

Re: Disproof of the Halting Problem's Conclusion

Source: <http://coding.derkeiler.com/Archive/General/comp.theory/2004-08/0347.html>

From: Chris Menzel (cmenzel_at_remove-this.tamu.edu)

Date: 07/22/04

Date: 22 Jul 2004 20:48:12 GMT

On Thu, 22 Jul 2004 00:05:33 GMT, Peter Olcott <olcott@worldnet.att.net> said:

> *It is not merely that I have not accepted a proof that everyone
> else accepts. It is that my disprove of this proof has not yet
> been found in error.*

Well, golly. I guess it wasn't clear to me that you thought you'd "disproved" the proof — not that it's ever been clear to me what you think your doing. Be that as it may, if you've "disproved" the proof, that means you think that either an invalid inference is made at some point, or you think one of the premises is false. So perhaps it will be useful if you explicitly identify the problem in a very clear and simple statement of the Halting Problem. Even if it might be pointless to try to get you to see that there is no problem, this will at least make it clear to posterity what it is that you see that everyone else doesn't, or vice versa. I've got an hour to kill, so let me join Daryl in the ranks of second-order crackpottery and try to spell out the argument carefully.

For simplicity, let's restrict ourselves to the very simple Turing machines found in the standard text *Computability and Logic* by Boolos, Jeffrey, and Burgess. These machines are capable of being in infinitely many distinct "states" q_0, q_1, \dots (one at a time) and operate on a tape containing infinitely many "cells" that are either blank or contain the single symbol X. (It is easy to show that increasing the number of symbols does not increase the computational power of Turing machines.) The "head" of a Turing machine is always located at some cell of the tape. Upon reading a cell, a TM can perform one of four actions: (i) write an X in the cell (whether or not one is already there), (ii) "write" a blank in the cell (i.e., erase an X or do nothing, if the cell is blank already), (iii) move left, or (iv) move right. The behavior of every Turing machine can then be described by means of a finite set of instructions of the form $\langle q_n, S, A, q_m \rangle$, which means: "If you are in state q_n and you see S, perform action A and go into state q_m ." Here, S will be either X or B (representing a blank) and A will be either X, B, L, or R representing the four actions just noted, respectively.

comp.theory: Re: Disproof of the Halting Problem's Conclusion

Note that it is easy to put the class of finite sets of instructions into a list, and this provides us with an easy way of listing the class of all TMs. Thus, every TM can be assigned a unique natural number n according to its place in the list. So by "Turing machine n " we mean the n^{th} TM in the list of TMs.

Now, in this simple model, we can think of a block of $n+1$ X's as representing the natural number n — in particular, a single X on its own represents the number 0. Consider now the following two definitions:

DEF 1: A Turing machine M *computes* the 1-place function f (from natural numbers to natural numbers) if and only if, for any natural number n on which f is defined, when started in state q_0 on the leftmost X of a block of $n+1$ X's on an otherwise blank tape, M halts at the leftmost X of a block of $m+1$ Xs (on an otherwise blank tape) if and only if $f(n) = m$. If f is undefined on n , then M does not halt when started in the configuration noted.

DEF 2: A Turing machine M *computes* the 2-place function g if and only if, for any pair of natural numbers m, n on which g is defined, when started in state q_0 on the leftmost X of a block of $m+1$ X's on a tape containing only that block and another block of $n+1$ X's to its right, separated by a single blank cell, M halts at the leftmost X of a block of $r+1$ Xs (on an otherwise blank tape) if and only if $g(m, n) = r$. If g is undefined on m, n , then M does not halt when started in the configuration noted.

Thus, if M^* computes, say, the multiplication function, then, when started on the leftmost X on a tape that looks like this ("B" represents a blank cell):

...BBBXXXXXXXXBXXXBBB...

M^* will halt on the leftmost X of a tape that looks like this:

...BBBXXXXXXXXXXXXXXXXBBB...

Now, consider the following definition of the 2-place function Halts on the natural numbers:

DEF 3: For natural numbers n, i , $\text{Halts}(n, i) = 1$ if Turing machine n eventually halts when started in state q_0 at the leftmost X of a block of $i+1$ Xs (on an otherwise blank tape), and $\text{Halts}(n, i) = 0$ otherwise, i.e., if Turing machine n does not eventually halt when started in the configuration noted.

Note this is just an ordinary, well-defined, 2-place function on the numbers. And note that it is a *total* function as well — it yields either 1 or 0 for any pair of arguments n, i . Here, then, is a nice, simple version of the Halting Problem:

comp.theory: Re: Disproof of the Halting Problem's Conclusion

HP: To find a Turing machine that computes the function Halts.

And here is an argument that HP is unsolvable, i.e., that there is no such Turing machine.

Note first that, given Halts, we can define another function Diag in terms of it:

DEF 4: For any natural number n , $\text{Diag}(n) = 1$ if $\text{Halts}(n,n) = 0$, and $\text{Diag}(n)$ is undefined otherwise.

Again, a perfectly legitimate mathematical function. Now for the argument.

1. There is a TM H that computes Halts. (Assumption for Reductio)
2. Given the instructions for H , we can easily write a set of instructions for a TM D that computes the 1-place function Diag .[*] D is a TM, so it occurs somewhere in our list of all TMs. So D is Turing machine n^* , for some natural number n^* .
3. Either $\text{Diag}(n^*) = 1$ or $\text{Diag}(n^*) \neq 1$.
4. If $\text{Diag}(n^*) = 1$, then, by definition of Diag (DEF 4), $\text{Halts}(n^*,n^*) = 0$, and hence, by the definition of Halts (DEF 3), Turing machine n^* fails to halt when started in state q_0 on the leftmost X of a block of n^*+1 X s. But Turing machine n^* is D . Hence, it is D that fails to halt when started in the configuration noted. But D computes Diag , and hence, since $\text{Diag}(n^*) = 1$, by the definition of computation (DEF 1) D must halt when started in the configuration noted after all. But we can't have it both ways. Hence, it must be false that $\text{Diag}(n^*) = 1$.
5. If, on the other hand, $\text{Diag}(n^*) \neq 1$, then, by DEF 4 again, $\text{Diag}(n^*)$ is undefined. Hence, since D computes Diag , by DEF 1 again D does not halt when started on the leftmost X of a block of n^*+1 X s. By DEF 3, then, since D is Turing machine n^* , $\text{Halts}(n^*,n^*) = 0$. But then by DEF 4 again, we have $\text{Diag}(n^*) = 1$, after all. So it's false as well that $\text{Diag}(n^*) \neq 1$.
6. So $\text{Diag}(n^*)$ can neither be equal to 1 nor not equal to 1, a contradiction.
7. Hence, our assumption above is false, i.e., there is no TM that computes the function Halts, i.e., HP is unsolvable.

Ok, that seems reasonably clear, if somewhat wordier than I'd planned on when I started. If you would, then, please "disprove" the proof, i.e., identify what you take to be the false premise or the unsound inference.

Chris Menzel

comp.theory: Re: Disproof of the Halting Problem's Conclusion

[*]Informally, D will take a block of $n+1$ Xs, make an adjacent copy, and then carry out the instructions for H; if that H "subroutine" returns a single X (i.e., if $\text{Halts}(n,n) = 0$), D will write another X to its left and halt (i.e., return "1"), otherwise D will loop, e.g., write an X and erase it over and over, forever.