

ECE120: Introduction to Computer Engineering

Notes Set 1.1 The Halting Problem

For some of the topics in this course, we plan to cover the material more deeply than does the textbook. We will provide notes in this format to supplement the textbook for this purpose. In order to make these notes more useful as a reference, definitions are highlighted with boldface, and italicization emphasizes pitfalls or other important points. *Sections marked with an asterisk are provided solely for your interest, but you probably need to learn this material in later classes.*

These notes are broken up into four parts, corresponding to the three midterm exams and the final exam. Each part is covered by one examination in our class. *The last section of each of the four parts gives you a summary of material that you are expected to know for the corresponding exam.* Feel free to read it in advance.

As discussed in the textbook and in class, a **universal computational device** (or **computing machine**) is a device that is capable of computing the solution to any problem that can be computed, provided that the device is given enough storage and time for the computation to finish.

One might ask whether we can describe problems that we cannot answer (other than philosophical ones, such as the meaning of life). The answer is yes: there are problems that are provably **undecidable**, for which no amount of computation can solve the problem in general. This set of notes describes the first problem known to be undecidable, the **halting problem**. For our class, you need only recognize the name and realize that one can, in fact, give examples of problems that cannot be solved by computation. In the future, you should be able to recognize this type of problem so as to avoid spending your time trying to solve it.

1.1.1 Universal Computing Machines*

The things that we call computers today, whether we are talking about a programmable microcontroller in a microwave oven or the Blue Waters supercomputer sitting on the south end of our campus (the United States' main resource to support computational science research), are all equivalent in the sense of what problems they can solve. These machines do, of course, have access to different amounts of memory, and compute at different speeds.

The idea that a single model of computation could be described and proven to be equivalent to all other models came out of a 1936 paper by Alan Turing, and today we generally refer to these devices as **Turing machines**. All computers mentioned earlier, as well as all computers with which you are familiar in your daily life, are provably equivalent to Turing machines.

Turing also conjectured that his definition of computable was identical to the "natural" definition (today, this claim is known as the **Church-Turing conjecture**). In other words, a problem that cannot be solved by a Turing machine cannot be solved in any systematic manner, with any machine, or by any person. This conjecture remains unproven! However, neither has anyone been able to disprove the conjecture, and it is widely believed to be true. Disproving the conjecture requires that one demonstrate a systematic technique (or a machine) capable of solving a problem that cannot be solved by a Turing machine. No one has been able to do so to date.

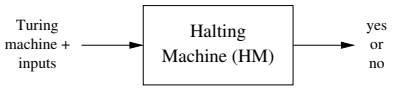
1.1.2 The Halting Problem*

You might reasonably ask whether any problems can be shown to be incomputable. More common terms for such problems—those known to be insolvable by any computer—are **intractable** or undecidable. In the same 1936 paper in which he introduced the universal computing machine, Alan Turing also provided an answer to this question by introducing (and proving) that there are in fact problems that cannot be computed by a universal computing machine. The problem that he proved undecidable, using proof techniques almost identical to those developed for similar problems in the 1880s, is now known as **the halting problem**.

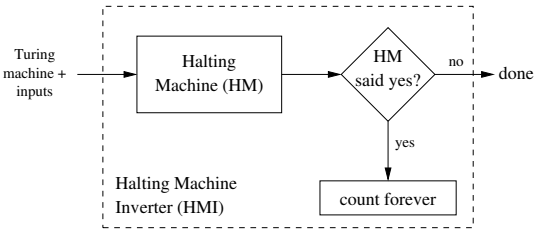
The halting problem is easy to state and easy to prove undecidable. The problem is this: given a Turing machine and an input to the Turing machine, does the Turing machine finish computing in a finite number of steps (a finite amount of time)? In order to solve the problem, an answer, either yes or no, must be given in a finite amount of time regardless of the machine or input in question. Clearly some machines never finish. For example, we can write a Turing machine that counts upwards starting from one.

You may find the proof structure for undecidability of the halting problem easier to understand if you first think about a related problem with which you may already be familiar, the Liar's paradox (which is at least 2,300 years old). In its stengthened form, it is the following sentence: "This sentence is not true."

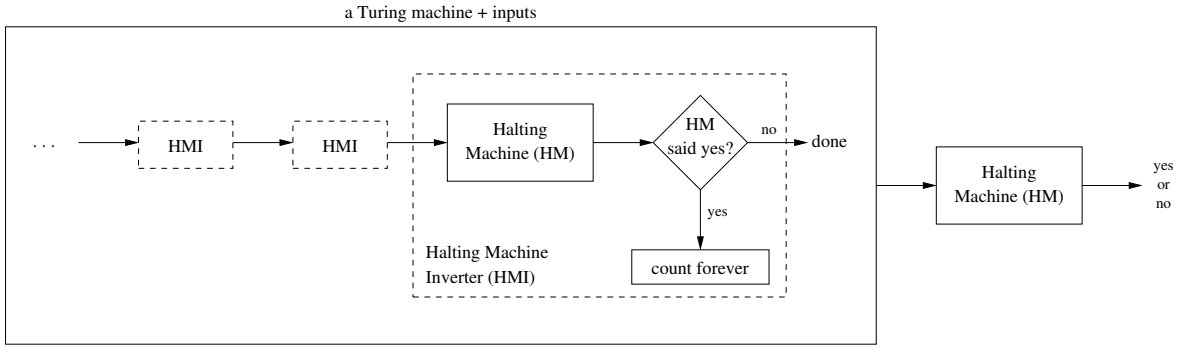
To see that no Turing machine can solve the halting problem, we begin by assuming that such a machine exists, and then show that its existence is self-contradictory. We call the machine the "Halting Machine," or HM for short. HM is a machine that operates on another Turing machine and its inputs to produce a yes or no answer in finite time: either the machine in question finishes in finite time (HM returns "yes"), or it does not (HM returns "no"). The figure illustrates HM's operation.



From HM, we construct a second machine that we call the HM Inverter, or HMI. This machine inverts the sense of the answer given by HM. In particular, the inputs are fed directly into a copy of HM, and if HM answers "yes," HMI enters an infinite loop. If HM answers "no," HMI halts. A diagram appears to the right.



The inconsistency can now be seen by asking HM whether HMI halts when given itself as an input (repeatedly), as shown below. Two copies of HM are thus being asked the same question. One copy is the rightmost in the figure below and the second is embedded in the HMI machine that we are using as the input to the rightmost HM. As the two copies of HM operate on the same input (HMI operating on HMI), they should return the same answer: a Turing machine either halts on an input, or it does not; they are deterministic.



Let's assume that the rightmost HM tells us that HMI operating on itself halts. Then the copy of HM in HMI (when HMI executes on itself, with itself as an input) must also say "yes." But this answer implies that HMI doesn't halt (see the figure above), so the answer should have been no!

Alternatively, we can assume that the rightmost HM says that HMI operating on itself does not halt. Again, the copy of HM in HMI must give the same answer. But in this case HMI halts, again contradicting our assumption.

Since neither answer is consistent, no consistent answer can be given, and the original assumption that HM exists is incorrect. Thus, no Turing machine can solve the halting problem.