CS276 Scribe Notes
Indistinguishability (cont.) and Pseudorandom Generators

Dominic Antonelli

Lecture 3 - January 24, 2006

1 Indistinguishability

The following two theorems come from the previous lecture:

Theorem 1 (Data Processing Inequality) *If X, and Y are (t, ϵ)-indistinguishable and f is a function that can be computed in time t', then $f(X)$ and $f(Y)$ are $(t - t', \epsilon)$-indistinguishable.*

Theorem 2 *If X and Y are (t, ϵ)-indistinguishable and Y and Z are (t, ϵ')-indistinguishable, then X and Z are $(t, \epsilon + \epsilon')$-indistinguishable.*

Now, we introduce a general notion of indistinguishability. We say X is indistinguishable from Y (written $X \sim Y$) if X and Y are (t, ϵ)-indistinguishable for some t and ϵ that are "not too large". The analogs to Theorems 1 and 2 hold here, namely:

Theorem 3 *If $X \sim Y$, then $F(X) \sim F(Y)$.*

Theorem 4 *If $X \sim Y$ and $Y \sim Z$, then $X \sim Z$.*

If we repeatedly apply Theorem 4, we get

Theorem 5 *If $X_0 \sim X_1$, $X_1 \sim X_2$, $X_2 \sim X_3$, $\ldots$ $X_{n-1} \sim X_n$, then $X_0 \sim X_n$.*

Next, we look at a more formal, asymptotic, notion of indistinguishability.

Definition 1 (Polynomially Indistinguishable) *Given two families of distributions $X = \{X^n : n = 1, 2, \ldots\}$ and $Y = \{Y^n : n = 1, 2, \ldots\}$, we say X and Y are **polynomially indistinguishable** iff for all polynomials $t(n)$, there exists a negligible function $\epsilon(n)$ such that for all n, X^n and Y^n are $(t(n), \epsilon(n)) - indistinguishable$.*

Definition 2 (Negligible) *A function $\epsilon(n)$ is **negligible** if, for all polynomials p, there exists a constant N, such that $\epsilon(n) \leq \frac{1}{p(n)}$ for all $n > N$.*

What are are saying here is that if the worst-case running time limit, t(n), grows faster than all polynomials while the maximum advantage any adversary running in time t(n) can get decreases faster than all inverse-polynomials, then we have polynomial indistinguishability.

2 Pseudorandom Generators

Now we can use these tools to discuss a useful application: Pseudorandom Generators (PRGs). The basic idea is to take in a k-bit random bit-string and output a longer bit-string that "looks" random. How random it "looks" can be based on many statistical tests. For example, does the output have approximately as many 0's as 1's? How long is the longest run of consecutive 0's? Ideally, the output of the pseudorandom generator would be indistinguishable from the uniform distribution on bit-strings of the same size. More concretely, the properties we'd like for any PRG, G, are

1. G should be efficiently computable
2. The output of G should be larger than the input
3. Given a uniform random input k, we want $G(k) \sim$ uniform

If I flip a coin to decide whether to give you an output from G or a uniformly randomly chosen bit-string of the same length, you should not be able to tell the difference. The reason this is useful is that it may be much cheaper to generate a short truly random string and run it through the PRG rather than generating a long truly random string.

Analogous to indistinguishability, we have the notion of a (t, ϵ)-secure PRG. This is simply a PRG whose output is (t, ϵ)-indistinguishable from uniform. We also have the notion of polynomially-secure families of PRGs. Define a set of PRGs $G = \{G^1, G^2, \ldots\}$. Now define the family of distributions obtained by giving each PRG in G a bit-string chosen uniformly at random from the bit-strings of the correct length for that generator. G is polynomially-secure if this family of distributions is polynomially-indistinguishable from the set of uniform distributions of corresponding sizes.

Now, let us look at some example generators and analyze whether they are pseudorandom or not. Define $G_M : \{0,1\}^{100} \rightarrow \{0,1\}^{1000}$ to use the following algorithm:

- M is a pre-initialized 1000 by 100 matrix of boolean values
- $G_M(k) = M \cdot k$

Is this a family of pseudorandom generators? One easy counterexample is if M is a matrix of all 0's. Then clearly, the output of the generator is always 0, and this is not pseudorandom. In fact, for any M, this is not pseudorandom. If an adversary has access to M, it can use Gaussian Elimination to see if a particular output is even possible from this generator. Since any random 2^{1000} bit string has a $\frac{2^{100}}{2^{1000}}$ probability of even being a possible output of the generator, the adversary can be fairly sure that if the Gaussian Elimination fails, the sample is from the uniform distribution and if it succeeds, the sample is from the generator. The moral of the story is that all the security should reside in the key, not the algorithm. In most cases, we must assume that the adversary has access to the algorithm.

2.1 Statistical Properties

Next we look at analyzing specific statistical properties of pseudorandom generators.

Theorem 6 *If G is a (t, ϵ)-PRG that maps from $\{0,1\}^a \rightarrow \{0,1\}^n$, then*

$$\Pr\left[G(k) \textit{ has} \leq \frac{n}{3} \textit{ ones}; k \xleftarrow{\$} \{0,1\}^a\right] \leq \exp\left\{\frac{-n}{18}\right\} + \epsilon$$

First we need the following fact, which comes from a Chernoff Bound,

$$\Pr\left[u \textit{ has} \leq \frac{n}{3} \textit{ ones}; u \xleftarrow{\$} \{0,1\}^n\right] \leq \exp\left\{\frac{-n}{18}\right\}$$

Then, we define an adversary A that takes an n-bit string and counts the number of 1's in it. If the number of ones is less than $\frac{n}{3}$, then output 1, otherwise output 0. Now we look at the advantage of this adversary at distinguishing the output of G from uniform:

$$Adv(A) = \left|\Pr\left[A(G(k)) = 1; k \xleftarrow{\$} \{0,1\}^a\right] - \Pr\left[A(u) = 1; u \xleftarrow{\$} \{0,1\}^n\right]\right|$$

Then, from how we defined A, we get

$$Adv(A) = \left|\Pr\left[G(k) \textit{ has} \leq \frac{n}{3} \textit{ ones}; k \xleftarrow{\$} \{0,1\}^a\right] - \Pr\left[u \textit{ has} \leq \frac{n}{3} \textit{ ones}; u \xleftarrow{\$} \{0,1\}^n\right]\right|$$

Then, because G is a (t, ϵ)-PRG, we know this advantage must be no more than ϵ, so

$$\left|\Pr\left[G(k) \textit{ has} \leq \frac{n}{3} \textit{ ones}; k \xleftarrow{\$} \{0,1\}^a\right] - \Pr\left[u \textit{ has} \leq \frac{n}{3} \textit{ ones}; u \xleftarrow{\$} \{0,1\}^n\right]\right| \leq \epsilon$$

which tells us

$$\Pr\left[G(k)\ has \leq \frac{n}{3}\ ones; k \xleftarrow{\$} \{0,1\}^a\right] \leq \Pr\left[u\ has \leq \frac{n}{3}\ ones; u \xleftarrow{\$} \{0,1\}^n\right] + \epsilon$$

and finally, using the Chernoff Bound fact, we get

$$\Pr\left[G(k)\ has \leq \frac{n}{3}\ ones; k \xleftarrow{\$} \{0,1\}^a\right] \leq \exp\left\{\frac{-n}{18}\right\} + \epsilon$$

In general, if we have a bound for how likely a particular statistical property is for a uniform distribution, then we get an equivalent bound for (t,ϵ)-PRGs that is off by a factor of at most ϵ.

2.2 Shrinking a PRG

We might be interested in making a PRG that produce fewer additional bits than one we have. This is fairly simple, we can simply discard any bits we do not need. As an example, assume we have a PRG, $G : \{0,1\}^{128} \to \{0,1\}^{10^6}$, and we want to create a PRG, $H : \{0,1\}^{128} \to \{0,1\}^{500}$. Then $H(k)$ is just $first_{500}(G(k))$.

Theorem 7 *If G is a (t,ϵ)-PRG then H is a $(t-500,\epsilon)$-PRG.*

Using the following definitions:

- $k \xleftarrow{\$} \{0,1\}^{128}$
- $v \xleftarrow{\$} \{0,1\}^{500}$
- $u \xleftarrow{\$} \{0,1\}^{10^6}$

The proof contains three main steps. The first is that, by the definition of H, $H(k) = first_{500}(G(k))$. The second step is to use the data processing inequality to show $first_{500}(G(k)) \sim first_{500}(u)$. The final step is to note that $first_{500}(u) = v$. Once we've done this, we've shown that $H(k)$ is indistinguishable from uniform, which means it is a PRG.

The formal proof is as follows. Assume we can break H, namely there exists an algorithm B that runs in time $t-500$ that can distinguish $H(k)$ from uniform with advantage ϵ. Define another algorithm A as $A(y) = B(first_{500}(y))$. The $first_{500}$ function takes no more than 500 time (may take less depending on the computing model), so because B takes $\leq t-500$ time, A takes $\leq t$ time. Now, we need to look at the advantage of A at distinguishing the output of G from uniform, namely,

$$Adv(A) = \left|\Pr\left[A(G(k)) = 1; k \xleftarrow{\$} \{0,1\}^{128}\right] - \Pr\left[A(u) = 1; u \xleftarrow{\$} \{0,1\}^{10^6}\right]\right|$$

Now, from how we defined A, we get

$$Adv(A) = \left|\Pr\left[B(first_{500}(G(k))) = 1; k \xleftarrow{\$} \{0,1\}^{128}\right] - \Pr\left[B(first_{500}(u)) = 1; u \xleftarrow{\$} \{0,1\}^{10^6}\right]\right|$$

and from the definition of H,

$$Adv(A) = \left|\Pr\left[B(H(k)) = 1; k \xleftarrow{\$} \{0,1\}^{128}\right] - \Pr\left[B(first_{500}(u)) = 1; u \xleftarrow{\$} \{0,1\}^{10^6}\right]\right|$$

and v,

$$Adv(A) = \left|\Pr\left[B(H(k)) = 1; k \xleftarrow{\$} \{0,1\}^{128}\right] - \Pr\left[B(v) = 1; v \xleftarrow{\$} \{0,1\}^{500}\right]\right|$$

but this is just the advantage of B at breaking H, which (by assumption) is at least ϵ.

$$Adv(A) \geq \epsilon$$

So, now we have an algorithm that runs in $\leq t$ time and has an advantage of $\geq \epsilon$ at breaking G, contradicting the fact that G is a (t,ϵ)-PRG, so H must be a $(t-500,\epsilon)$-PRG.

2.3 Chaining PRGs

Theorem 8 *If $G : \{0,1\}^{100} \rightarrow \{0,1\}^{1000}$ is a $(t,\epsilon) - PRG$, and $G' : \{0,1\}^{1000} \rightarrow \{0,1\}^{10^6}$ is a $(t,\epsilon) - PRG$ that runs in time t', then $H(k) = G'(G(k))$ is a $(t-t', 2\epsilon)$-PRG.*

The proof (not given) is fairly straightforward. The main step is to use the Data Processing Inequality using G' as our function.

2.4 Parallel PRGs

If we have a PRG, $G : \{0,1\}^{100} \rightarrow \{0,1\}^{200}$, then we can make a bigger PRG, $H : \{0,1\}^{200} \rightarrow \{0,1\}^{400}$. H is defined as follows:

1. split the 200-bit input into two separate 100-bit values
2. input each 100-bit value into a separate instance of G
3. take the 2 200-bit outputs and concatenate them together

If G is a (t,ϵ)-PRG, and G runs in time t', then H is a $(t-t', 2\epsilon)$-PRG. The proof (not given) is fairly straightforward, and involves using the Data Processing Inequality twice, using $f(n) = (n, u) : u \xleftarrow{\$} \{0,1\}^{200}$ and $g(n) = (G(k), n)$.