**CSE 201**
**Analysis of Algorithms**
**The Halting Problem**

For the following informal discussion, we fix some modern programming language such as Python or Java and consider programs $P$ written in that language. If $I$ is a string we write $P(I)$ to denote the output of program $P$ when run with $I$ as input. We restrict our discussion to programs $P$ having one of only three possible results.

$$P(I) \quad = \quad \begin{cases} 1 \text{ (indicating 'yes')} \\ 0 \text{ (indicating 'no')} \\ \text{no output (i.e. the program enters an infinite loop)} \end{cases}$$

In the first two cases we say that the program *halts*, while in the third case the program *does not halt*. Note that any such program $P$ can itself be considered a string and may therefore be used as input to another program. In particular $P(P)$ denotes the result of running $P$ with its own source code as input. Given this background we can now state (informally):

Halting Problem
Write a program $H$ that, given any program $P$ and any input string $I$, outputs the following:

$$H(P,\ I) \quad = \quad \begin{cases} 1 & \text{if } P \text{ halts on input } I \\ 0 & \text{if } P \text{ does not halt on input } I \end{cases}$$

Note that $H$ by definition must always halt. Observe also that it is unclear exactly how to construct such a program. One's first impulse might be to simply have $H$ run program $P$ on input $I$. This strategy cannot work however, since if $P$ does not halt on $I$, the simulation will not terminate and 0 will never be returned. How long must one wait before declaring that $H(P,\ I)$ returns 0? Since this is not known, just running $P$ is not a feasible strategy. Program $H$ must somehow analyze the structure of both $P$ and $I$ to determine whether or not $P(I)$ halts. A human being (with both programming experience and patience) can sometimes perform this task, but can we write a program that does it successfully in all cases?

Theorem
No such program $H$ can exist.

Proof
Assume, to get a contradiction, that program $H$ exists. We can use $H$ to define another program, call it $Z$, that outputs the following:

$$Z(P) \quad = \quad \begin{cases} 1 & \text{if } H(P,\ P) = 0 \\ \text{no output (infinite loop)} & \text{if } H(P,\ P) = 1 \end{cases}$$

Program $Z$ can be constructed by simulating $H$ on the pair of strings $(P,\ P)$. If $H(P,\ P)$ outputs 0, then $Z$ should output 1. If $H(P,\ P)$ outputs 1, then $Z$ enters an infinite loop.

Now consider what the result of $Z(Z)$ might be. Supposedly it either halts (outputting 1) or it enters an infinite loop. If $Z(Z)$ halts, then by the definition of $Z$, $H(Z, Z)$ must output 0. By the definition of $H$ then, we have that $Z(Z)$ does not halt, which is a contradiction. On the other hand if $Z(Z)$ enters an

infinite loop, then by the definition of $Z$, $H(Z,Z)$ must output 1. The definition of $H$ now says that $Z(Z)$ halts, another contradiction. So each possible result of $Z(Z)$ leads to a contradiction. The only way to resolve these contradictions is to conclude that no such program $H$ can exist. ■

This result was proved by Alan Turing in 1936 (*On computable numbers, with an application to the Entscheidungsproblem*.) This is the epochal paper in which Turing defines *Turing Machines*, formulates the halting problem, and shows that it is unsolvable. The above discussion must be considered informal however since we have not yet introduced any rigorous and formal definitions of program, computer or computability.