Theory of Computing	**Lecture Notes**
CS 6810, Spring 2026	**Feb 26 (Thursday)**
Lecturer: Eshan Chattopadhyay	
Scribe: Ramiro Deo-Campo Vuong	

Boolean Circuits and P/poly

In this lecture, we continue our discussion of Boolean circuits. We review Boolean circuit definitions, the basic properties of circuits, and how we measure the complexity of circuits (size and depth). We introduce the P/poly class, which contains languages decidable by Boolean circuits of polynomial size. We compare P/poly to the classes P and NP. Finally, we discuss how Boolean circuits may be useful for solving the P vs. NP problem; circuit size upper and lower bounds are provided for the problem of expressing functions of the form $f : \{0,1\}^n \to \{0,1\}$.

1 Boolean Circuit Review

Definition 1 (Boolean Circuit)**.** *A Boolean circuit C_n is a labeled directed acyclic graph (DAG). There are n source nodes (in-degree 0 nodes) labeled with input bits, and sink nodes (out-degree 0 nodes) which define the output of the circuit. Each internal node is associated with a Boolean operation. The circuit is computed by processing nodes in a topological order. Specifically, internal nodes are processed by applying their respective Boolean operation to the bits on input wires.*

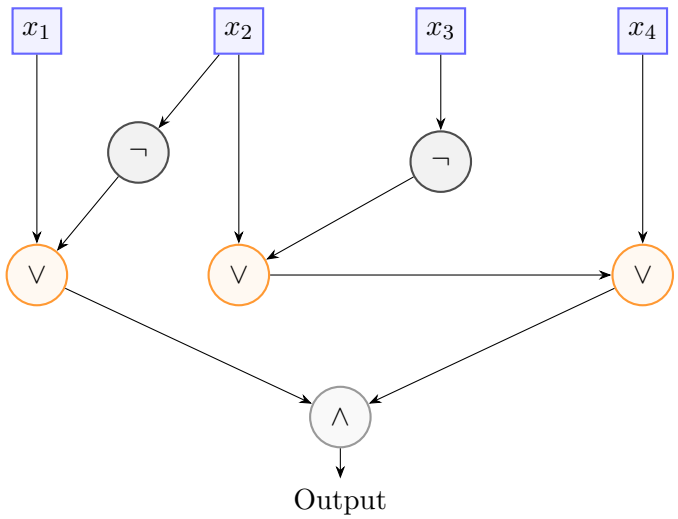

Figure 1: Depicted is a circuit encoding the Boolean formula $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee x_4)$.

Throughout class, we restrict ourselves to certain circuits. We assume circuits have fan-in 2: each node has an in-degree of at most 2. Further, we restrict internal nodes to be labeled with negation $\neg$, Boolean OR $\vee$, or Boolean AND $\wedge$.

Unlike Turing machines, each circuit has a fixed number of input bits. For this reason, a single circuit that has a fixed input length cannot decide a language L, as the language can contain strings of variable length. We compensate for fixed input lengths by considering a family of circuits. In general, models of computation that require different "programs" for inputs of different sizes are called non-uniform.

Definition 2 (Circuits and Language Decidability)**.** *A family of circuits $\{C_n\}_{n\geq 0}$ decides a language $L \subseteq \{0,1\}^*$ if C_n decides $L \cap \{0,1\}^n$ for every $n \geq 0$.*

Often, we denote a circuit family as C. When applying the appropriately sized circuit in family C to an input string x, we write $C(x)$.

There are two main ways to measure the complexity of a single circuit C_n. The first measure is size $\text{SIZE}(C_n)$, which equals the number of nodes in the circuit[1]. The second way to measure complexity is circuit depth, or the maximum length of a path from an input node to an output node. Since decidability involves a family of circuits, we also characterize the complexity of families of circuits. Below, we define the size of a family of circuits. Depth is similarly defined for circuit families.

Definition 3 (Circuit Family Size)**.** *A circuit family $\{C_n\}_{n\geq 0}$ is said to have size $s(n)$ if $\text{SIZE}(C_n) \leq s(n)$ for all $n \geq 0$.*

A notable circuit property is that they can be uniquely represented as a bit string. This is done by simply representing the circuit's underlying DAG and node labels as bits.

Observation 1 (Encoding Circuits as Bits)**.** *A circuit C of size s can be uniquely encoded as a bit string of length at most $cs\log(s)$ bits for some constant c.*

Proof. Each of the s nodes has at most two incoming edges. The neighbor on each edge can be specified using $\lceil \log s \rceil$ bits. Finally, each node label requires a constant number of bits since there are a constant number of labels: input, output, and one of three Boolean operations. This constitutes an encoding with $c \cdot s\log(s)$ bits for some constant $c > 0$. □

2 The P/poly Complexity Class

The complexity class analogous to P for circuits is called P/poly. This class contains languages that are decidable by a circuit family of polynomial size.

Definition 4 (Language Circuit Complexity)**.** *Overloading notation, the set $\text{SIZE}(s(n))$ contains languages $L \subseteq \{0,1\}^*$ which are decidable by some circuit family $\{C_n\}_{n\geq 0}$ of size $O(s(n))$.*

Definition 5 (P/poly)**.** *The complexity class* P/poly *contains languages decidable by circuit families of polynomial size:* $\mathsf{P/poly} = \bigcup_{c\geq 0} \text{SIZE}(n^c)$.

P/poly can alternatively be defined using Turing machines with advice. Although this equivalent representation of P/poly was not thoroughly discussed in lecture, supplementary details are provided in Section 2.2.

2.1 P/poly and Other Complexity Classes

In this section, we relate P/poly to other complexity classes discussed in lecture. Theorem 1 shows that P/poly contains P. Interestingly, P/poly is not contained by NP and likely does not contain NP. Theorem 2 proves that P/poly contains a variant of the Halting problem which is undecidable by both Turing machines and non-deterministic Turing machines; this shows that $\mathsf{P/poly} \not\subseteq \mathsf{NP}$. On the other hand, P/poly likely does not contain NP due to Karp-Lipton (Theorem 3), which states that $\mathsf{P/poly} \supseteq \mathsf{NP}$ would imply that the polynomial time hierarchy collapses to level 2.

Theorem 1 (P vs. P/poly)**.** P/poly *contains the class P.*

Proof. The Cook-Levin Theorem implies this theorem.

Consider a language $L \in P$ which is decidable by a Turing machine M running in time at most cn^k for constants c, k. Notably, the machine M visits at most cn^k different bits of each tape. Fix a feasible

[1]The size of a circuit is linear in its edge count, as we assumed a fan-in of at most 2.

configuration of M at a given time t. The configuration includes the state of M, the at most n^k bits stored in the work tapes, and the at most cn^k possible positions of each tape head. This configuration can be encoded as a bit string of size $O(n^k)$.

We use a circuit to simulate the Turing machine M. For each configuration bit and time t, we construct an internal node representing said bit at time t. The number of such internal nodes will be the number of bits in a configuration multiplied by the number of iterations performed: $O(n^{2k})$. To compute the values of these internal configuration nodes, we use the arguments of Cook-Levin. Given the $O(n^k)$ configuration bits associated with time t as input, Cook-Levin constructs a $O(\text{poly}(n^k))$-sized circuit to compute all the configuration bits associated with time $t+1$. The circuit to simulate M is a composition of the $O(n^k)$ circuits used to compute the configuration bits of an iteration, given the configuration bits of the previous iteration. Since there are $O(n^k)$ iterations and each circuit in the composition has size $O(\text{poly}(n^k))$, the circuit simulating M will have size $O(\text{poly}(n^k))$. □

Theorem 2 ($\mathsf{P/poly}$ and Undecidability)**.** *There is a language $L \in \mathsf{P/poly}$ that is undecidable by Turing machines.*

Proof. We show that any unary language $U \subseteq \{1^n \mid n \geq 0\}$ is in $\mathsf{P/poly}$. For n with $1^n \in U$, the circuit $x_1 \wedge \cdots \wedge x_n$ suffices. For n with $1^n \notin U$, we use the circuit that always outputs 0 (FALSE). The defined circuits constitute a circuit family of linear size. Thus, $U \in \mathsf{P/poly}$ for any unary language U.

Unary languages can encode the Halting problem. There are countably many Turing machines M and countably many bit strings x. Thus, there is a bijection from machine input pairs (M, x) to integers. The unary language $U = \{1^n \mid (M, x) \text{ corresponding to } n \text{ halts}\}$ contains an encoding of all Turing machines and input pairs that reach the HALT state. This language is undecidable by both Turing machines and non-deterministic Turing machines, but it is decidable by circuit families. □

Theorem 3 (Karp-Lipton)**.** *If $\mathsf{NP} \subseteq \mathsf{P/poly}$, then the polynomial time hierarchy collapses to level 2.*

Proof. Recall that $\Sigma_2 \subseteq \Pi_2$ or $\Pi_2 \subseteq \Sigma_2$ implies the collapse of the polynomial time hierarchy to the second level: $\mathsf{PH} = \Sigma_2 = \Pi_2$. The proof is completed by showing that $\Pi_2 \subseteq \Sigma_2$.

We use the Π_2-SAT problem, which is Π_2-complete; the goal of Π_2-SAT is to decide if $\forall u \exists v\, \phi(u, v)$ outputs TRUE for a given Boolean formula ϕ. If Π_2-SAT is in Σ_2, then $\Pi_2 \subseteq \Sigma_2$ as desired.

Define the language $L = \{(\phi, u) \mid \exists v\, \phi(u, v)\}$. Observe that $L \in \mathsf{NP}$ since a non-deterministic Turing machine can simulate $\phi(u, v)$ for all v in polynomial time. Further, by the assumption that $\mathsf{NP} \subseteq \mathsf{P/poly}$, there is a circuit family C of polynomial size that computes L.

Since there is a polynomial-sized circuit deciding L, there is a polynomial-sized circuit C' yielding $C'(\phi, u) = v$ when there is v yielding $\phi(u, v) = 1$. We prove this fact using self-reduction (search to decision reduction). Given (ϕ, u) as input, the circuit C' constructs a new formula ϕ' by assigning a single bit of v to be 1. Then the circuit invokes $C(\phi', u)$. In the case that circuit C outputs 1, the circuit is recursively applied to formula ϕ'. Otherwise, the circuit constructs formula ϕ'' with the specified bit of v assigned 0 instead of 1 and is recursively applied to ϕ''. The circuit C' has polynomial size since it makes $|v|$ different calls to C.

Consider the following problem in Σ_2: for a given ϕ decide $\exists w$ of size $\text{poly}(|\phi|)$ with $\forall u\, \phi(u, C_w(\phi, u))$, where C_w is the circuit encoded by w. The proof is completed by showing that ϕ is a TRUE instance of the above problem if and only if ϕ is a TRUE instance of Π_2-SAT (i.e., the problems are equivalent). For the backward implication, when $\forall u \exists v\, \phi(u, v)$ is True, there is a polynomial sized circuit $C'(\phi, u) = v$ with $v \in L$. Taking w with $C_w = C'$ satisfies $|w| = O(\text{poly}(|\phi|))$ and yields $\forall \phi(u, C_w(\phi, u)) = \text{TRUE}$. For the forward implication, assume there is a polynomial size w yielding $\forall u\, \phi(u, C_w(\phi, u))$. Then the circuit $C' = C_w$ computes v certifying $(\phi, u) \in L$, which implies ϕ is a TRUE instance of Π_2-SAT. Thus, Π_2-SAT is in Σ_2 which gives $\Pi_2 \subseteq \Sigma_2$ as desired. □

2.2 Turing Machines with Advice and P/poly

The relation between Turing machines with advice and P/poly was not extensively covered in the lecture. This section discusses this relation using information from `https://en.wikipedia.org/wiki/P/poly`. As the material in this section is not required, we provide informal definitions and proof sketches.

The class P/poly can also be characterized as the set of languages decidable by Turing machines in polynomial time, when the Turing machine is given advice dependent on the input length. Specifically, $L \in \mathsf{P/poly}$ when a Turing machine running in polynomial time can decide $x \in L$ with $n = |x|$, when equipped with advice α_n that satisfies $|\alpha_n| = O(\text{poly}(n))$. The advice is provided to the Turing machine in a special advice tape.

The proof requires showing that Turing machines with advice can be computed by a circuit family and vice versa. The forward direction is a result of the Cook-Levin theorem. As is done in Theorem 1, we can simulate a Turing machine with a circuit. The advice can be accounted for by hard-coding some of the input nodes of the circuit to be the bits of the advice string. In the reverse direction, the advice given to the Turing machine is the binary description of the circuit with the relevant input size. Equipped with a description of said circuit, a Turing machine can simulate the circuit in polynomial time.

3 Circuits and the P vs. NP Problem

One approach to resolving P vs. NP would be showing that there is an NP problem which requires exponentially sized circuit families. This would establish that $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, which combined with $\mathsf{P} \subseteq \mathsf{P/poly}$ from Theorem 1, would imply $\mathsf{P} \neq \mathsf{NP}$.

Open Question 1 (Circuit Size Lower Bounds)**.** *Is there a language $L \in \mathsf{NP}$ for which all circuit families deciding L have size at least $n^{\omega(1)}$?*

The current state-of-the-art lower bound shows that we require at least $(3.1 \cdot n)$-sized circuit families to solve NP problems. In this lecture, we consider the much simpler and related problem of characterizing the circuit size required to encode Boolean functions on n inputs. Provided below are two size upper bounds and a lower bound.

We prove two circuit size upper bounds. The first upper bound is a simpler argument, but does not provide as tight guarantees.

Lemma 1 (Circuit Size Upper Bounds Version 1)**.** *Any function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a $O(n \cdot 2^n)$-sized circuit. Therefore, all $L \in \{0,1\}^*$ satisfy $L \in \textsc{Size}(n \cdot 2^n)$.*

Proof. We can represent any function f as a CNF formula, which can be encoded as a Boolean circuit. Each clause in the CNF corresponds to an input string x on which $f(x) = 1$. The literals of the clause are the bit assignments of the input string corresponding to the clause. Since there are at most 2^n different input strings x yielding $f(x)$ (clauses) with n bits (literals) each, the circuit encoding the CNF formula has size $O(n \cdot 2^n)$. □

Lemma 2 (Circuit Size Upper Bounds Version 1)**.** *Any function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a $O\left(\frac{2^n}{n}\right)$-sized circuit. Therefore, all $L \in \{0,1\}^*$ satisfy $L \in \textsc{Size}\left(\frac{2^n}{n}\right)$.*

Proof. We use the following recurrence to build a circuit:

$$f(x_1, \ldots, x_n) = (f(x_1, \ldots, x_{n-1}, 0) \wedge \neg x_n) \vee (f(x_1, \ldots, x_{n-1}, 1) \wedge x_n)$$

The above recurrence uses $s(n) = 2 \cdot s(n-1) + O(1)$ different operations. Naive application of the recurrence gives a circuit size of $O(2^n)$.

To improve the above bound, consider performing the recurrence until there are k unassigned variables. The number of functions with k input bits is 2^{2^k}. Using the above recurrence, we can construct a circuit of size $O(2^k)$ for each of the 2^{2^k} functions on k unassigned variables. We wire these 2^{2^k} circuits to the relevant functions in our recurrence. The initial recursion to assign all but k variables requires a circuit of size $O(2^{n-k})$. Together with the circuits computing each of the 2^{2^k} functions on k variables, we find the entire circuit has size $O\left(2^{n-k} + 2^k \cdot 2^{2^k}\right)$. Setting $k = \log(n) - 1$ yields:

$$\begin{aligned}
O\left(2^{n-k} + 2^k \cdot 2^{2^k}\right) &= O\left(2^{n-\log(n)+1} + 2^{\log(n)-1} \cdot 2^{2^{\log(n)-1}}\right) \\
&\leq O\left(\frac{2^{n+1}}{2^{\log(n)}} + n \cdot 2^{\left(\frac{2^{\log(n)}}{2}\right)}\right) \\
&= O\left(2 \cdot \frac{2^n}{n} + n \cdot 2^{n/2}\right) \\
&= O\left(\frac{2^n}{n}\right)
\end{aligned}$$

□

Finally, we provide a circuit size lower bound matching the upper bound of Lemma 2.

Lemma 3 (Circuit Size Lower Bounds)**.** *There is a function $f : \{0,1\}^n \to \{0,1\}$ such that f cannot be computed by any circuit of size $\delta\frac{2^n}{n}$ for sufficiently small $\delta > 0$.*

Proof. The proof is a simple counting argument. There are 2^{2^n} possible Boolean functions f on n inputs. Consider the number of circuits of size s. Observation 1 says that we can uniquely encode all circuits of size s using $cs\log(s)$ bits for some constant $c > 0$. Since there are $2^{cs\log(s)}$ strings of length $cs\log(s)$, there are at most $2^{cs\log(s)}$ circuits of size s.

Now we show that $s = \delta\frac{2^n}{n}$ is insufficient for expressing the 2^{2^n} different formulas, for some constant δ. We take the log of both the number of circuits and the number of functions to simplify expressions.

$$\begin{aligned}
\log(\text{Circuits of size } s) &= \log(2^{cs\log(s)}) \\
&= cs\log s \\
&= c\left(\delta\frac{2^n}{n}\right)\log\left(\delta\frac{2^n}{n}\right) \\
&= c\delta \cdot 2^n\left(\frac{n + \log(\delta) - \log(n)}{n}\right) \\
&\leq c\delta \cdot 2^n && \text{For } n \geq \delta
\end{aligned}$$

On the other hand, the log of the number of functions is $\log(2^{2^n}) = 2^n$. Picking $\delta > 0$ sufficiently small means there are too few circuits to represent all functions. Notably, the above implies that a random function requires a large circuit. □