Practice example

- You are given the responsibility of building an automatic voting machine.
  - Assume there are 2 candidates.
  - Assume there are 3 voters, everyone gets a single vote.
  - The candidate with the most votes wins.

- What logical variables would you use?

- Can you write a logical expression, which evaluates who wins (True = Candidate A, False = Candidate B)?
Practice example

• Input logical variables:
  – V1 = Vote of person 1 (True=Candidate A, False=Candidate B)
  – V2 = Vote of person 2 (True=Candidate A, False=Candidate B)
  – V3 = Vote of person 3 (True=Candidate A, False=Candidate B)

• Output logical variables
  \( \text{WINNER} = (\text{True}=\text{Candidate A wins}, \text{False}=\text{Candidate B wins}) \)

• Logical expression:
  \( \text{WINNER} = (V1 \text{ AND } V2) \text{ OR } (V1 \text{ AND } V3) \text{ OR } (V2 \text{ AND } V3) \)

How would you check if the logical expression is correct?

Checking logical expressions

• Computer must be ready for any input, and must compute correct results in all cases.

• Must go through all possible input combinations:
  - V1=True, V2=True, V3=True  \( \text{WINNER} = ? \)
  - V1=True, V2=True, V3=False  \( \text{WINNER} = ? \)
  - V1=True, V2=False, V3=True  \( \text{WINNER} = ? \)
  - V1=True, V2=False, V3=False  \( \text{WINNER} = ? \)
  - V1=False, V2=True, V3=True  \( \text{WINNER} = ? \)
  - V1=False, V2=True, V3=False  \( \text{WINNER} = ? \)
  - V1=False, V2=False, V3=True  \( \text{WINNER} = ? \)
  - V1=False, V2=False, V3=False  \( \text{WINNER} = ? \)
Truth table

• Write-up a table with all possible input combinations, and check the output the output for each row.

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>Outputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1 V2 V3</td>
<td>WINNER</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 (= Candidate B)</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 (= Candidate B)</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 (= Candidate B)</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 (= Candidate A)</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 (= Candidate B)</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 (= Candidate A)</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 (= Candidate A)</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 (= Candidate A)</td>
</tr>
</tbody>
</table>

• This is called a Truth Table.

Comparing logical expressions

• Recall our previous expression:

\[ \text{WINNER} = (V1 \text{ AND } V2) \text{ OR } (V1 \text{ AND } V3) \text{ OR } (V2 \text{ AND } V3) \]

• You can also extract the logical expression directly from the Truth Table:

\[ \text{WINNER} = (\text{NOT } V1 \text{ AND } V2 \text{ AND } V3) \text{ OR } (V1 \text{ AND } \text{NOT } V2 \text{ AND } V3) \text{ OR } (V1 \text{ AND } V2 \text{ AND } \text{NOT } V3) \text{ OR } (V1 \text{ AND } V2 \text{ AND } V3) \]
Extracting logical expression from the truth table

- Recall:

  \[ \text{WINNER} = (\neg V_1 \land V_2 \land V_3) \lor (V_1 \land (\neg V_2) \land V_3) \lor (V_1 \land V_2 \land (\neg V_3)) \lor (V_1 \land V_2 \land V_3) \]

- How do we get this logical expression:
  - Consider each line in the table.
    - If the line has OUTPUT=1, this line must be included in the logical expression as a sub-expression.
    - The sub-expression includes all variables, where true variables are included without modification and negative variables are preceded by NOT operator.
    - The variables in a sub-expression are separated by “AND” logical operators.
  - Sub-expressions are separated by “OR” logical operators.

Pros / cons of the two logical expressions

- Compare:

  E1: \[ \text{WINNER} = (V_1 \land V_2) \lor (V_1 \land V_3) \lor (V_2 \land V_3) \]

  E2: \[ \text{WINNER} = (\neg V_1 \land V_2 \land V_3) \lor (V_1 \land (\neg V_2) \land V_3) \lor (V_1 \land V_2 \land (\neg V_3)) \lor (V_1 \land V_2 \land V_3) \]

  - E1 is more compact.
  - E2 we can get directly from the truth table.
How do we implement a logical expression?

• Assume we have logic gates (or blocks) that implement each logical operator.

• Logic gates are the building blocks of digital electronics and are used to build telecommunication devices, computers, etc.

Logic gates and their truth table: AND

• Truth table for the AND operator:

<table>
<thead>
<tr>
<th>Input A:</th>
<th>Input B:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• The AND gate is usually drawn as a half-moon.
• It is a double input gate.
Logic gates and their truth table: OR

• Truth table for the OR operator:

<table>
<thead>
<tr>
<th>Input A:</th>
<th>Input B:</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• The OR gate is usually drawn as a crescent-shape.
• It is a double input gate.

Logic gates and their truth table: NOT

• Truth table for the NOT operator:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• The NOT gate is usually drawn as a triangle, with a small circle.
• The circle is there to indicate the output inverts the input.
• It is a single input gate.
Can we implement E1 and E2?

- Problem:
  - E1 needs a 3-input OR gate.
  - E2 needs a 3-input AND gate.

- Solution: Make it by grouping gates

\[
\begin{array}{c}
A \\
\downarrow \text{OR} \\
B \\
\downarrow \text{OR} \\
C \\
\text{OR} \\
\text{Output}
\end{array}
= 
\begin{array}{c}
A \\
\downarrow \text{OR} \\
B \\
\downarrow \text{OR} \\
C \\
\text{Output}
\end{array}
\]

- And similarly for AND gates.

More complicated gates: NAND

- NAND = Not AND
- This corresponds to an AND gate followed by a NOT gate.
  - Output = NOT ( A AND B )

\[
\begin{array}{c}
A \\
\downarrow \text{Output}
\end{array}
\]

Truth Table

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Output Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
More complicated gates: NOR

- NOR = Not OR
- This corresponds to an OR gate followed by a NOT gate.
  - Output = NOT ( A OR B )

```
  A  B  Output
  0  0  1
  0  1  0
  1  0  0
  1  1  0
```

More complicated gates: EX-OR

- EX-OR = EXclusive OR
- This corresponds to an OR gate, but the output is negative if both inputs are true.

```
  A  B  Output
  0  0  0
  0  1  1
  1  0  1
  1  1  0
```

Combining logic gates

- Logic gates can be combined to produce complex logical expressions.
  
  E.g.: \( \text{WINNER} = (V1 \text{ AND } V2) \text{ OR } (V1 \text{ AND } V3) \text{ OR } (V2 \text{ AND } V3) \)

- Logic gates can also be combined to substitute for another type of gate.

Example

Is there a unique set of blocks to represent a given expression? No!

(Hint: Just write out the truth table for each set of gates, and see whether they are the same.)
De Morgan’s Theorem

**Theorem**: If neither A nor B is true, then both A and B must be false.

<table>
<thead>
<tr>
<th>Logical gates</th>
<th>Logical expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (\lor) B (\rightarrow) (\neg)</td>
<td>NOT (A OR B)</td>
</tr>
<tr>
<td>A (\neg) (\rightarrow) (\land) B (\neg)</td>
<td>(NOT A) AND (NOT B)</td>
</tr>
</tbody>
</table>

How do we choose which expression to implement?

- Sometimes function can be more compact (recall E1 vs E2).
- Multiple logic gates (of one type) are placed on a single chip; it may be more efficient to use all of them, rather than require another chip (of a different type).

4001 Chip: four 2-input NOR gates
Leveraging this insight

- Any logical gate can be replaced by a set of NAND gates.

- This means all we ever need, to implement any logical expression, is a lot of NAND gates!

- Total number of gates may be larger, but total number of chips is usually smaller.

(Note: Same thing can be done with NOR gates.)

A harder problem

- Imagine you play a game of Rock-Paper-Scissors against your friend.

- Assume you want a computer to automatically decide if you win or not.

- What logical variables would you use?

- Can you write a logical expression, which evaluates whether or not you win (True = win, False = loose)?
  
  E.g. If you play Rock and your friend plays Scissor, it returns True, and similarly for other possible plays.
Rock-Paper-Scissors: Logical variables

• Input: choice of player 1, choice of player 2
• Output: outcome of the game (according to the rules)
• Need to convert input and output to binary representation.
• Need 2 variables to represent the possible choice of each player
  01 = Scissors 10 = Paper 11 = Rock
  So we need 4 variables to represent the choice of both players.
• Need 2 variables to represent the possible outcomes.
  10 = Player 1 wins 01 = Player 2 wins 00 = Tie

Rock-Paper-Scissors: Other representations

• There are other possible binary representations.

• Some are equivalent:
  – same expressive power, same number of bits
  – E.g. Scissors = 00, Paper = 01, Rock = 11

• Some are not equivalent:
  – E.g. Scissors = 0, Paper = 1, Rock = 1 (Fewer bits, less expressive power)
  – E.g. Scissors = 000, Paper = 001, Rock = 011 (Same power, but more bits)
Rock-Paper-Scissors: Truth Table

<table>
<thead>
<tr>
<th>Input logical variables:</th>
<th>Output variables:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player1: A  B</td>
<td>Player2: C  D</td>
</tr>
<tr>
<td>Scissors 0  1</td>
<td>Scissors 0  1</td>
</tr>
<tr>
<td>Scissors 0  1</td>
<td>Paper 1  0</td>
</tr>
<tr>
<td>Scissors 0  1</td>
<td>Rock 1  1</td>
</tr>
<tr>
<td>Paper 1  0</td>
<td>Scissors 0  1</td>
</tr>
<tr>
<td>Paper 1  0</td>
<td>Paper 1  0</td>
</tr>
<tr>
<td>Paper 1  0</td>
<td>Rock 1  1</td>
</tr>
<tr>
<td>Rock 1  1</td>
<td>Scissors 0  1</td>
</tr>
<tr>
<td>Rock 1  1</td>
<td>Paper 1  0</td>
</tr>
<tr>
<td>Rock 1  1</td>
<td>Rock 1  1</td>
</tr>
</tbody>
</table>

What happens to the unspecified input (e.g. 0000)? Doesn't matter what the output is!

Rock-Paper-Scissors: Logical expressions

- Need two expressions, one for each of the output bits.

\[
E = \left( (\text{NOT } A) \text{ AND } B \text{ AND } C \text{ AND } (\text{NOT } D) \right) \text{ OR } \left( (A \text{ AND } (\text{NOT } B) \text{ AND } C \text{ AND } D) \right) \text{ OR } \left( (A \text{ AND } B \text{ AND } (\text{NOT } C) \text{ AND } D) \right)
\]

\[
F = \left( (\text{NOT } A) \text{ AND } B \text{ AND } C \text{ AND } D \right) \text{ OR } \left( (A \text{ AND } (\text{NOT } B) \text{ AND } (\text{NOT } C) \text{ AND } D) \right) \text{ OR } \left( (A \text{ AND } B \text{ AND } C \text{ AND } (\text{NOT } D) \right)
\]
Rock-Paper-Scissors: Logical gates

• Final step! Try this at home.

Take-home message

• Know how to build a truth table from a logical problem description.
• Know how to extract the logical expressions from the truth table.
• Learn to identify and use the basic gates: AND, OR, NOT.
• Understand the link between truth tables and logic gates.
• Know how to use combinations of gates to implement logical expressions.
• Understand that many different sets of gates can represent a given logical expression.
• Be able to state and understand De Morgan's theorem.

Homework 2 posted. Due next Thursday.