B.3 Efficient Algorithms

To generalize the results from Section B.1 we first define the concepts of advantage and error probability for PPCs.

Let $L \subseteq \mathbb{F}_2^*$ be a language over the binary alphabet $\mathbb{F}_2$, and set $L_n := L \cap \mathbb{F}_2^n$. Let f be a map

$$f\colon L \longrightarrow \mathbb{F}_2^* \text{ with } f(L_{r(n)}) \subseteq \mathbb{F}_2^{s(n)} \tag{2}$$

where $r(n)$ is the monotonically increasing sequence of indices i with $L_i \neq \emptyset$. We want to compute this map by a PPC as in (1).

Examples

1. The function $f(x, y, z) := xy \bmod z$ for n-bit integers x, y, z is computable by a (deterministic) circuit

$$C_n\colon \mathbb{F}_2^{3n} \longrightarrow \mathbb{F}_2^n$$

   of size $\#C_n = \mathrm{O}(n^3)$ (with error probability 0). Here $r(n) = 3n$ and $s(n) = n$.

2. Let L be the set of (binary encoded) odd integers ≥ 3, and $f\colon L \longrightarrow \mathbb{F}_2$ be the primality indicator as in Section B.1. There we saw a PPC for the strong pseudoprime test of size $\mathrm{O}(n^3)$ with advantage $\frac{1}{4}$ and error probability $\frac{1}{4}$ (constant with respect to n). Using t bases we get a size of $\mathrm{O}(tn^3)$, and an error probability of $\frac{1}{4^t}$.

Definition 1 A function $\varphi\colon \mathbb{N} \longrightarrow \mathbb{R}_+$ is called **(asymptotically) negligible** if for each nonconstant polynomial $\eta \in \mathbb{N}[X]$

$$\varphi(n) \leq \frac{1}{\eta(n)} \quad \text{for almost all } n \in \mathbb{N}.$$

In other words, $\varphi(n)$ tends to 0 faster than the inverse of any polynomial.

Example An obvious example is $\varphi(n) = 2^{-n}$.

Definition 2 Sei $f\colon L \longrightarrow \mathbb{F}_2^*$ be as in (2). Let C be a PPC that computes f on $L_{r(n)}$ with an error probability of ε_n. Assume ε_n is a negligible function of n. Then C is called an **efficient probabilistic algorithm** for f.

f is called **(probabilistically) efficiently computable** if there is an efficient algorithm for f.

This definition substantiates the idea of an algorithm that is "efficient for almost all input tuples" (or input strings if the input is taken from a language L).

For RABIN's primality test, that is the repeated execution of the strong pseudoprime test, we satisfy this requirement by letting the number t of bases grow with n. In order to get a polynomial family we upgrade t to a polynomial $\tau \in \mathbb{N}[X]$. Then C_n has n deterministic input nodes, and $n\tau(n)$ probabilistic ones. The size is $\mathrm{O}(n^3\tau(n))$, and the error probability, $\frac{1}{4^{\tau(n)}}$. Thus we have shown:

Proposition 28 RABIN*'s primality test is an efficient probabilistic algorithm for deciding primality.*