CMPUT 675: Computational Complexity Theory **Winter 2019**

Lecture 4 (Jan 17): NP Completeness

Lecturer: Zachary Friggstad *Scribe: Haozhou Pang*

The proofs of all of the results in this section can also be found in [AB09]. Recall the definition of TMSAT:

Definition 1 *TMSAT* $= \{(\alpha, x, 1^n, 1^t) : \exists u \in 0, 1^n s.t.\ M_\alpha(x, u)$ *accepts within* t *steps.*$\}$ *where* M_α *denotes the TM represented by* α*,* 1^n *is the bound on proof length and* 1^t *is the bound on number of steps*

In the previous lecture, we proved that TMSAT is **NP-complete**. However, TMSAT is not a very helpful and interesting **NP-complete** problem since its definition is closely tied to the notion of Turing Machine. In this lecture, we will discuss more examples of natural **NP-complete** problems.

4.1 SAT

Given a finite set of variables $X = x_1, x_2, ..., x_n$, and a finite set of clauses ρ, let ϕ be a boolean formula in CNF form over variables X. Such a boolean formula ϕ is **satisfiable** if there exists some truth assignment that makes all clauses true. Hence, SAT is all instances $\phi = (X, \rho)$, $\exists\ \tau : X \to \{T, F\}$ satisfying all clauses in ρ.

Theorem 1 (Cook / Levin Theorem) *SAT is* ***NP-complete***

Lemma 1 SAT $\in$ ***NP***

Proof. This is obvious: given a proposed solution, we can easily verify that the assignment satisfies all clauses in polynomial time. Hence, SAT $\in$ **NP**. ■

Lemma 2 SAT *is* ***NP-hard***

Proof. Let $L \in$ **NP**. By definition, we know that there is a polynomial time verifier M such that for every $x \in \{0, 1\}^*, x \in L \Leftrightarrow M_\alpha(x, u) = accept$ for some $u \in \{0, 1\}^{p(|x|)}$. We will show L is polynomial-time Karp reducible to SAT, i.e. we will show for each input, I, we specify a boolean formula which is satisfiable if and only if the machine M_α accepts I.

For each tape i of M_α, each cell $c \in [-t, t]$ of tape i, each symbol g, and each time step $\tau \in [0, t]$, we define the following variables:

$X_{i,c,g,\tau}$: True if cell c of tape i contains symbol g at time τ

$Y_{i,h,\tau}$: True if the head of tape i is at position h at time τ

$Z_{q,\tau}$: True if the state of M_α is q at time τ

Thus, we can define a boolean formula to be the conjunction of the following expressions:

For each time step τ, define the following (except those for which $\tau + 1$ would exceed t).

- $\forall q \neq q'$, $\overline{Z_{q,\tau}} \cup \overline{Z_{q',\tau}}$ and $\forall q, Z_{q,\tau}$ (exactly one state at a time)
- $\forall$ tapes i, $\forall h \neq h'$, $\overline{Y_{i,h,\tau}} \cup \overline{Y_{i,h',\tau}}$ and $\vee_h Y_{i,h,\tau}$ (exactly one one head position for each tape at a time.)
- $\forall$ tapes i, $\forall c \neq c'$, $\overline{X_{i,c,g,\tau}} \cup \overline{X_{i,c',g,\tau}}$ and $\vee_g X_{i,c,g,\tau}$ (exactly one one symbol/cell at each time)
- $Z_{q_{start},0}$ (initial state of M_α)
- $\forall$ tapes i, the clause $Y_{i,0,0}$ (initial positions of heads of all tapes)
- $\forall$ cells c of all tapes except the two input tapes, the clause $X_{i,c,\Box,0}$ (initial contents of tapes except input tapes are all blank symbols)
- $\forall$ cells $0 \leq c < |x| - 1$ of the first input tape (say tape 0), $X_{0,c,x_c,0}$ (the first input tape is initialized to x)
- $\forall$ symbols $g_1, ..., g_k$, $\forall$ head positions $h_1, ..., h_k$, $\forall$ states q, let $g'_1, ..., g'_k$ denote the contents of cells at old position, $h'_1, ..., h'_k$ denote the new head positions, and q' denote the new state. Then

  $$(\textstyle\bigwedge_{i=1}^{k} X_{i,h_i,g_i,\tau}) \wedge (\bigwedge_{i=1}^{k} Y_{i,h_i,\tau}) \wedge (Z_{q,\tau}) \Rightarrow (\bigwedge_{i=1}^{k} X_{i,h_i,g'_i,\tau}) \wedge (\bigwedge_{i=1}^{k} Y_{i,h'_i,\tau}) \wedge (Z_{q',\tau})$$

  (All possible transitions at step τ when head of tape i is at position h_i and current state is q.)
- $\forall g \neq g'$, $X_{i,c,g,\tau} \wedge X_{i,c,g',\tau+1} \Rightarrow Y_{i,c,\tau}$ (If a cell is not where the head is at a given time, it does not change)
- $Z_{q_{accept},t}$ (Must accept within t steps)

The implications can be turned into CNF instances. Generally, one can convert any boolean expression to a CNF with, perhaps, exponential size in the original expression. But the implications involve only a constant (independent of x) number of variables so this still results in a polynomial-time reduction. For a more streamlined approach, use facts like $a \wedge b \Rightarrow c \wedge d$ is equivalent to $(\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee d)$.

If the boolean formula defined above (the conjunction of all the sub-expressions) is satisfiable, then there is a accepting computation of M_α on input I that follows the steps indicated by each sub-expression. Conversely, if there is a accepting computation of M_α on input I, Then the boolean formula must be satisfiable since all the sub-expressions are extracted from the definition of M_α. This can be proven carefully by induction on τ. ■

In some sense, in the CNF instance above once the proof string values y are determined (i.e. once we have set $X_{1,c,g,0}$ for each cell c of the the proof string, say 1) then all other variables are determined by the implications: i.e. are determined by "simulating" the verifier on input (x, y).

Proof of Theorem 1. Follows from Lemma 1 and Lemma 2.

The proof of Theorem 1 shows that any **NP** problem can be reduced to SAT problem in polynomial time. That means if there exists a deterministic Turing Machine which can solve SAT in polynomial time, then all problems in **NP** can be solved in polynomial time, which implies **P**=**NP**. ■

4.2 Reducing SAT to 3SAT

The 3SAT problem is similar to SAT problem but each clause in the boolean formula has exactly 3 literals over three distinct variables. We show that 3SAT is **NP-complete** by showing SAT is polynomial time reducible to 3SAT. Namely, we will propose a transformation that maps each CNF formula φ into a 3CNF formula ψ such that ψ is satisfiable if and only if φ is satisfiable.

Lemma 3 *SAT $\leq_p$ 3SAT*

Proof. For any clause C_i in a SAT formula $C_1 \wedge C_2 \wedge ... \wedge C_n$, there can be only four cases:

- Case 1: It contains three literals. Then there is no reduction needed for this case.
- Case 2: It contains only one literal, say $C_i = l_i$. We introduce two new variables a_1 and a_2, then we can replace C_i by the conjunction of following clauses $(l_i \vee a_1 \vee a_2) \wedge (l_i \vee \overline{a_1} \vee a_2) \wedge (l_i \vee a_1 \vee \overline{a_2}) \wedge (l_i \vee \overline{a_1} \vee \overline{a_2})$, which is logically equivalent to C_i. Hence, C_i can be reduced to the conjunction of clauses and each of them contains exactly 3 literals.
- Case 3: It contains two literals, say $C_i = (l_{i_1} \vee l_{i_2})$. Similarly, we introduce a new variable b, then we can replace C_i by $(l_{i_1} \vee l_{i_2} \vee b) \wedge (l_{i_1} \vee l_{i_2} \vee \overline{b})$.
- Case 4: It contains k literals, where $k \geq 4$. When $k = 4$, say $C_i = (l_{i_1} \vee \overline{l_{i_2}} \vee \overline{l_{i_3}} \vee l_{i_4})$, we introduce a new variable z and replace C_i with the pair of clauses $C_{i_1} = l_{i_1} \vee \overline{l_{i_2}} \vee z$ and $C_{i_2} = l_{i_1} \vee \overline{l_{i_2}} \vee \overline{z}$. Obviously, if C_i is true, then there is an assignment to z that satisfies both C_{i_1} and C_{i_2}; and if C_i is false, then no matter what value we assign to z, either C_{i_1} or C_{i_2} will be false. By using induction, we can easily show that any clause C of size $k \geq 4$ can be changed into an equivalent set of clauses of size exactly 3.

The analysis above yields a polynomial-time transformation of a CNF formula into an equivalent 3CNF formula, as required. Hence, 3SAT is in **NP-complete**. ■

4.3 Reducing 3SAT to Independent Set

Consider the problem of INDEPENDENT SET (IS). Given a graph $G(V, E)$ and an integer m, is there some $I \subseteq V$ s.t. no two vertices $u, v \in I$ form an edge and $|I| = m$.

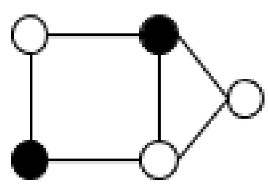

Figure 4.1: An example of an independent set of size 2

Lemma 4 INDEPENDENT SET *is **NP-complete**.*

Proof. We prove INDEPENDENT SET is **NP-complete** by showing 3SAT $\leq_p$ INDEPENDENT SET. Let ϕ be a SAT instance with m clauses. We say two literals conflict if they are over the same variable but with different signs (eg. x_1 and $\overline{x_1}$ conflict). For any clause $C_i, 1 \leq i \leq m$, say C_i has literals l_1^i, l_2^i, l_3^i. We construct a graph $G = (V, E)$ as follows:

- $V = \{(i,1), (i,2), (i,3), \forall i : 1 \leq i \leq m\}$
- $E = \{\{(i,1),(i,2)\}, \{(i,1),(i,3)\}, \{(i,2),(i,3)\} : \forall 1 \leq i \leq m\}$
- $U = \{\{(i,a),(j,b)\} : l_a^i$ and l_b^j conflict$\}$.

Thus, G contains m triangles, one for each clause in ϕ, with each node representing one of the literals in the clause. Also, we connect two nodes in different triangles if they represent literals where one is the negation of the other. For example, the boolean formula $(x \vee y \vee \overline{z}) \wedge (w \vee \overline{y} \vee z) \wedge (w \vee \overline{x} \vee y)$ can be converted to the following graph:

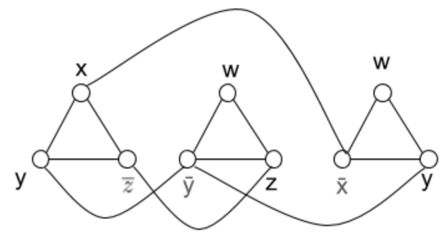

Figure 4.2: the graph constructed from the above boolean formula

Claim 1 *ϕ is satisfiable if and only if there exists an ind. set of size m in the graph.*

The claim is easy to prove. First, if ϕ is satisfiable, then we pick one node (i, e) from each triangle $1 \leq i \leq m$ s.t. l_e^i is true under the satisfying assignment. The nodes corresponding to these literals form an independent set of size m since the only edges among them would connect nodes that are negations of each other, hence it will never be the case that both of them have been selected.

Conversely, suppose I is an independent set of size m, then $|I \cap \{(i, 1), (i, 2), (i, 3)\}| = 1$, $\forall 1 \leq i \leq m$. For each node $(i, e) \in I$, set the variable in l_e^i to satisfy the literal. Note this will not attempt to set different truth values to the same variable because I is an independent set. Clearly even this partial assignment satisfies ϕ, so extending it to a full assignment by arbitrarily setting the remaining variables will satisfy ϕ. Therefore, we can reduce a 3CNF formula to an INDEPENDENT SET instance in polynomial time, so INDEPENDENT SET problem is in **NP-complete**. ■

4.4 Co-NP

Definition 2 ***Co-NP*** $= \{L : \overline{L} \in \textbf{\textit{NP}}\}$

Note that **Co-NP** is not the complement of **NP**. Actually, **Co-NP** and **NP** have a non-empty intersection since every language in **P** is in **Co-NP** $\cap$ **NP**. The following graph will give a better visualization of the relationships of the complexity classes that have been discussed so far:

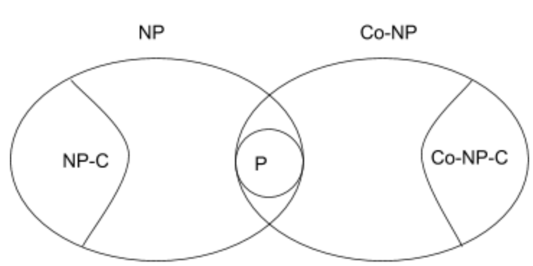

Figure 4.3: The relationship of different complexity classes

An alternative definition of **Co-NP**:

Definition 3 *L is in* ***Co-NP*** *if there exists a p.t.v. M s.t.*

- $\forall x \in L, \forall u \in \{0,1\}^* :$ *M(x,u) = accept*
- $\forall x \notin L, \exists u \in \{0,1\}^* :$ *M(x,u) = reject*

As with **NP**-completeness, a language L is co-**NP**-complete if $L \in$ co-**NP** and $L' \leq_p L$ for every $L' \in$ co-**NP**.

Claim 2 TAUTOLOGY *is* ***Co-NP-complete***

Proof. A boolean formula ϕ is a tautology if it is satisfied by every assignment. Clearly, TAUTOLOGY is in **Co-NP** by definition 3, so we need to show for every $L \in$ **Co-NP**, $L \leq_p$ TAUTOLOGY. Let $L \in$ **Co-NP**, then $\overline{L} \in$ **NP**, there exists a polynomial time transformation f() that can reduce $\overline{L}$ to SAT (from section 4.1). That is: $\forall x \in \{0,1\}^x, x \in L \Leftrightarrow f(x)$ is not satisfiable, so we simply output $\overline{f(x)}$. ■

References

AB09 S. ARORA and B. BARAK, Computational Complexity: A Modern Approach *Cambridge University Press*, 2009