PRINCETON UNIVERSITY COS 522: COMPUTATIONAL COMPLEXITY

Lecture 5: Circuits

Lecturer: *Sanjeev Arora* Scribe:*Scribename*

The boolean circuit model mimics a silicon chip. A circuit with n inputs is a directed acyclic graph with n leaves (called the *input wires*) and each internal node is labelled with $\vee, \wedge, \neg$. The $\vee$ and $\wedge$ nodes have fanin (i.e., number of incoming edges) of 2 and the $\neg$ nodes have fanin 1. There is a designated *output node*. We think of n bits being fed into the input wires. Each internal node/gate applies the boolean operation it is labeled with on its incoming wires to produce a single bit. Thus bits filter through the circuit —remember that it is acylic, so feedback effects are absent— until a single output bit emerges at the output node. Thus the circuit implements a function from $\{0,1\}^n \to \{0,1\}$. The *size* of the circuit is the number of nodes in it.

A *circuit family* $(W_n)_{n\geq 1}$ is a sequence of circuits where W_n is a circuit with n inputs. We say that the family *decides* a language L if for every input x,

$$x \in L \Leftrightarrow W_{|x|}(x) = 1. \tag{1}$$

Note that every language is decidable by a circuit family of size $O(n2^n)$, since the circuit for input length n could contain 2^n "hardwired" bits indicating which inputs are in the language. Given an input, the circuit looks up the answer from this table. (The reader may wish to work out an implementation of this circuit.)

DEFINITION 1 *The class* $\mathbf{P}/poly$ *consists of every language that is decidable by a circuit family of size* $O(n^c)$ *for some* $c > 0$.

THEOREM 1
$\mathbf{P} \subseteq \mathbf{P}/poly$.

PROOF: We show that every language that is decidable in $t(n)$ time has circuits of size $O(t(n)^2)$. (A more careful argument can actually yield circuits of size $O(t(n)\log t(n))$. The main idea is similar to that in the Cook-Levin theorem, where we noticed that computation is very local, since the machine affects its tape only in the cell currently occupied by its head.

On any input of size n, the tableau of the machine's computation is a matrix of size $t(n) \times t(n)$. Assume wlog that the machine enters the leftmost cell before accepting. Construct a circuit in which use a wire for each bit of the tableau. The bits of the machine's input enter at the input wires of the circuit. Construct a "next step" module, which is a circuit that, given the contents of three adjacent cells in a row of the tableau, computes the contents of the cell below the middle cell. Since the machine has a fixed transition diagram and alphabet, each tableau cell is represented by $k = O(1)$ bits, so the the next-step circuit has size $O(1)$ (could be exponential in k). Replicate this next-step module everywhere as needed so that all the bits in the tableau are computed correctly. (See Figure 1.) The output of the overall circuit is computed by examining the cell in the lower left corner of tableau; it is 1 iff the machine is in an accept state. $\Box$

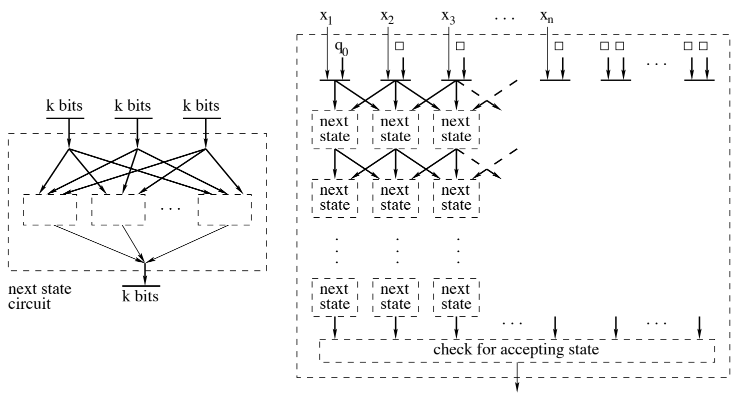

Figure 1: Circuit to simulate a Turing machine computation by constructing the tableau.

Can we give a good upperbound on the computational power represented by **P**/poly? The next example shows this is difficult since **P**/poly contains an undecidable language.

Example 1 Every unary language has linear size circuits since the circuit for an input size n only needs to have a single "hardwired" bit indicating whether or not 1^n is in the language. The following unary language is undecidable:

$$\left\{1^{<M,w>} : \text{ TM } M \text{ accepts } w\right\}, \tag{2}$$

where $< M, w >$ is an encoding that encodes strings with integers.

1 Karp-Lipton Theorem

Is SAT in **P**/poly? If so, one could imagine a government-financed project to construct a small circuit that solves SAT on, say, 10,000 variables. Upon discovering such a circuit we could incorporate it in a small chip, to be used in all kinds of commercial products. In particular, such a chip would jeopardise all current cryptographic schemes.

The next theorem seems to dash such hopes, at least if you believe that the polynomial hierarchy does not collapse.

Theorem 2 (Karp-Lipton, with improvements by Sipser)
If $\mathbf{NP} \subseteq \mathbf{P}/poly$ *then* $\mathbf{PH} = \mathbf{\Sigma}_2^p$.

Note that if $\mathbf{NP} \subseteq \mathbf{P}$/poly then SAT has a polynomial-size circuit family. Namely, for some $k > 0$ there is a sequence of circuits $W_1, W_2, W_3, \ldots$, where W_n is a circuit of size n^k that decides satisfiability of all boolean formulae whose encoding size is $\leq n$. Of course, $\mathbf{NP} \subseteq \mathbf{P}$/poly merely implies the *existence* of such a family; there may be no easy way to construct the circuits.

Lemma 3 (Self-reducibility of SAT)
There is a polynomial-time computable function h such that if $\{W_n\}_{n\geq 1}$ is a circuit family that solves SAT, then for all boolean formulae φ:

$$\varphi \in SAT \qquad \text{iff} \qquad h(\varphi, W_{|\varphi|}) \text{ is a satisfying assignment for } \varphi. \tag{3}$$

Proof: The main idea in the computation of h is to use the provided circuit to generate a satisfying assignment. Ask the circuit if φ is satisfiable. If so, ask it if $\varphi(x_1 = T)$ (i.e., φ with the first variable assigned True) is satisfiable. The circuit's answer allows us to reduce the size of the formula. If the circuit says no, we can conclude that $\varphi(x_1 = F)$ is true, and have thus reduced the number of variables by 1. If the circuit says yes, we can substitute $x_1 = T$ and again reduced the number of variables by 1. Continuing this way, we can generate a satisfying assignment. This proves the Lemma. (Aside: The formal name for the above property of SAT is *downward self-reducibility.* All **NP**-complete languages have this property.) $\square$

Now we prove the Theorem.

Proof:(Theorem 2) We show $\mathbf{NP} \subseteq \mathbf{P}/\text{poly}$ implies $\mathbf{\Pi}_2^p \subseteq \mathbf{\Sigma}_2^p$.

Let $L \in \mathbf{\Pi}_2^p$. Then there is a language $L_1 \in \mathbf{NP}$ and $c > 0$ such that

$$L = \{x \in \{0,1\}^* : \forall\, y, |y| \leq |x|^c, (x,y) \in L_1\}. \tag{4}$$

Since $L_1 \in \mathbf{NP}$, there is a polynomial-time reduction, say g, from L_1 to SAT. Thus

$$\forall\, z \in \{0,1\}^* : z \in L_1 \qquad \text{iff} \qquad g(z) \in \text{SAT}.$$

Suppose further that $d > 0$ is such that $|g(z)| \leq |z|^d$.

Now we can rewrite (4) as

$$L = \{x \in \{0,1\}^* : \forall\, y, |y| \leq |x|_1^c, g(x,y) \in \text{SAT}\}.$$

Note that $|g(x,y)| \leq (|x|+|y|)^d$. Let us simplify this as $|x|^{cd}$. Thus if $W_{n\geq 1}$ is a n^k-sized circuit family for SAT, then by (3) we have

$$L = \left\{x : \forall\, y, |y| \leq |x|_1^c, h(g(x,y), W_{|x|^{cd}}) \text{ is a satisfying assignment for } g(x,y)\right\}.$$

Now we are almost done. Even though there may be no way for us to construct the circuit for SAT, we can just try to "guess" it. Namely, an input x is in L iff

$$\exists W, \text{ a circuit with } |x|^{cd} \text{ inputs and size } |x|^{cdk} \text{ such that}$$
$$\forall\, y, |y| \leq |x|_1^c, h(g(x,y), W) \text{ is a satisfying assignment for } g(x,y).$$

Since h is computable in polynomial time , we have thus shown $L \in \mathbf{\Sigma}_2^p$. $\square$

2 Circuit lowerbounds

Theorem 1 implies that if $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$, then $\mathbf{P} \neq \mathbf{NP}$. The Karp-Lipton theorem gives hope that $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$. Can we resolve **P** versus **NP** by proving $\mathbf{NP} \not\subseteq \mathbf{P}/\text{poly}$? There is reason to invest hope in this approach as opposed to proving direct lowerbounds on Turing machines. By representing computation using circuits (Theorem 1) we seem to actually peer into the guts of it rather than treating it as a blackbox. Thus we may be able to get around the relativization results of Lecture 3.

Sadly, such hopes have not come to pass. After two decades, the best circuit size lowerbound for an **NP** language is only $4n$. (However, see the Problems for a better lowerbound for a language in **PH**.) On the positive side, we have had notable success in proving lowerbounds for more restricted circuit models, as we will see later in the course.

By the way, it is easy to show using counting arguments that for large enough n, almost every boolean function on n variables requires circuits of size at least $2^n/10n$. The reason is that there are at most s^{3s} circuits of size s (just count the number of labelled directed graphs, where each node has indegree at most 2). For $s = 2^n/10n$, this number is miniscule compared 2^{2^n}, the number of boolean functions on n variables. Hence most boolean functions do not have such small circuits.

Of course, such simple counting arguments do not give an explicit boolean function that requires large circuits.

3 Turing machines that take advice

There is a way to define $\mathbf{P}$/poly using TMs. We say that a Turing machine has *access to an advice family* $\{a_n\}_{n\geq 0}$ (where each a_n is a string) if while computing on an input of size n, the machine is allowed to examine a_n. The advice family is said to have *polynomial size* if there is a $c \geq 0$ such that $|a_n| \leq n^c$.

Theorem 4
$L \in \mathbf{P}$/poly iff L is decidable by a polynomial-time Turing machine with access to an advice family of polynomial size.

Proof: If $L \in \mathbf{P}$/poly, we can provide the description of its circuit family as advice to a Turing machine. When faced with an input of size n, the machine just simulates the circuit for this circuit provided to it.

Conversely, if L is decidable by a polynomial-time Turing machine with access to an advice family of polynomial size, then we can use the construction of Theorem 1 to construct an equivalent circuit for each input size with the corresponding advice string hardwired into it. $\square$

Exercises

§1 Show for every $k > 0$ that **PH** contains languages whose circuit complexity is $\Omega(n^k)$. (Hint: First show that such a language exists in $\mathbf{SPACE}(2^{\text{poly}(n)})$.)

§2 Show that if some unary language is **NP**-complete, then $\mathbf{P} = \mathbf{NP}$.

§3 (*Open*) Suppose we make a stronger assumption than $\mathbf{NP} \subseteq \mathbf{P}$/poly: every language in **NP** has linear size circuits. Can we show something stronger than $\mathbf{PH} = \mathbf{\Sigma}_2^p$?