**CS 221 Computational Complexity, Lecture 3** Jan 30, 2018

# Circuits, Formulas, and Branching Programs

*Instructor: Madhu Sudan* *Scribe: Alec Sun*

## 1 Overview

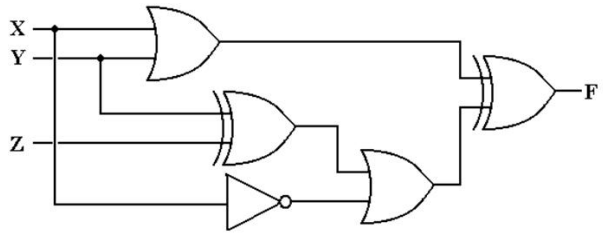
Today we will discuss circuits (definition, classes, and comparisons), other models (such as formula and branching program), the counting argument, and the Neciporuk bound.

## 2 Uniformity

The idea behind uniformity is that there is a fixed algorithm that is finite in size and that solves problems of all sizes. This is an amazing fact, and Turing was the first person to realize that you can have such finite descriptions.

However, we will also consider non-uniform models. Consider any language $L$ in $\{0,1\}^*$ and take subsets $L_n = L \cap \{0,1\}^n$ corresponding to size $n$. For each $n$ we will allow a different algorithm that solves it for inputs of length $n$, and we want the running time of such algorithm to be polynomial in $n$. The motivating question is when there exists small descriptions of algorithms for each $n$.

## 3 Circuits




A circuit is a collection of Boolean gates, each of which produces 1 output. The output can be used and reused as inputs to other Boolean gates. We require the circuit to be a directed acyclic graph (DAG). There will be $n$ designated input nodes labeled $x_1, \ldots, x_n$ with in-degree 0, and $m$ output nodes $y_1, \ldots, y_n$ with out-degree 0. All other nodes are called gates, labeled by elements in our circuit basis. One example of a universal basis is $\{AND, OR, NOT\}$. There is a natural associated function $f : \{0,1\}^n \to \{0,1\}^m$ to each circuit $C$, and we say that the circuit $C$ computes the function $f$.

## 3.1 Circuit Size and Depth

**Definition 1.** *Let $SIZE(C)$ denote the number of gates of a circuit $C$ and $DEPTH(C)$ to denote the length of the longest path in a circuit $C$.*

The longest-path problem in a DAG can be solved in polynomial time. We can also introduce the notions of the size and depth of a function.

**Definition 2.** *Let $SIZE(f)$ and $DEPTH(f)$ denote the minimum size and depth of a circuit $C$ that computes $f$.*

Note that $SIZE$ depends on the choice of universal basis. For any function there is a circuit with depth 2 that computes that function by writing the function as a 3-CNF expression and using the circuit corresponding. However, we are interested in the minimum depth of a polynomial-sized circuit that computes a function.

## 3.2 $\mathbf{P}$/poly

Let $\mathbf{P}$/poly denote the complexity class of polynomial sized circuits; that is, for every $n$ there is a polynomial sized circuit that computes the restriction of $f$ to size $n$ inputs. A language $L \in \mathbf{P}$/poly if there exists a polynomial $p$ and a circuit $C_n$ for every $n$ such that $C_n$ decides $L_n$ and $SIZE(C_n) \leq p(n)$. The size of a circuit is proxy for running time, with some exceptions. For example, $\mathbf{P}$/poly contains uncomputable functions. For example, consider the unary halting problem

$$L = \{1^n \mid n \text{ is an encoding of a Turing machine that halts on itself}\}.$$

Then for each input size $n$, either the all-0 or all-1 circuit computes the answer.

**Exercise 3.** *How does changing the basis affect $SIZE$?*

Functions in $\mathbf{P}$/poly are computable in polynomial time with poly$(n)$ bits of advice because we can hardwire the advice in for each circuit. In the case of the unary halting problem this advice is simply the answer. There is a complexity class $\mathbf{NP}/\log n$ is the class of non-deterministic polynomial time functions with $\log n$ advice.

Note that $P \subseteq \mathbf{P}$/poly because $TIME(t(n)) \subseteq SIZE(t(n) \log t(n))$. We can basically unravel all the computation into a circuit. The overarching goal of complexity theory is to prove that $\mathbf{P} \neq \mathbf{NP}$, but that is not the goal of this course. If it was, then we would all fail. One would hope that $\mathbf{NP} \not\subseteq \mathbf{P}$/poly.

**Theorem 4** (Karp-Lipton)**.** *If $\mathbf{NP} \subseteq \mathbf{P}$/poly then this is almost like $\mathbf{NP} = P$, or the weaker $\mathbf{NP} = \mathbf{coNP}$.*

**Exercise 5.** *Show that $TIME(t) \subseteq SIZE(t^2)$.*

### 3.3 Bounds on Circuit Size

**Theorem 6** (Iwama, 2002)**.** *There exists an explicit function $f : \{0,1\}^n \to \{0,1\}$ that requires $5n - o(n)$ gates in $\{AND, OR, NOT\}$ to compute.*

**Theorem 7** (Find, 2016)**.** *There exists an explicit function $f : \{0,1\}^n \to \{0,1\}$ that requires at least $3.01n$ gates in any basis.*

The above two theorems seem to sharply contrast with the following theorem.

**Theorem 8.** *There exists a function $f$ that requires $SIZE(\Omega\left(\frac{2^n}{n}\right))$ gates to compute.*

The proof is done using a counting argument. Namely, we find $s$ for which the number of functions on $n$ bits, $2^{2^n}$, is greater than the number of circuits of size $s$. Then we can conclude that there exists a function for which $SIZE(f) \geq s + 1$. Now let us count the number of circuits of size $s$. We know that this at most the number of DAGs of size $s$ times the number of gate labellings of such a DAG. To encode the DAG, we will specify the (at most 2) parents of each node, of which there are $s$ choices for each. Hence the number of DAGs is at most $(s^2)^s$ and the number of labeling is at most $3^s$. We conclude that the number of circuits with size $s$ is $2^{O(s \log s)}$. If $s < \frac{2^n}{n}$, then $s \log s < 2^n$, so we are done.

**Observation 9.** *We can summarize our discussion with the following paradigm: almost every function is very hard to compute, except the ones we know.*

The above proof does not show that there is an **NP** function $f$ such that $SIZE(f) \geq \exp(n)$. We also cannot prove that there exists $f : \{0,1\}^n \to \{0,1\}^n$ such that $f \in \mathbf{P}/\text{poly}$ but $f^{-1} \notin \mathbf{P}/\text{poly}$. These are both open problems.

# 4 Formula

Consider the expression

$$(x^2 - y^2 + 3)(x^2 - y^2 + 2)(x^2 - y^2 - 1).$$

To compute such an expression, a human would probably optimize the computation by first computing $x^2 - y^2$ and then plugging the result in. Circuits allow for such optimization. However, formulas do not allow reusing computations. The only thing you can do is to expand the formula.

We formally define a formula as a circuit where our DAG is a tree. This means that no value is being used twice or more, which captures the notion that we can no longer reuse computation.

**Definition 10.** *We can analogously define $FORMULASIZE(f)$ and $FORMULADEPTH(f)$ the same way we did for $SIZE(f)$ and $DEPTH(f)$.*

We know that

$$FORMULADEPTH(f) = O(\log FORMULASIZE(f)).$$

This follows because our DAG is a tree with at most 2 children for every node, so we have

$$FORMULASIZE(f) \leq 2^{FORMULADEPTH(f)}.$$

Hence we conclude that these two quantities are tightly related, so we can focus on $FORMULASIZE$ alone. Clearly

$$SIZE(f) \leq FORMULASIZE(f)$$

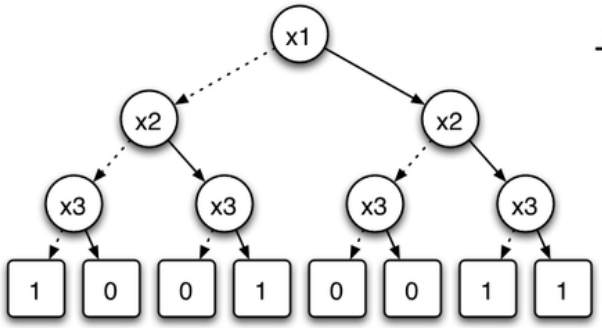
since the first is a minimum over all DAGs and the second is a minimum over all DAGs which are trees, a subset of the first. There is a state-of-the-art theorem that says that a $\tilde{O}(n^3)$ separation exists.

# 5 Branching Programs

We will introduce another size function that is roughly intermediate between the two we have just discussed,

$$SIZE(f) \leq BPSIZE(f) \leq FORMULASIZE(f).$$

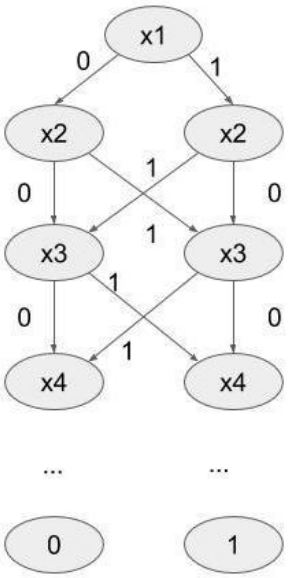
Consider a decision tree to compute a function $f$. Every function $f$ has a decision tree of depth at most $n$ and hence size at most $2^n$ by querying each bit.




| x1 | x2 | x3 | f |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

However, decision trees can certainly be optimized.

**Observation 11.** *As an example, consider the problem that sums up the input bits and outputs the result modulo 2. We can construct a decision tree that has linear size as follows:*




A decision tree is known as a branching program. A branching program is a DAG that has 1 start node and 2 output nodes with out-degree 0, labelled 0 and 1. Each non-output node is labeled with one of $x_1, x_2, \ldots, x_n$ and each node has 2 outgoing edges labeled 0 and 1.

**Definition 12.** *We define $SIZE(BP)$ as the number of nodes. The quantity $\log SIZE(BP)$ is a proxy for memory of the computation.*

**Exercise 13.** *Show that $SIZE(f) \leq O(BPSIZE(f)) \leq O(FORMULASIZE(f))$.*

**Exercise 14.** *Find small sized circuit, branching program, and formula DAGs for the MAJORITY function, which takes in $n$ bits as input and outputs which of 0 and 1 constitute the majority of the input bits.*

## 5.1 A Lower Bound on $BPSIZE$

Consider this problem which we will prove a $BPSIZE$ lower bound on. Let an array of $kn$ bits $y = [y_1, y_2, \ldots, y_n]$ where each $y_i$ has $k$ bits. Consider the problem $D_{n,k}$ of determining if there exists $i_1 \neq i_2$ such that $y_{i_1} = y_{i_2}$.

**Theorem 15.** *If $k \geq 2 \log n$, then $BPSIZE(D_{n,k}) = \Omega(n^2)$.*

*Proof.* Use $-i$ to denote all of the $y_i$ blocks except for $i$. Take the restriction of the original problem where all blocks besides $y_i$ have distinct values. Suppose that we have fixed the values of all the other bits except in the block $y_i$. Then

$$D_{n,k}(y_i, y_{-i}) = f_{y_{-i}}(y_i),$$

where $f_{y_{-i}}(y_i)$ is the problem of determining whether or not $y_i$ matches any of the fixed other blocks. Observe that any choice of $n-1$ fixed blocks leads to a different function $f_{y_{-1}}(y_i)$. This means that for every such combination of $n-1$ of them the functions $f_{y_{-1}}$ are distinct. We conclude that the number of possible functions is at least

$$\binom{2^k}{n-1} \approx n^n.$$

Consider a branching program diagram that computes $D_{n,k}$, the original problem. Now, fixing the other blocks except $y_i$ is equivalent to hardwiring decisions at all bits except the $k$ bits in $y_i$. Hence we can reduce our branching program to a new program that only has labels $x_{ij}$ corresponding to these $k$ bits. We claim that for every $i$ there are at least $O(n)$ gates corresponding to $y_i$, which will imply a $O(n^2)$ lower bound for $BPSIZE$. However, this follows by the same argument we gave for circuit gate lower bounds. We can encode the branching diagram by specifying for each node which node the 0 edge leads to and which node the 1 edge leads to. This takes at most $2 \log n$ bits for each node, so we can encode the entire graph in $2n \log n$ bits and hence there are at most

$$2^{2n \log n} \approx n^{O(n)}$$

branching programs with at most $n$ gates. Since we have showed that at least $n^n$ functions can be computed by hardwiring the other blocks and reducing the program to only gates corresponding with $y_i$, we conclude that there must be at least $O(n)$ gates, so the proof is finished.

□