Derandomization from Worst-Case Assumptions: Error-correcting codes and worst-case to average-case reductions.

April 3, 2006

From last time: We had the following assumption (called assumption 4 in that handout)

Definition: $CC_\rho(f) \geq s$ if s-sized circuits can compute f with probability at most ρ for a random input. That is, for every circuit family $\{C_n\}$ with $|C_n| \leq s(n)$, $\Pr_{x \leftarrow_R \{0,1\}^n}[C_n(x) = f(x)] < \rho$. If $CC_{1-1/(100n)}(f) \geq s$ we say that f is "mildly hard on the average" for s-sized circuits (every circuit will fail on a $1/(100n)$ fraction of the inputs) and if $CC_1(f) \geq s$ we say that f is "worst-case hard" for s-sized circuits (every circuit will fail on at least one input).

Assumption 1: $\exists f \in \mathbf{E}$ such that $CC_{1-1/(100n)}(f) \geq 2^{n^\epsilon}$. That is, for every large enough n and 2^{n^ϵ} sized circuit C,

$$\Pr_{x \leftarrow_R \{0,1\}^n}[C(x) = f(x)] \leq 1 - \frac{1}{100n}$$

We showed (using Yao's XOR Lemma + Nisan-Wigderson generator) that Assumption 1 implies $\mathbf{BPP} = \mathbf{QuasiP}$.

Today: We'll prove that Assumption 1 is implied by a seemingly much weaker assumption on *worst-case hardness* of functions. That is, we'll prove the following theorem:

Theorem 1 (BFNW)**.** *If $\exists f \in \mathbf{E}$ such that $CC(f) \geq 2^{n^\epsilon}$ for some $\epsilon > 0$ then Assumption 1 holds. In particular, in this case $\exists f \in \mathbf{E}$ such that $CC_{1-1/(100n)}(f) geq 2^{n^\epsilon/100}$.*

Aside: error correcting codes We'll move now to a seemingly irrelevant topic, that is interesting in its own right: the problem of *error correcting codes*:

Suppose Alice needs to record a string $f \in \{0,1\}^n$ so that she's able to retrieve it later, but the medium she uses (e.g., magnetic/optical disk) is *unreliable* and as much as 10% of the bits may be changed to an arbitrary value. How can she store f so that she'll be able to retrieve it later?

The basic idea is to use *redundancy*, but the obvious repetition idea does not really work: that is, suppose Alice writes down the string $\hat{f} \in \{0,1\}^{10n}$ where each bit in f is replicated 10 times in $\hat{f}$. When reading a corrupted version of $\hat{f}$ one might hope that since only 10% of the bits are changed, for every $i \in [n]$ the majority of the 10 repetitions of the bit f_i still contains the right value f_i. However, since the places where $\hat{f}$ is corrupted are arbitrary, there is no reason to believe this is true. Indeed, by only corrupting 10 bits, one may ensure that the value of the first bit f_1 is lost.

It may seem that this problem is hopeless, but Shannon showed that it may be in fact actually solveable. He suggested the notion of an *error correcting code*, which is a function $E : \{0,1\}^n \to \{0,1\}^m$ satisfying the following notion: for every $f \neq f' \in \{0,1\}^n$, $dist(E(f), E(f')) \geq 0.4$ where $dist(y, y')$ is the fraction of i's in $[m]$ such that $y_i \neq y'_i$.[1]

The idea is the simple: encode f using $\hat{f} = E(m)$. Now, let g be a corrupted version of $\hat{f}$ with at most 10% corruption, that is $dist(g, \hat{f}) \leq 0.1$. Then, for every other $f' \neq f$, $dist(g, E(f')) \geq 0.3$ since by the triangle inequality

$$dist(E(f), E(f')) \leq dist(E(f), g) + dist(g, E(f'))$$

and so f is the *unique* string such that $dist(E(f), g) < 0.2$ and thus at least in principle it can be recovered from g.

Shannon showed that such a function E exists from $\{0,1\}^n$ to $\{0,1\}^{10n}$ using the probabilistic method (in fact he calculated exactly the length of the output required as a function of the input).[2] This follows by essentially the Chernoff bounds that show that for such a random E the probability that the distance of $E(f)$ and $E(f')$ is less than 0.4 is so small that we can take a union bound over all the $< 2^{2n}$ pairs f, f'. However, as we know, such a function E will have complexity 2^n and for practical applications one needs a function E that is efficiently (i.e., polynomial-time) computable. In fact, we need E to also be *efficiently decodable* which means that we need an efficient algorithm that given g with $dist(g, E(f)) < 0.1$ finds f.

Larger alphabets How do we get such efficient codes? For starters, we'll make our job easier, and think of a *larger* alphabet of symbols than $\{0,1\}$. That is, for some set $\mathbb{F}$, we'll ask for a function $E : \mathbb{F}^n \to \mathbb{F}^m$ such that for every $f \neq f' \in \mathbb{F}^n$, $dist(E(f), E(f')) > 0.1$ ($dist$ is again defined to be the fraction of i's such that $E(f)_i \neq E(f')_i$). To see that this is an easier problem, think of the case that $|\mathbb{F}| > 2^{10}$: if we let $bin(g)$ be the representation of $g \in \mathbb{F}^m$ as a binary string in the obvious way, then we 10% of the bits by changing one bit in every one of the $\log |\mathbb{F}|$ bits that represent one symbol in g. Let's call this changed string y. We get that $dist(bin(g), y) \leq 0.1$ but $dist(g, bin^{-1}(y)) = 1$. (Note that we made the problem easier by moving to outputs in $\mathbb{F}^m$ and appropriate distance definition. It wouldn't have changed anything if we considered inputs in $\{0,1\}^{\log |F| n}$ instead of $\mathbb{F}^n$.)

Reed-Solomon codes We now define one of the most famous codes:

Choose $\mathbb{F}$ to be a field of size m (for $m > n$) and for every $f \in \mathbb{F}^n$ think of $f = (f_0, \ldots, f_{n-1})$ as defining a polynomial of degree $n-1$ over $\mathbb{F}$: that is, $f(x) = f_0 + f_1 x + \cdots + f_{n-1} x^{n-1}$. Define $E(f) = f(0), \ldots, f(m-1)$.

The well known polynomial-lemma implies that $dist(E(f), E(f')) \geq 1 - \frac{n-1}{m}$. For example, if $m > 5n$ then this is more than 0.8. This code is definitely efficiently computable, and there's also an efficient *decoding* algorithm (that we'll not show).

We'll show a decoding algorithm for the somewhat easier case where we are given a $g \in \mathbb{F}^m$ such that $dist(g, E(f)) < \frac{1}{10n}$. In this case we choose at random $x_1, \ldots, x_n$ and look at the values $y_1, \ldots, y_n$ where $y_i = g(x_i)$. With probability ≥ 0.9, for all these values $y_i = f(x_i)$. In this case, we can use *polynomial interpolation* to recover the polynomial f by solving a system of n linear equations (with the unknowns being the coefficients of the polynomial).

[1] Note that this condition really only refers to the *image set* of $E(\cdot)$. Thus, often in the coding literature a *code* is defined not as a function from $\{0,1\}^n \to \{0,1\}^m$ but as a subset of $\{0,1\}^m$ with cardinality at least 2^n.

[2] We note that Shannon was more interested in *everage* errors, and the notion of *worst-case* errors we present here is due to Hamming.

Reducing the alphabet Here's an idea to reduce the alphabet of the code's output: suppose you have a code E_1 from $\mathbb{F}^n$ to $\mathbb{F}^m$ and a code E_2 from $\{0,1\}^{\log|\mathbb{F}|}$ (which we can identify with $|\mathbb{F}|$) to $\{0,1\}^k$. Then we can define a code $E : \mathbb{F}^n \to \{0,1\}^{km}$ as follows: let $E(f)$ be the concatenation of $E_2(E_1(f)_i)$ for every $i \in [m]$. It's not hard to show that $dist(E) \geq dist(E_1)dist(E_2)$ (where $dist(E)$ is defined as the minimum over all $x \neq x'$ of $dist(E(x), E(x'))$), and that at least as far as encoding goes, E is efficient if E_1 and E_2 were.

However, this doesn't seem to help us reduce alphabet to binary since it's not clear where we are supposed to get the code E_2 in the first place! However, the crucial observation is that we need E_2 to have much shorter input length. For example, if E_1 is the Reed-Solomon code that maps $\mathbb{F}^n$ to $\mathbb{F}^{5n}$ for $|\mathbb{F}| = 5n$, we need $E_2 : \{0,1\}^{\log(5n)} \to \{0,1\}^k$ and thus it is enough to have a code that runs in time *exponential* in its input.

We'll now show such a code that was found very useful in many applications.

Hadamard code The Hadamard code H is a map from $\{0,1\}^\ell$ to $\{0,1\}^{2^\ell}$ defined as follows: for $x, y \in \{0,1\}^\ell$, the y^{th} bit of $H(x)$ is $\langle x, y\rangle \pmod 2$. We've already seen before the claim that $dist(H) \geq 0.5$: that is, that for every $x \neq x'$, the probability over a random y that $\langle x, y\rangle \neq \langle x', y\rangle$ or equivalently that $\langle x \oplus x', y\rangle = 1$ is $\frac{1}{2}$.

Thus, we can concatenate the Hadamard code with the Reed-Solomon code to obtain a code mapping $|\mathbb{F}|^n$ to $\{0,1\}^{O(n^2)}$ that can handle 10% corruption (actually this approach can get us arbitrarily close to 25% corruptions). With a bit more work (using a different code than Hadamard) we can have a code that maps $\{0,1\}^n$ to $\{0,1\}^{O(n)}$ with similar parameters.

Back to hardness amplifications What does any of the above have to do with hardness amplification? Think of what we're trying to do:

We have a function f and we want to convert it to a function $\hat{f}$ such that if someone solves $\hat{f}$ on a $1 - 1/(10n^2)$ fraction of the inputs then we can solve f in the worst-case. An algorithm that solves $\hat{f}$ on such a fraction of the inputs can be thought of as giving us a *corrupted version* of $\hat{f}$ and so what we're trying to do is *decode* it to obtain back the function f!

Local decoding The only problem in the above reasoning is that while naturally a decoding procedure to obtain f from a corrupted version of $\hat{f}$ takes $\text{poly}(|\hat{f}|)$ time, this is something we cannot afford in this context.

The reason is that the function f, as a string, is of length 2^n, and so $\hat{f}$ will be of at least this size. This means that polynomial in $|\hat{f}|$ is $2^{O(n)}$ time which is not very interesting: of course we can compute f if we were given that much time!

it may seem that we're back to square one and this really wasn't such a good way to think about functions, but in fact, this approach can still be salvaged: the idea is to use *local decoding*. A *lcoal decoding* procedure is a procedure that does not try to recover all of f in one shot, but rather is given an index $i \in |f|$ and computes f_i given oracle access to a corrupted version of $\hat{f} = E(f)$ and only runs in time $\text{polylog}(|\hat{f}|)$. One can see that a local decoding procedure is exactly what we need: it means that given black-box access to some algorithm that computes $\hat{f}$ on a good fraction of the inputs, we can actually come up with an algorithm of similar efficiency that computes f on *every* input.

Note that we don't need to encoding algorithm E to be local: polynomial in $|f|$ is enough, since we can afford to have $\hat{f}$ computable in $2^{O(n)}$ (after all, all we need is to have $\hat{f} \in \mathbf{E}$). In fact it can be shown that an encoding algorithm can *not* be local and needs to have bits of

the output depend on many bits of the input (which is one reason why we haven't been able to prove similar results for lower complexity classes such as **NP**).

The actual construction We're now ready to show how we can convert a function $f \in \mathbf{E}$ that is hard on the worst case for s sized circuits, to a function $\hat{f} \in \mathbf{E}$ that is hard to solve on a $1-1/(10n^2)$ fraction of its inputs for circuits of size $s' = s/\mathrm{poly}(n)$. We'll follow the approach above by using first a code with non-binary output (not Reed-Solomon but a related cousin) and then combine this with Hadamard to get a function with binary output.

Multilinear extension Let $f : \{0,1\}^n \to \{0,1\}$ be some function. Let $\mathbb{F}$ be some field. Since any function can be represented as a polynomial of some degree, we can find an n-variable polynomial $\hat{f} : \mathbb{F}^n \to \mathbb{F}$ such that for every $x_1, \ldots, x_n \in \{0,1\}$, $\hat{f}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$. In fact, since for every $x \in \{0,1\}$ and $k \geq 1$, $x^k = x$, we can find such a polynomial that is *multilinear*: that is the degree of each variable x_i in the polynomial is at most one.

We Choose $|\mathbb{F}| = 2n$ and make the following claim

Lemma 2. *If there's an s' sized circuit C that computes $\hat{f}(x_1, \cdots, x_n)$ with probability $\geq 1 - 1/(10n)$ over $x_1, \ldots, x_n$ then there's $s = s'\mathrm{poly}(n)$ circuit C' that computes f on* every input*.*

Proof. We'll make the following stronger claim: if there's a circuit C like that then we actually have a circuit C' that computes $\hat{f}$ on every input. Since $\hat{f}$ agrees with f on inputs in $\{0,1\}^n$ this proves the lemma. (This property that we can correct a typically successful algorithm for $\hat{f}$ to an always successful algorithm is called *self correction*.)

We'll show a *probabilistic* algorithm that computes $\hat{f}(\vec{x})$ for every $\vec{x} \in \mathbb{F}^n$ with probability at least 0.9. This can be converted into a deterministic circuit in the standard way (as in the proof that $\mathbf{BPP} \subseteq \mathbf{P}_{/\mathbf{poly}}$.

Let $\vec{x} \in \mathbb{F}^n$. We'll choose $\vec{y} \leftarrow_{\mathrm{R}} \mathbb{F}^n$, and for $i = 0, 1, \ldots, n+1$ (we treat these as elements of $\mathbb{F}$ with 0 being the zero element of $\mathbb{F}$), let $\vec{x}^i = \vec{x} + i\vec{y}$. Note that $\vec{x}^0 = \vec{x}$ and that for every $i \neq 0$, $\vec{x}^i$ is distributed uniformly over $\mathbb{F}^n$. For every $i \in \{1, \ldots, n+1\}$, we compute $\vec{y}^i = C'(\vec{x}^i)$. Then, by the union bound with probability 0.9 $\vec{y}^i = \hat{f}(\vec{x}^i)$ for *all* $i = 1, \ldots, n+1$ (let's call this event $GOOD$).

Now, considering $\vec{x}$ and $\vec{y}$ as fixed, consider the function $p : \mathbb{F} \to \mathbb{F}$ defined as follows: $p(i) = \hat{f}(\vec{x} + i\vec{y})$. This function is a degree $\leq n$ polynomial in i. Hence, from the values $p(1), \ldots, p(n+1)$ we can recover, using polynomial interpolation, all the coefficients of $p(\cdot)$ and in particular recover $p(0) = \hat{f}(\vec{x})$. We see that if $GOOD$ happens (which is the case with ≥ 0.9 probability) we manage to obtain $\hat{f}(\vec{x})$. □

Binary output As mentioned above, to get a function with binary input from $\hat{f} : \mathbb{F}^n \to \mathbb{F}$ we define $\tilde{f}(\vec{x}, r) = \langle \hat{f}(\vec{x}), r \rangle \pmod 2$. That is, $\tilde{f} : \mathbb{F}^n \times \{0,1\}^{\log |F|} \to \{0,1\}$.

We'll show that if $CC_{1-1/(10n)}(\hat{f}) \geq s(n)$ then $CC_{1-1/(100n)}(\tilde{f}) \geq s(n)/\mathrm{poly}(|\mathbb{F}|)$.

That is, we need to show that if $\tilde{C}$ computes $\tilde{f}$ with probability better than $1 - 1/(100n)$ then there's a circuit $\hat{C}$ with $|\hat{C}| \leq \mathrm{poly}(|\mathbb{F}|)|\tilde{C}|$ that computes $\hat{f}$ with probability at least $1 - 1/(10n)$.

We'll define $\hat{C}$ as follows: to compute $\hat{f}(\vec{x})$, compute $\tilde{C}(\vec{x}, r)$ for every $r \in \{0,1\}^{\log |F|}$ to obtain a string $\vec{z} \in \{0,1\}^{|F|}$. Now, for every $a \in \mathbb{F}$, let $\vec{w}_a$ be the Hadamard encoding of a.

If there's an a such $dist(\vec{w}_a, \vec{z}) < 0.25$ then output a (otherwise output something arbitrary). Note that by the distance of the Hadamard code, there'll be at most one such a.

To prove that $\hat{C}$ works note that by our assumption on $\tilde{C}$, there must be at least a $1-1/(10n)$ of the $\vec{x}$'s such that $\tilde{C}(\vec{x}, r) = \tilde{f}(\vec{x}, r)$ with probability $\geq 9/10$ over r. For every such $\vec{x}$, $\hat{C}$ will return the right value.

Recap We've by now proved the following theorem: If $\exists f \in \mathbf{E}$ and $\epsilon > 0$ with $CC(f) \geq 2^{n^\epsilon}$ then $\mathbf{BPP} \subseteq \mathbf{QuasiP}$. This follows from the following three steps:

- Using error-correcting codes (specifically multilinear extension and Hadamard) we moved from a function f that is hard for subexponential circuits to compute in the worst case, to a function $\tilde{f}$ that is hard for such circuits to compute with probability better than $1 - 1/(100n)$.
- Using the XOR lemma we moved from such a function $\tilde{f}$ into a function $\overline{f}$ that is hard for subexponential circuits to compute with probability better than $\frac{1}{2} + 2^{-n^{\epsilon'}}$ for some constant $\epsilon' > 0$.
- The NW generator uses such a function $\overline{f}$ to contruct a $2^{O(\ell)}$-time generator that maps ℓ bits to $2^{\ell^{\epsilon''}}$ bits and is indistinguishable by $2^{\ell^{\epsilon''}}$-size circuits. To derandomize a time-t algorithm A on input x, convert $r \mapsto A(x, r)$ to a circuit C of size t^2, set $t = 2^{\ell^{\epsilon''}}$ and you'll get an $2^{O(\ell)} = 2^{\mathrm{polylog}(t)}$ time derandomization.

Getting to BPP = P There are several approaches to get to $\mathbf{BPP} = \mathbf{P}$. One interesting approach is to skip the XOR Lemma and get using error correcting codes directly from a function f that is worst case hard for $2^{\epsilon n}$-sized circuits to a function $\hat{f}$ that is $\frac{1}{2} + 2^{\epsilon' n}$-hard for $2^{\epsilon' n}$ circuits.

However, it may seem quite strange that we're able to get even to a function that is 0.6-hard using error correcting codes. The reason is that if we go back to the coding analogy, one can see that if we want a code with at most a polynomial expansion then the code's distance must be at least 0.5 (in fact, the distance needs to be at most $0.5 - \epsilon$ for some constant $\epsilon > 0$, and if we want any non-trivial code, even allowing exponential or double exponential expansion we cannot go to, say, distance 0.51).

If we take a code $E : \{0,1\}^n \to \{0,1\}^m$ with distance ≤ 0.5 and take two strings $x \neq x'$ such that $dist(E(f), E(f')) \leq 0.5$ then we can find $g \in \{0,1\}^m$ such that $dist(E(f), g) \leq 0.25$ and $dist(g, E(f')) \leq 0.25$ (on half the bits $E(f)$ and $E(f')$ do not agree, take g to agree with $E(f)$ and on the other take g to agree with $E(f')$). This means that in general, given a string g, there's no *unique* f such that $E(f)$ is in 75% agreement with g.

This seems to imply that we won't be able to transform an algorithm computing $\hat{f} = E(f)$ with 75% success into an algorithm computing f on all inputs. However, it turns out that this is not the case: the reason is that even though, for a g agreeing with $\hat{f}$ on 75% of the inputs there may be other inputs $f_1, \ldots, f_k$ that have similar agreement with g, *there can't be too many of them!*. In fact, we have the following theorem:

Theorem 3. *Let $E : \{0,1\}^n \to \{0,1\}^m$ be a code with $dist(E) \geq 1/2 - \delta$. Then, for every $\rho > 2\delta$, and $g \in \{0,1\}^m$ there exist at most $k = O(1/\rho^2)$ distinct inputs $f_1, \ldots, f_k$ with $dist(f_i, g) \leq 1/2 - \rho$*

An algorithm that given g outputs this list of potential decodings $f_1, \ldots, f_k$ is called a *list decoding* algorithm. In recent years (motivated by complexity theory), people found efficient list decoding algorithms for a variety of popular codes including the Reed-Solomon code (by Sudan) and the Hadamard code (by Goldreich and Levin, this is known in cryptography as the hard-core bit theorem). We also have a notion of *local list decoding* which is that given oracle access to a g that agrees with $\hat{f} = E(f)$, we can compute the function $i, x \mapsto f_i(x)$ where there is some $i \in [k]$ with $f_i = f$ (in fact, this is what the Goldreich-Levin theorem provides for the Hadamard code, and using the Reed-Solomon list decoding, one can provide such an algorithm for multilinear extension and other related codes). How do we make a circuit for f out of this? the answer is to simply use *non-uniformity* to"hardwire" into the circuit the right value of i. Using more efficient codes than multilinear extension (with only polynomial expansion) and setting $\rho = 2^{\epsilon n/100}$ yields the right result.

List decoding in practice List decoding turns out to be very useful in practical applications as well. Indeed, it's often the case that given a few candidates to the uncorrupted version, it's easy to find the right one (perhaps there's only one input in the list that makes sense, or satisfies some integrity check). Thus, list decoding algorithms enable also practical error recovery from a much larger fraction of errors that was previously thought possible.

Black-box proofs Examining all the proofs so far, we see that they only use the assumed hard function in a *black-box way*. This turns out to be important for several applications:

- We can get that for any oracle A, if $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(n)})$ can not be computed by subexponential circuits that have additional "oracle gates" that compute A, then with have a pseudorandom generator against such circuits. In particular, setting A to be $3SAT$, we can use such a pseudorandom generator to show that if **EXP** can not be computed by subexponential circuits with $3SAT$-gates then $\mathbf{AM} = \mathbf{MA} = \mathbf{NP}$.
- Thinking of f as a function that comes from a high entropy distribution, similar reasoning can show that the NW generator applied to f will yield an output that is *statistically* indistinguishable from uniform. This has been used to construct *randomness extractors*. (See web site and book for more info.)

Obtaining a uniform result If $\mathbf{EXP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ then we get a subeponential derandomization of **BPP**. As mentioned before, we can actually obtain some partial derandomization from the much weaker and uniform assumption that $\mathbf{BPP} \neq \mathbf{EXP}$. Namely, we get a subexponential derandomization that works for infinitely many input lengths and may fail for some inputs, but it's infeasible for a polynomial algorithm to find these inputs.

This result was obtained by Impagliazzo and Wigderson in 98. The idea is the following (see also a different proof by Vadhan and Trevisan):

- If $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ then such a result already follows from what we've seen, and so we assume that $\mathbf{EXP} \subseteq \mathbf{P}_{/\mathbf{poly}}$ but $\mathbf{EXP} \neq \mathbf{BPP}$.
- A basic component is the fact that **EXP** has a *multi-prover* interactive proof in which the prover algorithm can also be implemented in **EXP**. This will follow from a variant of the **PCP** theorem.
- This implies that $\mathbf{EXP} = \mathbf{MA}$ since for any $L \in \mathbf{EXP}$, Merlin can prove that $x \in L$ by sending to Arthur the polynomial-sized circuit C for computing the provers strategy

in the multi-prover proof for L. Arthur can then run these circuits on his own and be convinced.

- In particular, since **MA** is in the polynomial hierarchy, because of Toda's theorem we get that the *permanent* is complete for **EXP**. We'll now run the NW generator using the permanent as a hard function.
- The two nice properties we use about the permanent are the following: (both properties are shown in the book chapter on interactive proofs)
  - It is randomly self reducible: if you can solve it on a random input you can solve it on all inputs. (This follows from the fact it's a low degree polynomial.) Thus, if it's hard on the worst-case it's also hard on the average.
  - It is downward self reducible: given polynomial-time nad access to an oracle that solves it on inputs of size $< n$ you can solve it on inputs of size n. (This follows from a formula, similar to determinant, that reduces computing the permanent of an n by n matrix A to computing the pernmanent of n submatrices of A of size $n-1 \times n-1$.)
- To get rid of the (seemingly quite inherent) non-uniformity in the proof of the NW generator, we do the following:
  - If there's a t-time algorithm P that breaks the generator, then, instead of trying to come up with a poly(t)-time algorithm to solve the permanent, we'll use P to solve the following seemingly easier task: Given a circuit C_{n-1} that computes the permanent for $n-1$ by $n-1$ matrices, find, *in a uniform way*, a circuit C_n of size poly$(t(n))$ circuit that computes the permanent for n by n matrices. Note that the size of the circuit C_n *does not* depend on the size of C_{n-1} and so if we can solve this task, running this procedure recursively, we can actually solve the permanent in polynomial-time.
  - The idea is to look at the proof of the NW generator: the only place where we needed non-uniformity was to obtain values of the hard function on some fixed inputs. However, since the permanent is downard self-reducible, we can obtain these values using the circuit C_{n-1}. Once we obtained them, we hardwire these values (and not the description of the circuit) to follow the proof of the NW generator and obtain from P a circuit that computes the permanent with a noticeable advantage. We then use the random self-reducibility of the permanent to obtain a circuit that computes the permanent on all inputs.

Derandomization implies lower bounds The above reasoning might lead us to hope that we can forget about circuits, and perhaps get an unconditional result that, say, **BPP** $\neq$ **EXP**. However, it turns out that this result would imply circuit lower bounds: see the survey by Kabanets and the papers by Impagliazzo-Kabanets-Wigderson and Impagliazzo-Kabanets for more information.

More resources There are *many* areas of derandomization and pseudorandomness we did not cover. The Kabanets survey, the book, and the links on the web site contain pointers to many of these. I encourage you to take a look.