1.2 The Binary Power Algorithm

The procedure for raising powers in a quite efficient way has a natural description in the abstract framework of a multiplicative semigroup H. The task is: Compute the power x^n for $x \in H$ and a positive integer n—the product of n factors x—by *as few multiplications as possible.* The naive direct method,

$$x^n = x \cdot (x \cdots x)\,,$$

involves $n-1$ multiplications. The expense is proportional with n, hence grows *exponentially* with the number $\ell(n)$ of bits (or decimal places) of n. A much better idea is the **binary power algorithm**. In the case of an additively written operation (strictly speaking for the semigroup $H = \mathbb{N}$) it is known also as Russian peasant multiplication, and was known in ancient Egypt as early as 1800 B. C., in ancient India earlier than 200 B. C.

The specification starts from the binary representation of the exponent n,

$$n = b_k 2^k + \cdots + b_0 2^0 \quad \text{with } b_i \in \{0,1\},\ b_k = 1,$$

thus $k = \lfloor \log_2 n \rfloor = \ell(n) - 1$. Then

$$x^n = (x^{2^k})^{b_k} \cdots (x^2)^{b_1} \cdot x^{b_0}.$$

This suggests the following procedure: Compute $x, x^2, x^4, \ldots, x^{2^k}$ in order by squaring k times (and keeping the intermediate results), and then multiply the x^{2^i} that have $b_i = 1$. The number of factors is $\nu(n)$, the number of 1's in the binary representation. In particular $\nu(n) \leq \ell(n)$. This makes a total of $\ell(n) + \nu(n) - 2$ multiplications.

We have shown:

Proposition 1 *Let H be a semigroup. Then for all $x \in H$ and $n \in \mathbb{N}$ we can compute x^n by at most $2 \cdot \lfloor \log_2 n \rfloor$ multiplications.*

This expense is only *linear* in the bit length of n. Of course to assess the complete expense we have to account for the cost of multiplication in the semigroup H.

Here is a description as pseudocode:

```
Procedure BinPot
  Input parameters:
    $x$ = base
      [locally used for storage of the iteratively computed squares]
    $n$ = exponent
  Output parameters:
    $y$ = result $x^n$
      [locally used for accumulation of the partial product]
  Instructions:
    $y := 1$.
    while $n > 0$:
      if $n$ is odd: $y := yx$.
      $x := x^2$.
      $n := \lfloor n/2 \rfloor$.
```

Remarks

1. The algorithm is almost optimal, but not completey. The theory of "addition chains" in number theory yields an asymptotic behaviour of $\log_2 n$ for the average minimum number of multiplications, roughly half the value from Proposition 1.

2. That the numbers of involved multiplications differ depending on the exponent is the starting point of *timing* and *power attacks* invented by Paul Kocher. Imagine a device, say a smart card, that computes powers with a secret exponent. Then the different timings or power consumptions reveal information about the exponent.