MIT 6.045J - Automata, Computability, Complexity
Lecture 5 - Turing Machines

Lecture 5 - Turing Machines

Alan Turing, in 1936, proposed a more powerful model of the finite automata, called the Turing machine (On Computable Numbers, A.M. Turing, 1936).

A Turing machine is a finite automata which has the ability to read a symbol and transition to another state, just like a DFA, but it can also write a symbol on the same paper it is reading from, and it can further move the paper it is reading from either to the right or to the left. It turns out that this model is powerful enough to be able to make arbitrary finite computations.

For example, we can have a Turing machine that accepts the language of palindromes. Let's say we have an infinitely long tape with an input. The start and end of the input are marked with special symbols, say `#`, so we know that the input has started or ended. Then we can design a Turing machine to accept the input only if it is a palindrome by the following (rough) algorithm -

1. Start in a some start state.
2. If you read a one in the start state, overwrite it with a `#` and keep moving to the right till you reach the end of the input.
3. If the last symbol read is also 1, replace it with a `#` and come back to the start of the input (the 1 that we overwrote with a #), and repeat.

Church-Turing Hypothesis

Now that we have such a powerful model of computation, you may wonder why not keep making it even more powerful. What if we have random access in our model of computation, or multiple tapes to read from and write to instead of just one. It turns out that all of these models of computation are no more powerful than a Turing machine, because they can be simulated by a Turing machine.

This is called the Church-Turing thesis. It is a hypothesis and doesn't have a proof, and could be disproved some time in the future, but for now we haven't found a

counterexample.

The Church-Turing thesis says that any model of computation you can conceive that follows the physical laws of our universe can be simulated by a regular Turing machine.

The lecture gives two examples of such "reductions". A Turing machine that has a one-way infinite tape instead of a two-way infinite tape is equivalent to a two-way infinite tape Turing machine. A Turing machine that can read from two tapes simultaneously is equivalent to a regular Turing machine.

Computable Language

Finally, we define a language that a Turing machine can accept. Just as a language that was accepted by a DFA was called a regular language, and a language that was accepted by a context-free grammar was called a context-free language, a language that is accepted by a Turing machine is called a computable language.