Contents

Chapter 1

Classes

1.1 poly.multiutil – 多変数多項式に対し実用的なもの

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

1.1.1 RingPolynomial

可換環係数を持つ一般の多項式.

Initialize (Constructor)

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

`keywords` はなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項の位数 (*TermOrder*)

このクラスは **BasicPolynomial**, **OrderProvider**, **NestProvider** and **RingElementProvider** を継承する.

Attributes

order :
項の位数.

Methods

1.1.1.1 getRing

getRing(`self`) → ***Ring***

多項式が所属する `Ring` のサブクラスのオブジェクトを返す.
(このメソッドは RingElementProvider 内の定義をオーバーライドする)

1.1.1.2 getCoefficientRing

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

すべての係数が所属する `Ring` のサブクラスのオブジェクトを返す.
(このメソッドは RingElementProvider 内の定義をオーバーライドする)

1.1.1.3 leading_variable

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

主変数の位置を返す (主とは一つの項の全てを足した総位数が 1 番多いということ).
主項は結果として項の位数に変化する. 項の位数は指定された特性を通って `order` になり得る.
(このメソッドは NestProvider から継承される)

1.1.1.4 nest

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

与えられた位置の変数 `outer` を引用することにより多項式をネストしてください.
(このメソッドは NestProvider から継承される)

1.1.1.5 unnest

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

与えられた位置の変数 `outer` を挿入することによりネストされた多項式 `q` をアンネストします.
(このメソッドは NestProvider から継承されます)

1.1.2 DomainPolynomial

定義域の係数を持つ多項式.

Initialize (Constructor)

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

`keywords` はなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項の位数 (*TermOrder*)

このクラスは **RingPolynomial** と **PseudoDivisionProvider** を継承する.

Operations

operator	explanation
`f / g`	除算 (結果は有理数関数)

Methods

1.1.2.1 pseudo_divmod

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

以下となる多項式 Q, R を返す:

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

固定値として `other` の主係数である d.
結果として主係数は項の係数に変わる. 項の位数は指定された特性を通って `order` になり得る.
(このメソッドは PseudoDivisionProvider から継承される.)

1.1.2.2 pseudo_floordiv

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

以下となる多項式 Q を返す:

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

固定値として `other` の主係数 d と 多項式 R.

結果として主係数は項の位数に変わる. 項の位数は指定された特性を通って `order` になり得る.
(このメソッドは PseudoDivisionProvider から継承される.)

1.1.2.3 pseudo_mod

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

以下となる多項式 R を返す:

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

d は `other` の主係数で Q は多項式.
結果として主係数は項の位数に変わる. 項の位数は指定された特性を通って `order` になり得る.
(このメソッドは PseudoDivisionProvider から継承される.)

1.1.2.4 exact_division

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

強制的な除算で商を返す.
(このメソッドは PseudoDivisionProvider から継承される.)

1.1.3 UniqueFactorizationDomainPolynomial

一意分解聖域 (UFD) 係数を持つ多項式.

Initialize (Constructor)

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

`keywords` はなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項の位数 (*TermOrder*)

このクラスは **DomainPolynomial** と **GcdProvider** を継承する.

Methods

1.1.3.1 gcd

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

gcd を返す. ネストされた多項式の gcd が使われる.
(このメソッドは GcdProvider から継承される.)

1.1.3.2 resultant

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

その位置 `var` によって指定された変数に関連した, 同じ環である二つの多項式の結果を返す.

1.1.4 polynomial – さまざまな多項式のための関数工場

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

多項式を返す.

† 設定により, 係数環から多項式の型を選ぶための方法がオーバーライドされ得る:
`special_ring_table[coeffring_type] = polynomial_type`
その関数が呼ばれる前に.

1.1.5 prepare_indeterminates – 不定連立宣言

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

不定な `names` によって分けられた空間から, 不定の関数表現を用意する. 結果は辞書 `ctx` に格納される.

変数はすぐに用意されるべきである, さもなくば間違った変数のエイリアスが計算を遅くし混乱するだろう.

もし任意の `coeffring` が与えられなければ, 不定値は整数係数多項式として初期化される.

Examples

```
>>> prepare_indeterminates("X Y Z", globals())
>>> Y
UniqueFactorizationDomainPolynomial({(0, 1, 0): 1})
```