Lecture 9: Boolean Circuits
Where we are, where we are going

*Computer science is no more about computers than astronomy is about telescopes.*

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aug 31</td>
<td>Sep 1</td>
<td>Sep 2</td>
<td>Sep 3</td>
<td>Sep 4</td>
</tr>
<tr>
<td>Introduction</td>
<td></td>
<td></td>
<td>On proofs</td>
<td>Quiz 1</td>
</tr>
<tr>
<td>Sep 7</td>
<td>Sep 8</td>
<td>Sep 9</td>
<td>Sep 10</td>
<td>Sep 11</td>
</tr>
<tr>
<td>Finite automata</td>
<td>hw1 w.s.</td>
<td></td>
<td>Turing machines</td>
<td>Quiz 2</td>
</tr>
<tr>
<td>Sep 14</td>
<td>Sep 15</td>
<td>Sep 16</td>
<td>Sep 17</td>
<td>Sep 18</td>
</tr>
<tr>
<td>Uncountability</td>
<td>hw2 w.s.</td>
<td></td>
<td>Undecidability</td>
<td>Quiz 3</td>
</tr>
<tr>
<td>Sep 21</td>
<td>Sep 22</td>
<td>Sep 23</td>
<td>Sep 24</td>
<td>Sep 25</td>
</tr>
<tr>
<td>Intro to complexity 1</td>
<td></td>
<td>hw3 w.s.</td>
<td>Intro to complexity 2</td>
<td>Quiz 4</td>
</tr>
<tr>
<td>Sep 28</td>
<td>Sep 29</td>
<td>Sep 30</td>
<td>Oct 1</td>
<td>Oct 2</td>
</tr>
<tr>
<td>Circuit complexity</td>
<td>hw4 w.s.</td>
<td></td>
<td>Graphs 1</td>
<td>Quiz 5</td>
</tr>
<tr>
<td>Oct 5</td>
<td>Oct 6</td>
<td>Oct 7</td>
<td>Oct 8</td>
<td>Oct 9</td>
</tr>
<tr>
<td>Graphs 2</td>
<td>hw5 w.s.</td>
<td></td>
<td>Graphs 3</td>
<td>Quiz 6</td>
</tr>
<tr>
<td>Reductions</td>
<td></td>
<td>Midterm 1</td>
<td>NP-completeness</td>
<td>Quiz 7</td>
</tr>
</tbody>
</table>
P \equiv NP
P vs NP is on the horizon

Millennium Problems

Yang–Mills and Mass Gap
Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang–Mills equations. But no proof of this property is known.

Riemann Hypothesis
The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann's 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part 1/2.

P vs NP Problem
If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Navier–Stokes Equation
This is the equation which governs the flow of fluids such as water and air. However, there is no proof for the most basic questions one can ask: do solutions exist, and are they unique? Why ask for a proof? Because a proof gives not only certitude, but also understanding.

Hodge Conjecture
The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four. But in dimension four it is unknown.

Poincaré Conjecture
In 1904 the French mathematician Henri Poincaré asked if the three dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston's geometrization conjecture. Perelman’s proof tells us that every three manifold is built from a set of standard pieces, each with one of eight well-understood geometries.

Birch and Swinnerton-Dyer Conjecture
Supported by much experimental evidence, this conjecture relates the number of points on an elliptic curve mod p to the rank of the group of rational points. Elliptic curves, defined by cubic equations in two variables, are fundamental mathematical objects that arise in many areas: Wiles' proof of the Fermat Conjecture, factorization of numbers into primes, and cryptography, to name three.
Computational complexity of an algorithm

Recall:

Definition:
The **running time** of an algorithm $A$ is defined as worst-case

$$T_A(n) = \max_{\text{instances } I \text{ of size } n} \{\text{# steps } A \text{ takes on } I\}$$
Computational complexity of a problem

The intrinsic complexity of a problem:
Complexity of the best algorithm computing the problem.

How to show an upper bound on the intrinsic complexity?
> Give an algorithm that solves the problem.

How to show a lower bound on the intrinsic complexity?
> Argue against all possible algorithms that solve the problem.

The dream: Get a matching upper and lower bound.
What is $P$?

$P$

The set of languages that can be decided in $O(n^k)$ steps for some constant $k$.

The theoretical divide between efficient and inefficient:

$L \in P$ \quad \rightarrow \quad \text{efficiently solvable.}

$L \notin P$ \quad \rightarrow \quad \text{not efficiently solvable.}
### What is P?

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>Awesome!</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>Great!</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Kind of efficient.</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>Barely efficient. (???)</td>
</tr>
<tr>
<td>$O(n^5)$</td>
<td>Would not call it efficient.</td>
</tr>
<tr>
<td>$O(n^{10})$</td>
<td>Definitely <strong>not</strong> efficient!</td>
</tr>
<tr>
<td>$O(n^{100})$</td>
<td>WTF?</td>
</tr>
</tbody>
</table>
Why P?

- P is not meant to mean “efficient in practice”
- It means “You have done something extraordinarily better than brute force (exhaustive) search.”
- So P is about mathematical insight into a problem’s structure.
- Robust to notion of what is an elementary step, what model we use, reasonable encoding of input, implementation details.
- Wouldn’t make sense to cut it off at some specific exponent.
- Plus, big exponents don’t really arise.
- If it does arise, usually can be brought down.
Summary: Being in P vs not being in P is a qualitative difference, not a quantitative one.
What is NP?

The set of languages that can be decided in $O(k^n)$ steps for some constant $k > 1$.

NP: A class between P and EXP.
What is NP?

$P \subseteq NP$ asks whether these two sets are equal.

How would you show $P = NP$?

Show that every problem in NP can be solved in poly-time.

How would you show $P \neq NP$?

Show that there is a problem in NP which cannot be solved in poly-time.
Boolean Circuits
Some preliminary questions

What is a Boolean circuit?

- It is a computational model for computing decision problems (or computational problems).

We already have TMs. Why Boolean circuits?

- The definition is simpler.
- Easier to understand, usually easier to reason about.
- Boolean circuits can efficiently simulate TMs. (efficient decider TM $\implies$ efficient/small circuits.)
- Circuits are good models to study parallel computation.
- Real computers are built with digital circuits.
Sounds awesome!
So why didn’t we just learn about circuits first?

There is a small catch.

An algorithm is a **finite** answer to **infinite** number of questions.

Stephen Kleene
(1909 - 1994)
Sounds awesome!
So why didn’t we just learn about circuits first?

There is a small catch.

Circuits are an infinite answer to infinite number of questions.
Dividing a problem according to length of input

\[ \Sigma = \{0, 1\} \]

\[ L \subseteq \{0, 1\}^* \]

\[ L_n = \{w \in L : |w| = n\} \]

\[ L = L_0 \cup L_1 \cup L_2 \cup \cdots \]

\[ f : \{0, 1\}^* \to \{0, 1\} \]

\[ \{0, 1\}^n = \text{all strings of length } n \]

\[ f^n : \{0, 1\}^n \to \{0, 1\} \]

for \( x \in \{0, 1\}^n \),

\[ f^n(x) = f(x) \]

\[ f = (f^0, f^1, f^2, \ldots) \]
Dividing a problem according to length of input

A TM is a finite object (finite number of states) but can handle any input length.

Imagine a model where we allow the TM to grow with input length.

```
<table>
<thead>
<tr>
<th>TM</th>
<th>input →TM→ output</th>
</tr>
</thead>
<tbody>
<tr>
<td>TM₀</td>
<td>computes ( L₀ )</td>
</tr>
<tr>
<td>TM₁</td>
<td>( L₁ )</td>
</tr>
<tr>
<td>TM₂</td>
<td>( L₂ )</td>
</tr>
<tr>
<td>TM₃</td>
<td>( L₃ )</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
```
So one machine does not compute $L$.

You use a family of machines:

$$(M_0, M_1, M_2, \ldots)$$

(Imagine having a different Python function for each input length.)

Is this a reasonable/realistic model of computation?

Boolean circuits work this way. Need a separate circuit for each input length.
Picture of a circuit
Picture of a circuit
Picture of a circuit

- **binary OR gate**
- **binary AND gate**
- **unary NOT gate**
- **input gate**
- **output gate**
Picture of a circuit

- **wires**
- **binary OR gate**
- **binary AND gate**
- **unary NOT gate**
- **input gate**
- **output gate**
Information flows from input gates to the output gate.

No feedback loops allowed!
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \).

So how does it compute \( f(x_1, x_2, \ldots, x_n) \)?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \).

So how does it compute \( f(x_1, x_2, \ldots, x_n) \)?
Computes a function $f : \{0, 1\}^n \to \{0, 1\}$.
So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \to \{0, 1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function  \( f : \{0, 1\}^n \rightarrow \{0, 1\} \).

So how does it compute  \( f(x_1, x_2, \ldots, x_n) \) ?
Poll: What does this circuit compute?

(sometimes circuits are drawn upside down)
Poll: What does this circuit compute?

(sometimes circuits are drawn upside down)
Poll: What does this circuit compute?

(sometimes circuits are drawn upside down)

Parity of $x_1 + x_2$

$x_1 \oplus x_2$

Parity of $x_3 + x_4$

$x_3 \oplus x_4$

$x_1 \oplus x_2 \oplus x_3 \oplus x_4$
How does a circuit decide/compute a language?

How do we measure the complexity of a circuit?
How can a circuit compute a language?

A circuit has a fixed number of inputs.

How can we compute/decide a decision problem 
\( f : \{0, 1\}^* \rightarrow \{0, 1\} \) with circuits?

\[
f = (f^0, f^1, f^2, \ldots) \quad \text{where} \quad f^n : \{0, 1\}^n \rightarrow \{0, 1\}
\]

Construct a circuit for each input length.

A circuit family \( C \) is a collection of circuits \((C_0, C_1, C_2, \ldots)\)

where each \( C_n \) takes \( n \) input variables.
How can a circuit compute a language?

A circuit has a fixed number of inputs.

How can we compute/decide a decision problem $f : \{0, 1\}^* \rightarrow \{0, 1\}$ with circuits?

$$f = (f^0, f^1, f^2, \ldots) \quad \text{where} \quad f^n : \{0, 1\}^n \rightarrow \{0, 1\}$$

A circuit family $C$ is a collection of circuits $(C_0, C_1, C_2, \ldots)$

where each $C_n$ takes $n$ input variables.

We say that a circuit family $C$ decides/computes $f : \{0, 1\}^* \rightarrow \{0, 1\}$ if $C_n$ computes $f^n$ for every $n$. 
Definition: [size of a circuit]
The **size of a circuit** is the total number of gates (counting the input variables as gates too) in the circuit.

Definition: [size of a circuit family]
The **size of a circuit family** $C = (C_0, C_1, C_2, \ldots)$ is a function $s(\cdot)$ such that $s(n)$ is the size of $C_n$.

Definition: [circuit complexity]
The **circuit complexity** of a decision problem is the size of the minimal circuit family that decides it.

(This is the intrinsic complexity with respect to circuit size)
Let \( f : \{0, 1\}^* \to \{0, 1\} \) be the parity decision problem.

\[
f(x) = x_1 + \ldots + x_n \mod 2 \quad \text{(where } n = |x|)\]

\[
f(x) = x_1 \oplus \cdots \oplus x_n\]

What is the circuit complexity of this function?

Choose the tightest one:

\[
\begin{align*}
O(n) & \quad \text{vs} \quad O(2^n) \\
O(n^2) & \quad \text{vs} \quad O(2^{2^n}) \\
O(n^{2.5}) & \quad \text{vs} \quad O(2^{\text{STACK}(n)}) \\
\text{None of the above.} & \quad \text{vs} \quad \text{Beats me.}
\end{align*}
\]
\[ s(n) = 2s\left(\frac{n}{2}\right) + 5 \]
\[ s(1) = 1 \]
\[ \implies s(n) = \Theta(n). \]
The big picture

Computability with respect to circuits

**Theorem:** Any decision problem \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).
Limits of efficient computability with respect to circuits

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/4n$.

In fact, **most** decision problems require exponential size.
The big picture

Circuits can efficiently “simulate” TMs

**Theorem:** Let \( f : \{0, 1\}^* \to \{0, 1\} \) be a decision problem which can be decided in time \( O(T(n)) \). Then it can be computed by a circuit family of size \( O(T(n)^2) \).

\[ \text{poly-time TM} \quad \iff \quad \text{poly-size circuits} \]

\[ \text{no poly-size circuits} \quad \implies \quad \text{no poly-time TM} \]
Circuits can efficiently “simulate” TMs

To show $P \neq NP$:

Find $h$ in NP whose circuit complexity is more than $\text{poly}(n)$. 
The big picture

So we can just work with circuits instead

This is awesome in 2 ways:


2. Restrict the circuit. Make it less powerful. e.g. (i) restrict depth (ii) restrict types of gates
The big picture

So we can just work with circuits instead

Exciting progress was made in the 1980s.

People thought $P \neq NP$ would be proved soon.

Alas…

After 60 years of research, best lower bound on circuit size for an explicit function:

$$5n \quad \text{— peanuts}$$
The big picture

**Theorem:** Any decision problem \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least \( 2^n / 4n \).

**Theorem:** Let \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) be a decision problem which can be decided in time \( O(T(n)) \). Then it can be computed by a circuit family of size \( O(T(n)^2) \).
Alan Turing
(1912 - 1954)
A small break
Theorem 1: Max circuit size of a function

Theorem: Any decision problem \( f : \{0, 1\}^* \to \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).

Proof:

Goal:
construct a circuit of size \( O(2^n) \) for \( f^n : \{0, 1\}^n \to \{0, 1\} \).

Observation:

\[
f^n(x_1, x_2, \ldots, x_n) = (x_1 \land f^n(1, x_2, \ldots, x_n)) \lor
(\neg x_1 \land f^n(0, x_2, \ldots, x_n))
\]
Theorem 1: Max circuit size of a function

Theorem: Any decision problem \( f : \{0, 1\}^* \to \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).

Proof:

Goal: construct a circuit of size \( O(2^n) \) for \( f^n : \{0, 1\}^n \to \{0, 1\} \).

Observation:

\[
f^n(x_1, x_2, \ldots, x_n) = (x_1 \land f^n(1, x_2, \ldots, x_n)) \lor (\neg x_1 \land f^n(0, x_2, \ldots, x_n))
\]

if \( x_1 = 1 \)
Theorem 1: Max circuit size of a function

**Theorem:** Any decision problem \( f : \{0, 1\}^* \rightarrow \{0, 1\} \)
can be computed by a circuit family of size \( O(2^n) \).

**Proof:**

**Goal:**
construct a circuit of size \( O(2^n) \) for \( f^n : \{0, 1\}^n \rightarrow \{0, 1\} \).

**Observation:**
\[
f^n(x_1, x_2, \ldots, x_n) = \begin{cases} 
0 & \text{if } x_1 = 0 \\
(\neg x_1 \land f^n(0, x_2, \ldots, x_n)) \lor (x_1 \land f^n(1, x_2, \ldots, x_n)) & \text{otherwise}
\end{cases}
\]
Theorem 1: Max circuit size of a function

Proof (continued):

\[ s(n) = \text{max size of a circuit computing } n\text{-variable function} \]

\[ s(n) \leq 2s(n - 1) + 5, \quad s(1) \leq 3 \quad \implies \quad s(n) = O(2^n) \]
How many different functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ are there?

- $n$
- $2n$
- $n^2$
- $2^n$
- $2^{2^n}$
- $2^{STACK(n)}$
- none of the above
- beats me
Theorem 2: Some functions are hard

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/4n$.

**Proof:**

Want to show: there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that cannot be computed by a circuit of size $< 2^n/4n$.

**Observation:** # possible functions is $2^{2^n}$.

**Claim 1:** # circuits of size at most $s$ is $\leq 2^{4s \log s}$.

**Claim 2:** For $s \leq 2^n/4n$, $2^{4s \log s} < 2^{2^n}$.

# circuits < # functions
Theorem 2: Some functions are hard

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/4n$.

**Proof:**

Want to show: there is a function $f : \{0, 1\}^n \to \{0, 1\}$ that cannot be computed by a circuit of size $< 2^n/4n$.

**Observation:** # possible functions is $2^{2n}$.

**Claim 1:** # circuits of size at most $s$ is $\leq 2^{4s \log s}$.

**Claim 2:** For $s \leq 2^n/4n$, $2^{4s \log s} < 2^{2^n}$.

We are done once we prove Claim 1. (Claim 2 is super easy.)
Theorem 2: Some functions are hard

Proof (continued):

Claim 1: # circuits of size at most $s$ is $\leq 2^{4s \log s}$.

Proof of claim:

Recall $|A| \leq |B|$ iff $B \rightarrow A$.

Let $A = \{\text{circuits of size at most } s\}$

$B = \{0, 1\}^{4s \log s} \quad |B| = 2^{4s \log s}$

To show $B \rightarrow A$:

encode a circuit with a binary string of length $4s \log s$.

(just like the CS method)
Theorem 2: Some functions are hard

Proof (continued):

Claim 1: # circuits of size at most $s$ is $\leq 2^{4s \log s}$.

Proof of claim (continued):

Encoding a circuit with a binary string of length $4s \log s$:

Number the gates: 1, 2, 3, 4, …, $s$

For each gate in the circuit, write down:
- type of the gate (2 bits)
- from which gates the inputs are coming from (2 log $s$ bits)

Total: $s(2 + 2 \log s)$ bits

$(2s + 2s \log s)$ bits $\leq (4s \log s)$ bits
Theorem 2: Some functions are hard

That was due to Claude Shannon (1949).

Father of *Information Theory*.

A non-constructive argument.

In fact, it is easy to show that most functions require exponential size circuits.
Theorem 3: Circuits can simulate TMs

**Theorem:** Let $f : \{0, 1\}^* \to \{0, 1\}$ be a decision problem which can be decided in time $O(T(n))$. Then it can be computed by a circuit family of size $O(T(n)^2)$.

How can you prove such a theorem?

If you like a challenge, try to prove it yourself.

If you don’t like a challenge, but still curious, see the course notes for a sketch of the proof.

If you don’t like a challenge, and are not curious, 😞 you can ignore the proof.
P \equiv NP