Last Time

• AND/OR/NOT

• DeMorgan’s Law
The most important logical operations are **and**, **or**, and **not**.

- **x and y** is True only if **both** x and y are True.
- **x or y** is True only if **either** x or y are True.
- **not x** is True only if x is False.

A lot like their English meanings, but unambiguous.
Relating And/Or/Not

- **Note**: not (not $x$) = $x$
- **DeMorgan’s Law**: not flips ands and ors
  - not ($x$ and $y$) = (not $x$) or (not $y$)
  - not ($x$ or $y$) = (not $x$) and (not $y$)
DeMorgan’s TV Schedule

• $x$: The show airs on Tuesdays
• $y$: The show airs on Mondays.
• not ($x$ or $y$)
• The show does not air on Mondays and the show does not air on Tuesdays.
• The show does not air on Mondays or Tuesdays.
Clicker Questions
Multiple Switches

A = False

B = True

C = True
D = False
Multiple Switches

Switches A and B are wired in parallel: either will light the bulb.
Switches A and B are wired in parallel: either will light the bulb.

Switches C and D are wired in series: both are needed to light the bulb.
Multiple Switches

Switches A and B are wired in parallel: either will light the bulb.

Switches C and D are wired in series: both are needed to light the bulb.
Multiple Switches

Switches A and B are wired in parallel: either will light the bulb.

Switches C and D are wired in series: both are needed to light the bulb.

light1 = A or B

light2 = A and B
Special switches allow a single mechanical switch to control two circuits simultaneously.
Multiple Circuits

Special switches allow a single mechanical switch to control two circuits simultaneously.
Multiple Circuits

- **Light 1:** \(A \text{ or } B\)
- **Light 2:** \(A \text{ and } B\)

Special switches allow a single mechanical switch to control two circuits simultaneously.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

- There's a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

• There’s a pile of objects, say 10.
• On her turn, a player can take away either one or two objects.
• Players alternate.
• The player to take the last object wins.
miniNim

• There’s a pile of objects, say 10.
• On her turn, a player can take away either one or two objects.
• Players alternate.
• The player to take the last object wins.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

- There’s a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
miniNim

• There’s a pile of objects, say 10.
• On her turn, a player can take away either one or two objects.
• Players alternate.
• The player to take the last object wins.
miniNim

- There's a pile of objects, say 10.
- On her turn, a player can take away either one or two objects.
- Players alternate.
- The player to take the last object wins.
Nim5Bot: Game Tree

- Let’s start by considering the 5-object version.
- We’ll design a strategy for the computer “C” to beat the user “U”.
Further Considerations

- To win miniNim: if possible, remove pieces to leave opponent with a multiple of 3.
- Why does it work? We win if opponent has 3; if opponent has a multiple of 3, can leave her with next smaller multiple multiple of 3.
- What if goal is to not take the last object?
- What if we can take 1, 2, or 3 objects per round? 2 or 3? 1 or 3? Is there a general rule?
Complete Nim5 Logic

• fiveLeft = True
• threeLeft = takeOne_1
• twoLeft = takeTwo_1
• C-Win = (threeLeft and (takeOne_2 or takeTwo_2)) or (twoLeft and takeOne_2)
• U-Win = twoLeft and takeTwo_2
Nim5 Circuit

http://scratch.mit.edu/projects/cs105/35736
What a Headache!

- Encoding Nim10 with switches is a nightmare. Why?
  - Since some switches are used in multiple places, needs more than a double-throw switch.
  - Since values are reused, hard to keep track of different circuits.
  - Appears to need a separate circuit for each output: Gets too complex too fast.
Bits, Inside and Out

Inputs: Switches

A = False
B = True

Outputs: Lights

light1On = False
light2On = True
Bits, Inside and Out

Inputs: Switches

A = False
B = True

Outputs: Lights

Inside: Logic

AND
AND
AND
AND

OR
OR
OR
OR

NOT
NOT

light2On = True
light1On = False
Need a New Approach

• Let’s consider an alternate way of building “and” and “or” logic.
  • Makes simple things more complex.
  • Makes complex things much simpler!
• That’s a tradeoff we can deal with.
Electricity Activated Switches

- Easy to make “and” and “or” out of switches, but we need to make switches that other switches can switch!
Before, we used switches to control the flow of current; now, we will use current (via electromagnetism) to control switches (which control the flow of current)!
Relay Circuit: A=F, B=F

A = False

B = False

lightOn = False
Relay Circuit: A=F, B=T

A = False

B = True

lightOn = False
Relay Circuit: A=T, B=F

A = True

B = False

lightOn = False
Relay Circuit: A = T, B = T

A = True

B = True

lightOn = True
### What Is This Thing?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

---

**Diagram:**

- **A**
- **B**
- **C**

Power source
What Is This Thing?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

A. “or” gate  
B. “and” gate  
C. “not” gate  
D. none of these
We saw several slides ago that a single relay “inverts” its input signal, turning a True to a False and vice versa.

These output summaries are known as “truth tables”.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Abstraction: The Black Box

- Our new relay-based "and" and "not" gates take current, not switches, as input.
- As a result, they are easier to chain together.
- The original switch-based scheme did not support this kind of modularity.
## A Third Black Box

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>2</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

The diagram shows a logic circuit with a truth table:

- **A** and **B** as inputs.
- **AND GATE** as a component.
- **NOT GATE** as output components.
- **C** as the output.
A Third Black Box

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

AND GATE

NOT GATE

NOT GATE

NOT GATE
A Third Black Box

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

\[
\text{AND GATE} \quad \text{NOT GATE} \quad \text{NOT GATE}
\]

\[
\begin{align*}
& \text{False} \quad A \quad \text{True} \\
& \text{True} \quad B
\end{align*}
\]

\[
\begin{align*}
& \text{True} \quad \text{NOT GATE} \\
& \text{False} \quad \text{AND GATE}
\end{align*}
\]

\[
\begin{align*}
& \text{False} \quad C
\end{align*}
\]
A Third Black Box

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

AND GATE

NOT GATE

NOT GATE

NOT GATE
A Third Black Box

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

AND GATE

NOT GATE

NOT GATE

NOT GATE

AND GATE

NOT GATE

True

False

True

False

True
It’s an “or gate”: a black box built out of other black boxes!
Simplified Nim5 Circuit

True

fiveLeft

threeLeft

takeOne₁ takeTwo₁

takeOne₂ takeTwo₂

twoLeft

AND GATE

OR GATE

OR GATE

AND GATE

AND GATE

U-Win

C-Win
How to Make a Gate

- switches/ relays
- hydraulic valves
- tinkertoys
- silicon: semiconductors/ transistors
- soap bubble
- DNA
- quantum material
- optics
- nanotubes
- neurons
- dominoes
- legos/ marbles
Movie
NOT Gate (v3)
Or Gate (v4)

Release bottom row first
Wire, transmitting bits (v1)
Could It Work?

- My domino “or” gate requires 24 dominoes.
- The first Pentium processor had 3.3M transistors, or roughly 800K gates.
- So, perhaps 19M dominoes needed.
- World record for domino toppling: 4M.
- Oh, and the Pentium did its computations 60M times a second, whereas dominoes might require a week to set up once.
But There Are Lots of Them

- iPod: 60Gb. 1Gb = one billion bytes, each of which is 8 bits.
- Format uses 128 Kbps (kilobits or 1000 bits per second of sound).
- So, 62,500 minutes of sound or 15,000 songs at 4 minutes per song.
- Screen: 320x240 pixels, each of which stores 1 byte each of R,G,B intensity (24 bits). That’s 76,800 pixels and 1.8M bits.
- At 30 frames per sec., that’s 55.3 million bits per second or 144 min. of (quiet) video.