6.2 Pseudo-randomness

The above definition of randomness is very robust, if not practical. True random generators are rarely used in computing. The problem is *not* that making a true random generator is impossible: we just saw efficient ways to perfect the distributions of biased random sources. The reason lies in many extra benefits provided by pseudorandom generators. E.g., when experimenting with, debugging, or using a program one often needs to repeat the exact same sequence. With a truly random generator, one actually has to record all its outcomes: long and costly. The alternative is to generate ***pseudo-random*** strings from a short seed. Such methods were justified in [Blum, Micali 84, Yao 82]:

First, take any one-way permutation $F_n(x)$ (see sec. 6.3) with a ***hard-core*** bit (see below) $B_p(x)$ which is easy to compute from x, p, but infeasible to guess from $p, n, F_n(x)$ with any noticeable correlation.
Then take a random ***seed*** of three k-bit parts x_0, p, n and Repeat: ($S_i {\leftarrow} B_p(x_i)$; $x_{i+1} {\leftarrow} F_n(x_i)$; $i {\leftarrow} i{+}1$).

We will see how distinguishing outputs S of this generator from strings of coin flips would imply the ability to invert F. This is infeasible if F is one-way. But if P=NP (a famous open problem), no one-way F, and no pseudorandom generators could exist.

By Kolmogorov's standards, pseudo-random strings are not random: let G be the generator; s be the seed, $G(s) = S$, and $\|S\| \gg k = \|s\|$. Then $K(S) \le O(1) + k \ll \|S\|$, thus violating Kolmogorov's definition. We can distinguish between truly random and pseudo-random strings by simply trying all short seeds. However this takes time exponential in the seed length. Realistically, pseudo-random strings will be as good as a truly random ones if they can't be distinguished in feasible time. Such generators we call ***perfect***.

Theorem: [Yao 82] Let $G(s) = S \in \{0,1\}^n$ run in time t_G. Let a probabilistic algorithm A in expected (over internal coin flips) time t_A accept $G(s)$ and truly random strings with different by d probabilities. Then, for random i, one can use A to guess S_i from $S_{i+1}, S_{i+2}, \ldots$ in time $t_A + t_G$ with correlation $d/O(n)$.

Proof. Let r_i be the probability that A accepts $S = G(s)$ modified by replacing its first i digits with truly random bits. Then r_0 is the probability of accepting $G(s)$ and must differ by d from the probability r_n of accepting random string. Then $r_{i-1} - r_i = d/n$, for randomly chosen i.
Let R_0 and R_1 be the probabilities of accepting $r0x$ and $r1x$ for $x = S_{i+1}, S_{i+2}, \ldots$, and random $(i{-}1)$-bit r.
Then $(R_1{+}R_0)/2$ averages to r_i, while $R_{S_i} = R_0{+}(R_1{-}R_0)S_i$ averages to r_{i-1} and $(R_1{-}R_0)(S_i{-}1/2)$ to $r_{i-1}{-}r_i = d/n$. So, $R_1{-}R_0$ has the stated correlation with S_i. $\square$

If the above generator was not perfect, one could guess S_i from the sequence $S_{i+1}, S_{i+2}, \ldots$ with a polynomial (in $1/\|s\|$) correlation. But, $S_{i+1}, S_{i+2} \ldots$ can be produced from p, n, x_{i+1}.
So, one could guess $B_p(x_i)$ from $p, n, F(x_i)$ with correlation d/n, which cannot be done for hard-core B.

Hard Core. The key to constructing a pseudorandom generator is finding a hard core for a one-way F. The following B is hard-core for any one-way F, e.g., for Rabin's OWF in sec. 6.3.
[Knuth 97] has more details and references.

Let $B_p(x) = (x \cdot p) = (\sum_i x_i p_i \bmod 2)$. [Goldreich, Levin 89] converts any method g of guessing $B_p(x)$ from $p, n, F(x)$ with correlation ε into an algorithm of finding x, i.e. inverting F (slower ε^2 times than g).

Proof. (Simplified with some ideas of Charles Rackoff.) Take $k = \|x\| = \|y\|$, $j = \log(2k/\varepsilon^2)$, $v_i = 0^i 1 0^{k-i}$. Let $B_p(x) = (x \cdot p)$ and $b(x,p) = (-1)^{B_p(x)}$. Assume, for $y = F_n(x)$, $g(y,p,w) \in \{\pm 1\}$ guesses $B_p(x)$ with correlation $\sum_p 2^{-\|p\|} b(x,p) g_p > \varepsilon$, where g_p abbreviates $g(y,p,w)$, since w, y are fixed throughout the proof.

$(-1)^{(x \cdot p)} g_p$ averaged over $> 2k/\varepsilon^2$ random pairwise independent p deviates from its mean (over all p) by $< \varepsilon$ (and so is > 0) with probability $> 1 - 1/2k$. The same for $(-1)^{(x \cdot [p+v_i])} g_{p+v_i} = (-1)^{(x \cdot p)} g_{p+v_i} (-1)^{x_i}$.

Take a random $k \times j$ binary matrix P. The vectors Pr, $r {\in} \{0,1\}^j \setminus \{0^j\}$ are pairwise independent. So, for a fraction $\ge 1 - 1/2k$ of P, $\operatorname{sign}(\sum_r (-1)^{xPr} g_{Pr+v_i}) = (-1)^{x_i}$. We could thus find x_i for all i with probability $> 1/2$ if we knew $z = xP$. But z is short: we can try all its 2^j possible values and check $y = F_n(x)$ for each !

So the inverter, for a random P and all i, r, computes $G_i(r) = g_{Pr+v_i}$. It uses Fast Fourier on G_i to compute $h_i(z) = \sum_r b(z,r) G_i(r)$. The sign of $h_i(z)$ is the i-th bit for the z-th member of output list. $\square$