Derandomizing BPP

Lecture Notes of a course by Avi Wigderson

Summarized by Ronen Shaltiel

Contents

1 A Tale of two Generators **2**
- 1.1 Introduction . . . 2
  - 1.1.1 Definitions . . . 2
  - 1.1.2 Generators . . . 3
  - 1.1.3 Hardness Vs. Randomness . . . 4
- 1.2 The Blum-Micali-Yao Generator . . . 4
  - 1.2.1 One way functions . . . 4
  - 1.2.2 Hard Bits and the Goldreich-Levin theorem . . . 5
  - 1.2.3 Proof of the Goldreich-Levin theorem . . . 7
  - 1.2.4 Prediction tests Vs. Statistical tests . . . 10
  - 1.2.5 The Generator's construction . . . 13
- 1.3 The Nisan-Wigderson Generator . . . 15
  - 1.3.1 One way functions Vs. Unapproximable functions . . . 15
  - 1.3.2 Nearly disjoint sets . . . 16
  - 1.3.3 Constructing the generator . . . 17
  - 1.3.4 A generator for constant depth circuits . . . 20

2 Using a worst case complexity assumption **21**
- 2.1 Worst case complexity Vs. Distributional complexity . . . 21
- 2.2 Randomness Vs. Non-Uniformity . . . 22
  - 2.2.1 Distributional Complexity . . . 22
  - 2.2.2 Worst case complexity . . . 22
- 2.3 From Worst-case hard to slightly unapproximable . . . 24
  - 2.3.1 Transforming a boolean function into a polynomial . . . 24
  - 2.3.2 Random self reducibility . . . 25
- 2.4 Hardness Amplification . . . 27
  - 2.4.1 The concept . . . 27
  - 2.4.2 Xor lemmas and Product theorems . . . 27
  - 2.4.3 Proof of product theorem . . . 29
- 2.5 The main theorem . . . 32

Chapter 1

A Tale of two Generators

1.1 Introduction

In the previous chapters, we studied the power of randomness in several settings. In this chapter we address the more general question. What is the power of randomized algorithms? Are they more powerful than deterministic ones? The current state of affairs is that there are computation problems for which we have polynomial randomized algorithms, and only exponential time deterministic algorithms. We don't know weather randomness can help save running time, and if it does, at what rate.
Our aim is to take any randomized algorithm that runs in polynomial time, and simulate it using a deterministic algorithm, trying to minimize the running time of the deterministic algorithm. We will call such simulations "derandomization of BPP".

1.1.1 Definitions

For completeness we repeat definitions from previous chapters. The letter n receives special treatment, and all functions that appear in the text are assumed to be functions of n if not otherwise defined. We are not consistent in the notation, it is sometimes tedious to explicitly state the dependence of parameters on n. We often speak about families of circuits, functions, and such. We keep the convention that for a family $f = \{f_n\}$ the domain of f_n is $\{0,1\}^n$. The meaning of "$f = \{f_n\}$ has property P", is "there exist some n_0 such that for all $n > n_0$, f_n has the property P_n".

Definition 1.1 $dtime(t) = \{L|$ *there exists a deterministic Algorithm which runs in time* $O(t)$ *and accepts* $L\}$.

Definition 1.2 $bptime_\epsilon(t, r) = \{L|$ *there exists a probabilistic Algorithm which runs in time* $O(t)$*, uses* $O(r)$ *random bits, and accepts* L *with two sided error of at most* $\epsilon\}$.

Definition 1.3 $P = \cup_{1 \le c < \infty} dtime(n^c)$.

Definition 1.4 $BPP = \cup_{1 \le c < \infty} bptime_{1/3}(n^c, n^c)$.

Definition 1.5 $EXP = \cup_{1 \le c < \infty} dtime(2^{n^c})$.

1.1.2 Generators

The obvious tradeoff between randomness and running time is the following:

Theorem 1.6 *for all* $\epsilon < \frac{1}{2}$, $bptime_\epsilon(t, r) \subseteq dtime(2^r \cdot t)$.

Proof: To make a randomized algorithm deterministic, simply run it (given an input x) using all possible strings of length r. Accept x iff the majority of the runs accepted x.

•

Corollary 1.7 $BPP \subseteq EXP$

Our goal is to simulate BPP using less time. Theorem 1.6 shows us that if we could simulate a randomized algorithm using less random bits, we could derandomize BPP. A generator is a machine that takes as input m random bits, and outputs n "pseudo-random" bits ($m < n$). the exact meaning of "Pseudo-Randomness" depends on the entity which receives the bits. They should look random to the receiver. Note that randomness is now defined not as a property of the probability distribution, but rather it is "in the eyes of the beholder"!

Definition 1.8 *A* (m, l, s, ϵ)*-Generator* G *is a family of functions* $\{G_n\}_{1 \leq n < \infty}$ *such that for all n:*

- $G_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n$.
- G_n *is computable in time* $l(n)$.
- *for all circuits* $C = \{C_n\}$ *of size* $s(n)$*:*

$$|Pr_{x \in_R \{0,1\}^n}(C(x) = 1) - Pr_{y \in_R \{0,1\}^m}(C(G_n(y)) = 1)| \leq \epsilon$$

Theorem 1.9 *if there exists a* (m, l, t^2, δ)*-Generator then*

$$bptime_\epsilon(t, r) \subseteq bptime_{\epsilon+\delta}(l(r) + t, m(r))$$

Proof: Given a randomized algorithm A that runs in time $t(n)$, and uses $r(n)$ random bits. construct a randomized algorithm B which uses $m(r(n))$ random bits denoted z. It computes $y = G_{r(n)}(z)$ and runs $A(x, y)$. this takes $t(n) + l(r(n))$ steps. If this scheme fails, then there is some n and x' of length n, on which B errs with probability at least $\epsilon + \delta$. We know that A errs with probability no more than ϵ. There is a standard construction that takes a (uniform) algorithm, and transforms it into a family of boolean circuits, Where the size of the circuit is t^2, where t is the running time of the algorithm. Let $C(x, y)$ be the circuit that is built from $A(x, y)$. We now use the non uniformity of boolean circuit to define the circuit $C_{x'}(y) = C(x', y)$, which is of the same size as C. Since The computation of $C_{x'}$ is identical to that of $A(x, \cdot)$, we get:

$$|Pr_{y \in_R \{0,1\}^n}(C_{x'}(y) = 1) - Pr_{z \in_R \{0,1\}^m}(C(G_n(z) = 1)| > \delta$$

Which is a contradiction.

•

Remark 1.10 *Note that our goal is to fool BPP Algorithms. Yet in order to achieve this, we need a generator that fools boolean circuits (of size roughly the running time of the algorithms we want to fool). This is an obstacle we will face the rest of this chapter. In order to fool uniform computation, we need to fool non-uniform one.*

Corollary 1.11 *if there exists a* (m, l, t^2, δ)*-Generator then*

$$bptime(t, r) \subseteq dtime(2^{m(r)}(l(r) + t))$$

Proof: Apply theorem 1.9 and 1.6 in sequence.

•

1.1.3 Hardness Vs. Randomness

Unlike the previous chapters, there are currently no constructions of generators that fool boolean polynomial circuits. We may find some comfort in the fact that the missing ingredient, that keeps us from achieving such constructions, is the standard hurdle confronted by complexity theory. That is, the failure to prove the existence of hard functions. (functions that cannot be computed by polynomial size circuits). Because of this difficulty, all the results we will prove will have the following form:

the existence of hard functions
implies
the existence of "good" generators
implies
$BPP \subseteq$"sub-exponential time".

The next sections will be devoted to derandomizing BPP using as weak as possible unproven assumption.

1.2 The Blum-Micali-Yao Generator

Definition 1.12 *A generator G is called* δ*-fast (for* $\delta < 1$*) if it is a* $(n^\delta, n^c, n^d, \frac{1}{n^e})$*-generator for some constant c and all* d, e.

Using corollary 1.11, the existence of a δ-fast generator implies:

$$bptime_{\frac{1}{3}}(n^c, n^c) \subseteq dtime(2^{n^{c\delta}} n^{cd})$$

In this section we will build a δ-fast generator for all $\delta > 0$. (Under the assumption that some class of hard functions exist) This implies: $BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$.

1.2.1 One way functions

Our first example of "hard functions" are one way functions. These are functions that are easy to compute, but hard to invert.

Definition 1.13 *a family of functions $f = \{f_n\}$, $f_n : \{0,1\}^n \to \{0,1\}^n$ is called one-way if:*

- *there is a polynomial Algorithm that computes f.*
- *for every family of polynomial circuits $C = \{C_n\}$*

$$Pr_{x \in_R \{0,1\}^n}(C_n(f_n(x) \in f_n^{-1}(f_n(x))) \leq \epsilon_n$$

- $\lim_{n\to\infty} \epsilon_n \cdot n^c = 0$ *for all c.*

we will actually need the function to be one to one.

Definition 1.14 *A one way permutation f is a function that is both one way, and one to one. in such a case f^{-1} is a function, and the former inequality can be written in the following form:*

$$Pr_{y \in_R \{0,1\}^n}(C_n(y) = f^{-1}(y)) \leq \epsilon_n$$

One way function seem suitable for constructing a generator. take as input $y_0 \in \{0,1\}^m$, and set $y_i = f(y_{i-1})$ for $1 \leq i \leq k = poly(m)$. One property of a random string is that given any prefix of it, the remaining bits cannot be predicted. Note that given $y_k, y_{k-1}, .., y_i$ a polynomial circuit cannot predict y_{i-1} with non negligible success probability. So we may hope that y_{i-1} seems random to a polynomial circuit. this leads us to define: $G(y_0) = (y_k, .., y_0)$
However, there are several difficulties with this scheme:

1. It is indeed true that for a random y, $f^{-1}(y)$ cannot be computed by a polynomial circuit. Yet, this does not rule out the possibility that the first bit of $f^{-1}(y)$ can be computed by a polynomial circuit.
2. Is the unpredictability of the suffix of a string, given it's prefix enough to prove that the string is pseudo-random? (note that in our case, the y's are not at all pseudo-random when read in the opposite direction $(y_0, .., y_k)$).

The next few sections deal with these problems.

1.2.2 Hard Bits and the Goldreich-Levin theorem

As we have seen in the previous section, given a hard function $g : \{0,1\}^n \to \{0,1\}^n$, (in our case $g = f^{-1}$ when f is a one way permutation), it may be the case that some bits are not hard. For example, the "discrete log" function (which is conjectured to be one way), has an "easy" bit (the least significant bit) and a "hard" bit (the most significant bit). By "easy" bit we mean one that can be computed in polynomial time, by "hard" bit, we mean one that computing it is as hard as computing the entire function.
It would be nice, if every hard function had a hard bit. But why restrict ourselves to bits? Any polynomial function will do.

Definition 1.15 *Given a family of functions $g = \{g_n\}$, $g_n : \{0,1\}^n \to \{0,1\}^n$ a hard bit for g is a family of functions $b = \{b_n\}$, $b_n : \{0,1\}^n \to \{0,1\}$ such that given a family of circuits $B = \{B_n\}$ which achieves:*

$$Pr_{x \in_R \{0,1\}^n}(B_n(x) = b_n(g_n(x))) \geq \frac{1}{2} + \epsilon_n$$

there exists a family of circuits $A = \{A_n\}$ (that may use B as a sub-circuit), such that the following holds:

- $Pr_{x \in_R \{0,1\}^n}(A_n(x) = g_n(x)) \geq (\frac{\epsilon}{n})^{O(1)}$
- *A is of size* $(\frac{size(B) \cdot n}{\epsilon})^{O(1)}$

the function b captures the "hardness" of g. The parameters in definition 1.15 are defined to achieve the following result:

Lemma 1.16 *If f is a one way permutation, and b is a hard bit for f^{-1} then for every family of polynomial circuits $C = \{C_n\}$ and constant d:*

$$Pr_{y \in_R \{0,1\}^n}(C_n(y) = b_n(f_n^{-1}(y))) \leq \frac{1}{2} + \frac{1}{n^d}$$

Remark 1.17 *We choose to define hard bits for the "hard" (inverse) side of one way functions, because this is more general and can be applied to functions that are not one way.*

We are now left with the need to build hard bits for hard functions. The parity of all the output bits seems suitable, but this may fail if two "hard" bits "cancel" each other. (For example if their parity is constant). The solution is to take the parity of a random subset of bits. To make this formal, we need to extend the function.

Definition 1.18 *For a function $f : \{0,1\}^n \to \{0,1\}^n$ define $\hat{f} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$ in the following way:*

$$\hat{f}(x, y) = (f(x), y)$$

Note that:

- if f is a permutation, so is $\hat{f}$.
- Given a circuit for $\hat{f}$, one can construct a circuit of the same size for f, which achieves the same success probability. (This means that if f is a one way permutation, so is $\hat{f}$).
- $(\hat{f})^{-1} = (\widehat{f^{-1}})$

Definition 1.19 *The Goldreich-Levin function $GL_{2n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is defined as:*

$$GL_{2n}(x, y) = (\sum_{1 \leq i \leq n} x_i y_i) mod 2 = \bigoplus_{\{i | y_i = 1\}} x_i$$

It will sometimes be useful to us to think of GL_{2n} as inner product of n-bit vectors over $GF(2)$.

Theorem 1.20 *(Goldreich-Levin) For every function g, GL is a hard bit for $\hat{g}$.*

The Goldreich-Levin theorem is a very useful one. whenever we have a hard non-boolean function, we can build a boolean function that is almost as hard.

1.2.3 Proof of the Goldreich-Levin theorem

It is enough prove that given a family of circuits $B = \{B_n\}$ such that:

$$Pr_{x,y \in_R \{0,1\}^n}(B_{2n}(x,y) = GL_{2n}((g_n(x), y))) \geq \frac{1}{2} + \epsilon_n$$

we can construct a family of circuits $A = \{A_n\}$ such that:

$$Pr_{x \in_R \{0,1\}^n}(A_n(x) = g_n(x)) \geq (\frac{\epsilon}{n})^{O(1)}$$

and $size(A) = (\frac{size(B)n}{\epsilon})^{O(1)}$.

It is instructive to first prove this using slightly different parameters, and then extend the technique for the general case. So we start by proving the following lemma, (which will actually be required as a special case of the Goldreich-Levin theorem in the next sections).

Lemma 1.21 *Given a family of circuits $B = \{B_n\}$, and functions δ_n, l_n, where $\delta = o(1)$, and $l\delta = o(1)$, such that:*

$$Pr_{x,y \in_R \{0,1\}^n}(B_{2n}(x,y) = GL((g_n(x), y))) \geq 1 - \delta_n$$

we can construct a family of circuits $A = \{A_n\}$ such that:

$$Pr_{x \in_R \{0,1\}^n}(A_n(x) = g_n(x)) \geq (1 - \frac{1}{l})(1 - \frac{1}{n^c})$$

for any constant c, and $size(A) = (size(B)n)^{O(1)}$.

Proof: define:

$$X = \{x | Pr_{y \in_R \{0,1\}^n}(B(x,y) = GL(g(x), y)) \geq 1 - l\delta\}$$

Since:

$$(1 - \mu(x))(1 - l\delta) + \mu(x) \geq 1 - \delta$$

We get that $\mu(x) \geq 1 - \frac{1}{l}$. call x good, if it is in X. Fix a good x, and set $z = g(x)$. Our aim is to compute the bits of z one by one. Note that for all y:

$$z_i = GL(z,y) \oplus GL(z, y^{(i)})$$

where

$$y_j^{(i)} = \begin{cases} \bar{y_j} & j = i \\ y_j & otherwise \end{cases}$$

So if we could compute $GL(x, \cdot)$ we would be done. Since x is good, B errs only for a δ fraction of the y's. Using the fact that if y is uniformly distributed, so is $y^{(i)}$, we get for all i:

$$Pr_{y \in_R \{0,1\}^n}(\{B(x,y) = GL(z,y)\} \wedge \{B(x,y^{(i)}) = GL(z, y^{(i)})\}) \geq 1 - 2l\delta \qquad (1.1)$$

$1 - 2l\delta$ is far enough from $\frac{1}{2}$, to use amplification. This suggests the following scheme:

for $1 \leq \alpha \leq t$, (where t is a parameter that will be determined later)
pick $y_\alpha \in_R \{0,1\}^n$.
for all α,i:
compute $h_\alpha = B(x, y_\alpha)$, and $h_\alpha^{(i)} = B(x, y_\alpha^{(i)})$.
compute $b_\alpha^i = h_\alpha \oplus h_\alpha^{(i)}$.
let e^i be the majority of the b_α^i.

Claim 1 *For all i, $Pr_{y_1,..,y_t}(e^i \neq z_i) \leq e^{-\Omega(t)}$.*

Proof: Since x is good, and the y_α are uniformly distributed, 1.1 holds. when the event occurs, $b_\alpha^i = z_i$. So this process computes z_i with some advantage over $\frac{1}{2}$. Since the y_α's are independent, taking the majority of the guesses amplifies the success probability. define the random indicator variable:

$$D_\alpha^i = \begin{cases} 1 & b_\alpha^i = z_i \\ 0 & otherwise \end{cases}$$

and set $D^i = \sum_\alpha D_\alpha^i$, using the chernoff inequality we get for all i:

$$Pr_{\{y_\alpha\}}(D^i < \frac{t}{2}) \leq e^{-\Omega(t)}$$

which in our terminology is the requested claim.

•

Using claim 1 we have:

$$Pr_{\{y_\alpha\}}(\exists i : e^i \neq z_i) \leq ne^{-\Omega(t)}$$

Fixing $t = \Theta(\log n)$ we get that the suggested scheme computes all the bits of z correctly, with probability $1 - \frac{1}{poly(n)}$. The scheme can easily be converted into a circuit C of size $(size(B)n)^{O(1)}$. (At this point the circuit is probabilistic, as it randomly picks the $\{y_\alpha\}$). When applying C to a uniformly selected input x. there is a $\mu(X) \geq 1 - \frac{1}{l}$ chance that x is good, when this happens the analysis we made is correct, and so C computes $z = g(x)$ correctly with probability $(1 - \frac{1}{l})(1 - \frac{1}{poly(n)})$. To transform C into a deterministic circuit, we simply note that, Since C achieves the desired success probability, on random x, $y_1, .., y_t$. There is a fixing of $y_1, .., y_t$, such that C achieves the same success probability on random x, and fixed $y_1, .., y_t$. Define $A(x)$ to be the circuit C when $y_1, .., y_t$ are fixed. note that $size(A) \leq size(C)$, and so A is the circuit we wanted to build in the first place.

•

We will now prove the general case.

Lemma 1.22 *Given a family of circuits $B = \{B_n\}$ such that:*

$$Pr_{x,y \in_R \{0,1\}^n}(B_{2n}(x, y) = GL_{2n}((g_n(x), y))) \geq \frac{1}{2} + \epsilon_n$$

we can construct a family of circuits $A = \{A_n\}$ such that:

$$Pr_{x \in_R \{0,1\}^n}(A_n(x) = g_n(x)) \geq (\frac{\epsilon}{n})^{O(1)}$$

and $size(A) = (\frac{size(B)n}{\epsilon})^{O(1)}$.

Proof: We will attempt to imitate the proof of lemma 1.21. define:

$$X = \{x | Pr_{y \in_R \{0,1\}^n}(B(x,y) = GL(g(x),y)) \geq \frac{1}{2} + \frac{\epsilon}{2}\}$$

It is obvious that $\mu(X) \geq \frac{\epsilon}{2}$. Again, call x good if $x \in X$. Fix a good x, and set $z = g(x)$. Once again, our strategy will be to compute the bits of z one by one.
However, we can't compute $GL(z,y)$ and $GL(z,y^{(i)})$, simultaneously, with probability greater than $\frac{1}{2}$ as we did in the previous case. Suppose we could construct a series of t vectors $\{y_\alpha\}$ such that:

1. the y_α's are uniformly distributed and pairwise independent.
2. we can compute $h_\alpha = GL(z, y_\alpha)$ for all α.

In this case we could use exactly the same strategy as in lemma 1.21, to continue. There is only one exception, the y_α's are not independent, only pairwise independent. However, we can still amplify the success probability using this assumption.

Claim 1 *for all i,* $Pr_{\{y_\alpha\}}(e^i \neq z_i) \leq \frac{4}{\epsilon^2 t}$

Proof: The proof is similar to the one of the claim in lemma 1.21. The only difference is that we cannot use the chernoff inequality, because the y_α's are only pairwise independent, and so we use the chebycheff inequality. Again we define the random indicator variable:

$$D^i_\alpha = \begin{cases} 1 & b^i_\alpha = z_i \\ 0 & otherwise \end{cases}$$

and set $D^i = \sum_\alpha D^i_\alpha$, using the chebychef inequality we get for all i:

$$Pr_{y_\alpha}(|D^i - E(D^i)| \geq \frac{t\epsilon}{2}) \leq \frac{4}{\epsilon^2 t}$$

which in our terminology is the requested claim.

•

Using claim 1 we have:

$$Pr_{\{y_\alpha\}}(\exists i : e^i \neq z_i) \leq \frac{4n}{\epsilon^2 t} \tag{1.2}$$

And thus, we can compute $z = g(x)$, with good probability. The problem is we don't know how to fulfill condition 2 above. However, we can achieve it with non-negligible probability.

2'. with probability $2^{-\log t}$, all the h_α's can be computed simultaneously.

Set $k = \log t$, and consider the following scheme:
Pick a random (uniformly chosen) matrix of size $n \times k$ with entries in $\{0,1\}$.
compute $y_\alpha = M \cdot \alpha$ for all $\alpha \in \{0,1\}^k$
pick a random $\beta \in_R \{0,1\}^k$

It is standard to check that the y_α's satisfy condition 1 above. note that for all α:

$$h_\alpha = GL(z, y_\alpha) = z^t(M\alpha) = (z^tM)^t\alpha$$

(where Matrix multiplication, is over GF(2)). since M is a random matrix, zM is uniformly distributed in B^k, which implies:

$$Pr_{M,\beta}(\beta = zM) = 2^{-k} = 2^{-\log t}$$

When $\beta = zM$, we can compute $h_\alpha = \beta^t\alpha$ for all $\alpha \in \{0,1\}^k$, (and Thus fulfill condition $2'$). Combining this with claim 1.2, we can build a probabilistic circuit C that computes all the z_i's correctly with probability $2^{-\log t}(1 - \frac{4n}{\epsilon^2 t})$. The circuit is of size $(size(B)n\log t)^{O(1)}$. choosing $t = \log\frac{n^2}{\epsilon^2}$, we get that C computes $z = g(x)$ with probability $(\frac{\epsilon}{n})^{O(1)}$, and $size(C) = (\frac{size(B)n}{\epsilon})^{O(1)}$. When activating C on uniformly chosen x there is a $\frac{\epsilon}{2}$ chance that x is good. When this happens the analysis we made is correct, and so:

$$Pr_{x,M,\beta}(C(x, M, \beta) = g(x)) \geq \frac{\epsilon}{2} \cdot (\frac{\epsilon}{n})^{O(1)} = (\frac{\epsilon}{n})^{O(1)}$$

This is exactly what we need except for the fact that C is probabilistic (it selects M, β randomly). However, since C computes $g(x)$ with some probability of success, when x and it's "random coins" are picked randomly, There has to be some way to fix the "random coins" which achieves the same probability. formally:

$$Pr_{x,M,\beta}(C(x, M, \beta) = g(x)) = E_{M,\beta}(Pr_x(C(x, M, \beta) = g(x)))$$

$$\leq max_{M,\beta}(Pr_x(C(x, M, \beta) = g(x)))$$

fixing $A(x) = C(x, M, \beta)$ for the M, β which achieve the maximum, we have that A is the circuit we wanted to build in the first place.

$\bullet$

1.2.4 Prediction tests Vs. Statistical tests

Recall that a string is "Pseudo-Random" if it passes some list of prescribed tests. (in our case if every circuit of some bounded size cannot distinguish between the pseudo-random string and a truly random one). We'll call these tests statistical tests:

Definition 1.23 *a distribution D on $\{0,1\}^n$, is said to ϵ-pass all statistical tests of size s, if for every circuit of size s:*

$$|Pr_{x\in_D\{0,1\}^n}(C(x) = 1) - Pr_{x\in_R\{0,1\}^n}(C(x) = 1)| \leq \epsilon$$

Using this notation, a $(\cdot, \cdot, s, \epsilon)$-generator is one that ϵ-passes all statistical tests of size s. At the end of section 1.2.1 we noticed that random strings pass another kind of tests, we will call: "prediction tests".

Definition 1.24 *a distribution D on $\{0,1\}^n$ is said to ϵ-pass all prediction tests of size s, if for every $1 \leq i \leq n$ and every circuit of size s on $i-1$ inputs:*

$$|Pr_{x \in_D \{0,1\}^n}(C(x_1,..,x_{i-1}) = x_i)| \leq \frac{1}{2} + \epsilon$$

Notice that if a distribution D, ϵ-passes all statistical tests of some size, then it ϵ-passes all prediction tests of slightly smaller size. This is true, since given a circuit C that predicts the i'th bit. We can construct a circuit C' which gets n bits, activates C on the first $i-1$ bits and compares the result to the given i'th bit. On random strings C' outputs 1, with probability $\frac{1}{2}$, where on strings selected from D it outputs 1, with probability $\frac{1}{2}+\epsilon$
It is quite surprising (and very useful) that the opposite direction is also true:

Theorem 1.25 *If a distribution D on $\{0,1\}^n$, $\frac{\epsilon}{n}$-passes all prediction tests of size s, then it ϵ-passes all statistical tests of size $s' = s - O(n)$.*

Proof: Let C be a circuit of size s' such that:

$$|Pr_{x \in_D \{0,1\}^n}(C(x) = 1) - Pr_{x \in_R \{0,1\}^n}(C(x) = 1)| \geq \epsilon$$

We can view the string chosen according to D as n bit random variables $b_1,..,b_n$. We will view the uniform distribution on n bits the same way, denoted $r_1,..,r_n$. define:

- $D_i = (b_1, b_2, .., b_i, r_{i+1}, r_{i+2}, .., r_n)$
- $P_i = Pr_{x \in D_i}(C(x) = 1)$

Notice that:

- $D_0 = U_n$ (The uniform distribution on n bits).
- $D_n = D$
- $P_0 = Pr_{x \in_R \{0,1\}^n}(C(x) = 1)$
- $P_n = Pr_{x \in_D \{0,1\}^n}(C(x) = 1)$

It follows that:

$$\epsilon \leq |P_n - P_0| = \sum_{1 \leq i \leq n} (P_i - P_{i-1}) \leq \sum_{1 \leq i \leq n} |P_i - P_{i-1}|$$

So there exists i such that: $|P_i - P_{i-1}| \geq \frac{\epsilon}{n}$. w.l.o.g. assume that:

$$P_i - P_{i-1} \geq \frac{\epsilon}{n} \tag{1.1}$$

Define: $\bar{D}_j = (b_1, b_2, .., b_{j-1}, \bar{b_j}.r_{j+1}, .., r_n)$, and $\bar{P}_j = Pr_{x \in \bar{D}_j}(C(x) = 1)$. Note that $D_{i-1} = \frac{D_i + \bar{D}_i}{2}$, (meaning that D_{i-1} is the same as flipping a coin and sampling from D_i and $\bar{D}_i$ depending on the outcome of the coin). This implies:

$$P_{i-1} = \frac{P_i + \bar{P}_i}{2} \tag{1.2}$$

Let's separate $\{0,1\}^n$ to three parts: $\{0,1\}^n = \{0,1\}^{i-1} \times \{0,1\} \times \{0,1\}^{n-i}$ denoted u, y, w respectively.
We will construct a circuit C' on $i-1$ bits (denoted u):
pick $d \in_R \{0,1\}$.
pick $w \in_R \{0,1\}^{n-i}$
"activate" $C(u, d, w)$.
if it outputted "1" output d, otherwise output $\bar{d}$.

Claim 1 *$Pr_{x \in_D \{0,1\}^n, d \in_R \{0,1\}, w \in_R \{0,1\}^{n-i}}(C'(x_1, .., x_{i-1}) = x_i) \geq \frac{1}{2} + \frac{\epsilon}{n}$ meaning that there is a small circuit that predicts the i'th bit given the previous bits.*

Proof: We want to calculate the probability that C' computes b_i correctly. The probability is over the choice of $u = (b_1, .., b_{i-1})$, and uniformly chosen d,w. (We think of d as r_i, and w as $(r_{i+1}, .., r_n)$).

$$\begin{aligned} Pr(C'(u) = b_i) = {} & Pr(b_i = d | C(u,d,w) = 1) Pr(C(u,d,w) = 1) \\ & + Pr(b_i = \bar{d} | C(u,d,w) = 0) Pr(C(u,d,w) = 0) \end{aligned}$$

which can be written as:

$$Pr(b_i = d | C(u,d,w) = 1) P_{i-1} + Pr(b_i = \bar{d} | C(u,d,w) = 0)(1 - P_{i-1}) \tag{1.3}$$

We will bound this expression by bounding both summands.

$$\begin{aligned} Pr(b_i = d | C(u,d,w) = 1) &= \frac{Pr(C(u,d,w) = 1 | b_i = d) Pr(b_i = d)}{Pr(C(u,d,w) = 1)} \\ &= \frac{P_i}{2P_{i-1}} \end{aligned} \tag{1.4}$$

$$Pr(b_i = \bar{d} | C(u,d,w) = 0) = \frac{Pr(C(u,d,w) = 0) | b_i = \bar{d}) Pr(b_i = \bar{d})}{Pr(C(u,d,w) = 0)} \tag{1.5}$$

$$= \frac{1 - \bar{P}_i}{2(1 - P_{i-1})}$$

Combining 1.3,1.4 and 1.5, we get:

$$Pr(C'(u) = b_i) = \frac{1}{2} + \frac{P_i - \bar{P}_i}{2} = \frac{1}{2} + \frac{\epsilon}{n}$$

Where the last equality is using 1.1 and 1.2. This completes the proof of the claim.

•

What is left is to transform C' to a deterministic circuit. This can be done by fixing d, w, using:

$$max_{d,w}(Pr_{x \in_D \{0,1\}^n}(C'_{d,w}(x_1,..,x_{i-1}) = x_i))$$

$$\geq E_{d,w}(Pr_{x \in_D \{0,1\}^n}(C'_{d,w}(x_1,..,x_{i-1}) = x_i))$$

$$= Pr_{d,w,x \in_D \{0,1\}^n}(C'_{d,w}(x_1,..,x_{i-1}) = x_i)$$

defining $C''(x) = C'_{d,w}(x)$ we get a circuit which predicts the i'th bit with $\frac{\epsilon}{n}$ advantage. Note that: $size(C'') = size(C) + O(n)$. ($w$ may be of length n), so $s = s' + O(n)$ as needed.

•

Using theorem 1.25 it suffices to show that a generator passes all prediction tests, in order to show that it passes all statistical tests.

1.2.5 The Generator's construction

We are now in a position to refine the attempts we made at subsection 1.2.1, and handle the difficulties we pointed out there. Recall, that we want to construct a δ-fast Generator, (for any given δ), using the sole assumption that one way permutations exist. In subsection 1.2.2 we proved that if one way permutations exist, then there exists one way permutations with hard bits (theorem 1.20). So we assume w.l.o.g. That f^{-1} has a hard bit, which we will denote $b = \{b_n\}$. the Generator is defined in the following way:

Definition 1.26 *(The Blum-Micali-Yao Generator) For $0 < \delta < 1$ define the generator $G^\delta = \{G^\delta_n\}$ which takes as input n^δ bits denoted y and output n bits. set:*

- $y_0 = y$.
- $y_i = f_{n^\delta}(y_{i-1})$ *for* $1 \leq i < n$.
- $b_i = b_{n^\delta}(y_i)$.
- $G^\delta_n(y) = (b_{n-1}, b_{n-2}, .., b_0)$.

recall (definition 1.12) that a δ-fast Generator, is one that uses a polynomial procedure to extend n^δ bits into n bits, and induces a distribution that ϵ-passes all statistical tests, where $\lim_{n\to\infty} \epsilon_n n^e = 0$ for all e.

Theorem 1.27 *For all $0 < \delta < 1$, G^δ is a δ-fast Generator.*

Proof: Computing G^δ_n is indeed polynomial in n, since f and b can be computed in polynomial time. For future use, we remark on the connection between δ and the running time of G^δ_n

Remark 1.28 *There exists c, such that for all δ, G^δ_n runs in time n^c.*

We now proceed with the proof of the theorem.

Claim 1 *For all e, G_n^δ $\frac{1}{n^e}$-passes all prediction tests of polynomial size.*

Proof: Suppose that there exists a polynomial family of circuits $C = \{C_n\}$, that "catches" the generator. That is, there exist e such that for infinitely many n there exist i with:

$$Pr_{y \in_R \{0,1\}^{n^\delta}}(C(G_n^\delta(y)|_{1..i-1}) = (G_n^\delta(y)|_i) \geq \frac{1}{n^e}$$

We will construct a polynomial circuit C' that computes the hard bit of $f_{n^\delta}^{-1}$ with non-negligible advantage:

On input $z \in \{0,1\}^{n^\delta}$:
Set $z_1 = z$, compute $z_{j+1} = f_{n^\delta}(z_j)$ (for $1 \leq j \leq i-2$).
Compute $h_j = b_{n^\delta}(z_j)$ (for $1 \leq j \leq i-1$).
output $C(h_{i-1}, .., h_1)$

Note that C' is of polynomial size. We will prove that:

$$Pr_{z \in_R \{0,1\}^{n^\delta}}(C'(z) = b_{n^\delta}(f_{n^\delta}^{-1}(z))) \geq \frac{1}{n^e} \tag{1.6}$$

The Random Variable $(h_{i-1}, .., h_1)$ has the same distribution as $G_n^\delta(y)|_{1..i-1}$. To see this, note that f_{n^δ} is a permutation. This means that if it's input is uniformly distributed, so is it's output. So, $y_{n-(i-1)}$ is uniformly distributed, and has the same distribution of z. From this point both the generator and the circuit use the same procedure to compute the i bits. This means that C' activates C with the same distribution of the generator. On such, C is promised to compute the i'th bit with non-negligible advantage. The i'th bit is exactly $b_{n^\delta}(f_{n^\delta}^{-1}(z))$. This proves 1.6. Note that $size(C')$ is polynomial in n, and since b is a hard bit of f^{-1}, this is a contradiction.

•

Using theorem 1.25 we deduce that G_n^δ $\frac{1}{n^{e-1}}$-passes all statistical tests of polynomial size, for all e. This completes all the conditions needed to certify G_n^δ as a δ-fast generator. •

Theorem 1.29 *(main theorem, Blum-Micali-Yao)*
If one way permutations exist then $BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$

Proof: For a fixed $\delta > 0$ we proved (theorem 1.27) that the existence of a one way permutation implies that $G^\delta = \{G_n^\delta\}$ is a δ-fast generator. Remark 1.28 says that there is some c (which does not depend on δ) such that G_n^δ runs in time n^c. we can use corollary 1.11 and definition 1.12 to get:

$$bptime_{\frac{1}{3}}(n^t, n^t) \subseteq dtime(2^{n^{t\delta}}(n^t + n^{ct})) \subseteq dtime(2^{n^{2t\delta}})$$

Taking $\delta(\rho, t) = \frac{\rho}{2t}$ we get:

$$bptime_{\frac{1}{3}}(n^t, n^t) \subseteq dtime(2^{n^\rho})$$

Since this is true for all t, ρ, (and switching between ρ and δ) we get:

$$BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$$

•

1.3 The Nisan-Wigderson Generator

1.3.1 One way functions Vs. Unapproximable functions

In the previous section, we were able to derandomize BPP using the assumption that one way permutations exist. We would like to get the same conclusion using a weaker hardness assumption. Apart from pure mathematical satisfaction, this is motivated by the observation that there are currently only few functions which are conjectured to be one way. Apart from that, in our efforts we try to measure the "distance" between deterministic and probabilistic computation. The following theorem shows that the existence of one way permutations already implies a limitation on the power of BPP.

Theorem 1.30 *If one way permutations exist, then:*

$$(NP \cap co-NP) \setminus BPP \neq \emptyset$$

"Randomness is not powerful enough to simulate $NP \cap co_N P$".

Proof: Let f be a one way permutation, using theorem 1.20 we can assume w.l.o.g. that f^{-1} has a hard bit which we will denote $b = \{b_n\}$. consider the languages:

$$L_i = \{x|\ b(f^{-1}(x)) = i\}, \text{ for } i \in \{0,1\}$$

Note that for $i \in \{0,1\}$, $L_i \in NP$, because given x one can guess $y = f^{-1}(x)$ which is of length $|x|$, and check in polynomial time that $f(y) = x$, and $b(y) = i$, and since f is a permutation, if $x \notin L_i$ there is no y that satisfies both conditions. Both L_i are also in $co-NP$, because they complement each other. On the other hand, none of them is in BPP. Suppose $L_0 \in BPP$, then there exists a probabilistic algorithm A, which accepts L_0 with two sided error of $\frac{1}{3}$. Using Adelman's theorem ($BPP \subseteq P/poly$, which we will prove in 2.2) we get a circuit that checks membership in L_0, such a circuit computes $(b(f^{-1}))(\cdot)$ on every input, so it has success probability 1.
Alternatively, we could take A, transform it into a probabilistic polynomial circuit (with the same success probability), and then fix the random input to a choice which achieves this probability.

•

As far as we know, BPP may be as powerful as EXP, or $\Sigma_P^2 \cap \Pi_P^2$, so we'd rather use an assumption that allows these situations.
Let's examine the way we used our hardness assumption: We need the function to be "easy" for uniform deterministic computation, because the generator has to compute the function. On the other hand, it (or it's inverse, in the case of one way functions) has to be hard for polynomial circuits. Observe that when we use corollary 1.11, to derandomaize BPP using a δ-fast generator, the significant factor we pay in the running time is 2^{n^δ}, which is added because of the enumeration of all possible seeds of the generator. The contribution of the running time of the generator is a much smaller factor, as it is not exponentiated. So we could still get the result if the generator $G_n^\delta : \{0,1\}^{n^\delta} \to \{0,1\}^n$ will run in time $2^{n^{O(\delta)}}$. Since the generator runs in exponential time, we may as well have the "easiness" condition be: the function can be computed in exponential time.

Definition 1.31 *A family of generators $G = \{G^\delta\}$ for $0 < \delta < 1$ is called a quick family of generators if there exist a constant c such that for all δ, G^δ is a $(n^\delta, 2^{n^{c\delta}}, n^d, \frac{1}{n^e})$-generator for all d, e.*

Compare this with the definition of δ-fast generators (definition 1.12). The first difference is methodological, since the running time of the generator is going to play a role, and is dependent on δ, we need to speak about a family of generators. The only other difference is that we allow the generator to run in exponential time. With this in mind we define the following class of hard functions:

Definition 1.32 *A function $f = \{f_n\}$, $f_n : \{0,1\}^n \to \{0,1\}^n$ is called "unapproximable by polynomial circuits" if for every family of polynomial circuits $C = \{C_n\}$:*

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = f_n(C_n(x))) \leq \epsilon_n$$

where $\lim_{n\to\infty} \epsilon_n \cdot n^c = 0$ for all c. We will call such functions "unapproximable", ommiting the size of the circuits.

Remark 1.33 *We also need to assume some "easiness" condition on $f = \{f_n\}$. We must have it, because the generator needs to compute $f = \{f_n\}$. Since the generator runs in exponential time, we may as well demand that $f = \{f_n\}$, is computable in exponential time. This is a much weaker condition then one way functions. Recall that the "easy" side of the function, was computable in polynomial time. Here both conditions apply to f, (rather than one to f and one to f^{-1}, in the case of one way functions). This changes nothing, and is more comfortable.*

Compare definition 1.32 of one way functions (definition 1.13), and note that using this notation, a one way permutation is a polynomial permutation whose inverse is unapproximable. Could we use the same construction, changing only the "easiness" condition of the one way permutation? The answer is negative. When we proved that the BMY-generator is fast (theorem 1.27), we used the fact that the easy side can be computed by a polynomial circuit. The circuit used it to produce $i - 1$ generator-like bits, from a single input. Note that this difficulty seems inescapable. When constructing a circuit that computes the hard function, from a circuit that breaks the generator, we have to somehow compute many bits of "Generator's output" to feed into the first circuit. It seems that this requires the circuit to have as much computational power as the generator. The clever idea behind the generator is that the circuit compensates for it's weakness by using it's non-uniformity.

1.3.2 Nearly disjoint sets

Definition 1.34 *A collection of sets $\{S_i\}_{1 \leq i \leq n}$, where $S_i \subseteq [l]$ is called a (k, m)-design if:*

- *For all i, $|S_i| = m$*
- *For all $i \neq j$, $|S_i \cap S_j| \leq k$.*

Our aim is to construct a design minimizing l, k while maximizing m. (l, k, m will be functions of n).

Theorem 1.35 *For a prime number m, such that $\log n \leq m \leq n$, there exists a $(\log n, m)$-design with $l = m^2$.*

Proof: Since m is prime, $GF(m)$ is a field over $[m]$. We will take $l = m^2$, and identify $[l]$ with $[m] \times [m]$. Given a polynomial q of degree $d \leq \log n$, over $GF(m)$ we define:

$$S_q = \{(t, q(t)) \mid 1 \leq t \leq m\}$$

Note that:

- For all q, $|S_q| = m$.
- There are $m^{\log n+1} \geq n$ polynomials q of degree $\log n$ over $GF(m)$.
- For polynomials $q \neq q'$, $|S_q \cap S_{q'}| \leq \log n$, (because two polynomials of degree d agree on at most d points).

•

For our purposes, we need the design to be computable by a polynomial algorithm. Given a prime number m, it is easy to compute the arithmetic of $GF(m)$, and so, the design is constructible in time polynomial in n. There are constructions of deterministic polynomial algorithms, that given a number m construct a prime number m' such that $m' = \Theta(m)$. Using this, we deduce:

Corollary 1.36 *There exists a deterministic polynomial algorithm A, which on input (m, n) (where $m \leq n$), outputs sets $\{S_i\}_{1 \leq i \leq n}$, which are a $(\log n, m)$-design, with $l = O(m^2)$.*

1.3.3 Constructing the generator

We will construct the generator using the assumption that there exist a function f that cannot be approximated by polynomial circuits (definition 1.32). At this point we will use the Goldreivh-Levin theorem (1.20) to get a boolean function that cannot be approximated with non-negligible advantage by polynomial circuits.

Lemma 1.37 *(A different formulation of theorem 1.20)*
For every function $f = \{f_n\}$, $f_n : \{0,1\}^n \to \{0,1\}^n$, there exists a function $b = \{b_n\}$, $b_n : \{0,1\}^n \to \{0,1\}$ such that:

1. *Given a family of circuits $C = \{C_n\}$ such that:*

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = b_n(x)) \geq \frac{1}{2} + \epsilon$$

there exists a family of circuits $C' = \{C'_n\}$ such that:

$$Pr_{x \in_R \{0,1\}^n}(C'_n(x) = f_n(x)) \geq (\frac{\epsilon}{n})^{O(1)}$$

and $size(C')$ is polynomial in $size(C), n, \frac{1}{\epsilon}$.

2. *Given a deterministic algorithm A that runs in time t and computes f, there exists an algorithm B that runs in time $t + n^{O(1)}$ and computes b.*

Proof: (The proof uses definitions from section 1.2.2)
Given $f = \{f_n\}$ define $b = \{b_n\}$, in the following way:

$$b_n(x) = GL(\hat{f}(x|_{1..\frac{n}{2}}, x|_{\frac{n}{2}+1,..,n})$$

b_n is based on $f_{\frac{n}{2}}$, but this doesn't matter since the conclusions are not affected by constants. The first conclusion is a straight from theorem 1.20, and the observation that a circuit that computes $\hat{f}$ with some success probability, computes f with the same success probability. The second conclusion is also straightforward using the fact that GL is computable in polynomial time.

•

Using lemma 1.37, we now have a function $b = \{b_n\}$, such that for every family of polynomial circuits $C = \{C_n\}$

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = b_n(x)) \leq \epsilon_n$$

For some function ϵ_n such that $\lim_{n\to\infty} \epsilon_n \cdot n^c = 0$, for all $c > 0$. We now define a quick family of generators.

Definition 1.38 *(The Nisan-Wigderson generator) For fixed δ we define the generator $G^\delta = \{G_n^\delta\}$, $G_n^\delta : \{0,1\}^{n^\delta} \to \{0,1\}^n$. The generator first computes a $(\log n, n^{\frac{\delta}{2}})$-design, $\{S_i\}_{1\leq i\leq n}$ where for all i, $S_i \subseteq [l]$ and $l = n^\delta$ (this can be done in time poly(n) using corollary 1.36). Given an input $x \in \{0,1\}^{n^\delta}$, Define:*

- $b_i = b_{n^\delta}(x|_{S_i})$.
- $G^\delta(x) = (b_1, .., b_n)$.

Theorem 1.39 *the family $\{G^\delta\}$ is a quick family of generators.*

Proof: We start by estimating the running time of G^δ.

Claim 1 *G^δ runs in time $2^{n^{O(\delta)}}$.*

Proof: We need to prove that there is a constant c, such that for all δ, G^δ runs in time $2^{n^{c\delta}}$. G^δ uses a polynomial (in n), procedure to construct the design. From there it only computes b_{n^δ}, n times. f is computable in exponential time, and using lemma 1.37, so is b. we conclude that G^δ runs in time $poly(n) + n \cdot 2^{(n^\delta)^d}$ for some constant d, (which depends on f). This is indeed $2^{n^{O(\delta)}}$.

•

To complete the proof we need to prove that G^δ ϵ_n-passes all statistical tests of polynomial size. For some function ϵ_n, such that $\lim_{n\to infty} \epsilon_n \cdot n^c = 0$ for all $c > 0$. Using theorem 1.25, it is enough to prove that there exist such a function ϵ_n such that G^δ ϵ_n-passes all prediction tests of polynomial size.

Claim 2 *There exist a function ϵ_n, such that* $\lim_{n\to\infty} \epsilon n^c = 0$ *for all c, and G^δ ϵ_n-passes all prediction tests of polynomial size.*

Proof: If the claim does not hold, then there exist a constant e, and a family of circuits $C = \{C_n\}$ which are polynomial in n, such that for infinitely many n, there exist i such that:

$$Pr_{x\in_R\{0,1\}^{n^\delta}}(C_n(G_n^\delta(x)|_{1..i-1}) = G_n^\delta(x)|_i) \geq \frac{1}{n^e} \tag{1.7}$$

We will construct a family of polynomial circuits $C' = \{C'_n\}$, such that for each n where 1.7 holds, C'_n will compute $b_{n^{\frac{\delta}{2}}}$ with non negligible advantage. It will be convenient for us to break $\{0,1\}^n$ to two parts (y, z), where y are the bits which appear in S_i and z are the rest of the bits. Since 1.7 is true for random $x = (y, z)$ there exists some z', such that the inequality holds for fixed z' and random y. Fixing $z = z'$, we view all b_j's as a function only of y. With this viewpoint for all $j \neq i$, $b_j = b_{n^{\frac{\delta}{2}}}((y, z')|_{S_j})$ is a function over $\log n$ bits. This is because for $j \neq i$, $|S_j \cap S_i| \leq \log n$. Any function over $\log n$ bits can be computed by a circuit of size n, by simply using it's CNF or DNF representation. On input $y \in \{0,1\}^{n^{\frac{\delta}{2}}}$, the circuit C' will compute $b_1, .., b_{i-1}$ and activate $C_n(b_1, .., b_{i-1})$. Note that $i < n$, and so $C'_{n^{\frac{\delta}{2}}}$ (which computes i bits, where each requires size n is of size at most $n^2 + size(C_n)$. So it is polynomial in $n^{\frac{\delta}{2}}$. On a uniformly chosen $y \in_R \{0,1\}^{n^{\frac{\delta}{2}}}$,

$$C'(y) = C(G_n^\delta(y, z')|_{1..i-1})$$

Recall that $G_n^\delta(y, z')|_i = b_{n^{\frac{\delta}{2}}}(y)$, and so, C' computes b with non-negligible advantage, which is a contradiction.

•

This completes the proof of the theorem.

•

Remark 1.40 *Note that the non-uniformity of the circuit played an important role in the proof. It enabled C' to "know" z' in advance. That is to have an accessible hint of polynomial size.*

Theorem 1.41 *(main theorem, Nisan-Wigderson)*
If there exist functions that are computable in exponential time and cannot be approximated by polynomial circuits then $BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$.

Proof: We have just proved that the existence of functions that cannot be approximated by polynomial circuits implies the existence of a quick family of generators. Using corollary 1.2 we get for all δ.

$$bptime_{\frac{1}{3}}(n^t, n^t) \subseteq dtime(2^{n^{t\delta}}(2^{n^{O(t\delta)}} + n^t)) \subseteq dtime(2^{n^{ct\delta}})$$

for some constant c. By choosing $\delta(\rho, t) = \frac{\rho}{ct}$, we get:

$$bptime_{\frac{1}{3}}(n^t, n^t) \subseteq dtime(2^{n^\rho})$$

Since this is true for all t, ρ, (and switching between ρ and δ), we get:

$$BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$$

•

1.3.4 A generator for constant depth circuits

In this section we deviate from the main subject of derandomizing BPP to show another use of the generator. It should be mentioned that historically, the generator was constructed for this cause, and only later used to derandomize BPP. An important observation is that in the proof of theorem 1.39 We used only the following properties:

1. b cannot be approximated by the model of computation we want to fool.

2. The model of computation we want to fool can compute functions over $\log n$ bits.

These are very general properties, and in this section we will use the construction to build a generator that fools constant depth circuits.

Definition 1.42 *An unbounded fan-in circuit, is a directed acyclic graph, where each node is labeled by a gate type (or,and,not). Vertices of in-degree zero, are called inputs, and those with out-degree zero are called outputs. The inputs are labeled by input bits, or their negation. The circuit "computes" by successively applying the functions at each gate. The depth of a circuit is the length of a maximal path from input to output. The size of a circuit is the number of edges in the graph.*

It makes no sense to place a restriction solely on circuit's depth, since depth 2 circuits, can compute all functions, by using their DNF representation. However, it becomes interesting adding a size restriction.

Definition 1.43 *The class AC_0 is the class of unbounded fan-in circuits, of constant depth and size polynomial in the input length.*

It seems that AC_0 circuits are weaker than regular polynomial circuits. An evidence to this is that we do know to prove that there are functions that cannot be approximated by AC_0 circuits. One such example is the parity function.

Theorem 1.44 *(Hastad)*
For any family $C = \{C_n\}$ of unbounded fan-in circuits of depth d, and size $2^{n^{\frac{1}{d-1}}}$

$$|Pr_{x \in_R \{0,1\}^n}(C_n(x) = parity_n(x)) - \frac{1}{2}| \leq 2^{-n^{\frac{1}{d-1}}}$$

So we have fulfilled the first condition above. For the second one, note that any function over $\log n$ bits can be computed in depth 2, and size n, using it's DNF representation. We could use $parity_n$ as a hard boolean function, and use exactly the same arguments as in 1.39 and 1.25, to conclude that the Nisan-Wigderson generator fools AC_0 circuits. It should be remarked that in this case, the parameters can be chosen in a better way than in the previous sections. We don't go into details, as it is outside the scope of this chapter.

Chapter 2

Using a worst case complexity assumption

2.1 Worst case complexity Vs. Distributional complexity

In the previous chapter we were able to prove that $BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$, Using an assumption that there exist a function that can be computed in exponential time, and cannot be approximated by polynomial circuits. Note the difference between "to compute", and "to approximate". This is a difference in the way we consider bounded resources computation. The "Worst case complexity" model, requires that the algorithm performs the task correctly on every given input. The "Distributional complexity" model requires only that the algorithm succeeds with good probability on a random input, Complexity theory, usually chooses to speak about "worst case complexity". Indeed, all the elementary definitions made in the first section, speak about "worst case complexity", in particular, our goal (to simulate BPP by deterministic computation) is stated in "worst case complexity" terms. It would be nice if we could prove the same conclusion, using an assumption that is on "worst case complexity" computation. It is surprising that to perform this, we will not need a new generator. The Nisan-Wigderson generator (described in the previous chapter) will suffice. What we will do, is prove that:

The existence of functions that cannot be computed by polynomial circuits

implies

The existence of functions that are "somewhat hard" to approximate

implies

The existence of functions that are "very hard" to approximate

This process will use the fact that the first function can be computed in exponential time. This property will be preserved, and thus the final function will be computable in exponential time, and unaproximable by polynomial circuits. All that is left is to use the Nisan-Wigderson generator (theorem 1.41) to conclude that:

$$Exp \setminus P/poly \neq \emptyset \;\Rightarrow\; BPP \subseteq \cap_{\delta>0} dtime(2^{n^\delta})$$

2.2 Randomness Vs. Non-Uniformity

In this section we will show that Non-Uniform computation (circuits), need not throw coins, in both "worst case complexity" and "distributional complexity" models. This is the reason why in the previous chapter we did not need to speak about probabilistic circuits.

2.2.1 Distributional Complexity

The result we prove now is very easy, and we used it implicitly in the previous chapter. We state it now separately, since we are now studying the property-s of distributional complexity in general.

Lemma 2.1 *Whenever there exist a family of (probabilistic) circuits $C = \{C_n\}$, which receive (apart from x), a random input r, chosen from an arbitrary distribution $D = \{D_n\}$ on range $R = \{R_n\}$, and*

$$Pr_{x \in_R \{0,1\}^n, r \in_{D_n} R_n}(C_n(x,r) = f(x)) \geq \rho$$

There exist a family of (deterministic) circuits $C' = \{C'_n\}$, such that:

$$Pr_{x \in_R \{0,1\}^n}(C'_n(x) = f(x)) \geq \rho$$

and $size(C') \leq size(C)$.

Proof:

$$\rho \leq Pr_{x \in_R \{0,1\}^n, r \in_{D_n} R_n}(C_n(x,r) = f(x)) = E_{r \in_{D_n} R_n}(Pr_{x \in_R \{0,1\}^n}(C_n(x,r) = f(x)))$$

$$\leq max_{r \in R_n}(Pr_{x \in_R \{0,1\}^n}(C_n(x,r) = f(x)))$$

Let r' be the maximal r, fixing $r = r'$, we get a circuit $C'_n(x) = C_n(x, r')$, which is of the same size as C, and achieves the same success probability.

•

We use the non-uniformity of the circuit, to have it "prepare in advance" a different string r for every length of input n. To prove such a result for uniform computation, we need an efficient way to compute r_n for ever n, the proof gives no clue, as to how to achieve this.

2.2.2 Worst case complexity

by "probabilistic worst case complexity computation" we simply mean that for every input x the (probabilistic) procedure answers correctly with probability that is distinguishably greater than $\frac{1}{2}$. The probability is not taken over the input x, it is taken on the random "coin flipping" of the procedure. We can always assume that all those are done in advance, and the procedure actually receives two inputs (the real input x, and a random input r). This is exactly the way we defined probabilistic algorithms, and *bptime* (definition 1.2). Again, we prove that coin-flipping is not necessary for circuits. This time the circuit will need to pay a little in size to achieve the task.

Theorem 2.2 *(Adelman) Whenever there exist a family of (probabilistic) circuits* $C = \{C_n\}$*, which receive (apart from* x*), a random input* r*, chosen from an arbitrary distribution* $D = \{D_n\}$ *on range* $R = \{R_n\}$*, and for all* x *of length* n*:*

$$Pr_{r \in_{D_n} R_n}(C_n(x,r) = f(x)) \geq \frac{1}{2} + \rho$$

There exist a family of (deterministic) circuits $C' = \{C'_n\}$*, which compute* f*. and* $size(C') = O(\frac{size(C)n}{\rho^2})$*.*

Proof: Our first step is to amplify the success probability of the circuit. This is done (as usual) by taking a lot of independent runs. take t copies of C_n, and have the circuit C''_n compute the majority over the outputs. It is standard to conclude (using the Chernoff inequality) that for all x of length n:

$$Pr_{r_1,..,r_t \in_{D_n} R_n}(C''_n(x; r_1, .., r_t) = f(x)) \geq 1 - 2^{-\Omega(\rho^2 t)}$$

Fixing $t = \Theta(\frac{n}{\rho^2})$, we get that C'' is wrong with probability less than 2^{-2n}. Define for every $x \in \{0,1\}^n$, a random indicator variable Z_x.

$$Z_x = \begin{cases} 1 & C''(x) \neq f(x) \\ 0 & otherwise \end{cases}$$

Set $Z = \sum Z_x$, the number of wrong answers over all x in $\{0,1\}^n$.

$$E(Z) = E(\sum Z_x) = \sum E(Z_x) = 2^n 2^{-2n} < 1$$

Since Z takes integer values and it's expectency is less than 1, there have to be $r'_1, .., r'_t$ such that $Z(r'_1, .., r'_t) = 0$, or in other terms, for all x, $C''(x; r'_1, .., r'_t) = f(x)$. Define the circuit $C'(x) = C''(x; r'_1, .., r'_t)$.

$$size(C') = size(C'') = t \cdot size(C) = O(\frac{n \cdot size(C)}{\rho^2})$$

•

This means that as long as we view the world through "polynomial" glasses. deterministic circuits are as powerful as probabilistic ones. The theorem is often stated (in a weaker form) the following way:

Corollary 2.3 $BPP \subseteq P/poly$

Proof: given a uniform algorithm that runs in time t, one can transform it into a circuit of size t^2. Given a polynomial probabilistic algorithm, simply transform it into a family of polynomial circuits and apply theorem 2.2

•

2.3 From Worst-case hard to slightly unapproximable

In this section we transform a function that is hard in the worst case to one that is somewhat hard to approximate. We will require some machinery.

2.3.1 Transforming a boolean function into a polynomial

One of the tools we need to develop, is a construction that transforms every boolean function into a polynomial of low degree. Let $f : \{0,1\}^n \to \{0,1\}$, be some function, and F a field. Clearly, F contains $\{0,1\}$. For every $a \in \{0,1\}^n$, define a polynomial $p_a \in F[X_1,..,X_n]$:

$$p_a(x_1,..,x_n) = \prod_{1\leq i\leq n}(1 - a_i - x_i)$$

p_a satisfies:

- For all $a \in \{0,1\}^n$, $deg(p_a) \leq n$
- For all $a, b \in \{0,1\}^n$, such that $b \neq a$, $p_a(b_1,..,b_n) = 0$
- For all $a \in \{0,1\}^n$, $p_a(a_1,..,a_n) = 1$.

Definition 2.4 *For a function $f : \{0,1\}^n \to \{0,1\}$, and a field F, we define a function $\bar{f} : F^n \to F$ in the following way:*

$$\bar{f}(x_1,..,x_n) = \sum_{\{a\in\{0,1\}^n | f(a)=1\}} p_a(x_1,..,x_n)$$

From the previous discussion it is clear that:

- $deg(\bar{f}) \leq n$.
- $\bar{f}$ identifies with f, when all n inputs are in $\{0,1\}$.

Assuming $|F| = n^{O(1)}$, one can encode it's elements in binary form, and view $\bar{f}$, as a function from $\{0,1\}^{O(n\log n)}$ to $\{0,1\}^{O(\log n)}$. In this case the input sizes defer only slightly. and if one takes a field with arithmetic that can be computed efficiently, we get the following lemma:

Lemma 2.5 *If there exists a family of boolean functions $f = \{f_n\}$, that is computable in exponential time, yet cannot be computed by polynomial circuits, then there exist a family of functions $g = \{g_n\}$, $g_n : F^n \to F$ which has the same properties, and also has a representation as a polynomial of degree at most n.*

Proof: We simply take $g_n = \bar{f}_n$, and we have seen that it is indeed a polynomial of degree at most n. Notice that the size of the input of g_n is polynomial in that of f_n. this means that "polynomial circuits" are of the same size, with respect to both functions. Since g_n contains a "copy" of f_n, then it is at least as hard. The polynomial of g_n can be computed in exponential time when one is given an oracle for f_n, and this oracle can be replaced by an exponential procedure. Evaluating the polynomial on a given input also takes exponential time. We conclude that g can be computed in exponential time.

•

Remark 2.6 *Actually, being a bit more careful we could have g_n be a function over $O(n)$ bits instead of $O(n \log n)$ bits. We can we find a copy of $\{0,1\}^{\log n}$, in F, and view f_n as a function from F^k (where $k = \frac{n}{\log n}$ to $\{0,1\}$. We would now need to generalize the definition of P_a. We define $P_{a_1,..,a_k}$:*

$$P_{a_1,..,a_k}(x_1,..,x_k) = \prod_{1 \le i \le k} \cdot \prod_{b \in (\{0,1\}^{\log n} \setminus \{a_i\})} \frac{x_i - b}{a_i - b}$$

Note that now the p's are of degree no more than $\frac{n^2}{\log n}$. Now we would define $\bar{f} : F^k \to F$:

$$\bar{f}(x_1,..,x_k) = \sum_{\{a \in \{0,1\}^n | f(a)=1\}} p_a(x_1,..,x_n)$$

We would get that $\bar{f}$ is a polynomial of degree no more than $\frac{n^2}{\log n}$, and counting the number of bits in the input of $\bar{f}$, we see that we reduced it to n. The interpolation part of the proof can continue as before.
While this remark doesn't improve the final result we prove in this chapter, It is essential for one of the improvements of the theorem that we sketch in the next chapters.

2.3.2 Random self reducibility

In this section we start from the assumption:

$$EXP \setminus P/poly \neq \emptyset$$

Or in words, that there exist a function $f = \{f_n\}$, that is computable in exponential time, and uncomputable by polynomial circuits. From the point of view of distributional complexity, the second condition means that for all families of polynomial circuits $C = \{C_n\}$:

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = f(x)) < 1$$

At the end of the section we will conclude that there exists a function $g = \{g_n\}$, which is computable in exponential time, and

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = g(x)) < 1 - \frac{1}{3n}$$

This means that g is somewhat hard in the distributional complexity sense.

Theorem 2.7 *If $EXP \setminus P/poly \neq \emptyset$, then there exist a family of functions $g = \{g_n\}$, such that:*

1. *$g_n : \{0,1\}^{O(n \log n)} \to \{0,1\}^{O(\log n)}$*
2. *g is computable in exponential time.*
3. *For every family of polynomial circuits $C = \{C_n\}$,*

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = g_n(x)) \le 1 - \frac{1}{3n}$$

Proof: We already know from 2.5 that $EXP \setminus P/poly \neq \emptyset$, implies that there exist a family of functions that can be computed in exponential time, cannot be computed by polynomial circuits, and take the form of $g_n : F^n \to F$, where F is a field with $n^{\Theta(1)}$ elements. Moreover, g_n is a polynomial of degree at most n. So such g satisfies the first two conditions. We will now prove that if there exist a family of polynomial circuits $C = \{C_n\}$, such that:

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = g_n(x)) > 1 - \frac{1}{3n}$$

Then there exists a family of polynomial circuits $C' = \{C'_n\}$, which compute g, for every input. For $x, \beta \in F^n$ define a polynomial in one variable over F:

$$p_{x,\beta}(t) = g_n(x + t \cdot \beta)$$

Note that:

- For all x, β, $deg(p_{x,\beta}) \leq n$.
- For all x, β, $p_{x,\beta}(0) = g_n(x)$.

Recall that there is a unique polynomial of degree n that passes through given $n+1$ points, and moreover that one can compute this polynomial efficiently, given $n+1$ points. We construct a probabilistic circuit C':

On input x:
Pick a uniformly chosen $\beta \in_R F^n$.
Take $n+1$ distinct values in F, $t_1, .., t_{n+1}$.
Activate C on $x + t_i \cdot \beta$, we will denote the i'th output by d_i.
Compute the polynomial q that passes through $t_i, d_{i_{1 \leq i \leq i+1}}$ output $q(0)$

First of all note that C' is of size $poly(n) + n \cdot size(C)$ which is polynomial in n. Secondly, the functions $h_1(y) = y + \beta$, $h_2(y) = t \cdot \beta$ are permutations over F^n for all β. This means that when β is chosen uniformly, $t_i \cdot \beta$, is uniformly distributed for all $1 \leq i \leq n+1$. This means that for all i:

$$Pr_\beta(C(x + t_i \cdot \beta) = g_n(x + t_i \cdot \beta)) > 1 - \frac{1}{3n}$$

We conclude that:

$$Pr_\beta(\exists i : C(x = t_i \cdot \beta) \neq g_n(x + t_i \cdot \beta)) < \frac{n}{3n} = \frac{1}{3}$$

This means that with probability $\frac{2}{3}$, all the d_i's equal $p(t_i)$. When this happens $q = p_{x,\beta}$, and so $q(0) = g_n(x)$. So we have managed to build a circuit that is correct with good probability on **every** input. (The process of reducing the problem of solving a function on every input (with good probability) to this of solving it over a randomly chosen input, is called: "Random self reducibility"). We now transform this circuit to a deterministic one that is correct on every every input by using theorem 2.2.

•

We have constructed a function that is somewhat inaproximable by polynomial circuits. As far as we know there may be a polynomial circuit that computes the function on a very large fraction of the inputs. The next section is devoted to "amplifying" the hardness of the function, making the success probability of every polynomial circuit negligible.

2.4 Hardness Amplification

2.4.1 The concept

At this point we have a function $g = \{g_n\}$, that is computable in exponential time, yet for every family of polynomial circuits

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = g_n(x)) \leq 1 - \frac{1}{3n}$$

We would like to have a function on which we can apply the Nisan-Wigderson generator. For such a function we need the success probability to negligible, (meaning smaller than every polynomial). Remember, that we have at our disposal a very powerful tool, namely the Goldreich-Levin theorem (1.20), which enables us to make a non-boolean hard function into a boolean one with the same hardness. So the fact that g is not boolean is not a real obstacle. Our basic idea is very simple, let's ask the circuit to compute the function g on a lot of independently chosen random inputs. It seems that taking n^2, such inputs will reduce the success probability of circuits to $(1 - \frac{1}{3n})^{n^2} = e^{-\Omega(n)}$, which is negligible. However, the situation here is much more complicated than the case of amplifying the success probability of probabilistic algorithms. The problem is that unlike the case of amplifying the success probability of probabilistic algorithms, the circuit C, may query all the inputs when trying to solve the first instance. While this doesn't seem helpful, it makes the runs dependent, and we can no longer multiply probabilities. Indeed, in some sense we know that a proof that works in this manner will not be able to reduce the success probability to exponentially small, as expected. However, for us it safises that the probability goes to zero multiplied by any polynomial, which we can achieve in such methods. It should be noted that hardness amplification, is an interesting field which has many other uses. Our failure to prove better results may not stop us from getting a good generator, but is quite surprising, since the question seems very basic. Current amplifications are also not optimal, in respect to the tradeoff between sizes of the circuits. Suppose that we know that every circuit of size s cannot compute the the function f, with success probability better than p. we already commented that taking k, independent inputs, we are not able to prove that the success probability drops to p^k. What we can prove is that it drops to $p^k + \epsilon$, for every circuit of size $s' = s(\frac{\epsilon}{n})^{O(1)}$, note that $s' << s$, this means that we are able to bound the success probability of a smaller circuit. Not only does the circuit need to solve a lot of instances, it also has to be much smaller than the original circuit! However, as mentioned before, from the point of view of derandomizing BPP this unsatifayable results are enough.

2.4.2 Xor lemmas and Product theorems

The basic situation is simple. We have a function f, such that we know that for every circuit C of size s:

$$Pr_{x \in_R \{0,1\}^n}(C(x) = f(x)) \leq p$$

We can now try to "amplify the hardness" in several ways:

Definition 2.8 *For every function $f : \{0,1\}^n \to \{0,1\}$, and k, define the following functions:*

1. $f^{\oplus k} : (\{0,1\}^n)^k \to \{0,1\}$,
$$f^{\oplus k}(x_1, .., x_k) = f(x_1) \oplus, .., \oplus f(x_k)$$
2. $f^{\otimes k} : (\{0,1\}^n)^k \to \{0,1\}^k$
$$f^{\otimes}(x_1, .., x_k) = (f(x_1), .., f(x_k))$$
3. $f^{GL(k)} : (\{0,1\}^n)^k \times \{0,1\}^k \to \{0,1\}$
$$f^{GL(k)}(x_1, .., x_k; y_1, .., y_k) = GL(f(x_1), .., f(x_k); y_1, .., y_k) = \oplus_{i:y_i=1} f(x_i)$$

In the first function, the circuit is required to compute the xor of the values of k instances of f. At the second it is required to just output the values of k instances of f. The third is a bit strange, additional inputs are added according to which, the instances for the xor are selected. All three formulations are based on the same intuition, if one instance is a bit hard, the function for k instances, seems very hard. The claim that $f^{\oplus k}$ is hard is called: a "xor lemma". The claim that $f^{\otimes k}$ is hard is called a "product theorem". Historically, xor lemmas appeared before product theorems. xor lemmas, are a bit better, as the target function is boolean. But since we have the Goldreich-Levin theorem, we can deduce the existence of a hard boolean function, from the existence of a hard non boolean one. Taking a better look, this claim can be phrased non-formally in the following way: If there exists a small circuit for $f^{GL(k)}$, which has a non-negligible advantage over $\frac{1}{2}$, then there exists a small circuit for $f^{\otimes k}$, which has non-negligible success probability. (for exact parameters, check theorem 1.20). If you have small circuits for $f^{\oplus k}$, for all k, then you certainly can build a relatively small circuit for $f^{GL(k)}$. We conclude that proving a direct product theorem, already proves a xor lemma. (again, the parameters work out alright for our uses). The other side is more intuitive but not trivial. now, we have a small circuit that computes $f^{\otimes k}$, with non negligible success probability, and you want to construct a small circuit which has non-negligible advantage when computing $f^{\oplus k}$. The trivial idea, of taking the xor of the outputs, does not work. It may very well be, that the xor of he values the circuit output is fixed, and so have no advantage over $\frac{1}{2}$. Instead, one shows that there exists a small circuit which has non-negligible advantage for $f^{GL(k)}$. This is true, since the randomness in the choice of the y's makes the answer of the circuit uniform, when it gives a bad answer. So in the (relatively rare) event, that the original circuit computed $f^{\otimes k}$ correctly, this induces an advantage over $\frac{1}{2}$. Now, one can construct a small circuit for $f^{\oplus \frac{k}{2}}$, since the size of the subset over which the xor takes place, is very rarely considerably greater than $\frac{k}{2}$.
So, we have shown that all three variants have roughly, the same hardness. In most of the original papers, the amplification was achieved using a xor lemma. In light of this discussion, this hardly matters, and since proving product theorems, seems a more general question, we go about proving one in the next section.

2.4.3 Proof of product theorem

We start by proving a product theorem for $k = 2$, (that is for two instances).

Theorem 2.9 *Let f_1, f_2 be two functions, over domains X, Y respectively. Let D_1, D_2 be probability distributions over X, Y. suppose that:*

1. *For all circuits C of size s_1:*

$$Pr_{x \in_{D_1} X}(C(x) = f_1(x)) \leq p_1$$

2. *For all circuits C of size s_2:*

$$Pr_{y \in_{D_2} Y}(C(y) = f_2(y)) \leq p_2$$

Then for every $\epsilon > 0$, and circuit C of size $s = min(O(\frac{s_1 \epsilon}{-\log \epsilon}), s_2)$:

$$Pr_{x \in_{D_1} X, y \in_{D_2} Y}(C(x, y) = (f_1(x), f_2(y))) \leq p_1 p_2 + \epsilon$$

Proof: Let C be a circuit of size s, which attempts to compute both functions. Informally, the proof can be described as follows: The success probability of C can be written as the probability that it is right on the first coordinate given that it is right on the second one, times the probability that it is right on the second coordinate. The second term is bounded by p_2. To bound the first term, we construct a circuit for the x's that samples y from the conditioned event, and activates C on x, y. This bounds the first term by p_1. There are two difficulties:

1. Conditioning changes the distribution of the x's. This is overcome by fixing x, before conditioning, and then taking the expectation over the x's.

2. For some x's, the probability over y's that C succeeds in the second coordinate, may be small, and we will not be able to sample such a y. This is overcome by noticing that though the proportion of those x's may be very large, their contribution to the success probability of C is small.

We denote the i'th output of C when given x, y, by $C_i(x, y)$. We will evaluate the success probability of C:

$$\begin{aligned} &Pr_{x \in_{D_1} X, y \in_{D_2} Y}(C(x, y) = (f_1(x), f_2(y))) \\ &= E_{x \in_{D_1} X}(Pr_{y \in_{D_2} Y}(C(x, y) = (f_1(x), f_2(y)))) \\ &= E_{x \in_{D_1} X}(Pr_{y \in_{D_2} Y}(C_1(x, y) = f_1(x) | C_2(x, y) = f_2(y)) \cdot Pr_{y \in_{D_2} Y}(C_2(x, y) = f_2(y))) \end{aligned}$$

(The last equality is correct even if the probability of the conditioned event is zero, in this case the conditional probability is defined to be zero). define:

$$A = \{x | Pr_{y \in_{D_2} Y}(C_2(x, y) = f_2(y)) \geq \frac{\epsilon}{2}\}$$

We separate the expectation, according to the different kind of x's.

$$= Pr_{x\in_{D_1}X}(A)E_{x\in_{D_1}A}(Pr_{y\in_{D_2}Y}(C_1(x,y)=f_1(x)|C_2(x,y)=f_2(y))Pr_{y\in_{D_2}Y}(C_2(x,y)=f_2(y))) \tag{2.1}$$

$$+Pr_{x\in_{D_1}X}(A^c)E_{x\in_{D_1}A^c}(Pr_{y\in_{D_2}Y}(C_1(x,y)=f_1(x)|C_2(x,y)=f_2(y))Pr_{y\in_{D_2}Y}(C_2(x,y)=f_2(y))) \tag{2.2}$$

To bound (2.2) we note that or each $x \notin A$, the term inside the expectation is bounded by $\frac{\epsilon}{2}$, and therefore (2.2) is bounded by $\frac{\epsilon}{2}$. We now proceed and bound (2.1). For all $x \in X$, (and therefore for all $x \in A$):

$$Pr_{y\in_{D_2}Y}(C_2(x,y)=f_2(y)) \leq p_2$$

This is because for all x, the circuit $C_x(y)=C(x,y)$, is of size $s<s_2$ Applying this to (2.1), we get:

$$\leq p_2 Pr_{x\in_{D_1}X}(A)\cdot E_{x\in_{D_1}A}(Pr_{y\in_{D_2}Y}(C_1(x,y)=f_1(x)|C_2(x,y)=f_2(y))) \tag{2.3}$$

Claim 3

$$E_{x\in_{D_1}A}(Pr_{y\in_{D_2}Y}(C_1(x,y)=f_1(x)|C_2(x,y)=f_2(y))) \leq \frac{p_1}{Pr_{x\in_{D_1}X}(A)}+\frac{\epsilon}{2}$$

Proof: Consider the following (probabilistic) circuit C':
On input $x \in A$:
Choose $y_1,..,y_t \in_{D_2} Y$
"Compute" $f_2(y_i)$ for all $1 \leq i \leq t$, and compare it to the output of $C_2(x,y_i)$.
denote:

$$S_x = \{y_i | C_2(x,y_i)=f_2(y_i)\}$$

If $S_x=\emptyset$, output: "don't know".
Otherwise picks $y' \in_R S_x$, and outputs $C(x,y')$.

(The circuit can "compute" f_2 over it's randomly picked $y's$, since a probabilistic circuit is merely a distribution over deterministic ones. From the point of view of each circuit, y is just a constant, and so is $f_2(y)$). Since x is in A we know that:

$$Pr_{y\in_{D_2}Y}(C_2(x,y)=f(y)) > \frac{\epsilon}{2}$$

This means that:

$$Pr_{y_1,..,y_t}(S_x=\emptyset) \leq (1-\frac{\epsilon}{2})^t \leq \frac{\epsilon}{2}$$

for $t=\Theta(\frac{-\log\epsilon}{\epsilon})$. When $S_x \neq \emptyset$, the distributions

- $y' \in_R S_x$
- $y \in_{D_2} Y | C_2(x,y)=f_2(y)$

are identical. We conclude that for all $x \in A$:

$$Pr_r(C'_r(x) = f_1(x)) \geq Pr_{y \in_{D_2} Y}(C_1(x,y) = f_1(x) | C_2(x,y) = f_2(y)) - \frac{\epsilon}{2}$$

(Where r is the "random coins" used by C'). This is also true taking expectation over x:

$$E_{x \in_{D_1} A}(Pr_r(C'_r(x) = f_1(x))) \geq E_{x \in_{D_1} A}(Pr_{y \in_{D_2} Y}(C_1(x,y) = f_1(x) | C_2(x,y) = f_2(y))) - \frac{\epsilon}{2} \quad (2.4)$$

Note that:

$$E_{x \in_{D_1} A}(Pr_r(C'(x) = f_1(x))) = Pr_{x \in_{D_1} A, r}(C'(x) = f_1(x))$$

We can fix r to some value r', which achieves the same success probability. This fixing does not change the size of the circuit. So we can continue, and get:

$$\leq Pr_{x \in_{D_1} A}(C'_{r'}(x) = f_1(x)) \leq \frac{Pr_{x \in_{D_1} X}(C'_{r'}(x) = f_1(x))}{Pr_{x \in_{D_1} X}(A)}$$

Where the last inequality is simply using properties of conditional probabilities. However, since $size(C'_{r'}) = O(size(C)\frac{-\log \epsilon}{\epsilon}) \leq s_1$, it cannot have success probability greater than p_1. The claim follows by putting everything together.

•

Using the bounds we have on (2.1) and (2.2), we have that:

$$Pr_{x \in_{D_1} X, y \in_{D_2} Y}(C(x,y) = (f_1(x), f_2(y))) \leq (p_1 + \frac{\epsilon}{2})p_2 + \frac{\epsilon}{2} \leq p_1 p_2 + \epsilon$$

•

The proof for arbitrary k, follows by induction. At each stage the k inputs may be split to two parts, defining two functions f_1, f_2 over the two groups of variables. If this is done with patience and care, one gets the following theorem:

Theorem 2.10 *Let f be a function over domain X. Let D be a probability distribution over X. Suppose that for all circuits C of size s:*

$$Pr_{x \in_D X}(C(x) = f(x)) \leq p$$

then for every $\epsilon > 0$, set $k = O(\frac{-\log \epsilon}{1-p})$. For all circuits C of size $sO(\frac{\epsilon}{-\log \epsilon})$

$$Pr_{x_1,..,x_k \in_{D^k} X^k}(C(x_1,..,x_k) = (f(x_1),..,f(x_k))) \leq \epsilon$$

For our purposes we will need the following version:

Theorem 2.11 *Let f be a function over $\{0,1\}^n$, Suppose that for every circuit C of size polynomial in n:*

$$Pr_{x \in_R \{0,1\}^n}(C(x) = f(x)) \leq 1 - \frac{1}{p(n}$$

(Where p is a polynomial). Then there exists a function $g : \{0,1\}^{kn} \to \{0,1\}^k$, where $k = poly(n)$, such that g is unapproximable by polynomial circuits.

Proof: We simply take $k = np(n)$, and have $g = f^{\otimes k}$. Using the previous theorem, we know that if there exists a circuit C that computes g with success probability $\epsilon(n) = \frac{1}{n^c}$, then there exists a circuit D of size $size(C)n^c \log n$, that computes f with success probability better than $1 - \frac{1}{p(n)}$. This is a contradiction. •

2.5 The main theorem

In this section we simply put everything together.

Theorem 2.12 *Babai-Fortnow-Nisan-Wigderson*

$$Exp \setminus P/Poly \neq \emptyset \Rightarrow BPP \subseteq \cap_{\delta>0} dtime(2^{n^{\delta}})$$

Proof: We have already proved (2.7) that if $EXP \setminus P/poly \neq \emptyset$, then there exist a family of functions $f = \{f_n\}$, such that:

1. $f_n : \{0,1\}^{O(n \log n)} \to \{0,1\}^{O(\log n)}$
2. f is computable in exponential time.
3. For every family of polynomial circuits $C = \{C_n\}$,

$$Pr_{x \in_R \{0,1\}^n}(C_n(x) = f_n(x)) \leq 1 - \frac{1}{3n}$$

We now use theorem 2.11 to construct a function g that is extremely hard from f which is only mildly hard. Note that any polynomial in $n \log n$ is bounded by some polynomial in n, and so we get that there exists a function $g : \{0,1\}^{poly(n)} \to \{0,1\}^{poly(n)}$, Which is unapproximable. g can be computed in exponential time. This is because computing g amounts to a polynomial number of computations of f. At this point we can use theorem 1.41. •