INFR11102: Computational Complexity 24/10/2019

Lecture 12: Circuit models; Karp-Lipton Theorem

Lecturer: Heng Guo

1 Circuit models

So far we have been focusing on the standard (uniform) Turing machine as our model of computation. Another arguably more natural and seemingly simpler model is Boolean circuits. Boolean circuits are also equivalent to TMs that "take advices". We will define these notions formally next.

A Boolean circuit C with n inputs is a directed acyclic graph (DAG), with n sources (no incoming arcs) and one sink (no outgoing arcs). The n sources are the input bits and the sink is the output. Every internal vertex is a "gate", labelled with $\neg$, $\wedge$, or $\vee$. The negation gate $\neg$ has fan-in (number of incoming arcs) 1, and we will assume $\wedge$ and $\vee$ have fan-in 2.[1] The size of C, denoted $|C|$, is the number of vertices in C.

Notice that we actually do not restrict the fan-out (number of outgoing arcs). In contrast, a Boolean *formula* is a circuit where all internal gates have fan-out 1. (A Boolean formula is essentially a tree.) The advantage of larger fan-out is that we may reuse an intermediate value more than once.

Given an input $\mathbf{x} = (x_1, \cdots, x_n) \in \{0,1\}^n$, the evaluation of $C(\mathbf{x})$ goes in the straightforward way. Namely, we evaluate each gate according to the topological ordering of C gradually, until the output is computed.

All Boolean functions $f : \{0,1\}^n \to \{0,1\}$ can be computed by a circuit, simply by transforming its truthtable into a CNF or DNF. The catch is that it may take exponential size. A function as simple as parity: $\oplus(\mathbf{x}) := \sum_{i=1}^n x_i \mod 2$ requires exponential size CNF or DNF.

Definition 1. *Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$-size circuit family $\mathcal{C}$ is a sequence $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs, and $|C_n| \le T(n)$ for every n.*

A circuit family $\mathcal{C}$ defines its corresponding language:

$$L(\mathcal{C}) := \{\mathbf{x} \mid C_n(\mathbf{x}) = 1 \textit{ if } |\mathbf{x}| = n \}.$$

For a function T, the class $\texttt{Size}[T]$ *is defined as*

$$\texttt{Size}[T] := \{L \mid \exists \textit{ a } T(n)\textit{-size circuit family } \mathcal{C} \textit{ s.t. } L = L(\mathcal{C})\}.$$

We are most interested in polynomial sized circuit families.

[1] This restriction does not change the power of the circuit. Since we can easily replace a $\wedge$ or $\vee$ with fan-in k by $k-1$ gates all with fan-in 2.

Definition 2. $\texttt{P}_{/\texttt{poly}} := \bigcup_{c\in\mathbb{N}} \texttt{Size}[n^c]$.

The circuit model is *non-uniform* in the sense that one can change the algorithm for inputs of different sizes. TM can also distinguish amongst cases, but it can only deal with finitely many cases. Circuits, on the other hand, can deal with infinitely many cases. There are also uniform versions where we require that an algorithm exists to compute the circuits for a given input length first. Nevertheless, the non-uniform version can be argued as efficient computation (as in hardware chips handling inputs up to a certain size), and is indeed more powerful than the standard polynomial time.

Theorem 1. $\texttt{P} \subset \texttt{P}_{/\texttt{poly}}$.

Proof. The proof of this theorem is similar to that of the Cook-Levin theorem. Suppose M is a TM with a polynomial-time bound $P(n)$. Recall that we construct a $P(n)$-by-$O(P(n))$ computational table. Similar to the proof of the Cook-Levin theorem, we can write down Boolean formulae to verify the following things: the first row is the initial configuration, the last row is accepting, and the content of a row is computed according to its previous row. Hence, the total size of the circuit is just $O((P(n))^2)$, which is still a polynomial. □

On the other hand, all unary languages are in $\texttt{P}_{/\texttt{poly}}$. A language L is called *unary* if $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. However, the following unary language is undecidable.

Name: Unary-Halting

Input: 1^n

Output: Does n encode a pair $\langle M, x\rangle$ such that M halts on the input x?

As Unary-Halting $\in \texttt{P}_{/\texttt{poly}}$, the class $\texttt{P}_{/\texttt{poly}}$ is far more powerful than $\texttt{P}$. It is actually very hard to place $\texttt{P}_{/\texttt{poly}}$ amongst the complexity classes we have introduced so far. People conjecture that $\texttt{NP} \not\subseteq \texttt{P}_{/\texttt{poly}}$ (which would imply $\texttt{P} \neq \texttt{NP}$), but there is no promising route towards proving it.

1.1 Circuit lower bounds

The attempt to separate $\texttt{NP}$ from $\texttt{P}_{/\texttt{poly}}$ initiated the study of circuit lower bounds. It is actually very easy to show that there exist hard functions. The difficulty is to find a hard function within $\texttt{NP}$.

Theorem 2. *For any n, there is a function $f : \{0,1\}^n \to \{0,1\}$ that requires at least $\frac{2^n}{Cn}$ gates to compute for some constant C.*

Proof. The proof is a simple counting argument. The total number of functions is 2^{2^n}. For a circuit of size S, it can be described using $C \cdot S \log S$ bits for some constant C. Hence the total number of such circuits is at most $2^{C\cdot S\log S}$.

Let $S = \frac{2^n}{(C+1)n}$. Then circuits of size S can compute at most

$$2^{C\cdot S\log S} \leq 2^{\frac{C}{C+1}\cdot\frac{2^n}{n}\cdot n} < 2^{2^n}.$$

Hence, there must be a function f that circuit up to size S cannot compute. □

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 6] and [Pap94, Chapter 17].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.