CS 2401 - Introduction to Complexity Theory

Lecture #2: Fall, 2015

Lecturer: Toniann Pitassi

Scribe Notes by: Shuvomoy Das Gupta

1 Alternative definition of NP

Definition (*Nondeterministic Turing Machine (NDTM)*)
An NDTM is similar to a TM, but can have any number of transition functions, e.g., two as follows:

$$\delta_0 : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k,$$
$$\delta_1 : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k,$$

and a special state $q_{\text{accept}} \in Q$.

When an NDTM M computes a function, at each step it makes a random choice of the transition function to be applied. A sequence of such random choices are called *nondeterministic choices* of M. For any input x,

$$M(x) = \begin{cases} 1, \text{if there exists a sequence of nondeterministic choices such that } M \text{ reaches the state } q_{\text{accept}} \\ 0, \text{if for every sequence of nondeterministic choices } M \text{ halts before reaching the state } q_{\text{accept}}. \end{cases}$$

The NDTM M runs in $T(n)$ time if for any input $x \in \{0,1\}^*$ and for any sequence of nondeterministic choices, M reaches either q_{halt} or q_{accept} within $T(|x|)$ steps.

Definition (*NTIME*)
For any function $T : \mathbb{N} \to \mathbb{N}$ and language $L \subseteq \{0,1\}^*$, $L \in \mathbf{NTIME}(T(n))$, if there exist a positive constant c and a $c \cdot T(n)$ time NDTM M such that for any $x \in \{0,1\}^*$ we have $x \in L \Leftrightarrow M(x) = 1$.

Theorem 1 $\boldsymbol{NP} = \cup_{c \in \mathbb{N}} \boldsymbol{NTIME}(n^c)$.

Proof
(Proof of $\cup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c) \subseteq \mathbf{NP}$)

Consider any language $L \in \cup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$, which is equivalent to saying that there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that L is determined by a NDTM N which runs in time $p(n)$. So by definition of NDTM, for any $x \in L$, we can find a sequence of nondeterministic choices that will cause N to reach q_{accept} in time $p(n)$. Now the key idea is using such a sequence as a witness for x. The witness has length $p(|x|)$ and it can be verified in polynomial time by a deterministic TM. This TM simulates the action of N using the witness and verifies that the TM reaches q_{accept}. So by using the definition of class NP, we have $L \in NP$.
(Proof of $\mathbf{NP} \subseteq \cup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$)

Consider any language $L \in \mathbf{NP}$. Then by definition of **NP**, there is polynomial time TM M such that $\forall x \in L$ there exists $u \in \{0,1\}^{p(|x|)}$ such that $M(x,u) = 1$, where p is a polynomial. We have to show that, there exists a positive constant c and $c \cdot T(n)$ time NDTM N such that $N(x) = 1$, i.e., we have to find a sequence of nondeterministic choices u such that N reaches the state q_{accept}. At first using the nondeterministic choice making capability of N, we construct a string $\bar{u}$ with length $p(|x|)$. We input $\bar{u}$ as the witness in $M(x,\cdot)$ and check if $M(x,\bar{u}) = 1$, and if that happens, then it enters q_{accept}. The NDTM N enters q_{accept} if and only if $\bar{u}$ is a valid witness. As there must exist a number $c > 1$, such that $p(n) = O(n^c)$, we have L and $L \in \mathbf{NTIME}(n^c) \subset \cup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$.

[QED]

2 NP-completeness

Definition *(Reduction)*
A language $L \subseteq \{0,1\}^*$ is polynomial-time Karp reducible to a language $L' \subseteq \{0,1\}^*$, denoted by $L \leq_p L'$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$, such that for every $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$. Figure 1 shows a Karp reduction from L to L', which is a polynomial function f which maps strings in L to strings in L'.

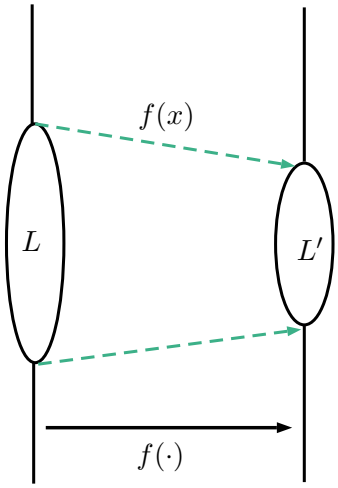

Figure 1: Karp reduction f from L to L'.

Definition ***(NP-hard)***
A language L' is **NP**-hard if for every $L \in \mathbf{NP}$ we have $L \leq_p L'$.

Definition ***(NP-complete)***
A language L' is **NP**-complete if L' is **NP**-hard and $L' \in NP$.

Cook Reduction. This is a more general polynomial time reduction, that allows you to use an oracle for L' to solve L in polynomial time.

Famous NP languages.

- The clique problem: Accept $\langle G, k\rangle$ if and only if G contains a clique of size k
- 3-SAT
- SAT

3 The Satisfiability Problem (SAT)

Boolean formula. A Boolean formula over the variables $x_1, \ldots, x_n$ consists of the variables and logical conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$). If A and B are Boolean formulas, then so are $A \wedge B$ and $A \vee B$. A Boolean formula ϕ is satisfiable if $\exists z \in \{0,1\}^n$ such that $\phi(z) = 1$. A Boolean formula over variables $x_1, \ldots, x_n$ is in *Conjunctive Normal Form* (in short, CNF), if it is AND of OR's of variables or their negations. For example,$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$ is a CNF formula. Figure 2 shows a CNF formula $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$, which is satisfied by $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0$, but is not satisfied by $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$.

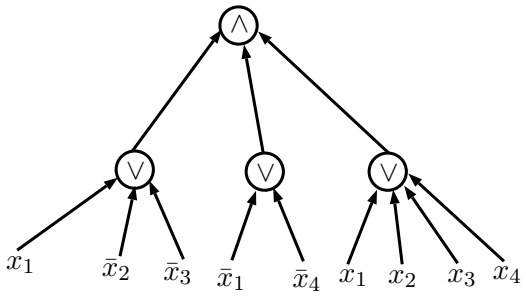

Figure 2: A CNF formula $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4)$

Boolean circuits. For any natural number n, an n input, single output Boolean circuit is a directed acyclic graph, which has

- n Boolean inputs (also called sources) denoted by $x_1, \ldots, x_n$,
- one output (also called sink) which denoted by $C(x)$, where $x = (x_1, \ldots, x_n) \in \{0,1\}^n$.

All the nonsource vertices are called gates. These gates compute the Boolean functions AND, OR and NOT, denoted by $\wedge, \vee$ and $\neg$ respectively. The gates $\wedge$ and $\vee$ has fanin 2, and the $\neg$ gate has fanin 1. Wires of the circuits connect the gates and inputs by carrying the Boolean value 0 or 1. The size of a circuit C, denoted by $|C|$, is the number of gates in it. A Boolean circuit C over $x_1, \ldots, x_n$ *accepts* $\alpha \in \{0,1\}^n$ if and only if $C(\alpha) = 1$. For example, the Boolean circuit in Figure 3 accepts $x_1 = 0, x_2 = 1, x_3 = 1$.

Circuit-SAT. The Circuit-SAT problem is posed as follows. Given a circuit C, does there exist an input $\alpha \in \{0,1\}^n$ such that $C(\alpha) = 1$?

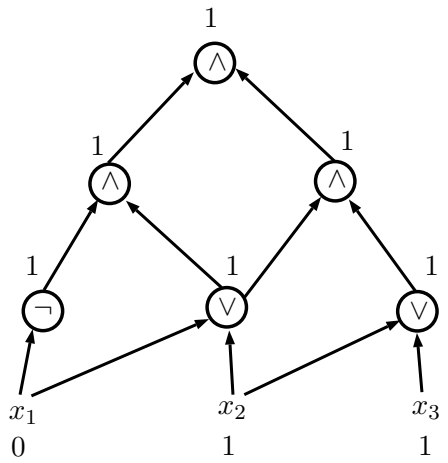

Figure 3: Boolean circuit C which accepts $x_1 = 0, x_2 = 1, x_3 = 1$.

SAT. This is a special case of circuit SAT, where the circuit represents a CNF formula, which has:

- an unbounded fanin $\wedge$ at top
- followed by unbounded fanin $\vee$
- followed by literals.

3-SAT. This is a special case of SAT where all clauses have size ≤ 3.

Theorem 2 *Circuit-SAT is **NP**-hard.*

Proof Sketch.

1. Let L be an arbitrary language. If L is accepted by a polynomial time TM M, then there exists a family of circuits $\{C_n \mid n = 0, 1, \ldots\}$ such that for any n and any x such that $|x| = n$, we have $C_n(x) = M(x)$ and size of C_n = polynomial in n (e.g. n^c where c is a nonnegative integer).

2. Let $L \in \mathbf{NP}$. Then there exist a c and a polynomial-time TM $M(x, y)$, such that $x \in L$ if and only if there exists a y such that $|y| = |x|^c$ and $M(x, y)$ accepts L. Now convert $M(x, y)$ into family of circuits $C_n(x, y)$ where $|x| = n, |y| \leq c|x|^k = cn^k$.

3. Reduction: Given α such that $|\alpha| = n$ we want to find out if $\alpha \in L$. Create circuit $C_n(\alpha, y)$ such that $\alpha \in L$ if and only if there exists β such that $|\beta| \leq n^c$ and $C_n(\alpha, \beta) = 1$. So, $f(\alpha) \to \langle C_n(x, y) \rangle$.

Encoding	Symbol / State
00	0
01	1
10	$\triangleright$
11	$\not b$
000	Designates that head is not here
001	q_1
$\vdots$	$\vdots$
101	q_5

Table 1: Representation of the configuration

More on NP-completeness of Circuit-SAT.

Tableau of M. We consider the following example. Suppose the symbols of M are $0, 1, \not b, \triangleright$, and it has 5 states denoted by q_1, q_2, q_3, q_4, q_5. We define a tableau for M to be a $t(n) \times t(n)$ table. The rows of this tables are configurations of M, with the top row representing the start configuration. The ith row represents the configuration at the ith step of the computation. In the tableau, both the state and symbol under the tape head are represented by a single composite character. For example if M is in state q_3, and the tape contains the symbol 1, then the composite character $q_3 1$ represents both the state q_3 and 1, the symbol under the head (Figure 5). The entry at the ith row and jth column of the tableau is denoted by index (i, j) We represent the configuration as follows. A cell becomes a square of 5 bits, where the first 3 are used to represent the states and the last 2 are used to represent the symbols. The encoding is shown in Table 3, and one example is shown in Figure 5.

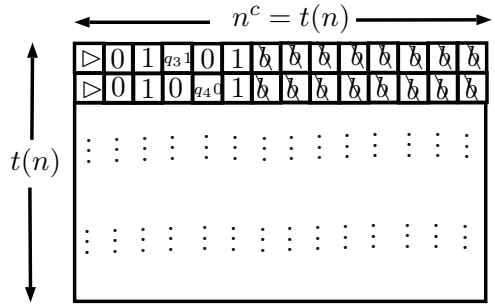

Figure 4: Representation of Circuit-SAT

Computation is local. Note that, cell (i, j) depends only on cell $(i-, j-1), (i-1, j)$, $(i-1, j+1)$ plus its own cell. Each cell has 5 bits,and for each bit has one gate is associated with it. So we can compute any bit of cell (i, j) as a function of 15 pair bits, i.e., OR of ANDs of 15 pair bits. Suppose at cell (i, j) we want to compute bit at position $k \in \{1, \ldots, 5\}$, which depends on bits $l, m, n, p \in \{1, \ldots, 5\}$ of cells $(i, j), (i-1, j-1), (i-1, j)$ and $(i-1, j+1)$ respectively. Clearly, different settings of the bits of these cells will result in cell (i, j) containing a particular bit at kth position. For each of these settings we can construct the circuit so that for each of them the output

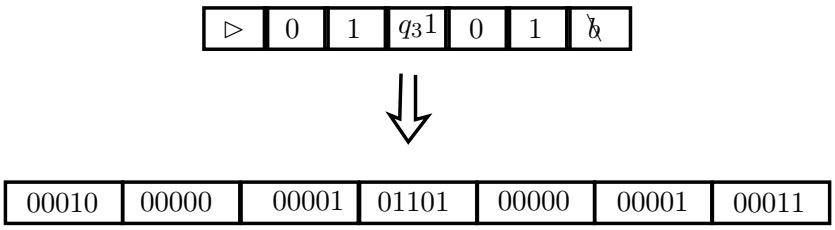

Figure 5: One example of the configuration

is connected to an AND gate, and all such AND gates are connected with an OR gate. Applying this scheme to the cells of the tableau, we can construct all the gates of the circuit. The final output gate of the circuit is one of these gates, associated with the accept state on a cell of the tableau. Note that the circuit is layered. For example the ith layer has $5 \cdot t(n)$ outputs descending the ith configuration of M on x.

Theorem 3 *3-SAT is* **NP-hard.**

Proof sketch. We want to prove that 3-SAT is **NP**-hard, i.e., for any $L \in \mathbf{NP}$ we have $L \leq_p$ 3-SAT. We reduce Circuit-SAT to 3-SAT in polynomial time, which converts a circuit C into a CNF formula ϕ such that all clauses in the formula have size less than or equal to 3. The circuit C is satisfiable if and only if the formula ϕ is satisfiable.

Proof

Configuration of a Boolean circuit. Consider any Boolean circuit C over $x_1, \ldots, x_n$ and m gates . The Boolean output values carried by the wires at the gate outputs are denoted by $g_1, \ldots, g_m$. Our goal is to construct a CNF formula ϕ from the configuration of C. The variables of the formula are denoted by $z_1 = x_1, z_2 = x_2, \ldots, z_n = x_n, z_{n+1} = g_1, \ldots, z_{n+m} = g_m$.

Constructing the formula ϕ. Recall that $P \Rightarrow Q$ is equivalent to $\neg P \vee Q$. Now we describe how to construct the clauses of ϕ.

- Every NOT gate with input z_i and output z_j can be written as

  $$\begin{aligned} &z_j \Leftrightarrow \neg z_i, \\ &\text{which is equivalent to } (\neg z_i \Rightarrow z_j) \wedge (z_i \Rightarrow \neg z_j), \\ &\text{which is equivalent to } (z_i \vee z_j) \wedge (\neg z_i \vee \neg z_j). \end{aligned}$$

  Note that the clauses on the final line will be be satisfied if and only if an assignment is made to the variables z_i and z_j associated with the proper functioning of the NOT gate.

- Every AND gate with input z_i, z_j and output z_k can be written as

  $$\begin{aligned} &z_k \Leftrightarrow (z_i \wedge z_j), \\ &\text{which is equivalent to } (z_k \Rightarrow (z_i \wedge z_j)) \wedge ((z_i \wedge z_j) \Rightarrow z_k), \\ &\text{which is equivalent to } (\neg z_k \vee (z_i \wedge z_j)) \wedge (\neg (z_i \wedge z_j) \vee z_k), \\ &\text{which is equivalent to } (\neg z_k \vee z_i) \wedge (\neg z_k \vee z_j) \wedge (\neg z_i \vee \neg z_j \vee z_k). \end{aligned}$$

The clauses on the final line will be be satisfied if and only if an assignment is made to the variables z_i, z_j and z_k associated with the proper functioning of the AND gate.

- Every OR gate with input z_i, z_j and output z_k can be written as

$$
\begin{aligned}
& z_k \Leftrightarrow (z_i \vee z_j), \\
& \text{which is equivalent to } \neg z_k \Leftrightarrow (\neg z_i \wedge \neg z_j), \\
& \text{which is equivalent to } (\neg z_k \Rightarrow (\neg z_i \wedge \neg z_j)) \wedge ((\neg z_i \wedge \neg z_j) \Rightarrow \neg z_k), \\
& \text{which is equivalent to } (z_k \vee (\neg z_i \wedge \neg z_j)) \wedge (\neg(\neg z_i \wedge \neg z_j) \vee \neg z_k), \\
& \text{which is equivalent to } (z_k \vee \neg z_i) \wedge (z_k \vee \neg z_j) \wedge (z_i \vee z_j \vee z_k).
\end{aligned}
$$

  The clauses on the final line will be be satisfied if and only if an assignment is made to the variables z_i, z_j and z_k associated with the proper functioning of the OR gate.

- We add the clause $z_m = g_m$, where g_m correspond to the output gate of C.

So, ϕ will be the conjunction of all the clauses constructed above, all of which have size less than or equal to 3. So the resultant reduction is that of a 3-SAT. Note that the number variables in the 3-SAT is equal to $n + m$. So the reduction is polynomial in the size of the input.

Justification that the construction works. At first, we show that if we have an assignment x which is accepted by the circuit C, then we can construct an satisfying assignment for the formula ϕ. We take $(z_1, \ldots, z_n) := (x_1, \ldots, x_n)$, and calculate the gate output $g_1, \ldots, g_m$ of C associated with the assignment x, and set $(z_{n+1}, \ldots, z_{n+m}) := (g_1, \ldots, g_m)$. By construction, $\phi(z) = 1$. On the other hand, if a satisfying assignment $z = (x, g)$ for ϕ exists, then x will be accepted by C, because it describes the entire computation of C when the output is one by construction.

[QED]

To explain the proof, we consider an example. Consider the Boolean circuit in Figure 6. To convert it into an 3-SAT problem, we proceed as follows.

- We introduce one new variable per gate
- Clauses that say intermediate variable take on *current value* and that final output is true.

Thus we have,

- $x_1 \wedge \bar{x}_2 \Leftrightarrow z_1$ is equivalent to $(\bar{x_1} \vee x_2 \vee z_1) \wedge (\bar{z_1} \vee x_1) \wedge (\bar{z_1} \vee \bar{x_2})$
- $\bar{x}_2 \wedge x_3 \Leftrightarrow z_2$ is equivalent to $(x_2 \vee \bar{x}_3 \vee z_2) \wedge (\bar{z}_2 \vee x_3) \wedge (\bar{z}_2 \vee \bar{x}_2)$
- $x_2 \vee \bar{x}_1 \Leftrightarrow z_3$ is equivalent to $(\bar{x}_2 \vee z_3) \wedge (x_1 \vee z_3) \wedge (\bar{z}_3 \vee x_2 \vee \bar{x}_1)$
- $z_1 \vee z_2 \Leftrightarrow z_4$ is equivalent to $(\bar{z}_1 \vee z_4) \wedge (\bar{z}_2 \vee z_4) \wedge (\bar{z}_4 \vee z_1 \vee z_3)$
- $z_2 \vee z_3 \Leftrightarrow z_5$ is equivalent to $(\bar{z}_2 \vee z_5) \wedge (\bar{z}_3 \vee z_5) \wedge (\bar{z}_5 \vee z_2 \vee z_3)$
- $z_4 \wedge z_5 \Leftrightarrow z_6$ is equivalent to $(\bar{z}_4 \vee \bar{z}_5 \vee z_6) \wedge (\bar{z}_6 \vee z_4) \wedge (\bar{z}_6 \vee z_5)$

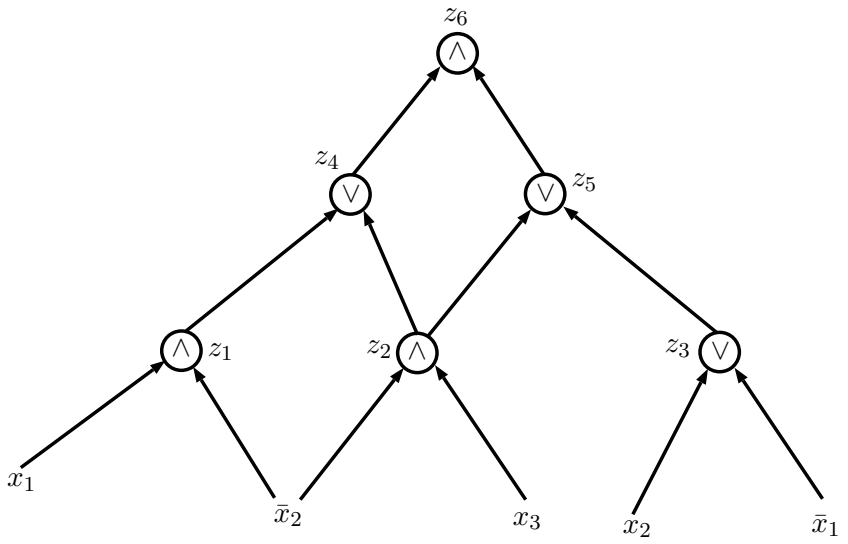

Figure 6: Boolean circuit associated with the Circuit-SAT problem. We want to reduce it to a 3-SAT problem.

- z_6

Note that we have used the fact that $P \Rightarrow Q$ is equivalent to $\neg P \vee Q$.

The size of the circuit is equal to the number of new variables, and number of 3-clauses is equal to 3× size of circuit +1, so f(x,z) SAT if and only if C(x) is a SAT.