The +Ideals package. Catalogue of routines

Jordi Guàrdia and Enric Nart

Contents

0 Computational representation of fractional ideals **3**
- 0.1 Attributes of number fields 3
- 0.2 Fractional ideals. The `IdealRecord` record 4
- 0.3 OM representations of prime ideals. The `TypeLevel` record 4

1 Prime ideal decomposition **6**
- 1.1 Montes(polynomial, p: NumberField:=false) 6
- 1.2 Montes(K, p) 7
- 1.3 Montesloop(∼Pol, ∼Leaves, ∼totalindex, mahler: NumberField:=false) 7
- 1.4 InitialPrimeIdeal(p, psi, power) 7
- 1.5 Taylor(polynomial, phi, omega) 8
- 1.6 Value(i, ∼P, ∼polynomial, ∼devs, ∼val) 8
- 1.7 Newton(i, ∼P, ∼phiadic, ∼sides, ∼devsEachSide) 9
- 1.8 ResidualPolynomial(i, ∼P, ∼devsSide, ∼psi) 9
- 1.9 Construct(i, ∼type, p, respol, point, ∼Ppol) 9
- 1.10 Representative(∼type, p) 10
- 1.11 LastLevel(Phiadic, ∼P, Pol, slope, dev: NumberField:=false) 10
- 1.12 PrescribedValue(∼P, value, ∼Phi, ∼logphi) 10
- 1.13 CrossValues(K, p, tree) 10
- 1.14 IndexOfCoincidence(P1, P2) 10
- 1.15 TrueDiscriminant(K) 11
- 1.16 PolToFieldElt(K, polynomial) 11

2 Single-factor lifting **11**
- 2.1 SFL(∼P, slope: update:=false) 11
- 2.2 SFLprecision(∼P, precision: update:=false) 11
- 2.3 Cancel(poly, vden: QUO:=true) 12
- 2.4 Inversionloop(A, ∼xnum, ∼xden, phi, precision, Zp) 12
- 2.5 AdaptPrecision(Zp, pol, llista) 12
- 2.6 FaithfulpAdicConversion(pol, p) 13
- 2.7 PathOfPrecisions(n, h) 13
- 2.8 UpdateLastLevel(∼P) 13
- 2.9 SFLInitialize(∼P) 13
- 2.10 LocalLift(class, P) 13
- 2.11 LocalLift(∼P, class, ∼numlift, ∼denlift) 14
- 2.12 ConvertLogs(∼P, log, ∼class) 14
- 2.13 Z(∼type, i, ∼z) 14

3 p-adic Factoritzation **14**
3.1 pAdicFactors(poly, p, precision) . . . 14
3.2 pDiscriminant(poly, p) . . . 14
3.3 Different(∼P, ∼different) . . . 15
3.4 pResultant(poly, poly2, p) . . . 15

4 $\mathfrak{p}$-adic valuations **15**
4.1 Localize(alpha, p) . . . 15
4.2 EqualizeLogs(∼log1,∼log2) . . . 15
4.3 PValuation(alpha, P: RED:=false) . . . 15
4.4 IsIntegralM(alpha) . . . 16

5 Reduction modulo powers of prime ideals **16**
5.1 ResidueField(P) . . . 16
5.2 Reduction(alpha, P, m) . . . 16
5.3 Reduction(alpha, P) . . . 16
5.4 LocalLift(class, P, m) . . . 16
5.5 Lift(class, P, m) . . . 17
5.6 MultiplierLift(∼P, exponents, ∼mult) . . . 17
5.7 TreeInterval(∼P, ∼tree) . . . 17
5.8 MultPiece(∼P, tree, expsTree, ∼N, ∼bp) . . . 17
5.9 CompensateValue(K, p, tree, expsTree) . . . 18
5.10 MultiplyByInverse(∼alpha, ∼P, m) . . . 18

6 Generators of prime ideals **18**
6.1 Multipliers(K, p, values) . . . 18
6.2 Generators(K, p) . . . 18

7 Manipulation of fractional ideals **19**
7.1 ideal(K, listgen) . . . 19
7.2 IsIdealRecord(I) . . . 19
7.3 IsPrimeIdeal(I) . . . 19
7.4 IsOne(I) . . . 19
7.5 IsZero(I) . . . 19
7.6 IsIntegral(I) . . . 19
7.7 I eq J . . . 19
7.8 alpha in J . . . 19
7.9 I subset J . . . 20
7.10 I*J . . . 20
7.11 I^n . . . 20
7.12 I/J . . . 20
7.13 I+J . . . 20
7.14 Factorization(∼I) . . . 20
7.15 Factorization(I) . . . 20
7.16 FactorListToString(list) . . . 20
7.17 Norm(I) . . . 20
7.18 PValuation(I, P) . . . 20
7.19 RationalRadical(I) . . . 20

7.20 RationalDenominator(I) 20
7.21 TwoElement(~I) 21
7.22 TwoElement(I) 21

8 Integral bases **21**
8.1 pIntegralBasis(I, p: HNF:=false, Separated:=false) 21
8.2 pIntegralBasis(K, p: HNF:=false) 21
8.3 reduceIdeal(I, p: exponents:=false) 21
8.4 IdealBasis(I: HNF:=false, Separated:=false) 21
8.5 SIdealBasis(I, S) 21
8.6 IntegralBasis(K: HNF:=false) 21

9 Chinese remainders theorem **22**
9.1 LocalCRT(K, p, exponents) 22
9.2 CRT(elements, ideals) 22

10 Construction of types **22**
10.1 InitializeType(p, psi) 22
10.2 EnlargeType(h, e, psi, ~type, ~Y, ~z) 22
10.3 CreateType(p, list) 23
10.4 CreateRandomType(p, r) 23
10.5 CreateRandomMultipleTypePolynomial(p, k, r, s) 23
10.6 RandomMultiplicityType(p, r, s) 23
10.7 CombineTypes(listoftypes) 24
10.8 CombinePolynomialsWithDifferentPrimes(f1, p1, f2, p2, k) 24

0 Computational representation of fractional ideals

Let us briefly describe the structure of the computational representation of fractional ideals. We freely use the notation of the papers [1, 2, 3, 4, 5, 6].

0.1 Attributes of number fields

Let $K = \mathbb{Q}(\theta)$ be a number field generated by a root θ of a monic irreducible polynomial $f(x) \in \mathbb{Z}[x]$, and let $\mathbb{Z}_K$ be the ring of integers of K.

The package creates several attributes for the number field K:

`Discriminant =` discriminant of the extension $K/\mathbb{Q}$,
`FactorizedDiscriminant = Factorization(Discriminant(`$f(x)$`))`,
`FactorizedPrimes =` whose prime ideal decomposition has been computed already,
`IndexPrimefactors =` list of prime numbers dividing the index $(\mathbb{Z}_K\colon \mathbb{Z}[\theta])$,
`IntegralBasis =` a triangular integral basis of K.

The rest of attributes are *associative arrays*, that may be indexed by a prime number.

`LocalIndex[p] = p`-adic valuation of the index $(\mathbb{Z}_K\colon \mathbb{Z}[\theta])$,
`PrimeIdeals[p] =` list of prime ideals lying over `p`,
`TreesIntervals[p] =` list of intervals `[i..j]` indicating the positions in the list `` K`PrimeIdeals[p] `` of the prime ideals attached to p-adic irreducible factors of $f(x)$ that are congruent to a power of the same irreducible polynomial modulo `p`.

0.2 Fractional ideals. The `IdealRecord` record

A fractional ideal `I` of K is represented as a record `IdealRecord`, whose attributes are:

`Parent`, the number field K,
`Generators`, a list of elements of K that generate `I`.
`IntegerGenerator`, least positive rational number belonging to `I`,
`Generator`, an element $\alpha \in K$ such that `I` is generated by its `IntegerGenerator` and α as a $\mathbb{Z}_K$-module,
`IsIntegral`, true or false,
`IsPrime`, true or false,
`Factorization`, a list `[...,[p,j,e],...]`, where a triple `[p,j,e]` represents the `e`-th power of the `j`-th prime ideal over `p` in the list ``K`PrimeIdeals[p]``.
`FactorizationString`, a literal expression of the factorization of `I`, in which every triple `[p,j,e]` is written as `P[p,j]^e`.

Finally, the `IdealRecord` has some more attributes that concern only prime ideals. If `I` is a non-zero prime ideal $\mathfrak{p} \in \operatorname{Spec}(\mathbb{Z}_K)$, these attributes are:

`Position`, position of $\mathfrak{p}$ in the list ``K`PrimeIdeals[p]``,
`e`, ramification index $e(\mathfrak{p}/p)$,
`f`, residual degree $f(\mathfrak{p}/p)$,
`Type`, an OM representation of $\mathfrak{p}$ (see next section),
`exponent`, exponent of the p-adic irreducible factor of $f(x)$ attached to $\mathfrak{p}$ [6, §5.1],
`LocalGenerator`, an element $\pi \in K$, with $v_{\mathfrak{p}}(\pi) = 1$,
`LogLG`, list of exponents $[\ell_0, \dots, \ell_r]$ such that $\pi = p^{\ell_0}\phi_1(\theta)^{\ell_1} \cdots \phi_r(\theta)^{\ell_r}$, where $[\phi_1, \dots, \phi_r]$ is the Okutsu frame contained in the `Type` (see next section).
`sflPols`, list of polynomials which are need for the SFL routine (see section 2),
`sfl`, list of data which are need for the SFL routine (see section 2),
`LastLevelNeedsUpdate`, true or false (see section 2).

0.3 OM representations of prime ideals. The `TypeLevel` record

Let p be a prime number. The prime ideals $\mathfrak{p}$ of $\mathbb{Z}_K$ lying over p are in 1-1 correspondence with the monic irreducible factors of $f(x)$ over $\mathbb{Z}_p[x]$. We denote by $F_{\mathfrak{p}}(x) \in \mathbb{Z}_p[x]$ the irreducible factor attached to $\mathfrak{p}$.

An *OM representation* of $F_{\mathfrak{p}}$ is a certain *type* of order $r+1$,

$$\mathbf{t} = (\psi_0; (\phi_1, \lambda_1, \psi_1); \dots; (\phi_{r+1}, \lambda_{r+1}, \psi_{r+1}))$$

where r is the *Okutsu depth* of $F_{\mathfrak{p}}$ [4, Secs. 3,4]. We abuse of language and we refer to $\mathbf{t}$ as an OM representation of the prime ideal $\mathfrak{p}$ as well.

Computationally, a type of order $r+1$ is just a list of $r+1$ records `TypeLevel`, which are called the *levels* of the type. Let us describe the attributes of such a record.

A. Attributes of `TypeLevel` linked to the irreducible factor $F := F_{\mathfrak{p}}$

At each level $1 \le i \le r+1$, we find the following attributes:

`Phi` $= \phi_i \in \mathbb{Z}[x]$,
`slope` $= \lambda_i \in \mathbb{Q} \cup \{\infty\}$,
`psi` $= \psi_i \in \mathbb{F}_i[y]$,

`V` $= v_{i-1}(\phi_i)$,
`h` $=$ `Numerator(slope)`$=: h_i$, (if `slope` $\ne \infty$)
`e` $=$ `Denominator(slope)`$=: e_i$, (if `slope` $\ne \infty$)
`f` $=$ `Degree(psi)`$=: f_i$,
`prode` $= e_0 \cdots e_{i-1}$,
`prodf` $= f_0 \cdots f_{i-1}$,
`invh` $=$ `InverseMod(`h_i`,`e_i`)`,
`Fq` $= \mathbb{F}_i = \mathbb{F}_{i-1}[y]/(\psi_{i-1}(y))$,
`FqY` $= \mathbb{F}_i[y]$,
`z` $= z_{i-1} \in \mathbb{F}_i$, the class of $y \in \mathbb{F}_{i-1}[y]$,
`logPi` $= \log \pi_i = (\mu_0, \dots, \mu_{i-1}, 0) \in \mathbb{Z}^{i+1}$,
`logPhi` $= \log \Phi_i = (\ell_0, \dots, \ell_{i-1}, 1) \in \mathbb{Z}^{i+1}$,
`logGamma` $= \log \gamma_i = (\nu_0, \dots, \nu_{i-1}, e_i) \in \mathbb{Z}^{i+1}$.

Let us recall the meaning of the fundamental data $(\phi_i, \lambda_i, \psi_i)$ of each level. Let $K_{\mathfrak{p}}$ be the completion of K with respect to the $\mathfrak{p}$-adic topology, and consider a topological embedding $K \subset K_{\mathfrak{p}} \hookrightarrow \overline{\mathbb{Q}}_p$. Denote again by $\theta \in \overline{\mathbb{Q}}_p$ the image of $\theta \in K$ under this embedding, so that θ becomes a root of $F(x)$. If we denote $m_i := \deg \phi_i$, then for every monic polynomial $g(x) \in \mathbb{Z}_p[x]$ we have:

$$m_i \le \deg g < m_{i+1} \implies \frac{v(g(\theta))}{\deg g} \le \frac{v(\phi_i(\theta))}{m_i} < \frac{v(\phi_{i+1}(\theta))}{m_{i+1}},$$

for all $1 \le i \le r$, where v is the canonical valuation of $\overline{\mathbb{Q}}_p$.

On the other hand, for each level $1 \le i \le r+1$, the type supports a Newton polygon operator N_i, a discrete valuation v_{i-1} on $\mathbb{Q}_p[x]$ and a residual polynomial operator $R_i \colon \mathbb{Q}_p[x] \to \mathbb{F}_i[y]$. The Newton polygon $N_i(F)$ is one-sided of slope $-\lambda_i$ and the residual polynomial $R_i(F)$ is a power of ψ_i.

The data of all levels $1 \le i \le r$, and the data `prode, prodf, V, Fq, FqY, z` from the $(r+1)$-th level, are linked to certain *Okutsu invariants* of F [4, §4.1]. For instance,

$$v(\phi_i(\theta)) = (V_i + \lambda_i)/e_0 \cdots e_{i-1}, \quad 1 \le i \le r.$$

The polynomial ϕ_{r+1} is an *Okutsu approximation* to F [1, Sec. 4]. We also say that ϕ_{r+1} and F are *Okutsu equivalent*, and we write $\phi_{r+1} \approx F$. The data `Phi, slope, psi, h, e` from the $(r+1)$-th level are linked to this approximation. The datum `slope` contains the absolute value of the slope of the first side of $N_{r+1}^-(f)$, whose end points have always abscissa 0 and 1. The value of `slope` is either a positive integer or `Infinity()`. The latter case occurs only when $f(x) = \phi_{r+1}(x)$.

The rational functions π_i, Φ_i, $\gamma_i \in \mathbb{Q}(x)$ are recurrently defined [4, §1.4]. They may be expressed as a product of powers of p and $\phi_1, \dots, \phi_r$, with integer exponents (either positive or negative). We denote $\log\left(p^{\ell_0}\phi_1^{\ell_1}\cdots\phi_r^{\ell_r}\right) := (\ell_0, \dots, \ell_r)$.

B. Attributes of `TypeLevel` linked to the defining polynomial $f(x)$

At each level $1 \le i \le r+1$, we find the following attributes:

`omega` $= \operatorname{ord}_{\psi_i} R_i(f)$,
`cuttingslope` $=$ previous slope λ_i if we are in a refinement step,
`Refinements` $=$ `[*...,[`ϕ`,`λ`],...*]`.

For the description of the process of *refinement* see [2, Sec. 3.2].

The list `Refinements` is only used for the computation of the *cross values* $v_{\mathfrak{q}}(\phi_i(\theta))$, for a prime ideal $\mathfrak{q}$ over p, different from $\mathfrak{p}$.

1 Prime ideal decomposition

Certain variables will have a common meaning in all routines:

- `K` is a number field defined by a monic polynomial with integer coefficients.
- `p` is a prime number.
- `polynomial` is a polynomial with integer coefficients.
- `P` is a prime ideal.

1.1 Montes(polynomial, p: NumberField:=false)

This routine applies the Montes algorithm to a monic polynomial with integer coefficients and a prime number p. It outputs three objects:

• `OMreps`: a list of prime ideals attached to the p-adic irreducible factors $F_1, \dots, F_t$ of the input polynomial.

• `TreesIntervals`: a list of intervals `[i..j]` indicating the positions in the list `OMreps` of the prime ideals whose corresponding p-adic irreducible factors are congruent to a power of the same irreducible polynomial modulo p.

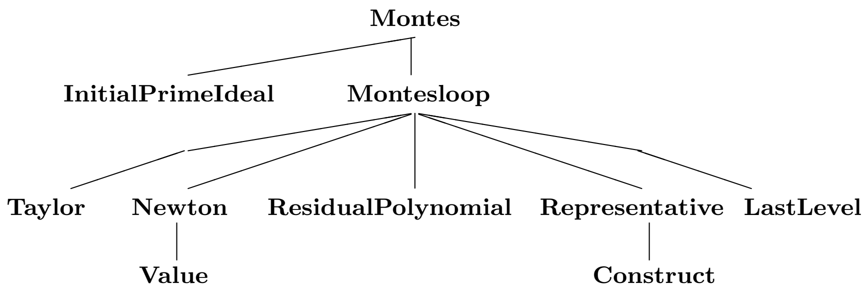
• `totalindex` $= \sum_{i=1}^{t} \operatorname{ind}_p(F_i) + \sum_{1\le i<j\le t} \operatorname{res}_p(F_i, F_j)$, where $\operatorname{ind}_p(F_i)$ is the p-adic valuation of the index of $\mathbb{Z}_p[\theta]$ in the maximal order of the extension of $\mathbb{Q}_p$ determined by F_i, and $\operatorname{res}_p(F_i, F_j)$ is the p-adic valuation of the resultant of F_i and F_j.

If `NumberField` is given the value `false`, then the routine detects a non-squarefree input polynomial by checking if the variable `totalindex` is larger than a certain bound. In this case, it outputs `totalindex = Infinity()`.

If `NumberField` is given the value `true`, then the routine fills up the $(r+1)$-th level of the type attached to each prime ideal. Otherwise, the polynomial ϕ_{r+1} is computed, but neither the slope λ_{r+1} nor the polynomial ψ_{r+1} are computed.

If we previously set `SetVerbose(¨montestalk¨,3)`, then along the execution of `Montes` some information on all partial computations is displayed.

Structure of subroutines.



1.2 Montes(K, p)

This routine calls `Montes(DefiningPolynomial(K),p: NumberField:=true)`.

The prime p is appended to the list `` K`FactorizedPrimes `` and the following items are computed:

- `` K`PrimeIdeals[p] ``: list of the prime ideals dividing `p`.
- `` K`LocalIndex[p] ``: `p`-adic valuation of $(\mathbb{Z}_K : \mathbb{Z}[\theta])$.
- `` K`TreesIntervals[p] ``: list of intervals `[i..j]`, one for each irreducible factor ψ_0 of the defining polynomial $f(x)$ modulo `p`. Each interval shows the positions in the list `` K`PrimeIdeals[p] `` of the prime ideals associated with the p-adic irreducible factors of $f(x)$ which are congruent to a power of ψ_0 modulo `p`.

1.3 Montesloop(~Pol, ~Leaves, ~totalindex, mahler: NumberField:=false)

Input.

- `Pol` is a monic polynomial with integer coefficients.
- `Leaves=[𝔭]` is a list containing initially a (fake) prime ideal constructed by the routine `InitialPrimeIdeal` from an irreducible factor ψ_0 of `Pol` modulo p.
- `mahler` is un upper bound for `totalindex` for a squarefree polynomial [7].

Output.

- A list `Leaves` of all prime ideals associated with the p-adic irreducible factors of `Pol`. These prime ideals are the leaves of a tree, but `Leaves` is simply the sequence of these leaves and the tree structure is lost.
- The global variable `totalindex` accumulates the contribution to the index of the prime factors of `Pol` (cf. routine 1.1).

If `NumberField` is given the value `true`, then `Pol` is the defining polynomial of a number field K and it does not change during the execution of the routine. Otherwise, `Pol` is a factor in $\mathbb{Z}[x]$ of the input polynomial of the routine `Montes(poly,p)`, and it may be changed along the execution of the routine.

1.4 InitialPrimeIdeal(p, psi, power)

Input.

- `psi` is a monic irreducible polynomial in $\mathbb{F}_p[x]$.
- `power` is the exponent with which `psi` divides the input polynomial of the `Montes` routine modulo `p`.

Output. A record `IdealRecord` with the following attributes:

```
IntegerGenerator:= p,
Type:=[level],
```

```
exponent:=0,
sflPols:=[* 0,0,0,0,0 *],
sfl:=[* 0,0,0,0,0 *].
```

The unique `level` of the `Type` is a `Typelevel` record with the following attributes:
`Phi:=`a monic lifting of `psi` to $\mathbb{Z}[x]$
`V:=0,`
`prode:=1,`
`prodf:=Degree(psi),`
`Fq:=ext<GF(p) | psi>,`
`FqY:=`the polynomial ring over `Fq`,
`z:=` $\begin{cases} \texttt{Fq.1}, & \text{if } \texttt{prodf} > 1, \\ \texttt{-Coefficient(psi,0)}, & \text{if } \texttt{prodf} = 1. \end{cases}$
`omega:=power,`
`cuttingslope:=0,`
`Refinements:=[* *],`
`logPi:=Vector([1,0]),`
`logPhi:=Vector([0,1]).`

1.5 Taylor(polynomial, phi, omega)

Input.

- `phi` is a monic polynomial with integer coefficients,
- `omega`$=: \omega$ is a non-negative integer.

Output. A list $[a_0,...,a_\omega]$ of $\omega+1$ polynomials with $\deg a_i < \deg$ `phi` such that
`polynomial` $\equiv a_0$ `+` a_1`phi + ... +` a_ω`phi`$^\omega$ `mod phi`$^{\omega+1}$.

1.6 Value(i, ~P, ~polynomial, ~devs, ~val)

Input. i is a positive index, less than or equal to #`` P`Type+1 ``.

Output.

- `val` is the non-negative integer v_{i-1}(`polynomial`), where v_{i-1} is the $(i-1)$-th discrete valuation of $\mathbb{Q}_p(x)$ determined by `` P`Type ``
- If $i=1$, then `devs=polynomial`. If $i>1$, `devs` is the computational representation of the λ_{i-1}-*component* $S_{\lambda_{i-1}}(N)$ of the Newton polygon $N=N_{i-1}$(`polynomial`) [3, §1].

Let us be more precise about the structure of the output `devs` for $i>1$.

Let `polynomial` $=\sum_{0\le k} a_k\phi_{i-1}^k$ be the canonical ϕ_{i-1}-expansion of `polynomial`, and denote $u_k:=v_{i-2}(a_k(\phi_{i-1})^k)$, so that the Newton polygon N is the lower convex hull of the cloud of points (k,u_k). In this case, `devs` is a nested list of lists storing the $(\phi_1,\dots,\phi_{i-1})$-multiadic expansion of the coefficients a_k for which (k,u_k) lies on $S_{\lambda_{i-1}}(N)$. Also, the last entry of `devs` stores the left end point of this segment. More precisely, suppose (s,u) is the left end point of $S_{\lambda_{i-1}}(N)$, and $s+de_{i-1}$ is the abscissa of the right end point. If we denote by $\texttt{devs}_j(g)$ the output list `devs` for the input $i=j$, $\texttt{polynomial}=g$, then:

$$\texttt{devs = devs}_i\texttt{(polynomial)=[* dv}_0\texttt{, dv}_1\texttt{, ..., dv}_d\texttt{, [}s,u\texttt{] *]},$$

where, for $0\le j\le d$,

$$\texttt{dv}_j=\begin{cases}\texttt{devs}_{i-1}(a_{s+je_{i-1}}), & \text{if } (s+je_{i-1},u_{s+je_{i-1}}) \text{ lies on N},\\ \texttt{[]}, & \text{if } (s+je_{i-1},u_{s+je_{i-1}}) \text{ lies above } N \text{ and } i>2,\\ \texttt{0}, & \text{if } (s+je_{i-1},u_{s+je_{i-1}}) \text{ lies above } N \text{ and } i=2.\end{cases}$$

1.7 Newton(i, ~P, ~phiadic, ~sides, ~devsEachSide)

Input.

- `i` is a positive index, less than or equal to #``P`Type``
- $\texttt{phiadic=[}a_0,\dots,a_\omega\texttt{]}\ne\texttt{[}0,\dots,0\texttt{]}$ is a piece of the ϕ_i-expansion of a certain polynomial with integer coefficients.

Output. Let N be the Newton polygon of the cloud of points $\big(k, v_{i-1}(a_k(\phi_i)^k)\big)$, for $0\le k\le\omega$.

- `sides` is a list of the sides of N. The structure of each side is: $\mathrm{S}=[-\lambda,s,u,s',u']$, where $-\lambda$ is the slope of the side and (s,u), (s',u') are the end points.

  If N is a single point (s,u), then $\texttt{sides=[[0,}s\texttt{,}u\texttt{,}s\texttt{,}u\texttt{]]}$.
- `devsEachSide` is a list of the same length as `sides`, containing the computational representations, as described in the routine `Value`, of the different λ-components $S_\lambda(N)$, for $-\lambda$ running on the slopes of the sides of N.

1.8 ResidualPolynomial(i, ~P, ~devsSide, ~psi)

Input.

- `i` is a positive index, less than or equal to #``P`Type``
- `devsSide` is the component $S_{\lambda_i}(N)$ of the Newton polygon $N:=N_i^-(g)$ of a certain polynomial $g(x)$, as it is computed by the routine `Newton`.

Output. $\texttt{psi}=R_i(g)(y)=R_{v_{i-1},\phi_i,\lambda_i}(g)(y)\in\mathbb{F}_i[y]$ is the residual polynomial of order i of $g(x)$, with respect to the slope λ_i.

1.9 Construct(i, ~type, p, respol, point, ~Ppol)

Input.

- `type` is a type
- `i` is a positive index, less than or equal to #`type`
- $\texttt{respol}=:\varphi(y)$ is a polynomial with coefficients in $\mathbb{F}_i$, of degree less than f_i
- $\texttt{point}=(s,u)\in\mathbb{Z}^2$ satisfies $0\le s<e_i$, $V:=ue_i+sh_i\ge V_{i+1}:=$``type`[i+1]`V``.

Output. A polynomial `Ppol`, with integer coefficients such that:

$$\deg\texttt{Ppol}=e_i\cdot\deg\varphi\cdot m_i,\quad v_i(\texttt{Ppol})=V,\quad y^{\mathrm{ord}_y(\varphi)}R_i(\texttt{Ppol})(y)=\varphi(y).$$

1.10 Representative(~type, p)

This routine constructs a *representative* ϕ_{s+1} of a type of order s, and it enlarges the type with an $(s+1)$-th level with an assigned value of the following attributes: `Phi:=`ϕ_{s+1}`, V, cuttingslope, Refinements, prode, prodf, Fq, FqY, z`.

1.11 LastLevel(Phiadic, ~P, Pol, slope, dev: NumberField:=false)

This subroutine is called by `Montesloop` when `P` has been detected to be a prime ideal associated with a p-adic irreducible factor of `Pol`. The input variable `Phiadic` contains a list of the two first coefficients $a_0(x), a_1(x)$ of the ϕ_{r+1}-expansion of `Pol`, where $r+1 = \#$`P`Type`. The input variable `slope` contains the absolute value $\lambda_{r+1}$ of the slope of the first side of $N := N_{r+1}($`Pol`$)$, whose end points have abscissa 0 and 1. The input variable `dev` contains the $\lambda_{r+1}$-component of $N$ as computed by the routine `Newton`.

The routine assigns some global attributes like `P`e`, `P`f`, `P`exponent`. Also, it assigns some attributes that are necessary for future calls to the routine `SFL` in order to improve the Okutsu approximation $\phi_{r+1}$ to the true $p$-adic factor of `Pol`:

`P`LastLevelNeedsUpdate:= not NumberField;` (if `slope` is finite)
`P`sflPols[1]:=Phiadic[1];`
`P`sflPols[2]:=Phiadic[2];`

Finally, if `NumberField = true` and `slope` is finite, then it fills up the $(r+1)$-level of `P`Type` with the adequate values of `P`Type[`$r+1$`]`psi` and `P`Type[`$r+1$`]`logGamma`.

1.12 PrescribedValue(~P, value, ~Phi, ~logphi)

Input. `value` is an integer.

Output.

- `logphi` is a vector $(\ell_0, \dots, \ell_r) \in \mathbb{Z}^{r+1}$, where r is the Okutsu depth of the prime ideal. We have: $\ell_j \geq 0$, for all $j > 0$
- `Phi` is the polynomial $\phi_1^{\ell_1} \cdots \phi_r^{\ell_r}$, and it satisfies: $v_{\mathfrak{p}}(p^{\ell_0}\mathtt{Phi}(\theta)) = \mathtt{value}$.

1.13 CrossValues(K, p, tree)

The variable `tree` is an interval of positions in the list `K`PrimeIdeals[p]`. The prime ideals marked by `tree` are $\mathfrak{p}_{s+1}, \dots, \mathfrak{p}_{s+t}$, where $t = \#\mathtt{tree}$ and $s = \mathtt{tree}[1] - 1$.

The routine outputs a matrix $M \in \mathbb{Q}^{t\times t}$ with entries:

$$M(i,j) = \begin{cases} v_{\mathfrak{p}_j}(\phi_{\mathfrak{p}_i}(\theta))/e(\mathfrak{p}_j/\mathtt{p}), & \text{if } i \neq j, \\ 0, & \text{if } i = j, \end{cases}$$

where $\phi_{\mathfrak{p}}$ denotes the last ϕ-polynomial of the OM representation of $\mathfrak{p}$.

1.14 IndexOfCoincidence(P1, P2)

Input. `P1, P2` are two *different* prime ideals lying over the same prime number.

Output. their index of coincidence $i \in \mathbb{Z}_{\geq 0}$.

If P1 and P2 belong to different trees, then $i = 0$. Otherwise, i is the least index such that the two triples (P`Type[$i$]`Phi,P`Type[$i$]`slope,P`Type[$i$]`psi) for P=P1 and P=P2 do not coincide.

1.15 TrueDiscriminant(K)

The discriminant of $K/\mathbb{Q}$ is stored in K`Discriminant.

The factorization of the discriminant of the defining polynomial of K is stored in K`FactorizedDiscriminant.

The list of prime numbers p with a positive value of K`LocalIndex[p] is stored in K`IndexPrimeFactors.

1.16 PolToFieldElt(K, polynomial)

It outputs polynomial(θ) $\in K$.

2 Single-factor lifting

This section deals with the applications of the routine introduced in [5] and revised in [4], for the improvement of a single Okutsu approximation to a prescribed precision.

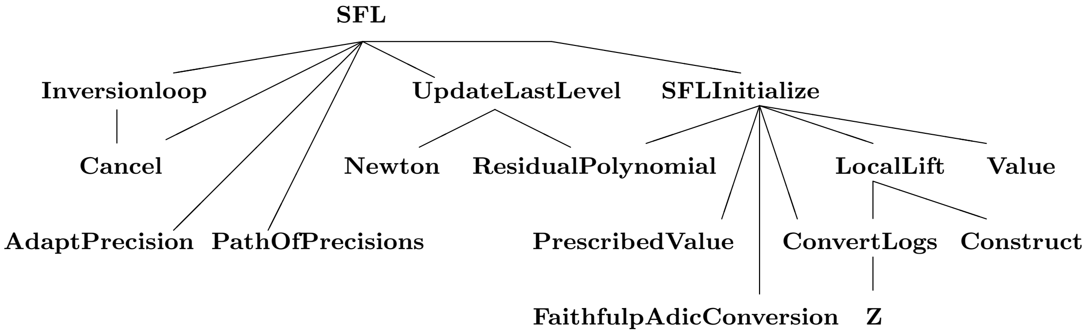
2.1 SFL(~P, slope: update:=false)

Input. slope is a positive integer.

Output. The Okutsu approximation ϕ_{r+1} at the last level of of P`Type is iteratively improved till we get P`Type[$r+1$]`slope $\geq$ slope.

If update is given the value true, then P`sfl, P`sflPols and the last level of P`Type are updated with data of the new Okutsu approximation.

Structure of subroutines.



2.2 SFLprecision(~P, precision: update:=false)

By an adequate call to SFL(~P, slope: update:=update), the Okutsu approximation ϕ_{r+1} at the last level of P`Type is improved to get $\phi_{r+1} \equiv F_{\mathtt{P}} \pmod{p^{\mathtt{precision}}}$, where $F_{\mathtt{P}}$ is the p-adic monic irreducible polynomial associated with P.

2.3 Cancel(poly, vden: QUO:=true)

Input.

- `poly` is a polynomial over a p-adic ring or a quotient of a p-adic ring.
- `den` is a non-negative integer.

Output. Two objects `outpoly, outvden`.

- `outpoly` is a polynomial over a p-adic ring or a quotient of a p-adic ring.
- `outvden` is a non-negative integer.

The input represents the polynomial $\texttt{poly}/p^{\texttt{vden}}$. The output is the same polynomial represented again by a pair `outpoly, outvden`, for which the highest possible power of p has been cancelled out from numerator and denominator.

The Boolean variable `QUO` tells if we work over a p-adic ring (`QUO:=false`) or a quotient of a p-adic ring (`QUO:=true`). It is necessary to distinguish these situations because they require a different management of the precision loss caused by the division by a power of the uniformizing element.

2.4 Inversionloop(A, ~xnum, ~xden, phi, precision, Zp)

Input.

- `Zp` is a p-adic ring.
- `phi` is a polynomial with coefficients in a certain quotient of `Zp`. Let $L = \mathbb{Q}_p(\alpha)$ be the extension obtained by adjoining a root α of `phi`.
- `A=[* anum,aden *]` represents the element $\texttt{A} = \texttt{anum}(\alpha)p^{\texttt{aden}} \in L$.
- `precision` is a positive integer.
- `xnum, xden` represents $\texttt{x} = \texttt{xnum}(\alpha)p^{\texttt{xden}} \in L$ such that $\texttt{A*x} \equiv 1 \pmod{(\mathfrak{m}_L)^h}$, for h satisfying `precision=2*exponent+Ceiling(2h/`$e(L/\mathbb{Q}_p)$`)`, where `exponent` is the least exponent δ such that $p^\delta \mathbb{Z}_L \subset \mathbb{Z}_p[\alpha]$.

Output. A pair `xnum, xden` such that $\texttt{A*x} \equiv 1 \pmod{(\mathfrak{m}_L)^{2h}}$.

The routine applies one iteration, $x_{n+1} = x_n(2 - \texttt{A}x_n)$, of the classical p-adic Newton method to find an approximation to `1/A`.

2.5 AdaptPrecision(Zp, pol, llista)

The variable `Zp` is a p-adic ring and `pol` is the p-adic conversion of a certain polynomial $f(x)$ with integer coefficients. The variable `llista` keeps trace of the indices of the negative coefficients of $f(x)$.

The routine outputs the p-adic conversion of $f(x)$ with a higher precision, stored as ``Zp`DefaultPrecision``. If the output polynomial is reconverted to $\mathbb{Z}[x]$ one recovers the original polynomial $f(x)$. A simple call to `ChangePrecision` would loose this property.

2.6 FaithfulpAdicConversion(pol, p)

The polynomial `pol` $\in \mathbb{Z}[x]$ is converted into a p-adic polynomial with a sufficiently high precision, so that the reconversion to a polynomial in $\mathbb{Z}[x]$ would recover `pol`.

The output is a pair `polZp, negcoeffs`, where `polZp` is the p-adic conversion of `pol` and `negcoeffs` is a list of the indices of the negative coefficients of `pol`. This list is necessary when we need to adapt `polZp` to a higher precision by calling `AdaptPrecision`.

2.7 PathOfPrecisions(n, h)

Computes a list of precisions $[h_1, \dots, h_t]$, with $1 \le h_1 \le h$, $h_{i+1} \in \{2h_i, 2h_i - 1\}$ for all $1 \le i < t$, and $h_t = n$.

2.8 UpdateLastLevel(~P)

The value of `slope` at the last level of ``P`Type`` is updated. If this value is finite, the attributes `h`, `psi`, `logGamma` of the last level of ``P`Type`` and the attribute ``P`sflPols`` are updated as well.

2.9 SFLInitialize(~P)

The initial values of some attributes of the prime ideal `P` that are necessary to run `SFL` are computed:

``P`sflPols:=[*`` a_0,a_1`,PolZp,PsiZp,x0num *],`

``P`sfl:=[* x0prec,nu,signsPol,signsPsi,x0den *]``

The data a_0, a_1 had been computed and stored by a call to `LastLevel`, during the execution of `Montesloop`. Along the different calls to `SFL`, these data will be updated by `UpdateLastLevel`.

`PolZp` is the p-adic conversion of the defining polynomial $f(x)$ of K (or the polynomial ``P`Pol`` if `P` has been constructed by a call to `pAdicFactors`).

`nu` is a non-negatve integer, and `PsiZp` is the p-adic conversion of a polynomial $\Psi \in \mathbb{Z}[x]$ such that $a_1(\theta)\Psi(\theta)/p^{\mathtt{nu}}$ has v-value zero. These two data will not change along the different calls to `SFL`.

`signsPol` and `signsPsi` store the indices of the negative coefficients of $f(x)$ and Ψ, respectively.

Finally, `x0num` is a polynomial and `x0den` an exponent, such that:

$$\left(a_1(\theta)\Psi(\theta)/p^{\mathtt{nu}}\right)\left(\mathtt{x0num}(\theta)/p^{\mathtt{x0den}}\right) \equiv 1 \pmod{p^{\mathtt{x0prec}}}, \tag{1}$$

where `x0prec = 1` in this initialization step.

During the execution of SFL, the values of `x0num`, `x0den` will be modified in order to satisfy (1) for an adequate increased precision `x0prec`.

2.10 LocalLift(class, P)

Input. `class` belongs to the residue field of `P`, stored in ``P`Type[#P`Type]`Fq``.

Output. A `P`-integral element of K of the form $g(\theta)/p^\nu$, whose class modulo `P` is `class`. The degree of $g(x)$ is less than $n_{\mathtt{P}}$ =`e(P/p)f(P/p)`; therefore, $\nu \le$``P`exponent``.

2.11 LocalLift(∼P, class, ∼numlift, ∼denlift)

Inner routine to compute local liftings. The output variables `numlift` $= g(x) \in \mathbb{Z}[x]$, `denlift` $= \nu$, yield an element $g(\theta)/p^{\nu}$ whose class modulo $\mathfrak{p}$ is `class`.

2.12 ConvertLogs(∼P, log, ∼class)

Input. `log` $= (\ell_0, \dots, \ell_i) \in \mathbb{Z}^{i+1}$, $0 \le i \le r+1$, such that $v_{\mathfrak{p}}\left(p^{\ell_0}\phi_1(\theta)^{\ell_1}\cdots\phi_i(\theta)^{\ell_i}\right) = 0$.

Output. `class` belongs to the finite field `P`Type[`$i$`]`Fq` and it is the class of $p^{\ell_0}\phi_1(\theta)^{\ell_1}\cdots\phi_i(\theta)^{\ell_i}$ modulo $\mathfrak{p}$.

2.13 Z(∼type, i, ∼z)

Input.

- `type = P`Type`, for a certain prime ideal `P` of `K`.
- `i` is an integer, $0 \le$ `i` $\le r+1$.

Output. `z` $= z_i$.

For $0 \le$ `i` $\le r$, we take simply z_i =`type[i+1]`z`. Since `type` has no $(r+2)$-th level, we compute $z_{r+1}$ =`-Coefficient(type[r]`psi,0)` every time we need it.

3 p-adic Factoritzation

A combination of `Montes` and `SFL` yields a p-adic factorization routine. As an application, we obtain routines for the computation of the p-adic valuation of discriminants and resultants for polynomials in $\mathbb{Z}[x]$ which are not necessarily irreducible [8].

3.1 pAdicFactors(poly, p, precision)

Input.

- `poly` is a squarefree monic polynomial in $\mathbb{Z}[x]$.
- `precision` is a non-negative integer.

Output. A list of Okutsu approximations to the irreducible `p`-adic factors of `poly`, all of them correct modulo `p^precision`.

The routine detects if `poly` is not squarefree and it displays a warning message.

3.2 pDiscriminant(poly, p)

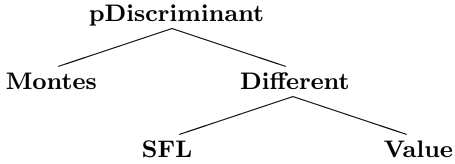
Input. `poly` is a monic polynomial in $\mathbb{Z}[x]$.

Output. Two objects `pdisf`, `pdisccK`.

- `pdiscf` is the `p`-adic valuation of the discriminant of `poly`.
- `pdiscK` is the sum of the `p`-adic valuations of the discriminants of all local extensions $L_G/\mathbb{Q}_{\mathtt{p}}$, where G runs on the irreducible `p`-adic factors of `poly`, and L_G is the local extension determined by G.

The routine detects if `poly` is not squarefree and it displays a warning message.

Structure of subroutines.



3.3 Different(∼P, ∼different)

Input. `P` is a prime ideal associated with a monic irreducible p-adic polynomial $G(x)$.

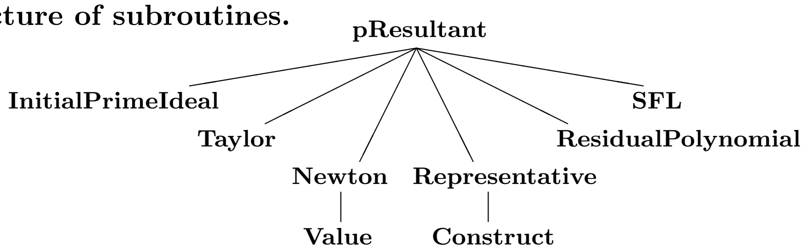
Output. `different` is the $\mathfrak{m}$-adic valuation of the different ideal of $L_G/\mathbb{Q}_{\mathtt{p}}$, where $\mathfrak{m}$ is the maximal ideal of the ring of integers of L_G.

3.4 pResultant(poly, poly2, p)

Input. `poly` and `poly2` are monic polynomials in $\mathbb{Z}[x]$.

Output. The `p`-adic valuation of the resultant of the two polynomials.

Structure of subroutines.



4 $\mathfrak{p}$-adic valuations

4.1 Localize(alpha, p)

Input. `alpha` $=: \alpha$ is a arbitrary element in the number field K.

Output. Three objects `den, exp, g`.

- `den, exp` are integers, and `den` is not divisible by `p`.
- `g` $\in \mathbb{Z}[x]$ is a polynomial such that $g \notin p\mathbb{Z}[x]$ and `alpha` $= p^{\mathtt{exp}}\mathtt{g}(\theta)/\mathtt{den}$.

4.2 EqualizeLogs(∼log1,∼log2)

The shorter logarithm is enlarged with zeros till both logarithms have the same length.

4.3 PValuation(alpha, P: RED:=false)

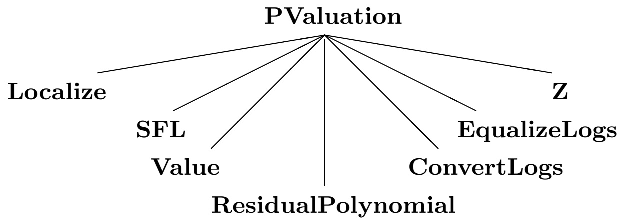
Input. `alpha` $=: \alpha$ is a arbitrary element in the number field K.

Output. Two objects `val, class`.

- `val` is $v_{\mathtt{P}}(\alpha)$

- If `RED = true`, the second output is the class modulo `P` of $\alpha\,\pi^{-\texttt{val}}$, where π is the local generator stored in `P`LocalGenerator`.

Structure of subroutines.



4.4 IsIntegralM(alpha)

Input. `alpha` $=: \alpha$ is an algebraic number.

Output. `true` if and only if `alpha` is integral.

5 Reduction modulo powers of prime ideals

5.1 ResidueField(P)

It outputs `P`Type[#P`Type]`Fq`, the computational representation of the residue field $\mathbb{Z}_K/$`P` of the prime ideal `P`.

5.2 Reduction(alpha, P, m)

Input.

- `alpha` $=: \alpha$ is a $\mathfrak{p}$-integral element of K
- `m` is a positive integer.

Output. The computational representation of the class of α modulo $\mathtt{P}^m$. That is, a list $[c_0, ..., c_{m-1}]$ of elements in the residue field of `P`, uniquely determined by:

$$\alpha \equiv a_0 + a_1\pi + \cdots + a_{m-1}\pi^{m-1} \pmod{\mathtt{P}^m}, \quad a_i := \mathtt{LocalLift}(c_i\mathtt{,P}),$$

where π is the local generator of `P`.

5.3 Reduction(alpha, P)

Equivalent to `Reduction(alpha, P, 1)[1]`.

5.4 LocalLift(class, P, m)

Input.

- `m` is a positive integer.
- `class` is a class modulo $\mathtt{P}^m$; i.e. a list $[c_0, ..., c_{m-1}]$ of m elements in the residue field of `P`.

Output. A `P`-integral element α in `K` whose class modulo $\mathtt{P}^m$ is equal to `class`.

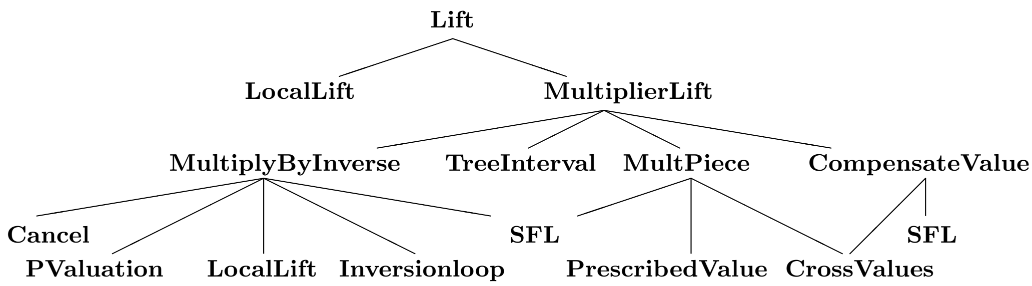
5.5 Lift(class, P, m)

Input.

- m is a positive integer.
- class is a class modulo $\mathtt{P}^m$; i.e. a list $[c_0, ..., c_{m-1}]$ of m elements in the residue field of P.

Output. An algebraic integer $\alpha \in \mathbb{Z}_K$ whose class modulo $\mathtt{P}^m$ is equal to class.

Structure of subroutines.



5.6 MultiplierLift(∼P, exponents, ∼mult)

Input. exponents $= [(a_{\mathtt{q}})_{\mathtt{q}|p}]$ is a list of non-negative integers, one for each prime ideal in K`PrimeIdeals[p], where p is the prime number beneath P.

Output. An algebraic integer mult satisfying:

$$\mathtt{mult} \equiv 1 \pmod{\mathtt{P}^{a_{\mathtt{P}}}}, \quad v_{\mathtt{q}}(\mathtt{mult}) \ge a_{\mathtt{q}}, \ \forall\, \mathtt{q} \ne \mathtt{P}.$$

5.7 TreeInterval(∼P, ∼tree)

It computes tree, the interval of positions in K`PrimeIdeals[p] of the tree to which P belongs. Here p is the prime number underlying P.

5.8 MultPiece(∼P, tree, expsTree, ∼N, ∼bp)

Input.

- tree is the interval of positions in K`PrimeIdeals[p] of the tree $\mathcal{T}$ to which P belongs. Here p is the prime number underlying $\mathfrak{p}$.
- expsTree $= [(a_{\mathtt{q}})_{\mathtt{q}\in\mathcal{T}}]$ is a list non-negative integers, one for each $\mathtt{q} \in \mathcal{T}$.

Output.

- An element bp in K satisfying:

$$v_{\mathtt{P}}(\mathtt{bp}) = 0, \ \ v_{\mathtt{q}}(\mathtt{bp}) \ge a_{\mathtt{q}}, \ \forall\, \mathtt{q} \in \mathcal{T},\, \mathtt{q} \ne \mathtt{P}.$$

- N is the p-valuation of the denominator of bp.

5.9 CompensateValue(K, p, tree, expsTree)

Input.

- `tree` is an interval of positions in `` K`PrimeIdeals[p] `` of a tree $\mathcal{T}$ of OM representations of the prime ideals of `K` over `p`.
- `expsTree` is a list of integers, one for each prime ideal of $\mathcal{T}$.

Output. A polynomial $g(x) \in \mathbb{Z}[x]$ such that $g(\theta) \in K$ satisfies:

$$v_{\mathfrak{p}}(g(\theta)) = 0,\ \forall\, \mathfrak{p} \notin \mathcal{T}; \quad v_{\mathfrak{q}}(g(\theta)) \geq \texttt{expsTree[i}_{\mathfrak{q}}\texttt{]},\ \forall\, \mathfrak{q} \in \mathcal{T},$$

where $1 \leq \texttt{i}_{\mathfrak{q}} \leq \#\texttt{tree}$ is the position of $\mathfrak{q}$ in $\mathcal{T}$.

5.10 MultiplyByInverse(~alpha, ~P, m)

Input.

- `alpha` $=: \alpha$ is a `P`-integral element of the number field `K`, having $v_{\texttt{P}}(\alpha) = 0$ and whose denominator is a power of p.
- `m` is a positive integer.

Output. α is replaced by $\alpha\alpha'$, where α' is `P`-integral and $\alpha\alpha' \equiv 1 \pmod{\texttt{P}^m}$.

6 Generators of prime ideals

Given a prime number p, we want to compute algebraic integers $\alpha_{\mathfrak{p}} \in \mathbb{Z}_K$, one for each prime ideal $\mathfrak{p} \mid p$, such that:

$$v_{\mathfrak{p}}(\alpha_{\mathfrak{p}}) = 1, \quad v_{\mathfrak{q}}(\alpha_{\mathfrak{p}}) = 0,\ \forall\, \mathfrak{q}|p,\ \mathfrak{q} \neq \mathfrak{p}. \tag{2}$$

Clearly, $\mathfrak{p} = p\mathbb{Z}_K + \alpha_{\mathfrak{p}}\mathbb{Z}_K$, but this property is weaker than (2). We abuse of language and we say that $\alpha_{\mathfrak{p}}$ is a *generator* of $\mathfrak{p}$.

6.1 Multipliers(K, p, values)

Input. `values`$= (a_{\mathfrak{p},\mathfrak{q}})$ is a square matrix of rational values indexed by the prime ideals dividing p.

Output. A list of multipliers $c_{\mathfrak{p}} \in \mathbb{Z}_K$ satisfying

$$v_{\mathfrak{p}}(c_{\mathfrak{p}}) = 0, \qquad v_{\mathfrak{q}}(c_{\mathfrak{p}}) \geq a_{\mathfrak{p},\mathfrak{q}},\ \forall\, \mathfrak{q} \mid p,\ \mathfrak{q} \neq \mathfrak{p}.$$

6.2 Generators(K, p)

A generator of each $\mathfrak{p} \mid$ `p` is computed. If $\mathfrak{p}$ is the `i`-th prime ideal over `p`, the generator is stored in `` K`PrimeIdeals[p,i]`Generator ``.

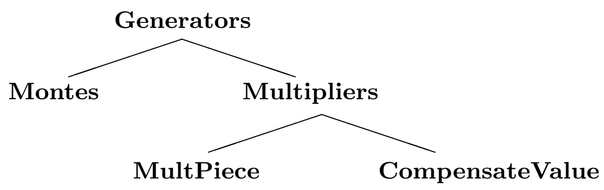
This routine calls `Multipliers(K, p, values)` for `values`$= (a_{\mathfrak{p},\mathfrak{q}})$, with

$$a_{\mathfrak{p},\mathfrak{q}} := \operatorname{Max}\{2, 1 + e(\mathfrak{q}/p)d_{\mathfrak{p}}\},$$

where $p^{d_{\mathfrak{p}}}$ is the denominator of the local generator of $\mathfrak{p}$.

The generator of $\mathfrak{p}$ is computed as $\alpha_{\mathfrak{p}} := c_{\mathfrak{p}}\pi_{\mathfrak{p}} + \sum_{\mathfrak{q}\neq\mathfrak{p}} c_{\mathfrak{q}}$.

Structure of subroutines.



7 Manipulation of fractional ideals

A fractional ideal is represented by an `IdealRecord` for which at least one of the two attributes `Generators` or `Factorization` has an assigned value.

We keep with the general meaning of the variables `K`, `p`, `P`, explained in section 1. Moreover, in all forthcoming routines:

- `I` is a non-zero fractional ideal of the number field `K`.

7.1 ideal(K, listgen)

Input. `listgen` is a list of elements in K.

Output. The ideal generated by the elements of `listgen`.

Calls of the form `ideal(K, a)`, where `a` is an element of K or an integer, are also admitted.

7.2 IsIdealRecord(I)

Self-explained.

7.3 IsPrimeIdeal(I)

Self-explained.

7.4 IsOne(I)

Self-explained.

7.5 IsZero(I)

Self-explained.

7.6 IsIntegral(I)

Self-explained.

7.7 I eq J

Self-explained.

7.8 alpha in J

Self-explained.

7.9 I subset J

Self-explained.

7.10 I*J

Self-explained.

7.11 I^n

Self-explained.

7.12 I/J

Self-explained.

7.13 I+J

Self-explained.

If neither `I` nor `J` have been factorized, we just output an ideal whose attribute `Generators` is given the value `I`Generators` $\cup$ `J`Generators`. In all other cases, the output is a factorized ideal.

7.14 Factorization(~I)

The factoritzation of `I` into a product of prime ideals is stored in `I`Factorization`.

7.15 Factorization(I)

It outputs `I`Factorization`.

7.16 FactorListToString(list)

Input. `list` is a list of triples `[p,j,e]` representing the factorization of a non-zero fractional ideal into a product of prime ideals.

Output. A literal expression where each triple is written as `P[p,j]^e`.

7.17 Norm(I)

It outputs $N_{K/\mathbb{Q}}(\mathtt{I})$.

7.18 PValuation(I, P)

It outputs $v_{\mathtt{P}}(\mathtt{I})$.

7.19 RationalRadical(I)

It outputs the list of prime numbers p admitting a prime ideal $\mathfrak{p} \mid p$ with $v_{\mathfrak{p}}(\mathtt{I}) \neq 0$.

7.20 RationalDenominator(I)

It outputs the least positive integer `a` such that `aI` is an integral ideal.

7.21 TwoElement(∼I)

The attributes `I`IntegerGenerator`, `I`Generator` are given a value.

7.22 TwoElement(I)

It outputs the list `[I`IntegerGenerator, I`Generator];`

8 Integral bases

8.1 pIntegralBasis(I, p: HNF:=false, Separated:=false)

It outputs a triangular p-integral basis of `I`, computed by the MaxMin algorithm [9].

If `HNF` is given the value `true` the Hermite normal form of the basis is computed.

If `Separated` is given the value `true` the basis is computed in the form of a triple output `nums, dexp, a`, where `nums` is a list of polynomials with integer coefficients, `dexp` is a list of integers and `a` is an integer. The `i`-th element of the basis is `num[i]`(θ) times $p^{\mathtt{a}-\mathtt{dexp[i]}}$.

8.2 pIntegralBasis(K, p: HNF:=false)

The output is a triangular (HNF) p-integral basis of the maximal order of K.

8.3 reduceIdeal(I, p: exponents:=false)

Double output `J, a`, where `J` is a fractional ideal with support in the prime ideals dividing p, and `a` is an integer such that the p-part of `I` is p^a`J`.

If `exponents` is given the value `true` then the output is `Exps, a`, where `Exps` is the list of all $v_{\mathfrak{p}}$(`J`) for the different prime ideals $\mathfrak{p}$ dividing p.

8.4 IdealBasis(I: HNF:=false, Separated:=false)

Computes a triangular basis of the fractional ideal `I` as a $\mathbb{Z}$-module.

If `HNF` is given the value `true` the Hermite normal form of the basis is computed.

If `Separated` is given the value `true` the basis is computed in the form of a triple output `nums, dens, factor`, where `nums` is a list of polynomials with integer coefficients, `dens` is a list of integers and `factor` is a rational number. The `i`-th element of the basis is `factor`$\cdot$`nums[i]`(θ)`/dens[i]`.

8.5 SIdealBasis(I, S)

Computes an `S`-triangular basis of the fractional ideal `I` as a $\mathbb{Z}$-module, where `S` is a finite set of prime numbers.

8.6 IntegralBasis(K: HNF:=false)

The output is a triangular (HNF) integral basis of the maximal order of K, which has been stored in `K`IntegralBasis` too.

9 Chinese remainders theorem

9.1 LocalCRT(K, p, exponents)

Input. `exponents` $= \left[(a_{\mathfrak{p}})_{\mathfrak{p}|p}\right]$ is a list of non-negative integers, one for each prime ideal $\mathfrak{p} \mid p$.

Output. A list of algebraic integers, $\left[(b_{\mathfrak{p}})_{\mathfrak{p}|p}\right]$, one for each prime ideal $\mathfrak{p} \mid p$, such that:

$$b_{\mathfrak{p}} \equiv 1 \pmod{\mathfrak{p}^{a_{\mathfrak{p}}}}, \quad v_{\mathfrak{q}}(b_{\mathfrak{p}}) \geq a_{\mathfrak{q}}, \ \forall\, \mathfrak{q} \neq \mathfrak{p}.$$

9.2 CRT(elements, ideals)

Input.

- `elements` $= [(\alpha_j)_{1\leq j\leq r}]$ is a list of algebraic integers of a number field.
- `ideals` $= [(I_j)_{1\leq j\leq r}]$ is a list of the same length as `elements`, of pairwise coprime integral ideals.

Output. An algebraic integer α such that $\alpha \equiv \alpha_j \pmod{I_j}, \quad 1 \leq j \leq r$.

10 Construction of types

The aim of these routines is to construct polynomials with a prescribed behaviour at a given prime number p. The types constructed for this purpose do not have assigned values for all the attributes that are necessary to construct OM representations of prime ideals. The only relevant datum of these types is their last ϕ-polynomial.

We shall display a type as a list `type = [...,`$(\phi_i, \lambda_i, \psi_i)$`,...]` of the three fundamental invariants at each level. Also, we denote $Y_i =$`(type[`i``]`FqY).1`` and, as usual, $z_i =$`type[`$i+1$``]`z``.

10.1 InitializeType(p, psi)

Input. `psi` is a monic irreducible polynomial with coefficients in the prime field $\mathbb{Z}/p\mathbb{Z}$.

Output. Three objects `type, Y, z`.

- `type` is a list consisting of a single record `TypeLevel` for which the following attributes have an assigned value: `Prime:=p,V,Phi,Fq,prodf,FqY,z`.
- `Y = [*` Y_1 `*]`, `z = [*` z_0 `*]`.

`type = [`$(\phi_1, -, -)$`]` is a type of order zero, which is half-way in the process of being enlarged to a type of order one. The polynomial ϕ_1 is a monic lift of `psi` to $\mathbb{Z}[x]$.

10.2 EnlargeType(h, e, psi, ~type, ~Y, ~z)

Input.

- `type = [...,`$(\phi_{i-1}, \lambda_{i-1}, \psi_{i-1})$`,`$(\phi_i, -, -)$`]` is a type of order $i-1$.
- `Y = [*` $Y_1, \dots, Y_i$ `*]`, `z = [*` $z_0, \dots, z_{i-1}$ `*]`.

- `h` and `e` are coprime positive integers.
- `psi` is a monic irreducible polynomial with coefficients in the field $\mathbb{F}_i$ =`type[`i`]`Fq`.

Output.

- `type = [...,(`$\phi_i, \lambda_i, \psi_i$`),(`$\phi_{i+1}, -, -$`)]` has been enlarged to a type of order i, with $\lambda_i := -$`h/e` and $\psi_i :=$`psi`.
- `Y = [*` $Y_1, \dots, Y_i, Y_{i+1}$ `*]`, `z = [*` $z_0, \dots, z_{i-1}, z_i$ `*]`.

10.3 CreateType(p, list)

Input. `list = [`$h_1, e_1, f_1, h_2, e_2, f_2$`, ... ,`$h_i, e_i, f_i$`]` is a list of $3i$ positive integers such that h_j, e_j are coprime, for all $1 \le j \le i$.

Output. A type of order i, `[...,(`$\phi_i, \lambda_i, \psi_i$`),(`$\phi_{i+1}, -, -$`)]`, where $\lambda_j = -h_j/e_j$ and $\psi_j \in \mathbb{F}_j[y]$ are randomly chosen polynomials such that $\deg \psi_j = f_j$, for all $1 \le j \le i$.

10.4 CreateRandomType(p, r)

Input. `r` is a positive integer.

Output. A type `[...,(`$\phi_r, \lambda_r, \psi_r$`),(`$\phi_{r+1}, -, -$`)]` of order r.

The type is built as in `CreateType(p, list)`, but for randomly chosen positive integers h_j, e_j, f_j satisfying the following constrains for all $1 \le j \le r$:

$$\gcd(h_j, e_j) = 1, \quad 1 \le h_j \le 11, \quad 1 \le e_j \le 4, \quad 1 \le f_j \le 3.$$

10.5 CreateRandomMultipleTypePolynomial(p, k, r, s)

Input. `k, r, s` are positive integers.

Output. A monic irreducible polynomial in $\mathbb{Z}[x]$ of the form

$$f(x) = \varphi_1(x) \cdots \varphi_{\mathtt{k}}(x) + ap^{\mathtt{s}},$$

where each φ_j is the `(r+1)`-th ϕ-polynomial of a type built by `CreateRandomType(p, r)` and a is the least positive integer such that $f(x)$ is irreducible.

If `s` is sufficiently large, then $f(x)$ will have `k` irreducible `p`-adic factors of Okutsu depth approximately `r`. The Okutsu depth of a factor will be less than `r` if at least one of the random levels of the corresponding type has $e_j = f_j = 1$.

10.6 RandomMultiplicityType(p, r, s)

Input. `r, s` are positive integers.

Output. A monic irreducible polynomial in $\mathbb{Z}[x]$ of the form

$$f(x) = \phi_{i_1}(x) \cdots \phi_{i_{s-1}}(x) \phi_{\mathtt{r}}(x),$$

where each ϕ_{i_k} is a randomly chosen ϕ-polynomial of a fixed random type of order r.

10.7 CombineTypes(listoftypes)

Input. `listoftypes` is a list of types.

Output. A monic irreducible polynomial in $\mathbb{Z}[x]$ of the form

$$f(x) = \varphi_1(x)\cdots\varphi_k(x) + ap^{20},$$

where each φ_j is the last ϕ-polynomial of the j-th type in `listoftypes` and a is the least positive integer such that $f(x)$ is irreducible.

10.8 CombinePolynomialsWithDifferentPrimes(f1, p1, f2, p2, k)

Input.

- `p1, p2` are prime numbers.
- `f1, f2` are monic polynomials in $\mathbb{Z}[x]$, of the same degree.
- `k` is a positive integer.

Output. A monic irreducible polynomial $f(x) \in \mathbb{Z}[x]$ satisfying:

$$f(x) \equiv \mathtt{f1}(x) \ \left(\mathrm{mod}\ \mathtt{(p1)}^{\mathtt{k}}\right), \qquad f(x) \equiv \mathtt{f2}(x) \ \left(\mathrm{mod}\ \mathtt{(p2)}^{\mathtt{k}}\right).$$

References

[1] J. Guàrdia, J. Montes, E. Nart, *Okutsu invariants and Newton polygons*, Acta Arithmetica **145** (2010), 83–108.

[2] J. Guàrdia, J. Montes, E. Nart, *Higher Newton polygons in the computation of discriminants and prime ideal decomposition in number fields*, Journal de Théorie des Nombres de Bordeaux **23** (2011), no. 3, 667–696.

[3] J. Guàrdia, J. Montes, E. Nart, *Newton polygons of higher order in algebraic number theory*, Trans. Amer. Math. Soc. **364** (2012), no. 1, 361–416.

[4] J. Guàrdia, E. Nart, *Genetics of polynomials over local fields*, Proceedings of AGCT14, Contemporary Mathematics **637** (2015), 207–241.

[5] J. Guàrdia, E. Nart, S. Pauli, *Single-factor lifting and factorization of polynomials over local fields*, J. Symb. Comput. **47** (2012), 1318–1346.

[6] J. Guàrdia, J. Montes, E. Nart, *A new computational approach to ideal theory in number fields*, Foundations of Computational Mathematics **13** (2013), 729–762.

[7] K. Mahler, *An inequality for the discriminant of a polynomial*, Michigan Math. J. **11** (1964), 257–262.

[8] E. Nart, *Local computation of differents and discriminants*, Mathematics of Computation **83** (2014), no. 287, 1513–1534.

[9] H. D. Stainsby, *Triangular bases of integral closures*, arXiv: 1506.01904v1 [mathNT].