Lecture 12: Boolean bases, memory and sequential ciruits

This lecture gives several complete bases for the boolean functions, and begins our look at sequential circuits and how they are used to implement memory in computers.

Boolean bases

By the sum-of-products method, we know that any boolean function can be represented using the boolean functions: AND, OR, and NOT. These three functions form a complete basis for the boolean functions.

In fact, just the two functions AND and NOT form a complete basis for the boolean functions. To see this, it suffices to express the function OR in terms of AND and NOT.

Two of the axioms of boolean algebra, called DeMorgan's laws, which are

$$(x + y)' = x' \cdot y'$$

and

$$(x \cdot y)' = x' + y',$$

can help us with this. Using the additional axiom that

$$x = x'',$$

we have that

$$(x + y) = (x + y)'' = (x' \cdot y')',$$

which expresses OR in terms of AND and NOT. An alternative check is to verify that the truth-table for $(x' \cdot y')'$ is the same as that of $(x + y)$.

Dually, the two functions OR and NOT form a complete basis for the boolean functions, using the fact that

$$(x \cdot y) = (x' + y')',$$

which expresses AND in terms of OR and NOT.

Question: what about the two functions XOR and NOT? Are they a complete basis for the boolean functions? Can they express AND? If not, how could you show that they could not express AND?

We now consider another boolean function of two inputs: NAND. The truth table for NAND is;

```
 x  y  |  (x NAND y)
--------------------
 0  0  |      1
 0  1  |      1
 1  0  |      1
 1  1  |      0
```

This is the negation of the values for `(x AND y)`, so a boolean expression for NAND is

$$(x \cdot y)'.$$

The gate drawing for NAND is an AND gate with a little circle where the output line attaches. The little circle denotes negation, as it does for the NOT gate.

The single function NAND by itself is a complete basis for the boolean functions. It suffices to show how to express AND and NOT using NAND, because AND and NOT form a complete basis. We do not have very many choices for NOT: there is only one input, while NAND requires two, so we consider `(x NAND x)`, which has the truth table:

```
 x  |  (x NAND x)
----------------
 0  |      1
 1  |      0
```

Thus, NOT can be expressed with NAND. Because $x = x''$, if we negate NAND, we get AND back:

```
 x  y  |  ((x NAND y) NAND (x NAND y))
-------------------------------------
 0  0  |               0
 0  1  |               0
 1  0  |               0
 1  1  |               1
```

Since NAND suffices to express both NOT and AND, it is a complete basis for the boolean functions. In other words, if we have an unbounded supply of NAND gates, we could build a circuit to compute any boolean function whatsoever.

We would expect a dual result for OR, and we are not disappointed. The boolean function NOR is the negation of OR, and has truth table:

```
 x  y  |  (x NOR y)
------------------
 0  0  |     1
 0  1  |     0
 1  0  |     0
 1  1  |     0
```

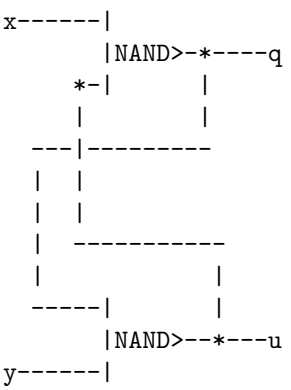
Its circuit symbol is the same as an OR gate, with a little circle (signifying negation, again) where the output line attaches. The single function NOR is a complete basis for the boolean functions. To see this, it suffices to express NOT and OR (left to the reader.)

To summarize, the complete bases we've considered are

```
1.  AND, OR, NOT
2.  AND, NOT
3.  OR, NOT
4.  NAND
5.  NOR
```

Sequential circuits and memory

Now we'll build a simple circuit using NAND gates that has some very different properties from the combinational circuits we have been considering:



This is meant to depict two NAND gates, the input to the first being x and the output of the second, and the input to the second being y and the output of the first. The output of the first NAND gate is q and the output of the second NAND gate is u.

What can this mean? There are loops in this circuit – if I trace the output wire of the first NAND gate around to the input of the second NAND gate, to the output of the second NAND gate, to the input of the first NAND gate, to the output of the first NAND gate, I am back where I started: a loop. Thus, this circuit is sequential. The rules for evaluating what its outputs will be given certain inputs are not the same as for combinational circuits, where we can just propagate the values from the inputs toward the outputs without fear of loops.

To figure out what is happening here, we consider the equations:

$$q = (x \cdot u)'$$

and

$$u = (y \cdot q)'.$$

A solution of these equations consists of an assignment of 0's and 1's to the variables x, y, q, and u such that both equations are true. There are 16 possible combinations to test, and we find that there are just 5 combinations that give a solution for both equations:

x	y	q	u
0	0	1	1
0	1	1	0
1	0	0	1
1	1	1	0
1	1	0	1

The solutions correspond to stable states of the corresponding circuit. The last two lines show that neither q nor u is a function of the values of x and y, because we can have $x = y = 1$ and $q = 1$ or $q = 0$, and similarly for u. The values of q and u are not uniquely determined by the values of x and y; the circuit has state, or memory.

We exploit the state to store information as follows. We AVOID setting x and y to 0 simultaneously, and thereby avoid the first line of the table. Then there are two states of the circuit: where $q = 1$ and $u = 0$,

and where $q = 0$ and $u = 1$. If the inputs are $x = 1$ and $y = 1$, the circuit stays in whichever state it is currently in. If the inputs are $x = 0$ and $y = 1$, the circuit goes from whichever state it is currently in to the state where $q = 1$ and $u = 0$. If the inputs are $x = 1$ and $y = 0$, the circuit goes from its current state to the state where $q = 0$ and $u = 1$.

This use of the circuit allows us to store one bit of information. That is, $q = 1$ signifies that the last time the inputs were not both 1, it was x that was 0. Correspondingly, $q = 0$ signifies that the last time the inputs were not both 1, it was y that was 0.