Relaxed algorithms for multiplication

This chapter introduces the notions of online and relaxed algorithms. First, we present a general context of p-adic computations that will be in use for the next few chapters. Then, we recall the current relaxed algorithms for the multiplication, and we give for the first time a thorough analysis of their arithmetic complexity. In a third time, we introduce a new relaxed algorithm for the multiplication using *middle* and *short* product, that improves by a constant factor the previous relaxed multiplication. Finally, we give some timings to confirm the good behavior of relaxed algorithms with middle product.

1.1 Computing with p-adics

This section introduces several important notions and notation regarding p-adic computations, which will be in use for the next few chapters.

1.1.1 Basic definitions

Let R be a commutative ring with unit. We consider an element $p \in R - \{0\}$, and we write R_p for the completion of the ring R for the p-adic valuation. We will assume that $R/(p)$ is a field (equivalently, that p generates a maximal ideal). This is not needed for the algorithms in this chapter, but will be useful later on when we deal with linear algebra modulo (p). We also assume that $\cap_{i \in \mathbb{N}} (p^i) = \{0\}$, so that R can be seen as a subset of R_p.

An element $a \in R_p$ is called a *p-adic*; it can always be written (in a non unique way) $a = \sum_{i \in \mathbb{N}} a_i p^i$ with coefficients $a_i \in R$.

To get a unique representation of elements in R_p, we will fix a subset M of R such that the projection $\pi: M \rightarrow R/(p)$ is a bijection. Then, any element $a \in R_p$ can be uniquely written $a = \sum_{i \in \mathbb{N}} a_i p^i$ with coefficients $a_i \in M$. The operations mod and quo are then defined as

$$a \bmod p = a_0 \quad \text{and} \quad a \operatorname{quo} p = \sum_{i > 0} a_i p^{i-1}.$$

We will suppose that for all $a \in M$, $-a$ is in M as well.

Two classical examples are the formal power series ring $\mathbb{k}[[X]]$, which is the completion of the ring of polynomials $\mathbb{k}[X]$ for the ideal (X), and the ring of p-adic integers $\mathbb{Z}_p$, which is the completion of the ring of integers $\mathbb{Z}$ for the ideal (p), with p a prime number. For $R = \mathbb{k}[X]$, we naturally take $M = \mathbb{k}$; for $R = \mathbb{Z}$, we choose $M = \left\{-\frac{p-1}{2}, \ldots, \frac{p-1}{2}\right\}$ if p is odd and $M = \{0, 1\}$ for $p = 2$.

Once M has been fixed, we have a well-defined notion of *length* of a (non-zero) p-adic: if $a = \sum_{i \in \mathbb{N}} a_i p^i$, then we define

$$\lambda(a) := 1 + \sup (i \in \mathbb{N} \mid a_i \neq 0),$$

so that $\lambda(a)$ is in $\mathbb{N}_{>0} \cup \{\infty\}$; for $a = 0$, we take $\lambda(a) = 0$. Since M is invariant through sign change, we have that $\lambda(-a) = \lambda(a)$ for all a. We will make the following assumptions:

- λ verifies the conditions

$$\begin{aligned} \lambda(a+b) &\leqslant \max (\lambda(a), \lambda(b)) + 1 \\ \lambda(a\,b) &\leqslant \lambda(a) + \lambda(b); \end{aligned}$$

- all elements of $R \subset R_p$ have finite length (this excludes cases where for instance R is already complete with respect to the (p)-adic topology).

These assumptions are satisfied in the two main cases above (with further simplifications in the polynomial case, since no carries appear in the case of addition); note that $\lambda(a-b)$ satisfies the same inequality as $\lambda(a+b)$.

For any $a \in R_p$ and integers $0 \leqslant r \leqslant s$, we define the truncated p-adic $a_{r \ldots s}$ as

$$a_{r \ldots s} := a_r + a_{r+1}\, p + \cdots + a_{s-1}\, p^{s-1-r} \in R.$$

We call *p-adics at precision n* the set of all truncations $a_{0 \ldots n}$ of p-adics $a \in R_p$ (for the two main cases we have in mind, they are simply plain integers, resp. polynomials). We say that we have computed a p-adic at precision n if the result holds modulo p^n.

1.1.2 Basic operations

Algorithmically, we represent p-adics through their base-M expansion, that is, through a sequence of coefficients in M. Roughly speaking, we measure the cost of an algorithm by the number of arithmetic operations with operands in M it performs. More precisely, we assume that we can do the following at unit cost:

- given a_0, b_0 in M, compute the coefficients c_0, c_1 of $c = a_0\, b_0$ at unit cost, and similarly for the coefficients of $a \pm b$
- given a_0 in $M - \{0\}$, compute b_0 in $M - \{0\}$ such that $a_0\, b_0 = 1 \bmod p$

Remark that when $R = \mathbb{k}[X]$, we are simply counting arithmetic operations in $\mathbb{k}$.

The main operations we will need on p-adics are sum and difference, as well as multiplication and a few of its variants (of course, these algorithms only operate on truncated p-adics). Addition (and subtraction) are easy to deal with:

Lemma 1.1. *The following holds:*

- *Given two p-adics a, b of length at most ℓ, one can compute $a+b$ in time $\mathcal{O}(\ell)$*
- *Given p-adics $a_1, \ldots, a_N$ of length at most ℓ, the p-adic $A = \sum_{i=1}^{N} a_i$ has length $\mathcal{O}(\log (N) + \ell)$, and one can compute it in time $\mathcal{O}(N \ell)$.*
- *Given p-adics $a_1, \ldots, a_N$ of length at most ℓ, the p-adic $A = \sum_{i=1}^{N} a_i\, p^i$ has length $\mathcal{O}(N + \ell)$, and one can compute it in time $\mathcal{O}(N \ell)$.*

Proof. The first point is easily dealt with by induction on ℓ; we will see the algorithm in more detail in Example 1.5 below. To handle the second one, we build a tree adder, which has depth $\mathcal{O}(\log(N))$. The length bound follows; the complexity bound comes from noticing that we do $\mathcal{O}(N)$ additions of p-adics of length ℓ, $\mathcal{O}(N/2)$ additions of p-adics of length $\ell+1$, $\mathcal{O}(N/4)$ additions of p-adics of length $\ell+2$, etc.

To deal with the last point, note that for all i, $a_i\,p^i$ has length at most $\ell + N$. Using the second point, we deduce the length bound, and the upper bound $\mathcal{O}(N\,(\ell+N))$ on the time it takes to compute the sum. If $N \leqslant \ell$, we are done. Else, we rewrite the sum as $\sum_{j=0}^{\ell-1} b_j\,p^j$, where b_j is the p-adic of length N whose coefficients are the coefficients of index j of $a_1, \ldots, a_N$. Thus, we have reversed the roles of ℓ and N, so the claim is valid in all cases. □

For multiplication, we will distinguish several variants; for the moment, we will simply define the problems, and introduce notation for their complexity.

First, we consider "plain" multiplication: given a and b of length at most n, compute their product (which has length at most $2n$). For this operation, we will let $\mathsf{I}\colon \mathbb{N}\to\mathbb{N}$ be such that all coefficients of $a\,b$ can be computed in $\mathsf{I}(n)$ operations. We will assume that $\mathsf{I}(n)$ satisfies the property that $\mathsf{I}(n)/n$ is non-decreasing. Using Fast Fourier Transform, it is possible to take $\mathsf{I}(n)$ *quasi-linear*, that is, linear up to logarithmic factors: we will review this in the next section.

Two related problems will be of interest: *short* and *middle* products. The short product at precision n is essentially the product of p-adics modulo p^n; precisely, on input a and b with $\max(\lambda(a), \lambda(b)) = n$, it computes the coefficients of

$$\mathrm{SP}(a,b) \ := \sum_{0\leqslant i+j<n} a_i\,b_j\,p^{i+j}.$$

The definition of the middle product is slightly more complex: if a and b are p-adics with $\lambda(b)=n$, the middle product of a and b is defined as (essentially) the middle part of the product $c := a\,b$; precisely, it computes

$$\mathrm{MP}(a,b) \ := \sum_{n-1\leqslant i+j\leqslant 2n-2} a_i\,b_j\,p^{i+j}.$$

In general, attention must be paid to carries: because of them, $\mathrm{MP}(a,b)$ may not consist in exactly the middle coefficients of $a\,b$. In the case where $R=\Bbbk[X]$, though, middle and short products simply compute a few of the coefficients of the product $a\,b$, so they can be computed by means of "plain" multiplication algorithms. We will see below that savings are possible: Section 1.2 gives algorithms for short and middle products, with a focus on the important particular case where $R=\Bbbk[X]$.

1.1.3 On-line and relaxed algorithms

Next, we introduce the "relaxed" model of computation for p-adics. Although this terminology is recent and was introduced in [Hoe02], it bears upon older and more general notions of lazy and on-line algorithms.

To the best of our knowledge, the notion of on-line Turing machine comes from [Hen66]. We give the definition formulated in [FS74].

Definition 1.2. ([Hen66, FS74]) *Let us consider a Turing machine which computes a function f on sequences, where $f\colon \Sigma^* \times \Sigma^* \rightarrow \Delta^*$, Σ and Δ are sets. The machine is said to compute f* on-line *if for all input sequences $a = a_0 a_1 \ldots a_n$, $b = b_0 b_1 \ldots b_n$ and corresponding outputs $f(a, b) = c_0 c_1 \ldots c_n$, with $a_i, b_j \in \Sigma$, $c_k \in \Delta$, it produces c_k before reading either a_j or b_j for $0 \leqslant k < j \leqslant n$.*

The machine computes f half-line *(with respect to the first argument) if it produces c_k before reading a_j for $0 \leqslant k < j \leqslant n$. The input a will be referred to as the* on-line argument *and b as the* off-line argument*.*

This definition can easily be adapted to more inputs and outputs by changing the sets Σ and Δ.

Lazy algorithms for power series are the adaptation of the *lazy evaluation* (also known as call-by-need) function evaluation scheme to computer algebra [Kar97], whose principle is to delay as much as possible the evaluation of the argument of a function. In the lazy approach, power series are represented as streams of coefficients, and the expressions they are involved in are evaluated as soon as the needed coefficients are provided; for this reason, algorithms in the lazy framework are on-line algorithms. Therefore, we will use the following informal definition.

Definition 1.3. Lazy algorithms *are on-line algorithms that try to minimize the cost at each step.*

Relaxed algorithms are also on-line algorithms. In opposition to lazy algorithms, they can do more computations at some step in order to anticipate future computations. Therefore we will use the following informal definition.

Definition 1.4. Relaxed algorithms *are on-line algorithms that try to minimize the asymptotic cost.*

Semi-relaxed algorithms are the counterpart of half-line algorithms. Although these notions were introduced for power series at first, they are easy to extend to any p-adic ring R_p.

The next chapters of this thesis present a fundamental application of relaxed algorithms, the computation of *recursive* p-adics. Meanwhile, we give some examples of on-line algorithms for two basic operations, sum and product. They are both based on an incremental process, which outputs one coefficient at a time.

Example 1.5. The first example of an on-line algorithm is the addition of p-adics. For computing the addition of p-adics a and b, we use a subroutine that takes as input another $c \in R_p$ that stores the current state of the computation and an integer i for the step of the computation we are at.

Algorithm `LazyAddStep`

Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$
Output: $c \in R_p$

1. $c = c + (a_i + b_i)\, p^i$
2. **return** c

The addition algorithm itself follows:

Algorithm `LazyAdd`

Input: $a, b \in R_p$ and $n \in \mathbb{N}$
Output: $c \in R_p$ such that $c = (a + b) \bmod p^{n+1}$

1. $c = 0$
2. **for** i from 0 to n
   a. $c = \texttt{LazyAddStep}(a, b, c, i)$
3. **return** c

This addition algorithm is on-line: it outputs the coefficient c_i of the addition $c = a + b$ without using any a_j or b_j of index $j > i$. After each step i, c represents the sum of $a \bmod p^{i+1}$ and $b \bmod p^{i+1}$; thus, it has length at most $i + 2$. As a result, at every step, we are simply computing $a_i + b_i + c_i$ (which we know has length at most 2), and insert the result in c_i and possibly c_{i+1}.

This algorithm is also lazy as it does only the minimal number of arithmetic operations at each step. Algorithm `LazyAdd` is relaxed because it does $\mathcal{O}(n)$ additions of length-1 p-adics, which is essentially optimal, to compute the addition of two p-adics at precision n. One can write an algorithm `LazySub` similarly.

Example 1.6. Let us next present the naive on-line algorithm for multiplication of p-adics.

Algorithm `LazyMulStep`

Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$
Output: $c \in R_p$

1. $c = c + \left(\sum_{j=0}^{i} a_j b_{i-j}\right) p^i$
2. **return** c

Algorithm `LazyMul`

Input: $a, b \in R_p$ and $n \in \mathbb{N}$
Output: $c \in R_p$ such that $c = (a\,b) \bmod p^{n+1}$

1. $c = 0$
2. **for** i from 0 to n
   a. $c = \texttt{LazyMulStep}(a, b, c, i)$
3. **return** c

Algorithm `LazyMul` is on-line because it outputs c_i without reading the coefficients a_j and b_j of the inputs for $j > i$. It is a lazy algorithm because it computes no more that $(a\,b)_i$ at step i. It allows the multiplication of two p-adics at precision n at cost $\mathcal{O}(n^2)$.

However, the cost of Algorithm `LazyMul` is prohibitive compared to the quasi-linear algorithms for the multiplication of high-order p-adics: this algorithm is not relaxed.

On the other hand, most fast algorithms for multiplication, such as those based on Fourier Transform, are not on-line. We remedy to this fact in Section 1.3 by presenting quasi-linear time on-line multiplication algorithms (which will thus be called *relaxed*).

1.2 Off-line multiplication

In this section, we review some existing off-line multiplication algorithms (for the plain, short and middle product), with a focus on the case where $R = \mathbb{k}[X]$. In the papers of van der Hoeven, off-line algorithms are also called *zealous* algorithms.

As customary, let us denote by $\mathsf{M}(n)$ a function such that over any ring, polynomials of degree at most $n-1$ can be multiplied in $\mathsf{M}(n)$ base operations, and such that $\mathsf{M}(n)/n$ is non-decreasing (super-linearity hypothesis, see [GG03, p. 242]). For the particular case of p-adics with ground ring $R = \mathbb{k}[X]$, we can thus take $\mathsf{I}(n) = \mathsf{M}(n)$.

In the first subsection, we review known results for the function M, followed by algorithms for the short and middle product. We briefly mention the case $R = \mathbb{Z}$, and then the general case, at the end of this section.

1.2.1 Plain multiplication of polynomials

We recall here the three main multiplication algorithms: the naive, Karatsuba's and the FFT algorithm. Given $a, b \in \mathbb{k}[X]_{<n}$ of degree less than n, we want to compute the product $c = a\,b \in \mathbb{k}[X]$ of a and b.

The naive algorithm computes the n^2 terms

$$c = \sum_{0 \leqslant i,j < n} a_i\, b_j\, X^{i+j}.$$

Therefore this algorithm performs n^2 multiplications and $(n-1)^2$ additions in $\mathbb{k}$.

The first subquadratic algorithm for multiplication was given by Karatsuba [KO63]. This algorithm starts by splitting the polynomial inputs in two halves:

$$a_{0\ldots n} = a_{0\ldots m} + a_{m\ldots n} X^m, \quad b_{0\ldots n} = b_{0\ldots m} + b_{m\ldots n} X^m$$

where $m := \lfloor n/2 \rfloor$. Then we compute three half-sized multiplications

$$d := a_{0\ldots m}\, b_{0\ldots m}, \quad e := (a_{0\ldots m} + a_{m\ldots n})\,(b_{0\ldots m} + b_{m\ldots n}), \quad f := a_{m\ldots n}\, b_{m\ldots n}.$$

Finally we recombine linearly these products to get

$$c := d + (e - d - f)\, X^m + f\, X^{2m}.$$

Therefore if $\mathsf{K}(n)$ denotes the cost of Karatsuba's multiplication algorithm for polynomials of degree less than n, one has

$$\mathsf{K}(n) = \mathsf{K}(\lfloor n/2 \rfloor) + 2\,\mathsf{K}(\lceil n/2 \rceil) + \mathcal{O}(n)$$

leading to $\mathsf{K}(n) = \mathcal{O}(n^{\log_2(3)})$.

The principle of Karatsuba's algorithm is related to an evaluation/interpolation at points $0, 1$ and $+\infty$. More general evaluation/interpolation schemes can be found in the algorithms of Toom-Cook [Too63, Coo66]. For any $\alpha > 1$, there exists a Toom-Cook algorithm that runs in time $\mathcal{O}(n^{\alpha})$.

The paper of Cooley and Tukey [CT65] founded the area of multiplication algorithms based on Fourier transforms. Let us describe the fast Fourier transform (FFT) algorithm, over a field $\mathbb{k}$. Let $m := 2^e$ be the smallest power of two greater or equal to $2\,n$. We start by assuming that there exists a mth primitive root of unity ω in $\mathbb{k}$. The discrete Fourier transform is the $\mathbb{k}$-linear isomorphism

$$\begin{array}{rccc} \mathrm{DFT}_{\omega}\colon & \mathbb{k}^m & \longrightarrow & \mathbb{k}^m \\ & (p_0, \ldots, p_{m-1}) & \longmapsto & (P(1), P(\omega), \ldots, P(\omega^{m-1})) \end{array}$$

where $P := \sum_{i=0}^{m-1} p_i\, X^i$. So DFT induces a bijection between $\mathbb{k}[X]_{<m}$ and $\mathbb{k}^m$. This transformation gives a new representation $(P(1), P(\omega), \ldots, P(\omega^{m-1}))$ of the polynomial P. The important fact is that the multiplication in $\mathbb{k}^m$ costs m multiplications, which is optimal.

Let us focus of the computation of DFT_{ω} and its inverse morphism $\mathrm{DFT}_{\omega}^{-1}$. A first important result is that

$$\mathrm{DFT}_{\omega} \circ \mathrm{DFT}_{\omega^{-1}} = \mathrm{DFT}_{\omega^{-1}} \circ \mathrm{DFT}_{\omega} = m\,\mathrm{Id}$$

and consequently

$$(\mathrm{DFT}_{\omega})^{-1} = \frac{1}{m}\,\mathrm{DFT}_{\omega^{-1}}.$$

It remains to give a fast algorithm to compute the discrete Fourier transform. A divide-and-conquer strategy is used. Write $P = P_0 + P_1\, X^{m/2}$ and let $R_0 = P_0 + P_1 \in \mathbb{k}[X]_{m/2}$ and $R_1(X) = (P_0 - P_1)(\omega\, X) \in \mathbb{k}[X]_{m/2}$. Then for $0 \leqslant i < m/2$, one has

$$\begin{array}{rcl} P(\omega^{2i}) & = & P_0(\omega^{2i}) + P_1(\omega^{2i}) = R_0(\omega^{2i}) \\ P(\omega^{2i+1}) & = & P_0(\omega^{2i+1}) - P_1(\omega^{2i+1}) = R_1(\omega^{2i}). \end{array}$$

Therefore the computation of $\mathrm{DFT}_{\omega}(P)$ reduces to two calls $\mathrm{DFT}_{\omega^2}(R_0)$ and $\mathrm{DFT}_{\omega^2}(R_1)$ and $\mathcal{O}(n)$ additions and multiplications. If $\mathsf{FFT}(m)$ is the cost of computing the discrete Fourier transform DFT_{ω}, then

$$\mathsf{FFT}(m) = 2\,\mathsf{FFT}(m/2) + \mathcal{O}(m)$$

which gives $\mathsf{FFT}(m) = \mathcal{O}(m \log(m))$. Finally the cost of multiplying two polynomials of degree less than n is $3\,\mathsf{FFT}(m) + \mathcal{O}(m) = \mathcal{O}(n \log(n))$.

When no roots of unity of sufficient order are available in the base field, we use the idea developed in [SS71] and [CK91]. This algorithm adds virtual roots of unity to our base field; it actually applies to any ring and multiplies polynomials of degrees less than n in time $\mathcal{O}(n \log(n) \log(\log(n)))$.

Let us now sum up all these algorithms.

Theorem 1.7. *One can take* $\mathsf{M}(n) \in \mathcal{O}(n \log(n) \log(\log(n)))$, *and thus* $\mathsf{I}(n) \in \mathcal{O}(n \log(n) \log(\log(n)))$ *when* $R = \mathbb{k}[X]$ *and* $p = X$.

1.2.2 Middle product of polynomials

The concept of middle product was introduced in [HQZ04]. That paper stresses the importance of this new operation in computer algebra and uses it to speed-up the division and square-root of power series.

Let $a, b \in \mathbb{k}[X]$ with $\lambda(b) = n$. Then, we have seen that the *middle product* $\mathrm{MP}(a, b)$ of a and b is defined as the part $c_{n-1\ldots 2n-1}$ of the product $c := a\, b$, so that $\deg(\mathrm{MP}(a, b)) \leqslant n - 1$. Naively, the middle product is computed via the full multiplication $c := (a\, b) \bmod p^{2n-1}$, which is done in time $2\,\mathsf{M}(n)$, but, as we will see, this not optimal. We denote by $\mathsf{MP}(n)$ the arithmetic complexity of the middle product of two polynomials a, b with $\lambda(b) \leqslant n$.

The middle product is closely related to the *transposed multiplication* [BLS03, HQZ04]. Thus, we will use the *Tellegen principle* that relates the complexities of a linear algorithm and its transposed algorithm. A linear algorithm can be formalized by linear straight-line programs (s.l.p.), which are s.l.p.'s with only linear operations. We refer to [BCS97, Chapter 13] for a precise exposition.

Theorem 1.8. ([BCS97, Th. 13.20]) *Let* $\Phi: R^n \to R^m$ *be a linear map that can be computed by a linear straight-line program of size* L *and whose matrix in the canonical bases has no zero rows or columns. Then the transposed map* Φ^t *can be computed by a linear straight-line program of size* $L - n + m$.

As it turns out, the middle product is a transposed multiplication, up to the reversal of polynomial. We deduce the following complexity result.

Corollary 1.9. *The complexity* MP *of the middle product satisfies*

$$\mathsf{MP}(n) = \mathsf{M}(n) + n - 1.$$

More precisely, while the number of additions can slightly differ between the multiplication and the middle product, the number of multiplications remains the same [HQZ04, Theorem 4].

Tellegen's principle gives more than the existence of a middle product algorithm with good complexity, it tells you how to build the transposed algorithm. It was first pointed out in [BLS03] that the transposition of algorithms can be done systematically and automatically (and the paper [DFS10] actually specifies an algorithm for automatic transposition based on the language *transalpyne*). We give a brief description of the middle product mechanisms corresponding to the transposition of the naive, Karatsuba and FFT multiplication algorithms.

Let us begin by a diagram. If we represent the polynomial coefficients $(a_i)_{0\leqslant i<2n-1}$ of a in abscissa and the coefficients $(b_j)_{0\leqslant j<n}$ of b in ordinate, the unit square whose left bottom corner is at coordinates $(i,\ j)$ corresponds to the elementary product $a_i\ b_j$. The big white square includes all the terms involved in the plain multiplication $a\ b$. The terms involved in the middle product $\mathrm{MP}(a,$ $b)$ form a gray rhombus on the diagram.

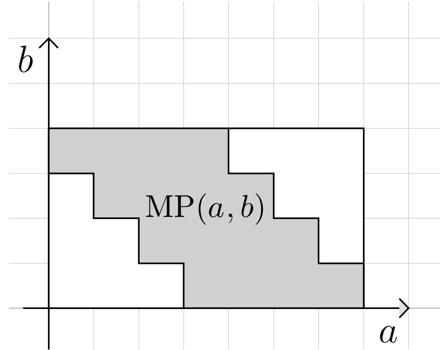

Figure 1.1. Plain and middle multiplication of polynomials

The naive multiplication algorithm gives the easiest scheme for middle product: only compute the coefficients $c_i=\sum_{j=0}^{n-1} a_{i-j}\,b_j$ of c for $n-1\leqslant i<2\,n-1$. This costs n^2 multiplications and $n\,(n-1)$ additions. Of course, the number of multiplications is the same as for the multiplication and the difference in the number of additions is predicted by Corollary 1.9. Indeed the difference of the number of additions between middle product and multiplication is $n\,(n-1)=(n-1)^2+n-1$.

Next, we sketch the Karatsuba middle product in the case of even length $n=\lambda(b)$.

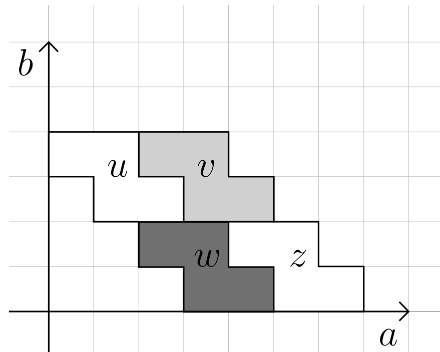

Figure 1.2. Karatsuba middle product on polynomials

The trick is to divide the diamond-shaped area of the middle product $\mathrm{MP}(a,b)$ into four parts u,v,w and z as depicted in Figure 1.2, that is

$$\begin{aligned}u &:= \mathrm{MP}(A_0,B_1)\\ v &:= \mathrm{MP}(A_1,B_1)\\ w &:= \mathrm{MP}(A_1,B_0)\\ z &:= \mathrm{MP}(A_2,B_0)\end{aligned}$$

where $A_0 := a_{0\ldots n-1}$, $A_1 := a_{n/2\ldots 3n/2-1}$, $A_2 := a_{n\ldots 2n-1}$, $B_0 := b_{0\ldots n/2}$ and $B_1 := b_{n/2\ldots n}$. Then we observe that by bilinearity $u + v = \mathrm{MP}(A_0 + A_1, B_1)$, $v - w = \mathrm{MP}(A_1, B_1 - B_0)$ and $w + z = \mathrm{MP}(A_1 + A_2, B_0)$. Therefore we get

$$\begin{aligned} \mathrm{MP}(a,b)_{0\ldots n/2} &= (u+v) - (v-w) \\ \mathrm{MP}(a,b)_{n/2\ldots n} &= (w+z) + (v-w). \end{aligned}$$

So we have reduced the problem of Karatsuba middle product to three half-sized recursive calls and a few additions. The case of odd length n is similar but somewhat more complicated. This algorithm is the transposed algorithm of Karatsuba's multiplication [HQZ04, BLS03].

For the FFT variant, and suppose that $\omega \in \Bbbk$ is a primitive $(2\,n-1)$th root of unity. We cut the product $c = a\,b$ in three parts $c_{0\ldots n-1}$, $c_{n-1\ldots 2n-1}$ and $c_{2n-1\ldots 3n-2}$ and remark that

$$(c_{0\ldots n-1} + c_{2n-1\ldots 3n-2}) + X^{n-1}\, c_{n-1\ldots 2n-1} = c \text{ rem } (X^{2n-1} - 1). \tag{1.1}$$

Consequently given

$$\begin{aligned} a(1), \ldots, a(\omega^{2n-2}) &:= \mathrm{FFT}(a, \omega) \\ b(1), \ldots, b(\omega^{2n-2}) &:= \mathrm{FFT}(b, \omega) \end{aligned}$$

we reconstruct $e := c \text{ rem } (X^{2n-1} - 1)$ by $e = \frac{1}{2\,n-1}\,\mathrm{FFT}\big(\big(\sum_{i=0}^{2n-2} a(\omega^i)\, b(\omega^i)\, X^i\big), \omega^{-1}\big)$. So finally $\mathrm{MP}(a,b) = e_{n-1\ldots 2n-1}$. In practice, we only work with 2^ℓth root of unity and a padding with zeroes is necessary to adjust Formula (1.1).

1.2.3 Short product of polynomials

We denote by $\Bbbk[X]_{<n}$ the set of polynomials of length lesser or equal to n. Let a, $b \in \Bbbk[X]_{<n}$ and define the *short product* $\mathrm{SP}(a,b)$ of a and b as the part $c_{0\ldots n}$ of the product $c := a\,b$. In other words, $c = (a\,b) \bmod X^n$. We denote by $\mathsf{SP}(n)$ the cost of the short product of $a, b \in \Bbbk[X]_{<n}$ and C_{SP} the ratio with plain multiplication, *i.e.* a constant such that $\mathsf{SP}(n) \leqslant C_{\mathsf{SP}}\,\mathsf{M}(n)$ for all $n \in \mathbb{N}^*$.

The situation with the short product is more contrasted than for the middle product. Although the size of the output is halved, we seldom gain a factor 2 in the cost: the actual cost of the short product is hard to pin down.

As always, it is easy to adapt the naive multiplication algorithm to compute only the first terms. In this case, we gain a factor two in the complexity, *i.e.* $C_{\mathsf{SP}} = 1/2$.

Let us now consider Karatsuba's multiplication. The paper [Mul00] published the first approach for having $C_{\mathsf{SP}} < 1$ for the cost function $\mathsf{M}(n) = n^{\log_2(3)}$, which is an approximation of the cost of Karatsuba's multiplication. The basic idea is to do two half-sized recursive calls and use a half-sized multiplication, but this does not improve the complexity. A refinement of previous idea consists in changing the size of the cutting of the problem and optimizing the complexity with respect to this size. It reaches a constant $C_{\mathsf{SP}} = 0.81$ for this approximated cost function. However, practical application of this method to Karatsuba's algorithm shows that the value 0.81 is not a upper bound of the ratio of timings but rather an estimation of its average.

The analysis of the ratio $\mathsf{SP}(n)/\mathsf{M}(n)$ for $\mathsf{M}(n)$ the exact number of multiplications of Karatsuba's algorithm is done [HZ04]. They find the optimal integer cutting for Karatsuba's short product and prove that $C_{\mathsf{SP}}=1$ is the best upper bound.

However, the situation is different if we consider an hybrid multiplication algorithm that uses the naive, quadratic algorithm for small values and switches to Karatsuba's method for larger values. In this case, another variant based on odd/even decomposition [HZ04] performs well. This variant does three half-sized recursive calls and, intuitively, transfers the factor $C_{\mathsf{SP}}=1/2$ attained by the naive methods to Karatsuba's. The paper show that, for a threshold $n_0 = 32$ between algorithm, one has $\mathsf{SP}(n) \leqslant 0.57\,\mathsf{M}(n)$ for $n > n_0$. The timings of the implementation of [HZ04] show that this factor $C_{\mathsf{SP}} = 0.6$ is also observed in practice in the degree range of the Karatsuba multiplication.

No improvement is known for the short product based on FFT multiplication. However, notice that we can compute $c_{0\ldots n} + c_{n\ldots 2n-1}$ in time $1/2\,\mathsf{M}(n)$ because it equals to c modulo $(X^n - 1)$. We will use this fact in Section 1.3.5 on middle relaxed multiplication.

1.2.4 The situation on integers

The presence of carries when computing with integers complicates the situation for all operations. Surprisingly, though, it is possible to obtain a slightly faster plain multiplication than in the polynomial case. We denote by $\log^*: \mathbb{R}_{>0} \to \mathbb{R}$ the iterated logarithm defined recursively by

$$\log^*(x) = \begin{cases} 0 & \text{if } 0 < x \leqslant 1 \\ 1 + \log^*(\log(x)) & \text{if } 1 < x \end{cases}.$$

Theorem 1.10. ([Für07, DKSS08]) *Two integers with n digits in base p can be multiplied in bit-complexity $\mathcal{O}(n \log(n)\, 2^{\log^*(n)})$.*

Note that this result involves a different complexity model than ours. It seems that the ideas of [DKSS08] could be adapted to give the same result in our complexity model.

As to middle and short product, few algorithms exist. Indeed, in the integer case, we face two kinds of problems, both due to carries. First, the middle and short product can have more than n coefficients. Moreover, the middle product $\mathrm{MP}(f, g)$ can no longer be seen as the middle part of their product.

As always, the naive algorithm adapts well for middle and short product of integers. The problems due to carries are solved for the Karatsuba middle product for integers in [Har12]. About the Karatsuba short product, we quote [HZ04]: "the carries are a simple matter to deal with in Mulders' method but are a real problem with our (odd/even) variant". Finally, it seems that the FFT middle product can be adapted to integers. Indeed if the middle product is not exactly the middle part of the multiplication, the difference concerns only a few of the lower and higher coefficients. Computing $a\,b$ modulo p^{2n-1}, we get most of the coefficients of the middle product and compute the missing coefficients in linear time.

We leave it as a future work to implement these methods and to assess their effect on the complexity of the relaxed multiplication of p-adic integers.

1.2.5 The situation on p-adics

Finally, we prove by a simple reduction that for any p-adic ring R_p, the cost function of off-line p-adic multiplication is always quasi-linear.

Theorem 1.11. *For any p-adic ring R_p, the cost $\mathsf{I}(n)$ of multiplication of p-adics of size n is bounded by $\mathcal{O}(\mathsf{M}(n)\log(n)^2)$.*

Proof. Let $a=\sum_{i=0}^{n-1} a_i\, p^i$ and $b=\sum_{i=0}^{n-1} b_i\, p^i$ be p-adics of length bounded by n. Introduce the polynomials $A=\sum_{i=0}^{n-1} a_i\, X^i$ and $B=\sum_{i=0}^{n-1} b_i\, X^i$ of $R[X]$ and let $C=\sum_{i=0}^{2n-2} c_i\, X^i \in R[X]$ be the product of A and B.

Since the length of the coefficients c_i of C is bounded by $\ell_c := \lceil \log_2(n) \rceil + 2$, we can multiply the polynomials A and B in the ring $(R/(p^{\ell_c}))[X]$ and recover C. Arithmetic operations in $(R/(p^{\ell_c}))$ can be computed in time $\mathcal{O}(\mathsf{I}(\log(n)))$, so we obtain C at cost $\mathcal{O}(\mathsf{I}(\log(n))\,\mathsf{M}(n))$; taking the naive bound $\mathsf{I}(n)=\mathcal{O}(n^2)$, we get the claimed cost $\mathcal{O}(\mathsf{M}(n)\log(n)^2)$.

Finally the p-adic $c:=a\,b$ equals to $C(p)$. In view of the third point in Lemma 1.1, the cost of the additions necessary to compute $C(p)$ is $\mathcal{O}(n\log(n))$. □

1.3 Relaxed algorithms for multiplication

In this section, we recall several relaxed algorithms for the on-line multiplication of p-adics, we analyze precisely their costs and give timings of our implementation using NTL. To our knowledge, no such precise comparison existed before.

Besides, we introduce a *new* relaxed multiplication algorithm using middle and short product, and show that it can perform better than some previous ones.

We start by recalling the state-of-the-art of on-line p-adic multiplication.

Theorem 1.12. ([FS74, Hoe97, BHL11]) *Whenever R_p is a power series ring or the ring of p-adic integers, the cost $\mathsf{R}(n)$ of multiplying two p-adics at precision n by an on-line algorithm is*

$$\mathcal{O}\left(\sum_{k=0}^{\lfloor \log_2(n) \rfloor} \frac{n}{2^k}\,\mathsf{I}(2^k)\right) = \begin{cases} \mathcal{O}(\mathsf{I}(n)) & \text{for naïve or Karatsuba's multiplication} \\ \mathcal{O}(\mathsf{I}(n)\log(n)) & \text{for FFT multiplication} \end{cases}.$$

The previous result was first discovered for integers in [FS74]; the details for the multiplication of power series were given in [Hoe97] and the paper [BHL11] generalizes relaxed algorithms for p-adic integers. The latter algorithm is correct for any p-adic ring but the authors analyze the complexity only for the p-adic integers. The issue with general p-adic rings is the management of carries. Although we do not prove it here, we believe that this complexity result carries forward to any p-adic ring.

Remark 1.13. Recent progress has been made on relaxed multiplication [Hoe07, Hoe12]. These papers give an on-line algorithm that multiplies power series on a wide range of rings, including all fields, in time

$$\mathsf{M}(n)\log(n)^{o(1)}.$$

Also, on-line multiplication of p-adic integers at precision n can be done in bit complexity

$$n \log(n)^{1+o(1)} \log(p) \log(\log(p)).$$

We will not give the details of these algorithms here.

In the next subsections, we will give a short presentation of the relaxed product algorithms that reach the bound of Theorem 1.12. Existing algorithms can be found in Sections 1.3.2, 1.3.3 and 1.3.4. Our new relaxed multiplication algorithm using short and middle product is presented in Section 1.3.5.

Although the algorithms are correct for any p-adic ring R_p, we will analyze their cost in the special case of power series rings: the exposition will be simplified since there are no carries.

To establish comparisons, and for the sake of completeness, we give for the first time *the constants hidden in the big-O notation* of the complexity estimates. All the following complexity analyses take into account only the number of basic multiplications, and do not count the basic additions. For the rest of this chapter, the *multiplicative complexity* of an algorithm is the number of basic multiplications it performs. We denote by M^* the multiplicative complexity function of polynomial multiplication. We sum up these bounds in the next two tables.

Table 1.1 gives bounds on the multiplicative complexity of *semi-relaxed* multiplication algorithms depending on the algorithm we use to multiply truncated power series (naive, Karatsuba or FFT).

The semi-relaxed multiplication algorithm which appears in [Hoe07] gives the costs of the first line; we give an overview of this algorithm in Section 1.3.2. The second line corresponds to the semi-relaxed algorithm using middle product presented in [Hoe03], which can be found in Section 1.3.3.

	naive	Karatsuba	FFT
semi-relaxed	$\leqslant 2\,\mathsf{M}^*(n)$	$\leqslant 3\,\mathsf{M}^*(n)$	$\sim \frac{1}{2}\,\mathsf{M}^*(n)\log_2(n)$
semi-relaxed with middle	$\leqslant 1.5\,\mathsf{M}^*(n)$	$\leqslant 2\,\mathsf{M}^*(n)$	$\sim \frac{1}{4}\,\mathsf{M}^*(n)\log_2(n)$

Table 1.1. Multiplicative complexity of the semi-relaxed multiplication of power series

Table 1.2 describes relaxed algorithms. The first line of Table 1.2 corresponds to the relaxed multiplication algorithm of [FS74, Hoe97, BHL11]. This algorithm is presented in Section 1.3.4. Our contribution, the relaxed multiplication using middle and short product, gives the results of the second line. It is presented in Section 1.3.5.

	naive	Karatsuba	FFT
relaxed	$\leqslant \mathsf{M}^*(n+1)$	$\leqslant 2.5\,\mathsf{M}^*(n+1)$	$\sim \mathsf{M}^*(n)\log_2(n)$
relaxed with short and middle	$\leqslant \mathsf{M}^*(n+1)$	$\leqslant \begin{cases} 1.75\,\mathsf{M}^*(n+1) & \text{if } C_{\mathsf{SP}} = \frac{1}{2} \\ 2.5\,\mathsf{M}^*(n+1) & \text{if } C_{\mathsf{SP}} = 1 \end{cases}$	$\sim \frac{1}{2}\,\mathsf{M}^*(n)\log_2(n)$

Table 1.2. Multiplicative complexity of the relaxed multiplication of power series

Remark 1.14. It was remarked in [Hoe97, Hoe02] that the Karatsuba multiplication could be rewritten as a relaxed algorithm, thus leading to a relaxed multiplication algorithm with exactly the same numbers of operations.

However, this algorithm is often not practical. The rewriting induces $\Omega(\log(n))$ function calls at each step of the multiplication, which makes it very poorly suited to practical implementations. For these reasons, we will not study this algorithm.

Remark 1.15. When the required precision n is known in advance, it is possible to adapt the on-line multiplication algorithms to this specific precision and thus lower the bounds given in Tables 1.1 and 1.2 (see [Hoe02, Hoe03]). An example of such an algorithm is given in the paragraph "Link between divide-and-conquer and semi-relaxed" in Section 1.3.3. However, these considerations are not developed further in this thesis.

We choose to present a simple form of the relaxed product algorithms, which will be convenient to understand the operations made in the computation of recursive p-adics in the next chapter. We allocate ourselves the memory to store the current state of the computation and we indicate to the program at which step we are. If one were to implement these algorithms, our description would not be appropriate. We would recommend the implementation described in [Hoe02, BHL11], which is actually very close to the implementation in MATHEMAGIX [HLM+02].

1.3.1 Complexity preliminaries

We introduce three auxiliary complexity functions from $\mathbb{N}$ to $\mathbb{N}$,

$$\begin{aligned} \mathsf{M}^{(1)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \mathsf{M}^*(2^k) \\ \mathsf{M}^{(2)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^k} \right\rfloor \mathsf{M}^*(2^k) \\ \mathsf{M}^{(3)}(n) &:= \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{n}{2^{(k+1)}} + \frac{1}{2} \right\rfloor \mathsf{M}^*(2^k). \end{aligned}$$

These functions will be used afterwards to assess the multiplicative complexity of our (semi-)relaxed multiplication algorithms.

Lemma 1.16. *Let* $\ell := \lfloor \log_2(n) \rfloor$ *and* $n = \sum_{i=0}^{\ell} \bar{n_i}\, 2^i$ *be the base-2 expansion of* n. *Assume that* $\mathsf{M}^*(1) = 1$ *and that there exists* $\alpha \in]1; +\infty[$ *such that for all* $n \in \mathbb{N}$, $\mathsf{M}^*(2\,n) = 2^\alpha\, \mathsf{M}^*(n)$. *Then*

$$\begin{aligned} \mathsf{M}^{(1)}(n) &:= \frac{2^\alpha}{2^\alpha - 1} \mathsf{M}^*(2^\ell) - \frac{1}{2^\alpha - 1} \\ \mathsf{M}^{(2)}(n) &:= \frac{2^\alpha}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{2\,n}{2^\alpha - 2} \\ \mathsf{M}^{(3)}(n) &:= \frac{2^\alpha - 1}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{n}{2^\alpha - 2}. \end{aligned}$$

Proof. First

$$\mathsf{M}^{(1)}(n) = \sum_{k=0}^{\ell} 2^{\alpha k} = \frac{2^{\alpha(\ell+1)} - 1}{2^{\alpha} - 1} = \frac{2^{\alpha}}{2^{\alpha} - 1} \mathsf{M}^*(2^{\ell}) - \frac{1}{2^{\alpha} - 1}.$$

Next, one has

$$\begin{aligned}
\mathsf{M}^{(2)}(n) &= \sum_{k=0}^{\ell} \sum_{i=k}^{\ell} \bar{n_i}\, 2^{i-k}\, \mathsf{M}^*(2^k) \\
&= \sum_{i=0}^{\ell} \bar{n_i}\, 2^i \sum_{k=0}^{i} 2^{(\alpha-1)k} \\
&= \sum_{i=0}^{\ell} \bar{n_i}\, 2^i \frac{2^{(\alpha-1)(i+1)} - 1}{2^{(\alpha-1)} - 1} \\
&= \frac{2^{\alpha}}{2^{\alpha} - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{2\,n}{2^{\alpha} - 2}.
\end{aligned}$$

Finally, we have the equalities

$$\begin{aligned}
\mathsf{M}^{(3)}(n) &= \sum_{k=0}^{\ell} \left(\left\lfloor \frac{n}{2^{(k+1)}} \right\rfloor + \bar{n_k} \right) \mathsf{M}^*(2^k) \\
&= \sum_{k=0}^{\ell} \sum_{i=k+1}^{\ell} \bar{n_i}\, 2^{i-(k+1)}\, \mathsf{M}^*(2^k) + \sum_{k=0}^{\ell} \bar{n_k}\, \mathsf{M}^*(2^k) \\
&= \sum_{i=1}^{\ell} \bar{n_i}\, 2^{i-1} \sum_{k=0}^{i-1} 2^{(\alpha-1)k} + \sum_{k=0}^{\ell} \bar{n_k}\, \mathsf{M}^*(2^k) \\
&= \sum_{i=1}^{\ell} \bar{n_i}\, 2^{i-1} \frac{2^{(\alpha-1)i} - 1}{2^{(\alpha-1)} - 1} + \sum_{k=0}^{\ell} \bar{n_k}\, \mathsf{M}^*(2^k) \\
&= \left(\frac{1}{2^{\alpha} - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{n}{2^{\alpha} - 2} \right) + \sum_{k=0}^{\ell} \bar{n_k}\, \mathsf{M}^*(2^k) \\
&= \frac{2^{\alpha} - 1}{2^{\alpha} - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{n}{2^{\alpha} - 2}.
\end{aligned}$$

□

Lemma 1.17. *Assume that* M^* *counts the number of multiplication of the naive or Karatsuba's algorithm. Let* $n = \sum_{i=0}^{\ell} \bar{n_i}\, 2^i$ *be the base-2 expansion of* n*. Then*

$$\mathsf{M}^*(2^{\ell}) + (\mathsf{M}^*(3) - \mathsf{M}^*(2)) \sum_{i=0}^{\ell-1} \bar{n_i}\, \mathsf{M}^*(2^i) \leqslant \mathsf{M}^*(n).$$

Proof. Under the same hypothesis on M^*, we start by proving that for any $n \geqslant 1$,

$$\mathsf{M}^*(2\,n) + (\mathsf{M}^*(3) - \mathsf{M}^*(2))\, \mathsf{M}^*(1) \leqslant \mathsf{M}^*(2\,n+1). \tag{1.2}$$

Let $C := (\mathsf{M}^*(3) - \mathsf{M}^*(2))$. For the naive multiplication algorithm, one has

$$C = 5 \leqslant \mathsf{M}^*(2\,n+1) - \mathsf{M}^*(2\,n) = 4\,n+1.$$

For Karatsuba's multiplication algorithm, we proceed as follows. Recall that the Karatsuba's cost function satisfies $\mathsf{M}^*(n) = 2\,\mathsf{M}^*(\lceil n/2\rceil) + \mathsf{M}^*(\lfloor n/2\rfloor)$. We have to prove the inequality

$$C = 4 \leqslant \mathsf{M}^*(2\,n+1) - \mathsf{M}^*(2\,n) = 2\,(\mathsf{M}^*(n+1) - \mathsf{M}^*(n)).$$

So we prove that for any $n \geqslant 1$, $\mathsf{M}^*(n+1) - \mathsf{M}^*(n) \geqslant 2$. First, $\mathsf{M}^*(2) - \mathsf{M}^*(1) = 2$ and $\mathsf{M}^*(3) - \mathsf{M}^*(2) \geqslant 2$. Then recursively, we assume that the result is true until $n \geqslant 2$ and prove it for $n+1$. We separate the odd and even cases. If $n+1 = 2\,k$, then $k \geqslant 1$ and

$$\mathsf{M}^*(n+2) - \mathsf{M}^*(n+1) = 2\,\mathsf{M}^*(k+1) + \mathsf{M}^*(k) - 3\,\mathsf{M}^*(k) = 2\,(\mathsf{M}^*(k+1) - \mathsf{M}^*(k)) \geqslant 4.$$

Else, if $n+1 = 2\,k+1$, then $k \geqslant 1$ and

$$\mathsf{M}^*(n+2) - \mathsf{M}^*(n+1) = 3\,\mathsf{M}^*(k+1) - (2\,\mathsf{M}^*(k+1) + \mathsf{M}^*(k)) = \mathsf{M}^*(k+1) - \mathsf{M}^*(k) \geqslant 2.$$

So Equation (1.2) is proved and we can prove the lemma. First,

$$\begin{aligned} \mathsf{M}^*(2^\ell) + C\,\bar{n}_{\ell-1}\,\mathsf{M}^*(2^{\ell-1}) &= 3^{\ell-1}\,(\mathsf{M}^*(2) + C\,\bar{n}_{\ell-1}\,\mathsf{M}^*(1)) \\ &\leqslant 3^{\ell-1}\,\mathsf{M}^*(2 + \bar{n}_{\ell-1}\cdot 1) \\ &= \mathsf{M}^*(2^\ell + \bar{n}_{\ell-1}\cdot 2^{\ell-1}). \end{aligned}$$

Then

$$\begin{aligned} \mathsf{M}^*(2^\ell) + C \sum_{i=\ell-2}^{\ell-1} \bar{n_i}\,\mathsf{M}^*(2^i) &\leqslant \mathsf{M}^*(2^\ell + \bar{n}_{\ell-1}\cdot 2^{\ell-1}) + C\,\bar{n}_{\ell-2}\,\mathsf{M}^*(2^{\ell-2}) \\ &= 3^{\ell-2}\,(\mathsf{M}^*(4 + \bar{n}_{\ell-1}\cdot 2) + C\,\bar{n}_{\ell-2}\,\mathsf{M}^*(1)) \\ &\leqslant 3^{\ell-2}\,\mathsf{M}^*(4 + \bar{n}_{\ell-1}\cdot 2 + \bar{n}_{\ell-2}\cdot 1) \\ &= \mathsf{M}^*(2^\ell + \bar{n}_{\ell-1}\cdot 2^{\ell-1} + \bar{n}_{\ell-2}\cdot 2^{\ell-2}). \end{aligned}$$

We repeat this process until we have

$$\mathsf{M}^*(2^\ell) + C \sum_{i=0}^{\ell-1} \bar{n_i}\,\mathsf{M}^*(2^i) \leqslant \mathsf{M}^*\left(\sum_{i=0}^{\ell} \bar{n_i}\,2^i\right) = \mathsf{M}^*(n). \qquad \square$$

Lemma 1.18. *If* $\mathsf{M}^*(n) = K\,n\,(\log_2 n)^i\,(\log_2\log_2 n)^j$ *with* $K \in \mathbb{R}_{>0}, (i, j) \in \mathbb{N}^2$, *one has*

$$\begin{aligned} \mathsf{M}^{(2)}(n) &\sim_{n\to\infty} \frac{1}{(i+1)}\,\mathsf{M}^*(n)\log_2(n) \\ \mathsf{M}^{(3)}(n) &\sim_{n\to\infty} \frac{1}{2\,(i+1)}\,\mathsf{M}^*(n)\log_2(n). \end{aligned}$$

Proof. We set the notation $\ell := \lfloor \log_2(n) \rfloor$. As M^* is a super-linear function, we get

$$\mathsf{M}^{(1)}(n) \leqslant \sum_{k=0}^{\ell} \frac{1}{2^{\ell-k}} \mathsf{M}^*((2^\ell/2^k)\, 2^k) \leqslant 2\, \mathsf{M}^*(n).$$

Also, one has

$$\mathsf{M}^{(2)}(n) = \sum_{k=0}^{\ell} \lfloor n/2^k \rfloor \, \mathsf{M}^*(2^k) = \sum_{k=0}^{\ell} (n/2^k)\, \mathsf{M}^*(2^k) + \mathcal{O}_{n\to\infty}\Big(\mathsf{M}^{(1)}(n)\Big).$$

Since $\mathsf{M}^{(1)}(n) = \mathcal{O}_{n\to\infty}(\mathsf{M}^*(n))$ and since one has for $n \to \infty$

$$\begin{aligned}\sum_{k=0}^{\ell} (n/2^k)\, \mathsf{M}^*(2^k) &\sim K \sum_{k=0}^{\ell} (n/2^k)\, 2^k k^i \log_2^j(k) \\ &\sim K n \left(\sum_{k=0}^{\ell} k^i \log_2^j(k) \right) \\ &\sim K n \left(\frac{\ell^{i+1}}{i+1} \log_2^j(\ell) \right)\end{aligned}$$

we deduce that $\mathsf{M}^{(2)}(n) \sim_{n\to\infty} \frac{1}{(i+1)} \mathsf{M}^*(n) \log_2(n)$. Finally, we deal with $\mathsf{M}^{(3)}$:

$$\mathsf{M}^{(3)}(n) = \sum_{k=0}^{\ell} \frac{n}{2^{k+1}} \mathsf{M}^*(2^k) + \mathcal{O}_{n\to\infty}\Big(\mathsf{M}^{(1)}(n)\Big) \sim_{n\to\infty} \frac{1}{2\,(i+1)} \mathsf{M}^*(n) \log_2(n). \qquad \square$$

1.3.2 Semi-relaxed multiplication

The forthcoming half-line algorithm for the multiplication of p-adics was introduced in [FS74, Hoe03]. We briefly recall its mechanism. To do the product of p-adics a and b, we use extra inputs $c \in R_p$ and $i \in \mathbb{N}$: the p-adic c stores the current state of the computation and the integer i indicates at which step we are. The `SemiRelaxedProductStep` algorithm requires multiplications between finite precision p-adics. Because the required coefficients of a and b are known at that moment, any multiplication algorithm that takes as input truncated p-adics can be used.

We denote by $\nu_2(n)$ the valuation in 2 of the integer n. We obtain the following algorithm of which a is the only on-line argument.

Algorithm `SemiRelaxedProductStep`
Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$ **Output:** $c \in R_p$ 1. **for** k from 0 to $\nu_2(i+1)$ a. $c = c + a_{i-2^k+1\ldots i+1}\, b_{2^k-1\ldots 2^{k+1}-1}\, p^i$ 2. **return** c

The diagram in Figure 1.3 will help us to understand the multiplications done in Algorithm `SemiRelaxedProductStep`. The coefficients $a_0, a_1, \ldots$ of a are placed in abscissa and the coefficients $b_0, b_1, \ldots$ of b in ordinate. Each unit square corresponds to a product between corresponding coefficients of a and b, *i.e.* the unit square whose left-bottom corner is at coordinates (i, j) stands for $a_i\, b_j$. Each bigger square corresponds to a product of finite precision p-adics; an $s \times s$ square whose left-bottom corner is at coordinates (i, j) stands for $a_{i \ldots i+s}\, b_{j \ldots j+s}$. The number inside the square indicates at which step this computation is done.

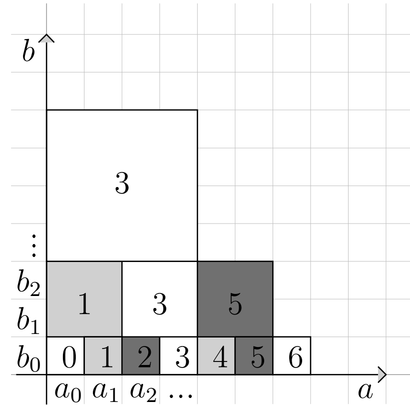

Figure 1.3. Semi-relaxed multiplication

We define two properties for any algorithm `Algo` with entries in $R_p^3 \times \mathbb{N}$ and output in R_p. These properties check that the algorithm computes progressively the product of the first two entries. The property $(\mathcal{HL})$ is the half-line variant and the property $(\mathcal{OL})$ is the on-line variant.

Property $(\mathcal{HL})$: For any $n \in \mathbb{N}$ and any a, b, $c_0 \in R_p$, the result $c \in R_p$ of the computation

Algorithm `Loop`$_{\texttt{Algo}}$
Input: $a, b, c_0 \in R_p$ and $n \in \mathbb{N}$ **Output:** $c \in R_p$ 1. $c = c_0$ 2. **for** i from 0 to n a. $c = \texttt{Algo}(a, b, c, i)$ 3. **return** c

satisfies $c = c_0 + a\, b$ modulo p^{n+1}. Moreover, during the computation, the Turing machine reads at most the coefficients $a_0, \ldots, a_n$ of the input a.

Property $(\mathcal{OL})$: Algorithm `Algo` must satisfy Property $(\mathcal{HL})$ and, additionally, read at most the coefficients $b_0, \ldots, b_n$ of the input b.

Property $(\mathcal{HL})$ states that the algorithm `Algo` is half-line and increments the number of correct p-adic coefficients of the product $c = a\,b$. This is the case for our algorithm `SemiRelaxedProductStep`.

Proposition 1.19. *Algorithm* `SemiRelaxedProductStep` *satisfies Property* $(\mathcal{HL})$.

We can check on Figure 1.3 that for all $n \in \mathbb{N}$, all the coefficients of the product $a\,b = \sum_{i=0}^{n} \sum_{j=0}^{i} a_j\, b_{i-j}\, p^i$ modulo p^{n+1} are computed by the semi-relaxed product before or at step n. We can also check that the algorithm is half-line in a since at step i, we use at most the coefficients $a_0, \ldots, a_i$ of a. However the operand b is off-line because, for example, it reads the coefficients $b_0, \ldots, b_6$ of b at step 3.

Complexity analysis As said before, we analyze the cost in the special case of R_p being a power series ring. For this reason, truncated p-adics are polynomials and their multiplication cost is denoted by $\mathsf{M}^*(n)$. For the sake of clarity, Algorithm $\texttt{Loop}_{\texttt{SemiRelaxedProductStep}}$ will also be called Algorithm `SemiRelaxedProduct`.

The cost $\mathsf{SR}^*(n)$ of all the off-line polynomial multiplications in the semi-relaxed algorithm `SemiRelaxedProduct` up to precision n (*i.e.* the terms in p^i for $0 \leqslant i < n$) is exactly $\mathsf{M}^{(2)}(n)$. Indeed, we do at each step a product of polynomials of degree 0 which each costs $\mathsf{M}^*(1) = 1$. We do every other step, starting for step 1, a product of polynomials of degree 1 which each costs $\mathsf{M}^*(2)$ and so on.

Proposition 1.20. *One has*

$$\mathsf{SR}^*(n) \leqslant \begin{cases} 2\,\mathsf{M}^*(n) & \textit{for the naive multiplication} \\ 3\,\mathsf{M}^*(n) & \textit{for Karatsuba's multiplication} \end{cases}$$

and these bound are asymptotically optimal since

$$\mathsf{SR}^*(2^m) \sim_{m\to\infty} \begin{cases} 2\,\mathsf{M}^*(2^m) & \textit{for the naive multiplication} \\ 3\,\mathsf{M}^*(2^m) & \textit{for Karatsuba's multiplication.} \end{cases}$$

Moreover when $\mathsf{M}^*(n) = K\,n \log_2(n) \log_2(\log_2(n))$ *with* $K \in \mathbb{R}_{>0}$, *one has*

$$\mathsf{SR}^*(n) \sim_{n\to\infty} \frac{1}{2}\,\mathsf{M}^*(n) \log_2(n).$$

Proof. Let us begin with the case where M^* is the cost function of the naive or Karatsuba's multiplication. Using Lemma 1.16 for the first equality and Lemma 1.17 for the second inequality, we have that for all $n \in \mathbb{N}$,

$$\begin{aligned} \mathsf{SR}^*(n) &= \frac{2^\alpha}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{2\,n}{2^\alpha - 2} \\ &\leqslant \frac{2^\alpha}{2^\alpha - 2}\, \mathsf{M}^*(n) + 0. \end{aligned}$$

When $n = 2^m$, one has

$$\mathsf{SR}^*(2^m) = \frac{2^\alpha}{2^\alpha - 2}\, \mathsf{M}^*(2^m) - \frac{2 \cdot 2^m}{2^\alpha - 2} \sim_{m\to\infty} \frac{2^\alpha}{2^\alpha - 2}\, \mathsf{M}^*(2^m).$$

At last, when $\mathsf{M}^*(n) = K\,n\log_2(n)\log_2(\log_2(n))$, we use Lemma 1.18 to obtain

$$\mathsf{SR}^*(n) \sim_{n\to\infty} \frac{1}{2}\,\mathsf{M}^*(n)\log_2(n). \qquad \square$$

This gives the entries of the first line in Table 1.1, keeping in mind that the cost of additions $\mathcal{O}(n\log(n))$ is omitted.

1.3.3 Semi-relaxed multiplication with middle product

Another semi-relaxed algorithm, using middle products, was introduced in [Hoe03]. Whereas the semi-relaxed product `SemiRelaxedProduct` used plain multiplication on truncated p-adics as a basic tool, middle products are used to compute incrementally the product $a\,b$. Naturally, the following algorithm is of interest when there exists efficient middle and short product algorithms, e.g. when $R_p = \mathbb{k}[[X]]$. This algorithm is on-line with respect to the input a.

Algorithm `SemiRelaxedProductMiddleStep`

Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$
Output: $c \in R_p$

1. Let $m := \nu_2(i+1)$
2. $c = c + \mathrm{MP}(a_{i-2^m+1\ldots i+1}, b_{0\ldots 2^{m+1}-1})\,p^i$
3. **return** c

The mechanism of the algorithm is sketched in Figure 1.4.

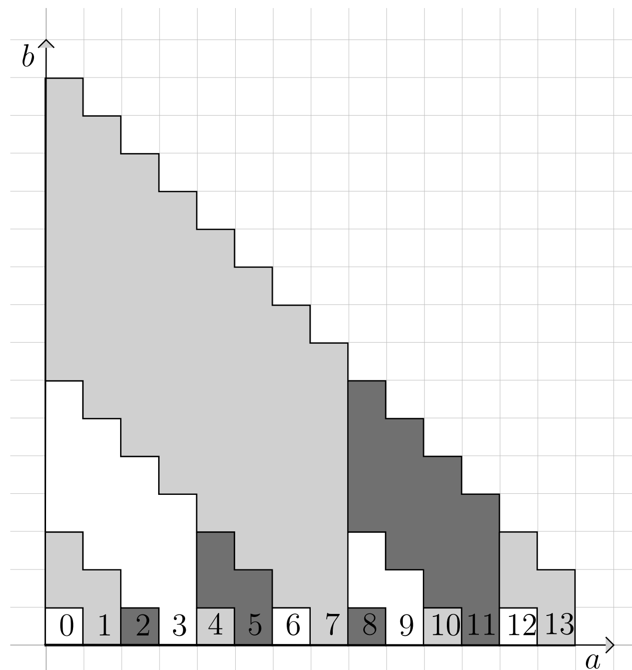

Figure 1.4. Semi-relaxed multiplication with middle product

Proposition 1.21. *Algorithm* `SemiRelaxedProductMiddleStep` *satisfies Property* $(\mathcal{HL})$*.*

This algorithm is still half-line for a because at step i, only the coefficients $a_0, \ldots,$ a_i are required. The input argument b is off-line because, for example, at step 3 the algorithms reads $b_0, \ldots, b_6$.

Complexity analysis Let MP^* be the multiplicative complexity function of the middle product. The multiplicative complexity $\mathsf{SRM}^*(n)$ of the semi-relaxed multiplication algorithm `SemiRelaxedProductMiddle`, that is Algorithm $\mathtt{Loop}_{\mathtt{SemiRelaxedProductMiddleStep}}$, for power series up to precision n is

$$\mathsf{SRM}^*(n) = \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{(n+2^k)}{2^{k+1}} \right\rfloor \mathsf{MP}^*(2^k).$$

Indeed, as we can see on Figure 1.4, we do a middle product of degree 2^k each 2^{k+1} step starting from step $2^k - 1$.

Proposition 1.22. *One has*

$$\mathsf{SRM}^*(n) \leqslant \begin{cases} 1.5\,\mathsf{M}^*(n) & \textit{for the naive multiplication} \\ 2\,\mathsf{M}^*(n) & \textit{for Karatsuba's multiplication} \end{cases}$$

and these bound are asymptotically optimal since

$$\mathsf{SRM}^*(2^m) \sim_{m\to\infty} \begin{cases} 1.5\,\mathsf{M}^*(2^m) & \textit{for the naive multiplication} \\ 2\,\mathsf{M}^*(2^m) & \textit{for Karatsuba's multiplication.} \end{cases}$$

Moreover when $\mathsf{M}^*(n) = K\, n \log_2(n) \log_2(\log_2(n))$ *with* $K \in \mathbb{R}_{>0}$*, one has*

$$\mathsf{SRM}^*(n) \sim_{n\to\infty} \frac{1}{4} \mathsf{M}^*(n) \log_2(n).$$

This proposition gives the entries of the second line in Table 1.1.

Proof. Let $\ell := \lfloor \log_2(n) \rfloor$. Since $\mathsf{MP}^*(n) = \mathsf{M}^*(n)$ (see Section 1.2.2), we deduce that

$$\mathsf{SRM}^*(n) := \sum_{k=0}^{\lfloor \log_2(n) \rfloor} \left\lfloor \frac{(n+2^k)}{2^{k+1}} \right\rfloor \mathsf{M}^*(2^k) = \mathsf{M}^{(3)}(n).$$

We start by taking M^* the cost function of the naive or Karatsuba's multiplication. By Lemma 1.16 and Lemma 1.17, one has

$$\mathsf{SRM}^*(n) = \mathsf{M}^{(3)}(n) = \frac{2^\alpha - 1}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{n}{2^\alpha - 2} \leqslant \frac{2^\alpha - 1}{2^\alpha - 2} \mathsf{M}^*(n).$$

When $n = 2^m$, one has

$$\mathsf{SRM}^*(2^m) = \frac{2^\alpha - 1}{2^\alpha - 2} \mathsf{M}^*(2^m) - \frac{2^m}{2^\alpha - 2} \sim_{m\to\infty} \frac{2^\alpha - 1}{2^\alpha - 2} \mathsf{M}^*(2^m).$$

Finally in the case where $\mathsf{M}^*(n) = K\, n \log_2(n) \log_2(\log_2(n))$ with $K \in \mathbb{R}_{>0}$, we use Lemma 1.18 to get

$$\mathsf{SRM}^*(n) \sim_{n\to\infty} \frac{1}{4}\, \mathsf{M}^*(n) \log_2(n). \qquad \square$$

Link between divide-and-conquer and semi-relaxed In fact, the algorithm of [Hoe03], referred as the DAC algorithm from now on, is a little bit different. It is based on the following divide-and-conquer approach. Let us fix the desired precision n in advance. The computation of $c = a\, b$ at precision n reduces to the computation of $c_{0\ldots k}$, $\mathrm{MP}(a_{0\ldots \ell}, b_{n+1-2\ell \ldots n})$ and $d_{0\ldots k}$ where $k := \lfloor n/2 \rfloor$, $\ell := \lceil n/2 \rceil$ and $d := a_{\ell \ldots n} b_{0\ldots k}$. Then

$$c_{0\ldots n} = c_{0\ldots k} + \mathrm{MP}(a_{0\ldots \ell}, b_{n+1-2\ell \ldots n})\, p^{n+1-\ell} + d_{0\ldots k}\, p^{\ell}.$$

This cutting of the problem can be seen geometrically on Figure 1.5.

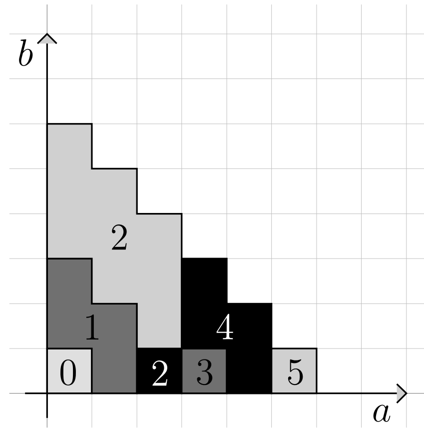

Figure 1.5. Divide-and-conquer truncated p-adics multiplication for $n = 6$

We have to compute the terms of the product inside a triangle. We make the biggest rhombus fit in the top left corner of the triangle; this corresponds to the middle product we do. Then the remaining area is the union of two triangles, which corresponds to two recursive calls. Now the DAC algorithm just reorders the computation so it can be relaxed up to precision n. Notice that our algorithm coincides with the DAC algorithm for precisions n that are powers of two minus one.

The first difference with our algorithm is that the scheme of computation of the DAC algorithm is adapted to the precision n; at step $n-1$, no unnecessary term of the product has been computed in the DAC algorithm. This differs with our algorithm which of course anticipates some computations. Therefore the DAC algorithm compares better to the off-line multiplication algorithm of Section 1.2.1.

Because the semi-relaxed multiplication using middle product comes from a divide-and-conquer approach, we should not be surprised if some relaxed algorithms for further problems based on this implementation of the multiplication coincides with divide-and-conquer algorithms. We will encounter two examples during this thesis: when solving a linear system over p-adics in Chapter 3 and when solving singular linear differential equations in Chapter 4.

1.3.4 Relaxed multiplication

Historically, the computation scheme of the forthcoming algorithm came from the on-line multiplication for integers of [FS74]. Then came the on-line multiplication for real numbers in [Sch97], and relaxed multiplication for power series [Hoe97, Hoe02], improved in [Hoe07] for some ground fields. This algorithm was extended to the multiplication of p-adic integers in [BHL11]. It is on-line with respect to both inputs a and b.

Algorithm `RelaxedProductStep`

Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$
Output: $c \in R_p$

1. **for** k from 0 to $\nu_2(i+2)$
   a. $c = c + a_{i+1-2^k \ldots i+1}\, b_{2^k-1 \ldots 2^{k+1}-1}\, p^i$
   b. **if** $(i+2 = 2^{k+1})$
      return c
   c. $c = c + a_{2^k-1 \ldots 2^{k+1}-1}\, b_{i+1-2^k \ldots i+1}\, p^i$
2. **return** c

Here is a diagram that sums up the computation made at each step. We can see on this figure that the algorithm is online and that at step i, the product is correct up to at least precision $i+1$.

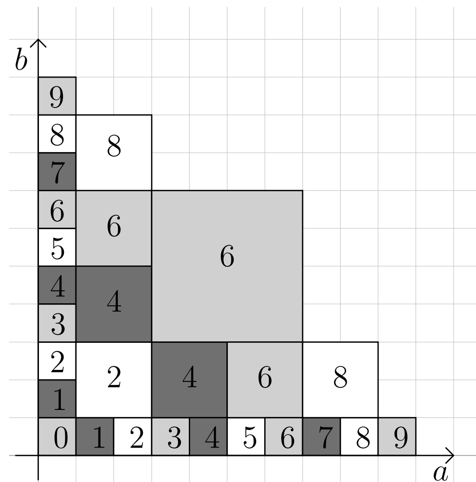

Figure 1.6. Relaxed multiplication

The relaxed algorithm is built recursively with the help of the semi-relaxed product. Suppose the relaxed product algorithm is constructed up to precision $2^m - 1$. Then one can extend it up to precision $2^{m+1} - 1$ with two semi-relaxed algorithms for $a_{2^m-1 \ldots \infty}\, b$ and $a\, b_{2^m-1 \ldots \infty}$. Then at precision $2^{m+1} - 1$, one completes the computations with the product $a_{2^m-1 \ldots 2^{m+1}-1}\, b_{2^m-1 \ldots 2^{m+1}-1}$ to obtain the terms $\sum_{0 \leqslant i,j \leqslant 2^{m+1}-1} a_i\, b_j\, p^{i+j}$ of the product $a\, b$. This construction is more obvious in Figure 1.6, where we identify the diagrams of the two semi-relaxed products.

Proposition 1.23. *Algorithm* `RelaxedProductStep` *satisfies Property* $(\mathcal{OL})$.

Once again, Figure 1.6 is of great help to see that the relaxed product algorithm does indeed compute the product $a\,b$. It is also easy to check that the algorithm is on-line on the diagram.

Complexity analysis Denote by $\mathsf{R}^*(n)$ the cost induced by all off-line multiplications done up to precision n, in the case where $R = \mathbb{k}[X]$. We can express it as

$$\mathsf{R}^*(n) = \sum_{k=0}^{\lfloor \log_2(n+1) \rfloor - 1} \left(2\left\lfloor \frac{n+1}{2^k} \right\rfloor - 3\right) \mathsf{M}^*(2^k).$$

Proposition 1.24. *One has*

$$\mathsf{R}^*(n) \leqslant \begin{cases} \mathsf{M}^*(n+1) & \text{for the naive multiplication} \\ 2.5\,\mathsf{M}^*(n+1) & \text{for Karatsuba's multiplication} \end{cases}$$

and these bounds are asymptotically optimal.

Moreover when $\mathsf{M}^*(n) = K\,n \log_2(n) \log_2(\log_2(n))$ *with* $K \in \mathbb{R}_{>0}$, *one has*

$$\mathsf{R}^*(n) \sim_{n \to \infty} \mathsf{M}^*(n) \log_2(n).$$

Proof. Let $\ell := \lfloor \log_2(n+1) \rfloor$ and $n + 1 = \sum_{i=0}^{\ell} \bar{n_i}\, 2^i$ be the base-2 expansion of $n+1$. We can express $\mathsf{R}^*(n)$ in terms of auxiliary complexity functions by

$$\mathsf{R}^*(n) = 2\,\mathsf{M}^{(2)}(n+1) - 3\,\mathsf{M}^{(1)}(n+1) + \mathsf{M}^*(2^\ell). \tag{1.3}$$

Assume that M^* is the cost function of the naive or Karatsuba's multiplication. Then, using Lemma 1.16, one has

$$\begin{aligned} \mathsf{R}^*(n) &= 2\left(\frac{2^\alpha}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{2\,n}{2^\alpha - 2}\right) - 3\left(\frac{2^\alpha}{2^\alpha - 1} \mathsf{M}^*(2^\ell) - \frac{1}{2^\alpha - 1}\right) + \mathsf{M}^*(2^\ell) \\ &= \left(\frac{2 \cdot 2^\alpha}{2^\alpha - 2} - \frac{3 \cdot 2^\alpha}{2^\alpha - 1} + 1\right) \mathsf{M}^*(2^\ell) + \frac{2 \cdot 2^\alpha}{2^\alpha - 2} \sum_{i=0}^{\ell-1} \bar{n_i}\, \mathsf{M}^*(2^i) + \frac{3}{2^\alpha - 1} - \frac{4\,n}{2^\alpha - 2} \\ &= C_1\, \mathsf{M}^*(2^\ell) + C_2 \sum_{i=0}^{\ell-1} \bar{n_i}\, \mathsf{M}^*(2^i) - C_3 \end{aligned}$$

with $C_1 = \frac{2^\alpha + 2}{(2^\alpha - 2)(2^\alpha - 1)}$, $C_2 = \frac{2 \cdot 2^\alpha}{2^\alpha - 2}$ and $C_3 = \frac{4\,n}{2^\alpha - 2} - \frac{3}{2^\alpha - 1}$. We begin by proving that for all $n \in \mathbb{N}_{>0}$, $C_3 \geqslant 0$. Indeed $(C_3 \geqslant 0) \Leftrightarrow \left(n \geqslant \frac{3\,(2^\alpha - 2)}{4\,(2^\alpha - 1)}\right)$ and

$$\frac{3\,(2^\alpha - 2)}{2\,(2^\alpha - 1)} = \begin{cases} 1/2 & \text{for } \alpha = 2 & \text{(naïve multiplication)} \\ 3/8 & \text{for } \alpha = \log_2(3) & \text{(Karatsuba's multiplication)} \end{cases}.$$

We can use Lemma 1.17 to deduce that $\mathsf{R}^*(n) \leqslant C_1\, \mathsf{M}^*(n+1)$ because $C_2/C_1 \leqslant (\mathsf{M}^*(3) - \mathsf{M}^*(2))$ for both naïve and Karatsuba's multiplication.

For $n = 2^m$, one has

$$\mathsf{R}^*(2^m) \;=\; C_1\, \mathsf{M}^*(2^m) - C_3 \sim_{m \to \infty} C_1\, \mathsf{M}^*(2^m).$$

The result for FFT multiplication is a consequence of Lemma 1.18 and Equation (1.3). □

The previous proposition proves the first row in the second table given in the introduction of this section.

1.3.5 Relaxed multiplication with middle and short products

In this subsection, we introduce a *new on-line algorithm* that uses both middle and short products. This algorithm improves by a constant factor the relaxed multiplication of the previous subsection.

We start by giving an overview of our scheme of computation. Figure 1.7 sums up the computations of the relaxed product algorithm using middle and short products.

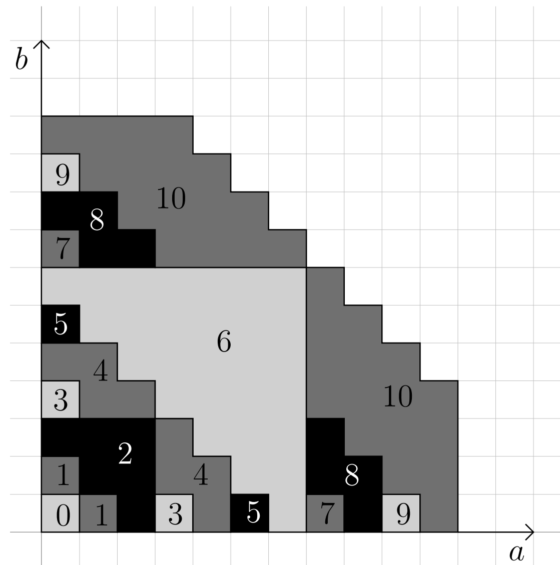

Figure 1.7. Relaxed multiplication with middle and short products

Similarly to the classical relaxed multiplication, we build our new relaxed multiplication algorithm on top of the semi-relaxed product with middle algorithm. The construction is recursive. Suppose that you have a relaxed multiplication algorithm up to precision $2^m - 1$ and that all the coefficients

$$\sum_{i=0}^{2^m-2} \sum_{j=0}^{2^m-2} a_i \, b_j \, p^{i+j}$$

of the product were computed at step $2^m - 2$. Then, for steps i with $2^m - 1 \leqslant i \leqslant 2^{m+1} - 3$, we perform two semi-relaxed products for computing $a_{2^m-1\ldots\infty} \, b$ and $a \; b_{2^m-1\ldots\infty}$. Therefore, at step $2^{m+1} - 3$, we have computed the coefficients $\sum_{0 \leqslant i+j \leqslant 2^{m+1}-3} a_i \; b_j \; p^{i+j}$ of $a \; b$. In order to continue the induction, we have to compute at step $d = 2^{m+1} - 2$ the missing terms

$$\sum_{0 \leqslant i,j \leqslant d, \; i+j \geqslant d} a_i \, b_j \, p^{i+j}.$$

These terms form a triangle on the diagram and can be computed by a short product

$$\sum_{0 \leqslant i,j \leqslant d, \; i+j \geqslant d} a_i \, b_j \, p^{i+j} := \mathrm{rev}_d(\mathrm{SP}(\mathrm{rev}_d(a), \mathrm{rev}_d(b))) \, p^d$$

where $\mathrm{rev}_d(a) = \sum_{i=0}^{d} a_{d-i}\, p^i$. Thus, right after step $2^{m+1}-2$, we have the terms $\sum_{i=0}^{2^{m+1}-2} \sum_{j=0}^{2^{m+1}-2} a_i\, b_j\, p^{i+j}$ of the product $a\, b$ and we can pursue the induction. This gives us the following algorithm, that is on-line with respect to both inputs a and b.

Algorithm `RelaxedProductMiddleStep`

Input: $a, b, c \in R_p$ and $i \in \mathbb{N}$
Output: $c \in R_p$

1. $m = \nu_2(i+2)$
2. **if** $(i+2 = 2^m)$
   a. $c = c + \mathrm{rev}_i(\mathrm{SP}(\mathrm{rev}_i(a_{0\ldots i+1}), \mathrm{rev}_i(b_{0\ldots i+1})))\, p^i$
   b. **return** c
3. $c = c + \mathrm{MP}(a_{i-2^m+1\ldots i+1}, b_{0\ldots 2^{m+1}-1})$
4. $c = c + \mathrm{MP}(b_{i-2^m+1\ldots i+1}, a_{0\ldots 2^{m+1}-1})$
5. **return** c

Remark 1.25. Even if there is no efficient short FFT multiplication algorithm, we can compute the short product of Step 2 efficiently. Indeed, we noticed in Section 1.2.3 that we adapt the FFT multiplication to compute $c_{0\ldots n} + c_{n\ldots 2n-1}$ where $c = a\, b$ and a, b are polynomials of length n. Since the part $c_{0\ldots n}$ was already computed by previous steps, we can access to $c_{n\ldots 2n-1} = \mathrm{rev}_{n-1}(\mathrm{SP}(\mathrm{rev}_{n-1}(a_{0\ldots n}), \mathrm{rev}_{n-1}(b_{0\ldots n})))$ in half the time of a multiplication.

As expected, our algorithm is a relaxed algorithm that computes the product of two elements $a, b \in R_p$. These properties can be read on Figure 1.7.

Proposition 1.26. *Algorithm `RelaxedProductMiddleStep` satisfies Property* $(\mathcal{OL})$.

Complexity analysis Denote by $\mathsf{RM}^*(n)$ the cost of the relaxed multiplication with middle products up to precision n. Let $\ell := \lfloor \log_2 (n+1) \rfloor$ so that this costs is given by

$$\mathsf{RM}^*(n) = \sum_{k=1}^{\ell} \mathsf{SP}^*(2^k - 1) + 2 \sum_{k=0}^{\ell-1} \left\lfloor \frac{n+1}{2^{k+1}} - \frac{1}{2} \right\rfloor \mathsf{MP}^*(2^k).$$

This formula comes from the fact that two middle products in size 2^k are done every 2^{k+1} steps, starting from step $3 \cdot 2^k - 2$. We distinguish two cases for Karatsuba's multiplication depending on the value of the ratio C_{SP} between short and plain multiplication.

Proposition 1.27. *One has*

$$\mathsf{RM}^*(n) \leqslant \begin{cases} \mathsf{M}^*(n+1) & \text{for the naive multiplication with } C_{\mathsf{SP}} = 1/2 \\ 1.75\, \mathsf{M}^*(n+1) & \text{for Karatsuba's multiplication if } C_{\mathsf{SP}} = 1/2 \\ 2.5\, \mathsf{M}^*(n+1) & \text{for Karatsuba's multiplication if } C_{\mathsf{SP}} = 1 \end{cases}$$

and these bounds are asymptotically optimal.

Moreover, when $\mathsf{M}^*(n) = K\, n \log_2(n) \log_2(\log_2(n))$ *with* $K \in \mathbb{R}_{>0}$*, one has*

$$\mathsf{RM}^*(n) \sim_{n \to \infty} \frac{1}{2} \mathsf{M}^*(n) \log_2(n).$$

As we will see in the following proof, the supremum of the ratio $\mathsf{RM}^*(n)/\mathsf{M}^*(n+1)$ depends linearly in C_{SP}. Therefore we can deduce this supremum for other C_{SP}. For example, in our implementation, we use the hybrid Karatsuba/naïve algorithm for plain multiplication (see Section 1.2.3) and an odd/even decomposition for short product. In this situation, the short product has a ratio $C_{\mathsf{SP}} = 0.6$. Although Proposition 1.27 does not deal with this hybrid multiplication algorithm, we believe the results for "pure" Karatsuba's multiplication should apply in this case for n large enough and yield a bound $\mathsf{RM}^*(n) \leqslant 1.9\, \mathsf{M}^*(n)$.

Proof. Let $\ell := \lfloor \log_2(n+1) \rfloor$ and $n+1 = \sum_{i=0}^{\ell} \bar{n_i}\, 2^i$ be the base-2 expansion of $n+1$. Since $\mathsf{MP}^*(n) = \mathsf{M}^*(n)$, we can express $\mathsf{RM}^*(n)$ in terms of auxiliary complexity functions by

$$\begin{aligned} \mathsf{RM}^*(n) &\leqslant C_{\mathsf{SP}} \sum_{k=1}^{\ell} \mathsf{M}^*(2^k - 1) + 2 \sum_{k=0}^{\ell-1} \left(\left\lfloor \frac{n+1}{2^{k+1}} + \frac{1}{2} \right\rfloor - 1 \right) \mathsf{M}^*(2^k) \\ &\leqslant (C_{\mathsf{SP}} - 2)\, \mathsf{M}^{(1)}(n+1) + 2\, \mathsf{M}^{(3)}(n+1) \end{aligned}$$

Assume that M^* is the cost function of the naive or Karatsuba's multiplication. Then, using Lemma 1.16, one has

$$\begin{aligned} \mathsf{R}^*(n) &= (C_{\mathsf{SP}} - 2) \left(\frac{2^\alpha}{2^\alpha - 1} \mathsf{M}^*(2^\ell) - \frac{1}{2^\alpha - 1} \right) + 2 \left(\frac{2^\alpha - 1}{2^\alpha - 2} \sum_{i=0}^{\ell} \bar{n_i}\, \mathsf{M}^*(2^i) - \frac{n}{2^\alpha - 2} \right) \\ &= C_1\, \mathsf{M}^*(2^\ell) + C_2 \sum_{i=0}^{\ell-1} \bar{n_i}\, \mathsf{M}^*(2^i) - C_3 \end{aligned}$$

with $C_1 = \frac{2^\alpha (2^\alpha - 2) C_{\mathsf{SP}} + 2}{(2^\alpha - 2)(2^\alpha - 1)}$, $C_2 = \frac{2 \cdot (2^\alpha - 1)}{2^\alpha - 2}$ and $C_3 = \frac{2n}{2^\alpha - 2} - \frac{2 - C_{\mathsf{SP}}}{2^\alpha - 1}$. We begin by noticing that for all $n \in \mathbb{N}_{>0}$, $C_3 \geqslant 0$. Indeed $(C_3 \geqslant 0) \Leftrightarrow \left(n \geqslant \frac{(2 - C_{\mathsf{SP}})(2^\alpha - 2)}{2(2^\alpha - 1)} \right)$ and since $C_{\mathsf{SP}} \geqslant 1/2$, one has

$$\frac{(2 - C_{\mathsf{SP}})(2^\alpha - 2)}{2(2^\alpha - 1)} \leqslant \begin{cases} 1/2 & \text{for } \alpha = 2 \quad \text{(naïve multiplication)} \\ 3/8 & \text{for } \alpha = \log_2(3) \quad \text{(Karatsuba's multiplication)} \end{cases}.$$

We can use Lemma 1.17 to deduce that $\mathsf{R}^*(n) \leqslant C_1\, \mathsf{M}^*(n+1)$ because $C_2/C_1 \leqslant (\mathsf{M}^*(3) - \mathsf{M}^*(2))$ for both naïve and Karatsuba's multiplication and any constant $1/2 \leqslant C_{\mathsf{SP}} \leqslant 1$.

These bounds are asymptotically optimal:

$$\mathsf{R}^*(2^m) = C_1\, \mathsf{M}^*(2^m) - C_3 \sim_{m \to \infty} C_1\, \mathsf{M}^*(2^m).$$

Lemma 1.18 also gives the result for FFT multiplication. □

1.3.6 Block variant

For large n, the ratio between on-line and off-line multiplication algorithms can get too big to be of any interest. This happens usually when using the FFT multiplication, as the ratio grows like $\log_2(n)$.

In this case, a d-block variant of an algorithm uses a p^d-adic representation of the p-adics in $R_{(p^d)} = R_{(p)}$. Instead of writing $y = \sum_{n \geqslant 0} y_n\, p^n$, we write $y = \sum_{n \geqslant 0} (y_n + y_{n+1}\, p + \cdots + y_{n+d-1}\, p^{d-1})\, p^{dn} \in R_{(p^d)}$. Then the d-block variant algorithm is on-line in the p^d-adic representation. It means that it computes d new coefficients at each step, instead of one coefficient at a time for an on-line algorithm in the p-adic representation.

By doing so, we can decrease the ratio between on-line and off-line multiplication algorithms by a constant; a complexity for relaxed product that was like $\mathsf{M}^*(n)\log_2(n)$ in p-adic representation gives a new complexity $\mathsf{M}^*(n)\log_2(n/d)$ in p^d-adic representation. We refer to [BHL11] for details.

1.4 Implementation and timings

We give timings, in seconds, of the different multiplication algorithms for the case of power series $\mathbb{F}_p[[X]]$ with the 29-bit prime number $p = 268435459$. Computations were done on one core of a INTEL CORE i5 at 2.40 GHz with 4Gb of RAM running a 32-bit LINUX. Our implementation uses the polynomial multiplication of NTL 5.5.2 [S+90]. The threshold between the naive and Karatsuba's multiplications is at degree 16 and the one between Karatsuba's and FFT multiplications at degree 1500.

In Figure 1.8, we plot the timings of the multiplication of polynomials and of several relaxed multiplication algorithms on power series depending on the precision in abscissa. Both coordinate axes use a logarithmic scale. The name `SRM` stands for the semi-relaxed multiplication using middle product of Section 1.3.3. The name `RM` stands for the relaxed multiplication using middle (and short) product of Section 1.3.5. And so on.

We can see that polynomial multiplication is faster from precision 8 on. The gap between any relaxed algorithm and the polynomial product remains constant in the Karatsuba range and grows as soon as we reach the FFT multiplication.

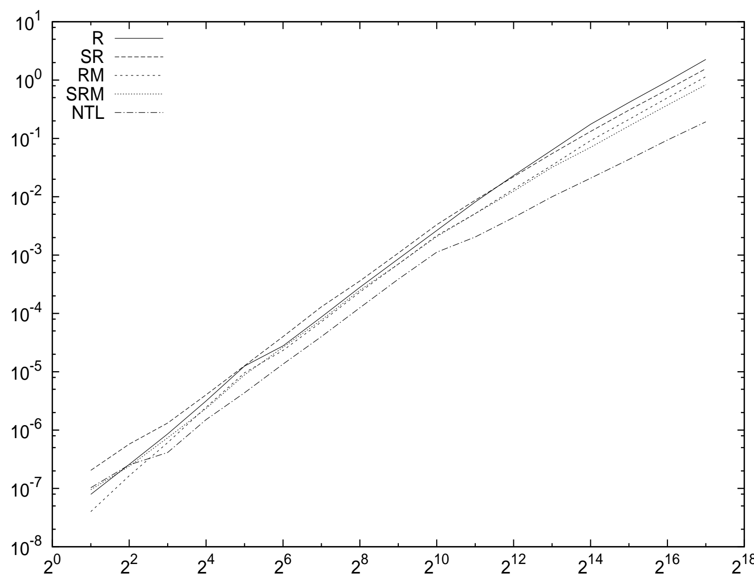

Figure 1.8. Timings of different multiplication algorithms

In Figure 1.9, we display the ratio of timings of several relaxed multiplication algorithms compared to the polynomial product depending on the precision in abscissa. This plot confirms the theoretical bounds for Karatsuba's multiplication, except on a few points, and the constants 1, 1/2 or 1/4 in the asymptotic equivalents for the FFT multiplication. We can see that the use of middle product always improves the performance of both the relaxed and semi-relaxed multiplication algorithms. We save up to a factor 2, which is attained for the FFT multiplication.

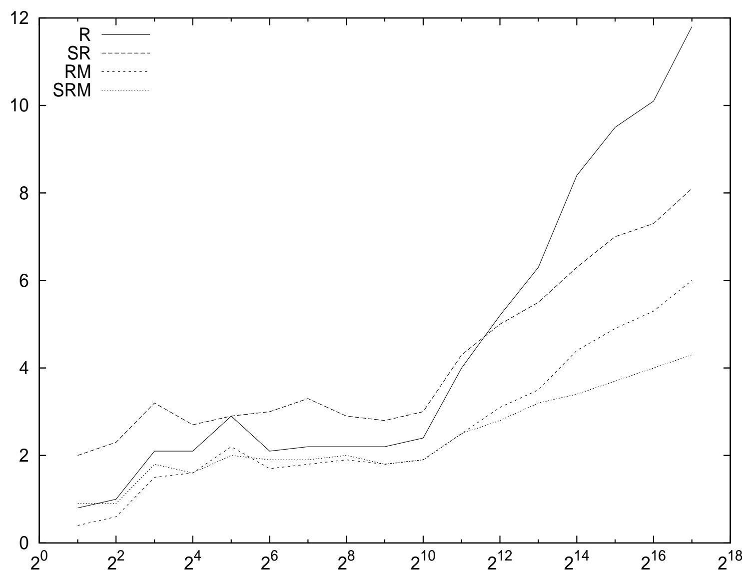

Figure 1.9. Ratio of timings of different relaxed products w.r.t. polynomial multiplication