CAS CS 535: Complexity Theory

Lecturer: Mark Bun Fall 2023

Lecture Notes 12: Circuits, Non-Uniform Computation

Reading.

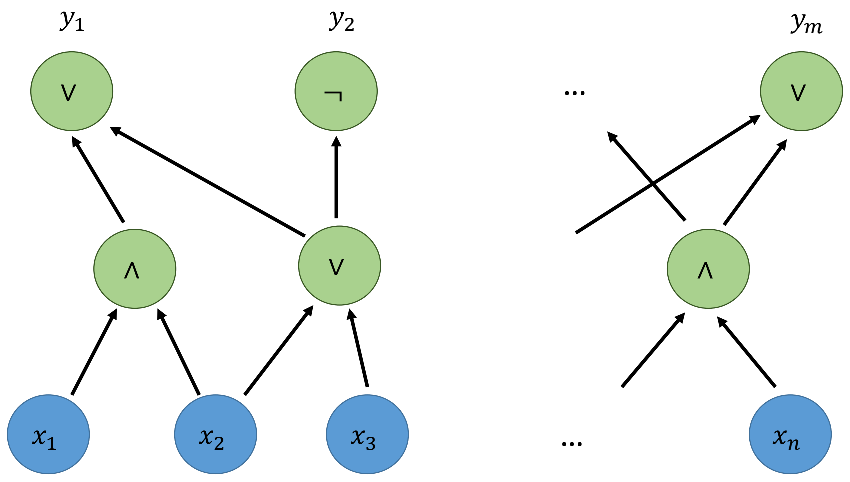
- Arora-Barak § 6.1-6.3

Last time: Alternation, Time-Space Tradeoffs for SAT

1 Boolean Circuits

A Boolean circuit is a directed, acyclic graph with

- n sources representing inputs,
- m sinks representing outputs,
- non-input vertices ("gates") labeled by $\vee$, $\wedge$, or $\neg$,
- fan-in (in-degree) of 1 or 2 on gates.



To evaluate a circuit on an input $x \in \{0,1\}^n$, evaluate the intermediate gates recursively until values are derived at the output gates.

A circuit defines a function $C_n : \{0,1\}^n \to \{0,1\}^m$. We'll be interested in using circuits to decide languages, but we immediately run into a syntatic mismatch. A circuit C_n only defines a function on Boolean strings of length exactly n, but a language defines a computational problem over inputs from $\{0,1\}^*$ that could have arbitrary length. The resolution is to study infinite families of circuits, one to handle every input length individually.

Definition 1. A circuit family is an infinite sequence of circuits $C = \{C_n\}_{n=1}^{\infty}$.

We say that C decides a language L if for every n and every $x \in \{0,1\}^n$, we have $x \in L \iff C_n(x) = 1$.

The size of a circuit, denoted $|C_n|$, is the number of vertices. Its depth is the length of the longest directed path from an input vertex to an output vertex. Both size and depth can be taken as measures of the complexity of a circuit, with size playing a role similar to time complexity on a TM, and depth corresponding to time on a parallel computer.

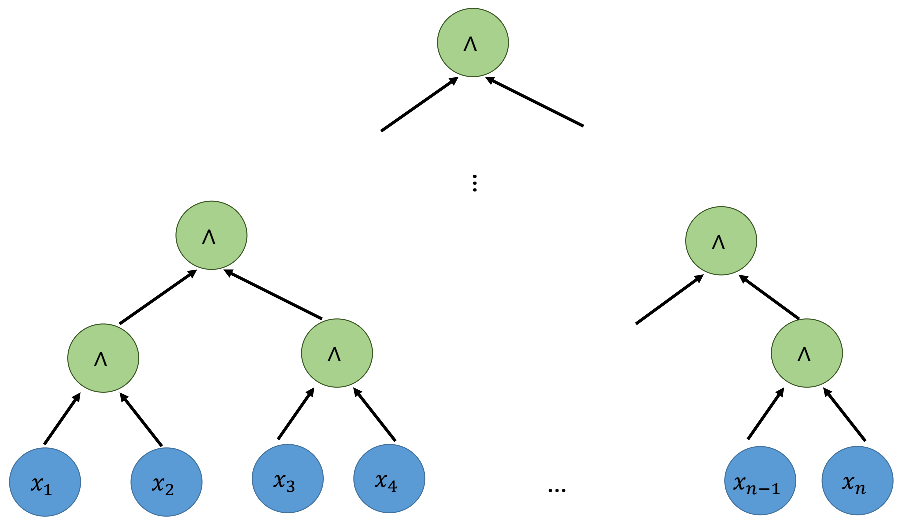
Definition 2 (Circuit size classes)**.** A circuit family such that $C = \{C_n\}_{n=1}^{\infty}$ has size $T(n)$ if $|C_n| \le T(n)$ for every n. The class $\mathbf{SIZE}(T(n))$ consists of languages L that are decidable by $T(n)$-size circuit families. The class $\mathbf{P_{/poly}} = \cup_{c=1}^{\infty}\mathbf{SIZE}(n^c)$ consists of languages that are decidable by polynomial-size circuits.

Some motivation for studying circuits:

- Circuits more closely model computer hardware (silicon chips) than TMs, allowing for the more realistic study of problems on concrete input lengths. They are also useful for modeling parallel computation.
- It's often easier to reason about circuits than it is to reason about TMs, but proving things about circuits can help us prove things about TMs. For example, much of the power of the Cook-Levin Theorem comes from how it translates questions about arbitrary NTMs into questions about CNF formulas (a restricted class of circuits). We'll see more examples of this later.
- There's a close connection between circuit complexity and oracle complexity, based on fruitful analogies between TM classes and classes of circuits (e.g., $\mathbf{NP} \approx$ DNF, $\mathbf{coNP} \approx$ CNF, $\mathbf{PH} \approx \mathbf{AC}^0$). Much of what we know about oracle classes comes from circuit complexity. And much of what we know about circuit complexity comes from asking questions about oracles.

Here are a few helpful examples for getting to know how to deal with circuits.

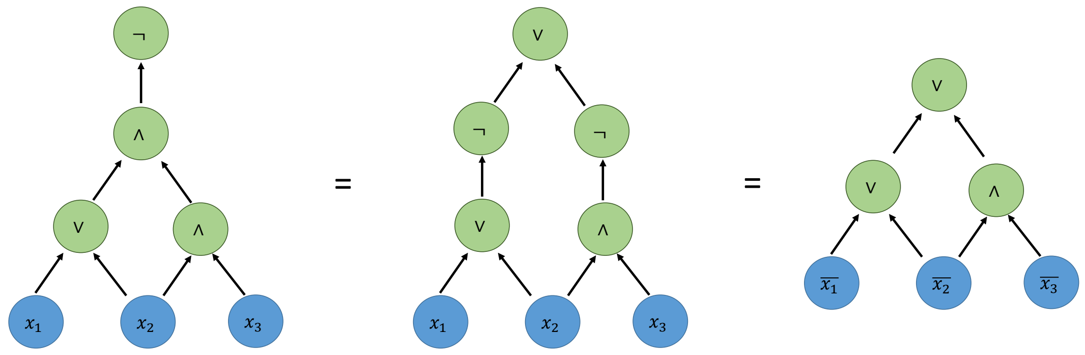
Example 3. Consider the n-variable $\mathrm{AND}_n : \{0,1\}^n \to \{0,1\}$ function. We can implement as a circuit with fan-in 2 gates using a binary tree of size n and depth $\log n$:



Thus, if you care only about circuit size up to polynomial factors, or circuit depth up to polylogarithmic factors, you can assume for simplicity that the n-input functions AND_n and OR_n are computed by a single fan-in n gate.

However: If you want to study finer gradations between classes of circuits, for instance by examining levels of the **NC** hierarchy of polylogarithmic-depth circuits, then it's important to take care that your AND and OR gates have fan-in 2.

Example 4. We've defined Boolean circuits to allow for arbitrary interlacing between ANDs, ORs, and NOTs. But using de Morgan's rules, we can "push all negations to the bottom" in an arbitrary circuit. So we can assume without loss of generality that all negations appear at the bottom level, or equivalently, that a Boolean circuit is just an AND/OR circuit over the set of literals $x_1, \overline{x_1}, \dots, x_n, \overline{x_n}$.



Example 5. Examples 3 and 4 above illustrate Boolean formulas, where each intermediate gate has fan-out only 1.

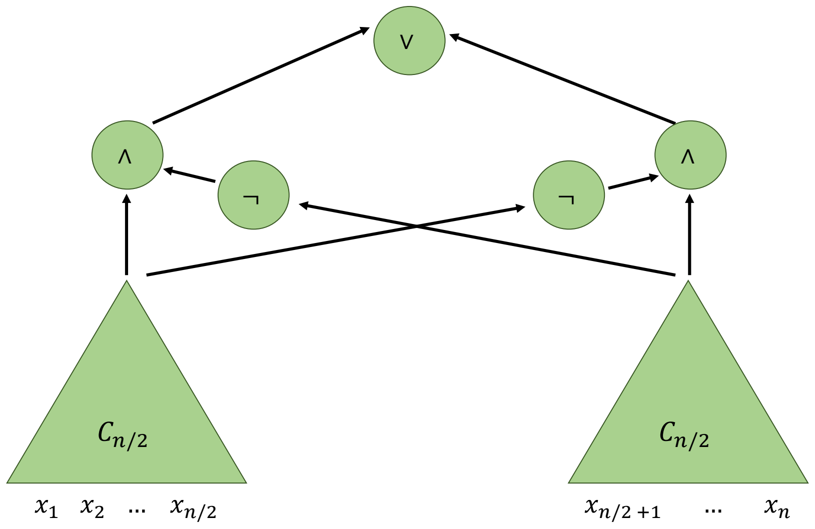
But general Boolean circuits are more powerful in that intermediate gates can have fan-out ≥ 2, which enables the reuse of intermediate comptuations. For example, consider computing the Parity (or XOR) function. On two bits, we have

$$x \oplus y = (x \wedge \overline{y}) \vee (\overline{x} \wedge y).$$

Thus, we can build a circuit C_n for Parity on n bits out of subcircuits $C_{n/2}$ for Parity on $n/2$ bits:

$$C_n(x_1,\ldots,x_n) = (C_{n/2}(x_1,\ldots,x_{n/2}) \wedge \neg C_{n/2}(x_{n/2+1},\ldots,x_n) \vee (\neg C_{n/2}(x_1,\ldots,x_{n/2}) \wedge C_{n/2}(x_{n/2+1},\ldots,x_n)).$$

This gives us the recurrence $|C_n| = 5 + |C_{n/2}| + |C_{n/2}|$, which resolves to $|C_n| = O(n)$.



On the other hand, one can show that every formula computing Parity requires size $\Omega(n^2)$.

Note that an inverted tree of fan-out 2 gates lets one build a gate of arbitrary fan-out. This increases circuit size by a constant factor, and depth by a logarithmic factor. So while there's potentially a big difference between fan-out 1 and fan-out 2, there's not much of a difference (unless one wants to study circuit depth) between fan-out 2 and arbitrary fan-out.

2 Circuits vs. Turing Machines

Circuit families are at least as powerful as TMs in that TMs can be compiled into circuits with low overhead in the compilation.

Theorem 6. *Let M be a TM running in time $T(n)$ where $\log T(n)$ is a space-constructible function. Then there exists a circuit family $C = \{C_n\}_{n=1}^{\infty}$ such that:*

1. *For every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, we have $C_n(x) = M(x)$.*
2. *For every n, we have $|C_n| \leq \mathrm{poly}(T(n))$.*
3. *Uniformity: There exists a TM N running in time $\mathrm{poly}(T(n))$ and space $O(\log T(n))$ that prints descriptions of the circuits, i.e., $N(1^n) = \lfloor C_n \rfloor$.*

Proof idea. Once again, this uses similar ideas to the proof of the Cook-Levin Theorem. The TM M accepts an input x of length n if there exists a sequence of $T(n)$ configurations $K_1, K_2, \ldots, K_{T(n)}$ such that K_1 is the start configuration, $K_{T(n)}$ is an accepting configuration, and it is possible to get from each K_i to K_{i+1} in one step of computation. These can be checked by replicating a constant (depending on M, of course) sized gadget across constant-sized windows of adjacent configurations. □

Corollary 7. $\mathbf{P} \subseteq \mathbf{P}_{/\mathbf{poly}}$.

How about the reverse containment? Is $\mathbf{P}_{/\mathbf{poly}}$ contained in $\mathbf{P}$? The answer is no, since (awkward terminology) $\mathbf{P}_{/\mathbf{poly}}$ can decide undecidable languages.

Fact 8. *Every unary language $L \subseteq \{1\}^*$ is in $\mathbf{P}_{/\mathbf{poly}}$.*

Proof. Each circuit C_n is just a constant indicating whether 1^n is in L. □

Fact 9. *There exists an undecidable unary language.*

Proof. Take your favorite undecidable language, e.g., the Halting problem, and encode it in unary. □

2.1 Uniformity and Non-uniformity

The key conceptual difference between Turing machines and circuits is that the former is a uniform model of computation (i.e., the same algorithm is used on inputs of every length), whereas the latter is a non-uniform model (i.e., a different algorithm is used on each input length.) In a sense, this is the only difference, as efficient TMs are equivalent to uniformly-generated circuit families, while arbitrary circuit families are equivalent to TMs that take "non-uniform advice."

Theorem 10. *A language $L \in \mathbf{P}$ if and only if there exists a logspace uniform circuit family computing L. (That is, there exists a logspace TM that on input 1^n outputs the description of the n'th circuit in the family.)*

Proof. The "only if" direction follows immediately from Theorem 6. For the "if" direction, suppose L is decidable by a circuit family generated by logspace TM N. Then the following poly-time TM decides L:

On input x:

1. Generate circuit $C_n = N(1^{|x|})$
2. Evaluate $C_n(x)$.

Logspace uniformity guarantees that C_n takes logspace, and hence polynomial time, to construct. Moreover, since the circuit has polynomial size, it takes polynomial time to evaluate. □

Theorem 11. *A language $L \in \mathbf{P}_{/\mathbf{poly}}$ if and only if L is decided in poly-time by a TM taking "polynomial advice." That is, there exists a TM M, a polynomial p, and an infinite sequence of strings $\{\alpha_n\}_{n=1}^{\infty}$, with $\alpha_n \in \{0,1\}^{p(n)}$ such that $x \in L \iff M(x, \alpha_{|x|}) = 1$.*

Proof. For the "only if" direction, let $L \in \mathbf{P}_{/\mathbf{poly}}$ be computed by a poly-size circuit family $\{C_n\}$. Define an advice sequence by $\alpha_n = \lfloor C_n \rfloor$. Then the following TM M with advice $\{\alpha_n\}$ decides L in poly-time:

On input x, α_n:

1. Construct circuit C_n corresponding to α_n
2. Evaluate $C_n(x)$.

For the "if" direction, let L be decided by a poly-time TM M using advice $\{\alpha_n\}$. For each n, use the transformation described in the proof of Theorem 6 on $M(\cdot, \alpha_n)$ to obtain a poly-size circuit family. □

2.2 Non-Uniform Advice vs. Witnesses/Certificates

Non-uniform advice bears a tantalizing similarity to the witnesses/certificates used to characterize $\mathbf{NP}$ languages. But the order of quantifiers makes a big difference! Specifically, a machine M decides a language L in each model if:

Non-uniform advice: $\exists\{\alpha_n\}_{n=1}^{\infty}\forall x \in \{0,1\}^* \qquad x \in L \iff M(x, \alpha_n) = 1$

Certificates: $\forall x \in \{0,1\}^* \qquad x \in L \iff \exists w\, M(x, w) = 1.$

That is, with non-uniform advice, the same advice string α_n has to work for all strings x of length n, whereas an NP certificate is allowed to depend on the particular instance x. At first glance, this may make non-uniform advice appear less powerful than NP certificates. But the upshot of the quantifier switch is that the same advice string equally helps certify non-membership in L. So the resources really do seem to be incomparable.

Next time, we'll study the Karp-Lipton Theorem, which tells us more about the relationship between these resources.