Notes for Lecture 10

In this lecture we discuss pseudorandomness and derandomization. These notes cover some optional material that was not discussed in class.

1 Probabilistic Algorithms versus Deterministic Algorithms

A probabilistic algorithm $A(\cdot,\cdot)$ is an algorithm that takes two inputs x and r, where x is an instance of some problem that we want to solve, and r is the output of a *random source*. A random source is an idealized device that outputs a sequence of bits that are uniformly and independently distributed. For example the random source could be a device that tosses coins, observes the outcome, and outputs it. A probabilistic algorithm A is good if it is efficient and if, say, for every x,

$$\mathop{\mathbb{P}}_r[A(x,r) = \text{ right answer for } x\ ] \geq \frac{3}{4}$$

We will typically restrict to the case where A solves a decision problem (e.g. it tests whether two read-once branching programs are equivalent). In this case we say that a language L is in **BPP** if there is a polynomial time algorithm $A(\cdot,\cdot)$ (polynomial in the length of the first input) such that for every x

$$\mathop{\mathbb{P}}_r[A(x,r) = \chi_L(x)] \geq \frac{3}{4}$$

or, said another way,

$$x \in L \Rightarrow \mathop{\mathbb{P}}_r[A(x,r) = 1] \geq \frac{3}{4}$$

and

$$x \notin L \Rightarrow \mathop{\mathbb{P}}_r[A(x,r) = 1] \leq \frac{1}{4} \ .$$

The choice of the constant $3/4$ is clearly quite arbitrary. For any constant $1/2 < p < 1$, if we had defined **BPP** by requiring the probabilistic algorithm to be correct with probability st least p, we would have given an equivalent definition. In fact, for any polynomial p, it would have been equivalent to define **BPP** by asking the algorithm to be correct with probability at least $1/2+1/p(n)$, where n is the size of the input, and it would have also been equivalent if we had asked the algorithm to be correct with probability at least $1-1/2^{p(n)}$. That is, for any two polynomials p and q, if for a decision problem L we have a probabilistic polynomial time A that solves L on every input of length n with probability at least $1/2+1/p(n)$, then there is another probabilistic algorithm A', still running in polynomial time, that solves L on every input of length n with probability at least $1-2^{-q(n)}$.

For quite a few interesting problems, the only known polynomial time algorithms are probabilistic. A well-known example is the problem of testing whether two multivariate low-degree polynomials given in an implicit representation are equivalent. Another example is the problem of extracting "square roots" modulo a prime, i.e. to find solutions, if they exist, to equations of the form $x^2 = a \pmod{p}$ where p and a are given, and p is prime. More generally, there are probabilistic polynomial time algorithms to find roots of polynomials modulo a prime. There is no known deterministic polynomial time algorithm for any of the above problems.

It is not clear whether the existence of such probabilistic algorithms suggests that probabilistic algorithms are inherently more powerful than deterministic ones, or that we have not been able yet to find the best possible deterministic algorithms for these problems. In general, it is quite an interesting question to determine what is the relative power of probabilistic and deterministic computations. This question is the main motivations for the results described in this lecture and the next ones.

1.1 A trivial deterministic simulation

Let A be a probabilistic algorithm that solves a decision problem L. On input x of length n, say that A uses a random string r of length $m = m(n)$ and runs in time $T = T(n)$ (note that $m \leq T$).

It is easy to come up with a deterministic algorithm that solves L in time $2^{m(n)}T(n)$. On input x, compute $A(x, r)$ for every r. The correct answer is the one that comes up the majority of the times, so, in order to solve our problem, we just have to keep track, during the computation of $A(x, r)$ for every r, of the number of strings r for which $A(x, r) = 1$ and the number of strings r for which $A(x, r) = 0$.

Notice that the running time of the simulation depends exponentially on the number of random bits used by A, but only polynomially on the running time of A. In particular, if A uses a logarithmic number of random bits, then the simulation is polynomial. However, typically, a probabilistic algorithm uses a linear, or more, number of random bits, and so this trivial simulation is exponential. As we will see in the next section, it is not easy to obtain more efficient simulations.

1.2 Exponential gaps between randomized and deterministic procedures

For some computational problems (e.g. approximating the size of a convex body) there are probabilistic algorithms that work even if the object on which they operate is exponentially big and given as a black box; in some cases one can prove that deterministic algorithms cannot solve the same problem in the same setting, unless they use exponential time. Let us see a particularly clean (but more artificial) example of this situation.

Suppose that there is some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ that is given as an oracle; we want to devise an algorithm that on input x finds an approximation (say, to within an additive factor $1/10$) to the value $\mathbb{P}_y[f(x,y) = 1]$. A probabilistic algorithm would pick $O(1)$ points $y_1, \ldots, y_t$ at random, evaluate $f(x, y_i)$, and then output the fraction of i such that $f(x, y_i) = 1$. This will be an approximation to within $1/10$ with good probability. However a deterministic subexponential algorithm, given x, can only look at a negligible fraction of the values $f(x, y)$. Suppose that f is zero everywhere. Now consider the function $g(x, y)$ that is equal to f on all the points that our algorithm queries, and is 1 elsewhere (note that, by this definition, the queries of the algorithm on input x will be the same for f and g). If the algorithm takes sub-exponential time, g is almost everywhere one, yet the algorithm will give the same answer as when accessing f, which is everywhere zero. If our algorithm makes less than 2^{n-1} oracle queries, it cannot solve the problem with the required accuracy.

2 De-randomization Under Complexity Assumptions

It is still not known how to improve, in the general case, the deterministic simulation of Section 1.1, and the observation of Section 1.2 shows one of the difficulties in achieving an improvement. If we want to come up with a general way of transforming probabilistic procedures into deterministic sub-exponential procedures, the transformation cannot be described and analyzed by modeling in a "black box" way the probabilistic procedure.[1] If we want to deterministically and sub-exponentially simulate **BPP** algorithms, we have to exploit the fact that a **BPP** algorithm $A(\cdot,\cdot)$ is not an arbitrary function, but an efficiently computable one, and this is difficult because we still have a very poor understanding of the nature of efficient computations.

The results described in these notes show that it is indeed possible to deterministically simulate probabilistic algorithms in sub-exponential (or even polynomial) time, provided that certain complexity-theoretic assumptions are true. It is quite usual in complexity theory that, using reductions, one can show that the answer to some open question is implied by (or even equivalent to) the answer to some other question, however the nature of the results of these notes is somewhat unusual. Typically a reduction from a computational problem A to a problem B shows that if B has an efficient algorithm then A has also an efficient algorithm, and, by counterpositive, if A is intractable then B is also intractable. In general, using reductions one shows that algorithmic assumptions imply algorithmic consequences, and intractability assumptions imply intractability consequences. In these notes we will see instead that the existence of efficient derandomized algorithms is implied by the *intractability* of some other problem, so that a hardness condition implies an algorithm consequence.

In the next section we will introduce some notation about computational problems and complexity measures, and then we will state some results about conditional de-randomization.

2.1 Formal Definitions of Complexity Measures and Complexity Classes

For a decision problem L and an integer n we denote by L_n the restriction of L to inputs of length n. It will be convenient to think of L_n as a Boolean function $L_n : \{0,1\}^n \to \{0,1\}$ (with the convention that $x \in L_n$ if and only if $L_n(x) = 1$).

For a function $f : \{0,1\}^n \to \{0,1\}$, consider the size of the smallest circuit that solves f; denote this number by $CC(f)$. By definition, we have that if C is a circuit with n inputs of size less than $CC(f)$ then there exists an $x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.

[1]More precisely, it is impossible to have a sub-exponential time deterministic "universal derandomization procedure" that given x and oracle access to an arbitrary function $A(\cdot,\cdot)$ outputs 1 when $\mathbb{P}_r[A(x,r) = 1] \geq 3/4$ and outputs 0 when $\mathbb{P}_r[A(x,r) = 1] \leq 1/4$. In fact, more generally, it is impossible to give sub-exponential time algorithms for all **BPP** problems by using "relativizing" techniques. It is beyond the scope of these notes to explain what this means, and why it is more general. "Relativizations" are discussed in [**?**], where it is possible to find pointers to the relevant literature.

Consider now, for every n, what is the largest s such that for every circuit C of size $\leq s$ we have $\mathbb{P}_{x \in \{0,1\}^n}[C(x) = f(x)] \leq 1/2 + 1/s$; denote this number by $H(f)$.

Recall that $DTIME(T(n))$ is the class of decision problems that can be solved by deterministic algorithms running in time at most $T(n)$ on inputs of length n. We have the classes $\mathbf{E} = DTIME(2^{O(n)})$ and $\mathbf{EXP} = DTIME(2^{n^{O(1)}})$.

2.2 Hardness versus Randomness

From our previous arguments, we have $\mathbf{BPP} \subseteq \mathbf{EXP}$. Since there are settings where probabilistic procedures require exponential time to be simulated, one would conjecture that $\mathbf{BPP} \not\subseteq 2^{n^{o(1)}}$; on the other hand, **BPP** seems to still represent a class of feasible computations, and it would be *very* surprising if $\mathbf{BPP} = \mathbf{EXP}$. As we will see in a moment, something is wrong with the above intuition. Either $\mathbf{BPP} = \mathbf{EXP}$, which sounds really implausible, or else it must be the case that **BPP** has sub-exponential time deterministic algorithms (that will work well only on average, but that would be quite remarkable enough).

Theorem 1 (Impagliazzo, Wigderson) *Suppose* $\mathbf{BPP} \neq \mathbf{EXP}$*; then for every* **BPP** *language* L *and every* $\epsilon > 0$ *there is a deterministic algorithm* A *that works in time* 2^{n^ϵ} *and, for infinitely many* n*, solves* L *on a fraction* $1 - 1/n$ *of the inputs of length* n*.*

This gives a non-trivial simulation of **BPP** under an uncontroversial assumption. We can also get an optimal simulation of **BPP** under an assumption that is much stronger, but quite believable.

Theorem 2 (Impagliazzo, Wigderson) *Suppose there is a problem* L *in* **E** *and a fixed* $\delta > 0$ *such that, for all sufficiently large* n*,* $CC(L_n) \geq 2^{\delta n}$*; then* $\mathbf{P} = \mathbf{BPP}$*.*

We will call the statement "there is a problem L in **E** and a fixed $\delta > 0$ such that, for all sufficiently large n, $CC(L_n) \geq 2^{\delta n}$" the "IW assumption." Note that if the IW assumption is true, then it is true in the case where

$$L = \{(M, x, 1^k) : \text{ machine } M \text{ halts within } 2^k \text{ steps on input } x\ \}$$

Notice also that L cannot be solved by algorithms running in time $2^{o(n)}$, and so it would be a little bit surprising if it could be solvable by circuits of size $2^{o(n)}$, because it would mean that, for general exponential time computations, non-uniformity buys more than a polynomial speed-up. In fact it would be very surprising if circuits of size $2^{.99n}$ existed for L.

The two theorems that we just stated are the extremes of a continuum of results showing that by making assumptions on the hardness of problems in **E** and exp it is possible to devise efficient deterministic algorithms for all **BPP** problems. The stronger the assumption, the more efficient the simulation.

Notice that the assumption in Theorem 2 is stronger than the assumption in Theorem 1 in two ways, and that, similarly, the conclusion of Theorem 2 is stronger than the conclusion in Theorem 1 in two ways. On the one hand, the assumption in Theorem 2 refers to circuit size, that is, to a non-uniform measure of complexity, whereas the assumption in Theorem 1 uses a uniform measure of complexity (running time of probabilistic algorithms). This difference accounts for the fact that the conclusion of Theorem 2 gives an algorithm that works for all inputs, while the conclusion of Theorem 1 gives an algorithm that works only for most inputs. The other difference is that Theorem 2 assumes exponential hardness, while Theorem 2 assumes only super-polynomial hardness. This is reflected in the running time of the consequent deterministic simulations (respectively, polynomial and sub-exponential).

3 Pseudorandom Generators

We say that a function $G : \{0,1\}^t \to \{0,1\}^m$ is a (s,ϵ)-pseudorandom generator if for every circuit D of size $\leq s$ we have

$$|\mathop{\mathbb{P}}_{r}[D(r) = 1] - \mathop{\mathbb{P}}_{z}[D(G(z)) = 1]| \leq \epsilon$$

Suppose that we have a probabilistic algorithm A such that for inputs x of length n the computation $A(x, \cdot)$ can be performed by a circuit of size $s(n)$; suppose that for every x we have $\mathbb{P}_r[A(x,r) = \text{ right answer }] \geq 3/4$, and suppose that we have a $(s, 1/8)$ pseudorandom generator $G : \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}$. Then we can define a new probabilistic algorithm A' such that $A'(x,z) = A(x, G(z))$. It is easy to observe that for every x we have

$$\mathop{\mathbb{P}}_{z}[A'(x,z) = \text{ right answer }] \geq 5/8$$

and that, using the trivial derandomization we can get a deterministic algorithm A'' that always works correctly and whose running time is 2^t times the sum of the running time of A plus the running time of G.

If t is logarithmic in m and s, and if G is computable in $\text{poly}(m, s)$ time, then the whole simulation runs in deterministic polynomial time. Notice also that if we have a (s,ϵ)-pseudorandom generator $G : \{0,1\}^t \to \{0,1\}^m$, then for every $m' \leq m$ we also have, for a stronger reason, a (s,ϵ) pseudorandom generator $G' : \{0,1\}^t \to \{0,1\}^{m'}$ (G' just computes G and omits the last $m - m'$ bits of the output). So there will be no loss in generality if we consider only generators for the special case where, say, $s = 2m$. (This is not really necessary, but it will help reduce the number of parameters in the statements of theorems.) We have the following easy theorem.

Theorem 3 *Suppose there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to \{0,1\}^m$ that are computable in $\text{poly}(m)$ time and that are $(2m, 1/8)$-pseudorandom; then $\mathbf{P} = \mathbf{BPP}$.*

Of course this is only a sufficient condition. There could be other approaches to proving (conditionally) $\mathbf{P} = \mathbf{BPP}$, without passing through the construction of such strong generators. Unfortunately we hardly know of any other approach, and anyway the (arguably) most interesting results are proved using pseudorandom generators.

4 The two main theorems

4.1 The Nisan-Wigderson Theorem

Theorem 4 (Nisan, Wigderson) *Suppose there is $L \in \mathbf{E}$ and $\delta > 0$ such that, for all sufficiently large n, $H(L_n) \geq 2^{\delta n}$; then there is a family of generators $G_m : \{0,1\}^{O(\log m)} \to$*

$\{0,1\}^m$ *that are computable in* $\mathrm{poly}(m)$ *time and that are* $(2m, 1/8)$*-pseudorandom (in particular,* $\mathbf{P} = \mathbf{BPP}$*).*

Notice the strength of the assumption. For almost every input length n, our problem has to be so hard that even circuits of size $2^{\delta n}$ have to be unable to solve the problem correctly on more than a fraction $1/2 + 2^{-\delta n}$ of the inputs. A circuit of size 1 can certainly solve the problem on a fraction at least $1/2$ of the inputs (either by always outputting 0 or by always outputting 1). Furthermore, a circuit of size 2^n always exist that solves the problem on every input. A circuit of size $2^{\delta n}$ can contain, for example, the right solution to our problem for every input whose first $(1-\delta)n$ bits are 0; the circuit can give the right answer on these $2^{\delta n}$ inputs, and answer always 0 or always 1 (whichever is better) on the other inputs. This way the circuit is good on about a fraction $1/2 + 2^{-(1-\delta)n}$ of the inputs. So, in particular, for every problem, there is a circuit of size $2^{n/2}$ that solves the problem on a fraction $1/2 + 2^{-n/2}$ of the inputs. It is somewhat more tricky to show that there is in fact even a circuit of size $2^{(1/3+o(1))n}$ that solves the problem on a fraction $1/2 + 2^{-(1/3+o(1))n}$ of the inputs, and this is about best possible for general problems.

4.2 Worst-case to Average-case Reduction

Theorem 5 (Impagliazzo, Wigderson) *Suppose there is* $L \in \mathbf{E}$ *and* $\delta > 0$ *such that, for all sufficiently large* n, $CC(L_n) \geq 2^{\delta n}$*; Then there is* $L' \in \mathbf{E}$ *and* $\delta' > 0$ *such that, for all sufficiently large* n, $H(L'_n) \geq 2^{\delta' n}$.

This is quite encouraging: the (believable) IW assumption implies the (a priori less believable) NW assumption.

5 The Nisan-Wigderson Construction

The Nisan-Wigderson generator is based on the existence of a decision problem L in $\mathbf{E}$ such that for almost every input length l we have $H(L_l) \geq 2^{\delta l}$, yet there is a uniform algorithm that solves L_l in $2^{O(l)}$ time. Our goal is to use these assumptions on L_l to build a generator whose input seed is of length $O(l)$, whose output is of length $2^{\Theta(l)}$ and indistinguishable from uniform by adversaries of size $2^{\Theta(l)}$, and the generator should be computable in time $2^{O(l)}$.

As we will see in a moment, it is not too hard to construct a generator that maps l bits into $l+1$ bits, and whose running time and pseudorandomness are as required. We will then present the Nisan-Wigderson construction, and present its analysis.

5.1 Impredictability versus Pseudorandomness

Let $f : \{0,1\}^l \to \{0,1\}$ be a function such that $H(f) \geq s$, and consider the pseudorandom generator $G : \{0,1\}^l \to \{0,1\}^{l+1}$ defined as $G(x) = x \cdot f(x)$, where '$\cdot$' is used to denote concatenation. We want to argue that G is a $(s-3, 1/s)$-pseudorandom generator.

The argument works by contradiction, and consists in the proof of the following result.

Lemma 6 *Let $f : \{0,1\}^l \to \{0,1\}$. Suppose that there is a circuit D of size s such that*

$$|\mathop{\mathbb{P}}_x[D(x \cdot f(x)) = 1] - \mathop{\mathbb{P}}_{x,b}[D(x \cdot b) = 1]| > \epsilon$$

then there is a circuit A of size $s+3$ such that

$$\mathop{\mathbb{P}}_x[A(x) = f(x)] > \frac{1}{2} + \epsilon$$

Proof: First of all, we observe that there is a circuit D' of size at most $s+1$ such that

$$\mathop{\mathbb{P}}_z[D'(x \cdot f(x)) = 1] - \mathop{\mathbb{P}}_{x,b}[D'(x \cdot b) = 1] > \epsilon \tag{1}$$

This is because Expression (1) is satisfied either by taking $D = D'$ or by taking $D = \neg D'$. A way to interpret Expression (1) is to observe that when the first l bits of the input of $D'()$ are a random string x, D' is more likely to accept if the last bit is $f(x)$ than if the last bit is random (and, for a stronger reason, if the last bit is $1 - f(x)$). This observation suggests the following strategy in order to use D' to predict f: given an input x, for which we want to compute $f(x)$, we guess a value b, and we compute $D'(x,b)$. If $D'(x,b) = 1$, we take it as evidence that b was a good guess for $f(x)$, and we output b. If $D'(x,b) = 0$, we take it as evidence that b was the wrong guess, and we output $1-b$. Let A_b be the procedure that we just described. We claim that

$$\mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x)] > \frac{1}{2} + \epsilon \tag{2}$$

The claim is proved by the following derivation

$$\begin{aligned}
& \mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x)] \\
= \; & \mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x) | b = f(x)] \mathop{\mathbb{P}}_{x,b}[b = f(x)] \\
& + \mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x) | b \neq f(x)] \mathop{\mathbb{P}}_{x,b}[b \neq f(x)] \\
= \; & \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x) | b = f(x)] + \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[A_b(x) = f(x) | b \neq f(x)] \\
= \; & \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b = f(x)] + \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 0 | b \neq f(x)] \\
= \; & \frac{1}{2} + \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b = f(x)] - \frac{1}{2} \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b \neq f(x)] \\
= \; & \frac{1}{2} + \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b = f(x)] \\
& - \frac{1}{2} \left(\mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b = f(x)] + \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1 | b \neq f(x)] \right) \\
= \; & \frac{1}{2} + \mathop{\mathbb{P}}_x[D'(x, f(x)) = 1] - \mathop{\mathbb{P}}_{x,b}[D'(x,b) = 1] \\
> \; & \frac{1}{2} + \epsilon
\end{aligned}$$

From Expression (2) we can observe that there must be a $b_0 \in \{0,1\}$ such that

$$\mathop{\mathbb{P}}_x[A_{b_0}(x) = f(x)] > \frac{1}{2} + \epsilon$$

And A_{b_0} is computed by a circuit of size at most $s+3$ because $A_{b_0}(x) = b_0 \oplus (\neg D'(x, b_0))$, which can be implemented with two more gates given a circuit for D'. □

5.2 Combinatorial Designs

Consider a family $(S_1, \ldots, S_m)$ of subsets of an universe U. We say the family is a (l, α)-design if, for every i, $|S_i| = l$, and, for every $i \neq j$, $|S_i \cap S_j| \leq \alpha$.

Theorem 7 *For every integer l, fraction $\gamma > 0$, there is an $(l, \log m)$ design $(S_1, \ldots, S_m)$ over the universe $[t]$, where $t = O(l/\gamma)$ and $m = 2^{\gamma l}$; such a design can be constructed in $O(2^t t m^2)$ time.*

We will use the following notation: if z is a string in $\{0,1\}^t$ and $S \subset [t]$, then we denote by $z_{|S}$ the string of length $|S|$ obtained from z by selecting the bits indexed by S. For example if $z = (0,0,1,0,1,0)$ and $S = \{1,2,3,5\}$ then $z_{|S} = (0,0,1,1)$.

5.3 The Nisan-Wigderson Generator

For a Boolean function $f : \{0,1\}^l \to \{0,1\}$, and a design $\mathcal{S} = (S_1, \ldots, S_m)$ over $[t]$, the Nisan-Wigderson generator is a function $NW_{f,\mathcal{S}} : \{0,1\}^t \to \{0,1\}^m$ defined as follows:

$$NW_{f,\mathcal{S}}(z) = f(z_{|S_1}) \cdot f(z_{|S_2}) \cdots f(z_{|S_m})$$

6 The Reduction from Distinguishing to Predicting

The following lemma implies Theorem 4.

Lemma 8 *Let $f : \{0,1\}^l \to \{0,1\}$ be a Boolean function and $\mathcal{S} = (S_1, \ldots, S_m)$ be a $(l, \log m)$ design over $[t]$. Suppose $D : \{0,1\}^m \to \{0,1\}$ is such that*

$$|\mathop{\mathbb{P}}_r[D(r) = 1] - \mathop{\mathbb{P}}_z[D(NW_{f,\mathcal{S}}(z)) = 1]| > \epsilon \ .$$

Then there exists a circuit C of size $O(m^2)$ such that

$$|\mathop{\mathbb{P}}_x[D(C(x)) = f(x)] - 1/2| \geq \frac{\epsilon}{m}$$

Proof: The main idea is that if D distinguishes $NW_{f,\mathcal{S}}(\cdot)$ from the uniform distribution, then we can find a bit of the output of the generator where this distinction is noticeable. On such a bit, D is distinguishing $f(x)$ from a random bit, and such a distinguisher can be turned into a predictor for f. In order to find the "right bit", we will use the *hybrid argument.*

Let us start with the hybrid argument. We define $m+1$ distributions $H_0, \ldots, H_m$; H_i is defined as follows: sample a string $v = NW_{f,\mathcal{S}}(z)$ for a random z, and then sample a string $r \in \{0,1\}^m$ according to the uniform distribution, then concatenate the first i bits of v with the last $m-i$ bits of r. By definition, H_m is distributed as $NW_{f,\mathcal{S}}(y)$ and H_0 is the uniform distribution over $\{0,1\}^m$.

Using the hypothesis of the Lemma, we know that there is a bit $b_0 \in \{0,1\}$ such that

$$\mathop{\mathbb{P}}_y[D'(NW_{f,\mathcal{S}}(y)) = 1] - \mathop{\mathbb{P}}_r[D'(r)] > \epsilon$$

where $D'(x) = b_0 \oplus D(x)$.

We then observe that

$$\begin{aligned} \epsilon &\leq \mathop{\mathbb{P}}_z[D'(NW_{f,\mathcal{S}}(z)) = 1] - \mathop{\mathbb{P}}_r[D'(r)] \\ &= \mathbb{P}[D'(H_m) = 1] - \mathbb{P}[D'(H_0) = 1] \\ &= \sum_{i=1}^{m} (\mathbb{P}[D'(H_i) = 1] - \mathbb{P}[D'(H_{i-1}) = 1]) \end{aligned}$$

In particular, there exists an index i such that

$$\mathbb{P}[D'(H_i) = 1] - \mathbb{P}[D'(H_{i-1}) = 1] \geq \epsilon/m \tag{3}$$

Now, recall that

$$H_{i-1} = f(z_{|S_1}) \cdots f(z_{|S_{i-1}}) r_i r_{i+1} \cdot r_m$$

and

$$H_i = f(z_{|S_1}) \cdots f(y_{|S_{i-1}}) f(y_{|S_i}) r_{i+1} \cdot r_m \ .$$

We can assume without loss of generality (up to a renaming of the indices) that $S_i = \{1, \ldots, l\}$. Then we can see $z \in \{0,1\}^t$ as a pair (x,y) where $x = z_{|S_i} \in \{0,1\}^l$ and $y = z_{|[t]-S_i} \in \{0,1\}^{t-l}$. For every $j < i$ and $z = (x,y)$, let us define $f_j(x,y) = f(z_{|S_j})$: note that $f_j(x,y)$ depends on $|S_i \cap S_j| \leq \log m$ bits of x and on $l - |S_i \cap S_j| \geq l - \log m$ bits of y. With this notation we have

$$\begin{aligned} &\mathop{\mathbb{P}}_{x,y,r_{i+1},\cdots,r_m}[D'(f_1(x,y), \ldots, f_{i-1}(x,y), f(x), \ldots, r_m) = 1] \\ &- \mathop{\mathbb{P}}_{x,y,r_{i+1},\cdots,r_m} D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1] > \epsilon/m \end{aligned}$$

That is, when D' is given a string that contains $f_j(x,y)$ for $j < i$ in the first $i-1$ entries, and random bits in the last $m-i$ entries, then D' is more likely to accept the string if it contains $f(x)$ in the i-th entry than if it contains a random bit in the i-th entry. This is good enough to (almost) get a predictor for f. Consider the following algorithm:

```
Algorithm $A$
Input: $x \in \{0,1\}^l$
Pick random $r_i, \ldots, r_m \in \{0,1\}$
Pick random $y \in \{0,1\}^{t-l}$
Compute $f_1(x,y), \ldots, f_{i-1}(x,y)$
If $D'(f_1(x,y), \ldots, f_{i-1}(x,y), r_i, \ldots, r_m) = 1$ output $r_i$
Else output $1 - r_i$
```

Let us forget for a moment about the fact that the step of computing $f_1(x,y),\ldots,f_{i-1}(x,y)$ looks very hard, and let us check that A is good predictor. Let us denote by $A(x,y,r_1,\ldots,r_m)$ the output of A on input x and random choices $y,r_1,\ldots,r_m$.

$$\begin{aligned}
& \mathop{\mathbb{P}}_{x,y,r}[A(x,y,r)=f(x)] \\
= & \mathop{\mathbb{P}}_{x,y,r}[A(x,y,r)=f(x)|r_i=f(x)]\mathop{\mathbb{P}}_{x,r_i}[r_i=f(x)] \\
& +\mathop{\mathbb{P}}_{x,y,r}[A(x,y,r)=f(x)|r_i\neq f(x)]\mathop{\mathbb{P}}_{x,r_i}[r_i\neq f(x)] \\
= & \frac{1}{2}\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)=r_i] \\
& +\frac{1}{2}\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=0|f(x)\neq r_i] \\
= & \frac{1}{2}+\frac{1}{2}\Bigg(\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)=b] \\
& -\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)\neq b]\Bigg) \\
= & \frac{1}{2}+\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)=b] \\
& -\frac{1}{2}\Bigg(\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)=b] \\
& +\mathop{\mathbb{P}}_{x,y,r}[D'(f_1(x,y),\ldots,f_{i-1}(x,y),r_i,\ldots,r_m)=1|f(x)\neq b]\Bigg) \\
= & \frac{1}{2}+\mathbb{P}[D'(H_i)=1]-\mathbb{P}[D'(H_{i-1})=1] \\
\geq & \frac{1}{2}+\frac{\epsilon}{m}
\end{aligned}$$

So A is good, and it is worthwhile to see whether we can get an efficient implementation. We said we have

$$\mathop{\mathbb{P}}_{x,y,r_i,\ldots,r_m}[A(x,y,r)=f(x)]\geq\frac{1}{2}+\frac{\epsilon}{m}$$

so there surely exist fixed values $c_i,\ldots,c_m$ to give to $r_i,\ldots,r_m$, and a fixed value w to give to y such that

$$\mathop{\mathbb{P}}_{x,r}[A(x,w,c_i,c_{i+1},\ldots,c_m)=f(x)]\geq\frac{1}{2}+\frac{\epsilon}{m}$$

At this point we are essentially done. Since w is fixed, now, in order to implement A, we only have to compute $f_j(x,w)$ given x. However, for each j, $f_j(x,w)$ is a function that depends only on $\leq \log m$ bits of x, and so is computable by a circuits of size $O(m)$. Even composing $i-1<m$ such circuit, we still have that the sequence $f_1(x,w),\ldots,f_{i-1}(x,w),c_i,c_{i+1},\ldots,c_m$ can be computed, given x, by a circuit C of size $O(m^2)$. Finally, we notice that at this point $A(x,w,c)$ is doing the following: output the xor between c_i and the complement of $D'(C(x))$. Since c_i is fixed, either $A(x,w,c)$ always equals $D(C(x))$, or one is the complement of the other. In either case the Lemma follows. □

At this point it should not be too hard to derive Theorem 4.