ITCS:CCT09 : Computational Complexity Theory	Mar 16, 2009
Lecture 4	
Lecturer: Jayalal Sarma M.N.	*Scribe: Kai Jin*

We will see some more of structural complexity results in this lecture and then switch to some basics of circuit complexity.

1 Oracle Turing Machines

We first define the notion of oracle computations which is classical in recursion theory and was also used extensively in classical complexity theory. We described these in an informal way in the last lecture, but we did not need the formal definition there.

Oracle model essneitally modelling the notion of comparison of problems. We will get back to this aspect when we formally define the reductions in upcoming lectures.

Definition 1 (Oracle Turing Machines) *A 2-tape deterministic oracle Turing machine can be definied by nine components* $(Q, q_0, q_?, q_Y, q_N, F, \Sigma, \Gamma, \delta)$*. The symbols* Q*,* F*,* q_0 *and* Σ *are interpreted exactly as in the case of usual DTMs. There are three special states: query state* $q_?$*, 'Yes' state* q_Y*, 'No' state* q_N*, and a special query tape.* δ *is a transition from* $(Q - F - \{q_?\}) \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R, S\}^2$*.*

M is trying to decide $x \in L$ with oracle access to L'. On input x, M can write a query string y to the query tape, and move to query state. Oracle checks if $y \in L'$ and puts M in 'Yes' state; otherwise puts M in 'No' state, and continues the computation. When the resources of the computation is referred to the resources used by the oracle to check the membership in L' are *not* counted. That is, the transition from the query state to the answer state is counted as only one step of the computation. We denote the language accepted by the machine as $L^{L'}$.

A *nondeterministic oracle Turing machine* is the same as the above definition for a DTM, except that the machine M can make non-deterministic moves only at the non-query states.

The definition of the oracle Turing machines are general enough to be applied to various complexity classes without any case specific variation. In particular, we can define *relativised* complexity classes based on this definition.

Definition 2 *For a complexity class* C*, and a language* L*,*

$$\mathsf{C}^L = \left\{ L \;\middle|\; \begin{array}{c} \exists \text{ OTM } M \text{ respecting resource bounds of } \mathsf{C} \\ \text{such that } L \text{ is accepted by } M^L. \end{array} \right\}$$

Generalizing this further, for two complexity classes C_1 *and* C_2*,*

$$\mathsf{C}_1^{\mathsf{C}_2} = \left\{ L \;\middle|\; \begin{array}{c} \exists \text{ OTM } M \text{ respecting resource bounds of } \mathsf{C}_1 \\ \text{such that } L \text{ is accepted by } M^L \text{ where } L \in \mathsf{C}_2. \end{array} \right\}$$

Proposition 3 *For any two complexity classes* C_1 *and* C_2*,* ${\mathsf{C}_1}^{\mathsf{C}_2} = {\mathsf{C}_1}^{\overline{\mathsf{C}_2}}$*.*

Proof Suppose $L \in {\mathsf{C}_1}^{\mathsf{C}_2}$, then $\exists M \in \mathsf{C}_1, N \in \mathsf{C}_2$, modify N to N' which output the opposite answer of N, s.t. $N' \in \overline{\mathsf{C}_2}$. Use N' to be the oracle, but after each query, we change the state to the opposite one. L can be accept by $M^{N'}$ hence $L \in {\mathsf{C}_1}^{\overline{\mathsf{C}_2}}$. $\therefore {\mathsf{C}_1}^{\mathsf{C}_2} \subseteq {\mathsf{C}_1}^{\overline{\mathsf{C}_2}}$. The same proof gives the reverese inclusion too. ■

Following is another easy proposition:

Proposition 4 *For any complexity classes* $\mathsf{C}_1, \mathsf{C}_2$ *and* C_3*, if* $\mathsf{C}_1 \subseteq \mathsf{C}_2$*, then* $\mathsf{C}_3^{\mathsf{C}_1} \subseteq \mathsf{C}_3^{\mathsf{C}_2}$*.*

To understand the notion of oracle access let us try out an example, take $\mathsf{C}_1 = \mathsf{NP}$ and $\mathsf{C}_2 = \mathsf{P}$. Now, $\mathsf{C}_1^{\mathsf{C}_2}$ is NP^{P}. Think of the NP machine M with oracle access to a language $L \in \mathsf{P}$. But then, it is clear that the machine does not need to actually query the oracle machine, it can simulate the polynomial time algorithm for deciding the membership in L within its own resource bounds. This demonstrates that $\mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$. To take an example in the other extreme, let $\mathsf{C}_1 = \mathsf{P}$ and $\mathsf{C}_2 = \mathsf{PSPACE}$. Clearly $\mathsf{PSPACE} \subseteq \mathsf{P}^{\mathsf{PSPACE}}$ (in fact this is an equality). In this case the oracle really "helps" the computation.

Can one apply the same simulation methods as above to prove that $\mathsf{NP}^{\mathsf{NP}} \subseteq \mathsf{NP}$? Not really. We discussed this informally. The details of this will be asked in the homeworks (!). It is not known whether $\mathsf{NP}^{\mathsf{NP}}$ is same as NP or not. In fact it is not even clear if P^{NP} can be simulated by an NP machine. Using these as the base cases we can define a heirarchy of complexity classes which contains NP, as the so-called *polynomial heirarchy.*

1.1 Polynomial Hierarchy

The class $\mathsf{NP}^{\mathsf{NP}}$ is denoted by Σ_2^p (The name of this class will be justified by a characterisation that we will see later). As we said in the previous section it is unclear whether $\Sigma_2^p \subseteq \mathsf{NP}$. The obvious simulation fails. One can further ask about $\mathsf{NP}^{\Sigma_2^p}$, and this class is called Σ_3^p again seemingly more powerful than Σ_2^p and NP. This leads to the following definition.

Definition 5 (Σ_k^p & Π_k^p) *Let us set $\Sigma_0^p = \Pi_0^p = \mathsf{P}$. Now the classes are defined inductively. For $k \geq 1$, $\Sigma_k^p = \mathsf{NP}^{\Pi_{k-1}^p}$, $\Pi_k^p = \mathsf{coNP}^{\Sigma_{k-1}^p}$.*

From the definition, $\Sigma_1^p = \mathsf{NP}^{\mathsf{P}}$, and as we saw this is same as NP. Similarly, $\Pi_1^p = \mathsf{coNP}$. In fact, from the definition itself the following relationship among these classes are easy to derive.

Proposition 6

$$\mathsf{P} = \begin{matrix}\Sigma_0 \\ \Pi_0\end{matrix} \subseteq \begin{matrix}\Sigma_1 = \mathsf{NP} \\ \Pi_1 = \mathsf{coNP}\end{matrix} \subseteq \begin{matrix}\Sigma_2 \\ \Pi_2\end{matrix} \subseteq \cdots \subseteq \begin{matrix}\Sigma_k \\ \Pi_k\end{matrix} \subseteq \cdots \subseteq \cdots$$

Thus there is an infinite series of complexity classes which contains P and NP in the base case. This heirarchy defines the polynomial heirarchy.

Definition 7 (PH)

$$\mathsf{PH} = \bigcup_{k \geq 0} \Sigma_i$$

What is an upperbound for PH. For example, we know that NP is in PSPACE from Savitch's theorem (Lecture 3). From simple simulations, it follows that following proposition to show $\mathsf{NP}^{\mathsf{NP}} \subseteq \mathsf{NP}^{\mathsf{PSPACE}} = \mathsf{PSPACE}^{\mathsf{PSPACE}} \subseteq \mathsf{PSPACE}$. Inductively applying this, will give us the following claim.

Proposition 8 $\mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{PSPACE}$.

We will show thefollowing characterisation for the classes in the polynomial heirarchy. However, we will postpone the proof to next-to-next lecture.

Theorem 9 *A Language L is in Σ_k if and only if there is a $B \in \mathsf{P}$ and a polynomial p such that*

$$x \in L \Leftrightarrow \exists y_1 \ \forall y_2 \ \cdots \ Q \ y_k, (x, y_1, y_2, \cdots, y_k) \in B$$

where $\forall i$, $|y_i| \leq p(|x|)$, and $Q = \forall/\exists$ depending on whether k is odd or even.

We will get back to this theorem again after one lecture. Now we will start with the basic notions of non-uniform complexity and introduce the basic circuit complexity classes so that you can follow tomorrow's ITCS seminar about this topic. Don't miss it !.

2 Non-uniform Complexity

In the Turing machine model, the description of the machine that works for all inputs is the same irrespective of the length of the input. A natural consideration is to relax this requirement, and consider a scenario where there is one "computational device" for each input length. However, this can conceivably compute more than what a Turing machine can compute, since there is no restriction on how the description of the n^{th} (which works of all inputs of length n) computational device is obtained. In fact, such a set up can solve even undecidable problems. We will make this more precise later.

Hence, it is reasonable to impose more contraints on how to obtain the description of device given n. These are called *uniformity constraints* for the computational model. We will now describe a specific example of this computational device called Boolean circuit.

Definition 10 (Boolean Circuit) *A Boolean circuit is a directed acyclic graph, $G = (V, E)$ such that each v in V with non-zero in-degree called the* gate *of the circuit is an element of $B = \{\neg, \vee, \wedge\}$ except leaves which are labeled by $x_1 \dots x_n$. We say that the circuit evaluates to 1 on input x (denoted by $C(x) = 1$) if and only if the root gate evaluates to 1 on input x.*

Clearly there is a Boolean circuit for each input n. Thus we need to talk about circuit families.

Definition 11 (Acceptace Condition) *A language $L \subset \Sigma^*$ is accepted by a circuit family $\{C_n\}_{n\geq 1}$ if $\forall n, x \in \{0,1\}^n : x \in L \Leftrightarrow C_n(x) = 1$.*

Now that we have defined a seemingly different computational model, we can talk about various parameters associated with it which could be considered as the resources of computation. Following are some of them which we will use in this course.

Definition 12 (Resources of computation) *For a circuit family $\{C_n\}_{n\geq 1}$, define for each n the following parameter for the circuit C_n.*

- SIZE*: Number of edges in the underlying graph C_n.(When it will not matter in the order notation, we can talk about size as the number of vertices of the graph.*
- DEPTH: *The length of the longest path from any leaf to root in the graph of C_n.*
- WIDTH: *Consider the layered graph of C_n with the constraint that gates in each layer takes in inputs from the previous layer. The number of edges in each layer is the width of the circuit.*

- FAN-IN: *maximum in-degree of any gate. Sometimes, $\vee$-fanin, $\wedge$-fanin are also resources.*

Now we get back to the question of uniformity specific to the circuit model of computation. We demonstrate the non-uniform family of circuits can solve ha Halting problem. Consider the halting set

$$H = \{1^n | n^{th} \text{ TM } M_n \text{ halts on input } 1^n\}$$

Here is a circuit family $\{C_n\}_{n \geq 1}$ that computes H.

$$C_n = \begin{cases} x_1 \vee \neg x_1 & \text{if } M_n \text{ halts on } 1^n \\ x_1 \wedge \neg x_1 & \text{if } M_n \text{ doesn't halt on } 1^n \end{cases}$$

Thus, it is undecidable to compute the description of the above C_n given n as the input although the family itself is computing the halting set H. This motivates the study of uniformity constraints.

Various uniformity constraints are studied for circuit families depending on the context in which it is applied. For example, a log space uniformity constraint will say given n the description of C_n should be computable by a TM in log space. Sometimes, this is made more precise by saying that the predicate CHILD(n, i, j, g_1, g_2) (in circuit C_n, the node i of type g_1 feeds into node j of type g_2) is checkable in log space. One could also consider other uniformity constraints based on various resources.

3 Circuit Complexity Classes

We will now define circuit complexity classes. But we motivate them using example problems. We will look out for doing as efficiently as possible in terms of the parameters that we defined. We consider first the parity problem of n bits.

Definition 13 (Parity Problem)
INPUT: *n bits $x_1, x_2, \cdots, x_n$;*
OUTPUT: $x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

Suppose we want to compute the parity of 2 bits, $x_1 \oplus x_2$. This can be written as $x_1\overline{x_2} + \overline{x_1}x_2$. Now the Parity Problem can be solved using this by divide and conquer. This gives a complete binary tree like circuit of two input Parity problem. Replace each of them with the small circuit above. The circuit has poly-size, fanin of each gate as 2, log-depth. This motivates the definition of a natural class of problems.

Definition 14 (NC^1)

$$\mathsf{NC}^1 = \left\{ L \;\middle|\; \begin{array}{c} \exists \text{ } a \text{ } \mathsf{poly}\text{-}size, \text{ } bounded \text{ } (constant) \text{ } fanin, \\ \log \text{ } depth \text{ } circuit \text{ } family \text{ } accepting \text{ } L. \end{array} \right\}.$$

Now we consider circuits for another natural problem of adding two n bit numbers.

Definition 15 (Addition Problem)
INPUT: *Two n-bit numbers a, b;*
OUTPUT: *The $(n+1)$-bit number $a+b$*

A straightforward way is the circuit implementation of the highschool method of adding two n bit numbers ; adding from the lowest bit to highest one trasnferring carry from one stage to the other. The circuit corresponding to this method need depth n. The reason is that the higher bits will depend on the carry is computed even at the first stage.

A smarter way to implement this is the well-known *"Look ahead method"*: Why not calculate the carry bits directly? Consider how the carry bit fluctuates between 0 and 1 as we move across stages. When the carry is 1, it is clear that both the input bits are 1, and this carry propogates as long as the two inputs bits at the stages are 1s. When it falls to 0, it is because both the input bits are 0 at that stage. It is not hard to find out the complete formula of c_j,

$$c_j = \bigvee_{i<j} \left[(x_i \wedge y_i) \wedge \left(\bigwedge_{j \leq k < i} (x_k \vee y_k) \right) \right]$$

.

So c_j could be evaluated by a constant depth (although unbounded fanin) circuit. With this carry bits, the Addition Problem could be solved by a poly size [1], constant depth, unbounded fanin circuit for addition. Now we define a circuit complexity class based on this kind of circuits.

Definition 16 (AC^0)

$$\mathsf{AC}^0 = \left\{ L \;\middle|\; \begin{array}{c} \exists \text{ } a \text{ } \mathsf{poly}\text{-}size, \text{ } bounded \text{ } (constant) \text{ } depth \\ circuit \text{ } family \text{ } accepting \text{ } L. \end{array} \right\}.$$

It is also easy to compare between these classes.

[1] the size of the circuit is $O(n^3)$ which can be made to $O(n^2 \log n)$, and it is unknown whether there is a linear sized adder !

Theorem 17 $\mathsf{AC}^0 \subseteq \mathsf{NC}^1$

Proof Take any AC^0 circuit family. Each circuit C_n has constant depth, $\mathsf{poly}(n)$ size. Replace each $\wedge$ and $\vee$ gates of unbounded fan-in by a log-depth tree of bounded fanin gates of the respective type. This gives a log depth circuit of poly size of bounded fan-in $\wedge$ and $\vee$ gates. ■

Later in this course we will prove that this inclusion is strict by proving that PARITY is not in AC^0.

We saw that PARITY problem is in NC^1. How powerful is this problem. In fact it also defines a gate $\oplus$ which is essentially addition in $\mathbb{F}_2$ or sum modulo 2. Generalising this, one can consider sum modulo an integer $q \in \mathbb{Z}$. Thus, one can study a new circuit complex class ACC^0, that is AC^0 + "all mod gates". In another words, we modify the definition of Boolean Circuit, that is, from $B = \{\neg, \vee, \wedge\}$ to $B = \left\{\neg, \vee, \wedge, \bigcup_{q\in\mathbb{Z}}\{ \mod (q)\}\right\}$, and other requirement remains the same.

Generalizing the proof of the fact that PARITY problem is in NC^1, we can prove that an arbitrary $\mod (q)$ gate can also be computed in NC^1. This give the following result.

Theorem 18 $\mathsf{ACC}^0 \subseteq \mathsf{NC}^1$.

Finally we will end this quick overview by setting up an upper bound for all these complexity classes. We claim that any NC^1 circuit can be evaluated in L. Given a circuit C_n and an input x, we can evaluate $C_n(x)$ by a depth first search starting from the root node on the graph G and keeping track of the evaluated bit of the root. Since the depth is log, and fan-in is bounded, the DFS can just store the path from the root to the current search point from the root as the information that is needed for backtracking. And this takes just $\log n$ bits of storage.

This gives,

Theorem 19

$$\mathsf{AC}^0 \subseteq \mathsf{ACC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L}$$