Sage Reference Manual: General Rings, Ideals, and Morphisms

Release 6.9

The Sage Development Team

October 13, 2015

1 Base Classes for Rings, Algebras and Fields **1**
1.1 Rings . . . 1
1.2 Abstract base class for commutative rings . . . 23
1.3 Abstract base class for commutative algebras . . . 23
1.4 Base class for Dedekind domains . . . 23
1.5 Abstract base class for Euclidean domains . . . 23
1.6 Abstract base class for integral domains . . . 23
1.7 Abstract base class for principal ideal domains . . . 24

2 Ideals **25**
2.1 Ideals of commutative rings . . . 25
2.2 Monoid of ideals in a commutative ring . . . 40
2.3 Ideals of non-commutative rings . . . 40

3 Ring Morphisms **43**
3.1 Homomorphisms of rings . . . 43
3.2 Space of homomorphisms between two rings . . . 55

4 Quotient Rings **57**
4.1 Quotient Rings . . . 57
4.2 Quotient Ring Elements . . . 68

5 Fraction Fields **73**
5.1 Fraction Field of Integral Domains . . . 73
5.2 Fraction Field Elements . . . 77

6 Utilities **83**
6.1 Big O for various types (power series, p-adics, etc.) . . . 83
6.2 Signed and Unsigned Infinities . . . 84
6.3 Miscellaneous utilities . . . 93
6.4 Asymptotic Expansions . . . 93
6.5 Indices and Tables . . . 113

7 Indices and Tables **115**

Bibliography **117**

CHAPTER

ONE

BASE CLASSES FOR RINGS, ALGEBRAS AND FIELDS

1.1 Rings

This module provides the abstract base class `Ring` from which all rings in Sage (used to) derive, as well as a selection of more specific base classes.

Warning: Those classes, except maybe for the lowest ones like `Ring`, `CommutativeRing`, `Algebra` and `CommutativeAlgebra`, are being progressively deprecated in favor of the corresponding categories. which are more flexible, in particular with respect to multiple inheritance.

The class inheritance hierarchy is:

- `Ring`
  - `Algebra`
  - `CommutativeRing`
    - `NoetherianRing`
    - `CommutativeAlgebra`
    - `IntegralDomain`
      - `DedekindDomain`
      - `PrincipalIdealDomain`

Subclasses of `PrincipalIdealDomain` are

- `EuclideanDomain`
- `Field`
  - `FiniteField`

Some aspects of this structure may seem strange, but this is an unfortunate consequence of the fact that Cython classes do not support multiple inheritance. Hence, for instance, `Field` cannot be a subclass of both `NoetherianRing` and `PrincipalIdealDomain`, although all fields are Noetherian PIDs.

(A distinct but equally awkward issue is that sometimes we may not know *in advance* whether or not a ring belongs in one of these classes; e.g. some orders in number fields are Dedekind domains, but others are not, and we still want to offer a unified interface, so orders are never instances of the `DedekindDomain` class.)

AUTHORS:

- David Harvey (2006-10-16): changed `CommutativeAlgebra` to derive from `CommutativeRing` instead of from `Algebra`.

- David Loeffler (2009-07-09): documentation fixes, added to reference manual.
- Simon King (2011-03-29): Proper use of the category framework for rings.
- Simon King (2011-05-20): Modify multiplication and _ideal_class_ to support ideals of non-commutative rings.

class sage.rings.ring.**Algebra**

Bases: sage.rings.ring.Ring

Generic algebra

characteristic()

Return the characteristic of this algebra, which is the same as the characteristic of its base ring.

See objects with the base_ring attribute for additional examples. Here are some examples that explicitly use the Algebra class.

EXAMPLES:

```
sage: A = Algebra(ZZ); A
<type 'sage.rings.ring.Algebra'>
sage: A.characteristic()
0
sage: A = Algebra(GF(7^3, 'a'))
sage: A.characteristic()
7
```

has_standard_involution()

Return True if the algebra has a standard involution and False otherwise. This algorithm follows Algorithm 2.10 from John Voight's $Identifying the Matrix Ring$. Currently the only type of algebra this will work for is a quaternion algebra. Though this function seems redundant, once algebras have more functionality, in particular have a method to construct a basis, this algorithm will have more general purpose.

EXAMPLES:

```
sage: B = QuaternionAlgebra(2)
sage: B.has_standard_involution()
True
sage: R.<x> = PolynomialRing(QQ)
sage: K.<u> = NumberField(x**2 - 2)
sage: A = QuaternionAlgebra(K,-2,5)
sage: A.has_standard_involution()
True
sage: L.<a,b> = FreeAlgebra(QQ,2)
sage: L.has_standard_involution()
Traceback (most recent call last):
...
NotImplementedError: has_standard_involution is not implemented for this algebra
```

class sage.rings.ring.**CommutativeAlgebra**

Bases: sage.rings.ring.CommutativeRing

Generic commutative algebra

is_commutative()

Return True since this algebra is commutative.

EXAMPLES:

Any commutative ring is a commutative algebra over itself:

```
sage: A = sage.rings.ring.CommutativeAlgebra
sage: A(ZZ).is_commutative()
True
sage: A(QQ).is_commutative()
True
```

Trying to create a commutative algebra over a non-commutative ring will result in a `TypeError`.

class `sage.rings.ring.`**CommutativeRing**

Bases: `sage.rings.ring.Ring`

Generic commutative ring.

extension(*poly*, *name=None*, *names=None*, *embedding=None*)

Algebraically extends self by taking the quotient `self[x] / (f(x))`.

INPUT:

- `poly` – A polynomial whose coefficients are coercible into `self`
- `name` – (optional) name for the root of f

Note: Using this method on an algebraically complete field does *not* return this field; the construction `self[x] / (f(x))` is done anyway.

EXAMPLES:

```
sage: R = QQ['x']
sage: y = polygen(R)
sage: R.extension(y^2 - 5, 'a')
Univariate Quotient Polynomial Ring in a over Univariate Polynomial Ring in x over Rational

sage: P.<x> = PolynomialRing(GF(5))
sage: F.<a> = GF(5).extension(x^2 - 2)
sage: P.<t> = F[]
sage: R.<b> = F.extension(t^2 - a); R
Univariate Quotient Polynomial Ring in b over Finite Field in a of size 5^2 with modulus b^2
```

fraction_field()

Return the fraction field of `self`.

EXAMPLES:

```
sage: R = Integers(389)['x,y']
sage: Frac(R)
Fraction Field of Multivariate Polynomial Ring in x, y over Ring of integers modulo 389
sage: R.fraction_field()
Fraction Field of Multivariate Polynomial Ring in x, y over Ring of integers modulo 389
```

frobenius_endomorphism(*n=1*)

INPUT:

- n – a nonnegative integer (default: 1)

OUTPUT:

The n-th power of the absolute arithmetic Frobenius endomorphism on this finite field.

EXAMPLES:

```
sage: K.<u> = PowerSeriesRing(GF(5))
sage: Frob = K.frobenius_endomorphism(); Frob
```

```
Frobenius endomorphism x |--> x^5 of Power Series Ring in u over Finite Field of size 5
sage: Frob(u)
u^5
```

We can specify a power:

```
sage: f = K.frobenius_endomorphism(2); f
Frobenius endomorphism x |--> x^(5^2) of Power Series Ring in u over Finite Field of size 5
sage: f(1+u)
1 + u^25
```

ideal_monoid()

Return the monoid of ideals of this ring.

EXAMPLES:

```
sage: ZZ.ideal_monoid()
Monoid of ideals of Integer Ring
sage: R.<x>=QQ[]; R.ideal_monoid()
Monoid of ideals of Univariate Polynomial Ring in x over Rational Field
```

is_commutative()

Return `True`, since this ring is commutative.

EXAMPLES:

```
sage: QQ.is_commutative()
True
sage: ZpCA(7).is_commutative()
True
sage: A = QuaternionAlgebra(QQ, -1, -3, names=('i','j','k')); A
Quaternion Algebra (-1, -3) with base ring Rational Field
sage: A.is_commutative()
False
```

krull_dimension()

Return the Krull dimension of this commutative ring.

The Krull dimension is the length of the longest ascending chain of prime ideals.

TESTS:

`krull_dimension` is not implemented for generic commutative rings. Fields and PIDs, with Krull dimension equal to 0 and 1, respectively, have naive implementations of `krull_dimension`. Orders in number fields also have Krull dimension 1:

```
sage: R = CommutativeRing(ZZ)
sage: R.krull_dimension()
Traceback (most recent call last):
...
NotImplementedError
sage: QQ.krull_dimension()
0
sage: ZZ.krull_dimension()
1
sage: type(R); type(QQ); type(ZZ)
<type 'sage.rings.ring.CommutativeRing'>
<class 'sage.rings.rational_field.RationalField_with_category'>
<type 'sage.rings.integer_ring.IntegerRing_class'>
```

All orders in number fields have Krull dimension 1, including non-maximal orders:

```
sage: K.<i> = QuadraticField(-1)
sage: R = K.maximal_order(); R
Maximal Order in Number Field in i with defining polynomial x^2 + 1
sage: R.krull_dimension()
1
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.is_maximal()
False
sage: R.krull_dimension()
1
```

class sage.rings.ring.**DedekindDomain**

Bases: sage.rings.ring.IntegralDomain

Generic Dedekind domain class.

A Dedekind domain is a Noetherian integral domain of Krull dimension one that is integrally closed in its field of fractions.

This class is deprecated, and not actually used anywhere in the Sage code base. If you think you need it, please create a category DedekindDomains, move the code of this class there, and use it instead.

integral_closure()

Return self since Dedekind domains are integrally closed.

EXAMPLES:

```
sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.integral_closure()
Maximal Order in Number Field in s with defining polynomial x^2 + 1
sage: OK.integral_closure() == OK
True

sage: QQ.integral_closure() == QQ
True
```

is_integrally_closed()

Return True since Dedekind domains are integrally closed.

EXAMPLES:

The following are examples of Dedekind domains (Noetherian integral domains of Krull dimension one that are integrally closed over its field of fractions).

```
sage: ZZ.is_integrally_closed()
True
sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.is_integrally_closed()
True
```

These, however, are not Dedekind domains:

```
sage: QQ.is_integrally_closed()
True
sage: S = ZZ[sqrt(5)]; S.is_integrally_closed()
False
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
```

```
sage: T.is_integral_domain()
True
```

is_noetherian()

Return `True` since Dedekind domains are Noetherian.

EXAMPLES:

The integers, **Z**, and rings of integers of number fields are Dedekind domains:

```
sage: ZZ.is_noetherian()
True
sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.is_noetherian()
True
sage: QQ.is_noetherian()
True
```

krull_dimension()

Return 1 since Dedekind domains have Krull dimension 1.

EXAMPLES:

The following are examples of Dedekind domains (Noetherian integral domains of Krull dimension one that are integrally closed over its field of fractions):

```
sage: ZZ.krull_dimension()
1
sage: K = NumberField(x^2 + 1, 's')
sage: OK = K.ring_of_integers()
sage: OK.krull_dimension()
1
```

The following are not Dedekind domains but have a `krull_dimension` function:

```
sage: QQ.krull_dimension()
0
sage: T.<x,y> = PolynomialRing(QQ,2); T
Multivariate Polynomial Ring in x, y over Rational Field
sage: T.krull_dimension()
2
sage: U.<x,y,z> = PolynomialRing(ZZ,3); U
Multivariate Polynomial Ring in x, y, z over Integer Ring
sage: U.krull_dimension()
4
```

```
sage: K.<i> = QuadraticField(-1)
sage: R = K.order(2*i); R
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.is_maximal()
False
sage: R.krull_dimension()
1
```

class sage.rings.ring.**EuclideanDomain**

Bases: sage.rings.ring.PrincipalIdealDomain

Generic Euclidean domain class.

This class is deprecated. Please use the `EuclideanDomains` category instead.

parameter()
Return an element of degree 1.

EXAMPLES:

```
sage: R.<x>=QQ[]
sage: R.parameter()
x
```

class sage.rings.ring.**Field**
Bases: sage.rings.ring.PrincipalIdealDomain

Generic field

algebraic_closure()
Return the algebraic closure of self.

Note: This is only implemented for certain classes of field.

EXAMPLES:

```
sage: K = PolynomialRing(QQ,'x').fraction_field(); K
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: K.algebraic_closure()
Traceback (most recent call last):
...
NotImplementedError: Algebraic closures of general fields not implemented.
```

divides(*x*, *y*, *coerce=True*)
Return True if x divides y in this field (usually True in a field!). If coerce is True (the default), first coerce x and y into self.

EXAMPLES:

```
sage: QQ.divides(2, 3/4)
True
sage: QQ.divides(0, 5)
False
```

fraction_field()
Return the fraction field of self.

EXAMPLES:

Since fields are their own field of fractions, we simply get the original field in return:

```
sage: QQ.fraction_field()
Rational Field
sage: RR.fraction_field()
Real Field with 53 bits of precision
sage: CC.fraction_field()
Complex Field with 53 bits of precision

sage: F = NumberField(x^2 + 1, 'i')
sage: F.fraction_field()
Number Field in i with defining polynomial x^2 + 1
```

ideal(**gens*, ***kwds*)
Return the ideal generated by gens.

EXAMPLES:

```
sage: QQ.ideal(2)
Principal ideal (1) of Rational Field
sage: QQ.ideal(0)
Principal ideal (0) of Rational Field
```

integral_closure()
Return this field, since fields are integrally closed in their fraction field.

EXAMPLES:

```
sage: QQ.integral_closure()
Rational Field
sage: Frac(ZZ['x,y']).integral_closure()
Fraction Field of Multivariate Polynomial Ring in x, y over Integer Ring
```

is_field(*proof=True*)
Return `True` since this is a field.

EXAMPLES:

```
sage: Frac(ZZ['x,y']).is_field()
True
```

is_integrally_closed()
Return `True` since fields are trivially integrally closed in their fraction field (since they are their own fraction field).

EXAMPLES:

```
sage: Frac(ZZ['x,y']).is_integrally_closed()
True
```

is_noetherian()
Return `True` since fields are Noetherian rings.

EXAMPLES:

```
sage: QQ.is_noetherian()
True
```

krull_dimension()
Return the Krull dimension of this field, which is 0.

EXAMPLES:

```
sage: QQ.krull_dimension()
0
sage: Frac(QQ['x,y']).krull_dimension()
0
```

prime_subfield()
Return the prime subfield of `self`.

EXAMPLES:

```
sage: k = GF(9, 'a')
sage: k.prime_subfield()
Finite Field of size 3
```

class sage.rings.ring.**IntegralDomain**
Bases: sage.rings.ring.CommutativeRing

Generic integral domain class.

This class is deprecated. Please use the `sage.categories.integral_domains.IntegralDomains` category instead.

is_field(*proof=True*)

Return `True` if this ring is a field.

EXAMPLES:

```
sage: GF(7).is_field()
True
```

The following examples have their own `is_field` implementations:

```
sage: ZZ.is_field(); QQ.is_field()
False
True
sage: R.<x> = PolynomialRing(QQ); R.is_field()
False
```

An example where we raise a `NotImplementedError`:

```
sage: R = IntegralDomain(ZZ)
sage: R.is_field()
Traceback (most recent call last):
...
NotImplementedError
```

is_integral_domain(*proof=True*)

Return `True`, since this ring is an integral domain.

(This is a naive implementation for objects with type `IntegralDomain`)

EXAMPLES:

```
sage: ZZ.is_integral_domain()
True
sage: QQ.is_integral_domain()
True
sage: ZZ['x'].is_integral_domain()
True
sage: R = ZZ.quotient(ZZ.ideal(10)); R.is_integral_domain()
False
```

is_integrally_closed()

Return `True` if this ring is integrally closed in its field of fractions; otherwise return `False`.

When no algorithm is implemented for this, then this function raises a `NotImplementedError`.

Note that `is_integrally_closed` has a naive implementation in fields. For every field F, F is its own field of fractions, hence every element of F is integral over F.

EXAMPLES:

```
sage: ZZ.is_integrally_closed()
True
sage: QQ.is_integrally_closed()
True
sage: QQbar.is_integrally_closed()
True
sage: GF(5).is_integrally_closed()
True
```

```
sage: Z5 = Integers(5); Z5
Ring of integers modulo 5
sage: Z5.is_integrally_closed()
Traceback (most recent call last):
...
AttributeError: 'IntegerModRing_generic_with_category' object has no attribute 'is_integrall
```

class sage.rings.ring.**NoetherianRing**
Bases: sage.rings.ring.CommutativeRing

Generic Noetherian ring class.

A Noetherian ring is a commutative ring in which every ideal is finitely generated.

This class is deprecated, and not actually used anywhere in the Sage code base. If you think you need it, please create a category NoetherianRings, move the code of this class there, and use it instead.

is_noetherian()
Return True since this ring is Noetherian.

EXAMPLES:

```
sage: ZZ.is_noetherian()
True
sage: QQ.is_noetherian()
True
sage: R.<x> = PolynomialRing(QQ)
sage: R.is_noetherian()
True
```

class sage.rings.ring.**PrincipalIdealDomain**
Bases: sage.rings.ring.IntegralDomain

Generic principal ideal domain.

This class is deprecated. Please use the PrincipalIdealDomains category instead.

class_group()
Return the trivial group, since the class group of a PID is trivial.

EXAMPLES:

```
sage: QQ.class_group()
Trivial Abelian group
```

content(*x*, *y*, *coerce=True*)
Return the content of x and y, i.e. the unique element c of self such that x/c and y/c are coprime and integral.

EXAMPLES:

```
sage: QQ.content(ZZ(42), ZZ(48)); type(QQ.content(ZZ(42), ZZ(48)))
6
<type 'sage.rings.rational.Rational'>
sage: QQ.content(1/2, 1/3)
1/6
sage: factor(1/2); factor(1/3); factor(1/6)
2^-1
3^-1
2^-1 * 3^-1
sage: a = (2*3)/(7*11); b = (13*17)/(19*23)
sage: factor(a); factor(b); factor(QQ.content(a,b))
```

```
2 * 3 * 7^-1 * 11^-1
13 * 17 * 19^-1 * 23^-1
7^-1 * 11^-1 * 19^-1 * 23^-1
```

Note the changes to the second entry:

```
sage: c = (2*3)/(7*11); d = (13*17)/(7*19*23)
sage: factor(c); factor(d); factor(QQ.content(c,d))
2 * 3 * 7^-1 * 11^-1
7^-1 * 13 * 17 * 19^-1 * 23^-1
7^-1 * 11^-1 * 19^-1 * 23^-1
sage: e = (2*3)/(7*11); f = (13*17)/(7^3*19*23)
sage: factor(e); factor(f); factor(QQ.content(e,f))
2 * 3 * 7^-1 * 11^-1
7^-3 * 13 * 17 * 19^-1 * 23^-1
7^-3 * 11^-1 * 19^-1 * 23^-1
```

gcd (*x*, *y*, *coerce=True*)

Return the greatest common divisor of `x` and `y`, as elements of `self`.

EXAMPLES:

The integers are a principal ideal domain and hence a GCD domain:

```
sage: ZZ.gcd(42, 48)
6
sage: 42.factor(); 48.factor()
2 * 3 * 7
2^4 * 3
sage: ZZ.gcd(2^4*7^2*11, 2^3*11*13)
88
sage: 88.factor()
2^3 * 11
```

In a field, any nonzero element is a GCD of any nonempty set of nonzero elements. In previous versions, Sage used to return 1 in the case of the rational field. However, since trac ticket #10771, the rational field is considered as the *fraction field* of the integer ring. For the fraction field of an integral domain that provides both GCD and LCM, it is possible to pick a GCD that is compatible with the GCD of the base ring:

```
sage: QQ.gcd(ZZ(42), ZZ(48)); type(QQ.gcd(ZZ(42), ZZ(48)))
6
<type 'sage.rings.rational.Rational'>
sage: QQ.gcd(1/2, 1/3)
1/6
```

Polynomial rings over fields are GCD domains as well. Here is a simple example over the ring of polynomials over the rationals as well as over an extension ring. Note that `gcd` requires x and y to be coercible:

```
sage: R.<x> = PolynomialRing(QQ)
sage: S.<a> = NumberField(x^2 - 2, 'a')
sage: f = (x - a)*(x + a); g = (x - a)*(x^2 - 2)
sage: print f; print g
x^2 - 2
x^3 - a*x^2 - 2*x + 2*a
sage: f in R
True
sage: g in R
False
sage: R.gcd(f,g)
Traceback (most recent call last):
```

```
...
TypeError: Unable to coerce 2*a to a rational
sage: R.base_extend(S).gcd(f,g)
x^2 - 2
sage: R.base_extend(S).gcd(f, (x - a)*(x^2 - 3))
x - a
```

is_noetherian()

Every principal ideal domain is noetherian, so we return `True`.

EXAMPLES:

```
sage: Zp(5).is_noetherian()
True
```

class sage.rings.ring.**Ring**

Bases: sage.structure.parent_gens.ParentWithGens

Generic ring class.

TESTS:

This is to test against the bug fixed in trac ticket #9138:

```
sage: R.<x> = QQ[]
sage: R.sum([x,x])
2*x
sage: R.<x,y> = ZZ[]
sage: R.sum([x,y])
x + y
sage: TestSuite(QQ['x']).run(verbose=True)
running ._test_additive_associativity() . . . pass
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_category() . . . pass
running ._test_characteristic() . . . pass
running ._test_distributivity() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . pass
  running ._test_nonzero_equal() . . . pass
  running ._test_not_implemented_methods() . . . pass
  running ._test_pickling() . . . pass
  pass
running ._test_elements_eq_reflexive() . . . pass
running ._test_elements_eq_symmetric() . . . pass
running ._test_elements_eq_transitive() . . . pass
running ._test_elements_neq() . . . pass
running ._test_eq() . . . pass
running ._test_euclidean_degree() . . . pass
running ._test_gcd_vs_xgcd() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_quo_rem() . . . pass
running ._test_some_elements() . . . pass
running ._test_zero() . . . pass
running ._test_zero_divisors() . . . pass
```

```
sage: TestSuite(QQ['x','y']).run()
sage: TestSuite(ZZ['x','y']).run()
sage: TestSuite(ZZ['x','y']['t']).run()
```

Test againgsanother bug fixed in trac ticket #9944:

```
sage: QQ['x'].category()
Join of Category of euclidean domains and Category of commutative algebras over quotient fiel
sage: QQ['x','y'].category()
Join of Category of unique factorization domains and Category of commutative algebras over qu
sage: PolynomialRing(MatrixSpace(QQ,2),'x').category()
Category of algebras over (algebras over quotient fields and infinite sets)
sage: PolynomialRing(SteenrodAlgebra(2),'x').category()
Category of algebras over graded hopf algebras with basis over Finite Field of size 2
```

TESTS::

```
sage: Zp(7)._repr_option('element_is_atomic')
False
sage: QQ._repr_option('element_is_atomic')
True
sage: CDF._repr_option('element_is_atomic')
False
```

base_extend(*R*)

EXAMPLES:

```
sage: QQ.base_extend(GF(7))
Traceback (most recent call last):
...
TypeError: no base extension defined
sage: ZZ.base_extend(GF(7))
Finite Field of size 7
```

cardinality()

Return the cardinality of the underlying set.

OUTPUT:

Either an integer or `+Infinity`.

EXAMPLES:

```
sage: Integers(7).cardinality()
7
sage: QQ.cardinality()
+Infinity
```

category()

Return the category to which this ring belongs.

Note: This method exists because sometimes a ring is its own base ring. During initialisation of a ring *R*, it may be checked whether the base ring (hence, the ring itself) is a ring. Hence, it is necessary that `R.category()` tells that `R` is a ring, even *before* its category is properly initialised.

EXAMPLES:

```
sage: FreeAlgebra(QQ, 3, 'x').category() # todo: use a ring which is not an algebra!
Category of algebras with basis over Rational Field
```

Since a quotient of the integers is its own base ring, and during initialisation of a ring it is tested whether the base ring belongs to the category of rings, the following is an indirect test that the `category()` method of rings returns the category of rings even before the initialisation was successful:

```
sage: I = Integers(15)
sage: I.base_ring() is I
True
sage: I.category()
Join of Category of finite commutative rings
    and Category of subquotients of monoids
    and Category of quotients of semigroups
    and Category of finite enumerated sets
```

epsilon()

Return the precision error of elements in this ring.

EXAMPLES:

```
sage: RDF.epsilon()
2.220446049250313e-16
sage: ComplexField(53).epsilon()
2.22044604925031e-16
sage: RealField(10).epsilon()
0.0020
```

For exact rings, zero is returned:

```
sage: ZZ.epsilon()
0
```

This also works over derived rings:

```
sage: RR['x'].epsilon()
2.22044604925031e-16
sage: QQ['x'].epsilon()
0
```

For the symbolic ring, there is no reasonable answer:

```
sage: SR.epsilon()
Traceback (most recent call last):
...
NotImplementedError
```

ideal(**args*, ***kwds*)

Return the ideal defined by `x`, i.e., generated by `x`.

INPUT:

- `*x` – list or tuple of generators (or several input arguments)
- `coerce` – bool (default: `True`); this must be a keyword argument. Only set it to `False` if you are certain that each generator is already in the ring.
- `ideal_class` – callable (default: `self._ideal_class_()`); this must be a keyword argument. A constructor for ideals, taking the ring as the first argument and then the generators. Usually a subclass of `Ideal_generic` or `Ideal_nc`.
- Further named arguments (such as `side` in the case of non-commutative rings) are forwarded to the ideal class.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: R.ideal(x,y)
Ideal (x, y) of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ideal(x+y^2)
Ideal (y^2 + x) of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ideal( [x^3,y^3+x^3] )
Ideal (x^3, x^3 + y^3) of Multivariate Polynomial Ring in x, y over Rational Field
```

Here is an example over a non-commutative ring:

```
sage: A = SteenrodAlgebra(2)
sage: A.ideal(A.1,A.2^2)
Twosided Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis
sage: A.ideal(A.1,A.2^2,side='left')
Left Ideal (Sq(2), Sq(2,2)) of mod 2 Steenrod algebra, milnor basis
```

TESTS:

Make sure that trac ticket #11139 is fixed:

```
sage: R.<x> = QQ[]
sage: R.ideal([])
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
sage: R.ideal(())
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
sage: R.ideal()
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
```

ideal_monoid()

Return the monoid of ideals of this ring.

EXAMPLES:

```
sage: F.<x,y,z> = FreeAlgebra(ZZ, 3)
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = sage.rings.ring.Ring.quotient(F,I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Algebra on 3 generators (x, y, z) over Integer Ring by
sage: F.<x,y,z> = FreeAlgebra(ZZ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: Q.ideal_monoid()
Monoid of ideals of Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) ov
```

is_commutative()

Return `True` if this ring is commutative.

EXAMPLES:

```
sage: QQ.is_commutative()
True
sage: QQ['x,y,z'].is_commutative()
True
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -1,-1)
sage: Q.is_commutative()
False
```

is_exact()

Return `True` if elements of this ring are represented exactly, i.e., there is no precision loss when doing arithmetic.

Note: This defaults to `True`, so even if it does return `True` you have no guarantee (unless the ring has properly overloaded this).

EXAMPLES:

```
sage: QQ.is_exact()     # indirect doctest
True
sage: ZZ.is_exact()
True
sage: Qp(7).is_exact()
False
sage: Zp(7, type='capped-abs').is_exact()
False
```

is_field(*proof=True*)

Return `True` if this ring is a field.

INPUT:

- `proof` – (default: `True`) Determines what to do in unknown cases

ALGORITHM:

If the parameter `proof` is set to `True`, the returned value is correct but the method might throw an error. Otherwise, if it is set to `False`, the method returns True if it can establish that self is a field and False otherwise.

EXAMPLES:

```
sage: QQ.is_field()
True
sage: GF(9,'a').is_field()
True
sage: ZZ.is_field()
False
sage: QQ['x'].is_field()
False
sage: Frac(QQ['x']).is_field()
True
```

This illustrates the use of the `proof` parameter:

```
sage: R.<a,b> = QQ[]
sage: S.<x,y> = R.quo((b^3))
sage: S.is_field(proof = True)
Traceback (most recent call last):
...
NotImplementedError
sage: S.is_field(proof = False)
False
```

is_finite()

Return `True` if this ring is finite.

EXAMPLES:

```
sage: QQ.is_finite()
False
sage: GF(2^10,'a').is_finite()
True
sage: R.<x> = GF(7)[]
```

```
sage: R.is_finite()
False
sage: S.<y> = R.quo(x^2+1)
sage: S.is_finite()
True
```

is_integral_domain (*proof=True*)

Return `True` if this ring is an integral domain.

INPUT:

- `proof` – (default: `True`) Determines what to do in unknown cases

ALGORITHM:

If the parameter `proof` is set to `True`, the returned value is correct but the method might throw an error. Otherwise, if it is set to `False`, the method returns `True` if it can establish that self is an integral domain and `False` otherwise.

EXAMPLES:

```
sage: QQ.is_integral_domain()
True
sage: ZZ.is_integral_domain()
True
sage: ZZ['x,y,z'].is_integral_domain()
True
sage: Integers(8).is_integral_domain()
False
sage: Zp(7).is_integral_domain()
True
sage: Qp(7).is_integral_domain()
True
sage: R.<a,b> = QQ[]
sage: S.<x,y> = R.quo((b^3))
sage: S.is_integral_domain()
False
```

This illustrates the use of the `proof` parameter:

```
sage: R.<a,b> = ZZ[]
sage: S.<x,y> = R.quo((b^3))
sage: S.is_integral_domain(proof = True)
Traceback (most recent call last):
...
NotImplementedError
sage: S.is_integral_domain(proof = False)
False
```

TESTS:

Make sure trac ticket #10481 is fixed:

```
sage: var(x)
x
sage: R.<a>=ZZ[x].quo(x^2)
sage: R.fraction_field()
Traceback (most recent call last):
...
NotImplementedError
sage: R.is_integral_domain()
```

```
Traceback (most recent call last):
...
NotImplementedError
```

is_noetherian()
Return `True` if this ring is Noetherian.

EXAMPLES:

```
sage: QQ.is_noetherian()
True
sage: ZZ.is_noetherian()
True
```

is_prime_field()
Return `True` if this ring is one of the prime fields $\mathbf{Q}$ or $\mathbf{F}_p$.

EXAMPLES:

```
sage: QQ.is_prime_field()
True
sage: GF(3).is_prime_field()
True
sage: GF(9,'a').is_prime_field()
False
sage: ZZ.is_prime_field()
False
sage: QQ['x'].is_prime_field()
False
sage: Qp(19).is_prime_field()
False
```

is_ring()
Return `True` since `self` is a ring.

EXAMPLES:

```
sage: QQ.is_ring()
True
```

is_subring(*other*)
Return `True` if the canonical map from `self` to `other` is injective.

Raises a `NotImplementedError` if not known.

EXAMPLES:

```
sage: ZZ.is_subring(QQ)
True
sage: ZZ.is_subring(GF(19))
False
```

one()
Return the one element of this ring (cached), if it exists.

EXAMPLES:

```
sage: ZZ.one()
1
sage: QQ.one()
1
```

```
sage: QQ['x'].one()
1
```

The result is cached:

```
sage: ZZ.one() is ZZ.one()
True
```

order()

The number of elements of `self`.

EXAMPLES:

```
sage: GF(19).order()
19
sage: QQ.order()
+Infinity
```

principal_ideal(*gen*, *coerce=True*)

Return the principal ideal generated by gen.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: R.principal_ideal(x+2*y)
Ideal (x + 2*y) of Multivariate Polynomial Ring in x, y over Integer Ring
```

quo(*I*, *names=None*)

Create the quotient of R by the ideal I. This is a synonym for `quotient()`

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quo((x^2, y))
sage: S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False
```

quotient(*I*, *names=None*)

Create the quotient of this ring by a twosided ideal `I`.

INPUT:

- `I` – a twosided ideal of this ring, R.
- `names` – (optional) names of the generators of the quotient (if there are multiple generators, you can specify a single character string and the generators are named in sequence starting with 0).

EXAMPLES:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient(I, 'a')
sage: S.gens()
(a,)
```

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quotient((x^2, y))
sage: S
```

```
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False
```

quotient_ring(*I*, *names=None*)

Return the quotient of self by the ideal I of `self`. (Synonym for `self.quotient(I)`.)

INPUT:

- `I` – an ideal of R
- `names` – (optional) names of the generators of the quotient. (If there are multiple generators, you can specify a single character string and the generators are named in sequence starting with 0.)

OUTPUT:

- `R/I` – the quotient ring of R by the ideal I

EXAMPLES:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I, 'a')
sage: S.gens()
(a,)

sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quotient_ring((x^2, y))
sage: S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2, y)
sage: S.gens()
(a, 0)
sage: a == b
False
```

random_element(*bound=2*)

Return a random integer coerced into this ring, where the integer is chosen uniformly from the interval `[-bound,bound]`.

INPUT:

- `bound` – integer (default: 2)

ALGORITHM:

Uses Python's randint.

TESTS:

The following example returns a `NotImplementedError` since the generic ring class `__call__` function returns a `NotImplementedError`. Note that `sage.rings.ring.Ring.random_element` performs a call in the generic ring class by a random integer:

```
sage: R = sage.rings.ring.Ring(ZZ); R
<type 'sage.rings.ring.Ring'>
sage: R.random_element()
Traceback (most recent call last):
...
NotImplementedError
```

unit_ideal()
Return the unit ideal of this ring.

EXAMPLES:

```
sage: Zp(7).unit_ideal()
Principal ideal (1 + O(7^20)) of 7-adic Ring with capped relative precision 20
```

zero()
Return the zero element of this ring (cached).

EXAMPLES:

```
sage: ZZ.zero()
0
sage: QQ.zero()
0
sage: QQ['x'].zero()
0
```

The result is cached:

```
sage: ZZ.zero() is ZZ.zero()
True
```

zero_ideal()
Return the zero ideal of this ring (cached).

EXAMPLES:

```
sage: ZZ.zero_ideal()
Principal ideal (0) of Integer Ring
sage: QQ.zero_ideal()
Principal ideal (0) of Rational Field
sage: QQ['x'].zero_ideal()
Principal ideal (0) of Univariate Polynomial Ring in x over Rational Field
```

The result is cached:

```
sage: ZZ.zero_ideal() is ZZ.zero_ideal()
True
```

TESTS:

Make sure that trac ticket #13644 is fixed:

```
sage: K = Qp(3)
sage: R.<a> = K[]
sage: L.<a> = K.extension(a^2-3)
sage: L.ideal(a)
Principal ideal (1 + O(a^40)) of Eisenstein Extension of 3-adic Field with capped relative p
```

zeta(*n=2*, *all=False*)
Return an `n`-th root of unity in `self` if there is one, or raise an `ArithmeticError` otherwise.

INPUT:

- `n` – positive integer
- `all` – bool, default: False. If True, return a list of all n-th roots of 1.

OUTPUT:

Element of `self` of finite order

EXAMPLES:

```
sage: QQ.zeta()
-1
sage: QQ.zeta(1)
1
sage: CyclotomicField(6).zeta()
zeta6
sage: CyclotomicField(3).zeta()
zeta3
sage: CyclotomicField(3).zeta().multiplicative_order()
3
sage: a = GF(7).zeta(); a
3
sage: a.multiplicative_order()
6
sage: a = GF(49,'z').zeta(); a
z
sage: a.multiplicative_order()
48
sage: a = GF(49,'z').zeta(2); a
6
sage: a.multiplicative_order()
2
sage: QQ.zeta(3)
Traceback (most recent call last):
...
ValueError: no n-th root of unity in rational field
sage: Zp(7, prec=8).zeta()
3 + 4*7 + 6*7^2 + 3*7^3 + 2*7^5 + 6*7^6 + 2*7^7 + O(7^8)
```

zeta_order()

Return the order of the distinguished root of unity in `self`.

EXAMPLES:

```
sage: CyclotomicField(19).zeta_order()
38
sage: GF(19).zeta_order()
18
sage: GF(5^3,'a').zeta_order()
124
sage: Zp(7, prec=8).zeta_order()
6
```

sage.rings.ring.**is_Ring**(*x*)

Return `True` if `x` is a ring.

EXAMPLES:

```
sage: from sage.rings.ring import is_Ring
sage: is_Ring(ZZ)
True
sage: MS = MatrixSpace(QQ,2)
sage: is_Ring(MS)
True
```

1.2 Abstract base class for commutative rings

sage.rings.commutative_ring.**is_CommutativeRing**(*R*)
Check to see if R is a CommutativeRing.

EXAMPLES:

```
sage: sage.rings.commutative_ring.is_CommutativeRing(ZZ)
True
```

1.3 Abstract base class for commutative algebras

sage.rings.commutative_algebra.**is_CommutativeAlgebra**(*x*)
Check to see if x is a CommutativeAlgebra.

EXAMPLES:

```
sage: sage.rings.commutative_algebra.is_CommutativeAlgebra(sage.rings.ring.CommutativeAlgebra(ZZ
True
```

1.4 Base class for Dedekind domains

sage.rings.dedekind_domain.**is_DedekindDomain**(*R*)
Check to see if R is a DedekindDomain.

EXAMPLES:

```
sage: sage.rings.dedekind_domain.is_DedekindDomain(DedekindDomain(QQ))
True
```

1.5 Abstract base class for Euclidean domains

sage.rings.euclidean_domain.**is_EuclideanDomain**(*R*)
Check to see if R is a EuclideanDomain.

EXAMPLES:

```
sage: sage.rings.euclidean_domain.is_EuclideanDomain(EuclideanDomain(ZZ))
True
```

1.6 Abstract base class for integral domains

sage.rings.integral_domain.**is_IntegralDomain**(*R*)
Check if R is an instance of IntegralDomain.

EXAMPLES:

```
sage: sage.rings.integral_domain.is_IntegralDomain(QQ)
True
sage: sage.rings.integral_domain.is_IntegralDomain(ZZ)
True
```

1.7 Abstract base class for principal ideal domains

sage.rings.principal_ideal_domain.**is_PrincipalIdealDomain**(*R*)

Check if R is a PrincipalIdealDomain.

EXAMPLES:

```
sage: sage.rings.principal_ideal_domain.is_PrincipalIdealDomain(ZZ)
True
sage: R.<x,y> = QQ[]
sage: sage.rings.principal_ideal_domain.is_PrincipalIdealDomain(R)
False
```

CHAPTER

TWO

IDEALS

2.1 Ideals of commutative rings

Sage provides functionality for computing with ideals. One can create an ideal in any commutative or non-commutative ring R by giving a list of generators, using the notation `R.ideal([a,b,...])`. The case of non-commutative rings is implemented in `noncommutative_ideals`.

A more convenient notation may be `R*[a,b,...]` or `[a,b,...]*R`. If R is non-commutative, the former creates a left and the latter a right ideal, and `R*[a,b,...]*R` creates a two-sided ideal.

`sage.rings.ideal.`**`Cyclic`**(*R*, *n=None*, *homog=False*, *singular=Singular*)

Ideal of cyclic `n`-roots from 1-st `n` variables of `R` if `R` is coercible to `Singular`.

INPUT:

- `R` – base ring to construct ideal for
- `n` – number of cyclic roots (default: `None`). If `None`, then `n` is set to `R.ngens()`.
- `homog` – (default: `False`) if `True` a homogeneous ideal is returned using the last variable in the ideal
- `singular` – singular instance to use

Note: `R` will be set as the active ring in `Singular`

EXAMPLES:

An example from a multivariate polynomial ring over the rationals:

```
sage: P.<x,y,z> = PolynomialRing(QQ,3,order='lex')
sage: I = sage.rings.ideal.Cyclic(P)
sage: I
Ideal (x + y + z, x*y + x*z + y*z, x*y*z - 1) of Multivariate Polynomial
Ring in x, y, z over Rational Field
sage: I.groebner_basis()
[x + y + z, y^2 + y*z + z^2, z^3 - 1]
```

We compute a Groebner basis for cyclic 6, which is a standard benchmark and test ideal:

```
sage: R.<x,y,z,t,u,v> = QQ['x,y,z,t,u,v']
sage: I = sage.rings.ideal.Cyclic(R,6)
sage: B = I.groebner_basis()
sage: len(B)
45
```

sage.rings.ideal.**FieldIdeal**(*R*)

Let `q = R.base_ring().order()` and $(x_0, ..., x_n)$ `= R.gens()` then if q is finite this constructor returns

$$\langle x_0^q - x_0, ..., x_n^q - x_n \rangle.$$

We call this ideal the field ideal and the generators the field equations.

EXAMPLES:

The field ideal generated from the polynomial ring over two variables in the finite field of size 2:

```
sage: P.<x,y> = PolynomialRing(GF(2),2)
sage: I = sage.rings.ideal.FieldIdeal(P); I
Ideal (x^2 + x, y^2 + y) of Multivariate Polynomial Ring in x, y over
Finite Field of size 2
```

Another, similar example:

```
sage: Q.<x1,x2,x3,x4> = PolynomialRing(GF(2^4,name='alpha'), 4)
sage: J = sage.rings.ideal.FieldIdeal(Q); J
Ideal (x1^16 + x1, x2^16 + x2, x3^16 + x3, x4^16 + x4) of
Multivariate Polynomial Ring in x1, x2, x3, x4 over Finite
Field in alpha of size 2^4
```

sage.rings.ideal.**Ideal**(**args*, ***kwds*)

Create the ideal in ring with given generators.

There are some shorthand notations for creating an ideal, in addition to using the `Ideal()` function:

- `R.ideal(gens, coerce=True)`
- `gens*R`
- `R*gens`

INPUT:

- `R` - A ring (optional; if not given, will try to infer it from `gens`)
- `gens` - list of elements generating the ideal
- `coerce` - bool (optional, default: `True`); whether `gens` need to be coerced into the ring.

OUTPUT: The ideal of ring generated by `gens`.

EXAMPLES:

```
sage: R.<x> = ZZ[]
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: I
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring
sage: Ideal(R, [4 + 3*x + x^2, 1 + x^2])
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring
sage: Ideal((4 + 3*x + x^2, 1 + x^2))
Ideal (x^2 + 3*x + 4, x^2 + 1) of Univariate Polynomial Ring in x over Integer Ring

sage: ideal(x^2-2*x+1, x^2-1)
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: ideal([x^2-2*x+1, x^2-1])
Ideal (x^2 - 2*x + 1, x^2 - 1) of Univariate Polynomial Ring in x over Integer Ring
sage: l = [x^2-2*x+1, x^2-1]
sage: ideal(f^2 for f in l)
Ideal (x^4 - 4*x^3 + 6*x^2 - 4*x + 1, x^4 - 2*x^2 + 1) of
Univariate Polynomial Ring in x over Integer Ring
```

This example illustrates how Sage finds a common ambient ring for the ideal, even though 1 is in the integers (in this case).

```
sage: R.<t> = ZZ['t']
sage: i = ideal(1,t,t^2)
sage: i
Ideal (1, t, t^2) of Univariate Polynomial Ring in t over Integer Ring
sage: ideal(1/2,t,t^2)
Principal ideal (1) of Univariate Polynomial Ring in t over Rational Field
```

This shows that the issues at trac ticket #1104 are resolved:

```
sage: Ideal(3, 5)
Principal ideal (1) of Integer Ring
sage: Ideal(ZZ, 3, 5)
Principal ideal (1) of Integer Ring
sage: Ideal(2, 4, 6)
Principal ideal (2) of Integer Ring
```

You have to provide enough information that Sage can figure out which ring to put the ideal in.

```
sage: I = Ideal([])
Traceback (most recent call last):
...
ValueError: unable to determine which ring to embed the ideal in

sage: I = Ideal()
Traceback (most recent call last):
...
ValueError: need at least one argument
```

Note that some rings use different ideal implementations than the standard, even if they are PIDs.:

```
sage: R.<x> = GF(5)[]
sage: I = R*(x^2+3)
sage: type(I)
<class 'sage.rings.polynomial.ideal.Ideal_1poly_field'>
```

You can also pass in a specific ideal type:

```
sage: from sage.rings.ideal import Ideal_pid
sage: I = Ideal(x^2+3,ideal_class=Ideal_pid)
sage: type(I)
<class 'sage.rings.ideal.Ideal_pid'>
```

TESTS:

```
sage: R.<x> = ZZ[]
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: I == loads(dumps(I))
True

sage: I = Ideal(R, [4 + 3*x + x^2, 1 + x^2])
sage: I == loads(dumps(I))
True

sage: I = Ideal((4 + 3*x + x^2, 1 + x^2))
sage: I == loads(dumps(I))
True
```

This shows that the issue at trac ticket #5477 is fixed:

```
sage: R.<x> = QQ[]
sage: I = R.ideal([x + x^2])
sage: J = R.ideal([2*x + 2*x^2])
sage: J
Principal ideal (x^2 + x) of Univariate Polynomial Ring in x over Rational Field
sage: S = R.quotient_ring(I)
sage: U = R.quotient_ring(J)
sage: I == J
True
sage: S == U
True
```

class sage.rings.ideal.**Ideal_fractional**(*ring*, *gens*, *coerce=True*)
Bases: sage.rings.ideal.Ideal_generic

Fractional ideal of a ring.

See Ideal().

class sage.rings.ideal.**Ideal_generic**(*ring*, *gens*, *coerce=True*)
Bases: sage.structure.element.MonoidElement

An ideal.

See Ideal().

absolute_norm()
Returns the absolute norm of this ideal.

In the general case, this is just the ideal itself, since the ring it lies in can't be implicitly assumed to be an extension of anything.

We include this function for compatibility with cases such as ideals in number fields.

Todo

Implement this method.

EXAMPLES:

```
sage: R.<t> = GF(9, names='a')[]
sage: I = R.ideal(t^4 + t + 1)
sage: I.absolute_norm()
Traceback (most recent call last):
...
NotImplementedError
```

apply_morphism(*phi*)
Apply the morphism phi to every element of this ideal. Returns an ideal in the domain of phi.

EXAMPLES:

```
sage: psi = CC['x'].hom([-CC['x'].0])
sage: J = ideal([CC['x'].0 + 1]); J
Principal ideal (x + 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field
sage: psi(J)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field
sage: J.apply_morphism(psi)
Principal ideal (x - 1.00000000000000) of Univariate Polynomial Ring in x over Complex Field
```

```
sage: psi = ZZ['x'].hom([-ZZ['x'].0])
sage: J = ideal([ZZ['x'].0, 2]); J
Ideal (x, 2) of Univariate Polynomial Ring in x over Integer Ring
sage: psi(J)
Ideal (-x, 2) of Univariate Polynomial Ring in x over Integer Ring
sage: J.apply_morphism(psi)
Ideal (-x, 2) of Univariate Polynomial Ring in x over Integer Ring
```

TESTS:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: A = K.ideal(a)
sage: taus = K.embeddings(K)
sage: A.apply_morphism(taus[0]) # identity
Fractional ideal (a)
sage: A.apply_morphism(taus[1]) # complex conjugation
Fractional ideal (-a)
sage: A.apply_morphism(taus[0]) == A.apply_morphism(taus[1])
True

sage: K.<a> = NumberField(x^2 + 5)
sage: B = K.ideal([2, a + 1]); B
Fractional ideal (2, a + 1)
sage: taus = K.embeddings(K)
sage: B.apply_morphism(taus[0]) # identity
Fractional ideal (2, a + 1)
```

Since 2 is totally ramified, complex conjugation fixes it:

```
sage: B.apply_morphism(taus[1]) # complex conjugation
Fractional ideal (2, a + 1)
sage: taus[1](B)
Fractional ideal (2, a + 1)
```

associated_primes()

Return the list of associated prime ideals of this ideal.

EXAMPLES:

```
sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.associated_primes()
Traceback (most recent call last):
...
NotImplementedError
```

base_ring()

Returns the base ring of this ideal.

EXAMPLES:

```
sage: R = ZZ
sage: I = 3*R; I
Principal ideal (3) of Integer Ring
sage: J = 2*I; J
Principal ideal (6) of Integer Ring
sage: I.base_ring(); J.base_ring()
Integer Ring
Integer Ring
```

We construct an example of an ideal of a quotient ring:

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 2)
sage: I.base_ring()
Rational Field
```

And p-adic numbers:

```
sage: R = Zp(7, prec=10); R
7-adic Ring with capped relative precision 10
sage: I = 7*R; I
Principal ideal (7 + O(7^11)) of 7-adic Ring with capped relative precision 10
sage: I.base_ring()
7-adic Ring with capped relative precision 10
```

category()

Return the category of this ideal.

Note: category is dependent on the ring of the ideal.

EXAMPLES:

```
sage: I = ZZ.ideal(7)
sage: J = ZZ[x].ideal(7,x)
sage: K = ZZ[x].ideal(7)
sage: I.category()
Category of ring ideals in Integer Ring
sage: J.category()
Category of ring ideals in Univariate Polynomial Ring in x
over Integer Ring
sage: K.category()
Category of ring ideals in Univariate Polynomial Ring in x
over Integer Ring
```

embedded_primes()

Return the list of embedded primes of this ideal.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = R.ideal(x^2, x*y)
sage: I.embedded_primes()
[Ideal (y, x) of Multivariate Polynomial Ring in x, y over Rational Field]
```

gen(i)

Return the `i`-th generator in the current basis of this ideal.

EXAMPLE:

```
sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.gen(1)
y + 1

sage: ZZ.ideal(5,10).gen()
5
```

gens()

Return a set of generators / a basis of `self`.

This is the set of generators provided during creation of this ideal.

EXAMPLE:

```
sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.gens()
[x, y + 1]

sage: ZZ.ideal(5,10).gens()
(5,)
```

gens_reduced()

Same as `gens()` for this ideal, since there is currently no special `gens_reduced` algorithm implemented for this ring.

This method is provided so that ideals in $\mathbf{Z}$ have the method `gens_reduced()`, just like ideals of number fields.

EXAMPLES:

```
sage: ZZ.ideal(5).gens_reduced()
(5,)
```

is_maximal()

Return `True` if the ideal is maximal in the ring containing the ideal.

Todo

This is not implemented for many rings. Implement it!

EXAMPLES:

```
sage: R = ZZ
sage: I = R.ideal(7)
sage: I.is_maximal()
True
sage: R.ideal(16).is_maximal()
False
sage: S = Integers(8)
sage: S.ideal(0).is_maximal()
False
sage: S.ideal(2).is_maximal()
True
sage: S.ideal(4).is_maximal()
False
```

is_primary(*P=None*)

Returns `True` if this ideal is primary (or P-primary, if a prime ideal P is specified).

Recall that an ideal I is primary if and only if I has a unique associated prime (see page 52 in [AtiMac]). If this prime is P, then I is said to be P-primary.

INPUT:

- `P` - (default: `None`) a prime ideal in the same ring

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = R.ideal([x^2, x*y])
```

```
sage: I.is_primary()
False
sage: J = I.primary_decomposition()[1]; J
Ideal (y, x^2) of Multivariate Polynomial Ring in x, y over Rational Field
sage: J.is_primary()
True
sage: J.is_prime()
False
```

Some examples from the Macaulay2 documentation:

```
sage: R.<x, y, z> = GF(101)[]
sage: I = R.ideal([y^6])
sage: I.is_primary()
True
sage: I.is_primary(R.ideal([y]))
True
sage: I = R.ideal([x^4, y^7])
sage: I.is_primary()
True
sage: I = R.ideal([x*y, y^2])
sage: I.is_primary()
False
```

REFERENCES:

is_prime()

Return `True` if this ideal is prime.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = R.ideal([x, y])
sage: I.is_prime()        # a maximal ideal
True
sage: I = R.ideal([x^2-y])
sage: I.is_prime()        # a non-maximal prime ideal
True
sage: I = R.ideal([x^2, y])
sage: I.is_prime()        # a non-prime primary ideal
False
sage: I = R.ideal([x^2, x*y])
sage: I.is_prime()        # a non-prime non-primary ideal
False

sage: S = Integers(8)
sage: S.ideal(0).is_prime()
False
sage: S.ideal(2).is_prime()
True
sage: S.ideal(4).is_prime()
False
```

Note that this method is not implemented for all rings where it could be:

```
sage: R.<x> = ZZ[]
sage: I = R.ideal(7)
sage: I.is_prime()        # when implemented, should be True
Traceback (most recent call last):
...
```

```
NotImplementedError
```

Note: For general rings, uses the list of associated primes.

is_principal()
Returns `True` if the ideal is principal in the ring containing the ideal.

Todo

Code is naive. Only keeps track of ideal generators as set during initialization of the ideal. (Can the base ring change? See example below.)

EXAMPLES:

```
sage: R = ZZ['x']
sage: I = R.ideal(2,x)
sage: I.is_principal()
Traceback (most recent call last):
...
NotImplementedError
sage: J = R.base_extend(QQ).ideal(2,x)
sage: J.is_principal()
True
```

is_trivial()
Return `True` if this ideal is (0) or (1).

TESTS:

```
sage: I = ZZ.ideal(5)
sage: I.is_trivial()
False

sage: I = ZZ['x'].ideal(-1)
sage: I.is_trivial()
True

sage: I = ZZ['x'].ideal(ZZ['x'].gen()^2)
sage: I.is_trivial()
False

sage: I = QQ['x', 'y'].ideal(-5)
sage: I.is_trivial()
True

sage: I = CC['x'].ideal(0)
sage: I.is_trivial()
True
```

minimal_associated_primes()
Return the list of minimal associated prime ideals of this ideal.

EXAMPLES:

```
sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.minimal_associated_primes()
Traceback (most recent call last):
...
```

```
NotImplementedError
```

ngens()
Return the number of generators in the basis.

EXAMPLE:

```
sage: P.<x,y> = PolynomialRing(QQ,2)
sage: I = Ideal([x,y+1]); I
Ideal (x, y + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.ngens()
2

sage: ZZ.ideal(5,10).ngens()
1
```

norm()
Returns the norm of this ideal.

In the general case, this is just the ideal itself, since the ring it lies in can't be implicitly assumed to be an extension of anything.

We include this function for compatibility with cases such as ideals in number fields.

EXAMPLES:

```
sage: R.<t> = GF(8, names='a')[]
sage: I = R.ideal(t^4 + t + 1)
sage: I.norm()
Principal ideal (t^4 + t + 1) of Univariate Polynomial Ring in t over Finite Field in a of s
```

primary_decomposition()
Return a decomposition of this ideal into primary ideals.

EXAMPLES:

```
sage: R = ZZ['x']
sage: I = R.ideal(7)
sage: I.primary_decomposition()
Traceback (most recent call last):
...
NotImplementedError
```

reduce(f)
Return the reduction of the element of f modulo `self`.

This is an element of R that is equivalent modulo I to f where I is `self`.

EXAMPLES:

```
sage: ZZ.ideal(5).reduce(17)
2
sage: parent(ZZ.ideal(5).reduce(17))
Integer Ring
```

ring()
Returns the ring containing this ideal.

EXAMPLES:

```
sage: R = ZZ
sage: I = 3*R; I
```

```
Principal ideal (3) of Integer Ring
sage: J = 2*I; J
Principal ideal (6) of Integer Ring
sage: I.ring(); J.ring()
Integer Ring
Integer Ring
```

Note that `self.ring()` is different from `self.base_ring()`

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 2)
sage: I.base_ring()
Rational Field
sage: I.ring()
Univariate Polynomial Ring in x over Rational Field
```

Another example using polynomial rings:

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 - 3)
sage: I.ring()
Univariate Polynomial Ring in x over Rational Field
sage: Rbar = R.quotient(I, names='a')
sage: S = PolynomialRing(Rbar, 'y'); y = Rbar.gen(); S
Univariate Polynomial Ring in y over Univariate Quotient Polynomial Ring in a over Rational
sage: J = S.ideal(y^2 + 1)
sage: J.ring()
Univariate Polynomial Ring in y over Univariate Quotient Polynomial Ring in a over Rational
```

class sage.rings.ideal.**Ideal_pid**(*ring*, *gen*)

Bases: sage.rings.ideal.Ideal_principal

An ideal of a principal ideal domain.

See `Ideal()`.

gcd(*other*)

Returns the greatest common divisor of the principal ideal with the ideal `other`; that is, the largest principal ideal contained in both the ideal and `other`

EXAMPLES:

An example in the principal ideal domain $\mathbf{Z}$:

```
sage: R = ZZ
sage: I = R.ideal(42)
sage: J = R.ideal(70)
sage: I.gcd(J)
Principal ideal (14) of Integer Ring
sage: J.gcd(I)
Principal ideal (14) of Integer Ring
```

TESTS:

We cannot take the gcd of a principal ideal with a non-principal ideal as well: (`gcd(I,J)` should be (7))

```
sage: I = ZZ.ideal(7)
sage: J = ZZ[x].ideal(7,x)
sage: I.gcd(J)
Traceback (most recent call last):
```

```
...
NotImplementedError
sage: J.gcd(I)
Traceback (most recent call last):
...
AttributeError: 'Ideal_generic' object has no attribute 'gcd'
```

Note:

```
sage: type(I)
<class 'sage.rings.ideal.Ideal_pid'>
sage: type(J)
<class 'sage.rings.ideal.Ideal_generic'>
```

is_maximal()

Returns whether this ideal is maximal.

Principal ideal domains have Krull dimension 1 (or 0), so an ideal is maximal if and only if it's prime (and nonzero if the ring is not a field).

EXAMPLES:

```
sage: R.<t> = GF(5)[]
sage: p = R.ideal(t^2 + 2)
sage: p.is_maximal()
True
sage: p = R.ideal(t^2 + 1)
sage: p.is_maximal()
False
sage: p = R.ideal(0)
sage: p.is_maximal()
False
sage: p = R.ideal(1)
sage: p.is_maximal()
False
```

is_prime()

Return `True` if the ideal is prime.

This relies on the ring elements having a method `is_irreducible()` implemented, since an ideal (a) is prime iff a is irreducible (or 0).

EXAMPLES:

```
sage: ZZ.ideal(2).is_prime()
True
sage: ZZ.ideal(-2).is_prime()
True
sage: ZZ.ideal(4).is_prime()
False
sage: ZZ.ideal(0).is_prime()
True
sage: R.<x>=QQ[]
sage: P=R.ideal(x^2+1); P
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: P.is_prime()
True
```

In fields, only the zero ideal is prime:

```
sage: RR.ideal(0).is_prime()
True
sage: RR.ideal(7).is_prime()
False
```

reduce(*f*)

Return the reduction of f modulo `self`.

EXAMPLES:

```
sage: I = 8*ZZ
sage: I.reduce(10)
2
sage: n = 10; n.mod(I)
2
```

residue_field()

Return the residue class field of this ideal, which must be prime.

EXAMPLES:

```
sage: P = ZZ.ideal(61); P
Principal ideal (61) of Integer Ring
sage: F = P.residue_field(); F
Residue field of Integers modulo 61
sage: pi = F.reduction_map(); pi
Partially defined reduction map:
  From: Rational Field
  To:   Residue field of Integers modulo 61
sage: pi(123/234)
6
sage: pi(1/61)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot reduce rational 1/61 modulo 61: it has negative valuation
sage: lift = F.lift_map(); lift
Lifting map:
  From: Residue field of Integers modulo 61
  To:   Integer Ring
sage: lift(F(12345/67890))
33
sage: (12345/67890) % 61
33
```

TESTS:

```
sage: ZZ.ideal(96).residue_field()
Traceback (most recent call last):
...
ValueError: The ideal (Principal ideal (96) of Integer Ring) is not prime

sage: R.<x>=QQ[]
sage: I=R.ideal(x^2+1)
sage: I.is_prime()
True
sage: I.residue_field()
Traceback (most recent call last):
...
TypeError: residue fields only supported for polynomial rings over finite fields.
```

class sage.rings.ideal.**Ideal_principal**(*ring*, *gens*, *coerce=True*)

Bases: sage.rings.ideal.Ideal_generic

A principal ideal.

See Ideal().

divides(*other*)

Return True if self divides other.

EXAMPLES:

```
sage: P.<x> = PolynomialRing(QQ)
sage: I = P.ideal(x)
sage: J = P.ideal(x^2)
sage: I.divides(J)
True
sage: J.divides(I)
False
```

gen()

Returns the generator of the principal ideal. The generators are elements of the ring containing the ideal.

EXAMPLES:

A simple example in the integers:

```
sage: R = ZZ
sage: I = R.ideal(7)
sage: J = R.ideal(7, 14)
sage: I.gen(); J.gen()
7
7
```

Note that the generator belongs to the ring from which the ideal was initialized:

```
sage: R = ZZ[x]
sage: I = R.ideal(x)
sage: J = R.base_extend(QQ).ideal(2,x)
sage: a = I.gen(); a
x
sage: b = J.gen(); b
1
sage: a.base_ring()
Integer Ring
sage: b.base_ring()
Rational Field
```

is_principal()

Returns True if the ideal is principal in the ring containing the ideal. When the ideal construction is explicitly principal (i.e. when we define an ideal with one element) this is always the case.

EXAMPLES:

Note that Sage automatically coerces ideals into principal ideals during initialization:

```
sage: R = ZZ[x]
sage: I = R.ideal(x)
sage: J = R.ideal(2,x)
sage: K = R.base_extend(QQ).ideal(2,x)
sage: I
Principal ideal (x) of Univariate Polynomial Ring in x
over Integer Ring
```

```
sage: J
Ideal (2, x) of Univariate Polynomial Ring in x over Integer Ring
sage: K
Principal ideal (1) of Univariate Polynomial Ring in x
over Rational Field
sage: I.is_principal()
True
sage: K.is_principal()
True
```

sage.rings.ideal.**Katsura**(*R*, *n=None*, *homog=False*, *singular=Singular*)

n-th katsura ideal of R if R is coercible to Singular.

INPUT:

- R – base ring to construct ideal for
- n – (default: None) which katsura ideal of R. If None, then n is set to R.ngens().
- homog – if True a homogeneous ideal is returned using the last variable in the ideal (default: False)
- singular – singular instance to use

EXAMPLES:

```
sage: P.<x,y,z> = PolynomialRing(QQ,3)
sage: I = sage.rings.ideal.Katsura(P,3); I
Ideal (x + 2*y + 2*z - 1, x^2 + 2*y^2 + 2*z^2 - x, 2*x*y + 2*y*z - y)
of Multivariate Polynomial Ring in x, y, z over Rational Field

sage: Q.<x> = PolynomialRing(QQ,1)
sage: J = sage.rings.ideal.Katsura(Q,1); J
Ideal (x - 1) of Multivariate Polynomial Ring in x over Rational Field
```

sage.rings.ideal.**is_Ideal**(*x*)

Return True if object is an ideal of a ring.

EXAMPLES:

A simple example involving the ring of integers. Note that Sage does not interpret rings objects themselves as ideals. However, one can still explicitly construct these ideals:

```
sage: from sage.rings.ideal import is_Ideal
sage: R = ZZ
sage: is_Ideal(R)
False
sage: 1*R; is_Ideal(1*R)
Principal ideal (1) of Integer Ring
True
sage: 0*R; is_Ideal(0*R)
Principal ideal (0) of Integer Ring
True
```

Sage recognizes ideals of polynomial rings as well:

```
sage: R = PolynomialRing(QQ, 'x'); x = R.gen()
sage: I = R.ideal(x^2 + 1); I
Principal ideal (x^2 + 1) of Univariate Polynomial Ring in x over Rational Field
sage: is_Ideal(I)
True
sage: is_Ideal((x^2 + 1)*R)
True
```

2.2 Monoid of ideals in a commutative ring

sage.rings.ideal_monoid.**IdealMonoid**(*R*)
Return the monoid of ideals in the ring R.

EXAMPLE:

```
sage: R = QQ['x']
sage: sage.rings.ideal_monoid.IdealMonoid(R)
Monoid of ideals of Univariate Polynomial Ring in x over Rational Field
```

class sage.rings.ideal_monoid.**IdealMonoid_c**(*R*)
Bases: sage.structure.parent.Parent

The monoid of ideals in a commutative ring.

TESTS:

```
sage: R = QQ['x']
sage: M = sage.rings.ideal_monoid.IdealMonoid(R)
sage: TestSuite(M).run()
  Failure in _test_category:
...
The following tests failed: _test_elements
```

(The "_test_category" test fails but I haven't the foggiest idea why.)

Element
alias of Ideal_generic

ring()
Return the ring of which this is the ideal monoid.

EXAMPLE:

```
sage: R = QuadraticField(-23, 'a')
sage: M = sage.rings.ideal_monoid.IdealMonoid(R); M.ring() is R
True
```

2.3 Ideals of non-commutative rings

Generic implementation of one- and two-sided ideals of non-commutative rings.

AUTHOR:

- Simon King (2011-03-21), <simon.king@uni-jena.de>, trac ticket #7797.

EXAMPLES:

```
sage: MS = MatrixSpace(ZZ,2,2)
sage: MS*MS([0,1,-2,3])
Left Ideal
(
  [ 0  1]
  [-2  3]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
sage: MS([0,1,-2,3])*MS
Right Ideal
```

```
(
  [ 0  1]
  [-2  3]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
sage: MS*MS([0,1,-2,3])*MS
Twosided Ideal
(
  [ 0  1]
  [-2  3]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

See `letterplace_ideal` for a more elaborate implementation in the special case of ideals in free algebras.

TESTS:

```
sage: A = SteenrodAlgebra(2)
sage: IL = A*[A.1+A.2,A.1^2]; IL
Left Ideal (Sq(2) + Sq(4), Sq(1,1)) of mod 2 Steenrod algebra, milnor basis
sage: TestSuite(IL).run(skip=['_test_category'],verbose=True)
running ._test_eq() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_pickling() . . . pass
```

class sage.rings.noncommutative_ideals.**IdealMonoid_nc**(*R*)

Bases: sage.rings.ideal_monoid.IdealMonoid_c

Base class for the monoid of ideals over a non-commutative ring.

Note: This class is essentially the same as `IdealMonoid_c`, but does not complain about non-commutative rings.

EXAMPLES:

```
sage: MS = MatrixSpace(ZZ,2,2)
sage: MS.ideal_monoid()
Monoid of ideals of Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

class sage.rings.noncommutative_ideals.**Ideal_nc**(*ring*, *gens*, *coerce=True*, *side='twosided'*)

Bases: sage.rings.ideal.Ideal_generic

Generic non-commutative ideal.

All fancy stuff such as the computation of Groebner bases must be implemented in sub-classes. See `LetterplaceIdeal` for an example.

EXAMPLES:

```
sage: MS = MatrixSpace(QQ,2,2)
sage: I = MS*[MS.1,MS.2]; I
Left Ideal
(
  [0 1]
  [0 0],

  [0 0]
  [1 0]
)
```

```
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: [MS.1,MS.2]*MS
Right Ideal
(
  [0 1]
  [0 0],

  [0 0]
  [1 0]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
sage: MS*[MS.1,MS.2]*MS
Twosided Ideal
(
  [0 1]
  [0 0],

  [0 0]
  [1 0]
)
 of Full MatrixSpace of 2 by 2 dense matrices over Rational Field
```

side()

Return a string that describes the sidedness of this ideal.

EXAMPLES:

```
sage: A = SteenrodAlgebra(2)
sage: IL = A*[A.1+A.2,A.1^2]
sage: IR = [A.1+A.2,A.1^2]*A
sage: IT = A*[A.1+A.2,A.1^2]*A
sage: IL.side()
'left'
sage: IR.side()
'right'
sage: IT.side()
'twosided'
```

CHAPTER

THREE

RING MORPHISMS

3.1 Homomorphisms of rings

We give a large number of examples of ring homomorphisms.

EXAMPLE:

Natural inclusion $\mathbf{Z} \hookrightarrow \mathbf{Q}$:

```
sage: H = Hom(ZZ, QQ)
sage: phi = H([1])
sage: phi(10)
10
sage: phi(3/1)
3
sage: phi(2/3)
Traceback (most recent call last):
...
TypeError: 2/3 fails to convert into the map's domain Integer Ring, but a `pushforward` method is not
```

There is no homomorphism in the other direction:

```
sage: H = Hom(QQ, ZZ)
sage: H([1])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

EXAMPLES:

Reduction to finite field:

```
sage: H = Hom(ZZ, GF(9, 'a'))
sage: phi = H([1])
sage: phi(5)
2
sage: psi = H([4])
sage: psi(5)
2
```

Map from single variable polynomial ring:

```
sage: R.<x> = ZZ[]
sage: phi = R.hom([2], GF(5))
sage: phi
Ring morphism:
```

```
  From: Univariate Polynomial Ring in x over Integer Ring
  To:   Finite Field of size 5
  Defn: x |--> 2
sage: phi(x + 12)
4
```

Identity map on the real numbers:

```
sage: f = RR.hom([RR(1)]); f
Ring endomorphism of Real Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: f(2.5)
2.50000000000000
sage: f = RR.hom( [2.0] )
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

Homomorphism from one precision of field to another.

From smaller to bigger doesn't make sense:

```
sage: R200 = RealField(200)
sage: f = RR.hom( R200 )
Traceback (most recent call last):
...
TypeError: Natural coercion morphism from Real Field with 53 bits of precision to Real Field with 20
```

From bigger to small does:

```
sage: f = RR.hom( RealField(15) )
sage: f(2.5)
2.500
sage: f(RR.pi())
3.142
```

Inclusion map from the reals to the complexes:

```
sage: i = RR.hom([CC(1)]); i
Ring morphism:
  From: Real Field with 53 bits of precision
  To:   Complex Field with 53 bits of precision
  Defn: 1.00000000000000 |--> 1.00000000000000
sage: i(RR('3.1'))
3.10000000000000
```

A map from a multivariate polynomial ring to itself:

```
sage: R.<x,y,z> = PolynomialRing(QQ,3)
sage: phi = R.hom([y,z,x^2]); phi
Ring endomorphism of Multivariate Polynomial Ring in x, y, z over Rational Field
  Defn: x |--> y
        y |--> z
        z |--> x^2
sage: phi(x+y+z)
x^2 + y + z
```

An endomorphism of a quotient of a multi-variate polynomial ring:

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = quo(R, ideal(1 + y^2))
sage: phi = S.hom([a^2, -b])
sage: phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the idea
  Defn: a |--> a^2
        b |--> -b
sage: phi(b)
-b
sage: phi(a^2 + b^2)
a^4 - 1
```

The reduction map from the integers to the integers modulo 8, viewed as a quotient ring:

```
sage: R = ZZ.quo(8*ZZ)
sage: pi = R.cover()
sage: pi
Ring morphism:
  From: Integer Ring
  To:   Ring of integers modulo 8
  Defn: Natural quotient map
sage: pi.domain()
Integer Ring
sage: pi.codomain()
Ring of integers modulo 8
sage: pi(10)
2
sage: pi.lift()
Set-theoretic ring morphism:
  From: Ring of integers modulo 8
  To:   Integer Ring
  Defn: Choice of lifting map
sage: pi.lift(13)
5
```

Inclusion of `GF(2)` into `GF(4,'a')`:

```
sage: k = GF(2)
sage: i = k.hom(GF(4, 'a'))
sage: i
Ring Coercion morphism:
  From: Finite Field of size 2
  To:   Finite Field in a of size 2^2
sage: i(0)
0
sage: a = i(1); a.parent()
Finite Field in a of size 2^2
```

We next compose the inclusion with reduction from the integers to `GF(2)`:

```
sage: pi = ZZ.hom(k)
sage: pi
Ring Coercion morphism:
  From: Integer Ring
  To:   Finite Field of size 2
sage: f = i * pi
sage: f
Composite map:
```

```
  From: Integer Ring
  To:   Finite Field in a of size 2^2
  Defn:   Ring Coercion morphism:
          From: Integer Ring
          To:   Finite Field of size 2
        then
          Ring Coercion morphism:
          From: Finite Field of size 2
          To:   Finite Field in a of size 2^2
sage: a = f(5); a
1
sage: a.parent()
Finite Field in a of size 2^2
```

Inclusion from $\mathbf{Q}$ to the 3-adic field:

```
sage: phi = QQ.hom(Qp(3, print_mode = 'series'))
sage: phi
Ring Coercion morphism:
  From: Rational Field
  To:   3-adic Field with capped relative precision 20
sage: phi.codomain()
3-adic Field with capped relative precision 20
sage: phi(394)
1 + 2*3 + 3^2 + 2*3^3 + 3^4 + 3^5 + O(3^20)
```

An automorphism of a quotient of a univariate polynomial ring:

```
sage: R.<x> = PolynomialRing(QQ)
sage: S.<sqrt2> = R.quo(x^2-2)
sage: sqrt2^2
2
sage: (3+sqrt2)^10
993054*sqrt2 + 1404491
sage: c = S.hom([-sqrt2])
sage: c(1+sqrt2)
-sqrt2 + 1
```

Note that Sage verifies that the morphism is valid:

```
sage: (1 - sqrt2)^2
-2*sqrt2 + 3
sage: c = S.hom([1-sqrt2])    # this is not valid
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

Endomorphism of power series ring:

```
sage: R.<t> = PowerSeriesRing(QQ); R
Power Series Ring in t over Rational Field
sage: f = R.hom([t^2]); f
Ring endomorphism of Power Series Ring in t over Rational Field
  Defn: t |--> t^2
sage: R.set_default_prec(10)
sage: s = 1/(1 + t); s
1 - t + t^2 - t^3 + t^4 - t^5 + t^6 - t^7 + t^8 - t^9 + O(t^10)
sage: f(s)
```

```
1 - t^2 + t^4 - t^6 + t^8 - t^10 + t^12 - t^14 + t^16 - t^18 + O(t^20)
```

Frobenius on a power series ring over a finite field:

```
sage: R.<t> = PowerSeriesRing(GF(5))
sage: f = R.hom([t^5]); f
Ring endomorphism of Power Series Ring in t over Finite Field of size 5
  Defn: t |--> t^5
sage: a = 2 + t + 3*t^2 + 4*t^3 + O(t^4)
sage: b = 1 + t + 2*t^2 + t^3 + O(t^5)
sage: f(a)
2 + t^5 + 3*t^10 + 4*t^15 + O(t^20)
sage: f(b)
1 + t^5 + 2*t^10 + t^15 + O(t^25)
sage: f(a*b)
2 + 3*t^5 + 3*t^10 + t^15 + O(t^20)
sage: f(a)*f(b)
2 + 3*t^5 + 3*t^10 + t^15 + O(t^20)
```

Homomorphism of Laurent series ring:

```
sage: R.<t> = LaurentSeriesRing(QQ, 10)
sage: f = R.hom([t^3 + t]); f
Ring endomorphism of Laurent Series Ring in t over Rational Field
  Defn: t |--> t + t^3
sage: s = 2/t^2 + 1/(1 + t); s
2*t^-2 + 1 - t + t^2 - t^3 + t^4 - t^5 + t^6 - t^7 + t^8 - t^9 + O(t^10)
sage: f(s)
2*t^-2 - 3 - t + 7*t^2 - 2*t^3 - 5*t^4 - 4*t^5 + 16*t^6 - 9*t^7 + O(t^8)
sage: f = R.hom([t^3]); f
Ring endomorphism of Laurent Series Ring in t over Rational Field
  Defn: t |--> t^3
sage: f(s)
2*t^-6 + 1 - t^3 + t^6 - t^9 + t^12 - t^15 + t^18 - t^21 + t^24 - t^27 + O(t^30)
```

Note that the homomorphism must result in a converging Laurent series, so the valuation of the image of the generator must be positive:

```
sage: R.hom([1/t])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
sage: R.hom([1])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

Complex conjugation on cyclotomic fields:

```
sage: K.<zeta7> = CyclotomicField(7)
sage: c = K.hom([1/zeta7]); c
Ring endomorphism of Cyclotomic Field of order 7 and degree 6
  Defn: zeta7 |--> -zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - zeta7 - 1
sage: a = (1+zeta7)^5; a
zeta7^5 + 5*zeta7^4 + 10*zeta7^3 + 10*zeta7^2 + 5*zeta7 + 1
sage: c(a)
5*zeta7^5 + 5*zeta7^4 - 4*zeta7^2 - 5*zeta7 - 4
sage: c(zeta7 + 1/zeta7)       # this element is obviously fixed by inversion
```

```
-zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - 1
sage: zeta7 + 1/zeta7
-zeta7^5 - zeta7^4 - zeta7^3 - zeta7^2 - 1
```

Embedding a number field into the reals:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<beta> = NumberField(x^3 - 2)
sage: alpha = RR(2)^(1/3); alpha
1.25992104989487
sage: i = K.hom([alpha],check=False); i
Ring morphism:
  From: Number Field in beta with defining polynomial x^3 - 2
  To:   Real Field with 53 bits of precision
  Defn: beta |--> 1.25992104989487
sage: i(beta)
1.25992104989487
sage: i(beta^3)
2.00000000000000
sage: i(beta^2 + 1)
2.58740105196820
```

An example from Jim Carlson:

```
sage: K = QQ # by the way :-)
sage: R.<a,b,c,d> = K[]; R
Multivariate Polynomial Ring in a, b, c, d over Rational Field
sage: S.<u> = K[]; S
Univariate Polynomial Ring in u over Rational Field
sage: f = R.hom([0,0,0,u], S); f
Ring morphism:
  From: Multivariate Polynomial Ring in a, b, c, d over Rational Field
  To:   Univariate Polynomial Ring in u over Rational Field
  Defn: a |--> 0
        b |--> 0
        c |--> 0
        d |--> u
sage: f(a+b+c+d)
u
sage: f( (a+b+c+d)^2 )
u^2
```

TESTS:

```
sage: H = Hom(ZZ, QQ)
sage: H == loads(dumps(H))
True

sage: K.<zeta7> = CyclotomicField(7)
sage: c = K.hom([1/zeta7])
sage: c == loads(dumps(c))
True

sage: R.<t> = PowerSeriesRing(GF(5))
sage: f = R.hom([t^5])
sage: f == loads(dumps(f))
True
```

class sage.rings.morphism.**FrobeniusEndomorphism_generic**
Bases: sage.rings.morphism.RingHomomorphism

A class implementing Frobenius endomorphisms on rings of prime characteristic.

power()
Return an integer n such that this endormorphism is the n-th power of the absolute (arithmetic) Frobenius.

EXAMPLES:

```
sage: K.<u> = PowerSeriesRing(GF(5))
sage: Frob = K.frobenius_endomorphism()
sage: Frob.power()
1
sage: (Frob^9).power()
9
```

class sage.rings.morphism.**RingHomomorphism**
Bases: sage.rings.morphism.RingMap

Homomorphism of rings.

inverse_image(I)
Return the inverse image of the ideal I under this ring homomorphism.

EXAMPLES:

This is not implemented in any generality yet:

```
sage: f = ZZ.hom(ZZ)
sage: f.inverse_image(ZZ.ideal(2))
Traceback (most recent call last):
...
NotImplementedError
```

is_injective()
Return whether or not this morphism is injective, or raise a NotImplementedError.

EXAMPLES:

Note that currently this is not implemented in most interesting cases:

```
sage: f = ZZ.hom(QQ)
sage: f.is_injective()
Traceback (most recent call last):
...
NotImplementedError
```

is_zero()
Return True if this is the zero map and False otherwise.

A *ring* homomorphism is considered to be 0 if and only if it sends the 1 element of the domain to the 0 element of the codomain. Since rings in Sage all have a 1 element, the zero homomorphism is only to a ring of order 1, where 1 == 0, e.g., the ring Integers(1).

EXAMPLES:

First an example of a map that is obviously nonzero:

```
sage: h = Hom(ZZ, QQ)
sage: f = h.natural_map()
sage: f.is_zero()
False
```

Next we make the zero ring as $\mathbf{Z}/1\mathbf{Z}$:

```
sage: R = Integers(1)
sage: R
Ring of integers modulo 1
sage: h = Hom(ZZ, R)
sage: f = h.natural_map()
sage: f.is_zero()
True
```

Finally we check an example in characteristic 2:

```
sage: h = Hom(ZZ, GF(2))
sage: f = h.natural_map()
sage: f.is_zero()
False
```

lift(*x=None*)

Return a lifting homomorphism associated to this homomorphism, if it has been defined.

If `x` is not `None`, return the value of the lift morphism on `x`.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: f = R.hom([x,x])
sage: f(x+y)
2*x
sage: f.lift()
Traceback (most recent call last):
...
ValueError: no lift map defined
sage: g = R.hom(R)
sage: f._set_lift(g)
sage: f.lift() == g
True
sage: f.lift(x)
x
```

pushforward(*I*)

Returns the pushforward of the ideal I under this ring homomorphism.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2,y^2]);  f = S.cover()
sage: f.pushforward(R.ideal([x,3*x+x*y+y^2]))
Ideal (xx, xx*yy + 3*xx) of Quotient of Multivariate Polynomial Ring in x, y over Rational F
```

class sage.rings.morphism.**RingHomomorphism_coercion**

Bases: sage.rings.morphism.RingHomomorphism

Initialize `self`.

INPUT:

- `parent` – ring homset
- `check` – bool (default: `True`)

EXAMPLES:

```
sage: f = ZZ.hom(QQ); f                    # indirect doctest
Ring Coercion morphism:
  From: Integer Ring
```

```
  To:   Rational Field

sage: f == loads(dumps(f))
True
```

class sage.rings.morphism.**RingHomomorphism_cover**
Bases: sage.rings.morphism.RingHomomorphism

A homomorphism induced by quotienting a ring out by an ideal.

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quo(x^2 + y^2)
sage: phi = S.cover(); phi
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:   Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2 +
  Defn: Natural quotient map
sage: phi(x+y)
a + b
```

kernel()
Return the kernel of this covering morphism, which is the ideal that was quotiented out by.

EXAMPLES:

```
sage: f = Zmod(6).cover()
sage: f.kernel()
Principal ideal (6) of Integer Ring
```

class sage.rings.morphism.**RingHomomorphism_from_base**
Bases: sage.rings.morphism.RingHomomorphism

A ring homomorphism determined by a ring homomorphism of the base ring.

AUTHOR:

- Simon King (initial version, 2010-04-30)

EXAMPLES:

We define two polynomial rings and a ring homomorphism:

```
sage: R.<x,y> = QQ[]
sage: S.<z> = QQ[]
sage: f = R.hom([2*z,3*z],S)
```

Now we construct polynomial rings based on `R` and `S`, and let `f` act on the coefficients:

```
sage: PR.<t> = R[]
sage: PS = S['t']
sage: Pf = PR.hom(f,PS)
sage: Pf
Ring morphism:
  From: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in x, y over Rational
  To:   Univariate Polynomial Ring in t over Univariate Polynomial Ring in z over Rational Field
  Defn: Induced from base ring by
        Ring morphism:
          From: Multivariate Polynomial Ring in x, y over Rational Field
          To:   Univariate Polynomial Ring in z over Rational Field
          Defn: x |--> 2*z
```

```
                y |--> 3*z
sage: p = (x - 4*y + 1/13)*t^2 + (1/2*x^2 - 1/3*y^2)*t + 2*y^2 + x
sage: Pf(p)
(-10*z + 1/13)*t^2 - z^2*t + 18*z^2 + 2*z
```

Similarly, we can construct the induced homomorphism on a matrix ring over our polynomial rings:

```
sage: MR = MatrixSpace(R,2,2)
sage: MS = MatrixSpace(S,2,2)
sage: M = MR([x^2 + 1/7*x*y - y^2, - 1/2*y^2 + 2*y + 1/6, 4*x^2 - 14*x, 1/2*y^2 + 13/4*x - 2/11*
sage: Mf = MR.hom(f,MS)
sage: Mf
Ring morphism:
  From: Full MatrixSpace of 2 by 2 dense matrices over Multivariate Polynomial Ring in x, y over
  To:   Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in z over Rati
  Defn: Induced from base ring by
        Ring morphism:
          From: Multivariate Polynomial Ring in x, y over Rational Field
          To:   Univariate Polynomial Ring in z over Rational Field
          Defn: x |--> 2*z
                y |--> 3*z
sage: Mf(M)
[           -29/7*z^2 -9/2*z^2 + 6*z + 1/6]
[       16*z^2 - 28*z   9/2*z^2 + 131/22*z]
```

The construction of induced homomorphisms is recursive, and so we have:

```
sage: MPR = MatrixSpace(PR, 2)
sage: MPS = MatrixSpace(PS, 2)
sage: M = MPR([(- x + y)*t^2 + 58*t - 3*x^2 + x*y, (- 1/7*x*y - 1/40*x)*t^2 + (5*x^2 + y^2)*t +
sage: MPf = MPR.hom(f,MPS); MPf
Ring morphism:
  From: Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in t over Mult
  To:   Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial Ring in t over Univ
  Defn: Induced from base ring by
        Ring morphism:
          From: Univariate Polynomial Ring in t over Multivariate Polynomial Ring in x, y over R
          To:   Univariate Polynomial Ring in t over Univariate Polynomial Ring in z over Ration
          Defn: Induced from base ring by
                Ring morphism:
                  From: Multivariate Polynomial Ring in x, y over Rational Field
                  To:   Univariate Polynomial Ring in z over Rational Field
                  Defn: x |--> 2*z
                        y |--> 3*z
sage: MPf(M)
[                    z*t^2 + 58*t - 6*z^2 (-6/7*z^2 - 1/20*z)*t^2 + 29*z^2*t + 6*z]
[    (-z + 1)*t^2 + 11*z^2 + 15/2*z + 1/4                            (20*z + 1)*t^2]
```

is_identity()

Return `True` if this morphism is the identity morphism.

EXAMPLES:

```
sage: K.<z> = GF(4)
sage: phi = End(K)([z^2])
sage: R.<t> = K[]
sage: psi = End(R)(phi)
sage: psi.is_identity()
False
```

underlying_map()
Return the underlying homomorphism of the base ring.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: S.<z> = QQ[]
sage: f = R.hom([2*z,3*z],S)
sage: MR = MatrixSpace(R,2)
sage: MS = MatrixSpace(S,2)
sage: g = MR.hom(f,MS)
sage: g.underlying_map() == f
True
```

class sage.rings.morphism.**RingHomomorphism_from_quotient**
Bases: sage.rings.morphism.RingHomomorphism

A ring homomorphism with domain a generic quotient ring.

INPUT:

- parent – a ring homset Hom(R,S)
- phi – a ring homomorphism C --> S, where C is the domain of R.cover()

OUTPUT: a ring homomorphism

The domain R is a quotient object $C \to R$, and R.cover() is the ring homomorphism $\varphi : C \to R$. The condition on the elements im_gens of S is that they define a homomorphism $C \to S$ such that each generator of the kernel of φ maps to 0.

EXAMPLES:

```
sage: R.<x, y, z> = PolynomialRing(QQ, 3)
sage: S.<a, b, c> = R.quo(x^3 + y^3 + z^3)
sage: phi = S.hom([b, c, a]); phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y, z over Rational Field by
  Defn: a |--> b
        b |--> c
        c |--> a
sage: phi(a+b+c)
a + b + c
sage: loads(dumps(phi)) == phi
True
```

Validity of the homomorphism is determined, when possible, and a TypeError is raised if there is no homomorphism sending the generators to the given images:

```
sage: S.hom([b^2, c^2, a^2])
Traceback (most recent call last):
...
TypeError: images do not define a valid homomorphism
```

morphism_from_cover()
Underlying morphism used to define this quotient map, i.e., the morphism from the cover of the domain.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<xx,yy> = R.quo([x^2,y^2])
sage: S.hom([yy,xx]).morphism_from_cover()
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:   Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x
```

```
        Defn: x |--> yy
              y |--> xx
```

class sage.rings.morphism.**RingHomomorphism_im_gens**
Bases: sage.rings.morphism.RingHomomorphism

A ring homomorphism determined by the images of generators.

im_gens()
Return the images of the generators of the domain.

OUTPUT:

- list – a copy of the list of gens (it is safe to change this)

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: f = R.hom([x,x+y])
sage: f.im_gens()
[x, x + y]
```

We verify that the returned list of images of gens is a copy, so changing it doesn't change f:

```
sage: f.im_gens()[0] = 5
sage: f.im_gens()
[x, x + y]
```

class sage.rings.morphism.**RingMap**
Bases: sage.categories.morphism.Morphism

Set-theoretic map between rings.

class sage.rings.morphism.**RingMap_lift**
Bases: sage.rings.morphism.RingMap

Given rings R and S such that for any $x \in R$ the function x.lift() is an element that naturally coerces to S, this returns the set-theoretic ring map $R \to S$ sending x to x.lift().

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: S.<xbar,ybar> = R.quo( (x^2 + y^2, y) )
sage: S.lift()
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2 +
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
sage: S.lift() == 0
False
```

Since trac ticket #11068, it is possible to create quotient rings of non-commutative rings by two-sided ideals. It was needed to modify RingMap_lift so that rings can be accepted that are no instances of sage.rings.ring.Ring, as in the following example:

```
sage: MS = MatrixSpace(GF(5),2,2)
sage: I = MS*[MS.0*MS.1,MS.2+MS.3]*MS
sage: Q = MS.quo(I)
sage: Q.0*Q.1   # indirect doctest
[0 1]
[0 0]
```

sage.rings.morphism.**is_RingHomomorphism**(*phi*)
Return True if phi is of type RingHomomorphism.

EXAMPLES:

```
sage: f = Zmod(8).cover()
sage: sage.rings.morphism.is_RingHomomorphism(f)
True
sage: sage.rings.morphism.is_RingHomomorphism(2/3)
False
```

3.2 Space of homomorphisms between two rings

sage.rings.homset.**RingHomset**(*R*, *S*, *category=None*)
Construct a space of homomorphisms between the rings R and S.

For more on homsets, see Hom().

EXAMPLES:

```
sage: Hom(ZZ, QQ) # indirect doctest
Set of Homomorphisms from Integer Ring to Rational Field
```

class sage.rings.homset.**RingHomset_generic**(*R*, *S*, *category=None*)
Bases: sage.categories.homset.HomsetWithBase

A generic space of homomorphisms between two rings.

EXAMPLES:

```
sage: Hom(ZZ, QQ)
Set of Homomorphisms from Integer Ring to Rational Field
sage: QQ.Hom(ZZ)
Set of Homomorphisms from Rational Field to Integer Ring
```

has_coerce_map_from(*x*)
The default for coercion maps between ring homomorphism spaces is very restrictive (until more implementation work is done).

Currently this checks if the domains and the codomains are equal.

EXAMPLES:

```
sage: H = Hom(ZZ, QQ)
sage: H2 = Hom(QQ, ZZ)
sage: H.has_coerce_map_from(H2)
False
```

natural_map()
Returns the natural map from the domain to the codomain.

The natural map is the coercion map from the domain ring to the codomain ring.

EXAMPLES:

```
sage: H = Hom(ZZ, QQ)
sage: H.natural_map()
Ring Coercion morphism:
  From: Integer Ring
  To:   Rational Field
```

class sage.rings.homset.**RingHomset_quo_ring**(*R*, *S*, *category=None*)

Bases: sage.rings.homset.RingHomset_generic

Space of ring homomorphisms where the domain is a (formal) quotient ring.

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: phi = S.hom([b,a]); phi
Ring endomorphism of Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the
  Defn: a |--> b
        b |--> a
sage: phi(a)
b
sage: phi(b)
a
```

TESTS:

We test pickling of a homset from a quotient.

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: H = S.Hom(R)
sage: H == loads(dumps(H))
True
```

We test pickling of actual homomorphisms in a quotient:

```
sage: phi = S.hom([b,a])
sage: phi == loads(dumps(phi))
True
```

sage.rings.homset.**is_RingHomset**(*H*)

Return `True` if `H` is a space of homomorphisms between two rings.

EXAMPLES:

```
sage: from sage.rings.homset import is_RingHomset as is_RH
sage: is_RH(Hom(ZZ, QQ))
True
sage: is_RH(ZZ)
False
sage: is_RH(Hom(RR, CC))
True
sage: is_RH(Hom(FreeModule(ZZ,1), FreeModule(QQ,1)))
False
```

CHAPTER

FOUR

QUOTIENT RINGS

4.1 Quotient Rings

AUTHORS:

- William Stein
- Simon King (2011-04): Put it into the category framework, use the new coercion model.
- Simon King (2011-04): Quotients of non-commutative rings by twosided ideals.

TESTS:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I);
```

Todo

The following skipped tests should be removed once trac ticket #13999 is fixed:

```
sage: TestSuite(S).run(skip=['_test_nonzero_equal', '_test_elements', '_test_zero'])
```

In trac ticket #11068, non-commutative quotient rings R/I were implemented. The only requirement is that the two-sided ideal I provides a `reduce` method so that `I.reduce(x)` is the normal form of an element x with respect to I (i.e., we have `I.reduce(x) == I.reduce(y)` if $x-y \in I$, and `x - I.reduce(x) in I`). Here is a toy example:

```
sage: from sage.rings.noncommutative_ideals import Ideal_nc
sage: class PowerIdeal(Ideal_nc):
...     def __init__(self, R, n):
...         self._power = n
...         self._power = n
...         Ideal_nc.__init__(self,R,[R.prod(m) for m in CartesianProduct(*[R.gens()]*n)])
...     def reduce(self,x):
...         R = self.ring()
...         return add([c*R(m) for m,c in x if len(m)<self._power],R(0))
...
sage: F.<x,y,z> = FreeAlgebra(QQ, 3)
sage: I3 = PowerIdeal(F,3); I3
Twosided Ideal (x^3, x^2*y, x^2*z, x*y*x, x*y^2, x*y*z, x*z*x, x*z*y,
x*z^2, y*x^2, y*x*y, y*x*z, y^2*x, y^3, y^2*z, y*z*x, y*z*y, y*z^2,
z*x^2, z*x*y, z*x*z, z*y*x, z*y^2, z*y*z, z^2*x, z^2*y, z^3) of
Free Algebra on 3 generators (x, y, z) over Rational Field
```

Free algebras have a custom quotient method that serves at creating finite dimensional quotients defined by multiplication matrices. We are bypassing it, so that we obtain the default quotient:

```
sage: Q3.<a,b,c> = F.quotient(I3)
sage: Q3
Quotient of Free Algebra on 3 generators (x, y, z) over Rational Field by
the ideal (x^3, x^2*y, x^2*z, x*y*x, x*y^2, x*y*z, x*z*x, x*z*y, x*z^2,
y*x^2, y*x*y, y*x*z, y^2*x, y^3, y^2*z, y*z*x, y*z*y, y*z^2, z*x^2, z*x*y,
z*x*z, z*y*x, z*y^2, z*y*z, z^2*x, z^2*y, z^3)
sage: (a+b+2)^4
16 + 32*a + 32*b + 24*a^2 + 24*a*b + 24*b*a + 24*b^2
sage: Q3.is_commutative()
False
```

Even though Q_3 is not commutative, there is commutativity for products of degree three:

```
sage: a*(b*c)-(b*c)*a==F.zero()
True
```

If we quotient out all terms of degree two then of course the resulting quotient ring is commutative:

```
sage: I2 = PowerIdeal(F,2); I2
Twosided Ideal (x^2, x*y, x*z, y*x, y^2, y*z, z*x, z*y, z^2) of Free Algebra
on 3 generators (x, y, z) over Rational Field
sage: Q2.<a,b,c> = F.quotient(I2)
sage: Q2.is_commutative()
True
sage: (a+b+2)^4
16 + 32*a + 32*b
```

Since trac ticket #7797, there is an implementation of free algebras based on Singular's implementation of the Letterplace Algebra. Our letterplace wrapper allows to provide the above toy example more easily:

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: Q3 = F.quo(F*[F.prod(m) for m in CartesianProduct(*[F.gens()]*3)]*F)
sage: Q3
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the idea
sage: Q3.0*Q3.1-Q3.1*Q3.0
xbar*ybar - ybar*xbar
sage: Q3.0*(Q3.1*Q3.2)-(Q3.1*Q3.2)*Q3.0
0
sage: Q2 = F.quo(F*[F.prod(m) for m in CartesianProduct(*[F.gens()]*2)]*F)
sage: Q2.is_commutative()
True
```

sage.rings.quotient_ring.**QuotientRing**(*R*, *I*, *names=None*)

Creates a quotient ring of the ring R by the twosided ideal I.

Variables are labeled by `names` (if the quotient ring is a quotient of a polynomial ring). If `names` isn't given, 'bar' will be appended to the variable names in R.

INPUT:

- `R` – a ring.
- `I` – a twosided ideal of R.
- `names` – (optional) a list of strings to be used as names for the variables in the quotient ring R/I.

OUTPUT: R/I - the quotient ring R mod the ideal I

ASSUMPTION:

`I` has a method `I.reduce(x)` returning the normal form of elements $x \in R$. In other words, it is required that `I.reduce(x)==I.reduce(y)` $\iff x - y \in I$, and `x-I.reduce(x) in I`, for all $x, y \in R$.

EXAMPLES:

Some simple quotient rings with the integers:

```
sage: R = QuotientRing(ZZ,7*ZZ); R
Quotient of Integer Ring by the ideal (7)
sage: R.gens()
(1,)
sage: 1*R(3); 6*R(3); 7*R(3)
3
4
0

sage: S = QuotientRing(ZZ,ZZ.ideal(8)); S
Quotient of Integer Ring by the ideal (8)
sage: 2*S(4)
0
```

With polynomial rings (note that the variable name of the quotient ring can be specified as shown below):

```
sage: R.<xx> = QuotientRing(QQ[x], QQ[x].ideal(x^2 + 1)); R
Univariate Quotient Polynomial Ring in xx over Rational Field with modulus x^2 + 1
sage: R.gens(); R.gen()
(xx,)
xx
sage: for n in range(4): xx^n
1
xx
-1
-xx

sage: S = QuotientRing(QQ[x], QQ[x].ideal(x^2 - 2)); S
Univariate Quotient Polynomial Ring in xbar over Rational Field with
modulus x^2 - 2
sage: xbar = S.gen(); S.gen()
xbar
sage: for n in range(3): xbar^n
1
xbar
2
```

Sage coerces objects into ideals when possible:

```
sage: R = QuotientRing(QQ[x], x^2 + 1); R
Univariate Quotient Polynomial Ring in xbar over Rational Field with
modulus x^2 + 1
```

By Noether's homomorphism theorems, the quotient of a quotient ring of R is just the quotient of R by the sum of the ideals. In this example, we end up modding out the ideal (x) from the ring $\mathbf{Q}[x, y]$:

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R,R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S,S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x, y^2 + 1)
sage: R.gens(); S.gens(); T.gens()
(x, y)
```

```
(a, b)
(0, d)
sage: for n in range(4): d^n
1
d
-1
-d
```

TESTS:

By trac ticket #11068, the following does not return a generic quotient ring but a usual quotient of the integer ring:

```
sage: R = Integers(8)
sage: I = R.ideal(2)
sage: R.quotient(I)
Ring of integers modulo 2
```

Here is an example of the quotient of a free algebra by a twosided homogeneous ideal (see trac ticket #7797):

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q.<a,b,c> = F.quo(I); Q
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: a*b
-b*c
sage: a^3
-b*c*a - b*c*b - b*c*c
sage: J = Q*[a^3-b^3]*Q
sage: R.<i,j,k> = Q.quo(J); R
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: i^3
-j*k*i - j*k*j - j*k*k
sage: j^3
-j*k*i - j*k*j - j*k*k
```

Check that trac ticket #5978 is fixed by if we quotient by the zero ideal (0) then we just return `R`:

```
sage: R = QQ['x']
sage: R.quotient(R.zero_ideal())
Univariate Polynomial Ring in x over Rational Field
sage: R.<x> = PolynomialRing(ZZ)
sage: R is R.quotient(R.zero_ideal())
True
sage: I = R.ideal(0)
sage: R is R.quotient(I)
True
```

class sage.rings.quotient_ring.**QuotientRing_generic**(*R*, *I*, *names*, *category=None*)

Bases: sage.rings.quotient_ring.QuotientRing_nc, sage.rings.ring.CommutativeRing

Creates a quotient ring of a *commutative* ring R by the ideal I.

EXAMPLE:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I); S
Quotient of Univariate Polynomial Ring in x over Integer Ring by the ideal (x^2 + 3*x + 4, x^2 +
```

class sage.rings.quotient_ring.**QuotientRing_nc**(*R*, *I*, *names*, *category=None*)
Bases: sage.rings.ring.Ring, sage.structure.parent_gens.ParentWithGens

The quotient ring of R by a twosided ideal I.

This class is for rings that do not inherit from CommutativeRing.

EXAMPLES:

Here is a quotient of a free algebra by a twosided homogeneous ideal:

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q.<a,b,c> = F.quo(I); Q
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: a*b
-b*c
sage: a^3
-b*c*a - b*c*b - b*c*c
```

A quotient of a quotient is just the quotient of the original top ring by the sum of two ideals:

```
sage: J = Q*[a^3-b^3]*Q
sage: R.<i,j,k> = Q.quo(J); R
Quotient of Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Field by the
sage: i^3
-j*k*i - j*k*j - j*k*k
sage: j^3
-j*k*i - j*k*j - j*k*k
```

For rings that *do* inherit from CommutativeRing, we provide a subclass QuotientRing_generic, for backwards compatibility.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(ZZ,'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I); S
Quotient of Univariate Polynomial Ring in x over Integer Ring by the ideal (x^2 + 3*x + 4, x^2 +

sage: R.<x,y> = PolynomialRing(QQ)
sage: S.<a,b> = R.quo(x^2 + y^2)
sage: a^2 + b^2 == 0
True
sage: S(0) == a^2 + b^2
True
```

Again, a quotient of a quotient is just the quotient of the original top ring by the sum of two ideals.

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = R.quo(1 + y^2)
sage: T.<c,d> = S.quo(a)
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x, y^2 + 1)
sage: T.gens()
(0, d)
```

Element
alias of QuotientRingElement

ambient()
Returns the cover ring of the quotient ring: that is, the original ring R from which we modded out an ideal,

I.

EXAMPLES:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.cover_ring()
Integer Ring

sage: Q = QuotientRing(QQ[x], x^2 + 1)
sage: Q.cover_ring()
Univariate Polynomial Ring in x over Rational Field
```

characteristic()

Return the characteristic of the quotient ring.

Todo

Not yet implemented!

EXAMPLES:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.characteristic()
Traceback (most recent call last):
...
NotImplementedError
```

construction()

Returns the functorial construction of `self`.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(ZZ,'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: R.quotient_ring(I).construction()
(QuotientFunctor, Univariate Polynomial Ring in x over Integer Ring)
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: Q.construction()
(QuotientFunctor, Free Associative Unital Algebra on 3 generators (x, y, z) over Rational Fi
```

TESTS:

```
sage: F, R = Integers(5).construction()
sage: F(R)
Ring of integers modulo 5
sage: F, R = GF(5).construction()
sage: F(R)
Finite Field of size 5
```

cover()

The covering ring homomorphism $R \to R/I$, equipped with a section.

EXAMPLES:

```
sage: R = ZZ.quo(3*ZZ)
sage: pi = R.cover()
sage: pi
Ring morphism:
  From: Integer Ring
```

```
  To:   Ring of integers modulo 3
  Defn: Natural quotient map
sage: pi(5)
2
sage: l = pi.lift()

sage: R.<x,y>  = PolynomialRing(QQ)
sage: Q = R.quo( (x^2,y^2) )
sage: pi = Q.cover()
sage: pi(x^3+y)
ybar
sage: l = pi.lift(x+y^3)
sage: l
x
sage: l = pi.lift(); l
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
sage: l(x+y^3)
x
```

cover_ring()

Returns the cover ring of the quotient ring: that is, the original ring R from which we modded out an ideal, I.

EXAMPLES:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.cover_ring()
Integer Ring

sage: Q = QuotientRing(QQ[x], x^2 + 1)
sage: Q.cover_ring()
Univariate Polynomial Ring in x over Rational Field
```

defining_ideal()

Returns the ideal generating this quotient ring.

EXAMPLES:

In the integers:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.defining_ideal()
Principal ideal (7) of Integer Ring
```

An example involving a quotient of a quotient. By Noether's homomorphism theorems, this is actually a quotient by a sum of two ideals:

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R,R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S,S.ideal(a))
sage: S.defining_ideal()
Ideal (y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field
sage: T.defining_ideal()
Ideal (x, y^2 + 1) of Multivariate Polynomial Ring in x, y over Rational Field
```

gen(*i=0*)

Returns the i-th generator for this quotient ring.

EXAMPLES:

```
sage: R = QuotientRing(ZZ,7*ZZ)
sage: R.gen(0)
1

sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R,R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S,S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x, y^2 +
sage: R.gen(0); R.gen(1)
x
y
sage: S.gen(0); S.gen(1)
a
b
sage: T.gen(0); T.gen(1)
0
d
```

ideal(**gens*, ***kwds*)

Return the ideal of `self` with the given generators.

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: S = R.quotient_ring(x^2+y^2)
sage: S.ideal()
Ideal (0) of Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ide
sage: S.ideal(x+y+1)
Ideal (xbar + ybar + 1) of Quotient of Multivariate Polynomial Ring in x, y over Rational Fi
```

TESTS:

We create an ideal of a fairly generic integer ring (see trac ticket #5666):

```
sage: R = Integers(10)
sage: R.ideal(1)
Principal ideal (1) of Ring of integers modulo 10
```

is_commutative()

Tell whether this quotient ring is commutative.

Note: This is certainly the case if the cover ring is commutative. Otherwise, if this ring has a finite number of generators, it is tested whether they commute. If the number of generators is infinite, a `NotImplementedError` is raised.

AUTHOR:

- Simon King (2011-03-23): See trac ticket #7797.

EXAMPLES:

Any quotient of a commutative ring is commutative:

```
sage: P.<a,b,c> = QQ[]
sage: P.quo(P.random_element()).is_commutative()
True
```

The non-commutative case is more interesting:

```
sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: Q.is_commutative()
False
sage: Q.1*Q.2==Q.2*Q.1
False
```

In the next example, the generators apparently commute:

```
sage: J = F*[x*y-y*x,x*z-z*x,y*z-z*y,x^3-y^3]*F
sage: R = F.quo(J)
sage: R.is_commutative()
True
```

is_field(*proof=True*)

Returns `True` if the quotient ring is a field. Checks to see if the defining ideal is maximal.

TESTS:

```
sage: Q = QuotientRing(ZZ,7*ZZ)
sage: Q.is_field()
True
```

Requires the `is_maximal` method of the defining ideal to be implemented:

```
sage: R.<x, y> = ZZ[]
sage: R.quotient_ring(R.ideal([2, 4 +x])).is_field()
Traceback (most recent call last):
...
NotImplementedError
```

is_integral_domain(*proof=True*)

With `proof` equal to `True` (the default), this function may raise a `NotImplementedError`.

When `proof` is `False`, if `True` is returned, then self is definitely an integral domain. If the function returns `False`, then either `self` is not an integral domain or it was unable to determine whether or not `self` is an integral domain.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: R.quo(x^2 - y).is_integral_domain()
True
sage: R.quo(x^2 - y^2).is_integral_domain()
False
sage: R.quo(x^2 - y^2).is_integral_domain(proof=False)
False
sage: R.<a,b,c> = ZZ[]
sage: Q = R.quotient_ring([a, b])
sage: Q.is_integral_domain()
Traceback (most recent call last):
...
NotImplementedError
sage: Q.is_integral_domain(proof=False)
False
```

is_noetherian()

Return `True` if this ring is Noetherian.

EXAMPLES:

```
sage: R = QuotientRing(ZZ, 102*ZZ)
sage: R.is_noetherian()
True

sage: R = QuotientRing(QQ[x], x^2+1)
sage: R.is_noetherian()
True
```

If the cover ring of `self` is not Noetherian, we currently have no way of testing whether `self` is Noetherian, so we raise an error:

```
sage: R.<x> = InfinitePolynomialRing(QQ)
sage: R.is_noetherian()
False
sage: I = R.ideal([x[1]^2, x[2]])
sage: S = R.quotient(I)
sage: S.is_noetherian()
Traceback (most recent call last):
...
NotImplementedError
```

lift (*x=None*)

Return the lifting map to the cover, or the image of an element under the lifting map.

Note: The category framework imposes that `Q.lift(x)` returns the image of an element x under the lifting map. For backwards compatibility, we let `Q.lift()` return the lifting map.

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.lift()
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
sage: S.lift(S.0) == x
True
```

lifting_map()

Return the lifting map to the cover.

EXAMPLES:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: pi = S.cover(); pi
Ring morphism:
  From: Multivariate Polynomial Ring in x, y over Rational Field
  To:   Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x
  Defn: Natural quotient map
sage: L = S.lifting_map(); L
Set-theoretic ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x
  To:   Multivariate Polynomial Ring in x, y over Rational Field
  Defn: Choice of lifting map
sage: L(S.0)
x
sage: L(S.1)
```

```
y
```

Note that some reduction may be applied so that the lift of a reduction need not equal the original element:

```
sage: z = pi(x^3 + 2*y^2); z
-xbar*ybar^2 + 2*ybar^2
sage: L(z)
-x*y^2 + 2*y^2
sage: L(z) == x^3 + 2*y^2
False
```

Test that there also is a lift for rings that are no instances of `Ring` (see trac ticket #11068):

```
sage: MS = MatrixSpace(GF(5),2,2)
sage: I = MS*[MS.0*MS.1,MS.2+MS.3]*MS
sage: Q = MS.quo(I)
sage: Q.lift()
Set-theoretic ring morphism:
  From: Quotient of Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5 by
(
  [0 1]
  [0 0],

  [0 0]
  [1 1]
)

  To:   Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 5
  Defn: Choice of lifting map
```

ngens()

Returns the number of generators for this quotient ring.

Todo

Note that `ngens` counts 0 as a generator. Does this make sense? That is, since 0 only generates itself and the fact that this is true for all rings, is there a way to "knock it off" of the generators list if a generator of some original ring is modded out?

EXAMPLES:

```
sage: R = QuotientRing(ZZ,7*ZZ)
sage: R.gens(); R.ngens()
(1,)
1

sage: R.<x,y> = PolynomialRing(QQ,2)
sage: S.<a,b> = QuotientRing(R,R.ideal(1 + y^2))
sage: T.<c,d> = QuotientRing(S,S.ideal(a))
sage: T
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x, y^2 +
sage: R.gens(); S.gens(); T.gens()
(x, y)
(a, b)
(0, d)
sage: R.ngens(); S.ngens(); T.ngens()
2
2
2
```

retract(*x*)

The image of an element of the cover ring under the quotient map.

INPUT:

- x – An element of the cover ring

OUTPUT:

The image of the given element in `self`.

EXAMPLE:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quotient(x^2 + y^2)
sage: S.retract((x+y)^2)
2*xbar*ybar
```

term_order()

Return the term order of this ring.

EXAMPLES:

```
sage: P.<a,b,c> = PolynomialRing(QQ)
sage: I = Ideal([a^2 - a, b^2 - b, c^2 - c])
sage: Q = P.quotient(I)
sage: Q.term_order()
Degree reverse lexicographic term order
```

sage.rings.quotient_ring.**is_QuotientRing**(*x*)

Tests whether or not x inherits from QuotientRing_nc.

EXAMPLES:

```
sage: from sage.rings.quotient_ring import is_QuotientRing
sage: R.<x> = PolynomialRing(ZZ,'x')
sage: I = R.ideal([4 + 3*x + x^2, 1 + x^2])
sage: S = R.quotient_ring(I)
sage: is_QuotientRing(S)
True
sage: is_QuotientRing(R)
False

sage: F.<x,y,z> = FreeAlgebra(QQ, implementation='letterplace')
sage: I = F*[x*y+y*z,x^2+x*y-y*x-y^2]*F
sage: Q = F.quo(I)
sage: is_QuotientRing(Q)
True
sage: is_QuotientRing(F)
False
```

4.2 Quotient Ring Elements

AUTHORS:

- William Stein

class sage.rings.quotient_ring_element.**QuotientRingElement**(*parent*, *rep*, *reduce=True*)

Bases: sage.structure.element.RingElement

An element of a quotient ring R/I.

INPUT:

- `parent` - the ring R/I
- `rep` - a representative of the element in R; this is used as the internal representation of the element
- `reduce` - bool (optional, default: True) - if True, then the internal representation of the element is `rep` reduced modulo the ideal I

EXAMPLES:

```
sage: R.<x> = PolynomialRing(ZZ)
sage: S.<xbar> = R.quo((4 + 3*x + x^2, 1 + x^2)); S
Quotient of Univariate Polynomial Ring in x over Integer Ring by the ideal (x^2 + 3*x + 4, x^2 +
sage: v = S.gens(); v
(xbar,)

sage: loads(v[0].dumps()) == v[0]
True

sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S = R.quo(x^2 + y^2); S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2 + y^2)
sage: S.gens()
(xbar, ybar)
```

We name each of the generators.

```
sage: S.<a,b> = R.quotient(x^2 + y^2)
sage: a
a
sage: b
b
sage: a^2 + b^2 == 0
True
sage: b.lift()
y
sage: (a^3 + b^2).lift()
-x*y^2 + y^2
```

is_unit()

Return True if self is a unit in the quotient ring.

TODO: This is not fully implemented, as illustrated in the example below. So far, self is determined to be unit only if its representation in the cover ring R is also a unit.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(1 - x*y); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: a*b
1
sage: a.is_unit()
Traceback (most recent call last):
...
NotImplementedError
sage: S(1).is_unit()
True
```

lc()

Return the leading coefficent of this quotient ring element.

EXAMPLE:

```
sage: R.<x,y,z>=PolynomialRing(GF(7),3,order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo( I )
sage: f = Q( z*y + 2*x )
sage: f.lc()
2
```

TESTS:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: (a+3*a*b+b).lc()
3
```

lift()

If self is an element of R/I, then return self as an element of R.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: a.lift()
x
sage: (3/5*(a + a^2 + b^2)).lift()
3/5*x
```

lm()

Return the leading monomial of this quotient ring element.

EXAMPLE:

```
sage: R.<x,y,z>=PolynomialRing(GF(7),3,order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo( I )
sage: f = Q( z*y + 2*x )
sage: f.lm()
xbar
```

TESTS:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: (a+3*a*b+b).lm()
a*b
```

lt()

Return the leading term of this quotient ring element.

EXAMPLE:

```
sage: R.<x,y,z>=PolynomialRing(GF(7),3,order='lex')
sage: I = sage.rings.ideal.FieldIdeal(R)
sage: Q = R.quo( I )
sage: f = Q( z*y + 2*x )
sage: f.lt()
2*xbar
```

TESTS:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: (a+3*a*b+b).lt()
3*a*b
```

monomials()

Return the monomials in `self`.

OUTPUT:

A list of monomials.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: a.monomials()
[a]
sage: (a+a*b).monomials()
[a*b, a]
sage: R.zero().monomials()
[]
```

reduce(*G*)

Reduce this quotient ring element by a set of quotient ring elements `G`.

INPUT:

- `G` - a list of quotient ring elements

EXAMPLE:

```
sage: P.<a,b,c,d,e> = PolynomialRing(GF(2), 5, order='lex')
sage: I1 = ideal([a*b + c*d + 1, a*c*e + d*e, a*b*e + c*e, b*c + c*d*e + 1])
sage: Q = P.quotient( sage.rings.ideal.FieldIdeal(P) )
sage: I2 = ideal([Q(f) for f in I1.gens()])
sage: f = Q((a*b + c*d + 1)^2  + e)
sage: f.reduce(I2.gens())
ebar
```

variables()

Return all variables occurring in `self`.

OUTPUT:

A tuple of linear monomials, one for each variable occurring in `self`.

EXAMPLES:

```
sage: R.<x,y> = QQ[]; S.<a,b> = R.quo(x^2 + y^2); type(a)
<class 'sage.rings.quotient_ring_element.QuotientRing_generic_with_category.element_class'>
sage: a.variables()
(a,)
sage: b.variables()
(b,)
sage: s = a^2 + b^2 + 1; s
1
sage: s.variables()
()
sage: (a+b).variables()
(a, b)
```

CHAPTER

FIVE

FRACTION FIELDS

5.1 Fraction Field of Integral Domains

AUTHORS:

- William Stein (with input from David Joyner, David Kohel, and Joe Wetherell)
- Burcin Erocal

EXAMPLES:

Quotienting is a constructor for an element of the fraction field:

```
sage: R.<x> = QQ[]
sage: (x^2-1)/(x+1)
x - 1
sage: parent((x^2-1)/(x+1))
Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

The GCD is not taken (since it doesn't converge sometimes) in the inexact case:

```
sage: Z.<z> = CC[]
sage: I = CC.gen()
sage: (1+I+z)/(z+0.1*I)
(z + 1.00000000000000 + I)/(z + 0.100000000000000*I)
sage: (1+I*z)/(z+1.1)
(I*z + 1.00000000000000)/(z + 1.10000000000000)
```

TESTS:

```
sage: F = FractionField(IntegerRing())
sage: F == loads(dumps(F))
True

sage: F = FractionField(PolynomialRing(RationalField(),'x'))
sage: F == loads(dumps(F))
True

sage: F = FractionField(PolynomialRing(IntegerRing(),'x'))
sage: F == loads(dumps(F))
True

sage: F = FractionField(PolynomialRing(RationalField(),2,'x'))
sage: F == loads(dumps(F))
True
```

sage.rings.fraction_field.**FractionField**(*R*, *names=None*)

Create the fraction field of the integral domain R.

INPUT:

- R – an integral domain
- names – ignored

EXAMPLES:

We create some example fraction fields:

```
sage: FractionField(IntegerRing())
Rational Field
sage: FractionField(PolynomialRing(RationalField(),'x'))
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: FractionField(PolynomialRing(IntegerRing(),'x'))
Fraction Field of Univariate Polynomial Ring in x over Integer Ring
sage: FractionField(PolynomialRing(RationalField(),2,'x'))
Fraction Field of Multivariate Polynomial Ring in x0, x1 over Rational Field
```

Dividing elements often implicitly creates elements of the fraction field:

```
sage: x = PolynomialRing(RationalField(), 'x').gen()
sage: f = x/(x+1)
sage: g = x**3/(x+1)
sage: f/g
1/x^2
sage: g/f
x^2
```

The input must be an integral domain:

```
sage: Frac(Integers(4))
Traceback (most recent call last):
...
TypeError: R must be an integral domain.
```

class sage.rings.fraction_field.**FractionField_1poly_field**(*R*, *element_class=<class 'sage.rings.fraction_field_element.FractionFieldEle*

Bases: sage.rings.fraction_field.FractionField_generic

The fraction field of a univariate polynomial ring over a field.

Many of the functions here are included for coherence with number fields.

class_number()

Here for compatibility with number fields and function fields.

EXAMPLES:

```
sage: R.<t> = GF(5)[]; K = R.fraction_field()
sage: K.class_number()
1
```

maximal_order()

Returns the maximal order in this fraction field.

EXAMPLES:

```
sage: K = FractionField(GF(5)['t'])
sage: K.maximal_order()
Univariate Polynomial Ring in t over Finite Field of size 5
```

ring_of_integers()
Returns the ring of integers in this fraction field.

EXAMPLES:

```
sage: K = FractionField(GF(5)['t'])
sage: K.ring_of_integers()
Univariate Polynomial Ring in t over Finite Field of size 5
```

class sage.rings.fraction_field.**FractionField_generic**(*R*, *element_class=<type 'sage.rings.fraction_field_element.FractionFieldElement'>*, *category=Category of quotient fields*)

Bases: sage.rings.ring.Field

The fraction field of an integral domain.

base_ring()
Return the base ring of `self`; this is the base ring of the ring which this fraction field is the fraction field of.

EXAMPLES:

```
sage: R = Frac(ZZ['t'])
sage: R.base_ring()
Integer Ring
```

characteristic()
Return the characteristic of this fraction field.

EXAMPLES:

```
sage: R = Frac(ZZ['t'])
sage: R.base_ring()
Integer Ring
sage: R = Frac(ZZ['t']); R.characteristic()
0
sage: R = Frac(GF(5)['w']); R.characteristic()
5
```

construction()
EXAMPLES:

```
sage: Frac(ZZ['x']).construction()
(FractionField, Univariate Polynomial Ring in x over Integer Ring)
sage: K = Frac(GF(3)['t'])
sage: f, R = K.construction()
sage: f(R)
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 3
sage: f(R) == K
True
```

gen(*i=0*)
Return the `i`-th generator of `self`.

EXAMPLES:

```
sage: R = Frac(PolynomialRing(QQ,'z',10)); R
Fraction Field of Multivariate Polynomial Ring in z0, z1, z2, z3, z4, z5, z6, z7, z8, z9 ove
sage: R.0
z0
sage: R.gen(3)
```

```
z3
sage: R.3
z3
```

is_exact()
Return if `self` is exact which is if the underlying ring is exact.

EXAMPLES:

```
sage: Frac(ZZ['x']).is_exact()
True
sage: Frac(CDF['x']).is_exact()
False
```

is_field(*proof=True*)
Return `True`, since the fraction field is a field.

EXAMPLES:

```
sage: Frac(ZZ).is_field()
True
```

is_finite()
Tells whether this fraction field is finite.

Note: A fraction field is finite if and only if the associated integral domain is finite.

EXAMPLE:

```
sage: Frac(QQ['a','b','c']).is_finite()
False
```

ngens()
This is the same as for the parent object.

EXAMPLES:

```
sage: R = Frac(PolynomialRing(QQ,'z',10)); R
Fraction Field of Multivariate Polynomial Ring in z0, z1, z2, z3, z4, z5, z6, z7, z8, z9 ove
sage: R.ngens()
10
```

random_element(**args*, ***kwds*)
Returns a random element in this fraction field.

The arguments are passed to the random generator of the underlying ring.

EXAMPLES:

```
sage: F = ZZ['x'].fraction_field()
sage: F.random_element()  # random
(2*x - 8)/(-x^2 + x)

sage: f = F.random_element(degree=5)
sage: f.numerator().degree()
5
sage: f.denominator().degree()
5
```

ring()
Return the ring that this is the fraction field of.

EXAMPLES:

```
sage: R = Frac(QQ['x,y'])
sage: R
Fraction Field of Multivariate Polynomial Ring in x, y over Rational Field
sage: R.ring()
Multivariate Polynomial Ring in x, y over Rational Field
```

sage.rings.fraction_field.**is_FractionField**(*x*)

Test whether or not x inherits from FractionField_generic.

EXAMPLES:

```
sage: from sage.rings.fraction_field import is_FractionField
sage: is_FractionField(Frac(ZZ['x']))
True
sage: is_FractionField(QQ)
False
```

5.2 Fraction Field Elements

AUTHORS:

- William Stein (input from David Joyner, David Kohel, and Joe Wetherell)
- Sebastian Pancratz (2010-01-06): Rewrite of addition, multiplication and derivative to use Henrici's algorithms [Ho72]

REFERENCES:

class sage.rings.fraction_field_element.**FractionFieldElement**

Bases: sage.structure.element.FieldElement

EXAMPLES:

```
sage: K = FractionField(PolynomialRing(QQ, 'x'))
sage: K
Fraction Field of Univariate Polynomial Ring in x over Rational Field
sage: loads(K.dumps()) == K
True
sage: x = K.gen()
sage: f = (x^3 + x)/(17 - x^19); f
(x^3 + x)/(-x^19 + 17)
sage: loads(f.dumps()) == f
True
```

TESTS:

Test if trac ticket #5451 is fixed:

```
sage: A = FiniteField(9,'theta')['t']
sage: K.<t> = FractionField(A)
sage: f= 2/(t^2+2*t); g =t^9/(t^18 + t^10 + t^2);f+g
(2*t^15 + 2*t^14 + 2*t^13 + 2*t^12 + 2*t^11 + 2*t^10 + 2*t^9 + t^7 + t^6 + t^5 + t^4 + t^3 + t^2
```

Test if trac ticket #8671 is fixed:

```
sage: P.<n> = QQ[]
sage: F = P.fraction_field()
sage: P.one()//F.one()
1
```

```
sage: F.one().quo_rem(F.one())
(1, 0)
```

denominator()
Return the denominator of `self`.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: f = x/y+1; f
(x + y)/y
sage: f.denominator()
y
```

is_one()
Return `True` if this element is equal to one.

EXAMPLES:

```
sage: F = ZZ['x,y'].fraction_field()
sage: x,y = F.gens()
sage: (x/x).is_one()
True
sage: (x/y).is_one()
False
```

is_square(*root=False*)
Returns whether or not `self` is a perfect square. If the optional argument `root` is `True`, then also returns a square root (or `None`, if the fraction field element is not square).

INPUT:

- `root` – whether or not to also return a square root (default: `False`)

OUTPUT:

- `bool` - whether or not a square
- `object` - (optional) an actual square root if found, and None otherwise.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: (1/t).is_square()
False
sage: (1/t^6).is_square()
True
sage: ((1+t)^4/t^6).is_square()
True
sage: (4*(1+t)^4/t^6).is_square()
True
sage: (2*(1+t)^4/t^6).is_square()
False
sage: ((1+t)/t^6).is_square()
False

sage: (4*(1+t)^4/t^6).is_square(root=True)
(True, (2*t^2 + 4*t + 2)/t^3)
sage: (2*(1+t)^4/t^6).is_square(root=True)
(False, None)

sage: R.<x> = QQ[]
```

```
sage: a = 2*(x+1)^2 / (2*(x-1)^2); a
(2*x^2 + 4*x + 2)/(2*x^2 - 4*x + 2)
sage: a.numerator().is_square()
False
sage: a.is_square()
True
sage: (0/x).is_square()
True
```

is_zero()

Return `True` if this element is equal to zero.

EXAMPLES:

```
sage: F = ZZ['x,y'].fraction_field()
sage: x,y = F.gens()
sage: t = F(0)/x
sage: t.is_zero()
True
sage: u = 1/x - 1/x
sage: u.is_zero()
True
sage: u.parent() is F
True
```

numerator()

Return the numerator of `self`.

EXAMPLES:

```
sage: R.<x,y> = ZZ[]
sage: f = x/y+1; f
(x + y)/y
sage: f.numerator()
x + y
```

reduce()

Divides out the gcd of the numerator and denominator.

Automatically called for exact rings, but because it may be numerically unstable for inexact rings it must be called manually in that case.

EXAMPLES:

```
sage: R.<x> = RealField(10)[]
sage: f = (x^2+2*x+1)/(x+1); f
(x^2 + 2.0*x + 1.0)/(x + 1.0)
sage: f.reduce(); f
x + 1.0
```

valuation(*v=None*)

Return the valuation of `self`, assuming that the numerator and denominator have valuation functions defined on them.

EXAMPLES:

```
sage: x = PolynomialRing(RationalField(),'x').gen()
sage: f = (x^3 + x)/(x^2 - 2*x^3)
sage: f
(x^2 + 1)/(-2*x^2 + x)
sage: f.valuation()
```

```
-1
sage: f.valuation(x^2+1)
1
```

class sage.rings.fraction_field_element.**FractionFieldElement_1poly_field**

Bases: sage.rings.fraction_field_element.FractionFieldElement

A fraction field element where the parent is the fraction field of a univariate polynomial ring.

Many of the functions here are included for coherence with number fields.

is_integral()

Returns whether this element is actually a polynomial.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: elt = (t^2 + t - 2) / (t + 2); elt # == (t + 2)*(t - 1)/(t + 2)
t - 1
sage: elt.is_integral()
True
sage: elt = (t^2 - t) / (t+2); elt # == t*(t - 1)/(t + 2)
(t^2 - t)/(t + 2)
sage: elt.is_integral()
False
```

support()

Returns a sorted list of primes dividing either the numerator or denominator of this element.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: h = (t^14 + 2*t^12 - 4*t^11 - 8*t^9 + 6*t^8 + 12*t^6 - 4*t^5 - 8*t^3 + t^2 + 2)/(t^6 +
sage: h.support()
[t - 1, t + 3, t^2 + 2, t^2 + t + 1, t^4 - 2]
```

sage.rings.fraction_field_element.**is_FractionFieldElement**(*x*)

Returns whether or not x is a :class‘FractionFieldElement‘.

EXAMPLES:

```
sage: from sage.rings.fraction_field_element import is_FractionFieldElement
sage: R.<x> = ZZ[]
sage: is_FractionFieldElement(x/2)
False
sage: is_FractionFieldElement(2/x)
True
sage: is_FractionFieldElement(1/3)
False
```

sage.rings.fraction_field_element.**make_element**(*parent*, *numerator*, *denominator*)

Used for unpickling FractionFieldElement objects (and subclasses).

EXAMPLES:

```
sage: from sage.rings.fraction_field_element import make_element
sage: R = ZZ['x,y']
sage: x,y = R.gens()
sage: F = R.fraction_field()
sage: make_element(F, 1+x, 1+y)
(x + 1)/(y + 1)
```

sage.rings.fraction_field_element.**make_element_old**(*parent*, *cdict*)

Used for unpickling old FractionFieldElement pickles.

EXAMPLES:

```
sage: from sage.rings.fraction_field_element import make_element_old
sage: R.<x,y> = ZZ[]
sage: F = R.fraction_field()
sage: make_element_old(F, {'_FractionFieldElement__numerator':x+y,'_FractionFieldElement__denomi
(x + y)/(x - y)
```

CHAPTER

SIX

UTILITIES

6.1 Big O for various types (power series, p-adics, etc.)

sage.rings.big_oh.**O**(**x*, ***kwds*)

Big O constructor for various types.

EXAMPLES:

This is useful for writing power series elements:

```
sage: R.<t> = ZZ[['t']]
sage: (1+t)^10 + O(t^5)
1 + 10*t + 45*t^2 + 120*t^3 + 210*t^4 + O(t^5)
```

A power series ring is created implicitly if a polynomial element is passed:

```
sage: R.<x> = QQ['x']
sage: O(x^100)
O(x^100)
sage: 1/(1+x+O(x^5))
1 - x + x^2 - x^3 + x^4 + O(x^5)
sage: R.<u,v> = QQ[[]]
sage: 1 + u + v^2 + O(u, v)^5
1 + u + v^2 + O(u, v)^5
```

This is also useful to create p-adic numbers:

```
sage: O(7^6)
O(7^6)
sage: 1/3 + O(7^6)
5 + 4*7 + 4*7^2 + 4*7^3 + 4*7^4 + 4*7^5 + O(7^6)
```

It behaves well with respect to adding negative powers of p:

```
sage: a = O(11^-32); a
O(11^-32)
sage: a.parent()
11-adic Field with capped relative precision 20
```

There are problems if you add a rational with very negative valuation to an O-Term:

```
sage: 11^-12 + O(11^15)
11^-12 + O(11^8)
```

The reason that this fails is that the constructor doesn't know the right precision cap to use. If you cast explicitly or use other means of element creation, you can get around this issue:

```
sage: K = Qp(11, 30)
sage: K(11^-12) + O(11^15)
11^-12 + O(11^15)
sage: 11^-12 + K(O(11^15))
11^-12 + O(11^15)
sage: K(11^-12, absprec = 15)
11^-12 + O(11^15)
sage: K(11^-12, 15)
11^-12 + O(11^15)
```

TESTS:

```
sage: var('x, y')
(x, y)
sage: O(x)
Traceback (most recent call last):
...
ArithmeticError: O(x) not defined
sage: O(y)
Traceback (most recent call last):
...
ArithmeticError: O(y) not defined
sage: O(x, y)
Traceback (most recent call last):
...
ArithmeticError: O(x, y) not defined
sage: O(4, 2)
Traceback (most recent call last):
...
ArithmeticError: O(4, 2) not defined
```

6.2 Signed and Unsigned Infinities

The unsigned infinity "ring" is the set of two elements

1. infinity
2. A number less than infinity

The rules for arithmetic are that the unsigned infinity ring does not canonically coerce to any other ring, and all other rings canonically coerce to the unsigned infinity ring, sending all elements to the single element "a number less than infinity" of the unsigned infinity ring. Arithmetic and comparisons then take place in the unsigned infinity ring, where all arithmetic operations that are well-defined are defined.

The infinity "ring" is the set of five elements

1. plus infinity
2. a positive finite element
3. zero
4. a negative finite element
5. negative infinity

The infinity ring coerces to the unsigned infinity ring, sending the infinite elements to infinity and the non-infinite elements to "a number less than infinity." Any ordered ring coerces to the infinity ring in the obvious way.

Note: The shorthand `oo` is predefined in Sage to be the same as `+Infinity` in the infinity ring. It is considered equal to, but not the same as `Infinity` in the `UnsignedInfinityRing`.

EXAMPLES:

We fetch the unsigned infinity ring and create some elements:

```
sage: P = UnsignedInfinityRing; P
The Unsigned Infinity Ring
sage: P(5)
A number less than infinity
sage: P.ngens()
1
sage: unsigned_oo = P.0; unsigned_oo
Infinity
```

We compare finite numbers with infinity:

```
sage: 5 < unsigned_oo
True
sage: 5 > unsigned_oo
False
sage: unsigned_oo < 5
False
sage: unsigned_oo > 5
True
```

Demonstrating the shorthand `oo` versus `Infinity`:

```
sage: oo
+Infinity
sage: oo is InfinityRing.0
True
sage: oo is UnsignedInfinityRing.0
False
sage: oo == UnsignedInfinityRing.0
True
```

We do arithmetic:

```
sage: unsigned_oo + 5
Infinity
```

We make `1 / unsigned_oo` return the integer 0 so that arithmetic of the following type works:

```
sage: (1/unsigned_oo) + 2
2
sage: 32/5 - (2.439/unsigned_oo)
32/5
```

Note that many operations are not defined, since the result is not well-defined:

```
sage: unsigned_oo/0
Traceback (most recent call last):
...
ValueError: unsigned oo times smaller number not defined
```

What happened above is that 0 is canonically coerced to "a number less than infinity" in the unsigned infinity ring, and the quotient is then not well-defined.

```
sage: 0/unsigned_oo
0
sage: unsigned_oo * 0
Traceback (most recent call last):
...
ValueError: unsigned oo times smaller number not defined
sage: unsigned_oo/unsigned_oo
Traceback (most recent call last):
...
ValueError: unsigned oo times smaller number not defined
```

In the infinity ring, we can negate infinity, multiply positive numbers by infinity, etc.

```
sage: P = InfinityRing; P
The Infinity Ring
sage: P(5)
A positive finite number
```

The symbol `oo` is predefined as a shorthand for `+Infinity`:

```
sage: oo
+Infinity
```

We compare finite and infinite elements:

```
sage: 5 < oo
True
sage: P(-5) < P(5)
True
sage: P(2) < P(3)
False
sage: -oo < oo
True
```

We can do more arithmetic than in the unsigned infinity ring:

```
sage: 2 * oo
+Infinity
sage: -2 * oo
-Infinity
sage: 1 - oo
-Infinity
sage: 1 / oo
0
sage: -1 / oo
0
```

We make `1 / oo` and `1 / -oo` return the integer 0 instead of the infinity ring Zero so that arithmetic of the following type works:

```
sage: (1/oo) + 2
2
sage: 32/5 - (2.439/-oo)
32/5
```

If we try to subtract infinities or multiply infinity by zero we still get an error:

```
sage: oo - oo
Traceback (most recent call last):
...
SignError: cannot add infinity to minus infinity
sage: 0 * oo
Traceback (most recent call last):
...
SignError: cannot multiply infinity by zero
sage: P(2) + P(-3)
Traceback (most recent call last):
...
SignError: cannot add positive finite value to negative finite value
```

Signed infinity can also be represented by RR / RDF elements. But unsigned infinity cannot:

```
sage: oo in RR, oo in RDF
(True, True)
sage: unsigned_infinity in RR, unsigned_infinity in RDF
(False, False)
```

TESTS:

```
sage: P = InfinityRing
sage: P == loads(dumps(P))
True
```

```
sage: P(2) == loads(dumps(P(2)))
True
```

The following is assumed in a lot of code (i.e., "is" is used for testing whether something is infinity), so make sure it is satisfied:

```
sage: loads(dumps(infinity)) is infinity
True
```

We check that trac ticket #17990 is fixed:

```
sage: m = Matrix([Infinity])
sage: m.rows()
[(+Infinity)]
```

class sage.rings.infinity.**AnInfinity**

Bases: object

lcm(*x*)

Return the least common multiple of oo and x, which is by definition oo unless x is 0.

EXAMPLES:

```
sage: oo.lcm(0)
0
sage: oo.lcm(oo)
+Infinity
sage: oo.lcm(-oo)
+Infinity
sage: oo.lcm(10)
+Infinity
sage: (-oo).lcm(10)
+Infinity
```

class sage.rings.infinity.**FiniteNumber**(*parent*, *x*)

Bases: sage.structure.element.RingElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.FiniteNumber(InfinityRing, 1)
A positive finite number
sage: sage.rings.infinity.FiniteNumber(InfinityRing, -1)
A negative finite number
sage: sage.rings.infinity.FiniteNumber(InfinityRing, 0)
Zero
```

sqrt()

EXAMPLES:

```
sage: InfinityRing(7).sqrt()
A positive finite number
sage: InfinityRing(0).sqrt()
Zero
sage: InfinityRing(-.001).sqrt()
Traceback (most recent call last):
...
SignError: cannot take square root of a negative number
```

class sage.rings.infinity.**InfinityRing_class**

Bases: sage.rings.infinity._uniq, sage.rings.ring.Ring

Initialize self.

TEST:

```
sage: sage.rings.infinity.InfinityRing_class() is sage.rings.infinity.InfinityRing_class() is In
True
```

fraction_field()

This isn't really a ring, let alone an integral domain.

TEST:

```
sage: InfinityRing.fraction_field()
Traceback (most recent call last):
...
TypeError: infinity 'ring' has no fraction field
```

gen(*n=0*)

The two generators are plus and minus infinity.

EXAMPLES:

```
sage: InfinityRing.gen(0)
+Infinity
sage: InfinityRing.gen(1)
-Infinity
sage: InfinityRing.gen(2)
Traceback (most recent call last):
...
IndexError: n must be 0 or 1
```

gens()
The two generators are plus and minus infinity.

EXAMPLES:

```
sage: InfinityRing.gens()
[+Infinity, -Infinity]
```

is_commutative()
The Infinity Ring is commutative

EXAMPLES:

```
sage: InfinityRing.is_commutative()
True
```

is_zero()
The Infinity Ring is not zero

EXAMPLES:

```
sage: InfinityRing.is_zero()
False
```

ngens()
The two generators are plus and minus infinity.

EXAMPLES:

```
sage: InfinityRing.ngens()
2
sage: len(InfinityRing.gens())
2
```

class sage.rings.infinity.**LessThanInfinity**(*parent=The Unsigned Infinity Ring*)
Bases: sage.rings.infinity._uniq, sage.structure.element.RingElement

Initialize self.

EXAMPLES:

```
sage: sage.rings.infinity.LessThanInfinity() is UnsignedInfinityRing(5)
True
```

class sage.rings.infinity.**MinusInfinity**
Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.MinusInfinityElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.MinusInfinity() is sage.rings.infinity.MinusInfinity() is -oo
True
```

sqrt()
EXAMPLES:

```
sage: (-oo).sqrt()
Traceback (most recent call last):
...
SignError: cannot take square root of negative infinity
```

class sage.rings.infinity.**PlusInfinity**
Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.PlusInfinityElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.PlusInfinity() is sage.rings.infinity.PlusInfinity() is oo
True
```

sqrt()
The square root of self.

The square root of infinity is infinity.

EXAMPLES:

```
sage: oo.sqrt()
+Infinity
```

exception sage.rings.infinity.**SignError**
Bases: exceptions.ArithmeticError

Sign error exception.

class sage.rings.infinity.**UnsignedInfinity**
Bases: sage.rings.infinity._uniq, sage.rings.infinity.AnInfinity, sage.structure.element.InfinityElement

Initialize self.

TESTS:

```
sage: sage.rings.infinity.UnsignedInfinity() is sage.rings.infinity.UnsignedInfinity() is unsign
True
```

class sage.rings.infinity.**UnsignedInfinityRing_class**
Bases: sage.rings.infinity._uniq, sage.rings.ring.Ring

Initialize self.

TESTS:

```
sage: sage.rings.infinity.UnsignedInfinityRing_class() is sage.rings.infinity.UnsignedInfinityRi
True
```

fraction_field()
The unsigned infinity ring isn't an integral domain.

EXAMPLES:

```
sage: UnsignedInfinityRing.fraction_field()
Traceback (most recent call last):
...
TypeError: infinity 'ring' has no fraction field
```

gen(*n=0*)
The "generator" of self is the infinity object.

EXAMPLES:

```
sage: UnsignedInfinityRing.gen()
Infinity
sage: UnsignedInfinityRing.gen(1)
```

```
Traceback (most recent call last):
...
IndexError: UnsignedInfinityRing only has one generator
```

gens()
The "generator" of `self` is the infinity object.

EXAMPLES:

```
sage: UnsignedInfinityRing.gens()
[Infinity]
```

less_than_infinity()
This is the element that represents a finite value.

EXAMPLES:

```
sage: UnsignedInfinityRing.less_than_infinity()
A number less than infinity
sage: UnsignedInfinityRing(5) is UnsignedInfinityRing.less_than_infinity()
True
```

ngens()
The unsigned infinity ring has one "generator."

EXAMPLES:

```
sage: UnsignedInfinityRing.ngens()
1
sage: len(UnsignedInfinityRing.gens())
1
```

sage.rings.infinity.**is_Infinite**(*x*)
This is a type check for infinity elements.

EXAMPLES:

```
sage: sage.rings.infinity.is_Infinite(oo)
True
sage: sage.rings.infinity.is_Infinite(-oo)
True
sage: sage.rings.infinity.is_Infinite(unsigned_infinity)
True
sage: sage.rings.infinity.is_Infinite(3)
False
sage: sage.rings.infinity.is_Infinite(RR(infinity))
False
sage: sage.rings.infinity.is_Infinite(ZZ)
False
```

sage.rings.infinity.**test_comparison**(*ring*)
Check comparison with infinity

INPUT:

- `ring` – a sub-ring of the real numbers

OUTPUT:

Various attempts are made to generate elements of `ring`. An assertion is triggered if one of these elements does not compare correctly with plus/minus infinity.

EXAMPLES:

```
sage: from sage.rings.infinity import test_comparison
sage: rings = [ZZ, QQ, RR, RealField(200), RDF, RLF, AA, RIF]
sage: for R in rings:
....:     print('testing {}'.format(R))
....:     test_comparison(R)
testing Integer Ring
testing Rational Field
testing Real Field with 53 bits of precision
testing Real Field with 200 bits of precision
testing Real Double Field
testing Real Lazy Field
testing Algebraic Real Field
testing Real Interval Field with 53 bits of precision
```

Comparison with number fields does not work:

```
sage: K.<sqrt3> = NumberField(x^2-3)
sage: (-oo < 1+sqrt3) and (1+sqrt3 < oo)      # known bug
False
```

The symbolic ring handles its own infinities, but answers `False` (meaning: cannot decide) already for some very elementary comparisons:

```
sage: test_comparison(SR)       # known bug
Traceback (most recent call last):
...
AssertionError: testing -1000.0 in Symbolic Ring: id = ...
```

`sage.rings.infinity.`**`test_signed_infinity`**(*pos_inf*)

Test consistency of infinity representations.

There are different possible representations of infinity in Sage. These are all consistent with the infinity ring, that is, compare with infinity in the expected way. See also trac ticket #14045

INPUT:

- `pos_inf` – a representation of positive infinity.

OUTPUT:

An assertion error is raised if the representation is not consistent with the infinity ring.

Check that trac ticket #14045 is fixed:

```
sage: InfinityRing(float('+inf'))
+Infinity
sage: InfinityRing(float('-inf'))
-Infinity
sage: oo > float('+inf')
False
sage: oo == float('+inf')
True
```

EXAMPLES:

```
sage: from sage.rings.infinity import test_signed_infinity
sage: for pos_inf in [oo, float('+inf'), RLF(oo), RIF(oo), SR(oo)]:
....:     test_signed_infinity(pos_inf)
```

6.3 Miscellaneous utilities

sage.rings.misc.**composite_field**(*K*, *L*)

Return a canonical field that contains both K and L, if possible. Otherwise, raise a ValueError.

INPUT: K – field L – field

OUTPUT: field

EXAMPLES: sage: composite_field(QQ,QQbar) Algebraic Field sage: composite_field(QQ,QQ[sqrt(2)]) Number Field in sqrt2 with defining polynomial x^2 - 2 sage: composite_field(QQ,QQ) Rational Field sage: composite_field(QQ,GF(7)) Traceback (most recent call last): ... ValueError: unable to find a common field

6.4 Asymptotic Expansions

6.4.1 (Asymptotic) Growth Groups

This module adds support for (asymptotic) growth groups. Such groups are equipped with a partial order: the elements can be seen as functions, and their behavior as the argument(s) get large (tend to ∞) is compared.

Besides an abstract base class `GenericGrowthGroup`, this module contains concrete realizations of growth groups. At the moment there is

- `MonomialGrowthGroup` (whose elements are powers of a fixed symbol).

More complex growth groups can be constructed via cartesian products (to be implemented).

These growth groups are used behind the scenes when performing calculations in an asymptotic ring (to be implemented).

AUTHORS:

- Benjamin Hackl (2015-01): initial version
- Daniel Krenn (2015-05-29): initial version and review
- Benjamin Hackl (2015-07): short representation strings

Warning: As this code is experimental, warnings are thrown when a growth group is created for the first time in a session (see `sage.misc.superseded.experimental`).
TESTS:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GenericGrowthGroup(ZZ); G
doctest:...: FutureWarning: This class/method/function is marked as
experimental. It, its functionality or its interface might change
without a formal deprecation.
See http://trac.sagemath.org/17601 for details.
Growth Group Generic(ZZ)
sage: G = agg.MonomialGrowthGroup(ZZ, 'x'); G
doctest:...: FutureWarning: This class/method/function is marked as
experimental. It, its functionality or its interface might change
without a formal deprecation.
See http://trac.sagemath.org/17601 for details.
Growth Group x^ZZ
```

class sage.rings.asymptotic.growth_group.**GenericGrowthElement**(*parent*, *raw_element*)

Bases: sage.structure.element.MultiplicativeGroupElement

A basic implementation of a generic growth element.

Growth elements form a group by multiplication, and (some of) the elements can be compared to each other, i.e., all elements form a poset.

INPUT:

- parent – a GenericGrowthGroup.
- raw_element – an element from the base of the parent.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GenericGrowthGroup(ZZ)
sage: g = agg.GenericGrowthElement(G, 42); g
GenericGrowthElement(42)
sage: g.parent()
Growth Group Generic(ZZ)
sage: G(raw_element=42) == g
True
```

class sage.rings.asymptotic.growth_group.**GenericGrowthGroup**(*base*, *category=None*)

Bases: sage.structure.parent.Parent, sage.structure.unique_representation.UniqueRepresenta

A basic implementation for growth groups.

INPUT:

- base – one of SageMath's parents, out of which the elements get their data (raw_element).
- category – (default: None) the category of the newly created growth group. It has to be a subcategory of Join of Category of groups and Category of posets. This is also the default category if None is specified.

Note: This class should be derived for concrete implementations.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GenericGrowthGroup(ZZ); G
Growth Group Generic(ZZ)
```

See also:

MonomialGrowthGroup

Element

alias of GenericGrowthElement

gens_monomial()

Return a monomial generator of this growth group, in case one exists.

INPUT:

Nothing.

OUTPUT:

An element of this growth group or None.

Note: A generator is called monomial generator if the variable of the underlying growth group is a valid identifier. For example, `x^ZZ` has `x` as a monomial generator, while `log(x)^ZZ` or `icecream(x)^ZZ` do not have monomial generators.

This method is only implemented for concrete growth group implementations.

TESTS:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.GenericGrowthGroup(ZZ).gens_monomial()
Traceback (most recent call last):
...
NotImplementedError: Only implemented for concrete growth group
implementations.
```

le(*left*, *right*)

Return if the growth of `left` is at most (less than or equal to) the growth of `right`.

INPUT:

- `left` – an element.
- `right` – an element.

OUTPUT:

A boolean.

Note: This function uses the coercion model to find a common parent for the two operands.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.MonomialGrowthGroup(ZZ, 'x')
sage: x = G.gen()
sage: G.le(x, x^2)
True
sage: G.le(x^2, x)
False
sage: G.le(x^0, 1)
True
```

one()

Return the neutral element of this growth group.

INPUT:

Nothing.

OUTPUT:

An element of this group.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: e1 = agg.MonomialGrowthGroup(ZZ, 'x').one(); e1
1
sage: e1.is_idempotent()
True
```

class sage.rings.asymptotic.growth_group.**GrowthGroupFactory**

Bases: sage.structure.factory.UniqueFactory

A factory creating asymptotic growth groups.

INPUT:

- `specification` – a string.

OUTPUT:

An asymptotic growth group.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.GrowthGroup('x^ZZ')
Growth Group x^ZZ
sage: agg.GrowthGroup('log(x)^QQ')
Growth Group log(x)^QQ
```

create_key_and_extra_args(*specification*, ***kwds*)

Given the arguments and keyword, create a key that uniquely determines this object.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.GrowthGroup.create_key_and_extra_args('x^ZZ')
(('x^ZZ',), {})
sage: agg.GrowthGroup.create_key_and_extra_args('asdf')
Traceback (most recent call last):
...
ValueError: 'asdf' is not a valid string describing a growth group.
```

create_object(*version*, *factors*, ***kwds*)

Create an object from the given arguments.

TESTS:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.GrowthGroup('as^df')
Traceback (most recent call last):
...
ValueError: 'as^df' is not a valid string describing a growth group.
sage: agg.GrowthGroup('x^y^z')
Traceback (most recent call last):
...
ValueError: Cannot decode x^y^z.
```

class sage.rings.asymptotic.growth_group.**MonomialGrowthElement**(*parent*, *raw_element*)

Bases: sage.rings.asymptotic.growth_group.GenericGrowthElement

An implementation of monomial growth elements.

INPUT:

- `parent` – a MonomialGrowthGroup.
- `raw_element` – an element from the base ring of the parent.

  This `raw_element` is the exponent of the created monomial growth element.

A monomial growth element represents a term of the type $\text{variable}^{\text{exponent}}$. The multiplication corresponds to the addition of the exponents.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x')
sage: e1 = P(1); e1
1
sage: e2 = P(raw_element=2); e2
x^2
sage: e1 == e2
False
sage: P.le(e1, e2)
True
sage: P.le(e1, P.gen()) and P.le(P.gen(), e2)
True
```

exponent
The exponent of this growth element.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x')
sage: P(x^42).exponent
42
```

class sage.rings.asymptotic.growth_group.**MonomialGrowthGroup**(*base*, *var*, *category*)
Bases: sage.rings.asymptotic.growth_group.GenericGrowthGroup

A growth group dealing with powers of a fixed object/symbol.

The elements MonomialGrowthElement of this group represent powers of a fixed base; the group law is the multiplication, which corresponds to the addition of the exponents of the monomials.

INPUT:

- base – one of SageMath's parents, out of which the elements get their data (raw_element).

  As monomials are represented by this group, the elements in base are the exponents of these monomials.

- var – an object.

  The string representation of var acts as a base of the monomials represented by this group.

- category – (default: None) the category of the newly created growth group. It has to be a subcategory of Join of Category of groups and Category of posets. This is also the default category if None is specified.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x'); P
Growth Group x^ZZ
sage: agg.MonomialGrowthGroup(ZZ, log(SR.var('y')))
Growth Group log(y)^ZZ
```

See also:

GenericGrowthGroup

Element
alias of MonomialGrowthElement

gen(*n=0*)
Return the n-th generator of this growth group.

INPUT:

- n – default: 0.

OUTPUT:

A MonomialGrowthElement.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x')
sage: P.gen()
x
```

gens()

Return a tuple of all generators of this monomial growth group.

INPUT:

Nothing.

OUTPUT:

A tuple whose entries are instances of MonomialGrowthElement.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x')
sage: P.gens()
(x,)
sage: agg.MonomialGrowthGroup(ZZ, 'log(x)').gens()
(log(x),)
```

gens_monomial()

Return a tuple containing monomial generators of this growth group.

INPUT:

Nothing.

OUTPUT:

A tuple containing elements of this growth group.

Note: A generator is called monomial generator if the variable of the underlying growth group is a valid identifier. For example, `x^ZZ` has `x` as a monomial generator, while `log(x)^ZZ` or `icecream(x)^ZZ` do not have monomial generators.

TESTS:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.MonomialGrowthGroup(ZZ, 'x').gens_monomial()
(x,)
sage: agg.MonomialGrowthGroup(QQ, 'log(x)').gens_monomial()
()
```

ngens()

Return the number of generators of this monomial growth group.

INPUT:

Nothing.

OUTPUT:

A Python integer.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: P = agg.MonomialGrowthGroup(ZZ, 'x')
sage: P.ngens()
1
sage: agg.MonomialGrowthGroup(ZZ, 'log(x)').ngens()
1
```

sage.rings.asymptotic.growth_group.**parent_to_repr_short**(*P*)

Helper method which generates a short(er) representation string out of a parent.

INPUT:

- P – a parent.

OUTPUT:

A string.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.parent_to_repr_short(ZZ)
'ZZ'
sage: agg.parent_to_repr_short(QQ)
'QQ'
sage: agg.parent_to_repr_short(SR)
'SR'
sage: agg.parent_to_repr_short(ZZ[x])
'(Univariate Polynomial Ring in x over Integer Ring)'
```

sage.rings.asymptotic.growth_group.**repr_short_to_parent**(*s*)

Helper method for the growth group factory, which converts a short representation string to a parent.

INPUT:

- s – a string, short representation of a parent.

OUTPUT:

A parent.

The possible short representations are shown in the examples below.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: agg.repr_short_to_parent('ZZ')
Integer Ring
sage: agg.repr_short_to_parent('QQ')
Rational Field
sage: agg.repr_short_to_parent('SR')
Symbolic Ring
```

TESTS:

```
sage: agg.repr_short_to_parent('abcdef')
Traceback (most recent call last):
...
ValueError: Cannot create a parent out of 'abcdef'.
```

6.4.2 Asymptotic Term Monoid

This module implements asymptotic term monoids. The elements of these monoids are used behind the scenes when performing calculations in an asymptotic ring (to be implemented).

The monoids build upon the (asymptotic) growth groups. While growth elements only model the growth of a function as it tends towards infinity (or tends towards another fixed point; see `growth_group` for more details), an asymptotic term additionally specifies its "type" and performs the actual arithmetic operations (multiplication and partial addition/absorption of terms).

Besides an abstract base term `GenericTerm`, this module implements the following types of terms:

- `OTerm` – O-terms at infinity, see Wikipedia article Big_O_notation.
- `TermWithCoefficient` – abstract base class for asymptotic terms with coefficients.
- `ExactTerm` – this class represents a growth element multiplied with some non-zero coefficient from a base ring.

Warning: As this code is experimental, a warning is thrown when a term monoid is created for the first time in a session (see `sage.misc.superseded.experimental`).
TESTS:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.MonomialGrowthGroup(ZZ, 'x')
doctest:...: FutureWarning: This class/method/function is marked as
experimental. It, its functionality or its interface might change
without a formal deprecation.
See http://trac.sagemath.org/17601 for details.
sage: T = atm.TermWithCoefficientMonoid(G, ZZ)
doctest:...: FutureWarning: This class/method/function is marked as
experimental. It, its functionality or its interface might change
without a formal deprecation.
See http://trac.sagemath.org/17601 for details.
```

Absorption of Asymptotic Terms

A characteristic property of asymptotic terms is that some terms are able to "absorb" other terms. This is realized with the method `absorb()`.

For instance, $O(x^2)$ is able to absorb $O(x)$ (with result $O(x^2)$). This is because the functions bounded by linear growth are bounded by quadratic growth as well. Another example would be that $3x^5$ is able to absorb $-2x^5$ (with result x^5), which simply corresponds to addition.

Essentially, absorption can be interpreted as the addition of "compatible" terms (partial addition).

We want to show step by step which terms can be absorbed by which other terms. We start by defining the necessary term monoids and some terms:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: OT = atm.OTermMonoid(growth_group=G)
```

```
sage: ET = atm.ExactTermMonoid(growth_group=G, base_ring=QQ)
sage: ot1 = OT(x); ot2 = OT(x^2)
sage: et1 = ET(x^2, 2)
```

- Because of the definition of O-terms (see Wikipedia article Big_O_notation), `OTerm` are able to absorb all other asymptotic terms with weaker or equal growth. In our implementation, this means that `OTerm` is able to absorb other `OTerm`, as well as `ExactTerm`, as long as the growth of the other term is less than or equal to the growth of this element:

  ```
  sage: ot1, ot2
  (O(x), O(x^2))
  sage: ot1.can_absorb(ot2), ot2.can_absorb(ot1)
  (False, True)
  sage: et1
  2*x^2
  sage: ot1.can_absorb(et1)
  False
  sage: ot2.can_absorb(et1)
  True
  ```

  The result of this absorption always is the dominant (absorbing) `OTerm`:

  ```
  sage: ot1.absorb(ot1)
  O(x)
  sage: ot2.absorb(ot1)
  O(x^2)
  sage: ot2.absorb(et1)
  O(x^2)
  ```

  These examples correspond to $O(x) + O(x) = O(x)$, $O(x^2) + O(x) = O(x^2)$, and $O(x^2) + 2x^2 = O(x^2)$.

- `ExactTerm` can only absorb another `ExactTerm` if the growth coincides with the growth of this element:

  ```
  sage: et1.can_absorb(ET(x^2, 5))
  True
  sage: any(et1.can_absorb(t) for t in [ot1, ot2])
  False
  ```

  As mentioned above, absorption directly corresponds to addition in this case:

  ```
  sage: et1.absorb(ET(x^2, 5))
  7*x^2
  ```

  When adding two exact terms, they might cancel out. For technical reasons, `None` is returned in this case:

  ```
  sage: ET(x^2, 5).can_absorb(ET(x^2, -5))
  True
  sage: ET(x^2, 5).absorb(ET(x^2, -5)) is None
  True
  ```

- The abstract base terms `GenericTerm` and `TermWithCoefficient` can neither absorb any other term, nor be absorbed by any other term.

If `absorb` is called on a term that cannot be absorbed, an ArithmeticError is raised:

```
sage: ot1.absorb(ot2)
Traceback (most recent call last):
...
ArithmeticError: O(x) cannot absorb O(x^2)
```

This would only work the other way around:

```
sage: ot2.absorb(ot1)
O(x^2)
```

Comparison

The comparison of asymptotic terms with $\leq$ is implemented as follows:

- When comparing `t1 <= t2`, the coercion framework first tries to find a common parent for both terms. If this fails, `False` is returned.
- In case the coerced terms do not have a coefficient in their common parent (e.g. `OTerm`), the growth of the two terms is compared.
- Otherwise, if the coerced terms have a coefficient (e.g. `ExactTerm`), we compare whether `t1` has a growth that is strictly weaker than the growth of `t2`. If so, we return `True`. If the terms have equal growth, then we return `True` if and only if the coefficients coincide as well.

  In all other cases, we return `False`.

Long story short: we consider terms with different coefficients that have equal growth to be incomparable.

Various

Todo

- Implementation of more term types (e.g. L terms, Ω terms, o terms, Θ terms).

AUTHORS:

- Benjamin Hackl (2015-01): initial version
- Benjamin Hackl, Daniel Krenn (2015-05): conception of the asymptotic ring
- Benjamin Hackl (2015-06): refactoring caused by refactoring growth groups
- Daniel Krenn (2015-07): extensive review and patches
- Benjamin Hackl (2015-07): cross-review; short notation

Classes and Methods

class `sage.rings.asymptotic.term_monoid.`**`ExactTerm`**(*parent*, *growth*, *coefficient*)

Bases: `sage.rings.asymptotic.term_monoid.TermWithCoefficient`

Class for asymptotic exact terms. These terms primarily consist of an asymptotic growth element as well as a coefficient specifying the growth of the asymptotic term.

INPUT:

- `parent` – the parent of the asymptotic term.
- `growth` – an asymptotic growth element from `parent.growth_group`.
- `coefficient` – an element from `parent.base_ring()`.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: ET = atm.ExactTermMonoid(G, QQ)
```

Asymptotic exact terms may be multiplied (with the usual rules applying):

```
sage: ET(x^2, 3) * ET(x, -1)
-3*x^3
sage: ET(x^0, 4) * ET(x^5, 2)
8*x^5
```

They may also be multiplied with O-terms:

```
sage: OT = atm.OTermMonoid(G)
sage: ET(x^2, 42) * OT(x)
O(x^3)
```

Absorption for asymptotic exact terms relates to addition:

```
sage: ET(x^2, 5).can_absorb(ET(x^5, 12))
False
sage: ET(x^2, 5).can_absorb(ET(x^2, 1))
True
sage: ET(x^2, 5).absorb(ET(x^2, 1))
6*x^2
```

Note that, as for technical reasons, 0 is not allowed as a coefficient for an asymptotic term with coefficient. Instead `None` is returned if two asymptotic exact terms cancel out each other during absorption:

```
sage: ET(x^2, 42).can_absorb(ET(x^2, -42))
True
sage: ET(x^2, 42).absorb(ET(x^2, -42)) is None
True
```

Exact terms can also be created by converting monomials with coefficient from the symbolic ring, or a suitable polynomial or power series ring:

```
sage: x = var('x'); x.parent()
Symbolic Ring
sage: ET(5*x^2)
5*x^2
sage: x = ZZ['x'].gen(); x.parent()
Univariate Polynomial Ring in x over Integer Ring
sage: ET(5*x^2)
5*x^2
sage: x = ZZ[['x']].gen(); x.parent()
Power Series Ring in x over Integer Ring
sage: ET(5*x^2)
5*x^2
```

can_absorb(*other*)

Check whether this exact term can absorb `other`.

INPUT:

- `other` – an asymptotic term.

OUTPUT:

A boolean.

Note: For ExactTerm, absorption corresponds to addition. This means that an exact term can absorb only other exact terms with the same growth.

See the *module description* for a detailed explanation of absorption.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: ET = atm.TermMonoid('exact', agg.GrowthGroup('x^ZZ'), ZZ)
sage: t1 = ET(x^21, 1); t2 = ET(x^21, 2); t3 = ET(x^42, 1)
sage: t1.can_absorb(t2)
True
sage: t2.can_absorb(t1)
True
sage: t1.can_absorb(t3) or t3.can_absorb(t1)
False
```

class sage.rings.asymptotic.term_monoid.**ExactTermMonoid**(*growth_group*, *base_ring*, *category=None*)

Bases: sage.rings.asymptotic.term_monoid.TermWithCoefficientMonoid

Parent for asymptotic exact terms, implemented in ExactTerm.

INPUT:

- growth_group – a growth group.
- category – The category of the parent can be specified in order to broaden the base structure. It has to be a subcategory of Join of Category of monoids and Category of posets. This is also the default category if None is specified.
- base_ring – the ring which contains the coefficients of the elements.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G_ZZ = agg.GrowthGroup('x^ZZ'); x_ZZ = G_ZZ.gen()
sage: G_QQ = agg.GrowthGroup('x^QQ'); x_QQ = G_QQ.gen()
sage: ET_ZZ = atm.ExactTermMonoid(G_ZZ, ZZ); ET_ZZ
Exact Term Monoid x^ZZ with coefficients from Integer Ring
sage: ET_QQ = atm.ExactTermMonoid(G_QQ, QQ); ET_QQ
Exact Term Monoid x^QQ with coefficients from Rational Field
sage: ET_QQ.coerce_map_from(ET_ZZ)
Conversion map:
  From: Exact Term Monoid x^ZZ with coefficients from Integer Ring
  To:   Exact Term Monoid x^QQ with coefficients from Rational Field
```

Exact term monoids can also be created using the term factory:

```
sage: atm.TermMonoid('exact', G_ZZ, ZZ) is ET_ZZ
True
sage: atm.TermMonoid('exact', agg.GrowthGroup('x^ZZ'), QQ)
Exact Term Monoid x^ZZ with coefficients from Rational Field
```

Element

alias of ExactTerm

class sage.rings.asymptotic.term_monoid.**GenericTerm**(*parent*, *growth*)

Bases: sage.structure.element.MonoidElement

Base class for asymptotic terms. Mainly the structure and several properties of asymptotic terms are handled here.

INPUT:

- `parent` – the parent of the asymptotic term.
- `growth` – an asymptotic growth element.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: T = atm.GenericTermMonoid(G)
sage: t1 = T(x); t2 = T(x^2); (t1, t2)
(Generic Term with growth x, Generic Term with growth x^2)
sage: t1 * t2
Generic Term with growth x^3
sage: t1.can_absorb(t2)
False
sage: t1.absorb(t2)
Traceback (most recent call last):
...
ArithmeticError: Generic Term with growth x cannot absorb Generic Term with growth x^2
sage: t1.can_absorb(t1)
False
```

absorb(*other*, *check=True*)

Absorb the asymptotic term `other` and return the resulting asymptotic term.

INPUT:

- `other` – an asymptotic term.
- `check` – a boolean. If `check` is `True` (default), then `can_absorb` is called before absorption.

OUTPUT:

An asymptotic term or `None` if a cancellation occurs. If no absorption can be performed, an ArithmeticError is raised.

Note: Setting `check` to `False` is meant to be used in cases where the respective comparison is done externally (in order to avoid duplicate checking).

For a more detailed explanation of the *absorption* of asymptotic terms see the *module description*.

EXAMPLES:

We want to demonstrate in which cases an asymptotic term is able to absorb another term, as well as explain the output of this operation. We start by defining some parents and elements:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G_QQ = agg.GrowthGroup('x^QQ'); x = G_QQ.gen()
sage: OT = atm.OTermMonoid(G_QQ)
sage: ET = atm.ExactTermMonoid(growth_group=G_QQ, base_ring=QQ)
sage: ot1 = OT(x); ot2 = OT(x^2)
sage: et1 = ET(x, 100); et2 = ET(x^2, 2)
sage: et3 = ET(x^2, 1); et4 = ET(x^2, -2)
```

O-Terms are able to absorb other O-terms and exact terms with weaker or equal growth.

```
sage: ot1.absorb(ot1)
O(x)
sage: ot1.absorb(et1)
O(x)
sage: ot1.absorb(et1) is ot1
True
```

`ExactTerm` is able to absorb another `ExactTerm` if the terms have the same growth. In this case, *absorption* is nothing else than an addition of the respective coefficients:

```
sage: et2.absorb(et3)
3*x^2
sage: et3.absorb(et2)
3*x^2
sage: et3.absorb(et4)
-x^2
```

Note that, for technical reasons, the coefficient 0 is not allowed, and thus `None` is returned if two exact terms cancel each other out:

```
sage: et2.absorb(et4)
sage: et4.absorb(et2) is None
True
```

TESTS:

When disabling the `check` flag, absorb might produce wrong results:

```
sage: et1.absorb(ot2, check=False)
O(x)
```

can_absorb(*other*)

Check whether this asymptotic term is able to absorb the asymptotic term `other`.

INPUT:

- `other` – an asymptotic term.

OUTPUT:

A boolean.

Note: A `GenericTerm` cannot absorb any other term.

See the *module description* for a detailed explanation of absorption.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GenericGrowthGroup(ZZ)
sage: T = atm.GenericTermMonoid(G)
sage: g1 = G(raw_element=21); g2 = G(raw_element=42)
sage: t1 = T(g1); t2 = T(g2)
sage: t1.can_absorb(t2)  # indirect doctest
False
sage: t2.can_absorb(t1)  # indirect doctest
False
```

class sage.rings.asymptotic.term_monoid.**GenericTermMonoid**(*growth_group*, *category=None*)

Bases: sage.structure.parent.Parent, sage.structure.unique_representation.UniqueRepresenta

Parent for generic asymptotic terms.

INPUT:

- growth_group – a growth group (i.e. an instance of GenericGrowthGroup).
- category – The category of the parent can be specified in order to broaden the base structure. It has to be a subcategory of Join of Category of Monoids and Category of posets. This is also the default category if None is specified.

In this class the base structure for asymptotic term monoids will be handled. These monoids are the parents of asymptotic terms (for example, see GenericTerm or OTerm). Basically, asymptotic terms consist of a growth (which is an asymptotic growth group element, for example MonomialGrowthElement); additional structure and properties are added by the classes inherited from GenericTermMonoid.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G_x = agg.GrowthGroup('x^ZZ'); x = G_x.gen()
sage: G_y = agg.GrowthGroup('y^QQ'); y = G_y.gen()
sage: T_x_ZZ = atm.GenericTermMonoid(G_x); T_y_QQ = atm.GenericTermMonoid(G_y)
sage: T_x_ZZ
Generic Term Monoid x^ZZ
sage: T_y_QQ
Generic Term Monoid y^QQ
```

Element
alias of GenericTerm

growth_group
The growth group underlying this term monoid.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: atm.ExactTermMonoid(G, ZZ).growth_group
Growth Group x^ZZ
```

le(*left*, *right*)
Return whether the term left is at most (less than or equal to) the term right.

INPUT:

- left – an element.
- right – an element.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: T = atm.GenericTermMonoid(growth_group=G)
sage: t1 = T(x); t2 = T(x^2)
```

```
sage: T.le(t1, t2)
True
```

class sage.rings.asymptotic.term_monoid.**OTerm**(*parent*, *growth*)

Bases: sage.rings.asymptotic.term_monoid.GenericTerm

Class for an asymptotic term representing an O-term with specified growth. For the mathematical properties of O-terms see Wikipedia article Big_O_Notation.

O-terms can *absorb* terms of weaker or equal growth.

INPUT:

- `parent` – the parent of the asymptotic term.
- `growth` – a growth element.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: OT = atm.OTermMonoid(G)
sage: t1 = OT(x^-7); t2 = OT(x^5); t3 = OT(x^42)
sage: t1, t2, t3
(O(x^(-7)), O(x^5), O(x^42))
sage: t1.can_absorb(t2)
False
sage: t2.can_absorb(t1)
True
sage: t2.absorb(t1)
O(x^5)
sage: t1 <= t2 and t2 <= t3
True
sage: t3 <= t1
False
```

The conversion of growth elements also works for the creation of O-terms:

```
sage: x = SR('x'); x.parent()
Symbolic Ring
sage: OT(x^17)
O(x^17)
```

can_absorb(*other*)

Check whether this O-term can absorb `other`.

INPUT:

- `other` – an asymptotic term.

OUTPUT:

A boolean.

Note: An `OTerm` can absorb any other asymptotic term with weaker or equal growth.

See the *module description* for a detailed explanation of absorption.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: OT = atm.TermMonoid('O', agg.GrowthGroup('x^ZZ'))
sage: t1 = OT(x^21); t2 = OT(x^42)
sage: t1.can_absorb(t2)
False
sage: t2.can_absorb(t1)
True
```

class sage.rings.asymptotic.term_monoid.**OTermMonoid**(*growth_group*, *category=None*)
Bases: sage.rings.asymptotic.term_monoid.GenericTermMonoid

Parent for asymptotic big O-terms.

INPUT:

- growth_group – a growth group.
- category – The category of the parent can be specified in order to broaden the base structure. It has to be a subcategory of Join of Category of monoids and Category of posets. This is also the default category if None is specified.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G_x_ZZ = agg.GrowthGroup('x^ZZ')
sage: G_y_QQ = agg.GrowthGroup('y^QQ')
sage: OT_x_ZZ = atm.OTermMonoid(G_x_ZZ); OT_x_ZZ
Asymptotic O-Term Monoid x^ZZ
sage: OT_y_QQ = atm.OTermMonoid(G_y_QQ); OT_y_QQ
Asymptotic O-Term Monoid y^QQ
```

O-term monoids can also be created by using the term factory:

```
sage: atm.TermMonoid('O', G_x_ZZ) is OT_x_ZZ
True
sage: atm.TermMonoid('O', agg.GrowthGroup('x^QQ'))
Asymptotic O-Term Monoid x^QQ
```

Element
alias of OTerm

class sage.rings.asymptotic.term_monoid.**TermMonoidFactory**
Bases: sage.structure.factory.UniqueFactory

Factory for asymptotic term monoids. It can generate the following term monoids:

- OTermMonoid,
- ExactTermMonoid.

INPUT:

- term – the kind of term that shall be created. Either 'exact' or 'O' (capital letter O).
- growth_group – a growth group.
- base_ring – the base ring for coefficients.

OUTPUT:

An asymptotic term monoid.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: OT = atm.TermMonoid('O', G); OT
Asymptotic O-Term Monoid x^ZZ
sage: ET = atm.TermMonoid('exact', G, ZZ); ET
Exact Term Monoid x^ZZ with coefficients from Integer Ring
```

create_key_and_extra_args(*term*, *growth_group*, *base_ring=None*, ***kwds*)

Given the arguments and keyword, create a key that uniquely determines this object.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: atm.TermMonoid.create_key_and_extra_args('O', G)
(('O', Growth Group x^ZZ, None), {})
sage: atm.TermMonoid.create_key_and_extra_args('exact', G, ZZ)
(('exact', Growth Group x^ZZ, Integer Ring), {})
sage: atm.TermMonoid.create_key_and_extra_args('exact', G)
Traceback (most recent call last):
...
ValueError: A base ring has to be specified
```

TESTS:

```
sage: atm.TermMonoid.create_key_and_extra_args('icecream', G)
Traceback (most recent call last):
...
ValueError: icecream has to be either 'exact' or 'O'
sage: atm.TermMonoid.create_key_and_extra_args('O', ZZ)
Traceback (most recent call last):
...
ValueError: Integer Ring has to be an asymptotic growth group
```

create_object(*version*, *key*, ***kwds*)

Create a object from the given arguments.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: atm.TermMonoid('O', G)  # indirect doctest
Asymptotic O-Term Monoid x^ZZ
sage: atm.TermMonoid('exact', G, ZZ)  # indirect doctest
Exact Term Monoid x^ZZ with coefficients from Integer Ring
```

class sage.rings.asymptotic.term_monoid.**TermWithCoefficient**(*parent*, *growth*, *coefficient*)

Bases: sage.rings.asymptotic.term_monoid.GenericTerm

Base class for asymptotic terms possessing a coefficient. For example, ExactTerm directly inherits from this class.

INPUT:

- parent – the parent of the asymptotic term.

- growth – an asymptotic growth element of the parent's growth group.
- coefficient – an element of the parent's base ring.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G = agg.GrowthGroup('x^ZZ'); x = G.gen()
sage: CT_ZZ = atm.TermWithCoefficientMonoid(G, ZZ)
sage: CT_QQ = atm.TermWithCoefficientMonoid(G, QQ)
sage: CT_ZZ(x^2, 5)
Asymptotic Term with coefficient 5 and growth x^2
sage: CT_QQ(x^3, 3/8)
Asymptotic Term with coefficient 3/8 and growth x^3
```

class sage.rings.asymptotic.term_monoid.**TermWithCoefficientMonoid**(*growth_group*, *base_ring*, *category=None*)

Bases: sage.rings.asymptotic.term_monoid.GenericTermMonoid

This class implements the base structure for parents of asymptotic terms possessing a coefficient from some coefficient ring. In particular, this is also the parent for TermWithCoefficient.

INPUT:

- growth_group – a growth group.
- category – The category of the parent can be specified in order to broaden the base structure. It has to be a subcategory of Join of Category of monoids and Category of posets. This is also the default category if None is specified.
- base_ring – the ring which contains the coefficients of the elements.

EXAMPLES:

```
sage: import sage.rings.asymptotic.term_monoid as atm
sage: import sage.rings.asymptotic.growth_group as agg
sage: G_ZZ = agg.GrowthGroup('x^ZZ'); x_ZZ = G_ZZ.gen()
sage: G_QQ = agg.GrowthGroup('x^QQ'); x_QQ = G_QQ.gen()
sage: TC_ZZ = atm.TermWithCoefficientMonoid(G_ZZ, QQ); TC_ZZ
Term Monoid x^ZZ with coefficients from Rational Field
sage: TC_QQ = atm.TermWithCoefficientMonoid(G_QQ, QQ); TC_QQ
Term Monoid x^QQ with coefficients from Rational Field
sage: TC_ZZ == TC_QQ or TC_ZZ is TC_QQ
False
sage: TC_QQ.coerce_map_from(TC_ZZ)
Conversion map:
  From: Term Monoid x^ZZ with coefficients from Rational Field
  To:   Term Monoid x^QQ with coefficients from Rational Field
```

Element

alias of TermWithCoefficient

base_ring()

The base ring of this term monoid, i.e. the ring where the coefficients are from.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
```

```
sage: atm.ExactTermMonoid(G, ZZ).base_ring()
Integer Ring
```

sage.rings.asymptotic.term_monoid.**absorption**(*left*, *right*)

Let one of the two passed terms absorb the other.

Helper function used by `AsymptoticExpression`.

Note: If neither of the terms can absorb the other, an ArithmeticError is raised.

See the *module description* for a detailed explanation of absorption.

INPUT:

- `left` – an asymptotic term.
- `right` – an asymptotic term.

OUTPUT:

An asymptotic term or `None`.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: T = atm.TermMonoid('O', G)
sage: atm.absorption(T(x^2), T(x^3))
O(x^3)
sage: atm.absorption(T(x^3), T(x^2))
O(x^3)

sage: T = atm.TermMonoid('exact', G, ZZ)
sage: atm.absorption(T(x^2), T(x^3))
Traceback (most recent call last):
...
ArithmeticError: Absorption between x^2 and x^3 is not possible.
```

sage.rings.asymptotic.term_monoid.**can_absorb**(*left*, *right*)

Return whether one of the two input terms is able to absorb the other.

Helper function used by `AsymptoticExpression`.

INPUT:

- `left` – an asymptotic term.
- `right` – an asymptotic term.

OUTPUT:

A boolean.

Note: See the *module description* for a detailed explanation of absorption.

EXAMPLES:

```
sage: import sage.rings.asymptotic.growth_group as agg
sage: import sage.rings.asymptotic.term_monoid as atm
sage: G = agg.GrowthGroup('x^ZZ')
sage: T = atm.TermMonoid('O', G)
```

```
sage: atm.can_absorb(T(x^2), T(x^3))
True
sage: atm.can_absorb(T(x^3), T(x^2))
True
```

6.5 Indices and Tables

- Index
- Module Index
- Search Page

CHAPTER

SEVEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

[AtiMac] Atiyah and Macdonald, “Introduction to commutative algebra”, Addison-Wesley, 1969.

[Ho72] E. Horowitz, “Algorithms for Rational Function Arithmetic Operations”, Annual ACM Symposium on Theory of Computing, Proceedings of the Fourth Annual ACM Symposium on Theory of Computing, pp. 108–118, 1972

r

sage.rings.asymptotic.growth_group, 93
sage.rings.asymptotic.term_monoid, 100
sage.rings.big_oh, 83
sage.rings.commutative_algebra, 23
sage.rings.commutative_ring, 23
sage.rings.dedekind_domain, 23
sage.rings.euclidean_domain, 23
sage.rings.fraction_field, 73
sage.rings.fraction_field_element, 77
sage.rings.homset, 55
sage.rings.ideal, 25
sage.rings.ideal_monoid, 40
sage.rings.infinity, 84
sage.rings.integral_domain, 23
sage.rings.misc, 93
sage.rings.morphism, 43
sage.rings.noncommutative_ideals, 40
sage.rings.principal_ideal_domain, 24
sage.rings.quotient_ring, 57
sage.rings.quotient_ring_element, 68
sage.rings.ring, 1

A

absolute_norm() (sage.rings.ideal.Ideal_generic method), 28
absorb() (sage.rings.asymptotic.term_monoid.GenericTerm method), 105
absorption() (in module sage.rings.asymptotic.term_monoid), 112
Algebra (class in sage.rings.ring), 2
algebraic_closure() (sage.rings.ring.Field method), 7
ambient() (sage.rings.quotient_ring.QuotientRing_nc method), 61
AnInfinity (class in sage.rings.infinity), 87
apply_morphism() (sage.rings.ideal.Ideal_generic method), 28
associated_primes() (sage.rings.ideal.Ideal_generic method), 29

B

base_extend() (sage.rings.ring.Ring method), 13
base_ring() (sage.rings.asymptotic.term_monoid.TermWithCoefficientMonoid method), 111
base_ring() (sage.rings.fraction_field.FractionField_generic method), 75
base_ring() (sage.rings.ideal.Ideal_generic method), 29

C

can_absorb() (in module sage.rings.asymptotic.term_monoid), 112
can_absorb() (sage.rings.asymptotic.term_monoid.ExactTerm method), 103
can_absorb() (sage.rings.asymptotic.term_monoid.GenericTerm method), 106
can_absorb() (sage.rings.asymptotic.term_monoid.OTerm method), 108
cardinality() (sage.rings.ring.Ring method), 13
category() (sage.rings.ideal.Ideal_generic method), 30
category() (sage.rings.ring.Ring method), 13
characteristic() (sage.rings.fraction_field.FractionField_generic method), 75
characteristic() (sage.rings.quotient_ring.QuotientRing_nc method), 62
characteristic() (sage.rings.ring.Algebra method), 2
class_group() (sage.rings.ring.PrincipalIdealDomain method), 10
class_number() (sage.rings.fraction_field.FractionField_1poly_field method), 74
CommutativeAlgebra (class in sage.rings.ring), 2
CommutativeRing (class in sage.rings.ring), 3
composite_field() (in module sage.rings.misc), 93
construction() (sage.rings.fraction_field.FractionField_generic method), 75
construction() (sage.rings.quotient_ring.QuotientRing_nc method), 62
content() (sage.rings.ring.PrincipalIdealDomain method), 10
cover() (sage.rings.quotient_ring.QuotientRing_nc method), 62

cover_ring() (sage.rings.quotient_ring.QuotientRing_nc method), 63
create_key_and_extra_args() (sage.rings.asymptotic.growth_group.GrowthGroupFactory method), 96
create_key_and_extra_args() (sage.rings.asymptotic.term_monoid.TermMonoidFactory method), 110
create_object() (sage.rings.asymptotic.growth_group.GrowthGroupFactory method), 96
create_object() (sage.rings.asymptotic.term_monoid.TermMonoidFactory method), 110
Cyclic() (in module sage.rings.ideal), 25

D

DedekindDomain (class in sage.rings.ring), 5
defining_ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 63
denominator() (sage.rings.fraction_field_element.FractionFieldElement method), 78
divides() (sage.rings.ideal.Ideal_principal method), 38
divides() (sage.rings.ring.Field method), 7

E

Element (sage.rings.asymptotic.growth_group.GenericGrowthGroup attribute), 94
Element (sage.rings.asymptotic.growth_group.MonomialGrowthGroup attribute), 97
Element (sage.rings.asymptotic.term_monoid.ExactTermMonoid attribute), 104
Element (sage.rings.asymptotic.term_monoid.GenericTermMonoid attribute), 107
Element (sage.rings.asymptotic.term_monoid.OTermMonoid attribute), 109
Element (sage.rings.asymptotic.term_monoid.TermWithCoefficientMonoid attribute), 111
Element (sage.rings.ideal_monoid.IdealMonoid_c attribute), 40
Element (sage.rings.quotient_ring.QuotientRing_nc attribute), 61
embedded_primes() (sage.rings.ideal.Ideal_generic method), 30
epsilon() (sage.rings.ring.Ring method), 14
EuclideanDomain (class in sage.rings.ring), 6
ExactTerm (class in sage.rings.asymptotic.term_monoid), 102
ExactTermMonoid (class in sage.rings.asymptotic.term_monoid), 104
exponent (sage.rings.asymptotic.growth_group.MonomialGrowthElement attribute), 97
extension() (sage.rings.ring.CommutativeRing method), 3

F

Field (class in sage.rings.ring), 7
FieldIdeal() (in module sage.rings.ideal), 25
FiniteNumber (class in sage.rings.infinity), 88
fraction_field() (sage.rings.infinity.InfinityRing_class method), 88
fraction_field() (sage.rings.infinity.UnsignedInfinityRing_class method), 90
fraction_field() (sage.rings.ring.CommutativeRing method), 3
fraction_field() (sage.rings.ring.Field method), 7
FractionField() (in module sage.rings.fraction_field), 73
FractionField_1poly_field (class in sage.rings.fraction_field), 74
FractionField_generic (class in sage.rings.fraction_field), 75
FractionFieldElement (class in sage.rings.fraction_field_element), 77
FractionFieldElement_1poly_field (class in sage.rings.fraction_field_element), 80
frobenius_endomorphism() (sage.rings.ring.CommutativeRing method), 3
FrobeniusEndomorphism_generic (class in sage.rings.morphism), 48

G

gcd() (sage.rings.ideal.Ideal_pid method), 35
gcd() (sage.rings.ring.PrincipalIdealDomain method), 11

gen() (sage.rings.asymptotic.growth_group.MonomialGrowthGroup method), 97
gen() (sage.rings.fraction_field.FractionField_generic method), 75
gen() (sage.rings.ideal.Ideal_generic method), 30
gen() (sage.rings.ideal.Ideal_principal method), 38
gen() (sage.rings.infinity.InfinityRing_class method), 88
gen() (sage.rings.infinity.UnsignedInfinityRing_class method), 90
gen() (sage.rings.quotient_ring.QuotientRing_nc method), 63
GenericGrowthElement (class in sage.rings.asymptotic.growth_group), 93
GenericGrowthGroup (class in sage.rings.asymptotic.growth_group), 94
GenericTerm (class in sage.rings.asymptotic.term_monoid), 104
GenericTermMonoid (class in sage.rings.asymptotic.term_monoid), 106
gens() (sage.rings.asymptotic.growth_group.MonomialGrowthGroup method), 98
gens() (sage.rings.ideal.Ideal_generic method), 30
gens() (sage.rings.infinity.InfinityRing_class method), 88
gens() (sage.rings.infinity.UnsignedInfinityRing_class method), 91
gens_monomial() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 94
gens_monomial() (sage.rings.asymptotic.growth_group.MonomialGrowthGroup method), 98
gens_reduced() (sage.rings.ideal.Ideal_generic method), 31
growth_group (sage.rings.asymptotic.term_monoid.GenericTermMonoid attribute), 107
GrowthGroupFactory (class in sage.rings.asymptotic.growth_group), 95

H

has_coerce_map_from() (sage.rings.homset.RingHomset_generic method), 55
has_standard_involution() (sage.rings.ring.Algebra method), 2

I

Ideal() (in module sage.rings.ideal), 26
ideal() (sage.rings.quotient_ring.QuotientRing_nc method), 64
ideal() (sage.rings.ring.Field method), 7
ideal() (sage.rings.ring.Ring method), 14
Ideal_fractional (class in sage.rings.ideal), 28
Ideal_generic (class in sage.rings.ideal), 28
ideal_monoid() (sage.rings.ring.CommutativeRing method), 4
ideal_monoid() (sage.rings.ring.Ring method), 15
Ideal_nc (class in sage.rings.noncommutative_ideals), 41
Ideal_pid (class in sage.rings.ideal), 35
Ideal_principal (class in sage.rings.ideal), 37
IdealMonoid() (in module sage.rings.ideal_monoid), 40
IdealMonoid_c (class in sage.rings.ideal_monoid), 40
IdealMonoid_nc (class in sage.rings.noncommutative_ideals), 41
im_gens() (sage.rings.morphism.RingHomomorphism_im_gens method), 54
InfinityRing_class (class in sage.rings.infinity), 88
integral_closure() (sage.rings.ring.DedekindDomain method), 5
integral_closure() (sage.rings.ring.Field method), 8
IntegralDomain (class in sage.rings.ring), 8
inverse_image() (sage.rings.morphism.RingHomomorphism method), 49
is_commutative() (sage.rings.infinity.InfinityRing_class method), 89
is_commutative() (sage.rings.quotient_ring.QuotientRing_nc method), 64
is_commutative() (sage.rings.ring.CommutativeAlgebra method), 2
is_commutative() (sage.rings.ring.CommutativeRing method), 4

is_commutative() (sage.rings.ring.Ring method), 15
is_CommutativeAlgebra() (in module sage.rings.commutative_algebra), 23
is_CommutativeRing() (in module sage.rings.commutative_ring), 23
is_DedekindDomain() (in module sage.rings.dedekind_domain), 23
is_EuclideanDomain() (in module sage.rings.euclidean_domain), 23
is_exact() (sage.rings.fraction_field.FractionField_generic method), 76
is_exact() (sage.rings.ring.Ring method), 15
is_field() (sage.rings.fraction_field.FractionField_generic method), 76
is_field() (sage.rings.quotient_ring.QuotientRing_nc method), 65
is_field() (sage.rings.ring.Field method), 8
is_field() (sage.rings.ring.IntegralDomain method), 9
is_field() (sage.rings.ring.Ring method), 16
is_finite() (sage.rings.fraction_field.FractionField_generic method), 76
is_finite() (sage.rings.ring.Ring method), 16
is_FractionField() (in module sage.rings.fraction_field), 77
is_FractionFieldElement() (in module sage.rings.fraction_field_element), 80
is_Ideal() (in module sage.rings.ideal), 39
is_identity() (sage.rings.morphism.RingHomomorphism_from_base method), 52
is_Infinite() (in module sage.rings.infinity), 91
is_injective() (sage.rings.morphism.RingHomomorphism method), 49
is_integral() (sage.rings.fraction_field_element.FractionFieldElement_1poly_field method), 80
is_integral_domain() (sage.rings.quotient_ring.QuotientRing_nc method), 65
is_integral_domain() (sage.rings.ring.IntegralDomain method), 9
is_integral_domain() (sage.rings.ring.Ring method), 17
is_IntegralDomain() (in module sage.rings.integral_domain), 23
is_integrally_closed() (sage.rings.ring.DedekindDomain method), 5
is_integrally_closed() (sage.rings.ring.Field method), 8
is_integrally_closed() (sage.rings.ring.IntegralDomain method), 9
is_maximal() (sage.rings.ideal.Ideal_generic method), 31
is_maximal() (sage.rings.ideal.Ideal_pid method), 36
is_noetherian() (sage.rings.quotient_ring.QuotientRing_nc method), 65
is_noetherian() (sage.rings.ring.DedekindDomain method), 6
is_noetherian() (sage.rings.ring.Field method), 8
is_noetherian() (sage.rings.ring.NoetherianRing method), 10
is_noetherian() (sage.rings.ring.PrincipalIdealDomain method), 12
is_noetherian() (sage.rings.ring.Ring method), 18
is_one() (sage.rings.fraction_field_element.FractionFieldElement method), 78
is_primary() (sage.rings.ideal.Ideal_generic method), 31
is_prime() (sage.rings.ideal.Ideal_generic method), 32
is_prime() (sage.rings.ideal.Ideal_pid method), 36
is_prime_field() (sage.rings.ring.Ring method), 18
is_principal() (sage.rings.ideal.Ideal_generic method), 33
is_principal() (sage.rings.ideal.Ideal_principal method), 38
is_PrincipalIdealDomain() (in module sage.rings.principal_ideal_domain), 24
is_QuotientRing() (in module sage.rings.quotient_ring), 68
is_Ring() (in module sage.rings.ring), 22
is_ring() (sage.rings.ring.Ring method), 18
is_RingHomomorphism() (in module sage.rings.morphism), 54
is_RingHomset() (in module sage.rings.homset), 56
is_square() (sage.rings.fraction_field_element.FractionFieldElement method), 78

is_subring() (sage.rings.ring.Ring method), 18
is_trivial() (sage.rings.ideal.Ideal_generic method), 33
is_unit() (sage.rings.quotient_ring_element.QuotientRingElement method), 69
is_zero() (sage.rings.fraction_field_element.FractionFieldElement method), 79
is_zero() (sage.rings.infinity.InfinityRing_class method), 89
is_zero() (sage.rings.morphism.RingHomomorphism method), 49

K

Katsura() (in module sage.rings.ideal), 39
kernel() (sage.rings.morphism.RingHomomorphism_cover method), 51
krull_dimension() (sage.rings.ring.CommutativeRing method), 4
krull_dimension() (sage.rings.ring.DedekindDomain method), 6
krull_dimension() (sage.rings.ring.Field method), 8

L

lc() (sage.rings.quotient_ring_element.QuotientRingElement method), 69
lcm() (sage.rings.infinity.AnInfinity method), 87
le() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 95
le() (sage.rings.asymptotic.term_monoid.GenericTermMonoid method), 107
less_than_infinity() (sage.rings.infinity.UnsignedInfinityRing_class method), 91
LessThanInfinity (class in sage.rings.infinity), 89
lift() (sage.rings.morphism.RingHomomorphism method), 50
lift() (sage.rings.quotient_ring.QuotientRing_nc method), 66
lift() (sage.rings.quotient_ring_element.QuotientRingElement method), 70
lifting_map() (sage.rings.quotient_ring.QuotientRing_nc method), 66
lm() (sage.rings.quotient_ring_element.QuotientRingElement method), 70
lt() (sage.rings.quotient_ring_element.QuotientRingElement method), 70

M

make_element() (in module sage.rings.fraction_field_element), 80
make_element_old() (in module sage.rings.fraction_field_element), 80
maximal_order() (sage.rings.fraction_field.FractionField_1poly_field method), 74
minimal_associated_primes() (sage.rings.ideal.Ideal_generic method), 33
MinusInfinity (class in sage.rings.infinity), 89
MonomialGrowthElement (class in sage.rings.asymptotic.growth_group), 96
MonomialGrowthGroup (class in sage.rings.asymptotic.growth_group), 97
monomials() (sage.rings.quotient_ring_element.QuotientRingElement method), 71
morphism_from_cover() (sage.rings.morphism.RingHomomorphism_from_quotient method), 53

N

natural_map() (sage.rings.homset.RingHomset_generic method), 55
ngens() (sage.rings.asymptotic.growth_group.MonomialGrowthGroup method), 98
ngens() (sage.rings.fraction_field.FractionField_generic method), 76
ngens() (sage.rings.ideal.Ideal_generic method), 34
ngens() (sage.rings.infinity.InfinityRing_class method), 89
ngens() (sage.rings.infinity.UnsignedInfinityRing_class method), 91
ngens() (sage.rings.quotient_ring.QuotientRing_nc method), 67
NoetherianRing (class in sage.rings.ring), 10
norm() (sage.rings.ideal.Ideal_generic method), 34
numerator() (sage.rings.fraction_field_element.FractionFieldElement method), 79

O

O() (in module sage.rings.big_oh), 83
one() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 95
one() (sage.rings.ring.Ring method), 18
order() (sage.rings.ring.Ring method), 19
OTerm (class in sage.rings.asymptotic.term_monoid), 108
OTermMonoid (class in sage.rings.asymptotic.term_monoid), 109

P

parameter() (sage.rings.ring.EuclideanDomain method), 6
parent_to_repr_short() (in module sage.rings.asymptotic.growth_group), 99
PlusInfinity (class in sage.rings.infinity), 89
power() (sage.rings.morphism.FrobeniusEndomorphism_generic method), 49
primary_decomposition() (sage.rings.ideal.Ideal_generic method), 34
prime_subfield() (sage.rings.ring.Field method), 8
principal_ideal() (sage.rings.ring.Ring method), 19
PrincipalIdealDomain (class in sage.rings.ring), 10
pushforward() (sage.rings.morphism.RingHomomorphism method), 50

Q

quo() (sage.rings.ring.Ring method), 19
quotient() (sage.rings.ring.Ring method), 19
quotient_ring() (sage.rings.ring.Ring method), 20
QuotientRing() (in module sage.rings.quotient_ring), 58
QuotientRing_generic (class in sage.rings.quotient_ring), 60
QuotientRing_nc (class in sage.rings.quotient_ring), 60
QuotientRingElement (class in sage.rings.quotient_ring_element), 68

R

random_element() (sage.rings.fraction_field.FractionField_generic method), 76
random_element() (sage.rings.ring.Ring method), 20
reduce() (sage.rings.fraction_field_element.FractionFieldElement method), 79
reduce() (sage.rings.ideal.Ideal_generic method), 34
reduce() (sage.rings.ideal.Ideal_pid method), 37
reduce() (sage.rings.quotient_ring_element.QuotientRingElement method), 71
repr_short_to_parent() (in module sage.rings.asymptotic.growth_group), 99
residue_field() (sage.rings.ideal.Ideal_pid method), 37
retract() (sage.rings.quotient_ring.QuotientRing_nc method), 67
Ring (class in sage.rings.ring), 12
ring() (sage.rings.fraction_field.FractionField_generic method), 76
ring() (sage.rings.ideal.Ideal_generic method), 34
ring() (sage.rings.ideal_monoid.IdealMonoid_c method), 40
ring_of_integers() (sage.rings.fraction_field.FractionField_1poly_field method), 74
RingHomomorphism (class in sage.rings.morphism), 49
RingHomomorphism_coercion (class in sage.rings.morphism), 50
RingHomomorphism_cover (class in sage.rings.morphism), 51
RingHomomorphism_from_base (class in sage.rings.morphism), 51
RingHomomorphism_from_quotient (class in sage.rings.morphism), 53
RingHomomorphism_im_gens (class in sage.rings.morphism), 54
RingHomset() (in module sage.rings.homset), 55

RingHomset_generic (class in sage.rings.homset), 55
RingHomset_quo_ring (class in sage.rings.homset), 55
RingMap (class in sage.rings.morphism), 54
RingMap_lift (class in sage.rings.morphism), 54

S

sage.rings.asymptotic.growth_group (module), 93
sage.rings.asymptotic.term_monoid (module), 100
sage.rings.big_oh (module), 83
sage.rings.commutative_algebra (module), 23
sage.rings.commutative_ring (module), 23
sage.rings.dedekind_domain (module), 23
sage.rings.euclidean_domain (module), 23
sage.rings.fraction_field (module), 73
sage.rings.fraction_field_element (module), 77
sage.rings.homset (module), 55
sage.rings.ideal (module), 25
sage.rings.ideal_monoid (module), 40
sage.rings.infinity (module), 84
sage.rings.integral_domain (module), 23
sage.rings.misc (module), 93
sage.rings.morphism (module), 43
sage.rings.noncommutative_ideals (module), 40
sage.rings.principal_ideal_domain (module), 24
sage.rings.quotient_ring (module), 57
sage.rings.quotient_ring_element (module), 68
sage.rings.ring (module), 1
side() (sage.rings.noncommutative_ideals.Ideal_nc method), 42
SignError, 90
sqrt() (sage.rings.infinity.FiniteNumber method), 88
sqrt() (sage.rings.infinity.MinusInfinity method), 89
sqrt() (sage.rings.infinity.PlusInfinity method), 90
support() (sage.rings.fraction_field_element.FractionFieldElement_1poly_field method), 80

T

term_order() (sage.rings.quotient_ring.QuotientRing_nc method), 68
TermMonoidFactory (class in sage.rings.asymptotic.term_monoid), 109
TermWithCoefficient (class in sage.rings.asymptotic.term_monoid), 110
TermWithCoefficientMonoid (class in sage.rings.asymptotic.term_monoid), 111
test_comparison() (in module sage.rings.infinity), 91
test_signed_infinity() (in module sage.rings.infinity), 92

U

underlying_map() (sage.rings.morphism.RingHomomorphism_from_base method), 52
unit_ideal() (sage.rings.ring.Ring method), 20
UnsignedInfinity (class in sage.rings.infinity), 90
UnsignedInfinityRing_class (class in sage.rings.infinity), 90

V

valuation() (sage.rings.fraction_field_element.FractionFieldElement method), 79

variables() (sage.rings.quotient_ring_element.QuotientRingElement method), 71

Z

zero() (sage.rings.ring.Ring method), 21
zero_ideal() (sage.rings.ring.Ring method), 21
zeta() (sage.rings.ring.Ring method), 21
zeta_order() (sage.rings.ring.Ring method), 22