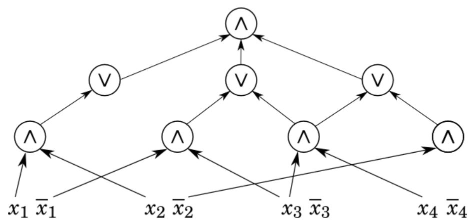
15-251
Great Ideas in
Theoretical Computer Science

Lecture 15:
Boolean Circuits



October 17th, 2017

Where we are, where we are going

Sep 11	Sep 12 ***Turing machines 1***	Sep 13 hw2 w.s.	Sep 14 ***Turing machines 2***	Sep 15
Sep 18	Sep 19 ***Uncountability***	Sep 20 hw3 w.s.	Sep 21 ***Undecidability***	Sep 22
Sep 25	Sep 26 ***Time complexity***	Sep 27 hw4 w.s.	Sep 28 ***Cake cutting***	Sep 29
Oct 2	Oct 3 ***Graphs 1***	Oct 4 hw5 w.s.	Oct 5 ***Graphs 2***	Oct 6
Oct 9	Oct 10 ***Graphs 3***	Oct 11 **Midterm 1**	Oct 12 ***Graphs 4***	Oct 13
Oct 16	Oct 17 ***Boolean circuits***	Oct 18 hw6 w.s.	Oct 19 ***NP 1***	Oct 20 MID-SEM. BREAK
Oct 23	Oct 24 ***NP 2***	Oct 25 hw7 w.s.	Oct 26 ***NP 3***	Oct 27

What is P ?

P

The theoretical divide between efficient and inefficient:

$L \in \mathrm{P} \longrightarrow$ efficiently solvable.

$L \notin \mathrm{P} \longrightarrow$ not efficiently solvable.

What is efficient in theory and in practice ?

In practice:

$O(n)$

$O(n \log n)$

$O(n^2)$

$O(n^3)$

$O(n^5)$

$O(n^{10})$

$O(n^{100})$

What is efficient in theory and in practice ?

In theory:

Polynomial time

Otherwise

What is efficient in theory and in practice ?

- Poly-time is **not** meant to mean "efficient in practice".
- Poly-time: extraordinarily better than brute force search.
- Poly-time: mathematical insight into problem's structure.
- Robust to notion of what is an
  elementary step, *what model we use*,
  reasonable encoding of input, *implementation details*.
- Nice closure property: Plug in a poly-time alg. into another poly-time alg. —> poly-time

What is efficient in theory and in practice ?

Brute-Force Algorithm: Exponential time

what we care about most in 15-251

usually the "magic" happens here

Algorithmic Breakthrough: Polynomial time

what we care about more in 15-451

Blood, sweat, and tears: Linear time

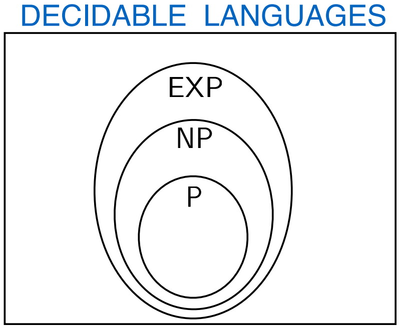
What is efficient in theory and in practice ?

Summary: Poly-time vs not poly-time
is a *qualitative* difference, not a *quantitative* one.

What is NP ?

EXP

DECIDABLE LANGUAGES

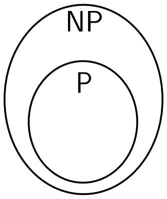


NP:

A class/set between P and EXP.

(set of languages we would love to solve efficiently)

What is NP ?



$\mathsf{P} \stackrel{?}{=} \mathsf{NP}$

asks whether these two sets are equal.

How would you show $\mathsf{P} = \mathsf{NP}$?

How would you show $\mathsf{P} \neq \mathsf{NP}$?

Boolean circuits are related to the P vs NP question in multiple ways.

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 $\Longrightarrow$ efficient/small circuits.)
- Circuits are good models to study *parallel computation*.
- Real computers are built with digital circuits.

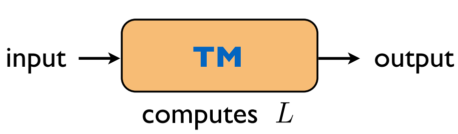
Dividing a problem according to length of input

$$\Sigma = \{0,1\}$$

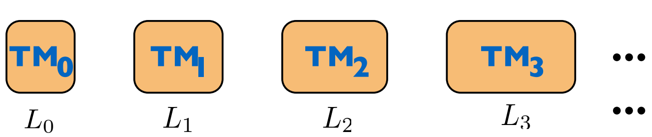
$L \subseteq \{0,1\}^*$	$f : \{0,1\}^* \to \{0,1\}$
$L_n = \{w \in L : \lvert w\rvert = n\}$	$\{0,1\}^n$ = all strings of length n $f^n : \{0,1\}^n \to \{0,1\}$ for $x \in \{0,1\}^n$, $f^n(x) = f(x)$
$L = L_0 \cup L_1 \cup L_2 \cup \cdots$	$f = (f^0, f^1, f^2, \ldots)$

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 change with input length.



Dividing a problem according to length of input

So one machine does not compute L.

You use a family of machines:

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

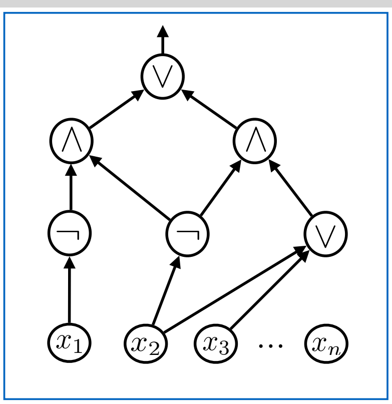
(Imagine having a different Python function for each input length.)

Is this a reasonable/realistic model of computation?!?

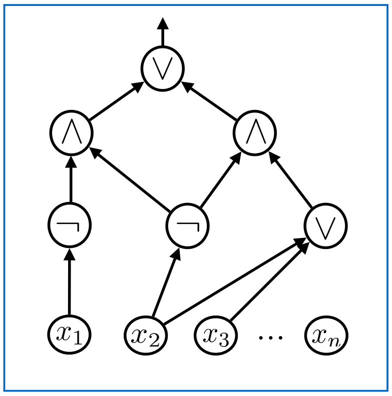
Boolean circuits work this way.
Need a separate circuit for each input length.
(but we still love them)

Boolean Circuit Definition

Picture of a circuit

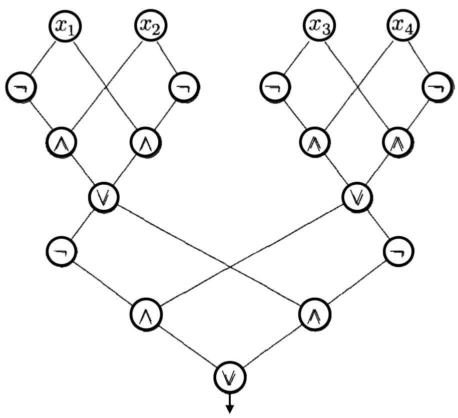


Picture of a circuit



Poll 1: What does this circuit compute ?

(sometimes circuits are drawn upside down)



How does a circuit decide a language?

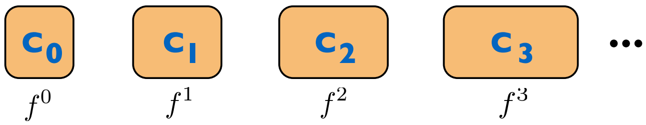
How do we measure the complexity of a circuit?

How can a circuit compute a language?

Given $f : \{0,1\}^* \to \{0,1\}$, write

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

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?

Circuit size and complexity

Definition (size of a circuit):

Definition (size of a circuit family):

Definition (circuit complexity):

Poll 2

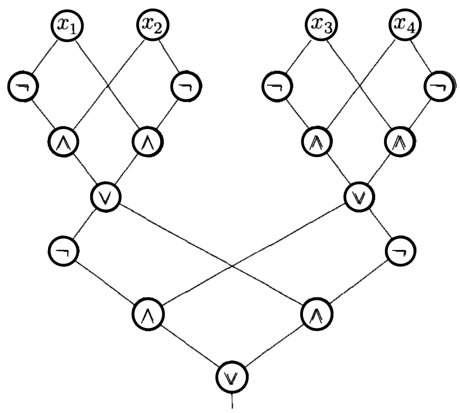
Let $f : \{0,1\}^* \to \{0,1\}$ be the parity decision problem.

$$f(x) = x_1 + \ldots + x_n \mod 2 \qquad (\text{where } n = |x|)$$

$$f(x) = x_1 \oplus \cdots \oplus x_n$$

What is the circuit complexity of this function?

Poll 2



The Big Picture Regarding Boolean Circuits

The big picture

Computability with respect to circuits

Theorem 1:

A universal exponential upper bound for all decision problems.

(We know this is not true in the TM model.)

The big picture

Limits of efficient computability with respect to circuits

Theorem 2 (Shannon's Theorem):

The big picture

Circuits can efficiently "simulate" TMs

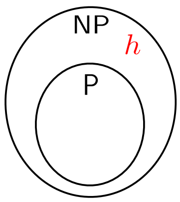
Theorem 3:

poly-time TM $\Longrightarrow$ poly-size circuits

Consequence of Theorem 3

poly-time TM $\Longrightarrow$ poly-size circuits

no poly-size circuits $\Longrightarrow$ no poly-time TM



To show P $\neq$ NP :

Find h in NP whose circuit complexity is more than any n^k.

Consequence of Theorem 3

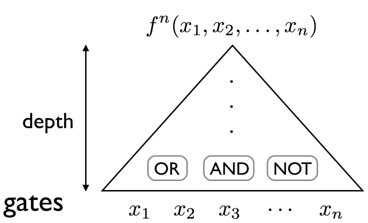
So we can just work with circuits instead

<u>This is awesome in 2 ways:</u>

1. Circuits: clean and simple definition of computation. "Just" a composition of AND, OR, NOT gates.

2. Restrict the circuit.
Make it less powerful.

e.g. (i) restrict *depth*
(ii) restrict types of gates



Poll 3

How many different functions $f : \{0,1\}^n \to \{0,1\}$ are there?

- n
- $2n$
- n^2
- 2^n
- 2^{2^n}
- none of the above
- beats me

Proof of Theorem 2

Theorem 2: Some functions are hard

Theorem: There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

Proof:

Theorem 2: Some functions are hard

Proof (continued):

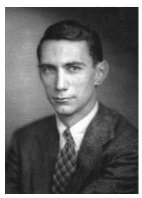
Theorem 2: Some functions are hard

Proof (continued):

Theorem 2: Some functions are hard

That was due to Claude Shannon (1949).

Father of *Information Theory*.



Claude Shannon
(1916 - 2001)

A non-constructive argument.

In fact, it is easy to show that **almost all** functions require exponential size circuits.

Concluding Remarks

Boolean circuits: another model of computation.
(arguably simpler definition, easier to reason about)

no poly-size circuits $\Longrightarrow$ no poly-time TM
(can attack P vs NP problem with circuits)

CIRCUIT-SAT decision problem:

Given as input the description of a circuit,
output True if the circuit is "satisfiable".

Whether CIRCUIT-SAT is in P or not
is intimately related to the P vs NP question!