MITOCW | MIT6_004S17_10-02-07_300k

This video is optional, but we wanted to answer the question "Are there uncomputable functions?" Yes, there are well-defined discrete functions that cannot be computed by any TM, i.e., no algorithm can compute f(x) for arbitrary finite x in a finite number of steps.

It's not that we don't know the algorithm, we can actually prove that no algorithm exists.

So the finite memory limitations of FSMs wasn't the only barrier as to whether we can solve a problem.

The most famous uncomputable function is the so-called Halting function.

When TMs undertake a computation there two possible outcomes.

Either the TM writes an answer onto the tape and halts, or the TM loops forever.

The Halting function tells which outcome we'll get: given two integer arguments k and j, the Halting function determines if the kth TM halts when given a tape containing j as the input.

Let's quick sketch an argument as to why the Halting function is not computable.

Well, suppose it was computable, then it would be equivalent to some TM, say T_H.

So we can use T_H to build another TM, T_N (the "N" stands for nasty!) that processes its single argument and either LOOPs or HALTs.

T_N[X] is designed to loop if TM X given input X halts.

And vice versa: T_N[X] halts if TM X given input X loops.

The idea is that T_N[X] does the opposite of whatever T_X[X] does.

T_N is easy to implement assuming that we have T_H to answer the "halts or loops" question.

Now consider what happens if we give N as the argument to T_N.

From the definition of T_N, T_N[N] will LOOP if the halting function tells us that T_N[N] halts.

And T_N[N] will HALT if the halting function tells us that T_N[N] loops.

Obviously T_N[N] can't both LOOP and HALT at the same time!

So if the Halting function is computable and T_H exists, we arrive at this impossible behavior for T_N[N].

This tells us that T_H cannot exist and hence that the Halting function is not computable.