**Halting Problem:**

Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist.

For any program f that might determine if programs halt, a "pathological" program g called with an input can pass its own source and its input to f and then specifically do the opposite of what f predicts g will do. No f can exist that handles this case.

A key part of the proof was a mathematical definition of a computer and program, which became known as a Turing machine; the halting problem is undecidable over Turing machines. Turing's proof is one of the first cases of decision problems to be concluded.

The theoretical conclusion that it is not solvable is significant to practical computing efforts, defining a class of applications which no programming invention can possibly perform perfectly.

The halting problem is a decision problem about properties of computer programs on a fixed Turing-complete model of computation, i.e., all programs that can be written in some given programming language that is general enough to be equivalent to a Turing machine. The problem is to determine, given a program and an input to the program, whether the program will eventually halt when run with that input. In this abstract framework, there are no resource limitations on the amount of memory or time required for the program's execution; it can take arbitrarily long and use an arbitrary amount of storage space before halting. The question is simply whether the given program will ever halt on a particular input.

For example, in pseudocode, the program

while (true) continue

does not halt; rather, it goes on forever in an infinite loop. On the other hand, the program

print "Hello, world!"

does halt.

While deciding whether these programs halt is simple, more complex programs prove problematic.

One approach to the problem might be to run the program for some number of steps and check if it halts. But if the program does not halt, it is unknown whether the program will eventually halt or run forever.

Turing proved no algorithm exists that always correctly decides whether, for a given arbitrary program and input, the program halts when run with that input. The essence of Turing's proof is that any such algorithm can be made to contradict itself and therefore cannot be correct.

The difficulty in the halting problem lies in the requirement that the decision procedure must work for all programs and inputs. A particular program either halts on a given input or does not halt. Consider one algorithm that always answers "halts" and another that always answers "doesn't halt". For any specific program and input, one of these two algorithms answers correctly, even though nobody may know which one. Yet neither algorithm solves the halting problem generally.

There are programs (interpreters) that simulate the execution of whatever source code they are given. Such programs can demonstrate that a program does halt if this is the case: the interpreter itself will eventually halt its simulation, which shows that the original program halted. However, an interpreter will not halt if its input program does not halt, so this approach cannot solve the halting problem as stated; it does not successfully answer "doesn't halt" for programs that do not halt.

The halting problem is theoretically decidable for linear bounded automata (LBAs) or deterministic machines with finite memory