Announcements

• Let’s use 1 for True and 0 for False
• Homework #1 has been posted
  – Submit on GradeScope
  – Did you get the email?
  – How to scan and submit
• Office Hours are in room...

• Quiz tomorrow
Arguments

Recall: An **argument** is a conjecture that says:
If you make certain assumptions, then a particular statement must follow.

• The assumptions are called **premises**
• The statement that (supposedly) follows is the **conclusion**

**Example:**

\[
\begin{align*}
& p \lor q \\
& q \rightarrow r \\
& \sim p \\
\hline
& \therefore r
\end{align*}
\]

Premises  \quad Conclusion
Validity

We say an argument is **valid** when:

Every interpretation that makes all of the premises true also makes the conclusion true.

*Not all arguments are valid!*

Is this argument valid? Let’s check.

\[
\begin{align*}
p \lor q \\
q \rightarrow r \\
\sim p \\
\hline
\therefore r
\end{align*}
\]

Premises

Conclusion
We will need this today...

<table>
<thead>
<tr>
<th></th>
<th>Given any statement variables $p$, $q$, and $r$, a tautology $t$ and a contradiction $c$, the following logical equivalences hold:</th>
</tr>
</thead>
</table>
| 1. Commutative laws: | $p \land q \equiv q \land p$  
$p \lor q \equiv q \lor p$ |
| 2. Associative laws: | $(p \land q) \land r \equiv p \land (q \land r)$  
$(p \lor q) \lor r \equiv p \lor (q \lor r)$ |
| 3. Distributive laws: | $p \land (q \lor r) \equiv (p \land q) \lor (p \land r)$  
$p \lor (q \land r) \equiv (p \lor q) \land (p \lor r)$ |
| 4. Identity laws: | $p \land t \equiv p$  
$p \lor c \equiv p$ |
| 5. Negation laws: | $p \lor \sim p \equiv t$  
$p \land \sim p \equiv c$ |
| 6. Double negative law: | $\sim(\sim p) \equiv p$ |
| 7. Idempotent laws: | $p \land p \equiv p$  
$p \lor p \equiv p$ |
| 8. DeMorgan’s laws: | $\sim(p \land q) \equiv \sim p \lor \sim q$  
$\sim(p \lor q) \equiv \sim p \land \sim q$ |
| 9. Universal bounds laws: | $p \lor t \equiv t$  
$p \land c \equiv c$ |
| 10. Absorption laws: | $p \lor (p \land q) \equiv p$  
$p \land (p \lor q) \equiv p$ |
| 11. Negations of $t$ and $c$: | $\sim t \equiv c$  
$\sim c \equiv t$ |
**Rules of Inference**

Rules of inference are short arguments that are known to be valid. We will use them to prove the validity of more complex arguments.

<table>
<thead>
<tr>
<th>Modus Ponens</th>
<th>Modus Tollens</th>
<th>Conjunction</th>
<th>Transitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \rightarrow q )</td>
<td>( p \rightarrow q )</td>
<td>( p )</td>
<td>( p \rightarrow q )</td>
</tr>
<tr>
<td>( p )</td>
<td>( \neg q )</td>
<td>( q )</td>
<td>( q \rightarrow r )</td>
</tr>
<tr>
<td>( \therefore q )</td>
<td>( \therefore \neg p )</td>
<td>( \therefore p \land q )</td>
<td>( \therefore p \rightarrow r )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elimination</th>
<th>Generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \lor q )</td>
<td>( p \lor q )</td>
</tr>
<tr>
<td>( \neg q )</td>
<td>( \neg p )</td>
</tr>
<tr>
<td>( \therefore p )</td>
<td>( \therefore q )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specialization</th>
<th>Contradiction rule</th>
<th>Proof by division into cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \land q )</td>
<td>( \neg p \rightarrow q )</td>
<td>( p \lor q )</td>
</tr>
<tr>
<td>( \therefore p )</td>
<td>( \therefore q )</td>
<td>( p \rightarrow r )</td>
</tr>
</tbody>
</table>

- You don’t need to memorize this
- Posted on class webpage (under “resources”)

5
Proof

Instead of using truth tables, we can try to \textit{prove} the validity of an argument.

For now, a \textit{proof} is a sequence of statements, beginning with the premises. Each subsequent statement must follow from the previous statements according to a valid “rule of inference” (or using one of the known equivalencies). The last statement should be the conclusion.
Practicing Formal Proofs

Let’s prove the validity of these arguments:

\[
\begin{align*}
\text{P1: } & \ p \lor q \\
\text{P2: } & \ q \rightarrow r \\
\text{P3: } & \ \neg p \\
\therefore & \ r
\end{align*}
\]

\[
\begin{align*}
\text{P1: } & \ p \land q \\
\text{P2: } & \ p \rightarrow s \\
\text{P3: } & \ \neg r \rightarrow \neg q \\
\therefore & \ s \land r
\end{align*}
\]

- Do these examples represent proofs in the “real world”?
- Are the proofs in the rest of this course going to be this tedious, mechanical, and dull?
Interesting Question

Do you think there could be a valid argument (in propositional logic) that is not provable using the Equivalence Laws and Rules of Inference that we have on our charts?
Unit 2
Digital Circuits
Number Base Review

What are number bases?

How do we convert a number from an arbitrary base into base 10?
How do we convert a number from base 10 into an arbitrary base?

We are mostly concerned with base 10 (decimal) and base 2 (binary).
Basic logic gates

Computer circuits are comprised of “logic gates”. These are physical devices which we will consider in abstract. The “inputs” and “outputs” are bits (0’s or 1’s)

• An **and** gate:

• An **or** gate:

• A **not** gate:
Digital Circuits

Circuits are formed by combining logic gates.

• How many input bits?
• How many output bits?
• What is the output when the input is 110?
Propositional Logic and Circuits

Each statement of propositional logic can be represented by a circuit with one input for each variable, and a single output bit.

Practice making circuits for these:

• $p \lor \neg(q \land r)$
• $p \leftrightarrow q$
Hardware representing Truth Tables

• Any column in a truth table can be represented with a statement of propositional logic. How?
• Now any truth table can be built from an actual circuit.

Example:

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>r</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Circuits that Calculate

Circuits can perform math!

Examples:
- Addition of integers
- Multiplication of integers
- Compute $3x^4 + 2x^2 + 7$, where $x$ is an integer
- Approximations of real-valued functions

Our goal today will be to build a circuit that can add numbers together:

**Inputs:** 77 and 49 (in binary)

**Output:** 126 (in binary)
Brute force: Addition by Truth Table

Adding 2-bit numbers:

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>+</th>
<th>Y</th>
<th>=</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 + 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 + 2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3 + 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 + 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2 + 3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2 + 2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2 + 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2 + 0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 + 3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 + 2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 + 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 + 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 + 3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 + 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 + 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 + 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Now we can build a circuit with 4 input bits and three output bits.

How big would this table be with 64-bit operands?

Is there a more elegant approach?
Addition of binary numbers

Practice:

\[
\begin{array}{cccc}
1001 & 1001 & 1011 & 1101 \\
+ 0010 & + 0011 & + 0010 & + 0111 \\
\hline
1011 & 1010 & 1101 & 1110 \\
\end{array}
\]

Can we create a circuit that models this process?
Half-Adder

Circuit that adds two bits together:

\[
\begin{array}{c}
\text{b1} \\
\text{b2}
\end{array}
\rightarrow
\begin{array}{c}
\text{AND} \\
\text{OR} \\
\text{NOT} \\
\text{AND}
\end{array}
\rightarrow
\begin{array}{c}
\text{carry bit} \\
\text{sum bit}
\end{array}
\]
Full adder

Circuit that adds three bits together:
Parallel adder (for three bit operands)

\[ X_1X_2X_3 + Y_1Y_2Y_3 = A_0A_1A_2A_3 \]

- Can be extended to add larger numbers