MIT 6.045J - Automata, Computability, Complexity
Lecture 2 - Logic, Circuits, and Gates

Lecture 2 - Logic, Circuits, and Gates

Gates and Universality

This lecture introduces the fundamental logic gates - AND, OR, and NOT, and then talks about their universality.

Universality in this case means that we can construct any boolean function using a combination of these gates. In computer science, universality is the norm, and most sets are usually universal. You have to work a little to get a set that is not universal. Some examples for non-universal sets in logic gates are - {AND, OR}, and {XOR, NOT}.

This is because AND gates and OR gates are monotonic functions. That is, changing one of the inputs from a 0 to a 1 can only change the output from a 0 to a 1, or leave it unaffected. Whereas to represent a NOT gate, we would need to change the output from a 1 to a 0 when the input went from 0 to 1, which is not possible to do with monotonic gates.

An example of a universal set of gates is {AND, OR, NOT}. To prove that this set is universal, we need to prove that we can use these gates to produce an arbitrary boolean expression of any number of bits. We can do this using a truth table.

For any arbitrary boolean expression, draw a truth table or all possible input combinations. Then, only consider the input combinations which output a 1. For each such input combination, build a circuit that would output a 1. That is, if one input combination that outputs a 1 for 3 inputs is $ {X, Y, Z} = {0, 0, 1} $, then we would build a circuit for this combination as $\neg X \wedge \neg Y \wedge Z$.

Once we have circuits for each such row (that outputs a 1 in the final boolean expression), we just OR all of those circuits to get our expression.

This is the same technique I learned in college. However, this is pretty inefficient, because it produces a circuit of size $O(n \cdot 2^n)$. The size of a circuit is the same as

the number of gates in the circuit. That's pretty bad, so we need to improve our approach, which we will do shortly.

Another set of universal gates we can use is {AND, NOT}. This is because we can simulate the OR gate using an AND gate and a NOT gate. Another set of universal gates is {NAND} . We can use the NAND gate to simulate any other gate, hence it is universal (so is the NOR gate). But these sets don't help us either, because using the same approach as above with these sets of universal gates would only affect the size of the circuit by a constant factor.

Gates and P vs NP

Now you might wonder if there is a bound on the number of gates we will need to represent any arbitrary boolean expression of n bits. That is, if we have any arbitrary boolean function of n bits, what is the smallest size of the circuit we can build to represent it? Using the approach above, we definitely know that it can be done in at most $O(n \cdot 2^n)$ bits, but can we come up with a better bound?

Proving that a specific circuit definitely needs an exponential number of gates and it cannot be represented in any fewer is an open problem. If you think about it, it's the same as P vs NP. If you build a circuit to represent an NP-complete problem, you can't prove that it needs an exponential number of gates (at least no one has proven it yet), and proving it would be the same as proving $P \neq NP$.

However, we can prove that a majority of boolean expressions need a circuit of exponential size. Sure, there are some boolean expressions for which we can have as small circuit, but most require a large circuit. This was proven by Claude Shannon in 1949 using a counting argument.

The gist of the argument is as follows - we count the number of boolean expressions which take n bits as input, we count the number of circuits that we can build using n inputs of size S, and we compare the two quantities to realize how large S has to be to equal the number of boolean expressions with n inputs.

The number of boolean expressions that take n bits as input is 2^{2^n}. This is because for an n bit boolean expression, the truth table has 2^n rows. Now for each row, the output value could be 0 or 1, so the number of possible truth tables is 2^{2^n}.

Now, let's say we want to build a circuit of size S, which takes n bits as input. The number of circuits of size 1 which take n bits as input is just $nC2$, since any 2 input bits could be used as the input to a gate. Similarly, the number of circuits of size 2 is $\binom{n}{2} \cdot \binom{(n+1)}{2}$, since adding the first gate creates a new input to which we can attach the second gate. If we keep going, we find that the number of circuits of size S using n inputs is $\binom{n}{2} \cdot \binom{(n+1)}{2} \cdot ... \cdot \binom{(n+S-1)}{2}$. We can approximate this to be at most $(n+S)^{2S}$.

Now we want

$ (n+S)^{2S} = 2^{{2}{n}} $

Which gives us

$S = \Theta(\frac{2^n}{n})$

This means that to represent a boolean expression with n inputs, most circuits will have to be of size $S = \Theta(\frac{2^n}{n})$.