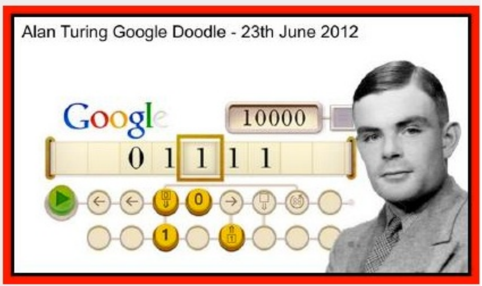
Turing Machines VII

Friday, October 31



Handouts:

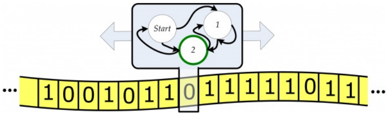
Assignment 20

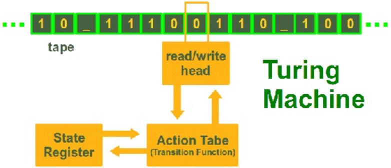
ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

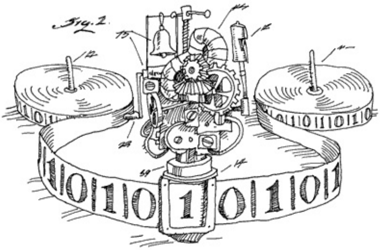
By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.


Start
1
2
1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 1


1 0 _ 1 1 1 0 0 1 1 0 _ 1 0 0
tape
read/write head
Turing Machine
State Register
Action Tabe
(Transition Function)



On some input, a TM may run indefinitely without ever accepting or rejecting.

For example, it may **loop** on that input. Such a machine will never **halt**.

Simple Examples

1. A TM to *recognize* 2 (a *decider*)
2. A TM to *compute* $f(x) = x + 1$ for an integer $x \geq 1$ (the *increment* or *plus one* function).
3. A TM that does not halt.
4. A TM to 'compute' $f(x) = 2x$ for an integer $x \geq 1$ (the *doubling function*)
5. A TM that adds two non-negative integers:

   $f(x, y) = x + y$ for two non-negative integers x and y.

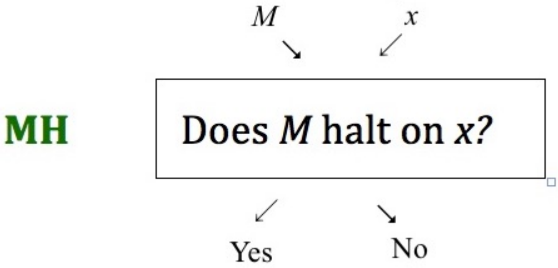
Note: Leavitt discusses a TM to add two integers (page 68) but since he never gives a careful, precise definition for a TM, it is difficult to follow some of what he says. Does his TM only work for $2 + 2$?.

Two More Examples

- A TM that accepts strings over the alphabet $\{a, b\}$ that contain an arbitrary number of a's followed by an arbitrary number of b's. So the strings $aabb, aaaabbbbbb, bb, aaab$ will all be accepted while $bbaa$ or $abba$ will be rejected. This TM is a *decider*.
- (Hard) A TM to carry out subtraction; that is, to compute $f(x, y) = x - y$ where $x \geq y \geq 0$

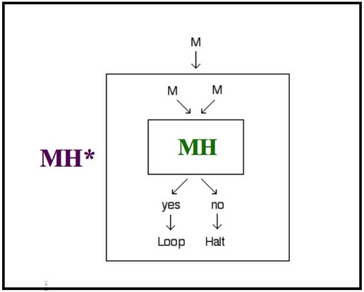
An Alternate Proof for The Halting Problem (HP)

Suppose there exists a **Turing Machine MH** that decides Halting Problem; that is, if we feed $\langle M, x\rangle$ to **MH** it will print 'yes' if M halts on input x and 'no' if M does not halt on x (for any Turing Machine M and input string x).



Given the existence of **MH** we can create a new machine **MH*** that on input M, run **MH** on $\langle M, M\rangle$; if **MH** prints "yes" Loop else (if **MH** prints "no") Halt.

Given the existence of **MH** we can create a new machine **MH***
that on input M, run **MH** on $\langle M, M \rangle$; if **MH** prints "yes" Loop
else (if **MH**prints "no") Halt.



So **MH*** on M halts if and only if M on M does not halt.
Now, what about **MH*** on **MH*** ? (a self reference)
MH* on **MH*** halts if and only **MH*** on **MH*** does not halt.
So, **MH** cannot exist (assuming so leads to a contradiction).

Turing used Self-Reference to attack the "Entscheidungsproblem"

We can also use the notion of "self-reference" to prove the **Halting Problem**. (Turing's Entscheidungs or "decision" problem).

Does an algorithm exist to "decide" the following problem:
Given a TM M and an input string x, does M eventually halt when it is run with x on its tape?

Given an algorithm H() and its input x, does H() halt when run on x?
(does H(x) halt?)
We will show the Halting Problem is *undecideable* in the sense that no Turing machine can ever answer "yes" or "no" to the above question(s).

What is wrong with the following attempt at an algorithm? "Just run M on x and see."
Answer: M may never answer "yes" or "no" since it may never halt!
Alternate form: Can we write a special program that will take any other program that someone has written as well as the data it will run on and decide whether

- it will eventually stop (halt) or
- it will run indefinitely?

We will see that Turing "stood on the shoulders of giants" in the sense that he was influenced by mathematicians such as Cantor (the idea of a "diagonalization" argument) and Gödel (the incompleteness theorems as well as the idea of encoding something like "This sentence is false" into a number).

Question: How many TMs are there?

It is possible to encode all of the information needed to describe a TM as a finite string. So, there are no more TMs than there are strings over a finite alphabet.

Given a finite alphabet such as $\Sigma = a, b$, how many strings are there? There are an infinite number of course but the degree of infinity is "countable" since we can list or "itemize" the strings by size:
$\Sigma = \epsilon, a, b, aa, ab, ba, bb, aaa, aab, ?$
So the total number of TMs is at most countably infinite (similar to the set of integers).

The Halting Problem (HP): "Given a TM M and an input string x, does M halt when it is run with x on its tape?"

Theorem: The Halting Problem, HP, is undecidable.

Proof: (by contradiction, using "diagonalization") -assume there is a rule for deciding if a given TM will halt.

Construct a table as follows:

List all Turing machines down the side List the possible inputs across the top In position (i, j) put the result of executing Turing machine i on input j.
If it halts, output H.
If it doesn't halt, output L (for loops)

	1	2	3	4	5	. . .
T1	**H**	H	L	H	H	. . .
T2	L	**L**	H	H	H	. . .
T3	H	H	**H**	H	H	. . .
T4	H	H	H	**L**	H	. . .
T5	H	H	H	H	**H**	. . .
. . .						

	1	2	3	4	5	. . .
T1	**H**	H	L	H	H	. . .
T2	L	**L**	H	H	H	. . .
T3	H	H	**H**	H	H	. . .
T4	H	H	H	**L**	H	. . .
T5	H	H	H	H	**H**	. . .
. . .						

Now define a new TM called D that will halt for all inputs.
It outputs H if Ti(i) does not halt and L if Ti(i) does halt
D:L H L H L ?
Now, our assumption is that the above table lists all possible TMs.
It can't be T1 since D differs from the operation of T1 in the first column,
it can't be T2 since D differs from the operation of T2 in column 2, and so forth.

An alternate way to reach this same contradiction is as follows:

We already said that the assumption is that we can always decide if a given TM halts. Also, we said this table lists all possible TMs. This is a contradiction. **We cannot determine if an arbitrary**

program/TM will halt when run on some input string x.