CS221: Computational Complexity Prof. Salil Vadhan

Lecture ???: Completeness in P, NP and NL

10/07 *Scribe: Loizos Michael*

Contents

1 Announcements

- For questions send email to cs225@eecs.harvard.edu.
- Scribe notes should be accompanied by text explaining the material. It is not sufficient to simply copy the material from the blackboard.
- Section today at 6:00pm, room MD323.
- Reading: Papadimitriou §7.3, §8.1, §8.2, §9 and Theorem 16.2.

2 P-Completeness

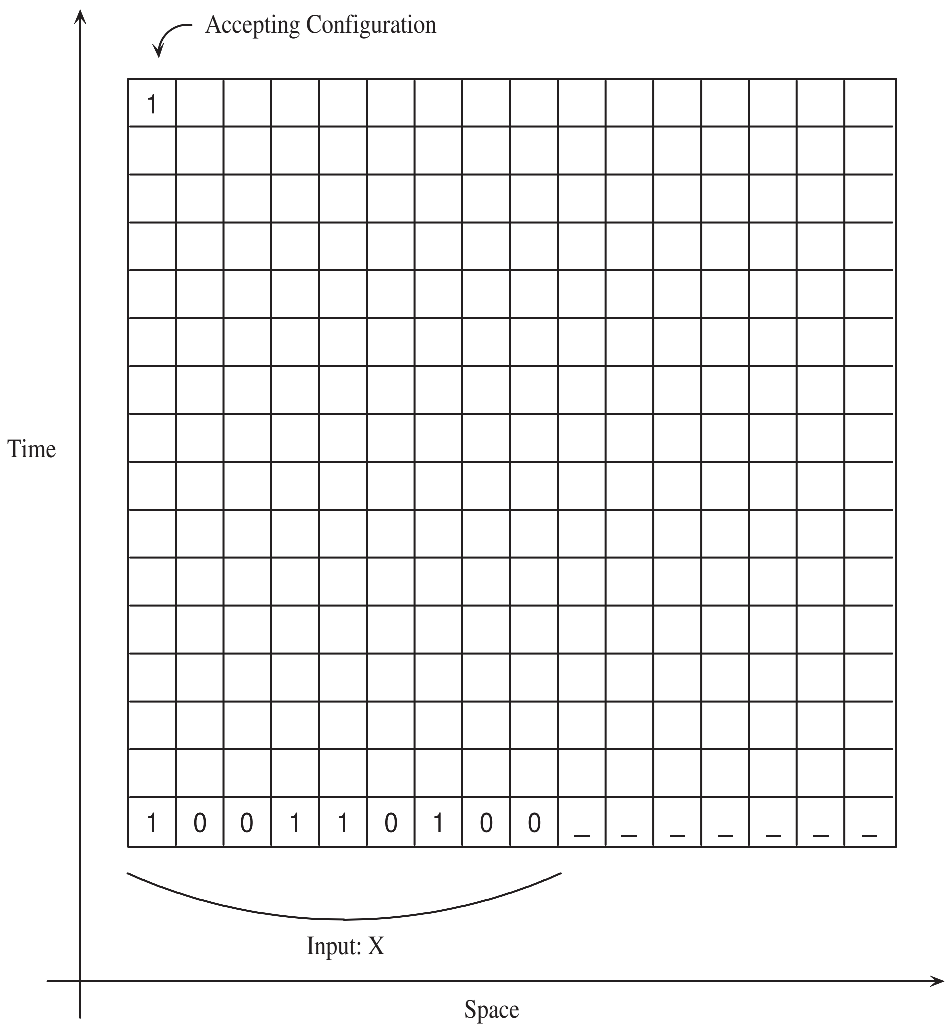
We remind from the previous lecture that the CIRCUIT-VALUE problem is the following: "Given a boolean circuit C with no inputs, decide if C evaluates to 1". This is formally described below.

$$\text{CIRCUIT-VALUE} \;=\; \{C \mid C \text{ is a boolean circuit with no inputs} \wedge C \equiv 1\}$$

We continue to prove the following theorem. Note that this theorem is indeed useful, since there are complexity classes below **P**, for instance **L**.

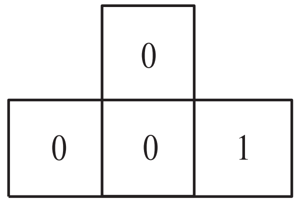
Theorem 1 CIRCUIT-VALUE *is* **P**-*complete.*

Proof: We will reduce every language $L \in \mathbf{P}$ to the CIRCUIT-VALUE problem. Without loss of generality, suppose that a single-tape Turing Machine M decides the language L in polynomial time. We show below how to transform every input x of M into a boolean circuit C, such that $x \in L \Leftrightarrow C \equiv 1$. We do so, through the use of a two-dimensional tableau as the one below.



The X-axis of the tableau represents space and the Y-axis of the tableau represents time. In particular, space corresponds to the tape of the Turing Machine M and time corresponds to the time taken for M to perform a computation on a given x. In addition, both dimensions are finitely long and of polynomial size with respect to $|x|$, since $L \in \mathbf{P}$. Clearly, each cell in the tableau corresponds to one position of the tape of M on a given computation step. Therefore, the input x is naturally represented at the bottom left corner of the tableau and the output (assuming that it is either 1 or 0) is naturally represented at the upper left corner cell of the tableau.

Note that, a correct computation of a Turing Machine can be represented as a set of constraints, that entail the value to be computed on the next step, from a limited set of current values. In fact, only the three cells (i.e. positions of the cursor of M on given computation steps) below a given cell can affect the value of the latter cell. Therefore, the constraints implied by the Turing Machine relate these four cells. Since the constraints only depend on the Turing Machine, then the cells in each reverse-T quadruplet (see figure below) of cells are related with each other under the same constraints.



Observe now that a set of constraints can be represented as a boolean formula and specifically a CNF boolean formula, where only negations, conjunctions and disjunctions are used. We can therefore construct a boolean circuit that represents this CNF formula, where only `NOT`, `AND` and `OR` gates are used. In addition, every `AND` or `OR` gate with more than two inputs can be represented as a set of `AND` and `OR` gates respectively, with exactly two inputs per gate. In total, every set of constraints can be represented as a boolean circuit of `NOT`, `AND` and `OR` gates with exactly two inputs per binary gate.

Based on the comments above, we can "fill in" the tableau with a boolean circuit, by simply copying for each reverse-T quadruplet the boolean circuit that corresponds to the set of constraints implied by the Turing machine. For each specific instance of input x, the boolean circuit has constant input and satisfies therefore the respective requirement.

It is easy now to see that a given input x is in L, if and only if the Turing Machine M accepts x (and hence outputs 1), if and only if the boolean circuit value is 1 (since the boolean circuit faithfully represents the computation of M), if and only if $C \in$ CIRCUIT-VALUE . This concludes the reduction and the fact that CIRCUIT-VALUE is **P**-complete follows. ■

Some commends can be made relating to the proof above. First observe that the computation of a Turing Machine is local, in the sense that it depends only on a small fixed number of adjacent cells in the tableau (space-time structure) to compute the value of a given cell. In fact, this observation carries on to any reasonable computational model. The second think to observe is that, boolean circuits can simulate Geil computations (i.e. Turing Machines), where the time taken during the computation reflects on the size of the boolean circuit that simulates it. We can view boolean circuits, therefore, as Turing Machines, that work however on a finite set of inputs. The last think to observe is that the boolean circuit of the proof, can be in fact constructed in *logspace*. This in fact is critical, since we defined a reduction to be a *logspace* reduction. The reason this holds is that, we can simply construct the respective boolean circuit for one reverse-T quadruplet of cells and then just copy that to all the other positions.

The proof that CIRCUIT-VALUE is **P**-complete implies the following self-evident corollary.

Corollary 2 CIRCUIT-VALUE $\in \mathbf{L} \Leftrightarrow \mathbf{P} = \mathbf{L}$.

Having proved the existence of an **P**-complete problem, we can now mention a few other problems known to be **P**-complete.

- MONOTONE-CIRCUIT-VALUE, which is the same as the CIRCUIT-VALUE problem, but `NOT` gates are not allowed. Note that monotone circuits cannot compute all problems in **P**, exactly due to their monotonicity.

- LINEAR-PROGRAMMING.
- MAX-FLOW, where the edges of the network have capacities.
- HORN-FORMULA-SAT, which is a special instance of the SAT problem.
- BOUNDED-HALTING, which is defined as BOUNDED-HALTING $= \{(M, x, 1^n) \mid M$ holds on x within n steps$\}$. This problem can be actually proven to be **P**-complete, without using reduction from the CIRCUIT-VALUE problem.

3 NP-Completeness

In this section we define a variation of the CIRCUIT-VALUE problem that we have used in the previous section. The CIRCUIT-SATISFIABILITY problem is the following: "Given a boolean circuit C, decide if there exists an assignment $\overline{a}$ to its input gates, such that C($\overline{a}$) evaluates to 1". This is formally described below.

$$\text{CIRCUIT-SATISFIABILITY} \;=\; \{C \mid C \text{ is a boolean circuit} \wedge \exists \overline{a} : C(\overline{a}) \equiv 1\}$$

Note that here the boolean circuits have input gates, in contrast to the boolean circuits in the CIRCUIT-VALUE problem. In addition this problem asks whether "there exists an assignment" that makes the boolean circuit evaluate to 1, rather than whether "the static circuit" evaluates to 1. Hence, while in the latter case we can simply evaluate the circuit and check whether it evaluates to 1 or not in polynomial time, here we cannot employ that approach.

As in the previous section, we continue to prove the following theorem. Due to the similarity with the corresponding theorem in the previous section, we will omit much of the detail in the proof.

Theorem 3 CIRCUIT-SATISFIABILITY *is* **NP**-*complete.*

Proof: Clearly, CIRCUIT-SATISFIABILITY is in **NP** since, given an assignment to a circuit's inputs, we can verify in polynomial time that the circuit evaluates to 1. It remains to show that the problem is also **NP**-hard.

As before, we will reduce every language $L \in \mathbf{NP}$ to the CIRCUIT-SATISFIABILITY problem. Since $L \in \mathbf{NP}$ then there exists a polynomially computable relation R and a polynomial p such that $\forall x[x \in L \Leftrightarrow \exists y : |y| = p(|x|) \wedge (x, y) \in R]$. Since $R \in \mathbf{P}$, without loss of generality, suppose that a single-tape Turing Machine M_R computes R in polynomial time.

We now proceed as in the proof of 1 to construct a boolean circuit $C_{R,x}$ that corresponds to the Turing Machine M_R on input x. The boolean circuit $C_{R,x}$ is represented in the figure below.

FIGURE

Note that the boolean circuit has now input gates, which effectively account for the unknown value of y. For a given input y, by construction the boolean circuit $C_{R,x}$ outputs 1 if and only if $(x, y) \in R$. We now simply observe that $x \in L$, if and only if $\exists y : (x, y) \in R$, if and only if $\exists y : C_{R,x}(y) \equiv 1$, if and only if $C_{R,x} \in$ CIRCUIT-SATISFIABILITY . This concludes the reduction and the fact that CIRCUIT-SATISFIABILITY is **NP**-complete follows. ■

Having proved that CIRCUIT-SATISFIABILITY is an **NP**-complete problem, we can show the following theorem, due to Cook and Levin. We remind that the 3-SAT problem is the following.

$$\text{3-SAT} \ = \ \{\varphi \mid \varphi \text{ is a 3-CNF boolean formula} \wedge \exists \overline{a} : \varphi(\overline{a}) \equiv 1\}$$

3-CNF formulas are the formulas that are conjunctions of three disjunctions of literals, where a literal is either a variable or its negation. An example of a 3-CNF formula would be: $(X_1 \vee \neg X_2 \vee X_3) \wedge (X_2 \vee X_4 \vee \neg X_5) \wedge (\neg X_6 \vee \neg X_1 \vee X_4)$.

Theorem 4 3-SAT *is* **NP**-*complete.*

Proof: Clearly, 3-SAT is in **NP** since, given an assignment to a 3-CNF formula, we can verify in polynomial time that the formula is satisfied. It remains to show that the problem is also **NP**-hard.

We will proceed by reducing 3-SAT from $Csat$. Given a boolean circuit C over a set of inputs $X_1, X_2, ..., X_n$, we will construct a formula φ over the variables $X_1, X_2, ..., X_n, X_{n+1}, X_{n+2}, ..., X_m$, as follows. Variables $X_1, X_2, ..., X_n$ will stand for the input nodes of the boolean circuit and variables $X_{n+1}, X_{n+2}, ..., X_m$ will stand for the remaining nodes of the boolean circuit. The values of the variables $X_{n+1}, X_{n+2}, ..., X_m$ are therefore completely depended on the values of the variables $X_1, X_2, ..., X_n$. We will capture this fact through defining proper clauses for φ.

Firstly, φ contains the clause X_m, which represents the output of the boolean circuit. Notice by having X_m as a clause, we require that in every satisfying assignment of φ, the variable X_m is set to true, analogous to the requirement that the boolean circuit should evaluate to 1. The fact that this clause is not the disjunction of three variables can be easily resolved by adding two dummy variables and expanding this single variable clause to four disjunctive clauses of three variables.

Now, each `AND` and `OR` gate connects three variables in a certain way. In each case, there exists exactly four assignments to the variables that violate the logic of the gate. That is to say, the type of the gate and its two inputs, uniquely specify the gate's output. Therefore, for each gate and input configuration, there exists exactly one output that contradicts the expected output. Since there are exactly four possible input configurations for each gate, then there are exactly four assignments to the three variables involved that are not acceptable. For each such assignment, we add a clause in φ that rules that assignment out, by requiring that at least one of the three variables does not take the truth value specified in that assignment. So each clause is a disjunction of the three variables.

Formula φ is the conjunction of all the clauses defined above. Notice that, $\exists \overline{a} = (a_1, a_2, ..., a_n) : C(\overline{a}) = 1$, if and only if $\exists \overline{a} = (a_1, a_2, ..., a_n, a_{n+1}, a_{n+2}, ..., a_m) : \varphi(\overline{a}) = 1$. This concludes the reduction and the fact that 3-SAT is **NP**-complete follows. ■

An interesting thing about the proof above is the technique we used of introducing extra variables during the reduction. The values of these extra variables can be thought as if they are non-deterministically guessed.

The proof that 3-SAT is **NP**-complete implies the following self-evident corollary.

Corollary 5 3-SAT $\in \mathbf{P} \Leftrightarrow \mathbf{NP} = \mathbf{P}$.

Having proved the existence of two **NP**-complete problems, we can now mention a few other problems known to be **NP**-complete.

- TRAVELING-SALESMAN.
- CLIQUE.
- 3-COLORING.
- SHORT-PROOF.
- LINEAR-ERROR-CORRECTING-CODES-DECODING.
- PROTEIN-FOLDING.

4 NL-Completeness

Having studies the completeness in time classes, we now turn our attention to space classes. In correspondence to the polynomial time classes, in this section we study the logarithmic space classes. Unlike the deterministic polynomial time, however, deterministic logarithmic space is not interesting for studying. Recall that our notion of reduction between languages is actually logspace reduction. From this we deduce that every non-trivial (i.e. other than $\emptyset$ and σ^*) language in **L** is logarithmic reducible to every other non-trivial language in **L** and is, then, **L**-complete.

When we move to non-deterministic logarithmic space, things stop being trivial. As in the previous sections, we will prove that a specific problem is complete within this class. Recall that the REACHABILITY problem is the following: "Given a directed graph G and two vertices s and t of G, decide if there exists a path in G from vertex s to vertex t". This is formally described below.

$$\text{REACHABILITY} = \{(G, s, t) \mid G = (V, E) \text{ is a directed graph}, s, t \in V \wedge \exists \; path \text{ in } G \text{ from } s \text{ to } t\}$$

We continue to prove the following theorem. As in the previous sections, this is a completeness result with respect to the problem that we have chosen.

Theorem 6 REACHABILITY *is* **NL**-*complete.*

Proof: First we will show that REACHABILITY is in **NL**. We give below an algorithm that given an input (G, s, t) decides in non-deterministic logarithmic space whether $(G, s, t) \in$ REACHABILITY .
Given an input (G, s, t):

- **Set** $v := s$.
- **Repeat** the following:
  - **If** $v = t$, **then** halt and accept.
  - **Else**, **set** v to be a non-deterministically chosen neighbor of v.
- **If** the algorithm runs for more than $|V|$ steps, where $G = (V, E)$, **then**, halt and reject.

This algorithm guaranties to find a path between s and t if and only if such a path exists. The space used by the algorithm is the space for storing a single vertex v of the graph. This vertex can be stored in space $log(|V|)$, which is less than logarithmic to the input size of the algorithm.

It remains to show that REACHABILITY is **NL**-hard. We will show this in the next lecture, by reducing from every language $L \in \mathbf{NL}$ to REACHABILITY . ■