Impossible to Solve Computational Problems

Richard Cole

August 28, 2007

Self-reference is well-known for creating paradoxes, as in the following conundrum:

This sentence is false.

For if the sentence is true then it is false, and if false then it is true.

Analogous difficulties arise with questions about programs, for programs (viewed as text) can be the input for other programs (e.g. compilers) including themselves (a compiler viewed as text could be the input to the same compiler viewed as a program).

We will now proceed to prove there is no algorithm (program) to solve the Halting Problem: Is there an algorithm which given inputs \( P \) and \( x \), where \( P \) is a program (in your favorite programming language) and \( x \) is an input to \( P \), answers whether \( P \) will eventually terminate its computation on input \( x \).

The answer is one of “Yes” or “No”. Note that simulating \( P \) on input \( x \) is not an adequate solution. For while if \( P \) does terminate on input \( x \), the simulation will eventually discover this, and then an answer of “Yes” can be given, if \( P \) does not terminate this will never be discovered and no answer will result.

Note that not terminating might be the result of a infinite loop or an unbounded recursion.

The question is not whether for some particular \( P \) and \( x \) termination can be determined, but whether there is a systematic procedure than can determine it for every possible pair of \( P \) and \( x \). As we will see, there is no such procedure.

Suppose our programs are written in binary and likewise for their inputs. (In practice they are written in the 256-character ASCII code, i.e. 8 bits per character). Now imagine listing all strings of binary characters in lexicographic order: length 1 strings ordered alphabetically, then length 2 strings, and so forth. i.e., 0, 1, 00, 01, 10, 11, 000, 001, etc. Clearly, this listing includes all legitimate programs, as well as many other strings.

To prove our result, let us assume, in order to obtain a contradiction, that there is a 2-input program \( H \) that solves the halting problem.

Given the program for \( H \), we create a new one-input program \( D \), defined as follows

\[
D(x) = \begin{cases} 
\text{Simulate } H(x, x); \\
\text{If } H(x, x) \text{ outputs “Yes” then loop forever} \\
\text{else (} * H(x, x) \text{ outputs “No” and } * \text{) } D(x) \text{ outputs “Yes”}
\end{cases}
\]

Side note: If \( x \) is not a program, we define \( H(x, y) = “Yes” \).
What does $D(D)$ output?

Well, if $H(D, D)$ outputs “No” then $D(D)$ outputs “Yes”, meaning that the computation of $D$ on input $D$ eventually terminates, in which case $H(D, D)$ outputs “Yes”. Likewise, if $H(D, D)$ outputs “Yes” then $D(D)$ loops forever, i.e. does not terminate, in which case $H(D, D)$ outputs “No”.

Either way, we have a contradiction, and thus the assumption that $H$ exists must be false.

**Comment** We have actually shown that there is no program $H$ that can answer the question of “does program $P$ on input $P$ eventually halt?” for all possible programs $P$.

Notice that $D$ is a self-referential program. Of course, as we have shown that $H$ does not exist, it also follows that $D$ does not exist.