Introduction to computability

Thursday, March 11, 2021 11:16 AM

Diophantine equation:

$\rightarrow x^2 + y^2 = z^2$

- give solutions in whole numbers
- give solutions in rational numbers

$x^2 + y^2 = 1$

$3^2 + 4^2 = 5^2$

$5^2 + 12^2 = 13^2$

$x^3 + y^3 = z^3$

$x = 1, y = 0, z = 1$ × trivial

$x^y - y^x = 1$

$3^2 - 2^3 = 1 \rightarrow$ only solution

MECHANICAL PROCEDURE??

The concept of algorithm was not yet formulated.

TODAY:

An algorithm is a process for computing something using finite resources, with a precise notion of primitive step; each

step involves a finite amount of information. SOMETIMES we require that the process <u>halts</u> (terminate).

1928 Hilbert & Ackerman asked for a mechanical procedure to determine if a formula of logic is <u>valid</u> or not.

VALID: True no matter how you interpret the variables.

$$\text{VALID}\left(\left(\forall x.\ P(x) \Rightarrow Q(x)\right) \wedge P(a)\right) \Rightarrow Q(a)$$

ENTSCHIEDUNGSPROBLEME!

Both these problems are <u>impossible</u>!

(USA) ALONZO CHURCH 1935

(UK) ALAN TURING 1936

Need a notion of algorithm:

KURT GÖDEL → TOTAL RECURSIVE FUNCTIONS

ALONZO CHURCH → λ-CALCULUS

ALAN TURING → TURING M/C

A. MARKOV, E. POST, S. KLEENE, B. ROSSER ....

WE HAVE MANY EQUIVALENT SYSTEMS:

(1) Partial recursive f^{ns} (KLEENE)

(2) FLOWCHARTS

(3) "While" programs

(4) ASSEMBLY LANGUAGE
(5) RAM MACHINES
(6) 2 STACK AUTOMATA
(7) 2 COUNTER M/C
(8) QUEUE M/C
(9) 1000001 PROGRAMMING LANGUAGE
(10) TURING MACHINES
(11) λ- CALCULUS
(12) PHRASE STRUCTURE GRAMMAR

CHURCH - TURING THESIS

---

(1) Algorithms can be expressed as a sequence of symbols and stored as data.
(2) There is a universal machine.
(3) There are unsolvable problems.

The HALTING PROBLEM:

ALONZO CHURCH - 1935

BASIC ASSUMPTIONS:

Every algorithm can be coded as a program and represented as a string.

$P \rightarrow$ program $\quad \# P \rightarrow$ source code

$P \rightarrow$ program , $x \rightarrow$ input

$P(x) \rightarrow$ run P with x as input

$P(x, y) \rightarrow \ldots \ldots x, y, \ldots$

PL has while loops, if_then_else and ;

What we want : $H(\cdot, \cdot)$

$H(\#P, x)$ → TRUE if $P(x)$ halts
→ FALSE if $P(x)$ loops

→ <u>always</u> halts.

CLAIM : SUCH AN H CANNOT EXIST!

Assume we have such an H.

Define a new program S:

S(#P) : if H(#P, #P) then loop
else halt

S has source code #S

S(#S) : Does this halt?

if H(#S, #S) then loop else halt.

so if <u>S(#S) halts</u>, H returns true,
we go to the <u>then</u> branch & <u>loop</u> ⊗

if S(#S) loops, H returns false
we go to the <u>else</u> branch & <u>halt</u> ⊗

such an H cannot exist!

EXAMPLE

COL(n) = if n=1 then return 1

else if n is even COL(n/2)

else COL(3n+1)

COL → COLLATZ

COL(11) → COL(34) → COL(17) → (52)

→ 26 → 13 → 40 → 20 → 10 → 5 → 16 →

8 → 4 → 2 → 1 HALT

For n > 1 does this always halt?

AS YET UNKNOWN!

RECAP Diagonalization

The set of <u>infinite</u> sequences of positive integers is not <u>countable</u>

1	6	3	39	4	29	13	...
3	7	1	9	...	...	...	
2	8	4	4	...	...	...	
⋮							

→ 5 3 2 . . . .

The new list differs in its nth place from the nth element of the nth list.

Hence it cannot be on the list!

The number of pairs of positive integers is countable.

(1,1)	(1,2)	(1,3)	(1,4)	. . . .
(2,1)	(2,2)	(2,3)	(2,4)	. . . .
(3,1)	(3,2)	(3,3)	(3,4)	. . . . .
(4,1)	(4,2)	(4,3)	(4,4)	. . . .

(1,1) (2,1) (1,2) (3,1), (2,2), (1,3) . . .