Contents

Chapter 1

Classes

1.1 poly.multiutil – utilities for multivariate polynomials

- **Classes**
  - **RingPolynomial**
  - **DomainPolynomial**
  - **UniqueFactorizationDomainPolynomial**
  - OrderProvider
  - NestProvider
  - PseudoDivisionProvider
  - GcdProvider
  - RingElementProvider
- **Functions**
  - **polynomial**

1.1.1 RingPolynomial

General polynomial with commutative ring coefficients.

Initialize (Constructor)

RingPolynomial(`coefficients`**:** ***terminit*****,** **`**`**`keywords`**:** ***dict*****)**
→ ***RingPolynomial***

The `keywords` must include:

coeffring a commutative ring (*CommutativeRing*)

number_of_variables the number of variables(*integer*)

order term order (*TermOrder*)

This class inherits **BasicPolynomial**, **OrderProvider**, **NestProvider** and **RingElementProvider**.

Attribute

order :
term order.

Methods

1.1.1.1 getRing

getRing(`self`) → ***Ring***

Return an object of a subclass of `Ring`, to which the polynomial belongs.
(This method overrides the definition in RingElementProvider)

1.1.1.2 getCoefficientRing

getCoefficientRing(`self`) → ***Ring***

Return an object of a subclass of `Ring`, to which the all coefficients belong.
(This method overrides the definition in RingElementProvider)

1.1.1.3 leading_variable

leading_variable(`self`) → ***integer***

Return the position of the leading variable (the leading term among all total degree one terms).
The leading term varies with term orders, so does the result. The term order can be specified via the attribute `order`.
(This method is inherited from NestProvider)

1.1.1.4 nest

nest(`self`, `outer`: ***integer***, `coeffring`: ***CommutativeRing***)
→ ***polynomial***

Nest the polynomial by extracting `outer` variable at the given position.
(This method is inherited from NestProvider)

1.1.1.5 unnest

nest(`self`, `q`: ***polynomial***, `outer`: ***integer***, `coeffring`: ***CommutativeRing***)
→ ***polynomial***

Unnest the nested polynomial `q` by inserting `outer` variable at the given position.
(This method is inherited from NestProvider)

1.1.2 DomainPolynomial

Polynomial with domain coefficients.

Initialize (Constructor)

DomainPolynomial(`coefficients`: ***terminit***, `**keywords`: ***dict***)
→ ***DomainPolynomial***

The `keywords` must include:

coeffring a commutative ring (*CommutativeRing*)

number_of_variables the number of variables(*integer*)

order term order (*TermOrder*)

This class inherits **RingPolynomial** and **PseudoDivisionProvider**.

Operations

operator	explanation
`f / g`	division (result is a rational function)

Methods

1.1.2.1 pseudo_divmod

pseudo_divmod(self, other: ***polynomial***) → ***polynomial***

Return Q, R polynomials such that:

$$d^{deg(self)-deg(other)+1} self = other \times Q + R$$

w.r.t. a fixed variable, where d is the leading coefficient of `other`.

The leading coefficient varies with term orders, so does the result. The term order can be specified via the attribute `order`.
(This method is inherited from PseudoDivisionProvider.)

1.1.2.2 pseudo_floordiv

pseudo_floordiv(self, other: ***polynomial***) → ***polynomial***

Return a polynomial Q such that

$$d^{deg(self)-deg(other)+1} self = other \times Q + R$$

w.r.t. a fixed variable, where d is the leading coefficient of `other` and R is a polynomial.

The leading coefficient varies with term orders, so does the result. The term order can be specified via the attribute `order`.
(This method is inherited from PseudoDivisionProvider.)

1.1.2.3 pseudo_mod

pseudo_mod(self, other: ***polynomial***) → ***polynomial***

Return a polynomial R such that

$$d^{deg(self)-deg(other)+1} \times self = other \times Q + R$$

where d is the leading coefficient of `other` and Q a polynomial.

The leading coefficient varies with term orders, so does the result. The term order can be specified via the attribute `order`.
(This method is inherited from PseudoDivisionProvider.)

1.1.2.4 exact_division

exact_division(self, other: ***polynomial***) → ***polynomial***

Return quotient of exact division.
(This method is inherited from PseudoDivisionProvider.)

1.1.3 UniqueFactorizationDomainPolynomial

Polynomial with unique factorization domain (UFD) coefficients.

Initialize (Constructor)

UniqueFactorizationDomainPolynomial(`coefficients`: ***terminit***, **`**keywords`: *dict***)
→ ***UniqueFactorizationDomainPolynomial***

The `keywords` must include:

coeffring a commutative ring (*CommutativeRing*)

number_of_variables the number of variables(*integer*)

order term order (*TermOrder*)

This class inherits **DomainPolynomial** and **GcdProvider**.

Methods

1.1.3.1 gcd

gcd(`self`, `other`: ***polynomial***) → ***polynomial***

Return gcd. The nested polynomials' gcd is used.
(This method is inherited from GcdProvider.)

1.1.3.2 resultant

resultant(`self`, `other`: ***polynomial***, `var`: ***integer***) → ***polynomial***

Return resultant of two polynomials of the same ring, with respect to the variable specified by its position `var`.

1.1.4 polynomial – factory function for various polynomials

polynomial(`coefficients`: ***terminit***, `coeffring`: ***CommutativeRing***, `number_of_variables`: ***integer***=None)
→ ***polynomial***

Return a polynomial.

†One can override the way to choose a polynomial type from a coefficient ring, by setting:
`special_ring_table[coeffring_type] = polynomial_type`
before the function call.

1.1.5 prepare_indeterminates – simultaneous declarations of indeterminates

prepare_indeterminates(`names`: ***string***, `ctx`: ***dict***, `coeffring`: ***CoefficientRing***=None)
→ ***None***

From space separated `names` of indeterminates, prepare variables representing the indeterminates. The result will be stored in `ctx` dictionary.

The variables should be prepared at once, otherwise wrong aliases of variables may confuse you in later calculation.

If an optional `coeffring` is not given, indeterminates will be initialized as integer coefficient polynomials.

Examples

```
>>> prepare_indeterminates("X Y Z", globals())
>>> Y
```

```
UniqueFactorizationDomainPolynomial({(0, 1, 0): 1})
```