The Halting Problem of Turing Machines

The Halting Problem

Is there a procedure that takes as input a program and the input to that program, and the procedure determines if that program terminates on that input?

Posed by Alan Turing in 1936, to prove that there are unsolvable problems

---

The Halting Problem

- assume we have procedure H(P,I), where P is a program and I its input
  - H(P,I): if halts(P,I)
    then "halts"
    else "loops"
- Note: a program is a bit string, and may be considered as input
  - hence H can take itself as input P or as input I
  - a call H(H,H) should be allowed
- Construct a new procedure K(P), where input P is a program
  - K(P): if H(P,P) = "loops"
    then "halt"
    else while true do skip; // loops forever
- K(P) does the opposite of H(P,P)
  - if P halts when given itself as input K loops
  - if P loops when given itself as input K halts
- Just as above, a call to K(K) should be allowed
  - this makes the call H(K,K)
    - if H(K,K) = "loops" then K(K) produces "halt"
    - if H(K,K) = "halt" then K(K) loops forever
      - and this violates what H(K,K) tells us
- Thus H cannot exist, as it would be absurd.

A replay of the proof — Part 1

Assume existence of function **halt(p:string,i:string)**
where p is a program file, given as a string
i is the input to p, given as a string

∴ function halt(p:string,i:string) : boolean
-> if program p halts with input i then return true else return false

Now define a new function **trouble(p:string)**

```
function trouble(p:string) : boolean
 -> if halt(p,p)
    then while true do();   // p applied to p halts, so loop forever
    else return true;       // p applied to p loops, so halt and return true
```

If halt(p,p) returns true then trouble loops forever
If halt(p,p) returns false then trouble halts and returns true

---

A replay of the proof — Part 2

```
function trouble(p:string) : boolean
 -> if halt(p,p)
    then while true do();
    else return true;
```

If halt(p,p) returns true then trouble loops forever
If halt(p,p) returns false then trouble halts and returns true

Assume **t** is the string that represents the function **trouble**

Does **trouble(t)** halt?

1. Assume **trouble(t)** halts
   From definition of function **trouble** above **trouble(t)** does not halt
   A contradiction

2. Assume **trouble(t)** loops forever
   From definition of function **trouble** above **trouble(t)** does halt
   A contradiction

A replay of the proof — Part 3

```
function trouble(p:string) : boolean
 -> if halt(p,p)
    then  while true do ();
    else return true;
```

Reality check: what is **trouble(t)**?

trouble(t)
We take function trouble and give it trouble (i.e. t) as a parameter
We call halt(t,t)
- test if function trouble terminates when given as input trouble

Note: arguably this has not been a proof as we have not defined our model of computation.

We have assumed that we all know what a function is, a computer, a program, ...

In 1936 Alan Turing had to invent a "computer" just to give the above proof.

That computer, model of computation, is now called a "Turing Machine"

His reviewers insisted that he show that a TM Was equivalent to Alonzo Church's Lambda Calculus

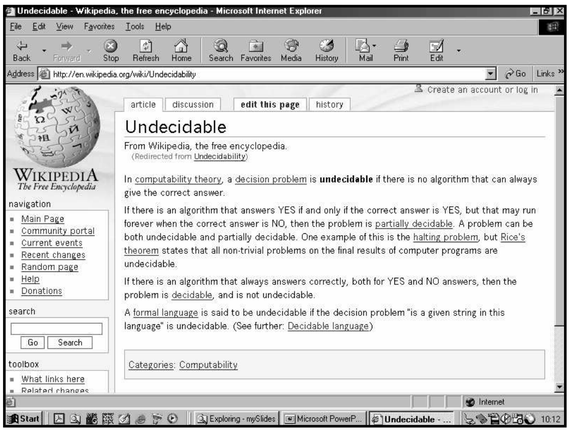
---

Is it weird that a program should take a program as input?

Is it weird that a program can take itself as input?

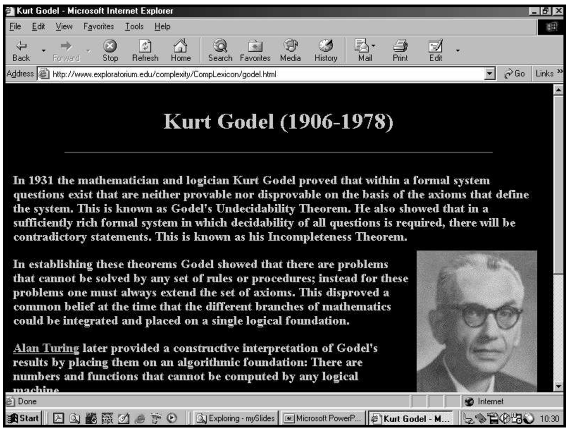
The importance of the halting problem

The first problem proved to be undecidable.



Consequences of the halting problem

The Entscheidungsproblem is unsolvable



Kurt Gödel (1906–1978)

Considered the greatest mathematical logician of the twentieth century, he was one of the founders of recursion theory.

A Princeton colleague of Alonzo Church and John von Neumann, his impact on computer science was seminal, but largely indirect.

Philosophical significance of the incompleteness theorems

Much later, in his Gibbs Lecture to the American Mathematical Society (1951),Gödel would suggest that the incompleteness theorems are relevant to the questions

(1) whether the powers of the human mind exceed those of any machine, and

(2) whether there are mathematical problems that are undecidable for the human mind.

The Gibbs Lecture (1951)

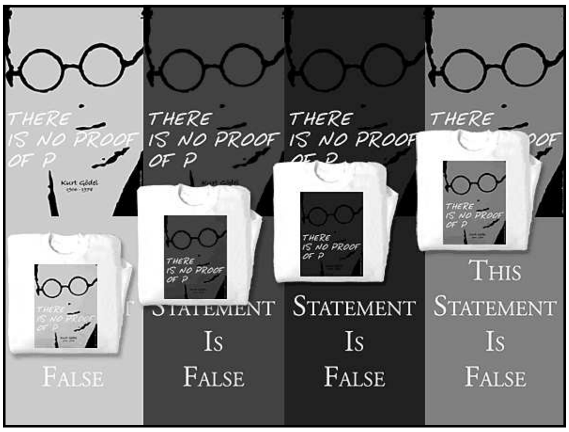
In his Gibbs Lecture, Gödel attempted to draw implications from the incompleteness theorems concerning three problems in the philosophy of mind:

1. Whether there are mathematical ques-tions that are “absolutely unsolvable” by any proof the human mind can conceive
2. Whether the powers of the human mind exceed those of any machine
3. Whether mathematics is our own creation or exists independently of the human mind

Gödel's conclusions

With regard to the first two questions, Gödel argued that "Either … the human mind (even within the realm of pure mathematics) infinitely surpasses the powers of any finite machine, or else there exist absolutely unsolvable diophantine problems". He believed the first alternative was more likely.

As to the ontological status of mathematics, Gödel claimed that the existence of absolutely unsolvable problems would seem "to disprove the view that mathematics is … our own creation; for [a] creator necessarily knows all properties of his creatures". He admitted that "we build machines and still cannot predict their behavior in every detail". But that objection, he said, is "very poor":



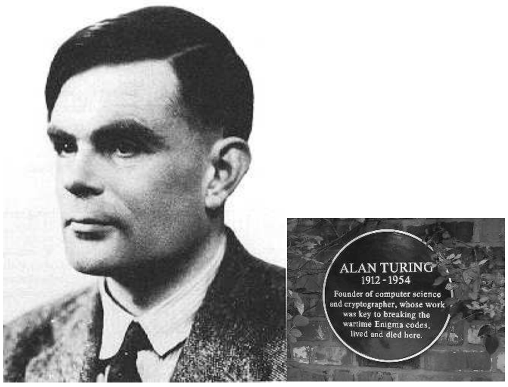
Can Humans solve the halting problem?

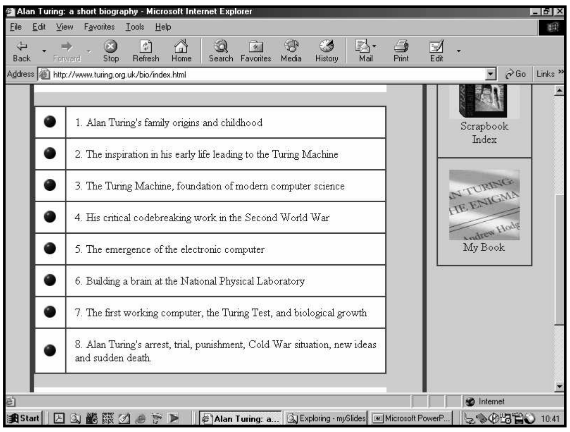
Look at a piece of code and tell me if it halts for a given input

```
[twinPrimes(n:integer) : boolean
 -> let p := n,
        found := false
    in (while not(found)
          (if prime(p) & prime(p+2)
              found := true
           else p := p + 1),
        found)]
```

The above function searches for twin primes greater than n, such that p > n and p is prime as is p+2 (examples, 17 and 19, 41 and 43, 57 and 59)

Will the function halt for all values of n?






DESTROY THE COMPUTER!!!
A HOMOSEXUAL INVENTION BY
ALAN TURING. WHO CARES IF HE CRACKED
THE GERMAN ENIGMA CODE AND WON
WORLD WAR II.
IT'S A HOMO'S DEVIL MACHINE!