University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Logical Completeness

Can We Count Functions?

A question for you:

How many different Boolean functions exist for N bits of input?

How can we find the answer?

Start by thinking about small values of **N**.

For example, **N=2**. Given **C = F(A,B)**, **how many choices do we have for F**?

Two Bits of Input Can be Combined into 16 Functions

Write a truth table for **C = F(A,B)**.

But instead of filling in values, call the outputs c_i.

The four c_i values uniquely specify **F**.

If we change any c_i, we get a different function.

A	B	C
0	0	c_0
0	1	c_1
1	0	c_2
1	1	c_3

We thus have **$2 \times 2 \times 2 \times 2 = 2^4$ choices for F**.

Three Bits of Input Can be Combined into 256 Functions

What about **N=3**: **D = G(A,B,C)**?

We can again write a truth table.

And call the outputs d_i.

Now we have **2^8 choices for G**.

Notice that $2^8 = 2^{(2^3)}$.

A	B	C	D
0	0	0	d_0
0	0	1	d_1
0	1	0	d_2
0	1	1	d_3
1	0	0	d_4
1	0	1	d_5
1	1	0	d_6
1	1	1	d_7

N Bits of Input Can be Combined into Many Functions

Can we generalize to **N** bits?

Without drawing a truth table, please?

N bits means $\mathbf{2^N}$ rows in the truth table.

Thus we need $\mathbf{2^N}$ Boolean values (bits) to specify a function.

Thus $\mathbf{2^{(2^N)}}$ **possible functions on N bits**.

We Need More Functions!

So why did we teach you only four functions?

(AND, OR, NOT, and XOR)

Your homework for next time:

Write down and name all functions on 10 bits.

Include a truth table for each function!

Alternate Homework: Understand Logical Completeness

Claim:

With enough 2-input AND, 2-input OR, and NOT functions, one can produce **any function on any number of variables**.

Believe me?

Proof: **by construction**

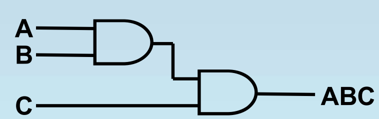
In other words, I'll show you how to produce an arbitrary function on an arbitrary number of variables.

Compose Functions to Produce Functions on More Inputs

Let's start the proof.

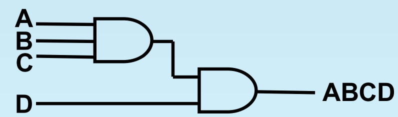
What does this circuit produce?

A **3-input AND**!



What about this one?

A **4-input AND**!

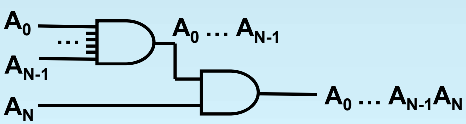


Use 2-Input Gates to Construct N-Input Gates

By induction, we build an **N**-input AND gate.

Base case (**N=2**): Use one 2-input AND.

N+1 case (given an **N**-input AND):
- Use one **N**-input AND
- and one 2-input AND
- to produce an (**N+1**)-input AND.



Comments on Functional Form and Practical Value

A couple of comments before we continue…
- Functional form of the inductive proof:
  - base: $\mathbf{AND_2\ (A, B) = AND_2\ (A, B)}$
  - inductive step:
    $$\mathbf{AND_{N+1}\ (A_0, \ldots, A_N) = AND_2\ (AND_N\ (A_0, \ldots, A_{N-1}), A_N)}$$
- This approach is an existence proof, **not a practical way to build bigger gates.**

The Claim is Now Slightly Simpler

Claim:

With enough ~~2-input~~ AND, ~~2-input~~ OR, and NOT functions, I can produce **any function on any number of variables**.

(For OR functions, use the same approach as we did with AND functions, replacing AND with OR.)

Let's first consider functions that
- produce an output of 1
- for exactly one combination of inputs (one row of the function's truth table).

One AND Suffices for Functions that Output One 1

The function **Q(A,B,C)** is an example of such a function.

When is **Q=1**?

Only when **A=1 AND B=0 AND C=1**.

Note that **B=0** when **(NOT B) = 1**.

In other words, **Q = AB'C**.

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Arbitrary Functions Require Only Two Steps

To produce **an arbitrary function** (which may produce the value 1 for more than one combination of inputs):

1. For each combination of inputs for which the function produces a 1, AND together the corresponding inputs or inverted inputs.*
2. OR together the results of all AND functions.

* The resulting AND is called a **minterm** on the input variables.

A Sum-of-Products Can Express Any Function

The construction described results in a **sum-of-products** form because

- we produce each row of the truth table with an AND (product / multiplication notation)
- we produce the final function by ORing the ANDs (sum / addition notation).

The approach described is **often inefficient**, but it **always works**.

{AND, OR, NOT} is Logically Complete

Definition: The set **{AND, OR, NOT}** is **logically complete** because, as we showed,

any Boolean logic function on any number of inputs can be produced using only AND, OR, and NOT.

To show that another set is logically complete

- You need not construct arbitrary functions.
- You need only show how to construct AND, OR, and NOT.

Why Do You Care? Abstraction!

Imagine working on a new device technology.

- Maybe it's based on DNA.
- Maybe it's based on new semiconductors.
- Maybe it's based on carbon nanotubes.
- Maybe you're still finishing your degree?!

What do you need to be able to build in order to replace the current technology?

AND, OR, and NOT.

Other people can then build higher layers of abstraction!

Example: 3-input XOR

Let's build XOR as an example.

First, write the truth table.

What function produces this row? **A'B'C**

And this row? **A'BC'**

And this one? **AB'C'**

And this one? **ABC**

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

3-input XOR Expressed Using AND, OR, and NOT

Putting these four functions together, we obtain:

A XOR B XOR C =
A'B'C + A'BC' + AB'C' + ABC

Now we are ready to begin building devices such as adders and comparators to manipulate our representations…