

Lecture 10: Non-Uniformity, Poly-Size Circuits, and Karp-Lipton

Instructor: Jin-Yi Cai

Scribe: Bess Berg, Yang Gao, Michael Schultz

In previous lectures we have been mainly concerned with uniform complexity, that is, the scenario of the finite algorithm attempting to compute some property for input of any size n . Here we will begin to discuss non-uniform complexity, which is concerned with the complexity of a language at some given length n . For example, consider a variation on the halting problem, H' , where H' includes only strings of the form 1^n , and the string 1^n is in H' if and only if the n^{th} turing machine halts on 1^n :

$$H' = \{1^n \mid M_n(1^n) \downarrow\}$$

H' is indeed undecidable, and yet for any given n , one of two very simple circuits will decide if some input $x = x_1x_2\dots x_n$ of length n is in H' : either $C_1 = x_1 \wedge \neg x_1$ if 1^n is not in H' , or $C_2 = x_1 \wedge x_2 \wedge \dots \wedge x_n$ if 1^n is in H' . Then we see that non-uniform complexity is capable of capturing qualities that may be overlooked in uniform complexity.

1 Polynomial Size Circuits

A Boolean circuit C_n is a finite acyclic directed graph where each node is labeled either as an input node (labeled with some variable x_i , where $1 \leq i \leq n$) or as a logical gate from the set $\{\vee, \wedge, \neg\}$. A Boolean circuit by default has one output gate (though it may have more). Though sometimes we will talk about unbounded fan-in, by default the \wedge - and \vee -gates have in-degree 2, while the \neg -gate has in-degree 1. Note that by De Morgan's laws it is always possible to "push down" the \neg -gates in a circuit, at most doubling the circuit's size. The size $S(C_n)$ of a circuit C_n is the number of wires (edges) in the circuit, and the depth $D(C_n)$ of a circuit C_n is the length of the longest path from input to output (not counting \neg -gates). By construction, C_n computes some function $\{0, 1\}^n \rightarrow \{0, 1\}$ for inputs of length n .

As one example, consider the parity function $\bigoplus_n : \{0, 1\}^n \rightarrow \{0, 1\}$, where $\bigoplus_n(x_1x_2\dots x_n) \rightarrow \sum_1^n x_i \pmod{2}$. The parity of 2 bits a and b can be computed by the constant size circuit $A = ((a \wedge \neg b) \vee (\neg a \wedge b))$. Since $\sum_1^n x_i \pmod{2} = \sum_1^{n/2} x_i \pmod{2} + \sum_{n/2+1}^n x_i \pmod{2}$, we can construct a circuit computing parity for n bits by combining two sub circuits computing parity for $n/2$ bits. The size of the circuit would be linear in n and its depth would be $O(\log n)$.

DEFINITION 1.1 *A language $L \subseteq \{0, 1\}^*$ is said to have polynomial size circuits if there exists a family of circuits $\{C_n\}$ such that*

1. *for any n the size of C_n is $n^{O(1)}$, and*
2. *C_n computes $\chi(L^=n)$, where χ is the characteristic function for $L^=n$, the set of strings in L of length n . That is, on input x of length n , C_n outputs 1 if $x \in L$ and 0 otherwise.*

THEOREM 1.2 *Every language in P has polynomial size circuits.*

Proof. Let L be a language computed by some polynomial-time DTM M . Fix an input length n , and we will construct a circuit C that computes L^n . A configuration of M on input an x (consisting of the state of the machine, tape contents and head position) can be represented in $p(n)$ bits for some polynomial $p(\cdot)$. We can assume that the machine runs for $p(n)$ time steps. The circuit C will have $p(n)$ “layers”. Each layer t simulates the computation of M at time step t , and outputs the configuration of machine at the end of time step t . So the layer 0 will simply output the initial configuration (consisting of start state, tape content being input x , and tape head position being 0). For $t > 1$, layer t takes as input configuration at the end of time step $t - 1$ and outputs the configuration at the end of time step t . It is easy to see that any layer t can be implemented using a constant depth (poly-size) circuit using the transition function δ of the machine. Finally, at the last layer, we check if the output is an accepting configuration. ♣

It is widely believed that NP does not have polynomial size circuits. The reason we believe NP problems are so difficult is the above conjecture and not the restrictions of uniformity (a finite algorithm having to act on inputs of any length).

2 The Karp-Lipton Theorem

Now we are ready to prove the Karp-Lipton Theorem, which provides evidence that NP does not have polynomial size circuits by showing that if it did, the polynomial hierarchy would collapse.

It is worth mentioning that neither the original Karp-Lipton statement nor proof is presented here, but it is close enough!

THEOREM 2.1 *If NP has poly-size circuits, then $PH = \Sigma_2^P \cap \Pi_2^P$.*

It is sufficient to show that if *SAT* has poly size circuits, then $PH = \Sigma_2^P \cap \Pi_2^P$. Now suppose we have a circuit C_n that is claimed to compute SAT for all boolean formulas of input length n . It may or may not compute SAT correctly. If it is incorrect for some input φ , then it has made exactly one of the following two errors. It has either rejected when indeed $\varphi \in \text{SAT}$ or it has accepted when in fact $\varphi \notin \text{SAT}$. Thus C_n may err on both sides. We can convert it to a circuit C'_n that errs only on one side. Meaning, it may reject a satisfiable formula, but never accept an unsatisfiable formula. Its construction is based on self-reducibility. C'_n implements the following algorithm.

INPUT: Any boolean formula φ of input length n over the variables x_1, x_2, \dots, x_n and a circuit C_n used as a black box.

OUTPUT: Either rejects φ or if it accepts, it will also produce satisfying assignment σ .

1. if $C_n(\varphi) = \text{no}$, then reject and halt.
2. else $C_n(\varphi) = \text{yes}$ and we do the following:

3. we attempt to construct a satisfying truth assignment σ .
4. for $i = 0$ to n do:
5. { set $\sigma(x_i) = 1$
6. $\varphi' = \varphi(\sigma)$ (i.e. φ' is obtained from φ by setting the first i variables according to the partial assignment σ .)
7. if $C_n(\varphi') = \text{no}$, then set $\sigma(i) = 0$ }.
8. verify that σ satisfies φ , and if so, accept and return σ
9. if verification fails, reject.

It should be clear that the above algorithm can be implemented in the form of a circuit C'_n whose size is polynomial in n . Moreover, we can design a polynomial algorithm, which constructs C'_n given C_n as input.

If C_n is a correct circuit for SAT, C'_n must also be a correct one, since C'_n only makes one type of error: it may reject when $\varphi \in SAT$. C'_n refuses all unsatisfiable formulas since it only accepts what it has verified.

Now to the proof! (As noted earlier, we will not use Karp and Lipton's original proof but rather use a proof noted by Hopcroft.)

Proof. Given any language $L \in \Pi_2^P$, for some deterministic polynomial time predicate D , L can be described as

$$x \in L \iff \forall^P y \exists^P z [D(x, y, z) = 1]$$

where $|y|$ and $|z|$ are polynomially bounded in $|x|$. Via Cook's theorem, we convert the question of

$$\exists^P z [D(x, y, z) = 1]$$

into a SAT question $\varphi_{x,y}$ such that

$$x \in L \iff \forall^P y [\varphi(x, y) \in SAT]$$

By our assumption, there are circuits C_n correctly computing SAT and polynomial in the length of their inputs. Since the inputs of C_n are the lengths of formulas, and since Cook's theorem assures the length of φ , $|\varphi|$, is polynomial in the length of x , the circuit C_n for $n = |\varphi|$ is also polynomial in x .

Remember we ultimately want to show that there is a Σ_2^P machine also computing this Π_2^P computation. So have our Σ_2^P machine guess, in the \exists phase, the circuit C_n for $n = |\varphi|$. Now construct the C'_n which only makes the one-sided errors as discussed before. Now our machine enters the \forall

phase and checks if for all polynomially bounded y 's, there is a satisfying formula for C'_n . Does this work?

\implies Suppose $x \in L$. During the \exists phase, one of the paths will guess C_n correctly, and for the universal stage branching from this path, $\varphi(x, y)$ will be satisfiable for all polynomially bounded y 's. Then this Σ_2^p machine will accept x .

\Leftarrow Suppose $x \notin L$. There must be some polynomially bounded y_0 such that the formula $\varphi(x, y)$ is not satisfied. Because C'_n rejects any unsatisfiable formula, for all the guesses C_n , the corresponding C'_n will also reject $\varphi(x, y_0)$. Then our Σ_2^p machine will reject x . So our Σ_2^p machine decides L .

Since a deterministic polynomial time algorithm can be used to compute C'_n from C_n , and another deterministic polynomial time algorithm (Cook's theorem) can be used to translate $D(x, y, x)$ into a *SAT* question, and then again since the circuits answer the *SAT* question in deterministic polynomial time, we can generate another deterministic polynomial time predicate D' such that

$$x \in L \iff \exists^p C \forall^p y [D'(x, C, y) = 1].$$

Then this is definitely a Σ_2^p machine deciding L , and since this can be done for any $L \in \Pi_2^p$, we have $\Pi_2^p \subseteq \Sigma_2^p$. ♣