MIT 6.045J - Automata, Computability, Complexity
Lecture 6 - Decidability

Lecture 6 - Decidability

An important theorem regarding Turing machines is that a Turing machine can simulate other Turing machines, which can potentially be more complicated than itself. That is, a Turing machine A can act as an *interpreter* for Turing machine B, and B can be much more complicated than A.

Alan Turing actually gave such an example of an interpreter in his 1936 paper, but it turned out to have a mistake, so he had to publish a correction in 1937. Regardless, it can be done.

Halting Problem

A question you might have now is what can Turing machines not do? One famous example of a problem Turing machines can't solve is the halting problem - given a description of a Turing machine, return whether the machine will halt on an empty input.

For some Turing machines (and some computer programs), we can easily tell if the machine will halt or get stuck in an infinite loop. However, we cannot know that for sure for all Turing machines.

If we did know how to solve the halting problem, we would also be able to solve many unsolved problems in mathematics, like the Riemann hypothesis, or Goldbach's conjecture.

The proof that we can't solve the halting problem is given by contradiction as follows. Say you have a Turing machine P that takes the description of a Turing machine as input and tells you whether the input will halt on an empty input. This is not a special case, as we can always convert a Turing machine that takes some non-empty input into another Turing machine that takes an empty input but performs the same operations as the first Turing machine on the specific non-empty input.

Now, say we have another machine Q that takes the description of a Turing machine

M, runs P on it, and goes into an infinite loop if M halts, and halts if M loops. If we pass the description of Q into Q itself, we get a paradox. Therefore, we contradict ourselves, and we cannot have a Turing machine P.

Busy Beaver & Infinities

Busy Beaver is a sequence that grows faster than any other countable sequence. Scott introduces it in the lecture as a way to introduce countability, sets of infinities, and prove that most languages are not countable.

The Busy Beaver sequence is defined as follows, `BB(n)` is the maximum number of steps an `n` state Turing Machine can take on an empty input before halting. Of course, the maximum number of steps an `n` state Turing Machine can take is infinity, since it can just go into an infinite loop, so we restrict our discussion only to those Turing machines that halt on an empty input.

The first few values of the `BB` sequence are as follows -

`BB(1)` | 1 |
`BB(2)` | 6 |
`BB(3)` | 21 |
`BB(4)` | 107 |
`BB(5)` | >= 47,176,870 |
`BB(6)` | $>3 \cdot 10^{1730}$ |

We are not yet sure about the values of `BB(5)` and above. It turns out that this sequence grows faster than any computable sequence. We prove this by contradiction. Let's say that the Busy Beaver sequence was a computable sequence. Then we could design a Turing Machine to take an integer N as input and output `BB(N)`. However, if we knew `BB(N)`, we could use that fact to solve the halting problem. We could just run a Turing Machine (with N states), and if it took more than `BB(N)` steps, then we could be sure that the Turing Machine never halted.

Since we *can't* solve the halting problem, this is a contradiction, and the Busy Beaver sequence is not countable.

Scott then introduces the different types of infinities - countable and uncountable.

A countable infinity is any set that can be shown to have a one-to-one mapping with the set of integers. For example, the set of all even numbers can be shown to have a one-to-one mapping with the set of integers. This means that the set of all even numbers is countably infinite (interestingly, it also means that there are as many even numbers as integers). Similarly, the rational numbers are countably infinite.

Also, the set of all Turing Machines is countable. This is because each Turing machine can be represented as a series of transitions from each of its states, and all

of these transitions can be encoded as a binary string. Since the set of all binary strings is just the set of all integers, the set of all Turing Machines is countable.

An important observation is that the set of all Languages is not countable. The lecture has a simple proof. This means that if we choose a language at random, the probability that a Turing Machine recognizes that language is 0%.

The set of countable infinity is called $\aleph$ (Aleph), and the set of uncountable infinities is called $2^{\aleph}$. In general, there is no largest set of infinity. You can always keep constructing sets that have more elements than the largest infinity you have by just taking the set of all subsets of that infinity.