Sage Reference Manual: Asymptotic Expansions

Release 7.1

The Sage Development Team

March 20, 2016

CONTENTS

CHAPTER ONE

THE ASYMPTOTIC RING

The asymptotic ring, as well as its main documentation is contained in the module

- *Asymptotic Ring*.

CHAPTER

TWO

ASYMPTOTIC EXPANSION GENERATORS

Some common asymptotic expansions can be generated in

- *Common Asymptotic Expansions*.

CHAPTER

THREE

SUPPLEMENTS

Behind the scenes of working with asymptotic expressions a couple of additional classes and tools turn up. For instance the growth of each summand is managed in growth groups, see below.

3.1 Growth Groups

The growth of a summand of an asymptotic expression is managed in

- *(Asymptotic) Growth Groups* and
- *Cartesian Products of Growth Groups*.

3.2 Term Monoids

A summand of an asymptotic expression is basically a term out of the following monoid:

- *(Asymptotic) Term Monoids*.

3.3 Miscellaneous

Various useful functions and tools are collected in

- *Asymptotic Expansions — Miscellaneous*.

CHAPTER

FOUR

ASYMPTOTIC EXPANSIONS — TABLE OF CONTENTS

4.1 Asymptotic Ring

This module provides a ring (called `AsymptoticRing`) for computations with asymptotic expansions.

4.1.1 (Informal) Definition

An asymptotic expansion is a sum such as

$$5z^3 + 4z^2 + O(z)$$

as $z \to \infty$ or

$$3x^{42}y^2 + 7x^3y^3 + O(x^2) + O(y)$$

as x and y tend to ∞. It is a truncated series (after a finite number of terms), which approximates a function.

The summands of the asymptotic expansions are partially ordered. In this module these summands are the following:

- Exact terms $c \cdot g$ with a coefficient c and an element g of a growth group (*see below*).
- O-terms $O(g)$ (see Big O notation; also called *Bachmann–Landau notation*) for a growth group element g (*again see below*).

See the Wikipedia article on asymptotic expansions for more details. Further examples of such elements can be found *here*.

Growth Groups and Elements

The elements of a *growth group* are equipped with a partial order and usually contain a variable. Examples—the order is described below these examples—are

- elements of the form z^q for some integer or rational q (growth groups with *description strings* `z^ZZ` or `z^QQ`),
- elements of the form $\log(z)^q$ for some integer or rational q (growth groups `log(z)^ZZ` or `log(z)^QQ`),
- elements of the form a^z for some rational a (growth group `QQ^z`), or
- more sophisticated constructions like products $x^r \cdot \log(x)^s \cdot a^y \cdot y^q$ (this corresponds to an element of the growth group `x^QQ * log(x)^ZZ * QQ^y * y^QQ`).

The order in all these examples is induced by the magnitude of the elements as x, y, or z (independently) tend to ∞. For elements only using the variable z this means that $g_1 \le g_2$ if

$$\lim_{z\to\infty} \frac{g_1}{g_2} \le 1.$$

Note: Asymptotic rings where the variable tend to some value distinct from ∞ are not yet implemented.

To find out more about

- growth groups,
- on how they are created and
- about the above used *descriptions strings*

see the top of the module *growth group*.

Warning: As this code is experimental, a warning is thrown when an asymptotic ring (or an associated structure) is created for the first time in a session (see `sage.misc.superseded.experimental`).
TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^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.
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: T = GenericTermMonoid(G, ZZ)
sage: R.<x, y> = AsymptoticRing(growth_group='x^ZZ * y^ZZ', coefficient_ring=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.
```

4.1.2 Introductory Examples

We start this series of examples by defining two asymptotic rings.

Two Rings

A Univariate Asymptotic Ring

First, we construct the following (very simple) asymptotic ring in the variable z:

```
sage: A.<z> = AsymptoticRing(growth_group='z^QQ', coefficient_ring=ZZ); A
Asymptotic Ring <z^QQ> over Integer Ring
```

A typical element of this ring is

```
sage: A.an_element()
z^(3/2) + O(z^(1/2))
```

This element consists of two summands: the exact term with coefficient 1 and growth $z^{3/2}$ and the O-term $O(z^{1/2})$. Note that the growth of $z^{3/2}$ is larger than the growth of $z^{1/2}$ as $z \to \infty$, thus this expansion cannot be simplified (which would be done automatically, see below).

Elements can be constructed via the generator z and the function `O()`, for example

```
sage: 4*z^2 + O(z)
4*z^2 + O(z)
```

A Multivariate Asymptotic Ring

Next, we construct a more sophisticated asymptotic ring in the variables x and y by

```
sage: B.<x, y> = AsymptoticRing(growth_group='x^QQ * log(x)^ZZ * QQ^y * y^QQ', coefficient_ring=QQ);
Asymptotic Ring <x^QQ * log(x)^ZZ * QQ^y * y^QQ> over Rational Field
```

Again, we can look at a typical (nontrivial) element:

```
sage: B.an_element()
1/8*x^(3/2)*log(x)^3*(1/8)^y*y^(3/2) + O(x^(1/2)*log(x)*(1/2)^y*y^(1/2))
```

Again, elements can be created using the generators x and y, as well as the function `O()`:

```
sage: log(x)*y/42 + O(1/2^y)
1/42*log(x)*y + O((1/2)^y)
```

Arithmetical Operations

In this section we explain how to perform various arithmetical operations with the elements of the asymptotic rings constructed above.

The Ring Operations Plus and Times

We start our calculations in the ring

```
sage: A
Asymptotic Ring <z^QQ> over Integer Ring
```

Of course, we can perform the usual ring operations $+$ and $*$:

```
sage: z^2 + 3*z*(1-z)
-2*z^2 + 3*z
sage: (3*z + 2)^3
27*z^3 + 54*z^2 + 36*z + 8
```

In addition to that, special powers—our growth group `z^QQ` allows the exponents to be out of **Q**—can also be computed:

```
sage: (z^(5/2)+z^(1/7)) * z^(-1/5)
z^(23/10) + z^(-2/35)
```

The central concepts of computations with asymptotic expansions is that the O-notation can be used. For example, we have

```
sage: z^3 + z^2 + z + O(z^2)
z^3 + O(z^2)
```

where the result is simplified automatically. A more sophisticated example is

```
sage: (z+2*z^2+3*z^3+4*z^4) * (O(z)+z^2)
4*z^6 + O(z^5)
```

Division

The asymptotic expansions support division. For example, we can expand $1/(z-1)$ to a geometric series:

```
sage: 1 / (z-1)
z^(-1) + z^(-2) + z^(-3) + z^(-4) + ... + z^(-20) + O(z^(-21))
```

A default precision (parameter `default_prec` of `AsymptoticRing`) is predefined. Thus, only the first 20 summands are calculated. However, if we only want the first 5 exact terms, we cut of the rest by using

```
sage: (1 / (z-1)).truncate(5)
z^(-1) + z^(-2) + z^(-3) + z^(-4) + z^(-5) + O(z^(-6))
```

or

```
sage: 1 / (z-1) + O(z^(-6))
z^(-1) + z^(-2) + z^(-3) + z^(-4) + z^(-5) + O(z^(-6))
```

Of course, we can work with more complicated expansions as well:

```
sage: (4*z+1) / (z^3+z^2+z+O(z^0))
4*z^(-2) - 3*z^(-3) - z^(-4) + O(z^(-5))
```

Not all elements are invertible, for instance,

```
sage: 1 / O(z)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot invert O(z).
```

is not invertible, since it includes 0.

Powers, Expontials and Logarithms

It works as simple as it can be; just use the usual operators `^`, `exp` and `log`. For example, we obtain the usual series expansion of the logarithm

```
sage: -log(1-1/z)
z^(-1) + 1/2*z^(-2) + 1/3*z^(-3) + ... + O(z^(-21))
```

as $z \to \infty$.

Similarly, we can apply the exponential function of an asymptotic expansion:

```
sage: exp(1/z)
1 + z^(-1) + 1/2*z^(-2) + 1/6*z^(-3) + 1/24*z^(-4) + ... + O(z^(-20))
```

Arbitrary powers work as well; for example, we have

```
sage: (1 + 1/z + O(1/z^5))^(1 + 1/z)
1 + z^(-1) + z^(-2) + 1/2*z^(-3) + 1/3*z^(-4) + O(z^(-5))
```

Note: In the asymptotic ring

```
sage: M.<n> = AsymptoticRing(growth_group='QQ^n * n^QQ', coefficient_ring=ZZ)
```

the operation

```
sage: (1/2)^n
Traceback (most recent call last):
...
ValueError: 1/2 is not in Exact Term Monoid QQ^n * n^QQ
with coefficients in Integer Ring. ...
```

fails, since the rational $1/2$ is not contained in M. You can use

```
sage: n.rpow(1/2)
(1/2)^n
```

instead. (See also the examples in `ExactTerm.rpow()` for a detailed explanation.) Another way is to use a larger coefficent ring:

```
sage: M_QQ.<n> = AsymptoticRing(growth_group='QQ^n * n^QQ', coefficient_ring=QQ)
sage: (1/2)^n
(1/2)^n
```

Multivariate Arithmetic

Now let us move on to arithmetic in the multivariate ring

```
sage: B
Asymptotic Ring <x^QQ * log(x)^ZZ * QQ^y * y^QQ> over Rational Field
```

Todo

write this part

4.1.3 More Examples

The mathematical constant e as a limit

The base of the natural logarithm e satisfies the equation

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

By using asymptotic expansions, we obtain the more precise result

```
sage: E.<n> = AsymptoticRing(growth_group='n^ZZ', coefficient_ring=SR, default_prec=5); E
Asymptotic Ring <n^ZZ> over Symbolic Ring
sage: (1 + 1/n)^n
e - 1/2*e*n^(-1) + 11/24*e*n^(-2) - 7/16*e*n^(-3) + 2447/5760*e*n^(-4) + O(n^(-5))
```

4.1.4 Selected Technical Details

Coercions and Functorial Constructions

The `AsymptoticRing` fully supports coercion. For example, the coefficient ring is automatically extended when needed:

```
sage: A
Asymptotic Ring <z^QQ> over Integer Ring
sage: (z + 1/2).parent()
Asymptotic Ring <z^QQ> over Rational Field
```

Here, the coefficient ring was extended to allow $1/2$ as a coefficent. Another example is

```
sage: C.<c> = AsymptoticRing(growth_group='c^ZZ', coefficient_ring=ZZ['e'])
sage: C.an_element()
e^3*c^3 + O(c)
sage: C.an_element() / 7
1/7*e^3*c^3 + O(c)
```

Here the result's coefficient ring is the newly found

```
sage: (C.an_element() / 7).parent()
Asymptotic Ring <c^ZZ> over
Univariate Polynomial Ring in e over Rational Field
```

Not only the coefficient ring can be extended, but the growth group as well. For example, we can add/multiply elements of the asymptotic rings `A` and `C` to get an expansion of new asymptotic ring:

```
sage: r = c*z + c/2 + O(z); r
c*z + 1/2*c + O(z)
sage: r.parent()
Asymptotic Ring <c^ZZ * z^QQ> over
Univariate Polynomial Ring in e over Rational Field
```

Data Structures

The summands of an `asymptotic expansion` are wrapped *growth group elements*. This wrapping is done by the *term monoid module*. However, inside an `asymptotic expansion` these summands (terms) are stored together with their growth-relationship, i.e., each summand knows its direct predecessors and successors. As a data structure a special poset (namely a `mutable poset`) is used. We can have a look at this:

```
sage: b = x^3*y + x^2*y + x*y^2 + O(x) + O(y)
sage: print b.summands.repr_full(reverse=True)
poset(x*y^2, x^3*y, x^2*y, O(x), O(y))
+-- oo
|   +-- no successors
|   +-- predecessors:   x*y^2, x^3*y
+-- x*y^2
|   +-- successors:   oo
|   +-- predecessors:   O(x), O(y)
+-- x^3*y
|   +-- successors:   oo
|   +-- predecessors:   x^2*y
+-- x^2*y
|   +-- successors:   x^3*y
```

```
|   +-- predecessors:   O(x), O(y)
+-- O(x)
|   +-- successors:   x*y^2, x^2*y
|   +-- predecessors:   null
+-- O(y)
|   +-- successors:   x*y^2, x^2*y
|   +-- predecessors:   null
+-- null
|   +-- successors:   O(x), O(y)
|   +-- no predecessors
```

4.1.5 Various

AUTHORS:

- Benjamin Hackl (2015)
- Daniel Krenn (2015)
- Clemens Heuberger (2016)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

4.1.6 Classes and Methods

class `sage.rings.asymptotic.asymptotic_ring.`**`AsymptoticExpansion`**(*parent*, *summands*, *simplify=True*, *convert=True*)

Bases: `sage.structure.element.CommutativeAlgebraElement`

Class for asymptotic expansions, i.e., the elements of an `AsymptoticRing`.

INPUT:

- `parent` – the parent of the asymptotic expansion.
- `summands` – the summands as a `MutablePoset`, which represents the underlying structure.
- `simplify` – a boolean (default: `True`). It controls automatic simplification (absorption) of the asymptotic expansion.
- `convert` – a boolean (default: `True`). If set, then the `summands` are converted to the asymptotic ring (the parent of this expansion). If not, then the summands are taken as they are. In that case, the caller must ensure that the parent of the terms is set correctly.

EXAMPLES:

There are several ways to create asymptotic expansions; usually this is done by using the corresponding `asymptotic rings`:

```
sage: R_x.<x> = AsymptoticRing(growth_group='x^QQ', coefficient_ring=QQ); R_x
Asymptotic Ring <x^QQ> over Rational Field
sage: R_y.<y> = AsymptoticRing(growth_group='y^ZZ', coefficient_ring=ZZ); R_y
Asymptotic Ring <y^ZZ> over Integer Ring
```

At this point, x and y are already asymptotic expansions:

```
sage: type(x)
<class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.element_class'>
```

The usual ring operations, but allowing rational exponents (growth group `x^QQ`) can be performed:

```
sage: x^2 + 3*(x - x^(2/5))
x^2 + 3*x - 3*x^(2/5)
sage: (3*x^(1/3) + 2)^3
27*x + 54*x^(2/3) + 36*x^(1/3) + 8
```

One of the central ideas behind computing with asymptotic expansions is that the O-notation (see Wikipedia article Big_O_notation) can be used. For example, we have:

```
sage: (x+2*x^2+3*x^3+4*x^4) * (O(x)+x^2)
4*x^6 + O(x^5)
```

In particular, `O()` can be used to construct the asymptotic expansions. With the help of the `summands()`, we can also have a look at the inner structure of an asymptotic expansion:

```
sage: expr1 = x + 2*x^2 + 3*x^3 + 4*x^4; expr2 = O(x) + x^2
sage: print(expr1.summands.repr_full())
poset(x, 2*x^2, 3*x^3, 4*x^4)
+-- null
|   +-- no predecessors
|   +-- successors:   x
+-- x
|   +-- predecessors:   null
|   +-- successors:   2*x^2
+-- 2*x^2
|   +-- predecessors:   x
|   +-- successors:   3*x^3
+-- 3*x^3
|   +-- predecessors:   2*x^2
|   +-- successors:   4*x^4
+-- 4*x^4
|   +-- predecessors:   3*x^3
|   +-- successors:   oo
+-- oo
|   +-- predecessors:   4*x^4
|   +-- no successors
sage: print(expr2.summands.repr_full())
poset(O(x), x^2)
+-- null
|   +-- no predecessors
|   +-- successors:   O(x)
+-- O(x)
|   +-- predecessors:   null
|   +-- successors:   x^2
+-- x^2
|   +-- predecessors:   O(x)
|   +-- successors:   oo
+-- oo
|   +-- predecessors:   x^2
|   +-- no successors
sage: print((expr1 * expr2).summands.repr_full())
poset(O(x^5), 4*x^6)
+-- null
|   +-- no predecessors
```

```
|   +-- successors:   O(x^5)
+-- O(x^5)
|   +-- predecessors:   null
|   +-- successors:   4*x^6
+-- 4*x^6
|   +-- predecessors:   O(x^5)
|   +-- successors:   oo
+-- oo
|   +-- predecessors:   4*x^6
|   +-- no successors
```

In addition to the monomial growth elements from above, we can also compute with logarithmic terms (simply by constructing the appropriate growth group):

```
sage: R_log = AsymptoticRing(growth_group='log(x)^QQ', coefficient_ring=QQ)
sage: lx = R_log(log(SR.var('x')))
sage: (O(lx) + lx^3)^4
log(x)^12 + O(log(x)^10)
```

See also:

(Asymptotic) Growth Groups, *(Asymptotic) Term Monoids*, `mutable_poset`.

O()

Convert all terms in this asymptotic expansion to O-terms.

INPUT:

Nothing.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: AR.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: O(x)
O(x)
sage: type(O(x))
<class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.element_class'>
sage: expr = 42*x^42 + x^10 + O(x^2); expr
42*x^42 + x^10 + O(x^2)
sage: expr.O()
O(x^42)
sage: (2*x).O()
O(x)
```

See also:

`sage.rings.power_series_ring.PowerSeriesRing()`, `sage.rings.laurent_series_ring.Laure`

TESTS:

```
sage: AR(0).O()
Traceback (most recent call last):
...
NotImplementedOZero: The error term in the result is O(0)
which means 0 for sufficiently large x.
```

compare_with_values (*variable*, *function*, *values*, *rescaled=True*, *ring=Real Interval Field with 53 bits of precision*)

Compute the (rescaled) difference between this asymptotic expansion and the given values.

INPUT:

- `variable` – an asymptotic expansion or a string.
- `function` – a callable or symbolic expression giving the comparison values.
- `values` – a list or iterable of values where the comparison shall be carried out.
- `rescaled` – (default: `True`) determines whether the difference is divided by the error term of the asymptotic expansion.
- `ring` – (default: `RIF`) the parent into which the difference is converted.

OUTPUT:

A list of pairs containing comparison points and (rescaled) difference values.

EXAMPLES:

```
sage: A.<n> = AsymptoticRing('QQ^n * n^ZZ', SR)
sage: catalan = binomial(2*x, x)/(x+1)
sage: expansion = 4^n*(1/sqrt(pi)*n^(-3/2)
....:     - 9/8/sqrt(pi)*n^(-5/2)
....:     + 145/128/sqrt(pi)*n^(-7/2) + O(n^(-9/2)))
sage: expansion.compare_with_values(n, catalan, srange(5, 10))
[(5, 0.5303924444775?),
 (6, 0.5455279498787?),
 (7, 0.556880411050?),
 (8, 0.565710587724?),
 (9, 0.572775029098?)]
sage: expansion.compare_with_values(n, catalan, [5, 10, 20], rescaled=False)
[(5, 0.3886263699387?), (10, 19.1842458318?), (20, 931314.63637?)]
sage: expansion.compare_with_values(n, catalan, [5, 10, 20], rescaled=False, ring=SR)
[(5, 168/5*sqrt(5)/sqrt(pi) - 42),
 (10, 1178112/125*sqrt(10)/sqrt(pi) - 16796),
 (20, 650486218752/125*sqrt(5)/sqrt(pi) - 6564120420)]
```

Instead of a symbolic expression, a callable function can be specified as well:

```
sage: A.<n> = AsymptoticRing('n^ZZ * log(n)^ZZ', SR)
sage: def H(n):
....:     return sum(1/k for k in srange(1, n+1))
sage: H_expansion = (log(n) + euler_gamma + 1/(2*n)
....:                - 1/(12*n^2) + O(n^-4))
sage: H_expansion.compare_with_values(n, H, srange(25, 30)) # rel tol 1e-6
[(25, -0.008326995?),
 (26, -0.008327472?),
 (27, -0.008327898?),
 (28, -0.00832828?),
 (29, -0.00832862?)]
```

See also:

`plot_comparison()`

TESTS:

```
sage: A.<x, y> = AsymptoticRing('x^ZZ*y^ZZ', QQ)
sage: expansion = x^2 + O(x) + O(y)
sage: expansion.compare_with_values(y, lambda z: z^2, srange(20, 30))
```

```
Traceback (most recent call last):
....
NotImplementedError: exactly one error term required
sage: expansion = x^2
sage: expansion.compare_with_values(y, lambda z: z^2, srange(20, 30))
Traceback (most recent call last):
....
NotImplementedError: exactly one error term required
sage: expansion = x^2 + O(x)
sage: expansion.compare_with_values(y, lambda z: z^2, srange(20, 30))
Traceback (most recent call last):
....
NameError: name 'x' is not defined
sage: expansion.compare_with_values(x, lambda z: z^2, srange(20, 30))
[(20, 0), (21, 0), ..., (29, 0)]
sage: expansion.compare_with_values(x, SR('x*y'), srange(20, 30))
Traceback (most recent call last):
....
NotImplementedError: expression x*y has more than one variable
```

exact_part()

Return the expansion consisting of all exact terms of this expansion.

INPUT:

Nothing

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: R.<x> = AsymptoticRing('x^QQ * log(x)^QQ', QQ)
sage: (x^2 + O(x)).exact_part()
x^2
sage: (x + log(x)/2 + O(log(x)/x)).exact_part()
x + 1/2*log(x)
```

TESTS:

```
sage: R.<x, y> = AsymptoticRing('x^QQ * y^QQ', QQ)
sage: (x + y + O(1/(x*y))).exact_part()
x + y
sage: O(x).exact_part()
0
```

exp(*precision=None*)

Return the exponential of (i.e., the power of e to) this asymptotic expansion.

INPUT:

- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

Note: The exponential function of this expansion can only be computed exactly if the respective growth element can be constructed in the underlying growth group.

ALGORITHM:

If the corresponding growth can be constructed, return the exact exponential function. Otherwise, if this term is $o(1)$, try to expand the series and truncate according to the given precision.

Todo

As soon as L-terms are implemented, this implementation has to be adapted as well in order to yield correct results.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing('(e^x)^ZZ * x^ZZ * log(x)^ZZ', SR)
sage: exp(x)
e^x
sage: exp(2*x)
(e^x)^2
sage: exp(x + log(x))
e^x*x

sage: (x^(-1)).exp(precision=7)
1 + x^(-1) + 1/2*x^(-2) + 1/6*x^(-3) + ... + O(x^(-7))
```

TESTS:

```
sage: A.<x> = AsymptoticRing('(e^x)^ZZ * x^QQ * log(x)^QQ', SR)
sage: exp(log(x))
x
sage: log(exp(x))
x

sage: exp(x+1)
e*e^x
```

See trac ticket #19521:

```
sage: A.<n> = AsymptoticRing('n^ZZ', SR.subring(no_variables=True))
sage: exp(O(n^(-3))).parent()
Asymptotic Ring <n^ZZ> over Symbolic Constants Subring
```

factorial()

Return the factorial of this asymptotic expansion.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: A.<n> = AsymptoticRing(growth_group='n^ZZ * log(n)^ZZ', coefficient_ring=ZZ, default_p
sage: n.factorial()
sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(1/2)
+ 1/12*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-1/2)
+ 1/288*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-3/2)
+ O(e^(n*log(n))*(e^n)^(-1)*n^(-5/2))
sage: _.parent()
Asymptotic Ring <(e^(n*log(n)))^(Symbolic Constants Subring) *
                 (e^n)^(Symbolic Constants Subring) *
                 n^(Symbolic Constants Subring) *
```

```
                    log(n)^(Symbolic Constants Subring)>
over Symbolic Constants Subring
```

Catalan numbers $\frac{1}{n+1}\binom{2n}{n}$:

```
sage: (2*n).factorial() / n.factorial()^2 / (n+1)  # long time
1/sqrt(pi)*(e^n)^(2*log(2))*n^(-3/2)
- 9/8/sqrt(pi)*(e^n)^(2*log(2))*n^(-5/2)
+ 145/128/sqrt(pi)*(e^n)^(2*log(2))*n^(-7/2)
+ O((e^n)^(2*log(2))*n^(-9/2))
```

Note that this method substitutes the asymptotic expansion into Stirling's formula. This substitution has to be possible which is not always guaranteed:

```
sage: S.<s> = AsymptoticRing(growth_group='s^QQ * log(s)^QQ', coefficient_ring=QQ, default_p
sage: log(s).factorial()
Traceback (most recent call last):
...
TypeError: Cannot apply the substitution rules {s: log(s)} on
sqrt(2)*sqrt(pi)*e^(s*log(s))*(e^s)^(-1)*s^(1/2)
+ O(e^(s*log(s))*(e^s)^(-1)*s^(-1/2)) in
Asymptotic Ring <(e^(s*log(s)))^QQ * (e^s)^QQ * s^QQ * log(s)^QQ>
over Symbolic Constants Subring.
...
```

See also:

`Stirling()`

TESTS:

```
sage: A.<m> = AsymptoticRing(growth_group='m^ZZ * log(m)^ZZ', coefficient_ring=QQ, default_p
sage: m.factorial()
sqrt(2)*sqrt(pi)*e^(m*log(m))*(e^m)^(-1)*m^(1/2)
+ 1/12*sqrt(2)*sqrt(pi)*e^(m*log(m))*(e^m)^(-1)*m^(-1/2)
+ 1/288*sqrt(2)*sqrt(pi)*e^(m*log(m))*(e^m)^(-1)*m^(-3/2)
+ O(e^(m*log(m))*(e^m)^(-1)*m^(-5/2))

sage: A(1/2).factorial()
1/2*sqrt(pi)
sage: _.parent()
Asymptotic Ring <m^ZZ * log(m)^ZZ> over Symbolic Ring

sage: B.<a, b> = AsymptoticRing('a^ZZ * b^ZZ', QQ, default_prec=3)
sage: b.factorial()
O(e^(b*log(b))*(e^b)^(-1)*b^(1/2))
sage: (a*b).factorial()
Traceback (most recent call last):
...
ValueError: Cannot build the factorial of a*b
since it is not univariate.
```

has_same_summands(*other*)

Return whether this asymptotic expansion and `other` have the same summands.

INPUT:

- `other` – an asymptotic expansion.

OUTPUT:

A boolean.

Note: While for example `O(x) == O(x)` yields `False`, these expansions *do* have the same summands and this method returns `True`.

Moreover, this method uses the coercion model in order to find a common parent for this asymptotic expansion and `other`.

EXAMPLES:

```
sage: R_ZZ.<x_ZZ> = AsymptoticRing('x^ZZ', ZZ)
sage: R_QQ.<x_QQ> = AsymptoticRing('x^ZZ', QQ)
sage: sum(x_ZZ^k for k in range(5)) == sum(x_QQ^k for k in range(5))  # indirect doctest
True
sage: O(x_ZZ) == O(x_QQ)
False
```

TESTS:

```
sage: x_ZZ.has_same_summands(None)
False
```

invert (*precision=None*)

Return the multiplicative inverse of this element.

INPUT:

- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

Warning: Due to truncation of infinite expansions, the element returned by this method might not fulfill `el * ~el == 1`.

Todo

As soon as L-terms are implemented, this implementation has to be adapted as well in order to yield correct results.

EXAMPLES:

```
sage: R.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=QQ, default_prec=4)
sage: ~x
x^(-1)
sage: ~(x^42)
x^(-42)
sage: ex = ~(1 + x); ex
x^(-1) - x^(-2) + x^(-3) - x^(-4) + O(x^(-5))
sage: ex * (1+x)
1 + O(x^(-4))
sage: ~(1 + O(1/x))
1 + O(x^(-1))
```

TESTS:

```
sage: A.<a> = AsymptoticRing(growth_group='a^ZZ', coefficient_ring=ZZ)
sage: (1 / a).parent()
```

```
Asymptotic Ring <a^ZZ> over Rational Field
sage: (a / 2).parent()
Asymptotic Ring <a^ZZ> over Rational Field

sage: ~A(0)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot invert 0 in
Asymptotic Ring <a^ZZ> over Integer Ring.

sage: B.<s, t> = AsymptoticRing(growth_group='s^ZZ * t^ZZ', coefficient_ring=QQ)
sage: ~(s + t)
Traceback (most recent call last):
...
ValueError: Cannot determine main term of s + t since there
are several maximal elements s, t.
```

is_exact()

Return whether all terms of this expansion are exact.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing('x^QQ * log(x)^QQ', QQ)
sage: (x^2 + O(x)).is_exact()
False
sage: (x^2 - x).is_exact()
True
```

TESTS:

```
sage: A(0).is_exact()
True
sage: A.one().is_exact()
True
```

is_little_o_of_one()

Return whether this expansion is of order $o(1)$.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing('x^ZZ * log(x)^ZZ', QQ)
sage: (x^4 * log(x)^(-2) + x^(-4) * log(x)^2).is_little_o_of_one()
False
sage: (x^(-1) * log(x)^1234 + x^(-2) + O(x^(-3))).is_little_o_of_one()
True
sage: (log(x) - log(x-1)).is_little_o_of_one()
True

sage: A.<x, y> = AsymptoticRing('x^QQ * y^QQ * log(y)^ZZ', QQ)
sage: (x^(-1/16) * y^32 + x^32 * y^(-1/16)).is_little_o_of_one()
```

```
False
sage: (x^(-1) * y^(-3) + x^(-3) * y^(-1)).is_little_o_of_one()
True
sage: (x^(-1) * y / log(y)).is_little_o_of_one()
False
sage: (log(y-1)/log(y) - 1).is_little_o_of_one()
True
```

log(*base=None*, *precision=None*)

The logarithm of this asymptotic expansion.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.
- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

Note: Computing the logarithm of an asymptotic expansion is possible if and only if there is exactly one maximal summand in the expansion.

ALGORITHM:

If the expansion has more than one summand, the asymptotic expansion for $\log(1 + t)$ as t tends to 0 is used.

Todo

As soon as L-terms are implemented, this implementation has to be adapted as well in order to yield correct results.

EXAMPLES:

```
sage: R.<x> = AsymptoticRing(growth_group='x^ZZ * log(x)^ZZ', coefficient_ring=QQ)
sage: log(x)
log(x)
sage: log(x^2)
2*log(x)
sage: log(x-1)
log(x) - x^(-1) - 1/2*x^(-2) - 1/3*x^(-3) - ... + O(x^(-21))
```

TESTS:

```
sage: log(R(1))
0
sage: log(R(0))
Traceback (most recent call last):
...
ArithmeticError: Cannot compute log(0) in
Asymptotic Ring <x^ZZ * log(x)^ZZ> over Rational Field.
sage: C.<s, t> = AsymptoticRing(growth_group='s^ZZ * t^ZZ', coefficient_ring=QQ)
sage: log(s + t)
Traceback (most recent call last):
...
ValueError: Cannot determine main term of s + t since
there are several maximal elements s, t.
```

map_coefficients (*f*, *new_coefficient_ring=None*)

Return the asymptotic expansion obtained by applying `f` to each coefficient of this asymptotic expansion.

INPUT:

- `f` – a callable. A coefficient c will be mapped to $f(c)$.
- `new_coefficient_ring` – (default: `None`) a ring.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: A.<n> = AsymptoticRing(growth_group='n^ZZ', coefficient_ring=ZZ)
sage: a = n^4 + 2*n^3 + 3*n^2 + O(n)
sage: a.map_coefficients(lambda c: c+1)
2*n^4 + 3*n^3 + 4*n^2 + O(n)
sage: a.map_coefficients(lambda c: c-2)
-n^4 + n^2 + O(n)
```

TESTS:

```
sage: a.map_coefficients(lambda c: 1/c, new_coefficient_ring=QQ)
n^4 + 1/2*n^3 + 1/3*n^2 + O(n)
sage: _.parent()
Asymptotic Ring <n^ZZ> over Rational Field
sage: a.map_coefficients(lambda c: 1/c)
Traceback (most recent call last):
...
ValueError: ... is not a coefficient in
Exact Term Monoid n^ZZ with coefficients in Integer Ring.
```

plot_comparison (*variable*, *function*, *values*, *rescaled=True*, *ring=Real Interval Field with 53 bits of precision*, *relative_tolerance=0.025*, ***kwargs*)

Plot the (rescaled) difference between this asymptotic expansion and the given values.

INPUT:

- `variable` – an asymptotic expansion or a string.
- `function` – a callable or symbolic expression giving the comparison values.
- `values` – a list or iterable of values where the comparison shall be carried out.
- `rescaled` – (default: `True`) determines whether the difference is divided by the error term of the asymptotic expansion.
- `ring` – (default: `RIF`) the parent into which the difference is converted.
- `relative_tolerance` – (default: `0.025`). Raise error when relative error exceeds this tolerance.

Other keyword arguments are passed to `list_plot()`.

OUTPUT:

A graphics object.

Note: If rescaled (i.e. divided by the error term), the output should be bounded.

This method is mainly meant to have an easily usable plausability check for asymptotic expansion created in some way.

EXAMPLES:

We want to check the quality of the asymptotic expansion of the harmonic numbers:

```
sage: A.<n> = AsymptoticRing('n^ZZ * log(n)^ZZ', SR)
sage: def H(n):
....:     return sum(1/k for k in srange(1, n+1))
sage: H_expansion = (log(n) + euler_gamma + 1/(2*n)
....:                - 1/(12*n^2) + O(n^-4))
sage: H_expansion.plot_comparison(n, H, srange(1, 30))
Graphics object consisting of 1 graphics primitive
```

Alternatively, the unscaled (absolute) difference can be plotted as well:

```
sage: H_expansion.plot_comparison(n, H, srange(1, 30),
....:                             rescaled=False)
Graphics object consisting of 1 graphics primitive
```

Additional keywords are passed to `list_plot()`:

```
sage: H_expansion.plot_comparison(n, H, srange(1, 30),
....:                             plotjoined=True, marker='o',
....:                             color='green')
Graphics object consisting of 1 graphics primitive
```

See also:

`compare_with_values()`

TESTS:

```
sage: H_expansion.plot_comparison(n, H, [600])
Traceback (most recent call last):
...
ValueError: Numerical noise is too high, the comparison is inaccurate
sage: H_expansion.plot_comparison(n, H, [600], relative_tolerance=2)
Graphics object consisting of 1 graphics primitive
```

pow(*exponent*, *precision=None*)

Calculate the power of this asymptotic expansion to the given `exponent`.

INPUT:

- `exponent` – an element.
- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: Q.<x> = AsymptoticRing(growth_group='x^QQ', coefficient_ring=QQ)
sage: x^(1/7)
x^(1/7)
sage: (x^(1/2) + O(x^0))^15
x^(15/2) + O(x^7)

sage: Z.<y> = AsymptoticRing(growth_group='y^ZZ', coefficient_ring=ZZ)
sage: y^(1/7)
```

```
y^(1/7)
sage: _.parent()
Asymptotic Ring <y^QQ> over Rational Field
sage: (y^2 + O(y))^(1/2)
y + O(1)
sage: (y^2 + O(y))^(-2)
y^(-4) + O(y^(-5))
sage: (1 + 1/y + O(1/y^3))^pi
1 + pi*y^(-1) + (1/2*pi*(pi - 1))*y^(-2) + O(y^(-3))

sage: B.<z> = AsymptoticRing(growth_group='z^QQ * log(z)^QQ', coefficient_ring=QQ)
sage: (z^2 + O(z))^(1/2)
z + O(1)

sage: A.<x> = AsymptoticRing('QQ^x * x^SR * log(x)^ZZ', QQ)
sage: x * 2^x
2^x*x
sage: 5^x * 2^x
10^x
sage: 2^log(x)
x^(log(2))
sage: 2^(x + 1/x)
2^x + log(2)*2^x*x^(-1) + 1/2*log(2)^2*2^x*x^(-2) + ... + O(2^x*x^(-20))
sage: _.parent()
Asymptotic Ring <QQ^x * x^SR * log(x)^QQ> over Symbolic Ring

sage: C.<c> = AsymptoticRing(growth_group='QQ^c * c^QQ', coefficient_ring=QQ, default_prec=5
sage: (3 + 1/c^2)^c
3^c + 1/3*3^c*c^(-1) + 1/18*3^c*c^(-2) - 4/81*3^c*c^(-3)
- 35/1944*3^c*c^(-4) + O(3^c*c^(-5))
sage: _.parent()
Asymptotic Ring <QQ^c * c^QQ> over Rational Field
sage: (2 + (1/3)^c)^c
2^c + 1/2*(2/3)^c*c + 1/8*(2/9)^c*c^2 - 1/8*(2/9)^c*c
+ 1/48*(2/27)^c*c^3 + O((2/27)^c*c^2)
sage: _.parent()
Asymptotic Ring <QQ^c * c^QQ> over Rational Field
```

TESTS:

See trac ticket #19110:

```
sage: O(x)^(-1)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot take O(x) to exponent -1.
> *previous* ZeroDivisionError: rational division by zero

sage: B.<z> = AsymptoticRing(growth_group='z^QQ * log(z)^QQ', coefficient_ring=QQ, default_p
sage: z^(1+1/z)
z + log(z) + 1/2*z^(-1)*log(z)^2 + 1/6*z^(-2)*log(z)^3 +
1/24*z^(-3)*log(z)^4 + O(z^(-4)*log(z)^5)
sage: _.parent()
Asymptotic Ring <z^QQ * log(z)^QQ> over Rational Field

sage: B(0)^(-7)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot take 0 to the negative exponent -7.
```

```
sage: B(0)^SR.var('a')
Traceback (most recent call last):
...
NotImplementedError: Taking 0 to the exponent a not implemented.

sage: C.<s, t> = AsymptoticRing(growth_group='s^QQ * t^QQ', coefficient_ring=QQ)
sage: (s + t)^s
Traceback (most recent call last):
...
ValueError: Cannot take s + t to the exponent s.
> *previous* ValueError: Cannot determine main term of s + t
since there are several maximal elements s, t.
```

Check that trac ticket #19946 is fixed:

```
sage: A.<n> = AsymptoticRing('QQ^n * n^QQ', SR)
sage: e = 2^n; e
2^n
sage: e.parent()
Asymptotic Ring <SR^n * n^SR> over Symbolic Ring
sage: e = A(e); e
2^n
sage: e.parent()
Asymptotic Ring <QQ^n * n^QQ> over Symbolic Ring
```

rpow (*base*, *precision=None*)

Return the power of `base` to this asymptotic expansion.

INPUT:

- `base` – an element or `'e'`.
- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing('x^ZZ', QQ)
sage: (1/x).rpow('e', precision=5)
1 + x^(-1) + 1/2*x^(-2) + 1/6*x^(-3) + 1/24*x^(-4) + O(x^(-5))
```

TESTS:

```
sage: x.rpow(SR.var('y'))
Traceback (most recent call last):
...
ArithmeticError: Cannot construct y^x in Growth Group x^ZZ
> *previous* TypeError: unsupported operand parent(s) for '*':
'Growth Group x^ZZ' and 'Growth Group SR^x'
```

Check that trac ticket #19946 is fixed:

```
sage: A.<n> = AsymptoticRing('QQ^n * n^QQ', SR)
sage: n.rpow(2)
2^n
sage: _.parent()
Asymptotic Ring <QQ^n * n^SR> over Symbolic Ring
```

show()

Pretty-print this asymptotic expansion.

OUTPUT:

Nothing, the representation is printed directly on the screen.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing('QQ^x * x^QQ * log(x)^QQ', SR.subring(no_variables=True))
sage: (pi/2 * 5^x * x^(42/17) - sqrt(euler_gamma) * log(x)^(-7/8)).show()
<html><script type="math/tex">\newcommand{\Bold}[1]{\mathbf{#1}}\frac{1}{2} \, \pi
5^{x} x^{\frac{42}{17}} - \sqrt{\gamma_E} \log\left(x\right)^{-\frac{7}{8}}</script></html>
```

TESTS:

```
sage: A.<x> = AsymptoticRing('(e^x)^QQ * x^QQ', SR.subring(no_variables=True))
sage: (zeta(3) * (e^x)^(-1/2) * x^42).show()
<html><script type="math/tex">\newcommand{\Bold}[1]{\mathbf{#1}}\zeta(3)
\left(e^{x}\right)^{-\frac{1}{2}} x^{42}</script></html>
```

sqrt(*precision=None*)

Return the square root of this asymptotic expansion.

INPUT:

- `precision` – the precision used for truncating the expansion. If `None` (default value) is used, the default precision of the parent is used.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: A.<s> = AsymptoticRing(growth_group='s^QQ', coefficient_ring=QQ)
sage: s.sqrt()
s^(1/2)
sage: a = (1 + 1/s).sqrt(precision=6); a
1 + 1/2*s^(-1) - 1/8*s^(-2) + 1/16*s^(-3)
- 5/128*s^(-4) + 7/256*s^(-5) + O(s^(-6))
```

See also:

`pow()`, `rpow()`, `exp()`.

TESTS:

```
sage: P.<p> = PowerSeriesRing(QQ, default_prec=6)
sage: bool(SR(a.exact_part()).subs(s=1/x) -
....:      SR((1+p).sqrt().polynomial()).subs(p=x) == 0)
True
```

subs(*rules=None*, *domain=None*, ***kwds*)

Substitute the given `rules` in this asymptotic expansion.

INPUT:

- `rules` – a dictionary.
- `kwds` – keyword arguments will be added to the substitution `rules`.
- `domain` – (default: `None`) a parent. The neutral elements 0 and 1 (rules for the keys `'_zero_'` and `'_one_'`, see note box below) are taken out of this domain. If `None`, then this is determined automatically.

OUTPUT:

An object.

Note: The neutral element of the asymptotic ring is replaced by the value to the key `'_zero_'`; the neutral element of the growth group is replaced by the value to the key `'_one_'`.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing(growth_group='(e^x)^QQ * x^ZZ * log(x)^ZZ', coefficient_ring=QQ

sage: (e^x * x^2 + log(x)).subs(x=SR('s'))
s^2*e^s + log(s)
sage: _.parent()
Symbolic Ring

sage: (x^3 + x + log(x)).subs(x=x+5).truncate(5)
x^3 + 15*x^2 + 76*x + log(x) + 130 + O(x^(-1))
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Rational Field

sage: (e^x * x^2 + log(x)).subs(x=2*x)
4*(e^x)^2*x^2 + log(x) + log(2)
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^QQ * log(x)^QQ> over Symbolic Ring

sage: (x^2 + log(x)).subs(x=4*x+2).truncate(5)
16*x^2 + 16*x + log(x) + log(4) + 4 + 1/2*x^(-1) + O(x^(-2))
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Symbolic Ring

sage: (e^x * x^2 + log(x)).subs(x=RIF(pi))
229.534211738584?
sage: _.parent()
Real Interval Field with 53 bits of precision
```

See also:

`sage.symbolic.expression.Expression.subs()`

TESTS:

```
sage: x.subs({'y': -1})
Traceback (most recent call last):
...
ValueError: Cannot substitute y in x since it is not a generator of
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Rational Field.
sage: B.<u, v, w> = AsymptoticRing(growth_group='u^QQ * v^QQ * w^QQ', coefficient_ring=QQ)
sage: (1/u).subs({'u': 0})
Traceback (most recent call last):
...
TypeError: Cannot apply the substitution rules {u: 0} on u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
>> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Exact Term Monoid u^QQ * v^QQ * w^QQ with coefficients in Rational Field.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ * v^QQ * w^QQ.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
```

```
Growth Group u^QQ.
>...> *previous* ZeroDivisionError: rational division by zero
sage: (1/u).subs({'u': 0, 'v': SR.var('v')})
Traceback (most recent call last):
...
TypeError: Cannot apply the substitution rules {u: 0, v: v} on u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
>> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Exact Term Monoid u^QQ * v^QQ * w^QQ with coefficients in Rational Field.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ * v^QQ * w^QQ.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ.
>...> *previous* ZeroDivisionError: rational division by zero

sage: u.subs({u: 0, 'v': SR.var('v')})
0
sage: v.subs({u: 0, 'v': SR.var('v')})
v
sage: _.parent()
Symbolic Ring

sage: u.subs({SR.var('u'): -1})
Traceback (most recent call last):
...
TypeError: Cannot substitute u in u since it is neither an
asymptotic expansion nor a string
(but a <type 'sage.symbolic.expression.Expression'>).

sage: u.subs({u: 1, 'u': 1})
1
sage: u.subs({u: 1}, u=1)
1
sage: u.subs({u: 1, 'u': 2})
Traceback (most recent call last):
...
ValueError: Cannot substitute in u: duplicate key u.
sage: u.subs({u: 1}, u=3)
Traceback (most recent call last):
...
ValueError: Cannot substitute in u: duplicate key u.
```

substitute(*rules=None*, *domain=None*, ***kwds*)

Substitute the given `rules` in this asymptotic expansion.

INPUT:

- `rules` – a dictionary.
- `kwds` – keyword arguments will be added to the substitution `rules`.
- `domain` – (default: `None`) a parent. The neutral elements 0 and 1 (rules for the keys `'_zero_'` and `'_one_'`, see note box below) are taken out of this domain. If `None`, then this is determined automatically.

OUTPUT:

An object.

Note: The neutral element of the asymptotic ring is replaced by the value to the key `'_zero_'`; the neutral element of the growth group is replaced by the value to the key `'_one_'`.

EXAMPLES:

```
sage: A.<x> = AsymptoticRing(growth_group='(e^x)^QQ * x^ZZ * log(x)^ZZ', coefficient_ring=QQ

sage: (e^x * x^2 + log(x)).subs(x=SR('s'))
s^2*e^s + log(s)
sage: _.parent()
Symbolic Ring

sage: (x^3 + x + log(x)).subs(x=x+5).truncate(5)
x^3 + 15*x^2 + 76*x + log(x) + 130 + O(x^(-1))
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Rational Field

sage: (e^x * x^2 + log(x)).subs(x=2*x)
4*(e^x)^2*x^2 + log(x) + log(2)
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^QQ * log(x)^QQ> over Symbolic Ring

sage: (x^2 + log(x)).subs(x=4*x+2).truncate(5)
16*x^2 + 16*x + log(x) + log(4) + 4 + 1/2*x^(-1) + O(x^(-2))
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Symbolic Ring

sage: (e^x * x^2 + log(x)).subs(x=RIF(pi))
229.534211738584?
sage: _.parent()
Real Interval Field with 53 bits of precision
```

See also:

`sage.symbolic.expression.Expression.subs()`

TESTS:

```
sage: x.subs({'y': -1})
Traceback (most recent call last):
...
ValueError: Cannot substitute y in x since it is not a generator of
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Rational Field.
sage: B.<u, v, w> = AsymptoticRing(growth_group='u^QQ * v^QQ * w^QQ', coefficient_ring=QQ)
sage: (1/u).subs({'u': 0})
Traceback (most recent call last):
...
TypeError: Cannot apply the substitution rules {u: 0} on u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
>> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Exact Term Monoid u^QQ * v^QQ * w^QQ with coefficients in Rational Field.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ * v^QQ * w^QQ.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ.
>...> *previous* ZeroDivisionError: rational division by zero
sage: (1/u).subs({'u': 0, 'v': SR.var('v')})
```

```
Traceback (most recent call last):
...
TypeError: Cannot apply the substitution rules {u: 0, v: v} on u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Asymptotic Ring <u^QQ * v^QQ * w^QQ> over Rational Field.
>> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Exact Term Monoid u^QQ * v^QQ * w^QQ with coefficients in Rational Field.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ * v^QQ * w^QQ.
>...> *previous* ZeroDivisionError: Cannot substitute in u^(-1) in
Growth Group u^QQ.
>...> *previous* ZeroDivisionError: rational division by zero

sage: u.subs({u: 0, 'v': SR.var('v')})
0
sage: v.subs({u: 0, 'v': SR.var('v')})
v
sage: _.parent()
Symbolic Ring

sage: u.subs({SR.var('u'): -1})
Traceback (most recent call last):
...
TypeError: Cannot substitute u in u since it is neither an
asymptotic expansion nor a string
(but a <type 'sage.symbolic.expression.Expression'>).

sage: u.subs({u: 1, 'u': 1})
1
sage: u.subs({u: 1}, u=1)
1
sage: u.subs({u: 1, 'u': 2})
Traceback (most recent call last):
...
ValueError: Cannot substitute in u: duplicate key u.
sage: u.subs({u: 1}, u=3)
Traceback (most recent call last):
...
ValueError: Cannot substitute in u: duplicate key u.
```

summands

The summands of this asymptotic expansion stored in the underlying data structure (a `MutablePoset`).

EXAMPLES:

```
sage: R.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: expr = 7*x^12 + x^5 + O(x^3)
sage: expr.summands
poset(O(x^3), x^5, 7*x^12)
```

See also:

`sage.data_structures.mutable_poset.MutablePoset`

symbolic_expression(*R=None*)

Return this asymptotic expansion as a symbolic expression.

INPUT:

- R – (a subring of) the symbolic ring or None. The output is will be an element of R. If None, then the symbolic ring is used.

OUTPUT:

A symbolic expression.

EXAMPLES:

```
sage: A.<x, y, z> = AsymptoticRing(growth_group='x^ZZ * y^QQ * log(y)^QQ * QQ^z * z^QQ', coe
sage: SR(A.an_element())  # indirect doctest
1/8*(1/8)^z*x^3*y^(3/2)*z^(3/2)*log(y)^(3/2) +
Order((1/2)^z*x*sqrt(y)*sqrt(z)*sqrt(log(y)))
```

TESTS:

```
sage: a = A.an_element(); a
1/8*x^3*y^(3/2)*log(y)^(3/2)*(1/8)^z*z^(3/2) +
O(x*y^(1/2)*log(y)^(1/2)*(1/2)^z*z^(1/2))
sage: a.symbolic_expression()
1/8*(1/8)^z*x^3*y^(3/2)*z^(3/2)*log(y)^(3/2) +
Order((1/2)^z*x*sqrt(y)*sqrt(z)*sqrt(log(y)))
sage: _.parent()
Symbolic Ring

sage: from sage.symbolic.ring import SymbolicRing
sage: class MySymbolicRing(SymbolicRing):
....:     pass
sage: mySR = MySymbolicRing()
sage: a.symbolic_expression(mySR).parent() is mySR
True
```

truncate (*precision=None*)

Truncate this asymptotic expansion.

INPUT:

- precision – a positive integer or None. Number of summands that are kept. If None (default value) is given, then default_prec from the parent is used.

OUTPUT:

An asymptotic expansion.

Note: For example, truncating an asymptotic expansion with precision=20 does not yield an expansion with exactly 20 summands! Rather than that, it keeps the 20 summands with the largest growth, and adds appropriate O-Terms.

EXAMPLES:

```
sage: R.<x> = AsymptoticRing('x^ZZ', QQ)
sage: ex = sum(x^k for k in range(5)); ex
x^4 + x^3 + x^2 + x + 1
sage: ex.truncate(precision=2)
x^4 + x^3 + O(x^2)
sage: ex.truncate(precision=0)
O(x^4)
sage: ex.truncate()
x^4 + x^3 + x^2 + x + 1
```

variable_names ()

Return the names of the variables of this asymptotic expansion.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: A.<m, n> = AsymptoticRing('QQ^m * m^QQ * n^ZZ * log(n)^ZZ', QQ)
sage: (4*2^m*m^4*log(n)).variable_names()
('m', 'n')
sage: (4*2^m*m^4).variable_names()
('m',)
sage: (4*log(n)).variable_names()
('n',)
sage: (4*m^3).variable_names()
('m',)
sage: (4*m^0).variable_names()
()
sage: (4*2^m*m^4 + log(n)).variable_names()
('m', 'n')
sage: (2^m + m^4 + log(n)).variable_names()
('m', 'n')
sage: (2^m + m^4).variable_names()
('m',)
```

class `sage.rings.asymptotic.asymptotic_ring.`**`AsymptoticRing`**(*growth_group*, *coefficient_ring*, *category*, *default_prec*)

Bases: `sage.rings.ring.Algebra`, `sage.structure.unique_representation.UniqueRepresentation`

A ring consisting of `asymptotic expansions`.

INPUT:

- `growth_group` – either a partially ordered group (see *(Asymptotic) Growth Groups*) or a string describing such a growth group (see `GrowthGroupFactory`).
- `coefficient_ring` – the ring which contains the coefficients of the expansions.
- `default_prec` – a positive integer. This is the number of summands that are kept before truncating an infinite series.
- `category` – the category of the parent can be specified in order to broaden the base structure. It has to be a subcategory of `Category of rings`. This is also the default category if `None` is specified.

EXAMPLES:

We begin with the construction of an asymptotic ring in various ways. First, we simply pass a string specifying the underlying growth group:

```
sage: R1_x.<x> = AsymptoticRing(growth_group='x^QQ', coefficient_ring=QQ); R1_x
Asymptotic Ring <x^QQ> over Rational Field
sage: x
x
```

This is equivalent to the following code, which explicitly specifies the underlying growth group:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G_QQ = GrowthGroup('x^QQ')
sage: R2_x.<x> = AsymptoticRing(growth_group=G_QQ, coefficient_ring=QQ); R2_x
Asymptotic Ring <x^QQ> over Rational Field
```

Of course, the coefficient ring of the asymptotic ring and the base ring of the underlying growth group do not need to coincide:

```
sage: R_ZZ_x.<x> = AsymptoticRing(growth_group='x^QQ', coefficient_ring=ZZ); R_ZZ_x
Asymptotic Ring <x^QQ> over Integer Ring
```

Note, we can also create and use logarithmic growth groups:

```
sage: R_log = AsymptoticRing(growth_group='log(x)^ZZ', coefficient_ring=QQ); R_log
Asymptotic Ring <log(x)^ZZ> over Rational Field
```

Other growth groups are available. See *Asymptotic Ring* for more examples.

Below there are some technical details.

According to the conventions for parents, uniqueness is ensured:

```
sage: R1_x is R2_x
True
```

Furthermore, the coercion framework is also involved. Coercion between two asymptotic rings is possible (given that the underlying growth groups and coefficient rings are chosen appropriately):

```
sage: R1_x.has_coerce_map_from(R_ZZ_x)
True
```

Additionally, for the sake of convenience, the coefficient ring also coerces into the asymptotic ring (representing constant quantities):

```
sage: R1_x.has_coerce_map_from(QQ)
True
```

TESTS:

```
sage: from sage.rings.asymptotic.asymptotic_ring import AsymptoticRing as AR_class
sage: class AR(AR_class):
....:     class Element(AR_class.Element):
....:         __eq__ = AR_class.Element.has_same_summands
sage: A = AR(growth_group='z^QQ', coefficient_ring=QQ)
sage: from itertools import islice
sage: TestSuite(A).run(  # not tested  # long time  # see #19424
....:     verbose=True,
....:     elements=tuple(islice(A.some_elements(), 10)),
....:     skip=('_test_some_elements',  # to many elements
....:           '_test_distributivity'))  # due to cancellations: O(z) != O(z^2)
```

Element

alias of `AsymptoticExpansion`

change_parameter(***kwds*)

Return an asymptotic ring with a change in one or more of the given parameters.

INPUT:

- `growth_group` – (default: `None`) the new growth group.
- `coefficient_ring` – (default: `None`) the new coefficient ring.
- `category` – (default: `None`) the new category.
- `default_prec` – (default: `None`) the new default precision.

OUTPUT:

An asymptotic ring.

EXAMPLES:

```
sage: A = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: A.change_parameter(coefficient_ring=QQ)
Asymptotic Ring <x^ZZ> over Rational Field
```

TESTS:

```
sage: A.change_parameter(coefficient_ring=ZZ) is A
True
sage: A.change_parameter(coefficient_ring=None) is A
True
```

coefficient_ring

The coefficient ring of this asymptotic ring.

EXAMPLES:

```
sage: AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.coefficient_ring
Integer Ring
```

coefficients_of_generating_function(*function*, *singularities*, *precision=None*, *return_singular_expansions=False*)

Return the asymptotic growth of the coefficients of some generating function by means of Singularity Analysis.

INPUT:

- `function` – a callable function in one variable.
- `singularities` – list of dominant singularities of the function.
- `precision` – (default: `None`) an integer. If `None`, then the default precision of the asymptotic ring is used.
- `return_singular_expansions` – (default: `False`) a boolean. If set, the singular expansions are also returned.

OUTPUT:

- If `return_singular_expansions=False`: An asymptotic expansion from this ring.
- If `return_singular_expansions=True`: A named tuple with components `asymptotic_expansion` and `singular_expansions`. The former contains an asymptotic expansion from this ring, the latter is a dictionary which contains the singular expansions around the singularities.

Todo

Make this method more usable by implementing the processing of symbolic expressions.

EXAMPLES:

Catalan numbers:

```
sage: def catalan(z):
....:     return (1-(1-4*z)^(1/2))/(2*z)
sage: B.<n> = AsymptoticRing('QQ^n * n^QQ', QQ)
sage: B.coefficients_of_generating_function(catalan, (1/4,), precision=3)
1/sqrt(pi)*4^n*n^(-3/2) - 9/8/sqrt(pi)*4^n*n^(-5/2)
+ 145/128/sqrt(pi)*4^n*n^(-7/2) + O(4^n*n^(-4))
sage: B.coefficients_of_generating_function(catalan, (1/4,), precision=2,
```

```
....:                              return_singular_expansions=True)
SingularityAnalysisResult(asymptotic_expansion=1/sqrt(pi)*4^n*n^(-3/2)
- 9/8/sqrt(pi)*4^n*n^(-5/2) + O(4^n*n^(-3)),
singular_expansions={1/4: 2 - 2*T^(-1/2)
+ 2*T^(-1) - 2*T^(-3/2) + O(T^(-2))})
```

Unit fractions:

```
sage: def logarithmic(z):
....:     return -log(1-z)
sage: B.coefficients_of_generating_function(logarithmic, (1,), precision=5)
n^(-1) + O(n^(-3))
```

Harmonic numbers:

```
sage: def harmonic(z):
....:     return -log(1-z)/(1-z)
sage: B.<n> = AsymptoticRing('QQ^n * n^QQ * log(n)^QQ', QQ)
sage: ex = B.coefficients_of_generating_function(harmonic, (1,), precision=13); ex
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4)
+ O(n^(-6))
sage: ex.has_same_summands(asymptotic_expansions.HarmonicNumber(
....:    'n', precision=5))
True
```

Warning: Once singular expansions around points other than infinity are implemented (trac ticket #20050), the output in the case `return_singular_expansions` will change to return singular expansions around the singularities.

TESTS:

```
sage: def f(z):
....:     return z/(1-z)
sage: B.coefficients_of_generating_function(f, (1,), precision=3)
Traceback (most recent call last):
...
NotImplementedOZero: The error term in the result is O(0)
which means 0 for sufficiently large n.
```

construction()

Return the construction of this asymptotic ring.

OUTPUT:

A pair whose first entry is an `asymptotic ring construction functor` and its second entry the coefficient ring.

EXAMPLES:

```
sage: A = AsymptoticRing(growth_group='x^ZZ * QQ^y', coefficient_ring=QQ)
sage: A.construction()
(AsymptoticRing<x^ZZ * QQ^y>, Rational Field)
```

See also:

Asymptotic Ring, `AsymptoticRing`, `AsymptoticRingFunctor`.

create_summand(*type*, *data=None*, ***kwds*)

Create a simple asymptotic expansion consisting of a single summand.

INPUT:

- `type` – 'O' or 'exact'.
- `data` – the element out of which a summand has to be created.
- `growth` – an element of the `growth_group()`.
- `coefficient` – an element of the `coefficient_ring()`.

Note: Either `growth` and `coefficient` or `data` have to be specified.

OUTPUT:

An asymptotic expansion.

Note: This method calls the factory `TermMonoid` with the appropriate arguments.

EXAMPLES:

```
sage: R = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: R.create_summand('O', x^2)
O(x^2)
sage: R.create_summand('exact', growth=x^456, coefficient=123)
123*x^456
sage: R.create_summand('exact', data=12*x^13)
12*x^13
```

TESTS:

```
sage: R.create_summand('exact', data='12*x^13')
12*x^13
sage: R.create_summand('exact', data='x^13 * 12')
12*x^13
sage: R.create_summand('exact', data='x^13')
x^13
sage: R.create_summand('exact', data='12')
12
sage: R.create_summand('exact', data=12)
12

sage: Z = R.change_parameter(coefficient_ring=Zmod(3))
sage: Z.create_summand('exact', data=42)
0

sage: R.create_summand('O', growth=42*x^2, coefficient=1)
Traceback (most recent call last):
...
ValueError: Growth 42*x^2 is not in O-Term Monoid x^ZZ with implicit coefficients in Integer
> *previous* ValueError: 42*x^2 is not in Growth Group x^ZZ.

sage: AR.<z> = AsymptoticRing('z^QQ', QQ)
sage: AR.create_summand('exact', growth='z^2')
Traceback (most recent call last):
...
TypeError: Cannot create exact term: only 'growth' but
no 'coefficient' specified.
```

default_prec

The default precision of this asymptotic ring.

This is the parameter used to determine how many summands are kept before truncating an infinite series (which occur when inverting asymptotic expansions).

EXAMPLES:

```
sage: AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.default_prec
20
sage: AR = AsymptoticRing('x^ZZ', ZZ, default_prec=123)
sage: AR.default_prec
123
```

gen (*n=0*)

Return the n-th generator of this asymptotic ring.

INPUT:

- n – (default: 0) a non-negative integer.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: R.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: R.gen()
x
```

gens ()

Return a tuple with generators of this asymptotic ring.

INPUT:

Nothing.

OUTPUT:

A tuple of asymptotic expansions.

Note: Generators do not necessarily exist. This depends on the underlying growth group. For example, monomial growth groups have a generator, and exponential growth groups do not.

EXAMPLES:

```
sage: AR.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.gens()
(x,)
sage: B.<y,z> = AsymptoticRing(growth_group='y^ZZ * z^ZZ', coefficient_ring=QQ)
sage: B.gens()
(y, z)
```

growth_group

The growth group of this asymptotic ring.

EXAMPLES:

```
sage: AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.growth_group
Growth Group x^ZZ
```

See also:

(Asymptotic) Growth Groups

ngens()

Return the number of generators of this asymptotic ring.

INPUT:

Nothing.

OUTPUT:

An integer.

EXAMPLES:

```
sage: AR.<x> = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.ngens()
1
```

some_elements()

Return some elements of this term monoid.

See `TestSuite` for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from itertools import islice
sage: A = AsymptoticRing(growth_group='z^QQ', coefficient_ring=ZZ)
sage: tuple(islice(A.some_elements(), 10))
(z^(3/2) + O(z^(1/2)),
 O(z^(1/2)),
 z^(3/2) + O(z^(-1/2)),
 -z^(3/2) + O(z^(1/2)),
 O(z^(-1/2)),
 O(z^2),
 z^6 + O(z^(1/2)),
 -z^(3/2) + O(z^(-1/2)),
 O(z^2),
 z^(3/2) + O(z^(-2)))
```

variable_names()

Return the names of the variables.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: A = AsymptoticRing(growth_group='x^ZZ * QQ^y', coefficient_ring=QQ)
sage: A.variable_names()
('x', 'y')
```

class `sage.rings.asymptotic.asymptotic_ring.`**AsymptoticRingFunctor**(*growth_group*)

Bases: `sage.categories.pushout.ConstructionFunctor`

A `construction functor` for `asymptotic rings`.

INPUT:

- growth_group – a partially ordered group (see AsymptoticRing or *(Asymptotic) Growth Groups* for details).

EXAMPLES:

```
sage: AsymptoticRing(growth_group='x^ZZ', coefficient_ring=QQ).construction()  # indirect doctes
(AsymptoticRing<x^ZZ>, Rational Field)
```

See also:

Asymptotic Ring, AsymptoticRing, sage.rings.asymptotic.growth_group.AbstractGrowthGroupFuncto
sage.rings.asymptotic.growth_group.ExponentialGrowthGroupFunctor,
sage.rings.asymptotic.growth_group.MonomialGrowthGroupFunctor,
sage.categories.pushout.ConstructionFunctor.

TESTS:

```
sage: X = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=QQ)
sage: Y = AsymptoticRing(growth_group='y^ZZ', coefficient_ring=QQ)
sage: cm = sage.structure.element.get_coercion_model()
sage: cm.record_exceptions()
sage: cm.common_parent(X, Y)
Asymptotic Ring <x^ZZ * y^ZZ> over Rational Field
sage: sage.structure.element.coercion_traceback()  # not tested

sage: from sage.categories.pushout import pushout
sage: pushout(AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ), QQ)
Asymptotic Ring <x^ZZ> over Rational Field
```

merge (*other*)

Merge this functor with other if possible.

INPUT:

- other – a functor.

OUTPUT:

A functor or None.

EXAMPLES:

```
sage: X = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=QQ)
sage: Y = AsymptoticRing(growth_group='y^ZZ', coefficient_ring=QQ)
sage: F_X = X.construction()[0]
sage: F_Y = Y.construction()[0]
sage: F_X.merge(F_X)
AsymptoticRing<x^ZZ>
sage: F_X.merge(F_Y)
AsymptoticRing<x^ZZ * y^ZZ>
```

exception sage.rings.asymptotic.asymptotic_ring.**NoConvergenceError**

Bases: exceptions.RuntimeError

A special RuntimeError which is raised when an algorithm does not converge/stop.

4.2 Common Asymptotic Expansions

Warning: As this code is experimental, a warning is thrown when an asymptotic ring (or an associated structure) is created for the first time in a session (see sage.misc.superseded.experimental).
TESTS:

```
sage: AsymptoticRing(growth_group='z^ZZ * log(z)^QQ', coefficient_ring=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.
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.
Asymptotic Ring <z^ZZ * log(z)^QQ> over Integer Ring
```

Asymptotic expansions in SageMath can be built through the asymptotic_expansions object. It contains generators for common asymptotic expressions. For example,

```
sage: asymptotic_expansions.Stirling('n', precision=5)
sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(1/2) +
1/12*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-1/2) +
1/288*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-3/2) +
O(e^(n*log(n))*(e^n)^(-1)*n^(-5/2))
```

generates the first 5 summands of Stirling's approximation formula for factorials.

To construct an asymptotic expression manually, you can use the class AsymptoticRing. See *asymptotic ring* for more details and a lot of examples.

Asymptotic Expansions

HarmonicNumber()	harmonic numbers
Stirling()	Stirling's approximation formula for factorials
log_Stirling()	the logarithm of Stirling's approximation formula for factorials
Binomial_kn_over_n()	an asymptotic expansion of the binomial coefficient
SingularityAnalysis()	an asymptotic expansion obtained by singularity analysis

AUTHORS:

- Daniel Krenn (2015)
- Clemens Heuberger (2016)
- Benjamin Hackl (2016)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.

4.2.1 Classes and Methods

class sage.rings.asymptotic.asymptotic_expansion_generators.**AsymptoticExpansionGenerators**
    Bases: sage.structure.sage_object.SageObject

A collection of constructors for several common asymptotic expansions.

A list of all asymptotic expansions in this database is available via tab completion. Type "`asymptotic_expansions.`" and then hit tab to see which expansions are available.

The asymptotic expansions currently in this class include:

- `HarmonicNumber()`
- `Stirling()`
- `log_Stirling()`
- `Binomial_kn_over_n()`
- `SingularityAnalysis()`

static `Binomial_kn_over_n`(*var*, *k*, *precision=None*, *skip_constant_factor=False*)

Return the asymptotic expansion of the binomial coefficient kn choose n.

INPUT:

- `var` – a string for the variable name.
- `k` – a number or symbolic constant.
- `precision` – (default: `None`) an integer. If `None`, then the default precision of the asymptotic ring is used.
- `skip_constant_factor` – (default: `False`) a boolean. If set, then the constant factor $\sqrt{k/(2\pi(k-1))}$ is left out. As a consequence, the coefficient ring of the output changes from `Symbolic Constants Subring` (if `False`) to `Rational Field` (if `True`).

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: asymptotic_expansions.Binomial_kn_over_n('n', k=2, precision=3)
1/sqrt(pi)*4^n*n^(-1/2)
- 1/8/sqrt(pi)*4^n*n^(-3/2)
+ 1/128/sqrt(pi)*4^n*n^(-5/2)
+ O(4^n*n^(-7/2))
sage: _.parent()
Asymptotic Ring <QQ^n * n^QQ> over Symbolic Constants Subring

sage: asymptotic_expansions.Binomial_kn_over_n('n', k=3, precision=3)
1/2*sqrt(3)/sqrt(pi)*(27/4)^n*n^(-1/2)
- 7/144*sqrt(3)/sqrt(pi)*(27/4)^n*n^(-3/2)
+ 49/20736*sqrt(3)/sqrt(pi)*(27/4)^n*n^(-5/2)
+ O((27/4)^n*n^(-7/2))

sage: asymptotic_expansions.Binomial_kn_over_n('n', k=7/5, precision=3)
1/2*sqrt(7)/sqrt(pi)*(7/10*7^(2/5)*2^(3/5))^n*n^(-1/2)
- 13/112*sqrt(7)/sqrt(pi)*(7/10*7^(2/5)*2^(3/5))^n*n^(-3/2)
+ 169/12544*sqrt(7)/sqrt(pi)*(7/10*7^(2/5)*2^(3/5))^n*n^(-5/2)
+ O((7/10*7^(2/5)*2^(3/5))^n*n^(-7/2))
sage: _.parent()
Asymptotic Ring <(Symbolic Constants Subring)^n * n^QQ>
over Symbolic Constants Subring
```

TESTS:

```
sage: expansion = asymptotic_expansions.Binomial_kn_over_n('n', k=7/5, precision=3)
sage: n = expansion.parent().gen()
sage: expansion.compare_with_values(n, lambda x: binomial(7/5*x, x), [5, 10, 20])  # rel tol
[(5, -0.0287383845047?), (10, -0.030845971026?), (20, -0.03162833549?)]
sage: asymptotic_expansions.Binomial_kn_over_n(
....:     'n', k=5, precision=3, skip_constant_factor=True)
(3125/256)^n*n^(-1/2)
- 7/80*(3125/256)^n*n^(-3/2)
+ 49/12800*(3125/256)^n*n^(-5/2)
+ O((3125/256)^n*n^(-7/2))
sage: _.parent()
Asymptotic Ring <QQ^n * n^QQ> over Rational Field
sage: asymptotic_expansions.Binomial_kn_over_n(
....:     'n', k=4, precision=1, skip_constant_factor=True)
(256/27)^n*n^(-1/2) + O((256/27)^n*n^(-3/2))

sage: S = asymptotic_expansions.Stirling('n', precision=5)
sage: n = S.parent().gen()
sage: all(  # long time
....:     SR(asymptotic_expansions.Binomial_kn_over_n(
....:         'n', k=k, precision=3)).canonicalize_radical() ==
....:     SR(S.subs(n=k*n) / (S.subs(n=(k-1)*n) * S)).canonicalize_radical()
....:     for k in [2, 3, 4])
True
```

static **`HarmonicNumber`**(*var*, *precision=None*, *skip_constant_summand=False*)

Return the asymptotic expansion of a harmonic number.

INPUT:

- `var` – a string for the variable name.
- `precision` – (default: `None`) an integer. If `None`, then the default precision of the asymptotic ring is used.
- `skip_constant_summand` – (default: `False`) a boolean. If set, then the constant summand `euler_gamma` is left out. As a consequence, the coefficient ring of the output changes from `Symbolic Constants Subring` (if `False`) to `Rational Field` (if `True`).

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: asymptotic_expansions.HarmonicNumber('n', precision=5)
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4) + O(n^(-6))
```

TESTS:

```
sage: ex = asymptotic_expansions.HarmonicNumber('n', precision=5)
sage: n = ex.parent().gen()
sage: ex.compare_with_values(n,                              # rel tol 1e-6
....:      lambda x: sum(1/k for k in srange(1, x+1)), [5, 10, 20])
[(5, 0.0038125360?), (10, 0.00392733?), (20, 0.0039579?)]
sage: asymptotic_expansions.HarmonicNumber('n')
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4)
- 1/252*n^(-6) + 1/240*n^(-8) - 1/132*n^(-10)
+ 691/32760*n^(-12) - 1/12*n^(-14) + 3617/8160*n^(-16)
- 43867/14364*n^(-18) + 174611/6600*n^(-20) - 77683/276*n^(-22)
+ 236364091/65520*n^(-24) - 657931/12*n^(-26)
```

```
+ 3392780147/3480*n^(-28) - 1723168255201/85932*n^(-30)
+ 7709321041217/16320*n^(-32)
- 151628697551/12*n^(-34) + O(n^(-36))
sage: _.parent()
Asymptotic Ring <n^ZZ * log(n)^ZZ> over Symbolic Constants Subring

sage: asymptotic_expansions.HarmonicNumber(
....:     'n', precision=5, skip_constant_summand=True)
log(n) + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4) + O(n^(-6))
sage: _.parent()
Asymptotic Ring <n^ZZ * log(n)^ZZ> over Rational Field
sage: for p in range(5):
....:     print asymptotic_expansions.HarmonicNumber(
....:         'n', precision=p)
O(log(n))
log(n) + O(1)
log(n) + euler_gamma + O(n^(-1))
log(n) + euler_gamma + 1/2*n^(-1) + O(n^(-2))
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + O(n^(-4))
sage: asymptotic_expansions.HarmonicNumber('m', precision=5)
log(m) + euler_gamma + 1/2*m^(-1) - 1/12*m^(-2) + 1/120*m^(-4) + O(m^(-6))
```

static SingularityAnalysis(*var*, *zeta=1*, *alpha=0*, *beta=0*, *delta=0*, *precision=None*, *normalized=True*)

Return the asymptotic expansion of the coefficients of an power series with specified pole and logarithmic singularity.

More precisely, this extracts the n-th coefficient

$$[z^n]\left(\frac{1}{1-z/\zeta}\right)^{\alpha}\left(\frac{1}{z/\zeta}\log\frac{1}{1-z/\zeta}\right)^{\beta}\left(\frac{1}{z/\zeta}\log\left(\frac{1}{z/\zeta}\log\frac{1}{1-z/\zeta}\right)\right)^{\delta}$$

(if `normalized=True`, the default) or

$$[z^n]\left(\frac{1}{1-z/\zeta}\right)^{\alpha}\left(\log\frac{1}{1-z/\zeta}\right)^{\beta}\left(\log\left(\frac{1}{z/\zeta}\log\frac{1}{1-z/\zeta}\right)\right)^{\delta}$$

(if `normalized=False`).

INPUT:

- `var` – a string for the variable name.
- `zeta` – (default: 1) the location of the singularity.
- `alpha` – (default: 0) the pole order of the singularty.
- `beta` – (default: 0) the order of the logarithmic singularity.
- `delta` – (default: 0) the order of the log-log singularity. Not yet implemented for `delta != 0`.
- `precision` – (default: `None`) an integer. If `None`, then the default precision of the asymptotic ring is used.
- `normalized` – (default: `True`) a boolean, see above.

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: asymptotic_expansions.SingularityAnalysis('n', alpha=1)
1
sage: asymptotic_expansions.SingularityAnalysis('n', alpha=2)
n + 1
sage: asymptotic_expansions.SingularityAnalysis('n', alpha=3)
1/2*n^2 + 3/2*n + 1
sage: _.parent()
Asymptotic Ring <n^ZZ> over Rational Field

sage: asymptotic_expansions.SingularityAnalysis('n', alpha=-3/2,
....:     precision=3)
3/4/sqrt(pi)*n^(-5/2)
+ 45/32/sqrt(pi)*n^(-7/2)
+ 1155/512/sqrt(pi)*n^(-9/2)
+ O(n^(-11/2))
sage: asymptotic_expansions.SingularityAnalysis('n', alpha=-1/2,
....:     precision=3)
-1/2/sqrt(pi)*n^(-3/2)
- 3/16/sqrt(pi)*n^(-5/2)
- 25/256/sqrt(pi)*n^(-7/2)
+ O(n^(-9/2))
sage: asymptotic_expansions.SingularityAnalysis('n', alpha=1/2,
....:     precision=4)
1/sqrt(pi)*n^(-1/2)
- 1/8/sqrt(pi)*n^(-3/2)
+ 1/128/sqrt(pi)*n^(-5/2)
+ 5/1024/sqrt(pi)*n^(-7/2)
+ O(n^(-9/2))
sage: _.parent()
Asymptotic Ring <n^QQ> over Symbolic Constants Subring

sage: S = SR.subring(rejecting_variables=('n',))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=S.var('a'),
....:     precision=4).map_coefficients(lambda c: c.factor())
1/gamma(a)*n^(a - 1)
+ (1/2*(a - 1)*a/gamma(a))*n^(a - 2)
+ (1/24*(3*a - 1)*(a - 1)*(a - 2)*a/gamma(a))*n^(a - 3)
+ (1/48*(a - 1)^2*(a - 2)*(a - 3)*a^2/gamma(a))*n^(a - 4)
+ O(n^(a - 5))
sage: _.parent()
Asymptotic Ring <n^(Symbolic Subring rejecting the variable n)>
over Symbolic Subring rejecting the variable n

sage: ae = asymptotic_expansions.SingularityAnalysis('n',
....:          alpha=1/2, beta=1, precision=4); ae
1/sqrt(pi)*n^(-1/2)*log(n) + ((euler_gamma + 2*log(2))/sqrt(pi))*n^(-1/2)
- 5/8/sqrt(pi)*n^(-3/2)*log(n) + (1/8*(3*euler_gamma + 6*log(2) - 8)/sqrt(pi)
- (euler_gamma + 2*log(2) - 2)/sqrt(pi))*n^(-3/2) + O(n^(-5/2)*log(n))
sage: n = ae.parent().gen()
sage: ae.subs(n=n-1).map_coefficients(lambda x: x.canonicalize_radical())
1/sqrt(pi)*n^(-1/2)*log(n)
+ ((euler_gamma + 2*log(2))/sqrt(pi))*n^(-1/2)
- 1/8/sqrt(pi)*n^(-3/2)*log(n)
+ (-1/8*(euler_gamma + 2*log(2))/sqrt(pi))*n^(-3/2)
+ O(n^(-5/2)*log(n))
```

```
sage: asymptotic_expansions.SingularityAnalysis('n',
....:     alpha=1, beta=1/2, precision=4)
log(n)^(1/2) + 1/2*euler_gamma*log(n)^(-1/2)
+ (-1/8*euler_gamma^2 + 1/48*pi^2)*log(n)^(-3/2)
+ (1/16*euler_gamma^3 - 1/32*euler_gamma*pi^2 + 1/8*zeta(3))*log(n)^(-5/2)
+ O(log(n)^(-7/2))

sage: ae = asymptotic_expansions.SingularityAnalysis('n',
....:     alpha=0, beta=2, precision=14)
sage: n = ae.parent().gen()
sage: ae.subs(n=n-2)
2*n^(-1)*log(n) + 2*euler_gamma*n^(-1) - n^(-2) - 1/6*n^(-3) + O(n^(-5))

sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=-1/2, beta=1, precision=2, normalized=False)
-1/2/sqrt(pi)*n^(-3/2)*log(n)
+ (-1/2*(euler_gamma + 2*log(2) - 2)/sqrt(pi))*n^(-3/2)
+ O(n^(-5/2)*log(n))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1/2, alpha=0, beta=1, precision=3, normalized=False)
2^n*n^(-1) + O(2^n*n^(-2))
```

ALGORITHM:

See [FS2009] together with the errata list.

REFERENCES:

TESTS:

```
sage: ex = asymptotic_expansions.SingularityAnalysis('n', alpha=-1/2,
....:     precision=4)
sage: n = ex.parent().gen()
sage: coefficients = ((1-x)^(1/2)).series(
....:     x, 21).truncate().coefficients(x, sparse=False)
sage: ex.compare_with_values(n,    # rel tol 1e-6
....:     lambda k: coefficients[k], [5, 10, 20])
[(5, 0.015778873294?), (10, 0.01498952777?), (20, 0.0146264622?)]
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=3, precision=2)
1/2*n^2 + 3/2*n + O(1)
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=3, precision=3)
1/2*n^2 + 3/2*n + 1
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=3, precision=4)
1/2*n^2 + 3/2*n + 1

sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=0)
Traceback (most recent call last):
...
NotImplementedOZero: The error term in the result is O(0)
which means 0 for sufficiently large n.
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=-1)
Traceback (most recent call last):
...
NotImplementedOZero: The error term in the result is O(0)
which means 0 for sufficiently large n.
```

```
sage: asymptotic_expansions.SingularityAnalysis(
....:     'm', alpha=-1/2, precision=3)
-1/2/sqrt(pi)*m^(-3/2)
- 3/16/sqrt(pi)*m^(-5/2)
- 25/256/sqrt(pi)*m^(-7/2)
+ O(m^(-9/2))
sage: _.parent()
Asymptotic Ring <m^QQ> over Symbolic Constants Subring
```

Location of the singularity:

```
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=1, zeta=2, precision=3)
(1/2)^n
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=1, zeta=1/2, precision=3)
2^n
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=1, zeta=CyclotomicField(3).gen(),
....:      precision=3)
(-zeta3 - 1)^n
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=4, zeta=2, precision=3)
1/6*(1/2)^n*n^3 + (1/2)^n*n^2 + 11/6*(1/2)^n*n + O((1/2)^n)
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=-1, zeta=2, precision=3)
Traceback (most recent call last):
...
NotImplementedOZero: The error term in the result is O(0)
which means 0 for sufficiently large n.
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=1/2, zeta=2, precision=3)
1/sqrt(pi)*(1/2)^n*n^(-1/2) - 1/8/sqrt(pi)*(1/2)^n*n^(-3/2)
+ 1/128/sqrt(pi)*(1/2)^n*n^(-5/2) + O((1/2)^n*n^(-7/2))
```

The following tests correspond to Table VI.5 in [FS2009].

```
sage: A.<n> = AsymptoticRing('n^QQ * log(n)^QQ', QQ)
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=-1/2, beta=1, precision=2,
....:     normalized=False) * (- sqrt(pi*n^3))
1/2*log(n) + 1/2*euler_gamma + log(2) - 1 + O(n^(-1)*log(n))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=0, beta=1, precision=3,
....:     normalized=False)
n^(-1) + O(n^(-2))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=0, beta=2,  precision=14,
....:     normalized=False) * n
2*log(n) + 2*euler_gamma - n^(-1) - 1/6*n^(-2) +  O(n^(-4))
sage: (asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=1/2, beta=1, precision=4,
....:     normalized=False) * sqrt(pi*n)).\
....:     map_coefficients(lambda x: x.expand())
log(n) + euler_gamma + 2*log(2) - 1/8*n^(-1)*log(n) +
(-1/8*euler_gamma - 1/4*log(2))*n^(-1) + O(n^(-2)*log(n))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=1, beta=1, precision=13,
```

```
....:     normalized=False)
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4)
+ O(n^(-6))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=1, beta=2, precision=4,
....:     normalized=False)
log(n)^2 + 2*euler_gamma*log(n) + euler_gamma^2 - 1/6*pi^2
+ O(n^(-1)*log(n))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=3/2, beta=1, precision=3,
....:     normalized=False) * sqrt(pi/n)
2*log(n) + 2*euler_gamma + 4*log(2) - 4 + 3/4*n^(-1)*log(n)
+ O(n^(-1))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=2, beta=1, precision=5,
....:     normalized=False)
n*log(n) + (euler_gamma - 1)*n + log(n) + euler_gamma + 1/2
+ O(n^(-1))
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, alpha=2, beta=2, precision=4,
....:     normalized=False) / n
log(n)^2 + (2*euler_gamma - 2)*log(n)
- 2*euler_gamma + euler_gamma^2 - 1/6*pi^2 + 2
+ n^(-1)*log(n)^2 + O(n^(-1)*log(n))
```

Be aware that the last result does *not* coincide with [FS2009], they do have a different error term.

Checking parameters:

```
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, 1, 1/2, precision=0, normalized=False)
Traceback (most recent call last):
...
ValueError: beta and delta must be integers
sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', 1, 1, 1, 1/2, normalized=False)
Traceback (most recent call last):
...
ValueError: beta and delta must be integers

sage: asymptotic_expansions.SingularityAnalysis(
....:     'n', alpha=0, beta=0, delta=1, precision=3)
Traceback (most recent call last):
...
NotImplementedError: not implemented for delta!=0
```

static Stirling (*var*, *precision=None*, *skip_constant_factor=False*)

Return Stirling's approximation formula for factorials.

INPUT:

- `var` – a string for the variable name.
- `precision` – (default: `None`) an integer ≥ 3. If `None`, then the default precision of the asymptotic ring is used.
- `skip_constant_factor` – (default: `False`) a boolean. If set, then the constant factor $\sqrt{2\pi}$ is left out. As a consequence, the coefficient ring of the output changes from `Symbolic Constants Subring` (if `False`) to `Rational Field` (if `True`).

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: asymptotic_expansions.Stirling('n', precision=5)
sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(1/2) +
1/12*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-1/2) +
1/288*sqrt(2)*sqrt(pi)*e^(n*log(n))*(e^n)^(-1)*n^(-3/2) +
O(e^(n*log(n))*(e^n)^(-1)*n^(-5/2))
sage: _.parent()
Asymptotic Ring <(e^(n*log(n)))^QQ * (e^n)^QQ * n^QQ * log(n)^QQ>
over Symbolic Constants Subring
```

See also:

log_Stirling(), factorial().

TESTS:

```
sage: expansion = asymptotic_expansions.Stirling('n', precision=5)
sage: n = expansion.parent().gen()
sage: expansion.compare_with_values(n, lambda x: x.factorial(), [5, 10, 20])  # rel tol 1e-6
[(5, 0.00675841118?), (10, 0.0067589306?), (20, 0.006744925?)]
sage: asymptotic_expansions.Stirling('n', precision=5,
....:                                skip_constant_factor=True)
e^(n*log(n))*(e^n)^(-1)*n^(1/2) +
1/12*e^(n*log(n))*(e^n)^(-1)*n^(-1/2) +
1/288*e^(n*log(n))*(e^n)^(-1)*n^(-3/2) +
O(e^(n*log(n))*(e^n)^(-1)*n^(-5/2))
sage: _.parent()
Asymptotic Ring <(e^(n*log(n)))^QQ * (e^n)^QQ * n^QQ * log(n)^QQ>
over Rational Field
sage: asymptotic_expansions.Stirling('m', precision=4)
sqrt(2)*sqrt(pi)*e^(m*log(m))*(e^m)^(-1)*m^(1/2) +
O(e^(m*log(m))*(e^m)^(-1)*m^(-1/2))
sage: asymptotic_expansions.Stirling('m', precision=3)
O(e^(m*log(m))*(e^m)^(-1)*m^(1/2))
sage: asymptotic_expansions.Stirling('m', precision=2)
Traceback (most recent call last):
...
ValueError: precision must be at least 3
```

static **log_Stirling**(*var*, *precision=None*, *skip_constant_summand=False*)

Return the logarithm of Stirling's approximation formula for factorials.

INPUT:

- var – a string for the variable name.
- precision – (default: None) an integer. If None, then the default precision of the asymptotic ring is used.
- skip_constant_summand – (default: False) a boolean. If set, then the constant summand $\log(2\pi)/2$ is left out. As a consequence, the coefficient ring of the output changes from Symbolic Constants Subring (if False) to Rational Field (if True).

OUTPUT:

An asymptotic expansion.

EXAMPLES:

```
sage: asymptotic_expansions.log_Stirling('n', precision=7)
n*log(n) - n + 1/2*log(n) + 1/2*log(2*pi) + 1/12*n^(-1)
- 1/360*n^(-3) + 1/1260*n^(-5) + O(n^(-7))
```

See also:

`Stirling()`, `factorial()`.

TESTS:

```
sage: expansion = asymptotic_expansions.log_Stirling('n', precision=7)
sage: n = expansion.parent().gen()
sage: expansion.compare_with_values(n, lambda x: x.factorial().log(), [5, 10, 20])  # rel tc
[(5, 0.000564287?), (10, 0.0005870?), (20, 0.0006?)]
sage: asymptotic_expansions.log_Stirling('n')
n*log(n) - n + 1/2*log(n) + 1/2*log(2*pi) + 1/12*n^(-1)
- 1/360*n^(-3) + 1/1260*n^(-5) - 1/1680*n^(-7) + 1/1188*n^(-9)
- 691/360360*n^(-11) + 1/156*n^(-13) - 3617/122400*n^(-15)
+ 43867/244188*n^(-17) - 174611/125400*n^(-19) + 77683/5796*n^(-21)
- 236364091/1506960*n^(-23) + 657931/300*n^(-25)
- 3392780147/93960*n^(-27) + 1723168255201/2492028*n^(-29)
- 7709321041217/505920*n^(-31) + O(n^(-33))
sage: _.parent()
Asymptotic Ring <n^ZZ * log(n)^ZZ> over Symbolic Constants Subring

sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=7, skip_constant_summand=True)
n*log(n) - n + 1/2*log(n) + 1/12*n^(-1) - 1/360*n^(-3) +
1/1260*n^(-5) + O(n^(-7))
sage: _.parent()
Asymptotic Ring <n^ZZ * log(n)^ZZ> over Rational Field
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=0)
O(n*log(n))
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=1)
n*log(n) + O(n)
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=2)
n*log(n) - n + O(log(n))
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=3)
n*log(n) - n + 1/2*log(n) + O(1)
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=4)
n*log(n) - n + 1/2*log(n) + 1/2*log(2*pi) + O(n^(-1))
sage: asymptotic_expansions.log_Stirling(
....:     'n', precision=5)
n*log(n) - n + 1/2*log(n) + 1/2*log(2*pi) + 1/12*n^(-1)
+ O(n^(-3))
sage: asymptotic_expansions.log_Stirling(
....:     'm', precision=7, skip_constant_summand=True)
m*log(m) - m + 1/2*log(m) + 1/12*m^(-1) - 1/360*m^(-3) +
1/1260*m^(-5) + O(m^(-7))
```

`sage.rings.asymptotic.asymptotic_expansion_generators.`**`asymptotic_expansions`**

A collection of several common asymptotic expansions.

This is an instance of `AsymptoticExpansionGenerators` whose documentation provides more details.

4.3 (Asymptotic) Growth Groups

This module provides support for (asymptotic) growth groups.

Such groups are equipped with a partial order: the elements can be seen as functions, and the behavior as their argument (or arguments) gets large (tend to ∞) is compared.

Growth groups are used for the calculations done in the *asymptotic ring*. There, take a look at the *informal definition*, where examples of growth groups and elements are given as well.

> **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: from sage.rings.asymptotic.growth_group import \
> ....:     GenericGrowthGroup, GrowthGroup
> sage: GenericGrowthGroup(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.
> Growth Group Generic(ZZ)
> sage: GrowthGroup('x^ZZ * log(x)^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.
> Growth Group x^ZZ * log(x)^ZZ
> ```

4.3.1 Description of Growth Groups

Many growth groups can be described by a string, which can also be used to create them. For example, the string `'x^QQ * log(x)^ZZ * QQ^y * y^QQ'` represents a growth group with the following properties:

- It is a growth group in the two variables x and y.
- Its elements are of the form

  $$x^r \cdot \log(x)^s \cdot a^y \cdot y^q$$

  for $r \in \mathbf{Q}$, $s \in \mathbf{Z}$, $a \in \mathbf{Q}$ and $q \in \mathbf{Q}$.
- The order is with respect to $x \to \infty$ and $y \to \infty$ independently of each other.
- To compare such elements, they are split into parts belonging to only one variable. In the example above,

  $$x^{r_1} \cdot \log(x)^{s_1} \leq x^{r_2} \cdot \log(x)^{s_2}$$

  if $(r_1, s_1) \leq (r_2, s_2)$ lexicographically. This reflects the fact that elements x^r are larger than elements $\log(x)^s$ as $x \to \infty$. The factors belonging to the variable y are compared analogously.

  The results of these comparisons are then put together using the product order, i.e., $\leq$ if each component satisfies $\leq$.

Each description string consists of ordered factors—yes, this means $*$ is noncommutative—of strings describing "elementary" growth groups (see the examples below). As stated in the example above, these factors are split by their variable; factors with the same variable are grouped. Reading such factors from left to right determines the order:

Comparing elements of two factors (growth groups) L and R, then all elements of L are considered to be larger than each element of R.

4.3.2 Creating a Growth Group

For many purposes the factory `GrowthGroup` (see `GrowthGroupFactory`) is the most convenient way to generate a growth group.

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
```

Here are some examples:

```
sage: GrowthGroup('z^ZZ')
Growth Group z^ZZ
sage: M = GrowthGroup('z^QQ'); M
Growth Group z^QQ
```

Each of these two generated groups is a `MonomialGrowthGroup`, whose elements are powers of a fixed symbol (above `'z'`). For the order of the elements it is assumed that $z \to \infty$.

Note: Growth groups where the variable tend to some value distinct from ∞ are not yet implemented.

To create elements of M, a generator can be used:

```
sage: z = M.gen()
sage: z^(3/5)
z^(3/5)
```

Strings can also be parsed:

```
sage: M('z^7')
z^7
```

Similarly, we can construct logarithmic factors by:

```
sage: GrowthGroup('log(z)^QQ')
Growth Group log(z)^QQ
```

which again creates a `MonomialGrowthGroup`. An `ExponentialGrowthGroup` is generated in the same way. Our factory gives

```
sage: E = GrowthGroup('QQ^z'); E
Growth Group QQ^z
```

and a typical element looks like this:

```
sage: E.an_element()
(1/2)^z
```

More complex groups are created in a similar fashion. For example

```
sage: C = GrowthGroup('QQ^z * z^QQ * log(z)^QQ'); C
Growth Group QQ^z * z^QQ * log(z)^QQ
```

This contains elements of the form

```
sage: C.an_element()
(1/2)^z*z^(1/2)*log(z)^(1/2)
```

The group C itself is a Cartesian product; to be precise a `UnivariateProduct`. We can see its factors:

```
sage: C.cartesian_factors()
(Growth Group QQ^z, Growth Group z^QQ, Growth Group log(z)^QQ)
```

Multivariate constructions are also possible:

```
sage: GrowthGroup('x^QQ * y^QQ')
Growth Group x^QQ * y^QQ
```

This gives a `MultivariateProduct`.

Both these Cartesian products are derived from the class `GenericProduct`. Moreover all growth groups have the abstract base class `GenericGrowthGroup` in common.

Some Examples

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G_x = GrowthGroup('x^ZZ'); G_x
Growth Group x^ZZ
sage: G_xy = GrowthGroup('x^ZZ * y^ZZ'); G_xy
Growth Group x^ZZ * y^ZZ
sage: G_xy.an_element()
x*y
sage: x = G_xy('x'); y = G_xy('y')
sage: x^2
x^2
sage: elem = x^21*y^21; elem^2
x^42*y^42
```

A monomial growth group itself is totally ordered, all elements are comparable. However, this does **not** hold for Cartesian products:

```
sage: e1 = x^2*y; e2 = x*y^2
sage: e1 <= e2 or e2 <= e1
False
```

In terms of uniqueness, we have the following behaviour:

```
sage: GrowthGroup('x^ZZ * y^ZZ') is GrowthGroup('y^ZZ * x^ZZ')
True
```

The above is `True` since the order of the factors does not play a role here; they use different variables. But when using the same variable, it plays a role:

```
sage: GrowthGroup('x^ZZ * log(x)^ZZ') is GrowthGroup('log(x)^ZZ * x^ZZ')
False
```

In this case the components are ordered lexicographically, which means that in the second growth group, `log(x)` is assumed to grow faster than `x` (which is nonsense, mathematically). See `CartesianProduct` for more details or see *above* for a more extensive description.

Short notation also allows the construction of more complicated growth groups:

```
sage: G = GrowthGroup('QQ^x * x^ZZ * log(x)^QQ * y^QQ')
sage: G.an_element()
(1/2)^x*x*log(x)^(1/2)*y^(1/2)
sage: x, y = var('x y')
sage: G(2^x * log(x) * y^(1/2)) * G(x^(-5) * 5^x * y^(1/3))
10^x*x^(-5)*log(x)*y^(5/6)
```

AUTHORS:

- Benjamin Hackl (2015)
- Daniel Krenn (2015)
- Clemens Heuberger (2016)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

4.3.3 Classes and Methods

class `sage.rings.asymptotic.growth_group.`**`AbstractGrowthGroupFunctor`**(*var*, *domain*)

Bases: `sage.categories.pushout.ConstructionFunctor`

A base class for the functors constructing growth groups.

INPUT:

- `var` – a string or list of strings (or anything else `Variable` accepts).
- `domain` – a category.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('z^QQ').construction()[0]  # indirect doctest
MonomialGrowthGroup[z]
```

See also:

Asymptotic Ring, `ExponentialGrowthGroupFunctor`, `MonomialGrowthGroupFunctor`, `sage.rings.asymptotic.asymptotic_ring.AsymptoticRingFunctor`, `sage.categories.pushout.ConstructionFunctor`.

`merge`(*other*)

Merge this functor with `other` of possible.

INPUT:

- `other` – a functor.

OUTPUT:

A functor or `None`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: F = GrowthGroup('QQ^t').construction()[0]
sage: G = GrowthGroup('t^QQ').construction()[0]
```

```
sage: F.merge(F)
ExponentialGrowthGroup[t]
sage: F.merge(G) is None
True
```

class sage.rings.asymptotic.growth_group.**ExponentialGrowthElement**(*parent*, *raw_element*)

Bases: sage.rings.asymptotic.growth_group.GenericGrowthElement

An implementation of exponential growth elements.

INPUT:

- parent – an ExponentialGrowthGroup.
- raw_element – an element from the base ring of the parent.

  This raw_element is the base of the created exponential growth element.

An exponential growth element represents a term of the type $\operatorname{base}^{\operatorname{variable}}$. The multiplication corresponds to the multiplication of the bases.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('ZZ^x')
sage: e1 = P(1); e1
1
sage: e2 = P(raw_element=2); e2
2^x
sage: e1 == e2
False
sage: P.le(e1, e2)
True
sage: P.le(e1, P(1)) and P.le(P(1), e2)
True
```

base

The base of this exponential growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('ZZ^x')
sage: P(42^x).base
42
```

class sage.rings.asymptotic.growth_group.**ExponentialGrowthGroup**(*base*, *var*, *category*)

Bases: sage.rings.asymptotic.growth_group.GenericGrowthGroup

A growth group dealing with expressions involving a fixed variable/symbol as the exponent.

The elements ExponentialGrowthElement of this group represent exponential functions with bases from a fixed base ring; the group law is the multiplication.

INPUT:

- base – one of SageMath's parents, out of which the elements get their data (raw_element).

  As exponential expressions are represented by this group, the elements in base are the bases of these exponentials.

- `var` – an object.

  The string representation of `var` acts as an exponent of the elements 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: from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup
sage: P = ExponentialGrowthGroup(QQ, 'x'); P
Growth Group QQ^x
```

See also:

`GenericGrowthGroup`

Element
alias of `ExponentialGrowthElement`

Magmas
alias of `Magmas`

Posets
alias of `Posets`

Sets
alias of `Sets`

construction()
Return the construction of this growth group.

OUTPUT:

A pair whose first entry is an `exponential construction functor` and its second entry the base.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('QQ^x').construction()
(ExponentialGrowthGroup[x], Rational Field)
```

gens()
Return a tuple of all generators of this exponential growth group.

INPUT:

Nothing.

OUTPUT:

An empty tuple.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: E = GrowthGroup('ZZ^x')
sage: E.gens()
()
```

some_elements()
Return some elements of this exponential growth group.

See TestSuite for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: tuple(GrowthGroup('QQ^z').some_elements())
((1/2)^z, (-1/2)^z, 2^z, (-2)^z, 1, (-1)^z,
 42^z, (2/3)^z, (-2/3)^z, (3/2)^z, (-3/2)^z, ...)
```

class sage.rings.asymptotic.growth_group.**ExponentialGrowthGroupFunctor**(*var*)

Bases: sage.rings.asymptotic.growth_group.AbstractGrowthGroupFunctor

A construction functor for exponential growth groups.

INPUT:

- var – a string or list of strings (or anything else Variable accepts).

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup, ExponentialGrowthGroupFunctor
sage: GrowthGroup('QQ^z').construction()[0]
ExponentialGrowthGroup[z]
```

See also:

Asymptotic Ring, AbstractGrowthGroupFunctor, MonomialGrowthGroupFunctor, sage.rings.asymptotic.asymptotic_ring.AsymptoticRingFunctor, sage.categories.pushout.ConstructionFunctor.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup, ExponentialGrowthGroupFunctor
sage: cm = sage.structure.element.get_coercion_model()
sage: A = GrowthGroup('QQ^x')
sage: B = ExponentialGrowthGroupFunctor('x')(ZZ['t'])
sage: cm.common_parent(A, B)
Growth Group QQ[t]^x
```

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: from sage.rings.asymptotic.growth_group import (GenericGrowthGroup,
....:                                                  GenericGrowthElement)
sage: G = GenericGrowthGroup(ZZ)
sage: g = GenericGrowthElement(G, 42); g
GenericGrowthElement(42)
sage: g.parent()
Growth Group Generic(ZZ)
sage: G(raw_element=42) == g
True
```

factors()

Return the atomic factors of this growth element. An atomic factor cannot be split further.

INPUT:

Nothing.

OUTPUT:

A tuple of growth elements.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ')
sage: G.an_element().factors()
(x,)
```

is_lt_one()

Return whether this element is less than 1.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ'); x = G(x)
sage: (x^42).is_lt_one()  # indirect doctest
False
sage: (x^(-42)).is_lt_one()  # indirect doctest
True
```

log(*base=None*)

Return the logarithm of this element.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.

OUTPUT:

A growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ')
sage: x, = G.gens_monomial()
```

```
sage: log(x)  # indirect doctest
log(x)
sage: log(x^5)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: When calculating log(x^5) a factor 5 != 1 appeared,
which is not contained in Growth Group x^ZZ * log(x)^ZZ.

sage: G = GrowthGroup('QQ^x * x^ZZ')
sage: x, = G.gens_monomial()
sage: el = x.rpow(2); el
2^x
sage: log(el)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: When calculating log(2^x) a factor log(2) != 1
appeared, which is not contained in Growth Group QQ^x * x^ZZ.
sage: log(el, base=2)  # indirect doctest
x

sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: x = GenericGrowthGroup(ZZ).an_element()
sage: log(x)  # indirect doctest
Traceback (most recent call last):
...
NotImplementedError: Cannot determine logarithmized factorization of
GenericGrowthElement(1) in abstract base class.

sage: x = GrowthGroup('x^ZZ').an_element()
sage: log(x)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot build log(x) since log(x) is not in
Growth Group x^ZZ.
```

TESTS:

```
sage: G = GrowthGroup("(e^x)^QQ * x^ZZ")
sage: x, = G.gens_monomial()
sage: log(exp(x))  # indirect doctest
x
sage: G.one().log()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: log(1) is zero, which is not contained in
Growth Group (e^x)^QQ * x^ZZ.

sage: G = GrowthGroup("(e^x)^ZZ * x^ZZ")
sage: x, = G.gens_monomial()
sage: log(exp(x))  # indirect doctest
x
sage: G.one().log()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: log(1) is zero, which is not contained in
Growth Group (e^x)^ZZ * x^ZZ.
```

```
sage: G = GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ')
sage: x, y = G.gens_monomial()
sage: (x * y).log()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Calculating log(x*y) results in a sum,
which is not contained in
Growth Group QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ.
```

log_factor(*base=None*)

Return the logarithm of the factorization of this element.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.

OUTPUT:

A tuple of pairs, where the first entry is a growth element and the second a multiplicative coefficient.

ALGORITHM:

This function factors the given element and calculates the logarithm of each of these factors.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ')
sage: x, y = G.gens_monomial()
sage: (x * y).log_factor()  # indirect doctest
((log(x), 1), (log(y), 1))
sage: (x^123).log_factor()  # indirect doctest
((log(x), 123),)
sage: (G('2^x') * x^2).log_factor(base=2)  # indirect doctest
((x, 1), (log(x), 2/log(2)))

sage: G(1).log_factor()
()

sage: log(x).log_factor()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot build log(log(x)) since log(log(x)) is
not in Growth Group QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ.
```

See also:

`factors()`, `log()`.

TESTS:

```
sage: G = GrowthGroup("(e^x)^ZZ * x^ZZ * log(x)^ZZ")
sage: x, = G.gens_monomial()
sage: (exp(x) * x).log_factor()  # indirect doctest
((x, 1), (log(x), 1))
```

rpow(*base*)

Calculate the power of `base` to this element.

INPUT:

- `base` – an element.

OUTPUT:

A growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^x * x^ZZ')
sage: x = G('x')
sage: x.rpow(2)  # indirect doctest
2^x
sage: x.rpow(1/2)  # indirect doctest
(1/2)^x
```

```
sage: x.rpow(0)  # indirect doctest
Traceback (most recent call last):
...
ValueError: 0 is not an allowed base for calculating the power to x.
sage: (x^2).rpow(2)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot construct 2^(x^2) in Growth Group QQ^x * x^ZZ
> *previous* TypeError: unsupported operand parent(s) for '*':
'Growth Group QQ^x * x^ZZ' and 'Growth Group ZZ^(x^2)'
```

```
sage: G = GrowthGroup('QQ^(x*log(x)) * x^ZZ * log(x)^ZZ')
sage: x = G('x')
sage: (x * log(x)).rpow(2)  # indirect doctest
2^(x*log(x))
```

```
sage: n = GrowthGroup('QQ^n * n^QQ')('n')
sage: n.rpow(2)
2^n
sage: _.parent()
Growth Group QQ^n * n^QQ
```

variable_names()

Return the names of the variables of this growth element.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('m^QQ')
sage: G('m^2').variable_names()
('m',)
sage: G('m^0').variable_names()
()
```

```
sage: G = GrowthGroup('QQ^m')
sage: G('2^m').variable_names()
('m',)
sage: G('1^m').variable_names()
()
```

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: G = GenericGrowthGroup(QQ)
sage: G(raw_element=2).variable_names()
Traceback (most recent call last):
...
AttributeError: 'GenericGrowthGroup_with_category.element_class' object
has no attribute 'is_one'
```

class `sage.rings.asymptotic.growth_group.`**`GenericGrowthGroup`**(*base*, *var*, *category*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

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.
- `ignore_variables` – (default: `None`) a tuple (or other iterable) of strings. The specified names are not considered as variables.

Note: This class should be derived for concrete implementations.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: G = GenericGrowthGroup(ZZ); G
Growth Group Generic(ZZ)
```

See also:

`MonomialGrowthGroup`, `ExponentialGrowthGroup`

`AdditiveMagmas`
alias of `AdditiveMagmas`

`Element`
alias of `GenericGrowthElement`

`Magmas`
alias of `Magmas`

`Posets`
alias of `Posets`

`Sets`
alias of `Sets`

`gen`(*n=0*)
Return the n-th generator (as a group) of this growth group.

INPUT:

- `n` – default: 0.

OUTPUT:

A `MonomialGrowthElement`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('x^ZZ')
sage: P.gen()
x

sage: P = GrowthGroup('QQ^x')
sage: P.gen()
Traceback (most recent call last):
...
IndexError: tuple index out of range
```

gens ()

Return a tuple of all generators of this growth group.

INPUT:

Nothing.

OUTPUT:

A tuple whose entries are growth elements.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('x^ZZ')
sage: P.gens()
(x,)
sage: GrowthGroup('log(x)^ZZ').gens()
(log(x),)
```

gens_monomial ()

Return a tuple containing monomial generators of this growth group.

INPUT:

Nothing.

OUTPUT:

An empty tuple.

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: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: GenericGrowthGroup(ZZ).gens_monomial()
()

sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^QQ').gens_monomial()
(x,)
sage: GrowthGroup('QQ^x').gens_monomial()
()
```

le (*left*, *right*)

Return whether 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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ')
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
```

ngens()

Return the number of generators (as a group) of this growth group.

INPUT:

Nothing.

OUTPUT:

A Python integer.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('x^ZZ')
sage: P.ngens()
1
sage: GrowthGroup('log(x)^ZZ').ngens()
1

sage: P = GrowthGroup('QQ^x')
sage: P.ngens()
0
```

some_elements()

Return some elements of this growth group.

See `TestSuite` for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: tuple(GrowthGroup('z^ZZ').some_elements())
(1, z, z^(-1), z^2, z^(-2), z^3, z^(-3),
 z^4, z^(-4), z^5, z^(-5), ...)
sage: tuple(GrowthGroup('z^QQ').some_elements())
(z^(1/2), z^(-1/2), z^2, z^(-2),
 1, z, z^(-1), z^42,
 z^(2/3), z^(-2/3), z^(3/2), z^(-3/2),
 z^(4/5), z^(-4/5), z^(5/4), z^(-5/4), ...)
```

variable_names()

Return the names of the variables of this growth group.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: GenericGrowthGroup(ZZ).variable_names()
()
```

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ').variable_names()
('x',)
sage: GrowthGroup('log(x)^ZZ').variable_names()
('x',)
```

```
sage: GrowthGroup('QQ^x').variable_names()
('x',)
sage: GrowthGroup('QQ^(x*log(x))').variable_names()
('x',)
```

sage.rings.asymptotic.growth_group.**GrowthGroup**

A factory for growth groups. This is an instance of GrowthGroupFactory whose documentation provides more details.

class sage.rings.asymptotic.growth_group.**GrowthGroupFactory**

Bases: sage.structure.factory.UniqueFactory

A factory creating asymptotic growth groups.

INPUT:

- specification – a string.
- keyword arguments are passed on to the growth group constructor. If the keyword ignore_variables is not specified, then ignore_variables=('e',) (to ignore e as a variable name) is used.

OUTPUT:

An asymptotic growth group.

Note: An instance of this factory is available as GrowthGroup.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ')
Growth Group x^ZZ
```

```
sage: GrowthGroup('log(x)^QQ')
Growth Group log(x)^QQ
```

This factory can also be used to construct Cartesian products of growth groups:

```
sage: GrowthGroup('x^ZZ * y^ZZ')
Growth Group x^ZZ * y^ZZ
sage: GrowthGroup('x^ZZ * log(x)^ZZ')
Growth Group x^ZZ * log(x)^ZZ
sage: GrowthGroup('x^ZZ * log(x)^ZZ * y^QQ')
Growth Group x^ZZ * log(x)^ZZ * y^QQ
sage: GrowthGroup('QQ^x * x^ZZ * y^QQ * QQ^z')
Growth Group QQ^x * x^ZZ * y^QQ * QQ^z
sage: GrowthGroup('exp(x)^ZZ * x^ZZ')
Growth Group exp(x)^ZZ * x^ZZ
sage: GrowthGroup('(e^x)^ZZ * x^ZZ')
Growth Group (e^x)^ZZ * x^ZZ
```

TESTS:

```
sage: G = GrowthGroup('(e^(n*log(n)))^ZZ')
sage: G, G._var_
(Growth Group (e^(n*log(n)))^ZZ, e^(n*log(n)))
sage: G = GrowthGroup('(e^n)^ZZ')
sage: G, G._var_
(Growth Group (e^n)^ZZ, e^n)
sage: G = GrowthGroup('(e^(n*log(n)))^ZZ * (e^n)^ZZ * n^ZZ * log(n)^ZZ')
sage: G, tuple(F._var_ for F in G.cartesian_factors())
(Growth Group (e^(n*log(n)))^ZZ * (e^n)^ZZ * n^ZZ * log(n)^ZZ,
 (e^(n*log(n)), e^n, n, log(n)))

sage: TestSuite(GrowthGroup('x^ZZ')).run(verbose=True)  # long time
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_category() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . 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_inverse() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_some_elements() . . . pass

sage: TestSuite(GrowthGroup('QQ^y')).run(verbose=True)  # long time
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
```

```
running ._test_category() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . 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_inverse() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_some_elements() . . . pass

sage: TestSuite(GrowthGroup('x^QQ * log(x)^ZZ')).run(verbose=True)  # long time
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_category() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . 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_inverse() . . . pass
running ._test_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_some_elements() . . . pass
```

create_key_and_extra_args (*specification*, ***kwds*)

Given the arguments and keyword, create a key that uniquely determines this object.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup.create_key_and_extra_args('asdf')
Traceback (most recent call last):
...
ValueError: 'asdf' is not a valid substring of 'asdf' describing a growth group.
```

create_object (*version*, *factors*, ***kwds*)

Create an object from the given arguments.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('as^df')  # indirect doctest
Traceback (most recent call last):
...
ValueError: 'as^df' is not a valid substring of as^df
describing a growth group.
> *previous* ValueError: Cannot create a parent out of 'as'.
>> *previous* SyntaxError: unexpected EOF while parsing (<string>, line 1)
> *and* ValueError: Cannot create a parent out of 'df'.
>> *previous* NameError: name 'df' is not defined
sage: GrowthGroup('x^y^z')
Traceback (most recent call last):
...
ValueError: 'x^y^z' is an ambigous substring of
a growth group description of 'x^y^z'.
Use parentheses to make it unique.
sage: GrowthGroup('(x^y)^z')
Traceback (most recent call last):
...
ValueError: '(x^y)^z' is not a valid substring of (x^y)^z
describing a growth group.
> *previous* ValueError: Cannot create a parent out of 'x^y'.
>> *previous* NameError: name 'x' is not defined
> *and* ValueError: Cannot create a parent out of 'z'.
>> *previous* NameError: name 'z' is not defined
sage: GrowthGroup('x^(y^z)')
Traceback (most recent call last):
...
ValueError: 'x^(y^z)' is not a valid substring of x^(y^z)
describing a growth group.
> *previous* ValueError: Cannot create a parent out of 'x'.
>> *previous* NameError: name 'x' is not defined
> *and* ValueError: Cannot create a parent out of 'y^z'.
>> *previous* NameError: name 'y' is not defined
```

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 $\mathrm{variable}^{\mathrm{exponent}}$. The multiplication corresponds to the addition of the exponents.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import MonomialGrowthGroup
sage: P = 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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('x^ZZ')
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: from sage.rings.asymptotic.growth_group import MonomialGrowthGroup
sage: P = MonomialGrowthGroup(ZZ, 'x'); P
Growth Group x^ZZ
sage: MonomialGrowthGroup(ZZ, log(SR.var('y')))
Growth Group log(y)^ZZ
```

See also:

GenericGrowthGroup

TESTS:

```
sage: L1 = MonomialGrowthGroup(QQ, log(x))
sage: L2 = MonomialGrowthGroup(QQ, 'log(x)')
sage: L1 is L2
True
```

AdditiveMagmas

alias of AdditiveMagmas

Element

alias of MonomialGrowthElement

Magmas
alias of Magmas

Posets
alias of Posets

Sets
alias of Sets

construction()
Return the construction of this growth group.

OUTPUT:

A pair whose first entry is a monomial construction functor and its second entry the base.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ').construction()
(MonomialGrowthGroup[x], Integer Ring)
```

gens_logarithmic()
Return a tuple containing logarithmic generators of this growth group.

INPUT:

Nothing.

OUTPUT:

A tuple containing elements of this growth group.

Note: A generator is called logarithmic generator if the variable of the underlying growth group is the logarithm of a valid identifier. For example, `x^ZZ` has no logarithmic generator, while `log(x)^ZZ` has `log(x)` as logarithmic generator.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ').gens_logarithmic()
()
sage: GrowthGroup('log(x)^QQ').gens_logarithmic()
(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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ').gens_monomial()
(x,)
sage: GrowthGroup('log(x)^QQ').gens_monomial()
()
```

class sage.rings.asymptotic.growth_group.**MonomialGrowthGroupFunctor**(*var*)

Bases: sage.rings.asymptotic.growth_group.AbstractGrowthGroupFunctor

A construction functor for monomial growth groups.

INPUT:

- var – a string or list of strings (or anything else Variable accepts).

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup, MonomialGrowthGroupFunctor
sage: GrowthGroup('z^QQ').construction()[0]
MonomialGrowthGroup[z]
```

See also:

Asymptotic Ring, AbstractGrowthGroupFunctor, ExponentialGrowthGroupFunctor, sage.rings.asymptotic.asymptotic_ring.AsymptoticRingFunctor, sage.categories.pushout.ConstructionFunctor.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup, MonomialGrowthGroupFunctor
sage: cm = sage.structure.element.get_coercion_model()
sage: A = GrowthGroup('x^QQ')
sage: B = MonomialGrowthGroupFunctor('x')(ZZ['t'])
sage: cm.common_parent(A, B)
Growth Group x^QQ[t]
```

class sage.rings.asymptotic.growth_group.**Variable**(*var*, *repr=None*, *latex_name=None*, *ignore=None*)

Bases: sage.structure.unique_representation.CachedRepresentation, sage.structure.sage_object.SageObject

A class managing the variable of a growth group.

INPUT:

- var – an object whose representation string is used as the variable. It has to be a valid Python identifier. var can also be a tuple (or other iterable) of such objects.
- repr – (default: None) if specified, then this string will be displayed instead of var. Use this to get e.g. log(x)^ZZ: var is then used to specify the variable x.
- latex_name – (default: None) if specified, then this string will be used as LaTeX-representation of var.
- ignore – (default: None) a tuple (or other iterable) of strings which are not variables.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import Variable
sage: v = Variable('x'); repr(v), v.variable_names()
('x', ('x',))
sage: v = Variable('x1'); repr(v), v.variable_names()
('x1', ('x1',))
```

```
sage: v = Variable('x_42'); repr(v), v.variable_names()
('x_42', ('x_42',))
sage: v = Variable(' x'); repr(v), v.variable_names()
('x', ('x',))
sage: v = Variable('x '); repr(v), v.variable_names()
('x', ('x',))
sage: v = Variable(''); repr(v), v.variable_names()
('', ())
```

```
sage: v = Variable(('x', 'y')); repr(v), v.variable_names()
('x, y', ('x', 'y'))
sage: v = Variable(('x', 'log(y)')); repr(v), v.variable_names()
('x, log(y)', ('x', 'y'))
sage: v = Variable(('x', 'log(x)')); repr(v), v.variable_names()
Traceback (most recent call last):
...
ValueError: Variable names ('x', 'x') are not pairwise distinct.
```

```
sage: v = Variable('log(x)'); repr(v), v.variable_names()
('log(x)', ('x',))
sage: v = Variable('log(log(x))'); repr(v), v.variable_names()
('log(log(x))', ('x',))
```

```
sage: v = Variable('x', repr='log(x)'); repr(v), v.variable_names()
('log(x)', ('x',))
```

```
sage: v = Variable('e^x', ignore=('e',)); repr(v), v.variable_names()
('e^x', ('x',))
```

```
sage: v = Variable('(e^n)', ignore=('e',)); repr(v), v.variable_names()
('e^n', ('n',))
sage: v = Variable('(e^(n*log(n)))', ignore=('e',)); repr(v), v.variable_names()
('e^(n*log(n))', ('n',))
```

static extract_variable_names (*s*)

Determine the name of the variable for the given string.

INPUT:

- s – a string.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import Variable
sage: Variable.extract_variable_names('')
()
sage: Variable.extract_variable_names('x')
('x',)
sage: Variable.extract_variable_names('exp(x)')
('x',)
sage: Variable.extract_variable_names('sin(cos(ln(x)))')
('x',)
```

```
sage: Variable.extract_variable_names('log(77w)')
('w',)
sage: Variable.extract_variable_names('log(x')
```

```
Traceback (most recent call last):
....
TypeError: Bad function call: log(x !!!
sage: Variable.extract_variable_names('x)')
Traceback (most recent call last):
....
TypeError: Malformed expression: x) !!!
sage: Variable.extract_variable_names('log)x(')
Traceback (most recent call last):
....
TypeError: Malformed expression: log) !!! x(
sage: Variable.extract_variable_names('log(x)+y')
('x', 'y')
sage: Variable.extract_variable_names('icecream(summer)')
('summer',)
```

```
sage: Variable.extract_variable_names('a + b')
('a', 'b')
sage: Variable.extract_variable_names('a+b')
('a', 'b')
sage: Variable.extract_variable_names('a +b')
('a', 'b')
sage: Variable.extract_variable_names('+a')
('a',)
sage: Variable.extract_variable_names('a+')
Traceback (most recent call last):
...
TypeError: Malformed expression: a+ !!!
sage: Variable.extract_variable_names('b!')
('b',)
sage: Variable.extract_variable_names('-a')
('a',)
sage: Variable.extract_variable_names('a*b')
('a', 'b')
sage: Variable.extract_variable_names('2^q')
('q',)
sage: Variable.extract_variable_names('77')
()
```

```
sage: Variable.extract_variable_names('a + (b + c) + d')
('a', 'b', 'c', 'd')
```

is_monomial()

Return whether this is a monomial variable.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import Variable
sage: Variable('x').is_monomial()
True
sage: Variable('log(x)').is_monomial()
False
```

variable_names()

Return the names of the variables.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import Variable
sage: Variable('x').variable_names()
('x',)
sage: Variable('log(x)').variable_names()
('x',)
```

4.4 Cartesian Products of Growth Groups

See *(Asymptotic) Growth Groups* for a description.

AUTHORS:

- Benjamin Hackl (2015)
- Daniel Krenn (2015)
- Clemens Heuberger (2016)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

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: from sage.rings.asymptotic.growth_group import GenericGrowthGroup, GrowthGroup
sage: GenericGrowthGroup(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.
Growth Group Generic(ZZ)
sage: GrowthGroup('x^ZZ * log(x)^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.
Growth Group x^ZZ * log(x)^ZZ
```

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: A = GrowthGroup('QQ^x * x^ZZ'); A
Growth Group QQ^x * x^ZZ
sage: A.construction()
(The cartesian_product functorial construction,
 (Growth Group QQ^x, Growth Group x^ZZ))
sage: A.construction()[1][0].construction()
```

```
(ExponentialGrowthGroup[x], Rational Field)
sage: A.construction()[1][1].construction()
(MonomialGrowthGroup[x], Integer Ring)
sage: B = GrowthGroup('x^ZZ * y^ZZ'); B
Growth Group x^ZZ * y^ZZ
sage: B.construction()
(The cartesian_product functorial construction,
 (Growth Group x^ZZ, Growth Group y^ZZ))
sage: C = GrowthGroup('x^ZZ * log(x)^ZZ * y^ZZ'); C
Growth Group x^ZZ * log(x)^ZZ * y^ZZ
sage: C.construction()
(The cartesian_product functorial construction,
 (Growth Group x^ZZ * log(x)^ZZ, Growth Group y^ZZ))
sage: C.construction()[1][0].construction()
(The cartesian_product functorial construction,
 (Growth Group x^ZZ, Growth Group log(x)^ZZ))
sage: C.construction()[1][1].construction()
(MonomialGrowthGroup[y], Integer Ring)
```

```
sage: cm = sage.structure.element.get_coercion_model()
sage: D = GrowthGroup('QQ^x * x^QQ')
sage: cm.common_parent(A, D)
Growth Group QQ^x * x^QQ
sage: E = GrowthGroup('ZZ^x * x^QQ')
sage: cm.record_exceptions()  # not tested, see #19411
sage: cm.common_parent(A, E)
Growth Group QQ^x * x^QQ
sage: for t in cm.exception_stack():  # not tested, see #19411
....:     print t
```

```
sage: A.an_element()
(1/2)^x*x
sage: tuple(E.an_element())
(1, x^(1/2))
```

4.4.1 Classes and Methods

class `sage.rings.asymptotic.growth_group_cartesian.`**`CartesianProductFactory`**

Bases: `sage.structure.factory.UniqueFactory`

Create various types of Cartesian products depending on its input.

INPUT:

- `growth_groups` – a tuple (or other iterable) of growth groups.
- `order` – (default: `None`) if specified, then this order is taken for comparing two Cartesian product elements. If `order` is `None` this is determined automatically.

Note: The Cartesian product of growth groups is again a growth group. In particular, the resulting structure is partially ordered.

The order on the product is determined as follows:

- Cartesian factors with respect to the same variable are ordered lexicographically. This causes `GrowthGroup('x^ZZ * log(x)^ZZ')` and `GrowthGroup('log(x)^ZZ * x^ZZ')` to produce two different growth groups.

- Factors over different variables are equipped with the product order (i.e. the comparison is component-wise).

Also, note that the sets of variables of the Cartesian factors have to be either equal or disjoint.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: A = GrowthGroup('x^ZZ'); A
Growth Group x^ZZ
sage: B = GrowthGroup('log(x)^ZZ'); B
Growth Group log(x)^ZZ
sage: C = cartesian_product([A, B]); C  # indirect doctest
Growth Group x^ZZ * log(x)^ZZ
sage: C._le_ == C.le_lex
True
sage: D = GrowthGroup('y^ZZ'); D
Growth Group y^ZZ
sage: E = cartesian_product([A, D]); E  # indirect doctest
Growth Group x^ZZ * y^ZZ
sage: E._le_ == E.le_product
True
sage: F = cartesian_product([C, D]); F  # indirect doctest
Growth Group x^ZZ * log(x)^ZZ * y^ZZ
sage: F._le_ == F.le_product
True
sage: cartesian_product([A, E]); G  # indirect doctest
Traceback (most recent call last):
...
ValueError: The growth groups (Growth Group x^ZZ, Growth Group x^ZZ * y^ZZ)
need to have pairwise disjoint or equal variables.
sage: cartesian_product([A, B, D])  # indirect doctest
Growth Group x^ZZ * log(x)^ZZ * y^ZZ
```

TESTS:

```
sage: from sage.rings.asymptotic.growth_group_cartesian import CartesianProductFactory
sage: CartesianProductFactory('factory')([A, B], category=Groups() & Posets())
Growth Group x^ZZ * log(x)^ZZ
sage: CartesianProductFactory('factory')([], category=Sets())
Traceback (most recent call last):
...
TypeError: Cannot create Cartesian product without factors.
```

create_key_and_extra_args(*growth_groups*, *category*, ***kwds*)

Given the arguments and keywords, create a key that uniquely determines this object.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group_cartesian import CartesianProductFactory
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: A = GrowthGroup('x^ZZ')
sage: CartesianProductFactory('factory').create_key_and_extra_args(
....:     [A], category=Sets(), order='blub')
(((Growth Group x^ZZ,), Category of sets), {'order': 'blub'})
```

create_object(*version*, *args*, ***kwds*)

Create an object from the given arguments.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: cartesian_product([GrowthGroup('x^ZZ')])  # indirect doctest
Growth Group x^ZZ
```

class sage.rings.asymptotic.growth_group_cartesian.**GenericProduct**(*sets*, *category*, ***kwds*)

Bases: sage.combinat.posets.cartesian_product.CartesianProductPoset, sage.rings.asymptotic.growth_group.GenericGrowthGroup

A Cartesian product of growth groups.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: P = GrowthGroup('x^QQ')
sage: L = GrowthGroup('log(x)^ZZ')
sage: C = cartesian_product([P, L], order='lex'); C  # indirect doctest
Growth Group x^QQ * log(x)^ZZ
sage: C.an_element()
x^(1/2)*log(x)
```

```
sage: Px = GrowthGroup('x^QQ')
sage: Lx = GrowthGroup('log(x)^ZZ')
sage: Cx = cartesian_product([Px, Lx], order='lex')  # indirect doctest
sage: Py = GrowthGroup('y^QQ')
sage: C = cartesian_product([Cx, Py], order='product'); C  # indirect doctest
Growth Group x^QQ * log(x)^ZZ * y^QQ
sage: C.an_element()
x^(1/2)*log(x)*y^(1/2)
```

See also:

CartesianProduct, CartesianProductPoset.

class Element

Bases: sage.combinat.posets.cartesian_product.CartesianProductPoset.Element

exp()

The exponential of this element.

INPUT:

Nothing.

OUTPUT:

A growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ * log(log(x))^ZZ')
sage: x = G('x')
sage: exp(log(x))
x
sage: exp(log(log(x)))
log(x)
```

```
sage: exp(x)
Traceback (most recent call last):
...
ArithmeticError: Cannot construct e^x in
```

```
Growth Group x^ZZ * log(x)^ZZ * log(log(x))^ZZ
> *previous* TypeError: unsupported operand parent(s) for '*':
'Growth Group x^ZZ * log(x)^ZZ * log(log(x))^ZZ' and
'Growth Group (e^x)^ZZ'
```

TESTS:

```
sage: E = GrowthGroup("(e^y)^QQ * y^QQ * log(y)^QQ")
sage: y = E('y')
sage: log(exp(y))
y
sage: exp(log(y))
y
```

factors()

Return the atomic factors of this growth element. An atomic factor cannot be split further and is not the identity (1).

INPUT:

Nothing.

OUTPUT:

A tuple of growth elements.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ * y^ZZ')
sage: x, y = G.gens_monomial()
sage: x.factors()
(x,)
sage: f = (x * y).factors(); f
(x, y)
sage: tuple(factor.parent() for factor in f)
(Growth Group x^ZZ, Growth Group y^ZZ)
sage: f = (x * log(x)).factors(); f
(x, log(x))
sage: tuple(factor.parent() for factor in f)
(Growth Group x^ZZ, Growth Group log(x)^ZZ)
```

```
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ * log(log(x))^ZZ * y^QQ')
sage: x, y = G.gens_monomial()
sage: f = (x * log(x) * y).factors(); f
(x, log(x), y)
sage: tuple(factor.parent() for factor in f)
(Growth Group x^ZZ, Growth Group log(x)^ZZ, Growth Group y^QQ)
```

```
sage: G.one().factors()
()
```

is_lt_one()

Return whether this element is less than 1.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ'); x = G(x)
sage: (x^42).is_lt_one()  # indirect doctest
False
sage: (x^(-42)).is_lt_one()  # indirect doctest
True
```

log (*base=None*)

Return the logarithm of this element.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.

OUTPUT:

A growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ')
sage: x, = G.gens_monomial()
sage: log(x)  # indirect doctest
log(x)
sage: log(x^5)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: When calculating log(x^5) a factor 5 != 1 appeared,
which is not contained in Growth Group x^ZZ * log(x)^ZZ.
```

```
sage: G = GrowthGroup('QQ^x * x^ZZ')
sage: x, = G.gens_monomial()
sage: el = x.rpow(2); el
2^x
sage: log(el)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: When calculating log(2^x) a factor log(2) != 1
appeared, which is not contained in Growth Group QQ^x * x^ZZ.
sage: log(el, base=2)  # indirect doctest
x
```

```
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: x = GenericGrowthGroup(ZZ).an_element()
sage: log(x)  # indirect doctest
Traceback (most recent call last):
...
NotImplementedError: Cannot determine logarithmized factorization of
GenericGrowthElement(1) in abstract base class.
```

```
sage: x = GrowthGroup('x^ZZ').an_element()
sage: log(x)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot build log(x) since log(x) is not in
Growth Group x^ZZ.
```

TESTS:

```
sage: G = GrowthGroup("(e^x)^QQ * x^ZZ")
sage: x, = G.gens_monomial()
sage: log(exp(x))  # indirect doctest
x
sage: G.one().log()  # indirect doctest
```

```
Traceback (most recent call last):
...
ArithmeticError: log(1) is zero, which is not contained in
Growth Group (e^x)^QQ * x^ZZ.
```

```
sage: G = GrowthGroup("(e^x)^ZZ * x^ZZ")
sage: x, = G.gens_monomial()
sage: log(exp(x))  # indirect doctest
x
sage: G.one().log()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: log(1) is zero, which is not contained in
Growth Group (e^x)^ZZ * x^ZZ.
```

```
sage: G = GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ')
sage: x, y = G.gens_monomial()
sage: (x * y).log()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Calculating log(x*y) results in a sum,
which is not contained in
Growth Group QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ.
```

log_factor(*base=None*)

Return the logarithm of the factorization of this element.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.

OUTPUT:

A tuple of pairs, where the first entry is a growth element and the second a multiplicative coefficient.

ALGORITHM:

This function factors the given element and calculates the logarithm of each of these factors.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ')
sage: x, y = G.gens_monomial()
sage: (x * y).log_factor()  # indirect doctest
((log(x), 1), (log(y), 1))
sage: (x^123).log_factor()  # indirect doctest
((log(x), 123),)
sage: (G('2^x') * x^2).log_factor(base=2)  # indirect doctest
((x, 1), (log(x), 2/log(2)))
```

```
sage: G(1).log_factor()
()
```

```
sage: log(x).log_factor()  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot build log(log(x)) since log(log(x)) is
not in Growth Group QQ^x * x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ.
```

See also:

`factors()`, `log()`.

TESTS:

```
sage: G = GrowthGroup("(e^x)^ZZ * x^ZZ * log(x)^ZZ")
sage: x, = G.gens_monomial()
sage: (exp(x) * x).log_factor()  # indirect doctest
((x, 1), (log(x), 1))
```

rpow (*base*)

Calculate the power of `base` to this element.

INPUT:

- `base` – an element.

OUTPUT:

A growth element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^x * x^ZZ')
sage: x = G('x')
sage: x.rpow(2)  # indirect doctest
2^x
sage: x.rpow(1/2)  # indirect doctest
(1/2)^x
```

```
sage: x.rpow(0)  # indirect doctest
Traceback (most recent call last):
...
ValueError: 0 is not an allowed base for calculating the power to x.
sage: (x^2).rpow(2)  # indirect doctest
Traceback (most recent call last):
...
ArithmeticError: Cannot construct 2^(x^2) in Growth Group QQ^x * x^ZZ
> *previous* TypeError: unsupported operand parent(s) for '*':
'Growth Group QQ^x * x^ZZ' and 'Growth Group ZZ^(x^2)'
```

```
sage: G = GrowthGroup('QQ^(x*log(x)) * x^ZZ * log(x)^ZZ')
sage: x = G('x')
sage: (x * log(x)).rpow(2)  # indirect doctest
2^(x*log(x))
```

```
sage: n = GrowthGroup('QQ^n * n^QQ')('n')
sage: n.rpow(2)
2^n
sage: _.parent()
Growth Group QQ^n * n^QQ
```

variable_names ()

Return the names of the variables of this growth element.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^m * m^QQ * log(n)^ZZ')
sage: G('2^m * m^4 * log(n)').variable_names()
('m', 'n')
sage: G('2^m * m^4').variable_names()
('m',)
sage: G('log(n)').variable_names()
('n',)
```

```
sage: G('m^3').variable_names()
('m',)
sage: G('m^0').variable_names()
()
```

GenericProduct.**cartesian_injection**(*factor*, *element*)

Inject the given element into this Cartesian product at the given factor.

INPUT:

- `factor` – a growth group (a factor of this Cartesian product).
- `element` – an element of `factor`.

OUTPUT:

An element of this Cartesian product.

TESTS:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * y^QQ')
sage: G.cartesian_injection(G.cartesian_factors()[1], 'y^7')
y^7
```

GenericProduct.**gens_monomial**()

Return a tuple containing monomial generators of this growth group.

INPUT:

Nothing.

OUTPUT:

A tuple containing elements of this growth group.

Note: This method calls the `gens_monomial()` method on the individual factors of this Cartesian product and concatenates the respective outputs.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * log(x)^ZZ * y^QQ * log(z)^ZZ')
sage: G.gens_monomial()
(x, y)
```

TESTS:

```
sage: all(g.parent() == G for g in G.gens_monomial())
True
```

GenericProduct.**some_elements**()

Return some elements of this Cartesian product of growth groups.

See `TestSuite` for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from itertools import islice
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('QQ^y * x^QQ * log(x)^ZZ')
sage: tuple(islice(G.some_elements(), 10))
(x^(1/2)*(1/2)^y,
 x^(-1/2)*log(x)*(-1/2)^y,
 x^2*log(x)^(-1)*2^y,
 x^(-2)*log(x)^2*(-2)^y,
 log(x)^(-2),
 x*log(x)^3*(-1)^y,
 x^(-1)*log(x)^(-3)*42^y,
 x^42*log(x)^4*(2/3)^y,
 x^(2/3)*log(x)^(-4)*(-2/3)^y,
 x^(-2/3)*log(x)^5*(3/2)^y)
```

GenericProduct.**variable_names**()

Return the names of the variables.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^ZZ * log(x)^ZZ * y^QQ * log(z)^ZZ').variable_names()
('x', 'y', 'z')
```

class sage.rings.asymptotic.growth_group_cartesian.**MultivariateProduct**(*sets*, *category*, ***kwargs*)

Bases: sage.rings.asymptotic.growth_group_cartesian.GenericProduct

A Cartesian product of growth groups with pairwise disjoint (or equal) variable sets.

Note: A multivariate product of growth groups is ordered by means of the product order, i.e. component-wise. This is motivated by the assumption that different variables are considered to be independent (e.g. x^ZZ * y^ZZ).

See also:

UnivariateProduct, GenericProduct.

class sage.rings.asymptotic.growth_group_cartesian.**UnivariateProduct**(*sets*, *category*, ***kwargs*)

Bases: sage.rings.asymptotic.growth_group_cartesian.GenericProduct

A Cartesian product of growth groups with the same variables.

Note: A univariate product of growth groups is ordered lexicographically. This is motivated by the assumption that univariate growth groups can be ordered in a chain with respect to the growth they model (e.g. x^ZZ * log(x)^ZZ: polynomial growth dominates logarithmic growth).

See also:

MultivariateProduct, GenericProduct.

4.5 (Asymptotic) Term Monoids

This module implements asymptotic term monoids. The elements of these monoids are used behind the scenes when performing calculations in an *asymptotic ring*.

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 *(Asymptotic) Growth Groups* 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 coefficient ring.

A characteristic property of asymptotic terms is that some terms are able to "absorb" other terms (see `absorb()`). For instance, $O(x^2)$ is able to absorb $O(x)$ (with result $O(x^2)$), and $3 \cdot x^5$ is able to absorb $-2 \cdot x^5$ (with result x^5). Essentially, absorption can be interpreted as the addition of "compatible" terms (partial addition).

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: G = GrowthGroup('x^ZZ * log(x)^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.
```

4.5.1 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: from sage.rings.asymptotic.term_monoid import OTermMonoid, ExactTermMonoid
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: OT = OTermMonoid(growth_group=G, coefficient_ring=QQ)
sage: ET = ExactTermMonoid(growth_group=G, coefficient_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)
```

4.5.2 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.

4.5.3 Various

Todo

- Implementation of more term types (e.g. L terms, Ω terms, o terms, Θ terms).

AUTHORS:

- Benjamin Hackl (2015)
- Daniel Krenn (2015)
- Clemens Heuberger (2016)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

4.5.4 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.coefficient_ring`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import (ExactTermMonoid, TermMonoid)
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: ET = 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 = TermMonoid('O', G, QQ)
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
```

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: ET = TermMonoid('exact', 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
```

is_constant()

Return whether this term is an (exact) constant.

INPUT:

Nothing.

OUTPUT:

A boolean.

Note: Only ExactTerm with constant growth (1) are constant.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: T('x * log(x)').is_constant()
False
sage: T('3*x').is_constant()
False
sage: T(1/2).is_constant()
True
sage: T(42).is_constant()
True
```

is_exact()

Return whether this term is an exact term.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: T('x * log(x)').is_exact()
True
sage: T('3 * x^2').is_exact()
True
```

TESTS:

```
sage: T = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
sage: T('x').is_exact()
False
```

is_little_o_of_one()

Return whether this exact term is of order $o(1)$.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ'), QQ)
sage: T(x).is_little_o_of_one()
False
sage: T(1).is_little_o_of_one()
False
sage: T(x^(-1)).is_little_o_of_one()
True

sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * y^ZZ'), QQ)
sage: T('x * y^(-1)').is_little_o_of_one()
False
sage: T('x^(-1) * y').is_little_o_of_one()
False
sage: T('x^(-2) * y^(-3)').is_little_o_of_one()
True

sage: T = TermMonoid('exact', GrowthGroup('x^QQ * log(x)^QQ'), QQ)
sage: T('x * log(x)^2').is_little_o_of_one()
False
sage: T('x^2 * log(x)^(-1234)').is_little_o_of_one()
False
sage: T('x^(-1) * log(x)^4242').is_little_o_of_one()
True
sage: T('x^(-1/100) * log(x)^(1000/7)').is_little_o_of_one()
True
```

log_term(*base=None*)

Determine the logarithm of this exact term.

INPUT:

- base – the base of the logarithm. If None (default value) is used, the natural logarithm is taken.

OUTPUT:

A tuple of terms.

Note: This method returns a tuple with the summands that come from applying the rule $\log(x \cdot y) = \log(x) + \log(y)$.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * log(x)^ZZ'), SR)
sage: T(3*x^2).log_term()
(log(3), 2*log(x))
```

```
sage: T(x^1234).log_term()
(1234*log(x),)
sage: T(49*x^7).log_term(base=7)
(log(49)/log(7), 7/log(7)*log(x))

sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ'), SR)
sage: T('x * y').log_term()
(log(x), log(y))
sage: T('4 * x * y').log_term(base=2)
(log(4)/log(2), 1/log(2)*log(x), 1/log(2)*log(y))
```

See also:

OTerm.log_term().

rpow (*base*)

Return the power of `base` to this exact term.

INPUT:

- `base` – an element or `'e'`.

OUTPUT:

A term.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ'), QQ)
sage: T('x').rpow(2)
2^x
sage: T('log(x)').rpow('e')
x
sage: T('42*log(x)').rpow('e')
x^42
sage: T('3*x').rpow(2)
8^x

sage: T('3*x^2').rpow(2)
Traceback (most recent call last):
...
ArithmeticError: Cannot construct 2^(x^2) in
Growth Group QQ^x * x^ZZ * log(x)^ZZ
> *previous* TypeError: unsupported operand parent(s) for '*':
'Growth Group QQ^x * x^ZZ * log(x)^ZZ' and 'Growth Group ZZ^(x^2)'

sage: T = TermMonoid('exact', GrowthGroup('QQ^n * n^QQ'), SR)
sage: n = T('n')
sage: n.rpow(2)
2^n
sage: _.parent()
Exact Term Monoid QQ^n * n^SR with coefficients in Symbolic Ring
```

Above, we get `QQ^n * n^SR`. The reason is the following: Since $n = 1_{SR} \cdot (1_{\mathbf{Q}})^n \cdot n^{1_{\mathbf{Q}}}$, we have

$$2^n = (2_{\mathbf{Q}})^{1_{SR} \cdot (1_{\mathbf{Q}})^n \cdot n^{1_{\mathbf{Q}}}} = \left((2_{\mathbf{Q}})^n \cdot n^{0_{\mathbf{Q}}}\right)^{1_{SR}} = \left((2_{\mathbf{Q}})^{1_{SR}}\right)^n \cdot n^{0_{\mathbf{Q}} 1_{SR}} = (2_{\mathbf{Q}})^n \cdot n^{0_{SR}}$$

where

```
sage: (QQ(2)^SR(1)).parent(), (QQ(0)*SR(1)).parent()
(Rational Field, Symbolic Ring)
```

was used.

class `sage.rings.asymptotic.term_monoid.`**`ExactTermMonoid`**(*growth_group*, *coefficient_ring*, *category*)

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.
- `coefficient_ring` – the ring which contains the coefficients of the elements.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import ExactTermMonoid
sage: G_ZZ = GrowthGroup('x^ZZ'); x_ZZ = G_ZZ.gen()
sage: G_QQ = GrowthGroup('x^QQ'); x_QQ = G_QQ.gen()
sage: ET_ZZ = ExactTermMonoid(G_ZZ, ZZ); ET_ZZ
Exact Term Monoid x^ZZ with coefficients in Integer Ring
sage: ET_QQ = ExactTermMonoid(G_QQ, QQ); ET_QQ
Exact Term Monoid x^QQ with coefficients in Rational Field
sage: ET_QQ.coerce_map_from(ET_ZZ)
Conversion map:
  From: Exact Term Monoid x^ZZ with coefficients in Integer Ring
  To:   Exact Term Monoid x^QQ with coefficients in Rational Field
```

Exact term monoids can also be created using the `term factory`:

```
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: TermMonoid('exact', G_ZZ, ZZ) is ET_ZZ
True
sage: TermMonoid('exact', GrowthGroup('x^ZZ'), QQ)
Exact Term Monoid x^ZZ with coefficients in Rational Field
```

`Element`

alias of `ExactTerm`

class `sage.rings.asymptotic.term_monoid.`**`GenericTerm`**(*parent*, *growth*)

Bases: `sage.structure.element.MultiplicativeGroupElement`

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
```

```
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: T = GenericTermMonoid(G, QQ)
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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: G_QQ = GrowthGroup('x^QQ'); x = G_QQ.gen()
sage: OT = TermMonoid('O', G_QQ, coefficient_ring=ZZ)
sage: ET = TermMonoid('exact', G_QQ, coefficient_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: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: G = GenericGrowthGroup(ZZ)
sage: T = GenericTermMonoid(G, QQ)
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
```

is_constant()

Return whether this term is an (exact) constant.

INPUT:

Nothing.

OUTPUT:

A boolean.

Note: Only `ExactTerm` with constant growth (1) are constant.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import (GenericTermMonoid, TermMonoid)
sage: T = GenericTermMonoid(GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: t = T.an_element(); t
Generic Term with growth x*log(x)
sage: t.is_constant()
False

sage: T = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
sage: T('x').is_constant()
False
sage: T(1).is_constant()
False
```

is_exact()

Return whether this term is an exact term.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: T = GenericTermMonoid(GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: T.an_element().is_exact()
False
```

is_little_o_of_one()

Return whether this generic term is of order $o(1)$.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import (GenericTermMonoid,
....:                                                  TermWithCoefficientMonoid)
sage: T = GenericTermMonoid(GrowthGroup('x^ZZ'), QQ)
sage: T.an_element().is_little_o_of_one()
Traceback (most recent call last):
...
NotImplementedError: Cannot check whether Generic Term with growth x is o(1)
in the abstract base class
Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field.
sage: T = TermWithCoefficientMonoid(GrowthGroup('x^ZZ'), QQ)
sage: T.an_element().is_little_o_of_one()
Traceback (most recent call last):
...
NotImplementedError: Cannot check whether Term with coefficient 1/2 and growth x
is o(1) in the abstract base class
Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field.
```

log_term(*base=None*)

Determine the logarithm of this term.

INPUT:

- base – the base of the logarithm. If None (default value) is used, the natural logarithm is taken.

OUTPUT:

A tuple of terms.

Note: This abstract method raises a NotImplementedError. See ExactTerm and OTerm for a concrete implementation.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: T = GenericTermMonoid(GrowthGroup('x^ZZ'), QQ)
sage: T.an_element().log_term()
Traceback (most recent call last):
...
NotImplementedError: This method is not implemented in
this abstract base class.

sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: T = TermWithCoefficientMonoid(GrowthGroup('x^ZZ'), QQ)
sage: T.an_element().log_term()
Traceback (most recent call last):
...
NotImplementedError: This method is not implemented in
this abstract base class.
```

See also:

ExactTerm.log_term(), OTerm.log_term().

rpow(*base*)

Return the power of base to this generic term.

INPUT:

- base – an element or 'e'.

OUTPUT:

A term.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: T = GenericTermMonoid(GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: T.an_element().rpow('e')
Traceback (most recent call last):
...
NotImplementedError: Cannot take e to the exponent
Generic Term with growth x*log(x) in the abstract base class
Generic Term Monoid x^ZZ * log(x)^ZZ with (implicit) coefficients in Rational Field.
```

variable_names()

Return the names of the variables of this term.

OUTPUT:

A tuple of strings.

EXAMPLES:

```
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('exact', 'QQ^m * m^QQ * log(n)^ZZ', QQ)
sage: T('4 * 2^m * m^4 * log(n)').variable_names()
('m', 'n')
sage: T('4 * 2^m * m^4').variable_names()
('m',)
sage: T('4 * log(n)').variable_names()
('n',)
sage: T('4 * m^3').variable_names()
('m',)
sage: T('4 * m^0').variable_names()
()
```

class `sage.rings.asymptotic.term_monoid.`**`GenericTermMonoid`**(*growth_group*, *coefficient_ring*, *category*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.parent.Parent`

Parent for generic asymptotic terms.

INPUT:

- `growth_group` – a growth group (i.e. an instance of `GenericGrowthGroup`).
- `coefficient_ring` – a ring which contains the (maybe implicit) coefficients of the elements.
- `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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: G_x = GrowthGroup('x^ZZ'); x = G_x.gen()
sage: G_y = GrowthGroup('y^QQ'); y = G_y.gen()
sage: T_x_ZZ = GenericTermMonoid(G_x, QQ)
sage: T_y_QQ = GenericTermMonoid(G_y, QQ)
sage: T_x_ZZ
Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field
sage: T_y_QQ
Generic Term Monoid y^QQ with (implicit) coefficients in Rational Field
```

`Element`

alias of `GenericTerm`

`change_parameter`(*growth_group=None*, *coefficient_ring=None*)

Return a term monoid with a change in one or more of the given parameters.

INPUT:

- `growth_group` – (default: `None`) the new growth group.

- `coefficient_ring` – (default: `None`) the new coefficient ring.

OUTPUT:

A term monoid.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: E = TermMonoid('exact', GrowthGroup('n^ZZ'), ZZ)
sage: E.change_parameter(coefficient_ring=QQ)
Exact Term Monoid n^ZZ with coefficients in Rational Field
sage: E.change_parameter(growth_group=GrowthGroup('n^QQ'))
Exact Term Monoid n^QQ with coefficients in Integer Ring
```

TESTS:

```
sage: E.change_parameter() is E
True
sage: E.change_parameter(growth_group=None) is E
True
sage: E.change_parameter(coefficient_ring=None) is E
True
sage: E.change_parameter(growth_group=None, coefficient_ring=None) is E
True
```

coefficient_ring

The coefficient ring of this term monoid, i.e. the ring where the coefficients are from.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: GenericTermMonoid(GrowthGroup('x^ZZ'), ZZ).coefficient_ring
Integer Ring
```

growth_group

The growth group underlying this term monoid.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: TermMonoid('exact', GrowthGroup('x^ZZ'), 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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
```

```
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: T = GenericTermMonoid(G, QQ)
sage: t1 = T(x); t2 = T(x^2)
sage: T.le(t1, t2)
True
```

some_elements()

Return some elements of this term monoid.

See `TestSuite` for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: tuple(TermMonoid('O', G, QQ).some_elements())
(O(1), O(x), O(x^(-1)), O(x^2), O(x^(-2)), O(x^3), ...)
```

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import OTermMonoid
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: OT = OTermMonoid(G, QQ)
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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: OT = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
sage: t1 = OT(x^21); t2 = OT(x^42)
sage: t1.can_absorb(t2)
False
sage: t2.can_absorb(t1)
True
```

is_little_o_of_one()

Return whether this O-term is of order $o(1)$.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
sage: T(x).is_little_o_of_one()
False
sage: T(1).is_little_o_of_one()
False
sage: T(x^(-1)).is_little_o_of_one()
True

sage: T = TermMonoid('O', GrowthGroup('x^ZZ * y^ZZ'), QQ)
sage: T('x * y^(-1)').is_little_o_of_one()
False
sage: T('x^(-1) * y').is_little_o_of_one()
False
sage: T('x^(-2) * y^(-3)').is_little_o_of_one()
True
```

```
sage: T = TermMonoid('O', GrowthGroup('x^QQ * log(x)^QQ'), QQ)
sage: T('x * log(x)^2').is_little_o_of_one()
False
sage: T('x^2 * log(x)^(-1234)').is_little_o_of_one()
False
sage: T('x^(-1) * log(x)^4242').is_little_o_of_one()
True
sage: T('x^(-1/100) * log(x)^(1000/7)').is_little_o_of_one()
True
```

log_term(*base=None*)

Determine the logarithm of this O-term.

INPUT:

- `base` – the base of the logarithm. If `None` (default value) is used, the natural logarithm is taken.

OUTPUT:

A tuple of terms.

Note: This method returns a tuple with the summands that come from applying the rule $\log(x \cdot y) = \log(x) + \log(y)$.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
sage: T(x^2).log_term()
(O(log(x)),)
sage: T(x^1234).log_term()
(O(log(x)),)
```

```
sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ'), QQ)
sage: T('x * y').log_term()
(O(log(x)), O(log(y)))
```

See also:

`ExactTerm.log_term()`.

rpow(*base*)

Return the power of `base` to this O-term.

INPUT:

- `base` – an element or `'e'`.

OUTPUT:

A term.

Note: For `OTerm`, the powers can only be constructed for exponents $O(1)$ or if `base` is 1.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
```

```
sage: T(1).rpow('e')
O(1)
sage: T(1).rpow(2)
O(1)

sage: T.an_element().rpow(1)
1
sage: T('x^2').rpow(1)
1

sage: T.an_element().rpow('e')
Traceback (most recent call last):
...
ValueError: Cannot take e to the exponent O(x*log(x)) in
O-Term Monoid x^ZZ * log(x)^ZZ with implicit coefficients in Rational Field
sage: T('log(x)').rpow('e')
Traceback (most recent call last):
...
ValueError: Cannot take e to the exponent O(log(x)) in
O-Term Monoid x^ZZ * log(x)^ZZ with implicit coefficients in Rational Field
```

class sage.rings.asymptotic.term_monoid.**OTermMonoid**(*growth_group*, *coefficient_ring*, *category*)

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import OTermMonoid
sage: G_x_ZZ = GrowthGroup('x^ZZ')
sage: G_y_QQ = GrowthGroup('y^QQ')
sage: OT_x_ZZ = OTermMonoid(G_x_ZZ, QQ); OT_x_ZZ
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
sage: OT_y_QQ = OTermMonoid(G_y_QQ, QQ); OT_y_QQ
O-Term Monoid y^QQ with implicit coefficients in Rational Field
```

O-term monoids can also be created by using the term factory:

```
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: TermMonoid('O', G_x_ZZ, QQ) is OT_x_ZZ
True
sage: TermMonoid('O', GrowthGroup('x^QQ'), QQ)
O-Term Monoid x^QQ with implicit coefficients in Rational Field
```

Element

alias of OTerm

sage.rings.asymptotic.term_monoid.**TermMonoid**

A factory for asymptotic term monoids. This is an instance of TermMonoidFactory whose documentation provides more details.

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.

Note: An instance of this factory is available as TermMonoid.

INPUT:

- term_monoid – the kind of terms held in the new term monoid. Either a string 'exact' or 'O' (capital letter O), or an existing instance of a term monoid.
- growth_group – a growth group or a string describing a growth group.
- coefficient_ring – a ring.
- asymptotic_ring – if specified, then growth_group and coefficient_ring are taken from this asymptotic ring.

OUTPUT:

An asymptotic term monoid.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: TermMonoid('O', G, QQ)
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
sage: TermMonoid('exact', G, ZZ)
Exact Term Monoid x^ZZ with coefficients in Integer Ring

sage: R = AsymptoticRing(growth_group=G, coefficient_ring=QQ)
sage: TermMonoid('exact', asymptotic_ring=R)
Exact Term Monoid x^ZZ with coefficients in Rational Field
sage: TermMonoid('O', asymptotic_ring=R)
O-Term Monoid x^ZZ with implicit coefficients in Rational Field

sage: TermMonoid('exact', 'QQ^m * m^QQ * log(n)^ZZ', ZZ)
Exact Term Monoid QQ^m * m^QQ * log(n)^ZZ
with coefficients in Integer Ring
```

TESTS:

```
sage: TermMonoid(TermMonoid('O', G, ZZ), asymptotic_ring=R)
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
sage: TermMonoid(TermMonoid('exact', G, ZZ), asymptotic_ring=R)
Exact Term Monoid x^ZZ with coefficients in Rational Field
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: TermMonoid(GenericTermMonoid(G, ZZ), asymptotic_ring=R)
Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field

sage: TestSuite(TermMonoid('exact', GrowthGroup('x^ZZ'), QQ)).run(verbose=True)  # long time
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_category() . . . pass
```

```
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . 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_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_some_elements() . . . pass
```

```
sage: TestSuite(TermMonoid('O', GrowthGroup('x^QQ'), ZZ)).run(verbose=True)  # long time
running ._test_an_element() . . . pass
running ._test_associativity() . . . pass
running ._test_cardinality() . . . pass
running ._test_category() . . . pass
running ._test_elements() . . .
  Running the test suite of self.an_element()
  running ._test_category() . . . pass
  running ._test_eq() . . . 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_not_implemented_methods() . . . pass
running ._test_one() . . . pass
running ._test_pickling() . . . pass
running ._test_prod() . . . pass
running ._test_some_elements() . . . pass
```

create_key_and_extra_args(*term_monoid*, *growth_group=None*, *coefficient_ring=None*, *asymptotic_ring=None*, ***kwds*)

Given the arguments and keyword, create a key that uniquely determines this object.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: TermMonoid.create_key_and_extra_args('O', G, QQ)
((<class 'sage.rings.asymptotic.term_monoid.OTermMonoid'>,
  Growth Group x^ZZ, Rational Field), {})
sage: TermMonoid.create_key_and_extra_args('exact', G, ZZ)
((<class 'sage.rings.asymptotic.term_monoid.ExactTermMonoid'>,
  Growth Group x^ZZ, Integer Ring), {})
sage: TermMonoid.create_key_and_extra_args('exact', G)
Traceback (most recent call last):
...
```

```
ValueError: A coefficient ring has to be specified
to create a term monoid of type 'exact'
```

TESTS:

```
sage: TermMonoid.create_key_and_extra_args('icecream', G)
Traceback (most recent call last):
...
ValueError: Term specification 'icecream' has to be either
'exact' or 'O' or an instance of an existing term.
sage: 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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: TermMonoid('O', G, QQ)  # indirect doctest
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
sage: TermMonoid('exact', G, ZZ)  # indirect doctest
Exact Term Monoid x^ZZ with coefficients in 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 coefficient ring.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: CT_ZZ = TermWithCoefficientMonoid(G, ZZ)
sage: CT_QQ = TermWithCoefficientMonoid(G, QQ)
sage: CT_ZZ(x^2, 5)
Term with coefficient 5 and growth x^2
sage: CT_QQ(x^3, 3/8)
Term with coefficient 3/8 and growth x^3
```

class sage.rings.asymptotic.term_monoid.**TermWithCoefficientMonoid** (*growth_group*, *coefficient_ring*, *category*)

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.
- coefficient_ring – the ring which contains the coefficients of the elements.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: G_ZZ = GrowthGroup('x^ZZ'); x_ZZ = G_ZZ.gen()
sage: G_QQ = GrowthGroup('x^QQ'); x_QQ = G_QQ.gen()
sage: TC_ZZ = TermWithCoefficientMonoid(G_ZZ, QQ); TC_ZZ
Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field
sage: TC_QQ = TermWithCoefficientMonoid(G_QQ, QQ); TC_QQ
Generic Term Monoid x^QQ with (implicit) coefficients in 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: Generic Term Monoid x^ZZ with (implicit) coefficients in Rational Field
  To:   Generic Term Monoid x^QQ with (implicit) coefficients in Rational Field
```

Element

alias of TermWithCoefficient

some_elements()

Return some elements of this term with coefficient monoid.

See TestSuite for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```
sage: from itertools import islice
sage: from sage.rings.asymptotic.term_monoid import TermMonoid
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('z^QQ')
sage: T = TermMonoid('exact', G, ZZ)
sage: tuple(islice(T.some_elements(), 10))
(z^(1/2), z^(-1/2), -z^(1/2), z^2, -z^(-1/2), 2*z^(1/2),
 z^(-2), -z^2, 2*z^(-1/2), -2*z^(1/2))
```

exception sage.rings.asymptotic.term_monoid.**ZeroCoefficientError**

Bases: exceptions.ValueError

sage.rings.asymptotic.term_monoid.**absorption**(*left*, *right*)

Let one of the two passed terms absorb the other.

Helper function used by AsymptoticExpansion.

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import (TermMonoid, absorption)
sage: T = TermMonoid('O', GrowthGroup('x^ZZ'), ZZ)
sage: absorption(T(x^2), T(x^3))
O(x^3)
sage: absorption(T(x^3), T(x^2))
O(x^3)

sage: T = TermMonoid('exact', GrowthGroup('x^ZZ'), ZZ)
sage: 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 `AsymptoticExpansion`.

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: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import (TermMonoid, can_absorb)
sage: T = TermMonoid('O', GrowthGroup('x^ZZ'), ZZ)
sage: can_absorb(T(x^2), T(x^3))
True
sage: can_absorb(T(x^3), T(x^2))
True
```

4.6 Asymptotic Expansions — Miscellaneous

AUTHORS:

- Daniel Krenn (2015)

ACKNOWLEDGEMENT:

- Benjamin Hackl, Clemens Heuberger and Daniel Krenn are supported by the Austrian Science Fund (FWF): P 24644-N26.
- Benjamin Hackl is supported by the Google Summer of Code 2015.

4.6.1 Functions, Classes and Methods

exception `sage.rings.asymptotic.misc.`**`NotImplementedOZero`**(*data=None*, *var=None*)

Bases: `exceptions.NotImplementedError`

A special NotImplementedError which is raised when the result is O(0) which means 0 for sufficiently large values of the variable.

`sage.rings.asymptotic.misc.`**`combine_exceptions`**(*e*, **f*)

Helper function which combines the messages of the given exceptions.

INPUT:

- `e` – an exception.
- `*f` – exceptions.

OUTPUT:

An exception.

EXAMPLES:

```
sage: from sage.rings.asymptotic.misc import combine_exceptions
sage: raise combine_exceptions(ValueError('Outer.'), TypeError('Inner.'))
Traceback (most recent call last):
...
ValueError: Outer.
> *previous* TypeError: Inner.
sage: raise combine_exceptions(ValueError('Outer.'),
....:                          TypeError('Inner1.'), TypeError('Inner2.'))
Traceback (most recent call last):
...
ValueError: Outer.
> *previous* TypeError: Inner1.
> *and* TypeError: Inner2.
sage: raise combine_exceptions(ValueError('Outer.'),
....:                          combine_exceptions(TypeError('Middle.'),
....:                                             TypeError('Inner.')))
Traceback (most recent call last):
...
ValueError: Outer.
> *previous* TypeError: Middle.
>> *previous* TypeError: Inner.
```

`sage.rings.asymptotic.misc.`**`log_string`**(*element*, *base=None*)

Return a representation of the log of the given element to the given base.

INPUT:

- `element` – an object.
- `base` – an object or `None`.

OUTPUT:

A string.

EXAMPLES:

```
sage: from sage.rings.asymptotic.misc import log_string
sage: log_string(3)
'log(3)'
sage: log_string(3, base=42)
'log(3, base=42)'
```

`sage.rings.asymptotic.misc.`**`merge_overlapping`**(*A*, *B*, *key=None*)

Merge the two overlapping tuples/lists.

INPUT:

- `A` – a list or tuple (type has to coincide with type of `B`).
- `B` – a list or tuple (type has to coincide with type of `A`).
- `key` – (default: `None`) a function. If `None`, then the identity is used. This `key`-function applied on an element of the list/tuple is used for comparison. Thus elements with the same key are considered as equal.

OUTPUT:

A pair of lists or tuples (depending on the type of `A` and `B`).

Note: Suppose we can decompose the list $A = ac$ and $B = cb$ with lists a, b, c, where c is nonempty. Then `merge_overlapping()` returns the pair (acb, acb).

Suppose a `key`-function is specified and $A = ac_A$ and $B = c_Bb$, where the list of keys of the elements of c_A equals the list of keys of the elements of c_B. Then `merge_overlapping()` returns the pair (ac_Ab, ac_Bb).

After unsuccessfully merging $A = ac$ and $B = cb$, a merge of $A = ca$ and $B = bc$ is tried.

TESTS:

```
sage: from sage.rings.asymptotic.misc import merge_overlapping
sage: def f(L, s):
....:     return list((ell, s) for ell in L)
sage: key = lambda k: k[0]
sage: merge_overlapping(f([0..3], 'a'), f([5..7], 'b'), key)
Traceback (most recent call last):
...
ValueError: Input does not have an overlap.
sage: merge_overlapping(f([0..2], 'a'), f([4..7], 'b'), key)
Traceback (most recent call last):
...
ValueError: Input does not have an overlap.
sage: merge_overlapping(f([4..7], 'a'), f([0..2], 'b'), key)
Traceback (most recent call last):
...
ValueError: Input does not have an overlap.
sage: merge_overlapping(f([0..3], 'a'), f([3..4], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'a'), (3, 'a'), (4, 'b')],
 [(0, 'a'), (1, 'a'), (2, 'a'), (3, 'b'), (4, 'b')])
```

```
sage: merge_overlapping(f([3..4], 'a'), f([0..3], 'b'), key)
([(0, 'b'), (1, 'b'), (2, 'b'), (3, 'a'), (4, 'a')],
 [(0, 'b'), (1, 'b'), (2, 'b'), (3, 'b'), (4, 'a')])
sage: merge_overlapping(f([0..1], 'a'), f([0..4], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'b'), (3, 'b'), (4, 'b')],
 [(0, 'b'), (1, 'b'), (2, 'b'), (3, 'b'), (4, 'b')])
sage: merge_overlapping(f([0..3], 'a'), f([0..1], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'a'), (3, 'a')],
 [(0, 'b'), (1, 'b'), (2, 'a'), (3, 'a')])
sage: merge_overlapping(f([0..3], 'a'), f([1..3], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'a'), (3, 'a')],
 [(0, 'a'), (1, 'b'), (2, 'b'), (3, 'b')])
sage: merge_overlapping(f([1..3], 'a'), f([0..3], 'b'), key)
([(0, 'b'), (1, 'a'), (2, 'a'), (3, 'a')],
 [(0, 'b'), (1, 'b'), (2, 'b'), (3, 'b')])
sage: merge_overlapping(f([0..6], 'a'), f([3..4], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'a'), (3, 'a'), (4, 'a'), (5, 'a'), (6, 'a')],
 [(0, 'a'), (1, 'a'), (2, 'a'), (3, 'b'), (4, 'b'), (5, 'a'), (6, 'a')])
sage: merge_overlapping(f([0..3], 'a'), f([1..2], 'b'), key)
([(0, 'a'), (1, 'a'), (2, 'a'), (3, 'a')],
 [(0, 'a'), (1, 'b'), (2, 'b'), (3, 'a')])
sage: merge_overlapping(f([1..2], 'a'), f([0..3], 'b'), key)
([(0, 'b'), (1, 'a'), (2, 'a'), (3, 'b')],
 [(0, 'b'), (1, 'b'), (2, 'b'), (3, 'b')])
sage: merge_overlapping(f([1..3], 'a'), f([1..3], 'b'), key)
([(1, 'a'), (2, 'a'), (3, 'a')],
 [(1, 'b'), (2, 'b'), (3, 'b')])
```

sage.rings.asymptotic.misc.**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: from sage.rings.asymptotic.misc import parent_to_repr_short
sage: parent_to_repr_short(ZZ)
'ZZ'
sage: parent_to_repr_short(QQ)
'QQ'
sage: parent_to_repr_short(SR)
'SR'
sage: parent_to_repr_short(ZZ['x'])
'ZZ[x]'
sage: parent_to_repr_short(QQ['d, k'])
'QQ[d, k]'
sage: parent_to_repr_short(QQ['e'])
'QQ[e]'
sage: parent_to_repr_short(SR[['a, r']])
'SR[[a, r]]'
sage: parent_to_repr_short(Zmod(3))
'Ring of integers modulo 3'
sage: parent_to_repr_short(Zmod(3)['g'])
'Univariate Polynomial Ring in g over Ring of integers modulo 3'
```

sage.rings.asymptotic.misc.**repr_op**(*left*, *op*, *right=None*, *latex=False*)

Create a string `left op right` with taking care of parentheses in its operands.

INPUT:

- `left` – an element.
- `op` – a string.
- `right` – an alement.
- `latex` – (default: `False`) a boolean. If set, then LaTeX-output is returned.

OUTPUT:

A string.

EXAMPLES:

```
sage: from sage.rings.asymptotic.misc import repr_op
sage: repr_op('a^b', '^', 'c')
'(a^b)^c'
```

TESTS:

```
sage: repr_op('a-b', '^', 'c')
'(a-b)^c'
sage: repr_op('a+b', '^', 'c')
'(a+b)^c'

sage: print repr_op(r'\frac{1}{2}', '^', 'c', latex=True)
\left(\frac{1}{2}\right)^c
```

sage.rings.asymptotic.misc.**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: from sage.rings.asymptotic.misc import repr_short_to_parent
sage: repr_short_to_parent('ZZ')
Integer Ring
sage: repr_short_to_parent('QQ')
Rational Field
sage: repr_short_to_parent('SR')
Symbolic Ring
sage: repr_short_to_parent('NN')
Non negative integer semiring
```

TESTS:

```
sage: repr_short_to_parent('abcdef')
Traceback (most recent call last):
...
ValueError: Cannot create a parent out of 'abcdef'.
> *previous* NameError: name 'abcdef' is not defined
```

sage.rings.asymptotic.misc.**split_str_by_op**(*string*, *op*, *strip_parentheses=True*)

Split the given string into a tuple of substrings arising by splitting by `op` and taking care of parentheses.

INPUT:

- `string` – a string.
- `op` – a string. This is used by str.split. Thus, if this is `None`, then any whitespace string is a separator and empty strings are removed from the result.
- `strip_parentheses` – (default: `True`) a boolean.

OUTPUT:

A tuple of strings.

TESTS:

```
sage: from sage.rings.asymptotic.misc import split_str_by_op
sage: split_str_by_op('x^ZZ', '*')
('x^ZZ',)
sage: split_str_by_op('log(x)^ZZ * y^QQ', '*')
('log(x)^ZZ', 'y^QQ')
sage: split_str_by_op('log(x)**ZZ * y**QQ', '*')
('log(x)**ZZ', 'y**QQ')
sage: split_str_by_op('a^b * * c^d', '*')
Traceback (most recent call last):
...
ValueError: 'a^b * * c^d' is invalid since a '*' follows a '*'.
sage: split_str_by_op('a^b * (c*d^e)', '*')
('a^b', 'c*d^e')

sage: split_str_by_op('(a^b)^c', '^')
('a^b', 'c')
sage: split_str_by_op('a^(b^c)', '^')
('a', 'b^c')

sage: split_str_by_op('(a) + (b)', op='+', strip_parentheses=True)
('a', 'b')
sage: split_str_by_op('(a) + (b)', op='+', strip_parentheses=False)
('(a)', '(b)')
sage: split_str_by_op(' ( t  ) ', op='+', strip_parentheses=False)
('( t  )',)

sage: split_str_by_op(' ( t  ) ', op=None)
('t',)
sage: split_str_by_op(' ( t  )s', op=None)
('(t)s',)
sage: split_str_by_op(' ( t  ) s', op=None)
('t', 's')

sage: split_str_by_op('(e^(n*log(n)))^SR.subring(no_variables=True)', '*')
('(e^(n*log(n)))^SR.subring(no_variables=True)',)
```

sage.rings.asymptotic.misc.**substitute_raise_exception**(*element*, *e*)

Raise an error describing what went wrong with the substitution.

INPUT:

- `element` – an element.
- `e` – an exception which is included in the raised error message.

OUTPUT:

Raise an exception of the same type as `e`.

TESTS:

```
sage: from sage.rings.asymptotic.misc import substitute_raise_exception
sage: substitute_raise_exception(x, Exception('blub'))
Traceback (most recent call last):
...
Exception: Cannot substitute in x in Symbolic Ring.
> *previous* Exception: blub
```

`sage.rings.asymptotic.misc.`**`transform_category`**(*category*, *subcategory_mapping*, *axiom_mapping*, *initial_category=None*)

Transform `category` to a new category according to the given mappings.

INPUT:

- `category` – a category.
- `subcategory_mapping` – a list (or other iterable) of triples `(from, to, mandatory)`, where
  - `from` and `to` are categories and
  - `mandatory` is a boolean.
- `axiom_mapping` – a list (or other iterable) of triples `(from, to, mandatory)`, where
  - `from` and `to` are strings describing axioms and
  - `mandatory` is a boolean.
- `initial_category` – (default: `None`) a category. When transforming the given category, this `initial_category` is used as a starting point of the result. This means the resulting category will be a subcategory of `initial_category`. If `initial_category` is `None`, then the `category of objects` is used.

OUTPUT:

A category.

Note: Consider a subcategory mapping `(from, to, mandatory)`. If `category` is a subcategory of `from`, then the returned category will be a subcategory of `to`. Otherwise and if `mandatory` is set, then an error is raised.

Consider an axiom mapping `(from, to, mandatory)`. If `category` is has axiom `from`, then the returned category will have axiom `to`. Otherwise and if `mandatory` is set, then an error is raised.

EXAMPLES:

```
sage: from sage.rings.asymptotic.misc import transform_category
sage: from sage.categories.additive_semigroups import AdditiveSemigroups
sage: from sage.categories.additive_monoids import AdditiveMonoids
sage: from sage.categories.additive_groups import AdditiveGroups
sage: S = [
....:     (Sets(), Sets(), True),
....:     (Posets(), Posets(), False),
....:     (AdditiveMagmas(), Magmas(), False)]
sage: A = [
....:     ('AdditiveAssociative', 'Associative', False),
....:     ('AdditiveUnital', 'Unital', False),
....:     ('AdditiveInverse', 'Inverse', False),
....:     ('AdditiveCommutative', 'Commutative', False)]
```

```
sage: transform_category(Objects(), S, A)
Traceback (most recent call last):
...
ValueError: Category of objects is not
a subcategory of Category of sets.
sage: transform_category(Sets(), S, A)
Category of sets
sage: transform_category(Posets(), S, A)
Category of posets
sage: transform_category(AdditiveSemigroups(), S, A)
Category of semigroups
sage: transform_category(AdditiveMonoids(), S, A)
Category of monoids
sage: transform_category(AdditiveGroups(), S, A)
Category of groups
sage: transform_category(AdditiveGroups().AdditiveCommutative(), S, A)
Category of commutative groups

sage: transform_category(AdditiveGroups().AdditiveCommutative(), S, A,
....:     initial_category=Posets())
Join of Category of commutative groups
    and Category of posets

sage: transform_category(ZZ.category(), S, A)
Category of commutative groups
sage: transform_category(QQ.category(), S, A)
Category of commutative groups
sage: transform_category(SR.category(), S, A)
Category of commutative groups
sage: transform_category(Fields(), S, A)
Category of commutative groups
sage: transform_category(ZZ['t'].category(), S, A)
Category of commutative groups

sage: A[-1] = ('Commutative', 'AdditiveCommutative', True)
sage: transform_category(Groups(), S, A)
Traceback (most recent call last):
...
ValueError: Category of groups does not have
axiom Commutative.
```

sage.rings.asymptotic.misc.**underlying_class**(*P*)

Return the underlying class (class without the attached categories) of the given instance.

OUTPUT:

A class.

EXAMPLES:

```
sage: from sage.rings.asymptotic.misc import underlying_class
sage: type(QQ)
<class 'sage.rings.rational_field.RationalField_with_category'>
sage: underlying_class(QQ)
<class 'sage.rings.rational_field.RationalField'>
```

4.7 Asymptotics of Multivariate Generating Series

Let $F(x) = \sum_{\nu \in \mathbf{N}^d} F_\nu x^\nu$ be a multivariate power series with complex coefficients that converges in a neighborhood of the origin. Assume that $F = G/H$ for some functions G and H holomorphic in a neighborhood of the origin. Assume also that H is a polynomial.

This computes asymptotics for the coefficients $F_{r\alpha}$ as $r \to \infty$ with $r\alpha \in \mathbf{N}^d$ for α in a permissible subset of d-tuples of positive reals. More specifically, it computes arbitrary terms of the asymptotic expansion for $F_{r\alpha}$ when the asymptotics are controlled by a strictly minimal multiple point of the alegbraic variety $H = 0$.

The algorithms and formulas implemented here come from [RaWi2008a] and [RaWi2012]. For a general reference take a look in the book [PeWi2013].

4.7.1 Introductory Examples

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFac
```

A univariate smooth point example:

```
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (x - 1/2)^3
sage: Hfac = H.factor()
sage: G = -1/(x + 3)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(-1/(x + 3), [(x - 1/2, 3)])
sage: alpha = [1]
sage: decomp = F.asymptotic_decomposition(alpha)
sage: decomp
(0, []) +
(-1/2*(x^2 + 6*x + 9)*r^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)
 - 1/2*(5*x^2 + 24*x + 27)*r/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)
 - 3*(x^2 + 3*x + 3)/(x^5 + 9*x^4 + 27*x^3 + 27*x^2),
 [(x - 1/2, 1)])
sage: F1 = decomp[1]
sage: p = {x: 1/2}
sage: asy = F1.asymptotics(p, alpha, 3)
sage: asy
(8/343*(49*r^2 + 161*r + 114)*2^r, 2, 8/7*r^2 + 184/49*r + 912/343)
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
[((1,), 7.555555556, [7.556851312], [-0.0001714971672]),
 ((2,), 14.74074074, [14.74052478], [0.00001465051901]),
 ((4,), 35.96502058, [35.96501458], [1.667911934e-7]),
 ((8,), 105.8425656, [105.8425656], [4.399565380e-11]),
 ((16,), 355.3119534, [355.3119534], [0.0000000000])]
```

Another smooth point example (Example 5.4 of [RaWi2008a]):

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: q = 1/2
sage: qq = q.denominator()
sage: H = 1 - q*x + q*x*y - x^2*y
sage: Hfac = H.factor()
sage: G = (1 - q*x)/Hfac.unit()
```

```
sage: F = FFPD(G, Hfac)
sage: alpha = list(qq*vector([2, 1 - q]))
sage: alpha
[4, 1]
sage: I = F.smooth_critical_ideal(alpha)
sage: I
Ideal (y^2 - 2*y + 1, x + 1/4*y - 5/4) of
 Multivariate Polynomial Ring in x, y over Rational Field
sage: s = solve([SR(z) for z in I.gens()],
....:           [SR(z) for z in R.gens()], solution_dict=true)
sage: s == [{SR(x): 1, SR(y): 1}]
True
sage: p = s[0]
sage: asy = F.asymptotics(p, alpha, 1, verbose=True)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
sage: asy
(1/24*2^(2/3)*(sqrt(3) + 4/(sqrt(3) + I) + I)*gamma(1/3)/(pi*r^(1/3)),
 1,
 1/24*2^(2/3)*(sqrt(3) + 4/(sqrt(3) + I) + I)*gamma(1/3)/(pi*r^(1/3)))
sage: r = SR('r')
sage: tuple((a*r^(1/3)).full_simplify() / r^(1/3) for a in asy)  # make nicer coefficients
(1/12*sqrt(3)*2^(2/3)*gamma(1/3)/(pi*r^(1/3)),
 1,
 1/12*sqrt(3)*2^(2/3)*gamma(1/3)/(pi*r^(1/3)))
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
[((4, 1), 0.1875000000, [0.1953794675...], [-0.042023826...]),
 ((8, 2), 0.1523437500, [0.1550727862...], [-0.017913673...]),
 ((16, 4), 0.1221771240, [0.1230813519...], [-0.0074009592...]),
 ((32, 8), 0.09739671811, [0.09768973377...], [-0.0030084757...]),
 ((64, 16), 0.07744253816, [0.07753639308...], [-0.0012119297...])]
```

A multiple point example (Example 6.5 of [RaWi2012]):

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - 2*x - y)**2 * (1 - x - 2*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(1, [(x + 2*y - 1, 2), (2*x + y - 1, 2)])
sage: I = F.singular_ideal()
sage: I
Ideal (x - 1/3, y - 1/3) of
 Multivariate Polynomial Ring in x, y over Rational Field
sage: p = {x: 1/3, y: 1/3}
sage: F.is_convenient_multiple_point(p)
(True, 'convenient in variables [x, y]')
sage: alpha = (var('a'), var('b'))
sage: decomp =  F.asymptotic_decomposition(alpha); decomp
(0, []) +
(-1/9*(2*b^2*x^2 - 5*a*b*x*y + 2*a^2*y^2)*r^2/(x^2*y^2)
  - 1/9*(6*b*x^2 - 5*(a + b)*x*y + 6*a*y^2)*r/(x^2*y^2)
  - 1/9*(4*x^2 - 5*x*y + 4*y^2)/(x^2*y^2),
 [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
```

```
sage: F1 = decomp[1]
sage: F1.asymptotics(p, alpha, 2)
(-3*((2*a^2 - 5*a*b + 2*b^2)*r^2 + (a + b)*r + 3)*((1/3)^(-a)*(1/3)^(-b))^r,
 (1/3)^(-a)*(1/3)^(-b), -3*(2*a^2 - 5*a*b + 2*b^2)*r^2 - 3*(a + b)*r - 9)
sage: alpha = [4, 3]
sage: decomp =  F.asymptotic_decomposition(alpha)
sage: F1 = decomp[1]
sage: asy = F1.asymptotics(p, alpha, 2)
sage: asy
(3*(10*r^2 - 7*r - 3)*2187^r, 2187, 30*r^2 - 21*r - 9)
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8], asy[1])
[((4, 3), 30.72702332, [0.0000000000], [1.000000000]),
 ((8, 6), 111.9315678, [69.00000000], [0.3835519207]),
 ((16, 12), 442.7813138, [387.0000000], [0.1259793763]),
 ((32, 24), 1799.879232, [1743.000000], [0.03160169385])]
```

TESTS:

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (1 - 2*x - y) * (1 - x - 2*y)
sage: G = 1
sage: Hfac = H.factor()
sage: G = G / Hfac.unit()
sage: F = FFPD(G, Hfac); F
(1, [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
sage: p = {x: 1, y: 1}
sage: alpha = [1, 1]
sage: F.asymptotics(p, alpha, 1)
(1/3, 1, 1/3)
```

```
sage: R.<x,y,t> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (1 - y) * (1 + x^2) * (1 - t*(1 + x^2 + x*y^2))
sage: G = (1 + x) * (1 + x^2 - x*y^2)
sage: Hfac = H.factor()
sage: G = G / Hfac.unit()
sage: F = FFPD(G, Hfac); F
(-x^2*y^2 + x^3 - x*y^2 + x^2 + x + 1,
 [(y - 1, 1), (x^2 + 1, 1), (x*y^2*t + x^2*t + t - 1, 1)])
sage: p = {x: 1, y: 1, t: 1/3}
sage: alpha = [1, 1, 1]
sage: F.asymptotics_multiple(p, alpha, 1, var('r'))  # not tested - see #19989
```

4.7.2 Various

AUTHORS:

- Alexander Raichev (2008)
- Daniel Krenn (2014, 2016)

4.7.3 Classes and Methods

class `sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`FractionWithFactore`**

Bases: `sage.structure.element.RingElement`

This element represents a fraction with a factored polynomial denominator. See also its parent `FractionWithFactoredDenominatorRing` for details.

Represents a fraction with factored polynomial denominator (FFPD) $p/(q_1^{e_1} \cdots q_n^{e_n})$ by storing the parts p and $[(q_1, e_1), \ldots, (q_n, e_n)]$. Here $q_1, \ldots, q_n$ are elements of a 0- or multi-variate factorial polynomial ring R , $q_1, \ldots, q_n$ are distinct irreducible elements of R , $e_1, \ldots, e_n$ are positive integers, and p is a function of the indeterminates of R (e.g., a Sage symbolic expression). An element r with no polynomial denominator is represented as `(r, [])`.

INPUT:

- `numerator` – an element p; this can be of any ring from which parent's base has coercion in
- `denominator_factored` – a list of the form $[(q_1, e_1), \ldots, (q_n, e_n)]$, where the $q_1, \ldots, q_n$ are distinct irreducible elements of R and the e_i are positive integers
- `reduce` – (optional) if `True`, then represent $p/(q_1^{e_1} \cdots q_n^{e_n})$ in lowest terms, otherwise this won't attempt to divide p by any of the q_i

OUTPUT:

An element representing the rational expression $p/(q_1^{e_1} \cdots q_n^{e_n})$.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWi
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: df = [x, 1], [y, 1], [x*y+1, 1]
sage: f = FFPD(x, df)
sage: f
(1, [(y, 1), (x*y + 1, 1)])
sage: ff = FFPD(x, df, reduce=False)
sage: ff
(x, [(y, 1), (x, 1), (x*y + 1, 1)])

sage: f = FFPD(x + y, [(x + y, 1)])
sage: f
(1, [])

sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: FFPD(f)
```

```
(5*x^7 - 5*x^6 + 5/3*x^5 - 5/3*x^4 + 2*x^3 - 2/3*x^2 + 1/3*x - 1/3,
[(x - 1, 1), (x, 1), (x^2 + 1/3, 1)])

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = 2*y/(5*(x^3 - 1)*(y + 1))
sage: FFPD(f)
(2/5*y, [(y + 1, 1), (x - 1, 1), (x^2 + x + 1, 1)])

sage: p = 1/x^2
sage: q = 3*x**2*y
sage: qs = q.factor()
sage: f = FFPD(p/qs.unit(), qs)
sage: f
(1/3/x^2, [(y, 1), (x, 2)])

sage: f = FFPD(cos(x)*x*y^2, [(x, 2), (y, 1)])
sage: f
(x*y^2*cos(x), [(y, 1), (x, 2)])

sage: G = exp(x + y)
sage: H = (1 - 2*x - y) * (1 - x - 2*y)
sage: a = FFPD(G/H)
sage: a
(e^(x + y), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
sage: a.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
sage: b = FFPD(G, H.factor())
sage: b
(e^(x + y), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
sage: b.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
```

Singular throws a 'not implemented' error when trying to factor in a multivariate polynomial ring over an inexact field:

```
sage: R.<x,y> = PolynomialRing(CC)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = (x + 1)/(x*y*(x*y + 1)^2)
sage: FFPD(f)
Traceback (most recent call last):
...
TypeError: Singular error:
   ? not implemented
   ? error occurred in or before STDIN line ...:
   `def sage...=factorize(sage...);`
```

AUTHORS:

- Alexander Raichev (2012-07-26)
- Daniel Krenn (2014-12-01)

algebraic_dependence_certificate()

Return the algebraic dependence certificate of `self`.

The algebraic dependence certificate is the ideal J of annihilating polynomials for the set of polynomials `[q^e for (q, e) in self.denominator_factored()]`, which could be the zero ideal. The ideal J lies in a polynomial ring over the field `self.denominator_ring.base_ring()` that has `m = len(self.denominator_factored())` indeterminates.

OUTPUT:

An ideal.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/(x^2 * (x*y + 1) * y^3)
sage: ff = FFPD(f)
sage: J = ff.algebraic_dependence_certificate(); J
Ideal (1 - 6*T2 + 15*T2^2 - 20*T2^3 + 15*T2^4 - T0^2*T1^3 -
 6*T2^5  + T2^6) of Multivariate Polynomial Ring in
 T0, T1, T2 over Rational Field
sage: g = J.gens()[0]
sage: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == 0
True
```

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: G = exp(x + y)
sage: H = x^2 * (x*y + 1) * y^3
sage: ff = FFPD(G, H.factor())
sage: J = ff.algebraic_dependence_certificate(); J
Ideal (1 - 6*T2 + 15*T2^2 - 20*T2^3 + 15*T2^4 - T0^2*T1^3 -
6*T2^5 + T2^6) of Multivariate Polynomial Ring in
T0, T1, T2 over Rational Field
sage: g = J.gens()[0]
sage: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == 0
True
```

```
sage: f = 1/(x^3 * y^2)
sage: J = FFPD(f).algebraic_dependence_certificate()
sage: J
Ideal (0) of Multivariate Polynomial Ring in T0, T1 over Rational Field
```

```
sage: f = sin(1)/(x^3 * y^2)
sage: J = FFPD(f).algebraic_dependence_certificate()
sage: J
Ideal (0) of Multivariate Polynomial Ring in T0, T1 over Rational Field
```

algebraic_dependence_decomposition(*whole_and_parts=True*)

Return an algebraic dependence decomposition of `self`.

Let $f = p/q$ where q lies in a d-variate polynomial ring $K[X]$ for some field K. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d \mid q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

$$(*) \quad \sum_A \frac{p_A}{\prod_{i \in A} q_i^{b_i}},$$

where the b_i are positive integers, each p_A is a products of p and an element in $K[X]$, and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that $|A| \leq d$ and $\{q_i \mid i \in A\}$ is algebraically independent.

We call $(*)$ an *algebraic dependence decomposition* of f. Algebraic dependence decompositions are not unique.

The algorithm used comes from [Raic2012].

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/(x^2 * (x*y + 1) * y^3)
sage: ff = FFPD(f)
sage: decomp = ff.algebraic_dependence_decomposition()
sage: decomp
(0, []) + (-x, [(x*y + 1, 1)]) +
(x^2*y^2 - x*y + 1, [(y, 3), (x, 2)])
sage: decomp.sum().quotient() == f
True
sage: for r in decomp:
....:     J = r.algebraic_dependence_certificate()
....:     J is None or J == J.ring().ideal()  # The zero ideal
True
True
True

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: G = sin(x)
sage: H = x^2 * (x*y + 1) * y^3
sage: f = FFPD(G, H.factor())
sage: decomp = f.algebraic_dependence_decomposition()
sage: decomp
(0, []) + (x^4*y^3*sin(x), [(x*y + 1, 1)]) +
(-(x^5*y^5 - x^4*y^4 + x^3*y^3 - x^2*y^2 + x*y - 1)*sin(x),
 [(y, 3), (x, 2)])
sage: bool(decomp.sum().quotient() == G/H)
True
sage: for r in decomp:
....:     J = r.algebraic_dependence_certificate()
....:     J is None or J == J.ring().ideal()
True
True
True
```

asymptotic_decomposition(*alpha*, *asy_var=None*)

Return the asymptotic decomposition of `self`.

The asymptotic decomposition of F is a sum that has the same asymptotic expansion as f in the direction `alpha` but each summand has a denominator factorization of the form $[(q_1, 1), \ldots, (q_n, 1)]$, where n is at most the `dimension()` of F.

INPUT:

- `alpha` – a d-tuple of positive integers or symbolic variables
- `asy_var` – (default: `None`) a symbolic variable with respect to which to compute asymptotics; if `None` is given, we set `asy_var = var('r')`

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

The output results from a Leinartas decomposition followed by a cohomology decomposition.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = (x^2 + 1)/((x - 1)^3*(x + 2))
sage: F = FFPD(f)
sage: alpha = [var('a')]
sage: F.asymptotic_decomposition(alpha)
(0, []) +
(1/54*(5*a^2*x^2 + 2*a^2*x + 11*a^2)*r^2/x^2
 - 1/54*(5*a*x^2 - 2*a*x - 33*a)*r/x^2 + 11/27/x^2, [(x - 1, 1)]) +
(-5/27, [(x + 2, 1)])

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - 2*x -y)*(1 - x -2*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = var('a, b')
sage: F.asymptotic_decomposition(alpha)
(0, []) +
(1/3*(2*b*x - a*y)*r/(x*y) + 1/3*(2*x - y)/(x*y),
 [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
```

asymptotics(*p*, *alpha*, *N*, *asy_var=None*, *numerical=0*, *verbose=False*)

Return the asymptotics in the given direction.

This function returns the first N terms (some of which could be zero) of the asymptotic expansion of the Maclaurin ray coefficients $F_{r\alpha}$ of the function F represented by `self` as $r \to \infty$, where r is `asy_var` and `alpha` is a tuple of positive integers of length d which is `self.dimension()`. Assume that

- F is holomorphic in a neighborhood of the origin;
- the unique factorization of the denominator H of F in the local algebraic ring at p equals its unique factorization in the local analytic ring at p;
- the unique factorization of H in the local algebraic ring at p has at most `d` irreducible factors, none of which are repeated (one can reduce to this case via `asymptotic_decomposition()`);
- p is a convenient strictly minimal smooth or multiple point with all nonzero coordinates that is critical and nondegenerate for `alpha`.

The algorithms used here come from [RaWi2008a] and [RaWi2012].

INPUT:

- `p` – a dictionary with keys that can be coerced to equal `self.denominator_ring.gens()`
- `alpha` – a tuple of length `self.dimension()` of positive integers or, if p is a smooth point, possibly of symbolic variables
- `N` – a positive integer
- `asy_var` – (default: `None`) a symbolic variable for the asymptotic expansion; if `none` is given, then `var('r')` will be assigned
- `numerical` – (default: 0) a natural number; if `numerical` is greater than 0, then return a numerical approximation of $F_{r\alpha}$ with `numerical` digits of precision; otherwise return exact values

- verbose – (default: False) print the current state of the algorithm

OUTPUT:

The tuple (asy, exp_scale, subexp_part). Here asy is the sum of the first N terms (some of which might be 0) of the asymptotic expansion of $F_{r\alpha}$ as $r \to \infty$; exp_scale**r is the exponential factor of asy; subexp_part is the subexponential factor of asy.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
```

A smooth point example:

```
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac); print(F)
(1, [(x*y + x + y - 1, 2)])
sage: alpha = [4, 3]
sage: decomp = F.asymptotic_decomposition(alpha); decomp
(0, []) + (-3/2*r*(y + 1)/y - 1/2*(y + 1)/y, [(x*y + x + y - 1, 1)])
sage: F1 = decomp[1]
sage: p = {y: 1/3, x: 1/2}
sage: asy = F1.asymptotics(p, alpha, 2, verbose=True)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
sage: asy
(1/6000*(3600*sqrt(5)*sqrt(3)*sqrt(2)*sqrt(r)/sqrt(pi)
  + 463*sqrt(5)*sqrt(3)*sqrt(2)/(sqrt(pi)*sqrt(r)))*432^r,
 432,
 3/5*sqrt(5)*sqrt(3)*sqrt(2)*sqrt(r)/sqrt(pi)
  + 463/6000*sqrt(5)*sqrt(3)*sqrt(2)/(sqrt(pi)*sqrt(r)))
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
[((4, 3), 2.083333333, [2.092576110], [-0.0044365330...]),
 ((8, 6), 2.787374614, [2.790732875], [-0.0012048112...]),
 ((16, 12), 3.826259447, [3.827462310], [-0.0003143703...]),
 ((32, 24), 5.328112821, [5.328540787], [-0.0000803222...]),
 ((64, 48), 7.475927885, [7.476079664], [-0.0000203023...])]
```

A multiple point example:

```
sage: R.<x,y,z>= PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (4 - 2*x - y - z)**2*(4 - x - 2*y - z)
sage: Hfac = H.factor()
sage: G = 16/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(-16, [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 2)])
sage: alpha = [3, 3, 2]
sage: decomp = F.asymptotic_decomposition(alpha); decomp
(0, []) +
(16*r*(4*y - 3*z)/(y*z) + 16*(2*y - z)/(y*z),
 [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 1)])
sage: F1 = decomp[1]
sage: p = {x: 1, y: 1, z: 1}
```

```
sage: asy = F1.asymptotics(p, alpha, 2, verbose=True) # long time
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
sage: asy # long time
(4/3*sqrt(3)*sqrt(r)/sqrt(pi) + 47/216*sqrt(3)/(sqrt(pi)*sqrt(r)),
 1, 4/3*sqrt(3)*sqrt(r)/sqrt(pi) + 47/216*sqrt(3)/(sqrt(pi)*sqrt(r)))
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8], asy[1]) # long time
[((3, 3, 2), 0.9812164307, [1.515572606], [-0.54458543...]),
 ((6, 6, 4), 1.576181132, [1.992989399], [-0.26444185...]),
 ((12, 12, 8), 2.485286378, [2.712196351], [-0.091301338...]),
 ((24, 24, 16), 3.700576827, [3.760447895], [-0.016178847...])]
```

asymptotics_multiple(*p*, *alpha*, *N*, *asy_var*, *coordinate=None*, *numerical=0*, *verbose=False*)

Return the asymptotics in the given direction of a multiple point nondegenerate for `alpha`.

This is the same as `asymptotics()`, but only in the case of a convenient multiple point nondegenerate for `alpha`. Assume also that `self.dimension >= 2` and that the `p.values()` are not symbolic variables.

The formulas used for computing the asymptotic expansion are Theorem 3.4 and Theorem 3.7 of [RaWi2012].

INPUT:

- `p` – a dictionary with keys that can be coerced to equal `self.denominator_ring.gens()`
- `alpha` – a tuple of length `d = self.dimension()` of positive integers or, if p is a smooth point, possibly of symbolic variables
- `N` – a positive integer
- `asy_var` – (optional; default: `None`) a symbolic variable; the variable of the asymptotic expansion, if none is given, `var('r')` will be assigned
- `coordinate` – (optional; default: `None`) an integer in $\{0, \dots, d-1\}$ indicating a convenient coordinate to base the asymptotic calculations on; if `None` is assigned, then choose `coordinate=d-1`
- `numerical` – (optional; default: 0) a natural number; if numerical is greater than 0, then return a numerical approximation of the Maclaurin ray coefficients of `self` with `numerical` digits of precision; otherwise return exact values
- `verbose` – (default: `False`) print the current state of the algorithm

OUTPUT:

The asymptotic expansion.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y,z>= PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (4 - 2*x - y - z)*(4 - x -2*y - z)
sage: Hfac = H.factor()
sage: G = 16/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(16, [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 1)])
sage: p = {x: 1, y: 1, z: 1}
sage: alpha = [3, 3, 2]
```

```
sage: F.asymptotics_multiple(p, alpha, 2, var('r'), verbose=True) # long time
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
(4/3*sqrt(3)/(sqrt(pi)*sqrt(r)) - 25/216*sqrt(3)/(sqrt(pi)*r^(3/2)),
 1,
 4/3*sqrt(3)/(sqrt(pi)*sqrt(r)) - 25/216*sqrt(3)/(sqrt(pi)*r^(3/2)))

sage: H = (1 - x*(1 + y))*(1 - z*x**2*(1 + 2*y))
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(1, [(x*y + x - 1, 1), (2*x^2*y*z + x^2*z - 1, 1)])
sage: p = {x: 1/2, z: 4/3, y: 1}
sage: alpha = [8, 3, 3]
sage: F.asymptotics_multiple(p, alpha, 2, var('r'), coordinate=1, verbose=True) # long time
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second-order differential operator actions...
(1/172872*108^r*(24696*sqrt(7)*sqrt(3)/(sqrt(pi)*sqrt(r))
  - 1231*sqrt(7)*sqrt(3)/(sqrt(pi)*r^(3/2))),
 108,
 1/7*sqrt(7)*sqrt(3)/(sqrt(pi)*sqrt(r))
  - 1231/172872*sqrt(7)*sqrt(3)/(sqrt(pi)*r^(3/2)))

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - 2*x - y) * (1 - x - 2*y)
sage: Hfac = H.factor()
sage: G = exp(x + y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(e^(x + y), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])
sage: p = {x: 1/3, y: 1/3}
sage: alpha = (var('a'), var('b'))
sage: F.asymptotics_multiple(p, alpha, 2, var('r')) # long time
(3*((1/3)^(-a)*(1/3)^(-b))^r*e^(2/3), (1/3)^(-a)*(1/3)^(-b), 3*e^(2/3))
```

asymptotics_smooth(*p*, *alpha*, *N*, *asy_var*, *coordinate=None*, *numerical=0*, *verbose=False*)

Return the asymptotics in the given direction of a smooth point.

This is the same as `asymptotics()`, but only in the case of a convenient smooth point.

The formulas used for computing the asymptotic expansions are Theorems 3.2 and 3.3 [RaWi2008a] with the exponent of H equal to 1. Theorem 3.2 is a specialization of Theorem 3.4 of [RaWi2012] with $n = 1$.

INPUT:

- `p` – a dictionary with keys that can be coerced to equal `self.denominator_ring.gens()`
- `alpha` – a tuple of length `d = self.dimension()` of positive integers or, if p is a smooth point, possibly of symbolic variables
- `N` – a positive integer
- `asy_var` – (optional; default: `None`) a symbolic variable; the variable of the asymptotic expansion, if none is given, `var('r')` will be assigned

- coordinate – (optional; default: None) an integer in $\{0, \dots, d-1\}$ indicating a convenient coordinate to base the asymptotic calculations on; if None is assigned, then choose coordinate=d-1
- numerical – (optional; default: 0) a natural number; if numerical is greater than 0, then return a numerical approximation of the Maclaurin ray coefficients of self with numerical digits of precision; otherwise return exact values
- verbose – (default: False) print the current state of the algorithm

OUTPUT:

The asymptotic expansion.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = 2 - 3*x
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(-1/3, [(x - 2/3, 1)])
sage: alpha = [2]
sage: p = {x: 2/3}
sage: asy = F.asymptotics_smooth(p, alpha, 3, asy_var=var('r'))
sage: asy
(1/2*(9/4)^r, 9/4, 1/2)

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = 1-x-y-x*y
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [3, 2]
sage: p = {y: 1/2*sqrt(13) - 3/2, x: 1/3*sqrt(13) - 2/3}
sage: F.asymptotics_smooth(p, alpha, 2, var('r'), numerical=3, verbose=True)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
(71.2^r*(0.369/sqrt(r) - 0.018.../r^(3/2)), 71.2, 0.369/sqrt(r) - 0.018.../r^(3/2))

sage: q = 1/2
sage: qq = q.denominator()
sage: H = 1 - q*x + q*x*y - x^2*y
sage: Hfac = H.factor()
sage: G = (1 - q*x)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = list(qq*vector([2, 1 - q]))
sage: alpha
[4, 1]
sage: p = {x: 1, y: 1}
sage: F.asymptotics_smooth(p, alpha, 5, var('r'), verbose=True) # not tested (140 seconds)
Creating auxiliary functions...
Computing derivatives of auxiallary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
(1/12*sqrt(3)*2^(2/3)*gamma(1/3)/(pi*r^(1/3))
```

```
 - 1/96*sqrt(3)*2^(1/3)*gamma(2/3)/(pi*r^(5/3)),
1,
1/12*sqrt(3)*2^(2/3)*gamma(1/3)/(pi*r^(1/3))
 - 1/96*sqrt(3)*2^(1/3)*gamma(2/3)/(pi*r^(5/3)))
```

cohomology_decomposition()

Return the cohomology decomposition of `self`.

Let $p/(q_1^{e_1}\cdots q_n^{e_n})$ be the fraction represented by `self` and let $K[x_1,\ldots,x_d]$ be the polynomial ring in which the q_i lie. Assume that $n \leq d$ and that the gradients of the q_i are linearly independent at all points in the intersection $V_1 \cap \ldots \cap V_n$ of the algebraic varieties $V_i = \{x \in L^d \mid q_i(x) = 0\}$, where L is the algebraic closure of the field K. Return a `FractionWithFactoredDenominatorSum` f such that the differential form $f dx_1 \wedge \cdots \wedge dx_d$ is de Rham cohomologous to the differential form $p/(q_1^{e_1}\cdots q_n^{e_n}) dx_1 \wedge \cdots \wedge dx_d$ and such that the denominator of each summand of f contains no repeated irreducible factors.

The algorithm used here comes from the proof of Theorem 17.4 of [AiYu1983].

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/(x^2 + x + 1)^3
sage: decomp = FFPD(f).cohomology_decomposition()
sage: decomp
(0, []) + (2/3, [(x^2 + x + 1, 1)])

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: FFPD(1, [(x, 1), (y, 2)]).cohomology_decomposition()
(0, [])

sage: p = 1
sage: qs = [(x*y - 1, 1), (x**2 + y**2 - 1, 2)]
sage: f = FFPD(p, qs)
sage: f.cohomology_decomposition()
(0, []) + (4/3*x*y + 4/3, [(x^2 + y^2 - 1, 1)]) +
(1/3, [(x*y - 1, 1), (x^2 + y^2 - 1, 1)])
```

critical_cone(*p*, *coordinate=None*)

Return the critical cone of the convenient multiple point `p`.

INPUT:

- `p` – a dictionary with keys that can be coerced to equal `self.denominator_ring.gens()` and values in a field
- `coordinate` – (optional; default: `None`) a natural number

OUTPUT:

A list of vectors.

This list of vectors generate the critical cone of `p` and the cone itself, which is `None` if the values of `p` don't lie in $\mathbf{Q}$. Divide logarithmic gradients by their component `coordinate` entries. If `coordinate = None`, then search from $d-1$ down to 0 for the first index `j` such that for all `i` we have `self.log_grads()[i][j] != 0` and set `coordinate = j`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y,z> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: G = 1
sage: H = (1 - x*(1 + y)) * (1 - z*x**2*(1 + 2*y))
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: p = {x: 1/2, y: 1, z: 4/3}
sage: F.critical_cone(p)
([(2, 1, 0), (3, 1, 3/2)], 2-d cone in 3-d lattice N)
```

denominator()

Return the denominator of `self`.

OUTPUT:

The denominator (i.e., the product of the factored denominator).

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F.denominator()
x^3*y^2 + 2*x^3*y + x^2*y^2 + x^3 - 2*x^2*y - x*y^2 - 3*x^2 - 2*x*y
- y^2 + 3*x + 2*y - 1
```

denominator_factored()

Return the factorization in `self.denominator_ring` of the denominator of `self` but without the unit part.

OUTPUT:

The factored denominator as a list of tuple `(f, m)`, where f is a factor and m its multiplicity.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F.denominator_factored()
[(x - 1, 1), (x*y + x + y - 1, 2)]
```

denominator_ring

Return the ring of the denominator.

OUTPUT:

A ring.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
sage: F = FFPD(G/H)
sage: F
(e^y, [(x - 1, 1), (x*y + x + y - 1, 2)])
sage: F.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
```

dimension()

Return the number of indeterminates of `self.denominator_ring`.

OUTPUT:

An integer.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F.dimension()
2
```

grads(*p*)

Return a list of the gradients of the polynomials `[q for (q, e) in self.denominator_factored()]` evalutated at `p`.

INPUT:

- `p` – (optional; default: `None`) a dictionary whose keys are the generators of `self.denominator_ring`

OUTPUT:

A list.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: p = exp(x)
sage: df = [(x^3 + 3*y^2, 5), (x*y, 2), (y, 1)]
sage: f = FFPD(p, df)
sage: f
(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])
sage: R.gens()
(x, y)
sage: p = None
sage: f.grads(p)
```

```
[(0, 1), (y, x), (3*x^2, 6*y)]

sage: p = {x: sqrt(2), y: var('a')}
sage: f.grads(p)
[(0, 1), (a, sqrt(2)), (6, 6*a)]
```

is_convenient_multiple_point (*p*)

Tests if p is a convenient multiple point of self.

In case p is a convenient multiple point, verdict = True and comment is a string stating which variables it's convenient to use. In case p is not, verdict = False and comment is a string explaining why p fails to be a convenient multiple point.

See [RaWi2012] for more details.

INPUT:

- p – a dictionary with keys that can be coerced to equal self.denominator_ring.gens()

OUTPUT:

A pair (verdict, comment).

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y,z> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (1 - x*(1 + y)) * (1 - z*x**2*(1 + 2*y))
sage: df = H.factor()
sage: G = 1 / df.unit()
sage: F = FFPD(G, df)
sage: p1 = {x: 1/2, y: 1, z: 4/3}
sage: p2 = {x: 1, y: 2, z: 1/2}
sage: F.is_convenient_multiple_point(p1)
(True, 'convenient in variables [x, y]')
sage: F.is_convenient_multiple_point(p2)
(False, 'not a singular point')
```

leinartas_decomposition ()

Return a Leinartas decomposition of self.

Let $f = p/q$ where q lies in a d -variate polynomial ring $K[X]$ for some field K. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d \mid q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

$$(*) \quad \sum_A \frac{p_A}{\prod_{i \in A} q_i^{b_i}},$$

where the b_i are positive integers, each p_A is a product of p and an element of $K[X]$, and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that

1. $|A| \leq d$,
2. $\bigcap_{i \in A} T_i \neq \emptyset$, and
3. $\{q_i \mid i \in A\}$ is algebraically independent.

In particular, any rational expression in d variables can be represented as a sum of rational expressions whose denominators each contain at most d distinct irreducible factors.

We call $(*)$ a *Leinartas decomposition* of f. Leinartas decompositions are not unique.

The algorithm used comes from [Raic2012].

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = (x^2 + 1)/((x + 2)*(x - 1)*(x^2 + x + 1))
sage: decomp = FFPD(f).leinartas_decomposition()
sage: decomp
(0, []) + (2/9, [(x - 1, 1)]) +
(-5/9, [(x + 2, 1)]) + (1/3*x, [(x^2 + x + 1, 1)])
sage: decomp.sum().quotient() == f
True

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/x + 1/y + 1/(x*y + 1)
sage: decomp = FFPD(f).leinartas_decomposition()
sage: decomp
(0, []) + (1, [(x*y + 1, 1)]) + (x + y, [(y, 1), (x, 1)])
sage: decomp.sum().quotient() == f
True
sage: def check_decomp(r):
....:     L = r.nullstellensatz_certificate()
....:     J = r.algebraic_dependence_certificate()
....:     return L is None and (J is None or J == J.ring().ideal())
sage: all(check_decomp(r) for r in decomp)
True

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = sin(x)/x + 1/y + 1/(x*y + 1)
sage: G = f.numerator()
sage: H = R(f.denominator())
sage: ff = FFPD(G, H.factor())
sage: decomp = ff.leinartas_decomposition()
sage: decomp
(0, []) +
(-(x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x)*y, [(y, 1)]) +
((x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x)*x*y, [(x*y + 1, 1)]) +
(x*y^2*sin(x) + x^2*y + x*y + y*sin(x) + x, [(y, 1), (x, 1)])
sage: bool(decomp.sum().quotient() == f)
True
sage: all(check_decomp(r) for r in decomp)
True

sage: R.<x,y,z>= PolynomialRing(GF(2, 'a'))
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/(x * y * z * (x*y + z))
sage: decomp = FFPD(f).leinartas_decomposition()
sage: decomp
(0, []) + (1, [(z, 2), (x*y + z, 1)]) +
(1, [(z, 2), (y, 1), (x, 1)])
sage: decomp.sum().quotient() == f
True
```

log_grads (*p*)

Return a list of the logarithmic gradients of the polynomials `[q for (q, e) in self.denominator_factored()]` evalutated at `p`.

The logarithmic gradient of a function f at point p is the vector $(x_1 \partial_1 f(x), \ldots, x_d \partial_d f(x))$ evaluated at p.

INPUT:

- `p` – (optional; default: `None`) a dictionary whose keys are the generators of `self.denominator_ring`

OUTPUT:

A list.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: p = exp(x)
sage: df = [(x^3 + 3*y^2, 5), (x*y, 2), (y, 1)]
sage: f = FFPD(p, df)
sage: f
(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])
sage: R.gens()
(x, y)
sage: p = None
sage: f.log_grads(p)
[(0, y), (x*y, x*y), (3*x^3, 6*y^2)]

sage: p = {x: sqrt(2), y: var('a')}
sage: f.log_grads(p)
[(0, a), (sqrt(2)*a, sqrt(2)*a), (6*sqrt(2), 6*a^2)]
```

maclaurin_coefficients (*multi_indices*, *numerical=0*)

Return the Maclaurin coefficients of `self` with given `multi_indices`.

INPUT:

- `multi_indices` – a list of tuples of positive integers, where each tuple has length `self.dimension()`
- `numerical` – (optional; default: 0) a natural number; if positive, return numerical approximations of coefficients with `numerical` digits of accuracy

OUTPUT:

A dictionary whose value of the key `nu` are the Maclaurin coefficient of index `nu` of `self`.

Note: Uses iterated univariate Maclaurin expansions. Slow.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = 2 - 3*x
sage: Hfac = H.factor()
sage: G = 1 / Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(-1/3, [(x - 2/3, 1)])
```

```
sage: F.maclaurin_coefficients([(2*k,) for k in range(6)])
{(0,): 1/2,
 (2,): 9/8,
 (4,): 81/32,
 (6,): 729/128,
 (8,): 6561/512,
 (10,): 59049/2048}

sage: R.<x,y,z> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (4 - 2*x - y - z) * (4 - x - 2*y - z)
sage: Hfac = H.factor()
sage: G = 16 / Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = vector([3, 3, 2])
sage: interval = [1, 2, 4]
sage: S = [r*alpha for r in interval]
sage: F.maclaurin_coefficients(S, numerical=10)
{(3, 3, 2): 0.7849731445,
 (6, 6, 4): 0.7005249476,
 (12, 12, 8): 0.5847732654}
```

nullstellensatz_certificate()

Return a Nullstellensatz certificate of `self` if it exists.

Let $[(q_1, e_1), \ldots, (q_n, e_n)]$ be the denominator factorization of `self`. The Nullstellensatz certificate is a list of polynomials $h_1, \ldots, h_m$ in `self.denominator_ring` that satisfies $h_1 q_1 + \cdots + h_m q_n = 1$ if it exists.

Note: Only works for multivariate base rings.

OUTPUT:

A list of polynomials or `None` if no Nullstellensatz certificate exists.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: G = sin(x)
sage: H = x^2 * (x*y + 1)
sage: f = FFPD(G, H.factor())
sage: L = f.nullstellensatz_certificate()
sage: L
[y^2, -x*y + 1]
sage: df = f.denominator_factored()
sage: sum([L[i]*df[i][0]**df[i][1] for i in xrange(len(df))]) == 1
True

sage: f = 1/(x*y)
sage: L = FFPD(f).nullstellensatz_certificate()
sage: L is None
True
```

nullstellensatz_decomposition()

Return a Nullstellensatz decomposition of `self`.

Let $f = p/q$ where q lies in a d -variate polynomial ring $K[X]$ for some field K and $d \geq 1$. Let

$q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in $K[X]$ into irreducible factors and let V_i be the algebraic variety $\{x \in L^d \mid q_i(x) = 0\}$ of q_i over the algebraic closure L of K. By [Raic2012], f can be written as

$$(*) \quad \sum_A \frac{p_A}{\prod_{i \in A} q_i^{e_i}},$$

where the p_A are products of p and elements in $K[X]$ and the sum is taken over all subsets $A \subseteq \{1, \ldots, m\}$ such that $\bigcap_{i \in A} T_i \neq \emptyset$.

We call $(*)$ a *Nullstellensatz decomposition* of f. Nullstellensatz decompositions are not unique.

The algorithm used comes from [Raic2012].

Note: Recursive. Only works for multivariate `self`.

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import *
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 1/(x*(x*y + 1))
sage: decomp = FFPD(f).nullstellensatz_decomposition()
sage: decomp
(0, []) + (1, [(x, 1)]) + (-y, [(x*y + 1, 1)])
sage: decomp.sum().quotient() == f
True
sage: [r.nullstellensatz_certificate() is None for r in decomp]
[True, True, True]

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: G = sin(y)
sage: H = x*(x*y + 1)
sage: f = FFPD(G, H.factor())
sage: decomp = f.nullstellensatz_decomposition()
sage: decomp
(0, []) + (sin(y), [(x, 1)]) + (-y*sin(y), [(x*y + 1, 1)])
sage: bool(decomp.sum().quotient() == G/H)
True
sage: [r.nullstellensatz_certificate() is None for r in decomp]
[True, True, True]
```

numerator()

Return the numerator of `self`.

OUTPUT:

The numerator.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
```

```
sage: F.numerator()
-e^y
```

numerator_ring

Return the ring of the numerator.

OUTPUT:

A ring.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F.numerator_ring
Symbolic Ring
sage: F = FFPD(G/H)
sage: F
(e^y, [(x - 1, 1), (x*y + x + y - 1, 2)])
sage: F.numerator_ring
Symbolic Ring
```

quotient()

Convert `self` into a quotient.

OUTPUT:

An element.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - x - y - x*y)**2*(1-x)
sage: Hfac = H.factor()
sage: G = exp(y)/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: F
(-e^y, [(x - 1, 1), (x*y + x + y - 1, 2)])
sage: F.quotient()
-e^y/(x^3*y^2 + 2*x^3*y + x^2*y^2 + x^3 - 2*x^2*y - x*y^2 - 3*x^2 -
2*x*y - y^2 + 3*x + 2*y - 1)
```

relative_error(*approx*, *alpha*, *interval*, *exp_scale=1*, *digits=10*)

Return the relative error between the values of the Maclaurin coefficients of `self` with multi-indices `r alpha` for `r` in `interval` and the values of the functions (of the variable `r`) in `approx`.

INPUT:

- `approx` – an individual or list of symbolic expressions in one variable
- `alpha` - a list of positive integers of length `self.denominator_ring.ngens()`
- `interval` – a list of positive integers
- `exp_scale` – (optional; default: 1) a number

OUTPUT:

A list of tuples with properties described below.

This outputs a list whose entries are a tuple `(r*alpha, a_r, b_r, err_r)` for `r` in `interval`. Here `r*alpha` is a tuple; `a_r` is the `r*alpha` (multi-index) coefficient of the Maclaurin series for `self` divided by `exp_scale**r`; `b_r` is a list of the values of the functions in `approx` evaluated at `r` and divided by `exp_scale**m`; `err_r` is the list of relative errors `(a_r - f)/a_r` for `f` in `b_r`. All outputs are decimal approximations.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = 1 - x - y - x*y
sage: Hfac = H.factor()
sage: G = 1 / Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [1, 1]
sage: r = var('r')
sage: a1 = (0.573/sqrt(r))*5.83^r
sage: a2 = (0.573/sqrt(r) - 0.0674/r^(3/2))*5.83^r
sage: es = 5.83
sage: F.relative_error([a1, a2], alpha, [1, 2, 4, 8], es) # long time
[((1, 1), 0.5145797599,
  [0.5730000000, 0.5056000000], [-0.1135300000, 0.01745066667]),
 ((2, 2), 0.3824778089,
  [0.4051721856, 0.3813426871], [-0.05933514614, 0.002967810973]),
 ((4, 4), 0.2778630595,
  [0.2865000000, 0.2780750000], [-0.03108344267, -0.0007627515584]),
 ((8, 8), 0.1991088276,
  [0.2025860928, 0.1996074055], [-0.01746414394, -0.002504047242])]
```

singular_ideal()

Return the singular ideal of `self`.

Let R be the ring of `self` and H its denominator. Let H_{red} be the reduction (square-free part) of H. Return the ideal in R generated by H_{red} and its partial derivatives. If the coefficient field of R is algebraically closed, then the output is the ideal of the singular locus (which is a variety) of the variety of H.

OUTPUT:

An ideal.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y,z> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (1 - x*(1 + y))^3 * (1 - z*x**2*(1 + 2*y))
sage: df = H.factor()
sage: G = 1 / df.unit()
sage: F = FFPD(G, df)
sage: F.singular_ideal()
Ideal (x*y + x - 1, y^2 - 2*y*z + 2*y - z + 1, x*z + y - 2*z + 1) of
 Multivariate Polynomial Ring in x, y, z over Rational Field
```

smooth_critical_ideal(*alpha*)

Return the smooth critical ideal of `self`.

Let R be the ring of `self` and H its denominator. Return the ideal in R of smooth critical points of the variety of H for the direction `alpha`. If the variety V of H has no smooth points, then return the ideal in R of V.

See [RaWi2012] for more details.

INPUT:

- `alpha` – a tuple of positive integers and/or symbolic entries of length `self.denominator_ring.ngens()`

OUTPUT:

An ideal.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: H = (1 - x - y - x*y)^2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = var('a1, a2')
sage: F.smooth_critical_ideal(alpha)
Ideal (y^2 + 2*a1/a2*y - 1, x + ((-a2)/a1)*y + (-a1 + a2)/a1) of
 Multivariate Polynomial Ring in x, y over Fraction Field of
 Multivariate Polynomial Ring in a1, a2 over Rational Field

sage: H = (1-x-y-x*y)^2
sage: Hfac = H.factor()
sage: G = 1/Hfac.unit()
sage: F = FFPD(G, Hfac)
sage: alpha = [7/3, var('a')]
sage: F.smooth_critical_ideal(alpha)
Ideal (y^2 + 14/(3*a)*y - 1, x + (-3/7*a)*y + 3/7*a - 1) of
 Multivariate Polynomial Ring in x, y over Fraction Field of
 Univariate Polynomial Ring in a over Rational Field
```

univariate_decomposition()

Return the usual univariate partial fraction decomposition of `self`.

Assume that the numerator of `self` lies in the same univariate factorial polynomial ring as the factors of the denominator.

Let $f = p/q$ be a rational expression where p and q lie in a univariate factorial polynomial ring R. Let $q_1^{e_1} \cdots q_n^{e_n}$ be the unique factorization of q in R into irreducible factors. Then f can be written uniquely as:

$$(*) \quad p_0 + \sum_{i=1}^{m} \frac{p_i}{q_i^{e_i}},$$

for some $p_j \in R$. We call $(*)$ the *usual partial fraction decomposition* of f.

Note: This partial fraction decomposition can be computed using `partial_fraction()` or `partial_fraction_decomposition()` as well. However, here we use the already obtained/cached factorization of the denominator. This gives a speed up for non-small instances.

OUTPUT:

An instance of FractionWithFactoredDenominatorSum.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
```

One variable:

```
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: f
(15*x^7 - 15*x^6 + 5*x^5 - 5*x^4 + 6*x^3 - 2*x^2 + x - 1)/(3*x^4 -
3*x^3 + x^2 - x)
sage: decomp = FFPD(f).univariate_decomposition()
sage: decomp
(5*x^3, []) +
(1, [(x - 1, 1)]) +
(1, [(x, 1)]) +
(1/3, [(x^2 + 1/3, 1)])
sage: decomp.sum().quotient() == f
True
```

One variable with numerator in symbolic ring:

```
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = 5*x^3 + 1/x + 1/(x-1) + exp(x)/(3*x^2 + 1)
sage: f
(5*x^5 - 5*x^4 + 2*x - 1)/(x^2 - x) + e^x/(3*x^2 + 1)
sage: decomp = FFPD(f).univariate_decomposition()
sage: decomp
(0, []) +
(15/4*x^7 - 15/4*x^6 + 5/4*x^5 - 5/4*x^4 + 3/2*x^3 + 1/4*x^2*e^x -
 3/4*x^2 - 1/4*x*e^x + 1/2*x - 1/4, [(x - 1, 1)]) +
(-15*x^7 + 15*x^6 - 5*x^5 + 5*x^4 - 6*x^3 -
 x^2*e^x + 3*x^2 + x*e^x - 2*x + 1, [(x, 1)]) +
(1/4*(15*x^7 - 15*x^6 + 5*x^5 - 5*x^4 + 6*x^3 + x^2*e^x -
      3*x^2 - x*e^x + 2*x - 1)*(3*x - 1), [(x^2 + 1/3, 1)])
```

One variable over a finite field:

```
sage: R.<x> = PolynomialRing(GF(2))
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: f
(x^6 + x^4 + 1)/(x^3 + x)
sage: decomp = FFPD(f).univariate_decomposition()
sage: decomp
(x^3, []) + (1, [(x, 1)]) + (x, [(x + 1, 2)])
sage: decomp.sum().quotient() == f
True
```

One variable over an inexact field:

```
sage: R.<x> = PolynomialRing(CC)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
sage: f
(15.0000000000000*x^7 - 15.0000000000000*x^6 + 5.00000000000000*x^5
 - 5.00000000000000*x^4 + 6.00000000000000*x^3
```

```
 - 2.00000000000000*x^2 + x - 1.00000000000000)/(3.00000000000000*x^4
 - 3.00000000000000*x^3 + x^2 - x)
sage: decomp = FFPD(f).univariate_decomposition()
sage: decomp
(5.00000000000000*x^3, []) +
(1.00000000000000, [(x - 1.00000000000000, 1)]) +
(-0.288675134594813*I, [(x - 0.577350269189626*I, 1)]) +
(1.00000000000000, [(x, 1)]) +
(0.288675134594813*I, [(x + 0.577350269189626*I, 1)])
sage: decomp.sum().quotient() == f # Rounding error coming
False
```

TESTS:

```
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = exp(x) / (x^2-x)
sage: f
e^x/(x^2 - x)
sage: FFPD(f).univariate_decomposition()
(0, []) + (e^x, [(x - 1, 1)]) + (-e^x, [(x, 1)])
```

AUTHORS:

- Robert Bradshaw (2007-05-31)
- Alexander Raichev (2012-06-25)
- Daniel Krenn (2014-12-01)

class `sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`FractionWithFactore`**

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.rings.ring.Ring`

This is the ring of fractions with factored denominator.

INPUT:

- `denominator_ring` – the base ring (a polynomial ring)
- `numerator_ring` – (optional) the numerator ring; the default is the `denominator_ring`
- `category` – (default: `Rings`) the category

See also:

`FractionWithFactoredDenominator`, `asymptotics_multivariate_generating_functions`

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWi
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: df = [x, 1], [y, 1], [x*y+1, 1]
sage: f = FFPD(x, df)   # indirect doctest
```

```
sage: f
(1, [(y, 1), (x*y + 1, 1)])
```

AUTHORS:

- Daniel Krenn (2014-12-01)

Element
alias of FractionWithFactoredDenominator

base_ring()
Returns the base ring.

OUTPUT:

A ring.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: P.<X, Y> = ZZ[]
sage: F = FractionWithFactoredDenominatorRing(P); F
Ring of fractions with factored denominator
over Multivariate Polynomial Ring in X, Y over Integer Ring
sage: F.base_ring()
Integer Ring
sage: F.base()
Multivariate Polynomial Ring in X, Y over Integer Ring
```

rename_keyword
alias of rename_keyword

class sage.rings.asymptotic.asymptotics_multivariate_generating_functions.**FractionWithFactore**
Bases: list

A list representing the sum of FractionWithFactoredDenominator objects with distinct denominator factorizations.

AUTHORS:

- Alexander Raichev (2012-06-25)
- Daniel Krenn (2014-12-01)

denominator_ring
Return the polynomial ring of the denominators of self.

OUTPUT:

A ring or None if the list is empty.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = FFPD(x + y, [(y, 1), (x, 1)])
sage: s = FractionWithFactoredDenominatorSum([f])
sage: s.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
sage: g = FFPD(x + y, [])
sage: t = FractionWithFactoredDenominatorSum([g])
sage: t.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
```

sum()

Return the sum of the elements in `self`.

OUTPUT:

An instance of `FractionWithFactoredDenominator`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: df = (x, 1), (y, 1), (x*y + 1, 1)
sage: f = FFPD(2, df)
sage: g = FFPD(2*x*y, df)
sage: FractionWithFactoredDenominatorSum([f, g])
(2, [(y, 1), (x, 1), (x*y + 1, 1)]) + (2, [(x*y + 1, 1)])
sage: FractionWithFactoredDenominatorSum([f, g]).sum()
(2, [(y, 1), (x, 1)])

sage: f = FFPD(cos(x), [(x, 2)])
sage: g = FFPD(cos(y), [(x, 1), (y, 2)])
sage: FractionWithFactoredDenominatorSum([f, g])
(cos(x), [(x, 2)]) + (cos(y), [(y, 2), (x, 1)])
sage: FractionWithFactoredDenominatorSum([f, g]).sum()
(y^2*cos(x) + x*cos(y), [(y, 2), (x, 2)])
```

whole_and_parts()

Rewrite `self` as a sum of a (possibly zero) polynomial followed by reduced rational expressions.

OUTPUT:

An instance of `FractionWithFactoredDenominatorSum`.

Only useful for multivariate decompositions.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import Fracti
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: f = x**2 + 3*y + 1/x + 1/y
sage: f = FFPD(f); f
(x^3*y + 3*x*y^2 + x + y, [(y, 1), (x, 1)])
sage: FractionWithFactoredDenominatorSum([f]).whole_and_parts()
(x^2 + 3*y, []) + (x + y, [(y, 1), (x, 1)])

sage: f = cos(x)**2 + 3*y + 1/x + 1/y; f
cos(x)^2 + 3*y + 1/x + 1/y
sage: G = f.numerator()
sage: H = R(f.denominator())
sage: f = FFPD(G, H.factor()); f
(x*y*cos(x)^2 + 3*x*y^2 + x + y, [(y, 1), (x, 1)])
sage: FractionWithFactoredDenominatorSum([f]).whole_and_parts()
(0, []) + (x*y*cos(x)^2 + 3*x*y^2 + x + y, [(y, 1), (x, 1)])
```

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**coerce_point**(R, p)

Coerce the keys of the dictionary `p` into the ring `R`.

Warning: This method assumes that it is possible.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWi
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = FFPD()
sage: p = {SR(x): 1, SR(y): 7/8}
sage: for k in sorted(p.keys(), key=str):
....:     print k, k.parent(), p[k]
x Symbolic Ring 1
y Symbolic Ring 7/8
sage: q = coerce_point(R, p)
sage: for k in sorted(q.keys(), key=str):
....:     print k, k.parent(), q[k]
x Multivariate Polynomial Ring in x, y over Rational Field 1
y Multivariate Polynomial Ring in x, y over Rational Field 7/8
```

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`diff_all`**(*f, V, n, ending=[ ], sub=None, sub_final=None, zero_order=0, rekey=None*)

Return a dictionary of representative mixed partial derivatives of f from order 1 up to order n with respect to the variables in V.

The default is to key the dictionary by all nondecreasing sequences in V of length 1 up to length n.

INPUT:

- `f` – an individual or list of $\mathcal{C}^{n+1}$ functions
- `V` – a list of variables occurring in f
- `n` – a natural number
- `ending` – a list of variables in V
- `sub` – an individual or list of dictionaries
- `sub_final` – an individual or list of dictionaries
- `rekey` – a callable symbolic function in V or list thereof
- `zero_order` – a natural number

OUTPUT:

The dictionary `{s_1:deriv_1, ..., sr:deriv_r}`.

Here `s_1, ..., s_r` is a listing of all nondecreasing sequences of length 1 up to length n over the alphabet V, where $w > v$ in X if and only if `str(w) > str(v)`, and `deriv_j` is the derivative of f with respect to the derivative sequence `s_j` and simplified with respect to the substitutions in `sub` and evaluated at `sub_final`. Moreover, all derivatives with respect to sequences of length less than `zero_order` (derivatives of order less than `zero_order`) will be made zero.

If `rekey` is nonempty, then `s_1, ..., s_r` will be replaced by the symbolic derivatives of the functions in `rekey`.

If `ending` is nonempty, then every derivative sequence `s_j` will be suffixed by `ending`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_all
sage: f = function('f')(x)
sage: dd = diff_all(f, [x], 3)
sage: dd[(x, x, x)]
D[0, 0, 0](f)(x)

sage: d1 = {diff(f, x): 4*x^3}
sage: dd = diff_all(f, [x], 3, sub=d1)
sage: dd[(x, x, x)]
24*x

sage: dd = diff_all(f, [x], 3, sub=d1, rekey=f)
sage: dd[diff(f, x, 3)]
24*x

sage: a = {x:1}
sage: dd = diff_all(f, [x], 3, sub=d1, rekey=f, sub_final=a)
sage: dd[diff(f, x, 3)]
24

sage: X = var('x, y, z')
sage: f = function('f')(*X)
sage: dd = diff_all(f, X, 2, ending=[y, y, y])
sage: dd[(z, y, y, y)]
D[1, 1, 1, 2](f)(x, y, z)

sage: g = function('g')(*X)
sage: dd = diff_all([f, g], X, 2)
sage: dd[(0, y, z)]
D[1, 2](f)(x, y, z)

sage: dd[(1, z, z)]
D[2, 2](g)(x, y, z)

sage: f = exp(x*y*z)
sage: ff = function('ff')(*X)
sage: dd = diff_all(f, X, 2, rekey=ff)
sage: dd[diff(ff, x, z)]
x*y^2*z*e^(x*y*z) + y*e^(x*y*z)
```

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`diff_op`**(*A*, *B*, *AB_derivs*, *V*, *M*, *r*, *N*)

Return the derivatives $DD^{(l+k)}(A[j]B^l)$ evaluated at a point p for various natural numbers j, k, l which depend on r and N.

Here DD is a specific second-order linear differential operator that depends on M , A is a list of symbolic functions, B is symbolic function, and `AB_derivs` contains all the derivatives of A and B evaluated at p that are necessary for the computation.

INPUT:

- A – a single or length r list of symbolic functions in the variables V
- B – a symbolic function in the variables V.
- AB_derivs – a dictionary whose keys are the (symbolic) derivatives of A[0], ..., A[r-1] up to order 2 * N-2 and the (symbolic) derivatives of B up to order 2 * N; the values of the dictionary are complex numbers that are the keys evaluated at a common point p
- V – the variables of the A[j] and B
- M – a symmetric $l \times l$ matrix, where l is the length of V
- r, N – natural numbers

OUTPUT:

A dictionary.

The output is a dictionary whose keys are natural number tuples of the form (j, k, l), where $l \leq 2k$, $j \leq r-1$, and $j+k \leq N-1$, and whose values are $DD^(l+k)(A[j]B^l)$ evaluated at a point p, where DD is the linear second-order differential operator $-\sum_{i=0}^{l-1}\sum_{j=0}^{l-1} M[i][j]\partial^2/(\partial V[j]\partial V[i])$.

Note: For internal use by FractionWithFactoredDenominator.asymptotics_smooth() and FractionWithFactoredDenominator.asymptotics_multiple().

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_op
sage: T = var('x, y')
sage: A = function('A')(*tuple(T))
sage: B = function('B')(*tuple(T))
sage: AB_derivs = {}
sage: M = matrix([[1, 2],[2, 1]])
sage: DD = diff_op(A, B, AB_derivs, T, M, 1, 2)
sage: sorted(DD.keys())
[(0, 0, 0), (0, 1, 0), (0, 1, 1), (0, 1, 2)]
sage: len(DD[(0, 1, 2)])
246
```

sage.rings.asymptotic.asymptotics_multivariate_generating_functions.**diff_op_simple**(*A, B, AB_derivs, x, v, a, N*)

Return $DD^(ek+vl)(AB^l)$ evaluated at a point p for various natural numbers e, k, l that depend on v and N.

Here DD is a specific linear differential operator that depends on a and v , A and B are symbolic functions, and AB_derivs contains all the derivatives of A and B evaluated at p that are necessary for the computation.

Note: For internal use by the function FractionWithFactoredDenominator.asymptotics_smooth().

INPUT:

- A, B – Symbolic functions in the variable x

- `AB_derivs` - a dictionary whose keys are the (symbolic) derivatives of `A` up to order `2 * N` if `v` is even or `N` if `v` is odd and the (symbolic) derivatives of `B` up to order `2 * N + v` if `v` is even or `N + v` if `v` is odd; the values of the dictionary are complex numbers that are the keys evaluated at a common point p
- `x` – a symbolic variable
- `a` – a complex number
- `v, N` – natural numbers

OUTPUT:

A dictionary.

The output is a dictionary whose keys are natural number pairs of the form (k, l), where $k < N$ and $l \le 2k$ and whose values are $DD^(ek + vl)(AB^l)$ evaluated at a point p. Here $e = 2$ if v is even, $e = 1$ if v is odd, and DD is the linear differential operator $(a^{-1/v} d/dt)$ if v is even and $(|a|^{-1/v} i \text{sgn}(a) d/dt)$ if v is odd.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_op_si
sage: A = function('A')(x)
sage: B = function('B')(x)
sage: AB_derivs = {}
sage: sorted(diff_op_simple(A, B, AB_derivs, x, 3, 2, 2).items())
[((0, 0), A(x)),
 ((1, 0), 1/2*I*2^(2/3)*D[0](A)(x)),
 ((1, 1), 1/4*2^(2/3)*(B(x)*D[0, 0, 0, 0](A)(x)
  + 4*D[0, 0, 0](A)(x)*D[0](B)(x) + 6*D[0, 0](A)(x)*D[0, 0](B)(x)
  + 4*D[0](A)(x)*D[0, 0, 0](B)(x) + A(x)*D[0, 0, 0, 0](B)(x)))]
```

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`diff_prod`**(*f_derivs*, *u*, *g*, *X*, *interval*, *end*, *uderivs*, *atc*)

Take various derivatives of the equation $f = ug$, evaluate them at a point c, and solve for the derivatives of u.

INPUT:

- `f_derivs` – a dictionary whose keys are all tuples of the form `s + end`, where `s` is a sequence of variables from `X` whose length lies in `interval`, and whose values are the derivatives of a function f evaluated at c
- `u` – a callable symbolic function
- `g` – an expression or callable symbolic function
- `X` – a list of symbolic variables
- `interval` – a list of positive integers Call the first and last values n and nn, respectively
- `end` – a possibly empty list of repetitions of the variable `z`, where `z` is the last element of `X`
- `uderivs` – a dictionary whose keys are the symbolic derivatives of order 0 to order $n - 1$ of `u` evaluated at c and whose values are the corresponding derivatives evaluated at c

- `atc` – a dictionary whose keys are the keys of c and all the symbolic derivatives of order 0 to order nn of `g` evaluated c and whose values are the corresponding derivatives evaluated at c

OUTPUT:

A dictionary whose keys are the derivatives of `u` up to order nn and whose values are those derivatives evaluated at c.

This function works by differentiating the equation $f = ug$ with respect to the variable sequence `s + end`, for all tuples `s` of `X` of lengths in `interval`, evaluating at the point c , and solving for the remaining derivatives of `u`. This function assumes that `u` never appears in the differentiations of $f = ug$ after evaluating at c.

Note: For internal use by `FractionWithFactoredDenominator.asymptotics_multiple()`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_prod
sage: u = function('u')(x)
sage: g = function('g')(x)
sage: fd = {(x,):1,(x, x):1}
sage: ud = {u(x=2): 1}
sage: atc = {x: 2, g(x=2): 3, diff(g, x)(x=2): 5}
sage: atc[diff(g, x, x)(x=2)] = 7
sage: dd = diff_prod(fd, u, g, [x], [1, 2], [], ud, atc)
sage: dd[diff(u, x, 2)(x=2)]
22/9
```

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`diff_seq`**(*V*, *s*)

Given a list `s` of tuples of natural numbers, return the list of elements of `V` with indices the elements of the elements of `s`.

INPUT:

- `V` – a list
- `s` – a list of tuples of natural numbers in the interval `range(len(V))`

OUTPUT:

The tuple `tuple([V[tt] for tt in sorted(t)])`, where `t` is the list of elements of the elements of `s`.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_seq
sage: V = list(var('x, t, z'))
sage: diff_seq(V,([0, 1],[0, 2, 1],[0, 0]))
(x, x, x, x, t, t, z)
```

Note: This function is for internal use by `diff_op()`.

`sage.rings.asymptotic.asymptotics_multivariate_generating_functions.`**`direction`**(*v*, *coordinate=None*)

Return `[vv/v[coordinate] for vv in v]` where `coordinate` is the last index of `v` if not specified otherwise.

INPUT:

- v – a vector
- coordinate – (optional; default: None) an index for v

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import direction
sage: direction([2, 3, 5])
(2/5, 3/5, 1)
sage: direction([2, 3, 5], 0)
(1, 3/2, 5/2)
```

sage.rings.asymptotic.asymptotics_multivariate_generating_functions.**permutation_sign**(*s*, *u*)

This function returns the sign of the permutation on 1, ..., len(u) that is induced by the sublist s of u.

Note: For internal use by FractionWithFactoredDenominator.cohomology_decomposition().

INPUT:

- s – a sublist of u
- u – a list

OUTPUT:

The sign of the permutation obtained by taking indices within u of the list s + sc, where sc is u with the elements of s removed.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import permutatic
sage: u = ['a', 'b', 'c', 'd', 'e']
sage: s = ['b', 'd']
sage: permutation_sign(s, u)
-1
sage: s = ['d', 'b']
sage: permutation_sign(s, u)
1
```

sage.rings.asymptotic.asymptotics_multivariate_generating_functions.**subs_all**(*f*, *sub*, *simplify=False*)

Return the items of f substituted by the dictionaries of sub in order of their appearance in sub.

INPUT:

- f – an individual or list of symbolic expressions or dictionaries
- sub – an individual or list of dictionaries
- simplify – (default: False) boolean; set to True to simplify the result

OUTPUT:

The items of f substituted by the dictionaries of sub in order of their appearance in sub. The subs() command is used. If simplify is True, then simplify() is used after substitution.

EXAMPLES:

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import subs_all
sage: var('x, y, z')
(x, y, z)
```

```
sage: a = {x:1}
sage: b = {y:2}
sage: c = {z:3}
sage: subs_all(x + y + z, a)
y + z + 1
sage: subs_all(x + y + z, [c, a])
y + 4
sage: subs_all([x + y + z, y^2], b)
[x + z + 2, 4]
sage: subs_all([x + y + z, y^2], [b, c])
[x + 5, 4]
```

```
sage: var('x, y')
(x, y)
sage: a = {'foo': x**2 + y**2, 'bar': x - y}
sage: b = {x: 1 , y: 2}
sage: subs_all(a, b)
{'bar': -1, 'foo': 5}
```

CHAPTER

FIVE

INDICES AND TABLES

- Index
- Module Index
- Search Page

[FS2009] Philippe Flajolet and Robert Sedgewick, Analytic combinatorics. Cambridge University Press, Cambridge, 2009.

[AiYu1983] I.A. Aizenberg and A.P. Yuzhakov. *Integral representations and residues in multidimensional complex analysis*. Translations of Mathematical Monographs, **58**. American Mathematical Society, Providence, RI. (1983). x+283 pp. ISBN: 0-8218-4511-X.

[Raic2012] Alexander Raichev. *Leinartas's partial fraction decomposition*. Arxiv 1206.4740.

[RaWi2008a] Alexander Raichev and Mark C. Wilson. *Asymptotics of coefficients of multivariate generating functions: improvements for smooth points*, Electronic Journal of Combinatorics, Vol. 15 (2008). R89 Arxiv 0803.2914.

[RaWi2012] Alexander Raichev and Mark C. Wilson. *Asymptotics of coefficients of multivariate generating functions: improvements for smooth points*. Online Journal of Analytic Combinatorics. Issue 6, (2011). Arxiv 1009.5715.

[PeWi2013] Robin Pemantle and Mark C. Wilson. *Analytic Combinatorics in Several Variables*. Cambridge University Press, 2013.

PYTHON MODULE INDEX

r

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z