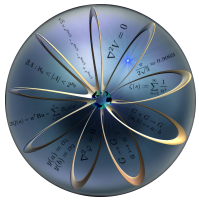

planetmath.org

Math for the people, by the people.

halting problem

Canonical name	HaltingProblem
Date of creation	2013-03-22 12:14:28
Last modified on	2013-03-22 12:14:28
Owner	rspuzio (6075)
Last modified by	rspuzio (6075)
Numerical id	12
Author	rspuzio (6075)
Entry type	Theorem
Classification	msc 03D75
Related topic	RecursivelyEnumerable

The *halting problem* is to determine, given a particular input to a particular computer program, whether the program will terminate after a finite number of steps.

The consequences of a solution to the halting problem are far-reaching. Consider some predicate $P(x)$ regarding natural numbers; suppose we conjecture that $P(x)$ holds for all $x \in \mathbb{N}$. (Goldbach's conjecture, for example, takes this form.) We can write a program that will count up through the natural numbers and terminate upon finding some n such that $P(n)$ is false; if the conjecture holds in general, then our program will never terminate. Then, *without running the program*, we could pass it along to a halting program to prove or disprove the conjecture.

In 1936, Alan Turing proved that the halting problem is undecideable; the argument is presented here informally. Consider a hypothetical program that decides the halting the problem:

Algorithm HALT(P, I)
Input: A computer program P and some input I for P
Output: True if P halts on I and false otherwise

The implementation of the algorithm, as it turns out, is irrelevant. Now consider another program:

Algorithm BREAK(x)
Input: Code x
Output:

```
begin
  if IsValidCode(x) and Halt(x,x) then
        while true do
              nothing
  else
        return true
end
```

If our halting solution determines that Break(Break) halts, then it will immediately enter an infinite loop; otherwise, Break will return immediately. We must conclude that the Halt program does not decide the halting problem. So for any attempted solution to the halting problem, we can find some input which breaks that solution.