

Can you systematically determine if any given program will HALT?

Source: <http://sci.tech-archive.net/Archive/sci.logic/2005-02/1277.html>

From: |-|erc (*h_at_r.c*)

Date: 02/13/05

Date: Sun, 13 Feb 2005 16:30:31 +1000

From: David C. Ullrich (ullrich@math.okstate.edu)

Subject: Re: Daryl, Dave, Barb and the Georges' source of Confusion

Not that I'm going to look it up, but irc his response is not nearly that coherent: He mumbles something about how there's no *_one_* program that can do it, the above is two programs.

And of course he doesn't say anything sensible when we point out that no, *_one_* of those two programs does it.

"Acme Diagnostics" <LFinezaphthis@partpostmark.net> wrote in >
> *Thanks for the probability idea about a real-world solution to*
> *the Halting Problem recently when you were feeling well. Of all*
> *those who talked about that in real-world context, you had the*
> *best idea in my opinion. I think that was a major contribution,*
> *and I am sure I will be able to use that idea someday.*
>
> *Larry*

Thank you Larry, Halt() is a big nut to crack. Here is the outline of the solution for those who missed it among the heated discussion.

halt(f, a) -> pHalt2(n) -> P(UTM(n, f(a))='halt') = 0.5
!halt(f, a) -> pHalt2(n) -> UTM(n, f(a))='don't know'

pHalt2 is an (infinite) set of functions that satisfy pHalt below, with the added constraint P(UTM(n, f(a))='halt') = 0.5. Therefore a randomly selected pHalt function has 50% odds of answering the halt program for some given input.

(different set notation)

halt(f, a) -> En, n e pHalt, UTM(n, f(a))=true
!halt(f, a) -> An, n e pHalt, UTM(n, f(a))='don't know'

For example, pHalt could be an infinite set of Universal Turing Machines,

sci.logic: Can you systematically determine if any given program will HALT?

but they timeout after various amounts of time. If the parameter $f(a)$ halted in that time then it outputs TRUE, otherwise it timed out and outputs DONTKNOW. Since the number of these partial Halt functions is infinite, we should come across one with enough test cycles to determine if any function halts.

By itself pHalt is still useless as it only gives one output value, HALTS. But if each pHalt function has 50% chance of being correct (something more powerful than the UTM), then determining NOTHALT is a simple probabilistic procedure.

e.g.

$\text{pHalt}2() = \{2, 6, 33, 655, \dots\}$

i.e.

$\text{pHalt}(2) = \text{true}$

$\text{pHalt}(6) = \text{true}$

$\text{pHalt}(33) = \text{true}$

..

pHalt2 is an infinite set of godel numbers whos functions can be parsed by a UTM.

each of these functions has independant probability of 50% of answering if the input function halts, otherwise it will return DONTKNOW.

As an example.

the program with godel number 999 is

10 goto 10

$\text{UTM}(2, 999) = \text{DONTKNOW}$ (ignoring the parameter of function 999)

$\text{UTM}(6, 999) = \text{DONTKNOW}$

$\text{UTM}(33, 999) = \text{DONTKNOW}$

$\text{UTM}(655, 999) = \text{DONTKNOW}$

...

Since :

$!\text{halt}(f, a) \rightarrow \text{pHalt}2(n) \rightarrow \text{UTM}(n, f(a)) = \text{'don't know'}$

the program with godel number 1000 is

10 print 10

$\text{UTM}(2, 1000) = \text{HALT}$

$\text{UTM}(6, 1000) = \text{DONTKNOW}$

$\text{UTM}(33, 1000) = \text{HALT}$

$\text{UTM}(655, 1000) = \text{DONTKNOW}$

Given :

$\text{halt}(f, a) \rightarrow \text{pHalt}2(n) \rightarrow P(\text{UTM}(n, f(a)) = \text{'halt'}) = 0.5$

Can you systematically determine if any given program will HALT?

sci.logic: Can you systematically determine if any given program will HALT?

>*From these outputs we can determine *without running the programs**

1/ program 999 does not halt with probability of error 1/16

2/ program 1000 halts

A **solution** to the halting problem is apparent, but it cannot be fully represented as an **algorithm**, so the halting proof will fail here to surface a contradiction.

Herc

This is the only article in this thread

--

10,000 WITNESSES TO US GOVERNMENT ABUSING GENESIS ADAM IN PUBLIC FOR 4 YEARS