6.5620/6.875/18.425: Cryptography and Cryptanalysis **September 15, 2023**

Recitation 2: Background on Complexity Theory

Instructor: Vinod Vaikuntanathan **TAs:** Chirag Falor, Neekon Vafa, and Hanshen Xiao

Contents

1 Models of Computation

Throughout this course, we will consider different models of computation. We mention a few of them during this recitation:

1.1 Turing Machines

- We won't formalize it here, but you should think of it as having three tapes: input tape, a work tape, and an output tape. Wikipedia, for example, has more formal details.
- The description of a Turing machine is independent of the input length, so has size $O(1)$.
- You always have a pointer to some cell in the work tape, and in each time step, it can move to the left, stay in the same position, or move to the right.
- The running-time of a Turing machine is how many time steps it takes before changing to its "accept" state, and the space usage of the Turing machine is how much of the work tape was accessed during the computation.

1.2 RAM (or Word RAM)

- You can think of this as a Turing machine with a fixed sized tape (say, length n), with the additional ability of being able to move the pointer to any position of the work tape in 1 time step.
- Another way to think of it is as a small-space Turing machine that outputs instructions from the set

$$(\{\mathsf{read}\} \times [n]) \bigcup (\{\mathsf{write}\} \times [n] \times \{0,1\}) .$$

That is, in each time-step, one can specify $i \in [n]$, and either read the bit at index i, or write a new bit $b \in \{0,1\}$ to index i. More generally, each position in the tape can be filled with a *word* of size w, i.e., an element of $\{0,1\}^w$ instead of just a bit, which would correspond to $w = 1$.

1.3 Circuits

- Unlike Turing machines, circuits have a fixed input and output length, e.g., $C : \{0,1\}^n \to \{0,1\}^m$.
- Circuits have input wires, output wires, and intermediate wires, each containing bit values.
- Wires are connected via *gates*, which are chosen from some gate set. A common gate set choice is $\{\mathsf{AND}, \mathsf{OR}, \mathsf{NOT}\}$, which is universal, meaning that all functions $f : \{0,1\}^n \to \{0,1\}^m$ can be expressed using just these gates. Another common gate set choice is $\{\mathsf{NAND}\}$, which is also universal.
- For circuits, the typical complexity measures we consider are *size*, the number of gates in the circuit, and *depth*, the length of the longest path from an input to output wire. If there is no upper bound on the size of the circuit, for example, circuits can compute *any function* from $\{0,1\}^n$ to $\{0,1\}^m$.

2 Complexity Classes

Formally, complexity classes typically contain *decision problems* or *languages*, which can be thought of as functions $D : \{0,1\}^* \to \{0,1\}$, or equivalently, as subsets $L \subseteq \{0,1\}^*$.

- The class P is defined as the set of languages L such that there exists a Turing machine M that runs in polynomial time such that $x \in L \iff M(x) = 1$.

Example 1: Language in P

Let $L = \{G = (V,E), v_1 \in V, v_2 \in V :$ there is a path from v_1 to v_2 in $G\}$. Here, $G = (V,E)$ denotes a graph with vertex set V and edge set E. Path finding runs in time $O(|V| + |E|)$, which is linear in the size of the input. Therefore, $L \in \mathsf{P}$.

- The class NP is defined as the set of languages L such that there exists a Turing machine V and a polynomial function $p(n)$ such that V runs in time at most $p(|x|)$ for inputs x, and $x \in L \iff \exists y \in \{0,1\}^{p(|x|)}$ such that $V(x,y) = 1$.

Example 2: Language in NP

Let $L = \mathsf{SAT} = \{\varphi : \exists y_1, \cdots, y_n \in \{0,1\}, \varphi(y_1, \cdots, y_n) = 1\}$, where φ is a Boolean formula on n variables. The algorithm V will simply evaluate and output $\varphi(y)$.

- The class BPP is defined as the set of languages L such that there exists a *randomized* Turing machine M that runs in polynomial time such that $x \in L \implies \Pr_r[M(x;r) = 1] \geq 2/3$, and $x \notin L \implies \Pr_r[M(x;r) = 1] \leq 1/3$, where $|r| \leq p(|x|)$ for some polynomial function $p(n)$.

Example 3: Language in BPP

Let $L = \{$Low-degree polynomial $p(x_1, \cdots, x_n) :$ p is functionally equivalent to the all 0 function$\}$. The algorithm here is to try random input points. The Schwartz–Zippel lemma says that this works with high probability. (Don't worry if this example is confusing.)

Once nice property (that we actually proved in our last recitation) is that the probability thresholds $1/3$ and $2/3$ can equivalently be made $1/2^n$ and $1 - 1/2^n$, respectively, thanks to Chernoff bounds.

3 Reductions

How can we relate different languages to each other? Ideally, we would like to say that some languages are computationally easier or harder than others. This can be formalized by the notion of a *reduction.* A reduction from language L_1 to language L_2 roughly says that L_2 is as hard as L_1 (if not harder). Sometimes, this is denoted as $L_1 \leq L_2$, where the inequality here can be thought of as comparing "hardness" of the decision problems. A bit more concretely, this will mean that any algorithm for L_2 can be used to construct an algorithm for L_1 with similar efficiency.

3.1 Many-One/Mapping/Karp Reductions

The standard notion of a reduction that you may have seen in previous classes is a *many-one/mapping/Karp* reduction (goes by different names depending on context). This will be some efficient (often, polynomial-time computable) algorithm R such that $x \in L_1 \iff R(x) \in L_2$. If such an R exists, and an efficient algorithm for L_2 exists, then an efficient algorithm for L_1 exists: first, compute $R(x)$, and then, use your algorithm for L_2! Here is a central theorem in complexity theory:

Theorem 1 (Cook-Levin Theorem) SAT *is* NP*-complete. That is,* $\mathsf{SAT} \in \mathsf{NP}$*, and for all languages* $L \in \mathsf{NP}$*, there is a Karp reduction from* L *to* SAT*.*

3.2 Turing/Cook Reductions

A more general notion of a reduction is a *Turing/Cook* reduction. This will be some algorithm R that could make many calls to an algorithm for L_2 and use that to solve L_1. More formally, a reduction R will have *oracle access* to L_2, meaning the reduction R can produce many values y and check, in $O(1)$ time, whether $y \in L_2$. If such an efficient R (assuming its L_2 answers are magically computed in $O(1)$ time) exists and solves L_1, then we say there is a Turing/Cook reduction from L_1 to L_2. Concretely, this means that if there is an efficient algorithm for L_2, then there is one for L_1, since the reduction can replace the L_2 oracle with the algorithm for L_2!

In cryptography, this notion of reduction is more important, since it more broadly captures the sentiment of "if there is an algorithm for L_2, then there is an algorithm for L_1", or equivalently, "if there is no algorithm for L_1, then there is no algorithm for L_2." More than that, R will often be randomized and not always succeed, so we will consider complex types of reductions in this course.

4 Problems

4.1 Search-to-Decision Reduction for NP

Suppose we are in a world where $\mathsf{P} = \mathsf{NP}$. Formally, this means that $\mathsf{SAT} \in \mathsf{P}$, so we have some polynomial-time machine M that solves SAT, in the sense that $M(\varphi) = 1 \iff \exists y_1, \cdots, y_n \in \{0,1\}$ such that $\varphi(y_1, \cdots, y_n) = 1$. What if we want more than that? What if we also want to output a satisfying assignment $y_1, \cdots, y_n$ if such an assignment exists? In general, we call such a reduction a *search-to-decision reduction*, because we are reducing a search problem (namely, finding a satisfying assignment) to a decision problem (namely, deciding whether a satisfying assignment exists). We will see other search-to-decision reductions throughout this course.

Problem 1: Show that if $\mathsf{SAT} \in \mathsf{P}$, then there is a polynomial-time algorithm A such that on input a Boolean formula φ, correctly outputs either (a) that φ is unsatisfiable, or (b) some satisfying assignment to φ. More generally, show that the same holds if given a SAT *oracle* instead of a polynomial time algorithm for SAT.

Solution 1: We describe the algorithm A as follows. On input φ, first we use the SAT oracle to check if $\varphi \in \mathsf{SAT}$. If not, A outputs that φ is not satisfiable. Otherwise, we know that there exists $y_1, \cdots, y_n \in \{0,1\}$ such that $\varphi(y_1, \cdots, y_n) = 1$. Next, we define $\varphi_1 = \varphi \wedge y_1$ and run φ_1 on the SAT oracle. Let the oracle output be $b_1 \in \{0,1\}$. If $b_1 = 1$, then *we know* that there exists a satisfying assignment to φ such that $y_1 = 1$, so we might as well set $y_1 = 1$. If $b_1 = 0$, then there is no satisfying assignment to φ such that $y_1 = 1$, which means that $y_1 = 0$, since we know there is some value of y_1 (that is not 1) that is included in a satisfying assignment. In either case, we know $y_1 = b_1$ is a valid setting of y_1. We then define $\varphi_2 = \varphi \wedge (y_1 = b_1) \wedge y_2$, and run φ_2 in the SAT oracle to get a bit b_2. By the same reasoning we know $y_2 = b_2$ is a valid assignment. We then define $\varphi_3 = \varphi \wedge (y_1 = b_1) \wedge (y_2 = b_2) \wedge y_3$, and continue all the way down until we have a full satisfying assignment $y = y_1, \cdots, y_n$. This runs in $O(n)$ time, with $O(n)$ calls to the SAT oracle.

4.2 PRG Reductions

Problem 2: Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be a PRG. Show that $G' : \{0,1\}^{n+1} \to \{0,1\}^{n+2}$ defined by $G'(x||b) = G(x)||b$ is a PRG.

Solution 2: We will proceed by contradiction. Suppose that G' is not a PRG. Then, we have a PPT distinguisher D', for which (possibly by negating the distinguisher) there exists a polynomial function p and infinitely many n such that

$$\Pr_{y \leftarrow \{0,1\}^{n+1}}[D'(G'(y)) = 1] - \Pr_{z \leftarrow \{0,1\}^{n+2}}[D'(z) = 1] \geq 1/p(n).$$

By using the definition of G' and rewriting, this means

$$\Pr_{x \leftarrow \{0,1\}^n, b \leftarrow \{0,1\}}[D'(G(x)||b) = 1] - \Pr_{z \leftarrow \{0,1\}^{n+1}, b \leftarrow \{0,1\}}[D'(z||b) = 1] \geq 1/p(n),$$

for infinitely many n. Now define the distinguisher $D(z) = D'(z||b)$ for uniformly randomly sampled $b \leftarrow \{0,1\}$. Since D' is PPT, so is D. This immediately implies

$$\Pr_{x \leftarrow \{0,1\}^n}[D(G(x)) = 1] - \Pr_{z \leftarrow \{0,1\}^{n+1}}[D(z) = 1] \geq 1/p(n),$$

for infinitely many n. This shows that D is a distinguisher for G, which is a contradiction, since we have assumed that G is a PRG. Thus, G' is a PRG, as desired.

Problem 3: Let $G : \{0,1\}^n \to \{0,1\}^{n+1}$ be an arbitrary PRG. Show that $G' : \{0,1\}^{n-1} \to \{0,1\}^{n+1}$ defined by $G'(x) = G\left(x|| \bigoplus_{i \in [n-1]} x_i\right)$ is not necessarily a PRG. (You may assume that there exist PRGs.)

Solution 3: Generally, these sorts of questions have three steps:

1. Assume there exists some PRG, H.
2. Using H, construct a new PRG G, and prove that G is a PRG (assuming that H is).
3. Plugging in this G from the previous step into G', show that G' is *not* a PRG by giving an explicit distinguisher.

We will now proceed with these three steps. Let $H : \{0,1\}^{n-1} \to \{0,1\}^n$ be an arbitrary PRG. Define $G : \{0,1\}^n \to \{0,1\}^{n+1}$ as

$$G(x||b) = H(x)||b \oplus \bigoplus_{i \in [n-1]} x_i,$$

where $|x| = n-1$ and $|b| = 1$. We now show that G is a PRG. To see this, suppose not, i.e., we have a PPT distinguisher D, for which (possibly by negating the distinguisher) there exists a polynomial function p and infinitely many n such that

$$\Pr_{y \leftarrow \{0,1\}^n}[D(G(y)) = 1] - \Pr_{z \leftarrow \{0,1\}^{n+1}}[D(z) = 1] \geq 1/p(n).$$

By definition of G, this means that

$$\Pr_{x \leftarrow \{0,1\}^{n-1}, b \leftarrow \{0,1\}}\left[D\left(H(x)||b \oplus \bigoplus_{i \in [n-1]} x_i\right) = 1\right] - \Pr_{z \leftarrow \{0,1\}^{n+1}}[D(z) = 1] \geq 1/p(n),$$

for infinitely many n. By a one-time pad argument, the distribution of $b' := b \oplus \bigoplus_{i \in [n-1]} x_i$ is uniformly random and independent of x, so one can re-write this as

$$\Pr_{x \leftarrow \{0,1\}^{n-1}, b' \leftarrow \{0,1\}}[D(H(x)||b') = 1] - \Pr_{z \leftarrow \{0,1\}^{n}, b' \leftarrow \{0,1\}}[D(z||b') = 1] \geq 1/p(n),$$

for infinitely many n. Now, consider the distinguisher defined by $D'(y) = D(y||b')$ for uniformly random $b' \leftarrow \{0,1\}$. We have

$$\Pr_{x \leftarrow \{0,1\}^{n-1}}[D'(H(x)) = 1] - \Pr_{z \leftarrow \{0,1\}^{n}}[D'(z) = 1] \geq 1/p(n),$$

for infinitely many n. Therefore, D' is a distinguisher for H, showing that H is not a PRG. This is a contradiction. Therefore, G is a PRG.

Now, we have $G'(x) = G\left(x|| \bigoplus_{i \in [n-1]} x_i\right) = H(x)|| \left(\bigoplus_{i \in [n-1]} x_i\right) \oplus \left(\bigoplus_{i \in [n-1]} x_i\right) = H(x)||0$. There is a simple distinguisher for G': check if the last bit is 0! This will be true with probability 1 for G', but only probability $1/2$ for a random string, so G' is not a PRG.