altro successore.

Tramite la duplicazione si può simulare la giustapposizione, infatti:

$x \xrightarrow{f \hat{} g} f(x), g(x)$ allora: $x \xrightarrow{D} x,x \xrightarrow{f \,||\, g} f(x), g(x)$

Si può quindi scrivere:

$f \hat{} g \equiv D; f \,||\, g$

Funzione esponenziale: $Pot : \mathbb{N}^2 \to \mathbb{N} \quad Pot \equiv \left(1||P^2_{1,2}\right); exp\left(* \wedge P^2_{2,2}\right); P^2_{1,1}$

cioè: $x,y \xrightarrow{1||P^2_{1,2}} 1,x,y \xrightarrow{exp\left(* \wedge P^2_{2,2}\right)} x^y,x \xrightarrow{P^2_{1,1}} x^y$ ovvero si aggiunge un uno, si applica la sequenziazione del prodotto esattamente y volte partendo dalla coppia *1,x*, infine si restituisce il valore x^y.

Funzione decremento di 1: $(-1) : \mathbb{N} \to \mathbb{N} \quad (-1) \equiv \left(0||0||P^2_{2,2}\right); exp\left(P^2_{2,2} \wedge \left(P^2_{2,2}; S\right)\right); P^2_{1,1}$

cioè: $x \xrightarrow{0||0||P^1_{1,1}} 0,0,x \xrightarrow{P^2_{2,2} \wedge \left(P^2_{2,2}; S\right)} (x-1), x \xrightarrow{P^2_{1,1}} (x-1)$

Funzione sottrazione: $sub : \mathbb{N}^2 \to \mathbb{N} \quad sub \equiv \begin{cases} x - y & se\ x > y\ \ exp((-1)) \\ 0 & altrimenti \end{cases}$

esempio:

$3,2 \xrightarrow{(-1)} 2 \xrightarrow{(-1)} 1$

Funzione $\overline{Sg} : \mathbb{N} \to \mathbb{N}$ Definizione:

$\overline{Sg}(x) = \begin{cases} 1 & se\ x = 0 \\ 0 & se\ x \neq 0 \end{cases}$

$x \xrightarrow{1\,||\,P^1_{1,1}} 1,x \xrightarrow{sub} 1 - x$

quindi:

$\overline{Sg} \equiv \left(1\,||\,P^1_{1,1}\right); (sub)$

Funzione $Sg : \mathbb{N} \to \mathbb{N}$ Definizione:

$Sg(x) = \begin{cases} 0 & se\ x = 0 \\ 1 & se\ x \neq 0 \end{cases}$

$x \xrightarrow{1\,||\,P^1_{1,1}} 1,x \xrightarrow{sub} 1 - x \quad \overline{Sg} \quad \begin{cases} 0 & se\, x = 0 \\ 1 & se\, x \neq 1 \end{cases}$

quindi:

$Sg \equiv \left(\overline{Sg}\right); \left(\overline{Sg}\right)$

Funzione Uguaglianza: $(x = y) \equiv \begin{cases} 1 & se\ x = y \\ 0 & se\ x \neq y \end{cases}$

Nel campo dei naturali la definizione di uguaglianza appena data può venir sostituita dalla seguente:

Confronto:

$$(x - y) + (y - x) = \begin{cases} 0 & sse\ x = y \\ x - y & sse\ x > y \\ y - x & sse\ y > x \end{cases}$$

Allora:

$x, y \xrightarrow{P^2_{1,2} \wedge P^2_{2,2} \wedge P^2_{1,1}} x, y, y, x \xrightarrow{sub\,||\,sub} (x - y), (y - x) \xrightarrow{+} (x - y) + (y - x)$

quindi, utilizzando $\overline{Sg}$:

Uguaglianza$\equiv \left(P^2_{1,2} \wedge P^2_{2,2} \wedge P^2_{1,1}\right); (sub\,||\,sub); (+); \left(\overline{Sg}\right)$

7 Decidibilità

Un insieme X si dice decidibile se esiste un algoritmo che, dato un valore x in input, definisce se x appartiene o meno all'insieme

$X \subseteq \mathbb{N}$, $f(x) \begin{cases} 1 & se\, x \in X \\ 0 & se\, x \notin X \end{cases}$, $f(x)$ è calcolabile totale

8 Semidecidibilità

Un insieme X si dice semidecidibile se, preso un algoritmo e dato un valore x in input definisce se x appartiene all'insieme, ma non il contrario.

$X \subseteq \mathbb{N}$, $g(x) \begin{cases} 1 & se\, x \in X \\ \uparrow & se\, x \notin X \end{cases}$, $g(x)$ è calcolabile.

9 Indecidibilità

Un insieme X è indecidibile se non è decidibile.

$k = \{x : P(x) \downarrow x\}$ NON è decidibile

$$f(x) = \begin{cases} 1 & se\, P_x \downarrow x,\ x \in K \\ 0 & se\, P_x \uparrow x,\ x \notin K \end{cases}$$

10 Teorema della decidibilità

Un insieme $A \subseteq \mathbb{N}$ è decidibile se e solo se A e $\overline{A}$ sono semidecidibili.

Dimostrazione $\rightarrow$

Sia A decidibile, prendo l'algoritmo che decide A e lo modifico in modo da semidecidere A (while(1) se $x \notin A$). Si procede analogamente per $\overline{A}$.

Dimostrazione $\leftarrow$

Siano A e $\overline{A}$ semidecidibili. Faccio lavorare in parallelo i due programmi, uno dei due termina e mi dice se $x \in A$ o se $x \notin A$.

$f(x)\begin{cases}1 & se\, x \in A \\ \uparrow & se\, x \notin A\end{cases}$, $g(x)\begin{cases}1 & se\, x \in \overline{A} \\ \uparrow & se\, x \notin \overline{A}\end{cases}$

11 Note

1. Dal momento che K non è decidibile, ma è semidecidibile, allora $\overline{K}$ non è semidecidibile
2. Se A è decidibile, allora anche $\overline{A}$ è decidibile

12 Insiemi ricorsivamente enumerabili

Un insieme $X \subseteq \mathbb{N}$ è r. e. se esiste $f : \mathbb{N} \to \mathbb{N}$ calcolabile totale (termina su tutti gli input) tale che $Cod(f) = X$.

NOTA: X è strettamente crescente (insieme dei numeri pari).

13 Note

1. Se l'insieme X è semidecidibile, allora X è r. e.
2. Se l'insieme X infinito è decidibile, $\exists\, f$ calcolabile totale strettamente crescente tale che $X = Cod(f)$

14 Teorema del parametro

Sia $f : \mathbb{N}^2 \to \mathbb{N}$ calcolabile, allora esiste una funzione calcolabile totale $S : \mathbb{N} \to \mathbb{N}$ tale che $\Phi_{S(x)}(y) = f(x, y)$

Definizioni:

- $n \to$ input numero
- $P_n \to$ programma n-esimo
- $\phi_n \to$ funzione calcolata da P_n

15 Insiemi riducibili

Siano definiti gli insiemi $A, B \subseteq \mathbb{N}$, $A \leqslant_T B$ (A è riducibile a B) se esiste una funzione calcolabile totale S: $\mathbb{N} \to \mathbb{N}$ tale che:

$x \in A \Rightarrow S(x) \in B$
$x \in \overline{A} \Rightarrow S(x) \in \overline{B}$

16 Teorema della decidibilità 2

Se $A \leqslant_T B$, allora:

1. B decidibile $\Rightarrow$ A decidibile
2. B semidecidibile $\Rightarrow$ A semidecidibile
3. A non è decidibile $\Rightarrow$ B non è decidibile
4. A non è semidecidibile $\Rightarrow$ B non è semidecidibile

NOTA:

- Le proprietà positive sulla riducibilità: $dx \to sx$
- Le proprietà negative sulla riducibilità: $sx \to dx$

17 Insiemi che rispettano le funzioni

Dato l'insieme $I \subseteq \mathbb{N}$, si dice che I rispetta le funzioni se $\forall n, k : n \in I$, $\phi_n = \phi_k \Rightarrow k \in I$

$I_1 = \{n \mid P_n \, calcola \, l'identità\}$ rispetta le funzioni
$I_2 = \{2, 5, 7\}$ NON rispetta le funzioni
$I_3 = \{x : \phi_x \downarrow y \, per \, almeno \, un \, y\}$ rispetta le funzioni
Nel caso di I_2 , se aggiungo a 2 delle istruzioni che non fanno niente, avrò lo stesso risultato ma 2 non farà più parte dell'insieme.

18 Teorema degli insiemi ricorsivamente enumerabili

Teorema

Un insieme non vuoto $A \subseteq \mathbb{N}$ si dice ricorsivamente enumerabile se e solo se A è semidecidibile.

Dimostrazione $[\to]$**:**

A è r. e., cioè $\exists f : \mathbb{N} \to \mathbb{N}$ calcolabile totale tale che Cod(f) = A.

Prendiamo f(0): se f(0) = x, allora $x \in A$, altrimenti calcolo f(1) e così via.

Se $x \notin A$, ciclo all'infinito; in questo modo ho semideciso A.

$$f(n) \begin{cases} 1 & se \, f(n) = x \\ \uparrow & altrimenti \end{cases} \quad (n \geqslant 0)$$

Dimostrazione $[\leftarrow]$**:**

Prendo un algoritmo P che semidecide A

- P(0) per 10 minuti termina $0 \in A \Rightarrow f(0) = 0$
- P(1) per 10 minuti non termina (scrivo 0)
- P(0) per 20 minuti termina $0 \in A \Rightarrow f(1) = 0$
- P(0) per 30 minuti termina $0 \in A \Rightarrow f(2) = 0$
- P(1) per 20 minuti termina $1 \in A \Rightarrow f(3) = 1$

Dom(f) $\rightarrow$ insieme ordinato delle volte in cui P termina

Cod(f) $\rightarrow$ input in cui P è terminato

19 Teorema decidibilità a infinito

Teorema: Un insieme A infinito è decidibile se e solo se A può essere enumerato in ordine strettamente crescente.

Dimostrazione $[\rightarrow]$**:**

A è decidibile quindi calcolo:

$P(0) \rightarrow 0 \in A$, allora f(0) = 0

$P(1) \rightarrow 1 \notin A$, salto

$P(2) \rightarrow 2 \notin A$, allora f(1) = 2

Proseguendo come sopra, ottengo f calcolabile totale strettamente crescente.

Dimostrazione $[\leftarrow]$**:**

Se posso enumerare A, allora avrò una sequenza $f(0) < f(1) < \ldots < f(n)$. Se x = f(n), allora posso dire che $x \in A$, altrimenti $f(n-1) < x < f(n)$, da concludo che $x \notin A$. In questo modo A è decidibile.

20 Teorema di Rice 1

Teorema: Sia $I \subseteq \mathbb{N}$ un insieme che rispetta le funzioni. Allora:

1. I è decidibile se e solo se $I = \emptyset$ o$I = \mathbb{N}$
2. se $\{n : \Phi_n = f_0\} \subseteq I \neq \mathbb{N}$, allora I non è semidecidibile
3. se $\{n : \Phi_n = f_0\} \subseteq \overline{I} \neq \mathbb{N}$, allora $\overline{I}$ non è semidecidibile

Dimostrazione $[1 \rightarrow]$**:**

Banale: se $I = \emptyset$ o $I = \mathbb{N}$, è facile trovare un programma che decida I.

Dimostrazione $[1 \leftarrow]$**:**

Supponiamo $I \neq \emptyset$ e $I \neq \mathbb{N}$. Prendo $z \in \overline{I}$ tale che $\phi_z \neq f_0$

$$f(n)\begin{cases}\Phi_z(y) & se\, x \in K\\ \uparrow & altrimenti\end{cases}$$

f(x,y) è calcolabile. Allora posso applicare il teorema del parametro.

$$\phi_{S(x)}(y) = f(x,y) = \begin{cases}\Phi_z(y) & se\, x \in K\\ \uparrow & altrimenti\end{cases}$$

S(x) è calcolabile totale.

Se $x \in K \Rightarrow \phi_{S(x)}(y) = \phi_z(y)\ \forall y \Rightarrow S(x) \in \overline{I}$

Se $x \in K \Rightarrow \phi_{S(x)}(y) = \uparrow\ \forall y \Rightarrow S(x) \in I$

In questo modo abbiamo ridotto $\overline{I}$ a K, cioè $k \leqslant_T I$.

K non è decidibile $\Rightarrow$ $\overline{I}$ non è decidibile $\Rightarrow$ I non è decidibile.

Dimostrazione $[2 \rightarrow]$**:**

Ho ridotto $\overline{I}$ a K, ma posso fare anche $\overline{K} \leqslant_T I$. Allora $\overline{K}$ non è semidecidibile $\Rightarrow I$ non è semidecidibile.

21 Teorema di Rice 2

Teorema: Sia $I \subseteq \mathbb{N}$ un insieme che rispetta le funzioni e supponiamo che esistano due funzioni calcolabili f e g tali che:

1. $\{x : \phi_x = f\} \subseteq I$
2. $\{x : \phi_x = g\} \subseteq \overline{I}$
3. $f \leqslant g$

Allora I non è semidecidibile.

Dimostrazione:

Supponiamo di avere un insieme I e due funzioni f e g; vogliamo ridurre $\overline{K}$ a I $\left(\overline{K} \leqslant_T I\right)$. Considero:

$$h(x,y)\begin{cases}g(y) & se\, x \in K\\ f(y) & se\, x \notin K\end{cases}$$

Eseguo in parallelo A, un algoritmo che semidecide x, e B, un algoritmo che calcola f(y).

Possono verificarsi i seguenti casi:

1. Termina A $\Rightarrow$ $x \in K$, calcolo g(x) e blocco B
2. Termina B $\Rightarrow$ (per Rice 2) $f < g \Rightarrow$ ho f(y) = g(y)
3. Non termina né A né b $\Rightarrow$ $h(x,y)$ è indefinita

h è calcolabile, allora applico il teorema del parametro

$$\phi_{S(x)}(y) = h(x,y) \begin{cases} g(y) & se\, x \in K \\ f(y) & se\, x \notin K \end{cases}$$

Se $x \in K \Rightarrow \phi_{S(x)}(y) = g(y) \Rightarrow S(x) \in \overline{I}$

Se $x \in K \Rightarrow \phi_{S(x)}(y) = f(y) \Rightarrow S(x) \in I$

In questo modo ho ridotto $\overline{K}$ a I ($\overline{K} \leqslant_T I$).

$\overline{K}$ non è semidecidibile $\Rightarrow$ I non è semidecidibile.

22 Teorema di Rice 3

Teorema: Sia $I \subseteq \mathbb{N}$ un insieme che rispetta le funzioni e sia f, funzione calcolabile con dominio infinito tale che:

1. $\{x : \phi_x = f\} \subseteq I$
2. $\{x : Dom(\phi_x)\, è\, finito\, e\, f \leqslant \phi_x\}\, è \subseteq \overline{I}$

Alora I non è semidecidibile

Dimostrazione:

Il nostro obiettivo è dimostrare che $\overline{K}$ è riducibile a I.

$$h(x,y) \begin{cases} f(y) & se\, P_x \uparrow x\, in\, n\, passi \leqslant y \\ \uparrow & altrimenti \end{cases}$$

h(x,y) è è calcolabile perché basta far partire il programma P_x con input x e controllare cosa succede dopo y passi. Se non ha ancora terminato, eseguo f con input y altrimenti ciclo.

$$\phi_{S(x)}(y) = h(x,y) = \begin{cases} f(y) & se\, P_x \uparrow x\, in\, n\, passi \leqslant y \\ \uparrow & altrimenti \end{cases}$$

Considero la seguente funzione come restrizione finita di f:

$$\phi_{S(x)}(y) = \begin{cases} f(y) & se\, y \leqslant C_0\, passi \\ \uparrow & se\, y > C_0\, passi \end{cases}$$

$x \in K \Rightarrow P_x \downarrow x$ in C_0 passi $\Rightarrow S(x) \in \overline{I}$

$x \notin K \Rightarrow P_x \uparrow x$ sempre $\phi_{S(x)}(y) = f(y) \Rightarrow S(x) \in I$

Ho ridotto $\overline{K}$ a I $\left(\overline{K} \leqslant_T I\right)$ quindi:

$\overline{K}$ non è semidecidibile $\Rightarrow I$ non è semidecidibile.

23 Secondo teorema della ricorsione

Teorema: Data una qualsiasi funzione calcolabile h, allora esiste un certo n per cui $\phi_n = \phi_{h(n)}$ (h è totale).

Dimostrazione:

Considero h funzione calcolabile totale e la seguente funzione f(x,y) in due variabili:

$f(x,y) = \phi_{h(\phi_x(x))}(y)$

Considero m un programma che calcola $S(x) \Rightarrow S(m) = \phi_m(m)$ e considero n la quantità $\phi_m(m)$.

$\phi_{\phi_m(m)}(y) = f(x,y) = \phi_{h(\phi_m(m))}(y) \Rightarrow \phi_n(y) = \phi_{h(n)}(y)$

24 Primo teorema della ricorsione

Teorema: Sia h calcolabile totale estensionale, allora esiste una funzione calcolabile $fix_\tau : \mathbb{N} \to \mathbb{N}$ tale che:

1. $\tau_h(fix_\tau) = fix_\tau$
2. Se $\tau_h(y) = g \Longrightarrow fix_\tau \leqslant g$
3. fix_τ è calcolabile

25 Funzioni estensionali

Sia $h : \mathbb{N} \to \mathbb{N}$ funzione calcolabile totale. h si dice estensionale se $\Phi_x = \Phi_y \Longrightarrow \Phi_{h(x)} = \Phi_{h(y)}$.

Lemma:

Sia S calcolabile totale ed estensionale; allora $\Phi_{S(y)}(x) = z$ se e solo se $\exists \phi_n$ funzione finita, dove $\Phi_n \leqslant \Phi_y$ tale che $\Phi_{S(n)}(x) = z$