Asymptotic Expansions

Release 10.4

The Sage Development Team

Jul 23, 2024
1 The Asymptotic Ring ................................. 1

2 Asymptotic Expansion Generators .................. 3

3 Supplements ........................................ 5
   3.1 Growth Groups .................................. 5
   3.2 Term Monoids ................................... 5
   3.3 Miscellaneous ................................... 5

4 Asymptotic Expansions — Table of Contents .............. 7
   4.1 Asymptotic Ring .................................. 7
   4.2 Common Asymptotic Expansions ................. 54
   4.3 (Asymptotic) Growth Groups .................. 67
   4.4 Cartesian Products of Growth Groups .......... 104
   4.5 (Asymptotic) Term Monoids .................. 117
   4.6 Asymptotic Expansions — Miscellaneous ....... 163
   4.7 Asymptotics of Multivariate Generating Series .. 173

5 Indices and Tables .................................. 231

Python Module Index ................................ 233

Index ................................................. 235
The asymptotic ring, as well as its main documentation is contained in the module

- *Asymptotic Ring.*
Some common asymptotic expansions can be generated in

- *Common Asymptotic Expansions.*
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*.
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

\[ 3x^2y^2 + 7x^3y^3 + O(x^2) + O(y) \]

as \( x \) and \( y \) tend to \( \infty \). 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).
- \( O(g) \) (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 \( z^\mathbb{Z} \) or \( z^\mathbb{Q} \)),
- elements of the form \( \log(z)^q \) for some integer or rational \( q \) (growth groups \( \log(z)^\mathbb{Z} \) or \( \log(z)^\mathbb{Q} \)),
- elements of the form \( a^z \) for some rational \( a \) (growth group \( \mathbb{Q}^z \)), 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 \( x^\mathbb{Q} \cdot \log(x)^\mathbb{Z} \cdot \mathbb{Q}^y \cdot y^\mathbb{Q} \)).

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

\[ \lim_{z \to \infty} \frac{g_1}{g_2} \leq 1. \]
Asymptotic Expansions, Release 10.4

Note: Asymptotic rings where the variable tend to some value distinct from $\infty$ 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.

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
```

```
>>> from sage.all import *

>>> A = AsymptoticRing(growth_group='z^QQ', coefficient_ring=ZZ, names=('z',)); (z,) ∈ A._first_ngens(1); 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))
```

```
>>> from sage.all import *

>>> 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)
```

```
>>> from sage.all import *

>>> Integer(4)*z**Integer(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', coe-
.Supicient_ring=QQ); B
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))
```

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
```
Asymptotic Expansions, Release 10.4

```python
>>> from sage.all import *
>>> z**Integer(2) + Integer(3)*z*(Integer(1)-z)
-2*z^2 + 3*z
>>> (Integer(3)*z + Integer(2))**Integer(3)
27*z^3 + 54*z^2 + 36*z + 8
```

In addition to that, special powers—our growth group $z^{\mathbb{Q}}$ allows the exponents to be out of $\mathbb{Q}$—can also be computed:

```python
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

```python
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

```python
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:

```python
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 off the rest by using

```python
sage: (1 / (z-1)).truncate(5)
z^(-1) + z^(-2) + z^(-3) + z^(-4) + z^(-5) + O(z^(-6))
```
>>> from sage.all import *
>>> (Integer(1) / (z-Integer(1))).truncate(Integer(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))

>>> from sage.all import *
>>> Integer(1) / (z-1) + O(z**(-Integer(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))

>>> from sage.all import *
>>> (Integer(4)*z+Integer(1)) / (z**Integer(3)+z**Integer(2)+z+O(z**Integer(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).

>>> from sage.all import *
>>> Integer(1) / O(z)
Traceback (most recent call last):
  ...  
ZeroDivisionError: Cannot invert O(z).

is not invertible, since it includes 0.

Powers, Exponentials 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))

>>> from sage.all import *
>>> -log(Integer(1)-Integer(1)/z)
z^(-1) + 1/2*z^(-2) + 1/3*z^(-3) + ... + O(z^(-21))

as z → ∞.

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))

4.1. Asymptotic Ring 11
Arbitrary powers work as well; for example, we have

\[
\text{sage: } (1 + \frac{1}{z} + O(1/z^5))^{(1 + \frac{1}{z})} \\
1 + z^{-1} + z^{-2} + 1/2*z^{-3} + 1/3*z^{-4} + O(z^{-5})
\]

Multivariate Arithmetic

Now let us move on to arithmetic in the multivariate ring

\[
\text{sage: } B \\
\text{Asymptotic Ring } <x^\mathbb{Q} * \log(x)^\mathbb{Z} * \mathbb{Q}^y * y^\mathbb{Q}> \text{ over Rational Field}
\]

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

\[
\text{sage: } E.<n> = \text{AsymptoticRing}(\text{growth_group}=\text{'}n^\mathbb{Z}\text{'}, \text{coefficient_ring}=\text{SR}, \text{default_} \\
\text{prec=5}); E \\
\text{Asymptotic Ring } <n^\mathbb{Z}> \text{ over Symbolic Ring} \\
\text{sage: } (1 + 1/n)^n \\
e = 1/2*e*n^(-1) + 11/24*e*n^(-2) - 7/16*e*n^(-3) + 2447/5760*e*n^(-4) + O(n^(-5))
\]
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
sage: A
Asymptotic Ring <z^QQ> over Integer Ring
sage: (z + 1/2).parent()
Asymptotic Ring <z^QQ> over Rational Field
```

```python
>>> from sage.all import *
>>> A
Asymptotic Ring <z^QQ> over Integer Ring
>>> (z + Integer(1)/Integer(2)).parent()
Asymptotic Ring <z^QQ> over Rational Field
```

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

```sage
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)
```

```python
>>> from sage.all import *
>>> C = AsymptoticRing(growth_group='c^ZZ', coefficient_ring=ZZ['e'], names=('c',));
˓→(c,) = C._first_ngens(1)
>>> C.an_element()
e^3*c^3 + O(c)
>>> C.an_element() / Integer(7)
1/7*e^3*c^3 + O(c)
```

Here the result's coefficient ring is the newly found

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

```python
>>> from sage.all import *
>>> (C.an_element() / Integer(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
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
```

4.1. Asymptotic Ring
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:

```python
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: 0(x), 0(y)
  +-- x^3*y
  |   +-- successors: oo
  |   +-- predecessors: x^2*y
  +-- x^2*y
  |   +-- successors: x^3*y
  |   +-- predecessors: 0(x), 0(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: 0(x), 0(y)
  |   +-- no predecessors
```

(continues on next page)
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

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

Bases: 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:

```python
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:

\[
\text{sage: type(x)}
\]

\[
<\text{class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.}\ ...
\]

The usual ring operations, but allowing rational exponents (growth group \( x^{\mathbb{Q}} \)) can be performed:

\[
\text{sage: } x^2 + 3*(x - x^{(2/5)})
\]

\[
x^2 + 3*x - 3*x^{(2/5)}
\]

\[
\text{sage: } (3*x^{(1/3)} + 2)^3
\]

\[
27*x + 54*x^{(2/3)} + 36*x^{(1/3)} + 8
\]

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:

\[
\text{sage: } (x+2*x^2+3*x^3+4*x^4) * (O(x)+x^2)
\]

\[
4*x^6 + O(x^5)
\]

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

\[
\text{sage: expr1 = x + 2*x^2 + 3*x^3 + 4*x^4; expr2 = O(x) + x^2}
\]

\[
\text{sage: print(expr1.summands.repr_full())}
\]

\[
\text{poset(x, 2*x^2, 3*x^3, 4*x^4)}
\]

\[
\text{null}
\]

\[
\text{no predecessors}
\]

\[
\text{successors: } x
\]

\[
\text{predecessors: null}
\]

\[
\text{successors: } 2*x^2
\]

(continues on next page)
\begin{verbatim}
>>> from sage.all import *

>>> expr1 = x + Integer(2)*x**Integer(2) + Integer(3)*x**Integer(3) +
       Integer(4)*x**Integer(4); expr2 = O(x) + x**Integer(2)

>>> print(expr1.summands.repr_full())
poset(x, 2*x^2, 3*x^3, 4*x^4)

>>> print((expr1 * expr2).summands.repr_full())
poset(0(x^5), 4*x^6)

>>> print(expr2.summands.repr_full())
poset(O(x), x^2)

>>> print((expr1 * expr2).summands.repr_full())
poset(O(x^5), 4*x^6)
\end{verbatim}

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

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

See also:

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

**B** *(valid_from=0)*

Convert all terms in this asymptotic expansion to B-terms.
INPUT:

- `valid_from` – dictionary mapping variable names to lower bounds for the corresponding variable. The bound implied by this term is valid when all variables are at least their corresponding lower bound. If a number is passed to `valid_from`, then the lower bounds for all variables of the asymptotic expansion are set to this number.

OUTPUT:

An asymptotic expansion

EXAMPLES:

```python
sage: AR.<x, z> = AsymptoticRing(growth_group='x^ZZ * z^ZZ', coefficient_ring=ZZ)
sage: AR.B(2*x^2, {x: 10})  # indirect doctest
B(2*x^2, x >= 10)
sage: expr = 42*x^42 + x^10 + AR.B(x^2, 20); expr  # indirect doctest
42*x^42 + x^10 + B(x^2, x >= 20, z >= 20)
sage: type(AR.B(x, 10))  # indirect doctest
<class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.element_class'>
sage: 2*z^3 + AR.B(5*z^2, {z: 20})  # indirect doctest
2*z^3 + B(5*z^2, z >= 20)
sage: AR.B(42*x^2)  # indirect doctest
B(42*x^2, x >= 0, z >= 0)
```

```
from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ * z^ZZ', coefficient_ring=ZZ,
        names=('x', 'z'),); (x, z,) = AR._first_ngens(2)
>>> AR.B(Integer(2)*x**Integer(2), {x: Integer(10)})  # indirect doctest
B(2*x^2, x >= 10)
>>> expr = Integer(42)*x**Integer(42) + x**Integer(10) + AR.B(x**Integer(2),
        expr # indirect doctest
42*x^42 + x^10 + B(x^2, x >= 20, z >= 20)
>>> type(AR.B(x, Integer(10)))  # indirect doctest
<class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.element_class'>
```

```
>>> Integer(2)*z**Integer(3) + AR.B(Integer(5)*z**Integer(2), {z: Integer(20)})  # indirect doctest
2*z^3 + B(5*z^2, z >= 20)
>>> (Integer(2)*x).B((x: Integer(20)))
B(2*x, x >= 20)
>>> AR.B(Integer(4)*x**Integer(2)*z**Integer(3), valid_from=Integer(10))  # indirect doctest
B(4*x^2*z^3, x >= 10, z >= 10)
>>> AR.B(Integer(42)*x**Integer(2))  # indirect doctest
B(42*x^2, x >= 0, z >= 0)
```

```
O()  # Convert all terms in this asymptotic expansion to O-terms.
```

OUTPUT:

An asymptotic expansion
EXAMPLES:

```python
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)
```

```python
>>> from sage.all import *
... AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ, names=('x',)); (x,) = AR._first_ngens(1)
... O(x)
O(x)
... type(O(x))
<class 'sage.rings.asymptotic.asymptotic_ring.AsymptoticRing_with_category.element_class'>
... expr = Integer(42)*x**Integer(42) + x**Integer(10) + O(x**Integer(2)); expr
42*x^42 + x^10 + O(x^2)
... expr.O()
O(x^42)
... (Integer(2)*x).O()
O(x)
```

See also:

```python
sage.rings.power_series_ring.PowerSeriesRing(), sage.rings.laurent_series_ring.LaurentSeriesRing().
```

**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:**

```python
sage: assume(SR.an_element() > 0)
sage: A.<n> = AsymptoticRing('QQ^n * n^ZZ', SR)
```
```python
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.exact_part().compare_with_values(n, catalan, [5, 10, 20])
Traceback (most recent call last):
  ... NotImplementedError: exactly one error term required
sage: expansion.exact_part().compare_with_values(n, catalan, [5, 10, 20], rescaled=False, ring=SR)
[(5, 0.3886263699387?), (10, 19.1842458318?), (20, 931314.63637?)]
```
Asymptotic Expansions, Release 10.4

(continued from previous page)

```
....:     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()

**error_part()**

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

**OUTPUT:**

An asymptotic expansion.

**EXAMPLES:**

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

```
>>> from sage.all import *
>>> R = AsymptoticRing('x^QQ * log(x)^QQ * y^QQ', QQ, names=('x', 'y',)); (x,___
˓→y,) = R._first_ngens(2)
>>> (x*log(x) + y**Integer(2) + O(x) + O(y)).error_part()  
O(x) + O(y)
```

**exact_part()**

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

**OUTPUT:**

An asymptotic expansion.

**EXAMPLES:**

```
exp (\texttt{precision=None})

Return the exponential of (i.e., the power of }e\text{ to) this asymptotic expansion.

INPUT:

• \texttt{precision} – the precision used for truncating the expansion. If \texttt{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)\text{, try to expand the series and truncate according to the given precision.

Todo: As soon as }L\text{-terms are implemented, this implementation has to be adapted as well in order to yield
correct results.

EXAMPLES:

\begin{verbatim}
sage: A.<x> = AsymptoticRing("(e^x)^ZZ * x^ZZ * log(x)^ZZ", SR)
sage: exp(x)  
es^x
sage: exp(2*x)  
(e^x)^2
sage: exp(x + log(x))  
e^x*x

>>> from sage.all import *

>>> A = AsymptoticRing("(e^x)^ZZ * x^ZZ * log(x)^ZZ", SR, names=('x',)); (x,) = A._first_ngens(1)

>>> exp(x)  
e^x

>>> exp(Integer(2)*x)  
(e^x)^2
\end{verbatim}
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))

from sage.all import *

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

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_prec=5)
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)))^QQ * (e^n)^QQ * n^QQ * log(n)^QQ>
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(2)*sqrt(pi)*e^(n^2)*e^n*(-3/2) - 9/8/sqrt(pi)*e^n*(2*log(2))*e^n*(-5/2) + 145/128/sqrt(pi)*e^n*(2*log(2))*e^n*(-7/2) + O((e^n)^2*(2*log(2))*e^n*(-9/2))

from sage.all import *

(2^n).factorial() / n.factorial()^2 / (n+1) # long time
1/sqrt(2)*sqrt(pi)*e^n*(2*log(2))*e^n*(-3/2) - 9/8/sqrt(pi)*e^n*(2*log(2))*e^n*(-5/2)

(continues on next page)
Note that this method substitutes the asymptotic expansion into Stirling’s formula. This substitution has to be possible which is not always guaranteed:

\[
\frac{145}{128}\sqrt{\pi} \left( e^n \right)^{2\log(2)b} n^{-7/2} + O\left( e^n \right)^{2\log(2)b} n^{-9/2}
\]

```
sage: S.<s> = AsymptoticRing(growth_group='s^QQ * log(s)^QQ', coefficient_ring=QQ, default_prec=4)
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.
```

```
>>> from sage.all import *
>>> S = AsymptoticRing(growth_group='s^QQ * log(s)^QQ', coefficient_ring=QQ, default_prec=Integer(4), names=('s',)); (s,) = S._first_ngens(1)
>>> 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()<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)) # ...
```
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:

```python
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))
```
is_exact()  
Return whether all terms of this expansion are exact.

OUTPUT:
A boolean.

EXAMPLES:

```sage
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
```

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

OUTPUT:
A boolean.

EXAMPLES:

```sage
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
>>> from sage.all import *
>>> A = AsymptoticRing('x^ZZ * log(x)^ZZ', QQ, names=('x',)); (x,) = A._first_ngens(1)
>>> (x**Integer(4) * log(x)**(-Integer(2)) + x**(-Integer(4)) * log(x)**2).is_little_o_of_one()
False
>>> (x**(-Integer(1)) * log(x)**Integer(1234) + x**(-Integer(2)) + O(x**(-4))).is_little_o_of_one()
```

(continues on next page)
See also:

limit()

limit()

Compute the limit of this asymptotic expansion.

OUTPUT:

An element of the coefficient ring.

EXAMPLES:

sage: A.<s> = AsymptoticRing("s^ZZ", SR, default_prec=3)
sage: (3 + 1/s + O(1/s^2)).limit()
3
sage: ((1+1/s)^s).limit()
e
sage: (1/s).limit()
0
sage: (s + 3 + 1/s + O(1/s^2)).limit()
Traceback (most recent call last):
...
ValueError: Cannot determine limit of s + 3 + s^{-1} + O(s^{-2})
sage: (0(s^0)).limit()
Traceback (most recent call last):
...
ValueError: Cannot determine limit of O(1)
```python
>>> from sage.all import *

>>> A = AsymptoticRing("s^ZZ", SR, default_prec=Integer(3), names=('s',)); (s, ...) = A._first_ngens(1)

>>> (Integer(3) + Integer(1)/s + O(Integer(1)/s**Integer(2))).limit()
3

>>> ((Integer(1)+Integer(1)/s)**s).limit()
e

>>> (Integer(1)/s).limit()
0

>>> (s + Integer(3) + Integer(1)/s + O(Integer(1)/s**Integer(2))).limit()
Traceback (most recent call last):
...
ValueError: Cannot determine limit of s + 3 + s^(-1) + O(s^(-2))

>>> (O(s**Integer(0))).limit()
Traceback (most recent call last):
...
ValueError: Cannot determine limit of O(1)
```

See also:

`is_little_o_of_one()`

```
log(base=None, precision=None, locals=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.
- `locals` – a dictionary which may contain the following keys and values:
  - `'log'` – value: a function. If not used, then the usual `log` is taken.

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)
```

(continues on next page)
2*log(x)
sage: log(x-1)
log(x) - x^(-1) - 1/2*x^(-2) - 1/3*x^(-3) - ... + O(x^(-21))

>>> from sage.all import *
>>> R = AsymptoticRing(growth_group='x^ZZ * log(x)^ZZ', coefficient_ring=QQ,˓
→names=('x',)); (x,) = R._first_ngens(1)
>>> log(x)
log(x)
>>> log(x^Integer(2))
2*log(x)
>>> log(x-Integer(1))
log(x) - x^(-1) - 1/2*x^(-2) - 1/3*x^(-3) - ... + O(x^(-21))

The coefficient ring is automatically extended if needed:

sage: R.<x> = AsymptoticRing(growth_group='x^ZZ * log(x)^ZZ', coefficient_˓
→ring=ZZ, default_prec=3)
sage: (49*x^3-1).log()
3*log(x) + 2*log(7) - 1/49*x^(-3) - 1/4802*x^(-6) ... + O(x^(-12))
sage: _.parent()
Asymptotic Ring <x^ZZ * log(x)^ZZ> over Symbolic Ring

If one wants to avoid this extending to the Symbolic Ring, then the following helps:

sage: L.<log7> = ZZ[]
sage: def mylog(z, base=None):
....: try:
....: if ZZ(z).is_power_of(7):
....: return log(ZZ(z), 7) * log7
....: except (TypeError, ValueError):
....: pass
....: return log(z, base)
sage: R.<x> = AsymptoticRing(growth_group='x^ZZ * log(x)^ZZ', coefficient_˓
→ring=L, default_prec=3)
sage: (49*x^3-1).log(locals={'log': mylog})
3*log(x) + 2*log7 - 1/49*x^(-3) - 1/4802*x^(-6) ... + O(x^(-12))

>>> from sage.all import *
>>> L = ZZ['log7']; (log7,) = L._first_ngens(1)
>>> def mylog(z, base=None):
... try:
... if ZZ(z).is_power_of(Integer(7)):
... return log(ZZ(z), Integer(7)) * log7
... except (TypeError, ValueError):
... pass
... return log(z, base)

(continues on next page)
A \log\text{-}function can also be specified to always be used with the asymptotic ring:

```python
code
sage: R.<x> = AsymptoticRing(growth_group=x^ZZ * log(x)^ZZ, coefficient_ring=L, default_prec=3, locals={log: mylog})
sage: log(49*x^3-1)
3*log(x) + 2*log7 - 1/49*x^(-3) - 1/4802*x^(-6) - 1/352947*x^(-9) + O(x^(-12))
```

**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:**

```python
code
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)
```

**monomial\_coefficient** (**monomial**)

Return the coefficient in the base ring of the given monomial in this expansion.

**INPUT:**

- **monomial** – a monomial element which can be converted into the asymptotic ring of this element
OUTPUT:

An element of the coefficient ring.

EXAMPLES:

```python
sage: R.<m, n> = AsymptoticRing("m^QQ*n^QQ", QQ)
sage: ae = 13 + 42/n + 2/n/m + O(n^-2)
sage: ae.monomial_coefficient(1/n)
42
sage: ae.monomial_coefficient(1/n^3)
0
sage: R.<n> = AsymptoticRing("n^QQ", ZZ)
sage: ae.monomial_coefficient(1/n)
42
sage: ae.monomial_coefficient(1)
13
```

```python
>>> from sage.all import *
>>> R = AsymptoticRing("m^QQ*n^QQ", QQ, names=('m', 'n',)); (m, n) = R._first_ngens(2)
>>> ae = Integer(13) + Integer(42)/n + Integer(2)/n/m + O(n**-Integer(2))
>>> ae.monomial_coefficient(Integer(1)/n)
42
>>> ae.monomial_coefficient(Integer(1)/n**Integer(3))
0
>>> R = AsymptoticRing("n^QQ", ZZ, names=('n',)); (n) = R._first_ngens(1)
>>> ae.monomial_coefficient(Integer(1)/n)
42
>>> ae.monomial_coefficient(Integer(1))
13
```

**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 plausibility check for asymptotic expansion created in some way.
EXAMPLES:

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

```python
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:

```python
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()`:

```python
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()`

`pow()`

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:

```plaintext
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)

>>> from sage.all import *

>>> Q = AsymptoticRing(growth_group='x^QQ', coefficient_ring=QQ, names=('x',)); (x,) = Q._first_ngens(1)
>>> x**(Integer(1)/Integer(7))
x^(1/7)
>>> (x**(Integer(1)/Integer(2)) + O(x**Integer(0)))**Integer(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))^(1/2)
y^(-4) + O(y^(-5))

>>> from sage.all import *

>>> Z = AsymptoticRing(growth_group='y^ZZ', coefficient_ring=ZZ, names=('y',)); (y,) = Z._first_ngens(1)
>>> y**(Integer(1)/Integer(7))
y^(1/7)
>>> _.parent()
Asymptotic Ring <y^QQ> over Rational Field
>>> (y**Integer(2) + O(y))**(Integer(1)/Integer(2))
y + O(1)
>>> (y**Integer(2) + O(y))**(Integer(1)/Integer(2))
y^(-4) + O(y^(-5))

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

>>> from sage.all import *

>>> B = AsymptoticRing(growth_group='z^QQ * log(z)^QQ', coefficient_ring=QQ, names=('z',)); (z,) = B._first_ngens(1)
>>> (z**Integer(2) + O(z))**(Integer(1)/Integer(2))
z + O(1)
```

Chapter 4. Asymptotic Expansions — Table of Contents
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 * Signs^x> over Symbolic Ring

>>> from sage.all import *
>>> A = AsymptoticRing('QQ^x * x^SR * log(x)^ZZ', QQ, names=('x',)); (x,) = A._first_ngens(1)
>>> x * Integer(2)**x
2^x*x
>>> Integer(5)**x * Integer(2)**x
10^x
>>> Integer(2)**log(x)
x^((log(2)))
>>> Integer(2)**(x + Integer(1)/x)
2^x + log(2)*2^x*x^(-1) + 1/2*log(2)^2*2^x*x^(-2) + ... + O(2^x*x^(-20))
>>> _.parent()
Asymptotic Ring <QQ^x * x^SR * log(x)^QQ * Signs^x> 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 * Signs^c> 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 * Signs^c> over Rational Field

>>> from sage.all import *
>>> C = AsymptoticRing(growth_group='QQ^c * c^QQ', coefficient_ring=QQ,
˓→default_prec=Integer(5), names=('c',)); (c,) = C._first_ngens(1)
>>> (Integer(3) + Integer(1)/c**Integer(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))
>>> _.parent()
Asymptotic Ring <QQ^c * c^QQ * Signs^c> over Rational Field

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

Return the power of base to this asymptotic expansion.

INPUT:

4.1. Asymptotic Ring 35
• 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.

• locals – a dictionary which may contain the following keys and values:
  – 'log' – value: a function. If not used, then the usual log is taken.

**OUTPUT:**
An asymptotic expansion.

**EXAMPLES:**

```python
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))
```

```python
>>> from sage.all import *
>>> A = AsymptoticRing('x^ZZ', QQ, names=('x',)); (x,) = A._first_ngens(1)
>>> (Integer(1)/x).rpow('e', precision=Integer(5))
1 + x^(-1) + 1/2*x^(-2) + 1/6*x^(-3) + 1/24*x^(-4) + O(x^(-5))
```

**show()**

Pretty-print this asymptotic expansion.

**OUTPUT:**
Nothing, the representation is printed directly on the screen.

**EXAMPLES:**

```python
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()
1/2*pi*5^x*x^(42/17) - sqrt(euler_gamma)*log(x)^(-7/8)
```

```python
>>> from sage.all import *
>>> A = AsymptoticRing('QQ^x * x^QQ * log(x)^QQ', SR.subring(no_˓→variables=True)), names=('x',)); (x,) = A._first_ngens(1)
>>> (pi/Integer(2) * Integer(5)**x * x**(Integer(42)/Integer(17)) - sqrt(euler_gamma) * log(x)**(-Integer(7)/Integer(8))).show()
1/2*pi*5^x*x^(42/17) - sqrt(euler_gamma)*log(x)^(-7/8)
```

**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:**
Asymptotic Expansions, Release 10.4

```python
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))
```

```python
>>> from sage.all import *
>>> A = AsymptoticRing(growth_group='s^QQ', coefficient_ring=QQ, names=('s',)); (s,) = A._first_ngens(1)
>>> s.sqrt()
s^(1/2)
>>> a = (Integer(1) + Integer(1)/s).sqrt(precision=Integer(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()`.

```python
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:**

```python
sage: A.<x> = AsymptoticRing(growth_group='(e^x)^QQ * x^ZZ * log(x)^ZZ',
                           coefficient_ring=QQ, default_prec=5)
```

```python
>>> from sage.all import *
>>> A = AsymptoticRing(growth_group='(e^x)^QQ * x^ZZ * log(x)^ZZ',
                      coefficient_ring=QQ, default_prec=Integer(5), names=('x',)); (x,) = A._
                      first_ngens(1)
```

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

4.1. Asymptotic Ring 37
```python
>>> from sage.all import *
>>> (e**x * x**Integer(2) + log(x)).subs(x=SR('s'))
s^2*e^s + log(s)
>>> _.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: (x**Integer(3) + x + log(x)).subs(x=Integer(5)+x).truncate(Integer(5))
x^3 + 15*x^2 + 76*x + log(x) + 130 + O(x^(-1))
>>> _.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: (e**x * x**Integer(2) + log(x)).subs(x=Integer(2)*x)
4*(e^x)^2*x^2 + log(x) + log(2)
>>> _.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) + 2*log(2) + 4 + 1/2*x^(-1) + O(x^(-2))
sage: _.parent()
Asymptotic Ring <(e^x)^QQ * x^ZZ * log(x)^ZZ> over Symbolic Ring
```

```
sage: (x**Integer(2) + log(x)).subs(x=Integer(4)*x+Integer(2)).truncate(Integer(5))
16*x^2 + 16*x + log(x) + 2*log(2) + 4 + 1/2*x^(-1) + O(x^(-2))
>>> _.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
```

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

See also:

```
sage.symbolic.expression.Expression.subs()
```
**property summands**

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

**EXAMPLES:**

```plaintext
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 will be an element of `R`. If `None`, then the symbolic ring is used.

**OUTPUT:**

A symbolic expression.

**EXAMPLES:**

```plaintext
sage: A.<x, y, z> = AsymptoticRing(growth_group=x^ZZ * y^QQ * log(y)^QQ * →(QQ_+)^z * z^QQ, coefficient_ring=QQ)
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)))
```

Asymptotic Expansions, Release 10.4

(continued from previous page)

\[
\frac{1}{8} \cdot \left(\frac{1}{8}\right)^z x^3 y^{3/2} z^{3/2} \log(y)^{3/2} + \text{Order}\left(\frac{1}{2} z x\sqrt{y} \sqrt{z} \sqrt{\log(y)}\right)
\]

\textbf{truncate (precision=\textit{None})}

Truncate this asymptotic expansion.

**INPUT:**

- \textit{precision} – a positive integer or \textit{None}. Number of summands that are kept. If \textit{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 \textit{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:**

```python
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
```

```python
>>> from sage.all import *
>>> R = AsymptoticRing('x^ZZ', QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> ex = sum(x**k for k in range(Integer(5))); ex
x^4 + x^3 + x^2 + x + 1
>>> ex.truncate(precision=Integer(2))
x^4 + x^3 + O(x^2)
>>> ex.truncate(precision=Integer(0))
O(x^4)
>>> ex.truncate()
x^4 + x^3 + x^2 + x + 1
```

\textbf{variable_names()}

Return the names of the variables of this asymptotic expansion.

**OUTPUT:**

A tuple of strings.

**EXAMPLES:**

```python
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',)
```
class sage.rings.asymptotic.asymptotic_ring.AsymptoticRing(growth_group, coefficient_ring, category, default_prec, term_monoid_factory, locals)

Bases: Parent, UniqueRepresentation, WithLocals

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.

- **term_monoid_factory** – a TermMonoidFactory. If None, then DefaultTermMonoidFactory is used.

- **locals** – a dictionary which may contain the following keys and values: 
Asymptotic Expansions — Table of Contents

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
```

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

```sage
from sage.rings.asymptotic.growth_group import GrowthGroup
g = GrowthGroup('x^QQ')
R2_x.<x> = AsymptoticRing(growth_group=g, coefficient_ring=QQ); R2_x
```

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
```

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

```sage
R_log = AsymptoticRing(growth_group=log(x)^ZZ, coefficient_ring=QQ); R_log
```

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:

```python
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):

```python
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):

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

It is possible to customize the terms in an asymptotic expansion:

```python
sage: from sage.rings.asymptotic.term_monoid import ExactTermMonoid, OTermMonoid
sage: from sage.rings.asymptotic.term_monoid import TermMonoidFactory
sage: class MyExactTermMonoid(ExactTermMonoid):
    ....:     pass
sage: class MyOTermMonoid(OTermMonoid):
    ....:     pass
sage: MyTermMonoid = TermMonoidFactory(MyTermMonoid, 
    ....:     exact_term_monoid_class=MyExactTermMonoid, 
    ....:     O_term_monoid_class=MyOTermMonoid)
sage: G = GrowthGroup('x^ZZ')
sage: A.<n> = AsymptoticRing(growth_group=G, coefficient_ring=QQ, term_monoid_= 
    ....:     factory=MyTermMonoid)
sage: a = A.an_element(); a
1/8*x^3 + O(x)
sage: for t in a.summands.elements_topological(reverse=True):
    ....:     print(t, type(t))
1/8*x^3 <class '__main__.MyExactTermMonoid_with_category.element_class'>
O(x) <class '__main__.MyOTermMonoid_with_category.element_class'>
```

(continues on next page)
Asymptotic Expansions, Release 10.4

... pass
>>> MyTermMonoid = TermMonoidFactory('MyTermMonoid',
    exact_term_monoid_class=MyExactTermMonoid,
    O_term_monoid_class=MyOTermMonoid)

>>> G = GrowthGroup('x^ZZ')
>>> A = AsymptoticRing(growth_group=G, coefficient_ring=QQ, term_monoid_factory=MyTermMonoid, names=('n',)); (n,) = A._first_ngens(1)
>>> a = A.an_element(); a
1/8*x^3 + O(x)

static B(expression, valid_from=0)
Create a B-term.

INPUT:

- valid_from – dictionary mapping variable names to lower bounds for the corresponding variable. The bound implied by this term is valid when all variables are at least their corresponding lower bound. If a number is passed to valid_from, then the lower bounds for all variables of the asymptotic expansion are set to this number

OUTPUT:

A B-term

EXAMPLES:

sage: A.<x> = AsymptoticRing(growth_group='x^ZZ * QQ^y', coefficient_ring=QQ)
sage: A.B(2*x^3, {x: 5})
B(2*x^3, x >= 5)

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:
Asymptotic Expansions, Release 10.4

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

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

**property coefficient_ring**

The coefficient ring of this asymptotic ring.

**EXAMPLES:**

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

>>> from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
>>> AR.coefficient_ring
Integer Ring
```

**coefficients_of_generating_function**

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.
- `error_term` – (default: None) an asymptotic expansion. If None, then this is interpreted as zero. The contributions of the coefficients are added to `error_term` during Singularity Analysis.

**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:

4.1. Asymptotic Ring 45
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))})

>>> from sage.all import *
>>> def catalan(z):
...     return (Integer(1)-(Integer(1)-Integer(4)*z)**(Integer(1)/Integer(2)))/(Integer(2)*z)
>>> B = AsymptoticRing('QQ^n * n^QQ', QQ, names=('n',)); (n,) = B._first_ngens(1)
>>> B.coefficients_of_generating_function(catalan, (Integer(1)/Integer(4),), precision=Integer(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))
>>> B.coefficients_of_generating_function(catalan, (Integer(1)/Integer(4),), precision=Integer(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))

>>> from sage.all import *
>>> def logarithmic(z):
...     return -log(Integer(1)-z)
>>> B.coefficients_of_generating_function(logarithmic, (Integer(1),), precision=Integer(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
>>> from sage.all import *
>>> def harmonic(z):
...     return -log(Integer(1)-z)/(Integer(1)-z)
>>> B = AsymptoticRing('QQ\n * n^QQ * log(n)^QQ', QQ, names=('n',)); (n,) = B.
˓→_first_ngens(1)
>>> ex = B.coefficients_of_generating_function(harmonic, (Integer(1),),
˓→precision=Integer(13)); ex
log(n) + euler_gamma + 1/2*n^(-1) - 1/12*n^(-2) + 1/120*n^(-4)
+ O(n^(-6))
>>> ex.has_same_summands(asymptotic_expansions.HarmonicNumber(
...     'n', precision=Integer(5)))
True

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

In the following example, the result is an exact asymptotic expression for sufficiently large \( n \) (i.e., there might
be finitely many exceptional values). This is encoded by an \( O(0) \) error term:

```sage
>>> def f(z):
...     return z/(1-z)
>>> B.coefficients_of_generating_function(f, (1,), precision=3)
Traceback (most recent call last):
... NotImplementedOZero: got 1 + O(0)
The error term O(0) means 0 for sufficiently large \( n \).
```

In this case, we can manually intervene by adding an error term that suits us:

```sage
>>> from sage.all import *
>>> def f(z):
...     return z/(Integer(1)-z)
>>> B.coefficients_of_generating_function(f, (Integer(1),),
˓→precision=Integer(3))
Traceback (most recent call last):
... NotImplementedOZero: got 1 + O(0)
The error term O(0) means 0 for sufficiently large \( n \).
```

```sage
>>> B.coefficients_of_generating_function(f, (1,), precision=3,
˓→error_term=O(n^(-100)))
1 + O(n^(-100))
```

```sage
>>> from sage.all import *
>>> B.coefficients_of_generating_function(f, (Integer(1),),
˓→precision=Integer(3),
˓→error_term=O(n**-Integer(100)))
1 + O(n^(-100))
```

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:

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

```python
>>> from sage.all import *

>>> A = AsymptoticRing(growth_group='x^ZZ * QQ^y', coefficient_ring=QQ)

>>> A.construction()
(AsymptoticRing<x^ZZ * QQ^y * Signs^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:

```python
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
```

```python
>>> from sage.all import *

>>> R = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)

>>> R.create_summand('O', x**Integer(2))
O(x^2)

>>> R.create_summand('exact', growth=x**Integer(456), coefficient=Integer(123))
123*x^456

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

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

```python
>>> from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=IntegerRing())
>>> AR.default_prec
20
>>> AR = AsymptoticRing('x^ZZ', IntegerRing(), default_prec=Integer(123))
>>> 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:**

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

```python
>>> from sage.all import *
>>> R = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=IntegerRing(), names=('x',))
>>> (x,) = R._first_ngens(1)
>>> R.gen()
x
```

**gens()**

Return a tuple with generators of this asymptotic ring.

**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)

>> from sage.all import *
>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ, names=('x',)); (x,) = AR._first_ngens(1)
>> AR.gens()
(x,)
>> B = AsymptoticRing(growth_group='y^ZZ * z^ZZ', coefficient_ring=QQ, names=('y', 'z',)); (y, z,) = B._first_ngens(2)
>> B.gens()
(y, z)

property 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.

OUTPUT:
An integer.

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

>>> from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ, names=('x',)); (x,) = AR._first_ngens(1)
>>> AR.ngens()
1

some_elements()
Return some elements of this term monoid.
See TestSuite for a typical use case.

OUTPUT:

An iterator.

EXAMPLES:

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

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

term_monoid(type)

Return the term monoid of this asymptotic ring of specified type.

INPUT:

• type – ‘O’ or ‘exact’, or an instance of an existing term monoid. See TermMonoidFactory for more details.

OUTPUT:

A term monoid object derived from GenericTermMonoid.

EXAMPLES:

```python
sage: AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.term_monoid('exact')
Exact Term Monoid x^ZZ with coefficients in Integer Ring
sage: AR.term_monoid('O')
O-Term Monoid x^ZZ with implicit coefficients in Integer Ring
sage: AR.term_monoid(AR.term_monoid('exact'))
Exact Term Monoid x^ZZ with coefficients in Integer Ring
```
>>> from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
>>> AR.term_monoid('exact')
Exact Term Monoid x^ZZ with coefficients in Integer Ring
>>> AR.term_monoid('O')
O-Term Monoid x^ZZ with implicit coefficients in Integer Ring
>>> AR.term_monoid(AR.term_monoid('exact'))
Exact Term Monoid x^ZZ with coefficients in Integer Ring

property term_monoid_factory

The term monoid factory of this asymptotic ring.

EXAMPLES:

sage: AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
sage: AR.term_monoid_factory
Term Monoid Factory sage.rings.asymptotic.term_monoid.˓
→DefaultTermMonoidFactory

>>> from sage.all import *
>>> AR = AsymptoticRing(growth_group='x^ZZ', coefficient_ring=ZZ)
>>> AR.term_monoid_factory
Term Monoid Factory sage.rings.asymptotic.term_monoid.
→DefaultTermMonoidFactory

See also:

(Asymptotic) Term Monoids

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')

>>> from sage.all import *
>>> A = AsymptoticRing(growth_group='x^ZZ * QQ^y', coefficient_ring=QQ)
>>> A.variable_names()
('x', 'y')
• growth_group – a partially ordered group (see AsymptoticRing or (Asymptotic) Growth Groups for details).
• default_prec – None (default) or an integer.
• category – None (default) or a category.
• cls – AsymptoticRing (default) or a derived class.

EXAMPLES:

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

```python
>>> from sage.all import *
>>> AsymptoticRing(growth_group='x^ZZ', coefficient_ring=QQ).construction()  # indirect doctest
(AsymptoticRing<x^ZZ>, Rational Field)
```

See also:
AsymptoticRing, AsymptoticRing, sage.rings.asymptotic.growth_group.
AbstractGrowthGroupFunctor, sage.rings.asymptotic.growth_group.
ExponentialGrowthGroupFunctor, sage.rings.asymptotic.growth_group.

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>
```

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

rank = 13
exception sage.rings.asymptotic.asymptotic_ring.NoConvergenceError

Bases: RuntimeError

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

4.2 Common Asymptotic Expansions

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)
```

\[
sqrt{2}\sqrt{\pi} e^{n \log(n)} (e^n)^{-1} n^{1/2} + \\
1/12 \sqrt{2} \sqrt{\pi} e^{n \log(n)} (e^n)^{-1} n^{-1/2} + \\
1/288 \sqrt{2} \sqrt{\pi} e^{n \log(n)} (e^n)^{-1} n^{-3/2} + \\
O(e^{n \log(n)} (e^n)^{-1} n^{-5/2})
\]

```
>>> from sage.all import *
```

```
>>> asymptotic_expansions.Stirling(n, precision=Integer(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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HarmonicNumber()</td>
<td>harmonic numbers</td>
</tr>
<tr>
<td>Stirling()</td>
<td>Stirling's approximation formula for factorials</td>
</tr>
<tr>
<td>log_Stirling()</td>
<td>the logarithm of Stirling’s approximation formula for factorials</td>
</tr>
<tr>
<td>Binomial_kn_over_n()</td>
<td>an asymptotic expansion of the binomial coefficient</td>
</tr>
<tr>
<td>SingularityAnalysis()</td>
<td>an asymptotic expansion obtained by singularity analysis</td>
</tr>
<tr>
<td>ImplicitExpansion()</td>
<td>the singular expansion of a function ( y(z) ) satisfying ( y(z) = z\Phi(y(z)) )</td>
</tr>
<tr>
<td>ImplicitExpansionPeriodicPart()</td>
<td>the singular expansion of the periodic part of a function ( y(z) ) satisfying ( y(z) = z\Phi(y(z)) )</td>
</tr>
<tr>
<td>InverseFunctionAnalysis()</td>
<td>coefficient growth of a function ( y(z) ) defined implicitly by ( y(z) = z\Phi(y(z)) )</td>
</tr>
</tbody>
</table>

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: 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()
• ImplicitExpansion()
• ImplicitExpansionPeriodicPart()
• InverseFunctionAnalysis()

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

Return the asymptotic expansion of the binomial coefficient $\binom{kn}{n}$.

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 $\sqrt{\frac{k}{2\pi(k-1)}}$ 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
```
\[
\binom{n}{k} \sim \frac{1}{\sqrt{\pi}} 4^n n^{\frac{1}{2}} \left( -\frac{1}{8} \sqrt{\pi} 4^n n^{\frac{3}{2}} + \frac{1}{128} \sqrt{\pi} 4^n n^{\frac{5}{2}} + O(4^n n^{\frac{7}{2}}) \right)
\]

\[
\binom{n}{3} \sim \frac{1}{2} \sqrt{3} \sqrt{\pi} \left( \frac{27}{4} \right)^n n^{\frac{1}{2}} \left( -\frac{7}{144} \sqrt{3} \sqrt{\pi} \left( \frac{27}{4} \right)^n n^{\frac{3}{2}} + \frac{49}{20736} \sqrt{3} \sqrt{\pi} \left( \frac{27}{4} \right)^n n^{\frac{5}{2}} + O(\left( \frac{27}{4} \right)^n n^{\frac{7}{2}}) \right)
\]

\[
\binom{n}{7/5} \sim \frac{1}{2} \sqrt{7} \sqrt{\pi} \left( \frac{7}{10} \frac{7^{2/5} 2^{3/5}}{2} \right)^n n^{\frac{1}{2}} \left( -\frac{13}{112} \sqrt{7} \sqrt{\pi} \left( \frac{7}{10} \frac{7^{2/5} 2^{3/5}}{2} \right)^n n^{\frac{3}{2}} + \frac{139}{12544} \sqrt{7} \sqrt{\pi} \left( \frac{7}{10} \frac{7^{2/5} 2^{3/5}}{2} \right)^n n^{\frac{5}{2}} + O\left( \frac{7}{10} \frac{7^{2/5} 2^{3/5}}{2} \right)^n n^{\frac{7}{2}} \right)
\]

\[
\text{\texttt{static HarmonicNumber}} \text{\texttt{(var, precision=None, skip_constant_summand=False)}}
\]

Return the asymptotic expansion of a harmonic number.

INPUT:

- \texttt{var} – a string for the variable name.

- \texttt{precision} – (default: None) an integer. If None, then the default precision of the asymptotic ring is used.

- \texttt{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 \texttt{(if False)} to Rational Field \texttt{(if True)}.
OUTPUT:

An asymptotic expansion.

EXAMPLES:

```sage
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))
```

```python
>>> from sage.all import *
```

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

**static ImplicitExpansion**(var, phi=None, tau=None, precision=None)

Return the singular expansion for a function \(y(z)\) defined implicitly by \(y(z) = z\Phi(y(z))\).

The function \(\Phi\) is assumed to be analytic around 0. Furthermore, \(\Phi\) is not allowed to be an affine-linear function and we require \(\Phi(0) \neq 0\).

Furthermore, it is assumed that there is a unique positive solution \(\tau\) of \(\Phi(\tau) - \tau\Phi'(\tau) = 0\).

All details are given in Chapter VI.7 of [FS2009].

INPUT:

- **var** – a string for the variable name.
- **phi** – the function \(\Phi\). See the extended description for assumptions on \(\Phi\).
- **tau** – (default: None) the fundamental constant described in the extended description. If None, then \(\tau\) is determined automatically if possible.
- **precision** – (default: None) an integer. If None, then the default precision of the asymptotic ring is used.

OUTPUT:

An asymptotic expansion.

Note: In the given case, the radius of convergence of the function of interest is known to be \(\rho = \tau/\Phi(\tau)\). Until Issue #20050 is implemented, the variable in the returned asymptotic expansion represents a singular element of the form \((1 - z/\rho)^{-1}\), for the variable \(z \to \rho\).

EXAMPLES:

We can, for example, determine the singular expansion of the well-known tree function \(T\) (which satisfies \(T(z) = z\exp(T(z))\)):

```sage
sage: asymptotic_expansions.ImplicitExpansion('Z', phi=exp, precision=8)
doctest:warning
...
FutureWarning: This class/method/function is marked as experimental. It, its functionality or its interface might change without a formal deprecation. See https://github.com/sagemath/sage/issues/20050 for details.
1 - sqrt(2)*Z^(-1/2) + 2/3*Z^(-1) - 11/36*sqrt(2)*Z^(-3/2) + 43/135*Z^(-2) - 769/4320*sqrt(2)*Z^(-5/2) + 1768/8505*Z^(-3) + O(Z^(-7/2))
```

```python
>>> from sage.all import *
```

```sage
>>> asymptotic_expansions.ImplicitExpansion('Z', phi=exp, _
```

(continues on next page)
Another classical example in this context is the generating function $B(z)$ enumerating binary trees with respect to the number of inner nodes. The function satisfies $B(z) = z(1+2B(z)+B(z)^2)$, which can also be solved explicitly, yielding $B(z) = \frac{1-\sqrt{1-4z}}{2z} - 1$. We compare the expansions from both approaches:

```python
sage: def B(z):
....:     return (1 - sqrt(1 - 4*z))/(2*z) - 1

sage: A.<Z> = AsymptoticRing(Z^QQ, QQ, default_prec=Integer(3), names=(Z,)); (Z,)

sage: asymptotic_expansions.ImplicitExpansion(Z, phi=lambda u: 1 + 2*u + u^2, precision=Integer(7))
```

Neither $\tau$ nor $\Phi$ have to be known explicitly, they can also be passed symbolically:

```python
sage: tau = var('tau')
sage: phi = function('phi')
sage: asymptotic_expansions.ImplicitExpansion('Z', phi=phi, tau=tau, precision=Integer(3))
```

Neither $\tau$ nor $\Phi$ have to be known explicitly, it's functionality or its interface might change without a formal deprecation. See https://github.com/sagemath/sage/issues/20050 for details.
Note that we do not check whether a passed $\tau$ actually satisfies the requirements. Only the first of the following expansions is correct:

\begin{verbatim}
sage: asymptotic_expansions.ImplicitExpansion(Z, ....: phi=lambda u: 1 + 2*u + u^2, precision=5) # correct expansion 1 - 2*Z^(-1/2) + 2*Z^(-1) - 2*Z^(-3/2) + O(Z^(-2))
sage: asymptotic_expansions.ImplicitExpansion('Z', phi=lambda u: 1 + 2*u + u^2, tau=2, precision=5) Traceback (most recent call last):
...
ZeroDivisionError: symbolic division by zero
sage: asymptotic_expansions.ImplicitExpansion('Z', phi=lambda u: 1 + 2*u + u^2, tau=3, precision=5) 3 - 4*I*sqrt(3)*Z^(-1/2) + 6*I*sqrt(3)*Z^(-3/2) + O(Z^(-2))
\end{verbatim}

See also: `ImplicitExpansionPeriodicPart()`, `InverseFunctionAnalysis()`.

\texttt{static ImplicitExpansionPeriodicPart}(var, phi, period, tau=None, precision=None)

Return the singular expansion for the periodic part of a function $y(z)$ defined implicitly by $y(z) = z\Phi(y(z))$.

The function $\Phi$ is assumed to be analytic around 0. Furthermore, $\Phi$ is not allowed to be an affine-linear function and we require $\Phi(0) \neq 0$. For an integer $p$, $\Phi$ is called $p$-periodic if we have $\Psi(u^p) = \Phi(u)$ for a power series $\Psi$ where $p$ is maximal.

Furthermore, it is assumed that there is a unique positive solution $\tau$ of $\Phi(\tau) - \tau\Phi'(\tau) = 0$.

If $\Phi$ is $p$-periodic, then we have $y(z) = zg(z^p)$. This method returns the singular expansion of $g(z)$.

\textbf{INPUT:}

- var – a string for the variable name.
- phi – the function $\Phi$. See the extended description for assumptions on $\Phi$.
- period – the period of the function $\Phi$. See the extended description for details.
- tau – (default: None) the fundamental constant described in the extended description. If None, then $\tau$ is determined automatically if possible.
- precision – (default: None) an integer. If None, then the default precision of the asymptotic ring is used.

\textbf{OUTPUT:}

An asymptotic expansion.
Note: In the given case, the radius of convergence of the function of interest is known to be \( \rho = \tau / \Phi(\tau) \).
Until Issue #20050 is implemented, the variable in the returned asymptotic expansion represents a singular element of the form \((1 - z/\rho)^{-1}\), for the variable \( z \to \rho \).

See also:

*ImplicitExpansion()*, *InverseFunctionAnalysis()*.  

EXAMPLES:

The generating function enumerating binary trees with respect to tree size satisfies \( B(z) = z(1 + B(z)^2) \). This function can be written as \( B(z) = zg(z^2) \), and as \( B(z) \) can be determined explicitly we have \( g(z) = \frac{1 - \sqrt{1 - 4z}}{2z} \). We compare the corresponding expansions:

```python
sage: asymptotic_expansions.ImplicitExpansionPeriodicPart('Z', phi=lambda u: u + u^2, period=2,...)
```

FutureWarning: This class/method/function is marked as experimental. It, its functionality or its interface might change without a formal deprecation. See https://github.com/sagemath/sage/issues/20050 for details.

\[ 2 - 2z^(-1/2) + 2z^(-1) - 2z^(-3/2) + 2z^(-2) - 2z^(-5/2) + O(z^(-3)) \]

```python
def g(z):
    return (1 - sqrt(1 - 4*z))/(2*z)
sage: A.<Z> = AsymptoticRing(Z^QQ, QQ, default_prec=3)
sage: g((1 - 1/Z)/4)
```

\[ 2 - 2z^(-1/2) + 2z^(-1) - 2z^(-3/2) + 2z^(-2) - 2z^(-5/2) + O(z^(-3)) \]

```python
>>> from sage.all import *
>>> asymptotic_expansions.ImplicitExpansionPeriodicPart('Z', phi=lambda u: Integer(1) + u**Integer(2), ...
    period=Integer(2),
    precision=Integer(7))
```

FutureWarning: This class/method/function is marked as experimental. It, its functionality or its interface might change without a formal deprecation. See https://github.com/sagemath/sage/issues/20050 for details.

\[ 2 - 2*z^(-1/2) + 2*z^(-1) - 2*z^(-3/2) + 2*z^(-2) - 2*z^(-5/2) + O(z^(-3)) \]

```python
def g(Integer(x) - sqrt(Integer(1) - Integer(4)*x))/(Integer(2)*x)
>>> A = AsymptoticRing('Z^QQ', QQ, default_prec=Integer(3), names=('Z',)); (Z, ...
    A._first_ngens(1)
>>> g((Integer(1) - Integer(1)/2)/Integer(4))
```

\[ 2 - 2*z^(-1/2) + 2*z^(-1) - 2*z^(-3/2) + 2*z^(-2) - 2*z^(-5/2) + O(z^(-3)) \]

**static** *InverseFunctionAnalysis*(var, phi, tau=None, period=1, precision=None)

Return the coefficient growth of a function \( y(z) \) defined implicitly by \( y(z) = z\Phi(y(z)) \).

The function \( \Phi \) is assumed to be analytic around 0. Furthermore, \( \Phi \) is not allowed to be an affine-linear function and we require \( \Phi(0) \neq 0 \). For an integer \( p \), \( \Phi \) is called \( p \)-periodic if we have \( \Psi(u^p) = \Phi(u) \) for a power series \( \Psi \) where \( p \) is maximal.

Furthermore, it is assumed that there is a unique positive solution \( \tau \) of \( \Phi(\tau) - \tau\Phi'(\tau) = 0 \).

INPUT:
• \textbf{\texttt{var}} – a string for the variable name.
• \textbf{\texttt{phi}} – the function \( \Phi \). See the extended description for assumptions on \( \Phi \).
• \textbf{\texttt{tau}} – (default: \texttt{None}) the fundamental constant described in the extended description. If \texttt{None}, then \( \tau \) is determined automatically if possible.
• \textbf{\texttt{period}} – (default: 1) the period of the function \( \Phi \). See the extended description for details.
• \textbf{\texttt{precision}} – (default: \texttt{None}) an integer. If \texttt{None}, then the default precision of the asymptotic ring is used.

\textbf{OUTPUT:}

An asymptotic expansion.

\textbf{Note:} It is not checked that the passed period actually fits to the passed function \( \Phi \).

The resulting asymptotic expansion is only valid for \( n \equiv 1 \mod p \), where \( p \) is the period. All other coefficients are 0.

\textbf{EXAMPLES:}

There are \( C_n \) (the \( n \)-th Catalan number) different binary trees of size \( 2n + 1 \), and there are no binary trees with an even number of nodes. The corresponding generating function satisfies \( B(z) = z(1 + B(z)^2) \), which allows us to compare the asymptotic expansions for the number of binary trees of size \( n \) obtained via \( C_n \) and obtained via the analysis of \( B(z) \):

\begin{verbatim}
sage: assume(SR.an_element() > 0)
sage: A.<n> = AsymptoticRing('QQ^n * n^QQ', SR)
sage: binomial_expansion = asymptotic_expansions.Binomial_kn_over_n(n, k=Integer(2), precision=Integer(3))
sage: catalan_expansion = binomial_expansion / (n+Integer(1))
sage: catalan_expansion.subs(n=(n-Integer(1))/Integer(2))
2*sqrt(1/2)/sqrt(pi)*2^n*n^(-3/2) - 3/2*sqrt(1/2)/sqrt(pi)*2^n*n^(-5/2) + 25/16*sqrt(1/2)/sqrt(pi)*2^n*n^(-7/2) + O(2^n*n^(-9/2))
sage: asymptotic_expansions.InverseFunctionAnalysis(n, phi=lambda u: 1 + u^2, period=Integer(2),....: tau=Integer(1), precision=8)
2*sqrt(1/2)/sqrt(pi)*2^n*n^(-3/2) - 3/2*sqrt(1/2)/sqrt(pi)*2^n*n^(-5/2) + 25/16*sqrt(1/2)/sqrt(pi)*2^n*n^(-7/2) + O(2^n*n^(-9/2))

>>> from sage.all import *
>>> assume(SR.an_element() > Integer(0))
>>> A = AsymptoticRing('QQ^n * n^QQ', SR, names=('n',)); (n,) = A._first_ngens(1)
>>> binomial_expansion = asymptotic_expansions.Binomial_kn_over_n(n,...
...=Integer(2), precision=Integer(3))
>>> catalan_expansion = binomial_expansion / (n+Integer(1))
>>> catalan_expansion.subs(n=(n-Integer(1))/Integer(2))
2*sqrt(1/2)/sqrt(pi)*2^n*n^(-3/2) - 3/2*sqrt(1/2)/sqrt(pi)*2^n*n^(-5/2) + 25/16*sqrt(1/2)/sqrt(pi)*2^n*n^(-7/2) + O(2^n*n^(-9/2))
>>> asymptotic_expansions.InverseFunctionAnalysis(n, phi=lambda u: Integer(1)...+
...+ u**Integer(2), period=Integer(2), tau=Integer(1),....: precision=Integer(8))
2*sqrt(1/2)/sqrt(pi)*2^n*n^(-3/2) - 3/2*sqrt(1/2)/sqrt(pi)*2^n*n^(-5/2) + 25/16*sqrt(1/2)/sqrt(pi)*2^n*n^(-7/2) + O(2^n*n^(-9/2))
\end{verbatim}
The code in the aperiodic case is more efficient, however. Therefore, it is recommended to use combinatorial identities to reduce to the aperiodic case. In the example above, this is well-known: we now count binary trees with $n$ internal nodes. The corresponding generating function satisfies $B(z) = z(1 + 2B(z) + B(z)^2)$:

$$B(z) = \frac{z}{1 + 2z + z^2}.$$ 

```
sage: catalan_expansion
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^(-9/2))
```

```
sage: asymptotic_expansions.InverseFunctionAnalysis(n, phi=lambda u: 1 + 2*u → u^2,
                                   tau=1, precision=8)
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^(-9/2))
```

```
sage: forget()
```

```
>>> from sage.all import *
>>> catalan_expansion
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^(-9/2))
```

```
>>> asymptotic_expansions.InverseFunctionAnalysis(n, phi=lambda u: Integer(1) → Integer(2)*u + u^2*Integer(2),
                                   tau=Integer(1), precision=Integer(8))
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^(-9/2))
```

```
>>> forget()
```

See also:

ImplicitExpansion(), ImplicitExpansionPeriodicPart().

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

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

More precisely, this extracts the $n$-th coefficient

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

(if normalized=True, the default) or

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

(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 singularity.
• 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
```

```
>>> from sage.all import *

>>> asymptotic_expansions.SingularityAnalysis('n', alpha=Integer(1))
1
>>> asymptotic_expansions.SingularityAnalysis('n', alpha=Integer(2))
n + 1
>>> asymptotic_expansions.SingularityAnalysis('n', alpha=Integer(3))
1/2*n^2 + 3/2*n + 1
>>> _.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: asymptotic_expansions.SingularityAnalysis('n', alpha=-Integer(3)/
   -Integer(2),
...: precision=Integer(3))
3/4/sqrt(pi)*n^(-5/2)
  + 45/32/sqrt(pi)*n^(-7/2)
```

(continues on next page)
+ 1155/512/sqrt(pi)*n^(-9/2)
+ O(n^(-11/2))

```plaintext
>>> asymptotic_expansions.SingularityAnalysis('n', alpha=-Integer(1)/
→Integer(2),
...    precision=Integer(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))
```
Asymptotic Expansions, Release 10.4

- \( \frac{1}{8}/\sqrt{\pi} \cdot n^{-3/2} \cdot \log(n) \) \\
  + \( -(\frac{1}{8} \cdot (\text{euler}\_\text{gamma} + 2 \cdot \log(2))/\sqrt{\pi}) \cdot n^{-3/2} \) \\
  + O(n^{-5/2} \cdot \log(n))

```python
>>> from sage.all import *

>>> ae = asymptotic_expansions.SingularityAnalysis('n',
...     alpha=Integer(1)/Integer(2), beta=Integer(1),
...     precision=Integer(4)); ae
1/\sqrt{\pi} \cdot n^{-1/2} \cdot \log(n) + \left((\text{euler}\_\text{gamma} + 2 \cdot \log(2))/\sqrt{\pi}\right) \cdot n^{-1/2} \\
- 5/8/\sqrt{\pi} \cdot n^{-3/2} \cdot \log(n) + (1/8 \cdot (3\cdot\text{euler}\_\text{gamma} + 6\cdot\log(2) - 8)/\sqrt{\pi} - (\text{euler}\_\text{gamma} + 2\cdot\log(2) - 2)/\sqrt{\pi}) \cdot n^{-3/2} + O(n^{-5/2} \cdot \log(n))

>>> n = ae.parent().gen()

>>> ae.subs(n=n-Integer(1)).map_coefficients(lambda x: x.canonicalize_radical())
1/\sqrt{\pi} \cdot n^{-1/2} \cdot \log(n) \\
+ \left((\text{euler}\_\text{gamma} + 2\cdot\log(2))/\sqrt{\pi}\right) \cdot n^{-1/2} \\
- \frac{1}{8}/\sqrt{\pi} \cdot n^{-3/2} \cdot \log(n) \\
+ \left(-\frac{1}{8} \cdot (\text{euler}\_\text{gamma} + 2\cdot\log(2))/\sqrt{\pi}\right) \cdot n^{-3/2} + O(n^{-5/2} \cdot \log(n))

sage: asymptotic_expansions.SingularityAnalysis('n',
...     alpha=1, beta=1/2, precision=4)
log(n)^{(1/2)} + 1/2\cdot\text{euler}\_\text{gamma}\cdot\log(n)^{(-1/2)} \\
+ (-1/8\cdot\text{euler}\_\text{gamma}^2 + 1/48\cdot\pi^2)\cdot\log(n)^{(-3/2)} \\
+ (1/16\cdot\text{euler}\_\text{gamma}^3 - 1/32\cdot\text{euler}\_\text{gamma}\cdot\pi^2 + 1/8\cdot\zeta(3))\cdot\log(n)^{(-5/2)} \\
+ O(\log(n)^{(-7/2)})

>>> from sage.all import *

>>> ae = asymptotic_expansions.SingularityAnalysis('n',
...     alpha=Integer(0), beta=Integer(2), precision=Integer(14))

>>> n = ae.parent().gen()

>>> ae.subs(n=n-Integer(2))
2\cdot n^{-1}\cdot \log(n) + 2\cdot\text{euler}\_\text{gamma}\cdot n^{(-1)} - n^{(-2)} - 1/6\cdot n^{(-3)} + O(n^{(-5)})

sage: asymptotic_expansions.SingularityAnalysis('n',
...     alpha=-1/2, beta=1, precision=2, normalized=False)
-1/2/\sqrt{\pi} \cdot n^{-3/2} \cdot \log(n) \\
+ (-1/2 \cdot (\text{euler}\_\text{gamma} + 2\cdot\log(2) - 2)/\sqrt{\pi}) \cdot n^{-3/2} \\
+ O(n^{-5/2} \cdot \log(n))

sage: asymptotic_expansions.SingularityAnalysis(
...     'n', 1, alpha=-1/2, beta=1, precision=2, normalized=False)

4.2. Common Asymptotic Expansions

(continues on next page)
....:  'n', 1/2, alpha=0, beta=1, precision=3, normalized=False)
2^n*n^(-1) + O(2^n*n^(-2))

>>> from sage.all import *
>>> asymptotic_expansions.SingularityAnalysis(...  'n', Integer(1), alpha=Integer(-1)/Integer(2), beta=Integer(1),
˓→precision=Integer(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))

ALGORITHM:
See [FS2009].

**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:**

```powershell
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>
```

```powershell
>>> from sage.all import *
>>> asymptotic_expansions.Stirling('n', precision=Integer(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))
```

```powershell
>>> _.parent()
Asymptotic Ring <(e^(n*log(n)))^QQ * (e^n)^QQ * n^QQ * log(n)^QQ>
```

over Symbolic Constants Subring
See also:

\( \log_{\text{Stirling}}() \), \( \text{factorial}() \).

**static log\_Stirling** (\( \text{var, precision=None, skip constant summand=False} \))

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

**INPUT:**

- \( \text{var} \) – a string for the variable name.
- \( \text{precision} \) – (default: None) an integer. If None, then the default precision of the asymptotic ring is used.
- \( \text{skip constant summand} \) – (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))
```

```
>>> from sage.all import *
>>> asymptotic_expansions.log_Stirling('n', precision=Integer(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:

\( \text{Stirling}() \), \( \text{factorial}() \).

\sage\text{.rings.asymptotic.asymptotic\_expansion\_generators.asymptotic\_expansions} = <\sage\text{.rings.asymptotic.asymptotic\_expansion\_generators.}
\text{AsymptoticExpansionGenerators object}>

A collection of several common asymptotic expansions.

This is an instance of \( \text{AsymptoticExpansionGenerators} \) 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 \( \infty \)) 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.
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 \( x^{QQ} \times \log(x)^{ZZ} \times QQ^y \times y^{QQ} \) 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 \( r \in \mathbb{Q}, s \in \mathbb{Z}, a \in \mathbb{Q} \) and \( q \in \mathbb{Q} \).
- 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 \( \ast \) 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 \texttt{GrowthGroup} (see \texttt{GrowthGroupFactory}) is the most convenient way to generate a growth group.

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

Here are some examples:

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

Each of these two generated groups is a \texttt{MonomialGrowthGroup}, whose elements are powers of a fixed symbol (above \texttt{\textquotesingle} z\textquotesingle\textquoteright ). 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 $\infty$ are not yet implemented.

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

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

```
>>> from sage.all import *
>>> z = M.gen()
>>> z**(Integer(3)/Integer(5))
```

Strings can also be parsed:

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

```
>>> from sage.all import *
>>> M('z^7')
```

Similarly, we can construct logarithmic factors by:

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

```
>>> from sage.all import *
>>> GrowthGroup('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
```

```
>>> from sage.all import *
>>> E = GrowthGroup('(QQ_+)^z'); E
```

and a typical element looks like this:

```
sage: E.an_element()
```

```
>>> from sage.all import *
>>> E.an_element()
```

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

```
sage: C = GrowthGroup('(QQ_+)^z * z^QQ * log(z)^QQ'); C
```

```
>>> from sage.all import *
>>> C
```
This contains elements of the form

```sage
(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
(Growth Group QQ^z, Growth Group z^QQ, Growth Group log(z)^QQ)
```

Multivariate constructions are also possible:

```sage
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
g = G_x = GrowthGroup('x^ZZ'); g
Growth Group x^ZZ
g = G_xy = GrowthGroup('x^ZZ * y^ZZ'); g
Growth Group x^ZZ * y^ZZ
g = G.xy.an_element()
x*y
g = x = G_xy('x'); y = G_xy('y')
x = x^2
y = y^2
g = elem = x^21*y^21; elem^2
x^42*y^42
```

(continues on next page)
A monomial growth group itself is totally ordered, all elements are comparable. However, this does not hold for Cartesian products:

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

```python
>>> from sage.all import *

>>> e1 = x**Integer(2)*y; e2 = x*y**Integer(2)
>>> e1 <= e2 or e2 <= e1
False
```

In terms of uniqueness, we have the following behaviour:

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

```python
>>> from sage.all import *

>>> 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:

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

```python
>>> from sage.all import *

>>> 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:
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: 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]
```

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

See also:

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
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>> P = GrowthGroup((ZZ_+)^x)
>>> e1 = P(1); e1
```

```
rank = 13

exception sage.rings.asymptotic.growth_group.DecreasingGrowthElementError(element, *args, **kwds)
```

Bases: ValueError

A special ValueError which is raised when a growth element is less than one.

INPUT:

- element -- a GenericGrowthElement

The remaining arguments are passed on to ValueError.

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

Bases: 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 base \cdot 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
```

(continues on next page)
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

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>>

property 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(Integer(42)**x).base
42

class sage.rings.asymptotic.growth_group.ExponentialGrowthGroup(base, *args, **kwds)
Bases: 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
growth_group.from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup
growth_group: P = ExponentialGrowthGroup(QQ, 'x'); P
Growth Group QQ^x
```

See also:

- GenericGrowthGroup
- DivisionRings
  - alias of DivisionRings
- Element
  - alias of ExponentialGrowthElement
- Groups
  - alias of Groups
- 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
growth_group.from sage.rings.asymptotic.growth_group import GrowthGroup
growth_group: GrowthGroup('(QQ_)^x').construction()
(ExponentialGrowthGroup[x], Rational Field)
```

classmethod factory(base, var, extend_by_non_growth_group=True, return_factors=False, **kwds)
Create an exponential growth group.

This factory takes care of the splitting of the bases into their absolute values and arguments.

INPUT:
• base, var, keywords – use in the initialization of the exponential growth group; see \texttt{ExponentialGrowthGroup} for details.

• extend\_by\_non\_growth\_group – a boolean (default True). If set, then the growth group consists of two parts, one part dealing with the absolute values of the bases and one for their arguments.

• return\_factors – a boolean (default: False). If set, then a tuple of the (cartesian) factors of this growth group is returned.

\textbf{OUTPUT:}

A growth group or tuple of growth groups.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup
sage: ExponentialGrowthGroup.factory(QQ, 'n')
Growth Group QQ^n * Signs^n

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup
>>> ExponentialGrowthGroup.factory(QQ, 'n')
Growth Group QQ^n * Signs^n
\end{verbatim}

\texttt{gens()}

Return a tuple of all generators of this exponential growth group.

\textbf{INPUT:}

Nothing.

\textbf{OUTPUT:}

An empty tuple.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: E = GrowthGroup('(\ZZ_+)^x')
sage: E.gens()
()

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> E = GrowthGroup('(\ZZ_+)^x')
>>> E.gens()
()
\end{verbatim}

\texttt{non\_growth\_group()}

Return a non-growth group (with an argument group, e.g. roots of unity, as base) compatible with this exponential growth group.

\textbf{OUTPUT:}

A group group.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('(\QQ_+)^x').non\_growth\_group()
Growth Group Signs^x
\end{verbatim}
sage: GrowthGroup('(RR_+)^x').non_growth_group()
Growth Group Signs^x
sage: GrowthGroup('(RIF_+)^x').non_growth_group()
Growth Group Signs^x
sage: GrowthGroup('(RBF_+)^x').non_growth_group()
Growth Group Signs^x
sage: GrowthGroup('(CC_+)^x').non_growth_group()
Growth Group UU_RR^x
sage: GrowthGroup('(CIF_+)^x').non_growth_group()
Growth Group UU_RIF^x
sage: GrowthGroup('(CBF_+)^x').non_growth_group()
Growth Group UU_RBF^x

```
>>> from sage.all import *
```  
```
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
```  
```
>>> tuple(GrowthGroup('(QQ_+)^z').some_elements())
((1/2)^z, 2^z, 1, 42^z, (2/3)^z, (3/2)^z, ...)
```

```
class sage.rings.asymptotic.growth_group.ExponentialGrowthGroupFunctor(var)
Bases: 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]
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup,
         ExponentialGrowthGroupFunctor
>>> GrowthGroup('(QQ_+)^z').construction()[Integer(0)]
ExponentialGrowthGroup[z]
```

See also:

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

class sage.rings.asymptotic.growth_group.ExponentialNonGrowthElement (parent,
raw_element)

Bases: GenericNonGrowthElement, ExponentialGrowthElement

An element of ExponentialNonGrowthGroup.

class sage.rings.asymptotic.growth_group.ExponentialNonGrowthGroup (base,*args,
**kws)

Bases: GenericNonGrowthGroup, ExponentialGrowthGroup

A growth group whose base is an argument group.

EXAMPLES:

```
sage: from sage.groups.misc_gps.argument_groups import RootsOfUnityGroup
sage: ExponentialNonGrowthGroup(RootsOfUnityGroup(), 'n')
(-1)^n
```

```
>>> from sage.all import *
>>> from sage.groups.misc_gps.argument_groups import RootsOfUnityGroup
>>> ExponentialNonGrowthGroup(RootsOfUnityGroup(), 'n')
(-1)^n
```

Element

alias of ExponentialNonGrowthElement

classification()

Return the construction of this growth group.

OUTPUT:

A pair whose first entry is an ExponentialNonGrowthGroupFunctor and its second entry the base.
EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('U^x').construction()
(ExponentialNonGrowthGroup[x], Group of Roots of Unity)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> GrowthGroup('U^x').construction()
(ExponentialNonGrowthGroup[x], Group of Roots of Unity)
```

class `sage.rings.asymptotic.growth_group.ExponentialNonGrowthGroupFunctor`(var)
Bases: `ExponentialGrowthGroupFunctor`

A construction functor for `ExponentialNonGrowthGroup`.

class `sage.rings.asymptotic.growth_group.GenericGrowthElement`(parent, raw_element)
Bases: `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:

- `raw_element` - an element from the base of the parent.

EXAMPLES:

```python
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
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import (GenericGrowthGroup,
...:                GenericGrowthElement)

>>> G = GenericGrowthGroup(ZZ)
>>> g = GenericGrowthElement(G, Integer(42)); g
GenericGrowthElement(42)
>>> g.parent()
Growth Group Generic(ZZ)
>>> G(raw_element=Integer(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:

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

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

`is_lt_one()`

Return whether this element is less than 1.

INPUT:

Nothing.

OUTPUT:

A boolean.

EXAMPLES:

```python
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
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup(x^ZZ); x = G(x)
>>> (x**Integer(42)).is_lt_one()  # indirect doctest
False
>>> (x**(-Integer(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:

```python
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
```

(continues on next page)
Asymptotic Expansions, Release 10.4

4.3. (Asymptotic) Growth Groups

(continued from previous page)

\[
\log(x)
\]

\textbf{sage:} \log(x^5) \quad \# \text{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 \( x^\mathbb{ZZ} \times \log(x)^\mathbb{ZZ} \).

>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>> G = GrowthGroup('x^\mathbb{ZZ} \times \log(x)^\mathbb{ZZ}')

>>> x, = G.gens_monomial()

>>> \log(x) \quad \# \text{indirect doctest}

\log(x)

>>> \log(x^\text{Integer}(5)) \quad \# \text{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 \( x^\mathbb{ZZ} \times \log(x)^\mathbb{ZZ} \).

sage: G = GrowthGroup((\QQ_+)^x \times x^\mathbb{ZZ})

sage: x, = G.gens_monomial()

sage: el = x.rpow(Integer(2)); el
2^x

sage: \log(el) \quad \# \text{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 \( \QQ^x \times x^\mathbb{ZZ} \).

sage: \log(el, base=Integer(2)) \quad \# \text{indirect doctest}

x

>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>> G = GrowthGroup((\QQ_+)^x \times x^\mathbb{ZZ})

>>> x, = G.gens_monomial()

>>> el = x.rpow(2); el
2^x

>>> \log(el) \quad \# \text{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 \( \QQ^x \times x^\mathbb{ZZ} \).

>>> \log(el, base=Integer(2)) \quad \# \text{indirect doctest}

x

sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup

sage: x = GenericGrowthGroup(ZZ).an_element()

sage: \log(x) \quad \# \text{indirect doctest}

Traceback (most recent call last):
...
NotImplementedError: Cannot determine logarithmized factorization of \GenericGrowthElement(1)\ in abstract base class.

>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GenericGrowthGroup

>>> x = GenericGrowthGroup(ZZ).an_element()

(continues on next page)
log_factor\( (base=None, locals=None) \)

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.
- `locals` – a dictionary which may contain the following keys and values:
  - `'log'` – value: a function. If not used, then the usual `log` 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:**

```python
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)))
```
Asymptotic Expansions, Release 10.4

(continued from previous page)

```
>>> (x**Integer(123)).log_factor()  # indirect doctest
((log(x), 123),)
```

```
>>> (G(2^x) * x**Integer(2)).log_factor(base=Integer(2))  # indirect doctest
((x, 1), (log(x), 2/log(2)))
```

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

```
>>> from sage.all import *
```

```
>>> G(Integer(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.
```

```
>>> from sage.all import *
```

```
>>> 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().
```

\textbf{rpow} (base)

Calculate the power of \texttt{base} to this element.

\textbf{INPUT:}

- \texttt{base} – an element.

\textbf{OUTPUT:}

A growth element.

\textbf{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
```

(continues on next page)
>>> x.rpow(Integer(1)/Integer(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)'

>>> from sage.all import *

>>> x.rpow(Integer(0))  # indirect doctest
Traceback (most recent call last):
...
ValueError: 0 is not an allowed base for calculating the power to x.

>>> (x^Integer(2)).rpow(Integer(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))

>>> from sage.all import *

>>> G = GrowthGroup('QQ^(x*log(x)) * x^ZZ * log(x)^ZZ')

>>> x = G('x')

>>> (x * log(x)).rpow(Integer(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

>>> from sage.all import *

>>> n = GrowthGroup('(QQ_+)^n * n^QQ')('n')

>>> n.rpow(Integer(2))
2^n

>>> _.parent()
Growth Group QQ^n * n^QQ

sage: n = GrowthGroup('QQ^n * n^QQ')('n')
sage: n.rpow(-2)
2^n*(-1)^n
variable_names()

Return the names of the variables of this growth element.

OUTPUT:

A tuple of strings.

EXAMPLES:

```python
>>> from sage.all import *
>>> n = GrowthGroup('QQ^n * n^QQ')(n)
>>> n.rpow(-Integer(2))
2^n*(-1)^n
```

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

```python
sage: G(m^0).variable_names()
()
```

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

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup('QQ^m')
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

```python
sage: G = GrowthGroup('QQ^m')
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup(QQ^m)
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup('QQ^m')
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup(QQ^m)
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup('QQ^m')
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

```python
sage: G = GrowthGroup(QQ^m)
```

```python
sage: G(2^m).variable_names()
('m',)
```

```python
sage: G(1^m).variable_names()
()
```

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

Bases: UniqueRepresentation, Parent, WithLocals

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

CartesianProduct =
<sage.rings.asymptotic.growth_group_cartesian.CartesianProductFactory
object>

Element
    alias of GenericGrowthElement

Magmas
    alias of Magmas

Posets
    alias of Posets

Sets
    alias of Sets

extended_by_non_growth_group()
    Extend to a cartesian product of this growth group and a suitable non growth group.

OUTPUT:
    A group group.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('(QQ_+)^x').extended_by_non_growth_group()
Growth Group QQ^x * Signs^x
sage: GrowthGroup('(RR_+)^x').extended_by_non_growth_group()
Growth Group RR^x * Signs^x
sage: GrowthGroup('(RIF_+)^x').extended_by_non_growth_group()
Growth Group RIF^x * Signs^x
sage: GrowthGroup('(RBF_+)^x').extended_by_non_growth_group()
Growth Group RBF^x * Signs^x
sage: GrowthGroup('(CC_+)^x').extended_by_non_growth_group()
Growth Group CC^x * UU_RR^x
sage: GrowthGroup('(CIF_+)^x').extended_by_non_growth_group()
```

(continues on next page)
Growth Group CIF^x * UU_RIF^x

sage: GrowthGroup('CBF_+')^x).extended_by_non_growth_group()
Growth Group CBF^x * UU_RBF^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((QQ_+)^x).extended_by_non_growth_group()
Growth Group QQ^x * Signs^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((RR_+)^x).extended_by_non_growth_group()
Growth Group RR^x * Signs^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((RIF_+)^x).extended_by_non_growth_group()
Growth Group RIF^x * Signs^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((RBF_+)^x).extended_by_non_growth_group()
Growth Group RBF^x * Signs^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((CC_+)^x).extended_by_non_growth_group()
Growth Group CC^x * UU_RR^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((CIF_+)^x).extended_by_non_growth_group()
Growth Group CIF^x * UU_RIF^x

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>>
GrowthGroup((CBF_+)^x).extended_by_non_growth_group()
Growth Group CBF^x * UU_RBF^x

\textbf{gen} (n=0)

Return the $n$-th generator (as a group) of this growth group.

**INPUT:**

- \( n \) – default: 0.

**OUTPUT:**

A \texttt{MonomialGrowthElement}.

**EXAMPLES:**

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

```
x
```

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> P = GrowthGroup('x^ZZ')
>>> P.gen()

```
x
```

sage: P = GrowthGroup('(QQ_+)^x')

```
Traceback (most recent call last):
...
IndexError: tuple index out of range
```

>>> from sage.all import *
>>> P = GrowthGroup('(QQ_+)^x')
>>> P.gen()

```
Traceback (most recent call last):
...
IndexError: tuple index out of range
```

4.3. (Asymptotic) Growth Groups 87
**gens()**

Return a tuple of all generators of this growth group.

**INPUT:**
Nothing.

**OUTPUT:**
A tuple whose entries are growth elements.

**EXAMPLES:**

```python
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, \(x^\text{ZZ}\) has \(x\) as a monomial generator, while \(\log(x)^\text{ZZ}\) or \(\text{icecream}(x)^\text{ZZ}\) do not have monomial generators.

**is_compatible(other)**

Return whether this growth group is compatible with `other` meaning that both are of the same type and have the same variables, but maybe a different base.

**INPUT:**
- `other` – a growth group

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup,
    ExponentialNonGrowthGroup
sage: from sage.groups.misc_gps.argument_groups import RootsOfUnityGroup
sage: EQ = ExponentialGrowthGroup(QQ, 'n')
(sage: EZ = ExponentialGrowthGroup(ZZ, 'n')

sage: UU = ExponentialNonGrowthGroup(RootsOfUnityGroup(), 'n')
```

(continues on next page)
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
growth = GrowthGroup('x^ZZ')
sage: x = growth.gen()
sage: x * 2
```

(continues on next page)
True

\texttt{sage: G.le(x^2, x)}
False
\texttt{sage: G.le(x^0, 1)}
True

\begin{verbatim}
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup('x^ZZ')
>>> x = G.gen()
>>> G.le(x, x**Integer(2))
True
>>> G.le(x**Integer(2), x)
False
>>> G.le(x**Integer(0), Integer(1))
True
\end{verbatim}

\textbf{ngens ()}

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

\textbf{INPUT:}

Nothing.

\textbf{OUTPUT:}

A Python integer.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage: from sage.rings.asymptotic.growth_group import GrowthGroup}
\texttt{sage: P = GrowthGroup('x^ZZ')}
\texttt{sage: P.ngens()}
1
\texttt{sage: GrowthGroup('log(x)^ZZ').ngens()}
1
\end{verbatim}

\begin{verbatim}
\texttt{>>> from sage.all import *}
\texttt{>>> from sage.rings.asymptotic.growth_group import GrowthGroup}
\texttt{>>> P = GrowthGroup('x^ZZ')}
\texttt{>>> P.ngens()}
1
\texttt{>>> GrowthGroup('log(x)^ZZ').ngens()}
1
\end{verbatim}

\begin{verbatim}
\texttt{sage: P = GrowthGroup('(QQ_+)^x')}
\texttt{sage: P.ngens()}
0
\end{verbatim}

\begin{verbatim}
\texttt{>>> from sage.all import *}
\texttt{>>> P = GrowthGroup('(QQ_+)^x')}
\texttt{>>> P.ngens()}
0
\end{verbatim}

\textbf{non_growth_group ()}

Return a non-growth group compatible with this growth group.
OUTPUT:
A group group.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: GenericGrowthGroup(ZZ, 'n').non_growth_group()
Traceback (most recent call last):
...:
NotImplementedError: only implemented in concrete realizations
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GenericGrowthGroup
>>> GenericGrowthGroup(ZZ, 'n').non_growth_group()
Traceback (most recent call last):
...:
NotImplementedError: only implemented in concrete realizations
```

**some_elements()**

Return some elements of this growth group.

See `TestSuite` for a typical use case.

**INPUT:**
Nothing.

**OUTPUT:**
An iterator.

**EXAMPLES:**

```python
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), ...)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> 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), ...)
```

```python
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), ...)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> 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), ...)
```

**variable_names()**

Return the names of the variables of this growth group.

**OUTPUT:**
A tuple of strings.
EXAMPLES:

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

```python
>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GenericGrowthGroup

>>> GenericGrowthGroup(ZZ).variable_names()
()
```

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

```python
sage: GrowthGroup('log(x)^ZZ').variable_names()
('x',)
```

```python
>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>> GrowthGroup('x^ZZ').variable_names()
('x',)
```

```python
>>> GrowthGroup('log(x)^ZZ').variable_names()
('x',)
```

```python
sage: GrowthGroup((QQ_+)^x).variable_names()
('x',)
```

```python
sage: GrowthGroup((QQ_+)^(x*log(x))).variable_names()
('x',)
```

```python
>>> from sage.all import *

>>> GrowthGroup((QQ_+)^x).variable_names()
('x',)
```

```python
>>> GrowthGroup((QQ_+)^(x*log(x))).variable_names()
('x',)
```

```python
class sage.rings.asymptotic.growth_group.GenericNonGrowthElement (parent, raw_element)

Bases: GenericGrowthElement

An element of GenericNonGrowthGroup.

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

Bases: GenericGrowthGroup

A (abstract) growth group whose elements are all of the same growth 1.

See ExponentialNonGrowthGroup for a concrete realization.
```

```python
sage.rings.asymptotic.growth_group.GrowthGroup = 
<sage.rings.asymptotic.growth_group.GrowthGroupFactory object>

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

class sage.rings.asymptotic.growth_group.GrowthGroupFactor (cls, base, var, extend_by_non_growth_group)

Bases: tuple
```
Asymptotic Expansions, Release 10.4

base
  Alias for field number 1

cls
  Alias for field number 0

extend_by_non_growth_group
  Alias for field number 3

var
  Alias for field number 2

class sage.rings.asymptotic.growth_group.GrowthGroupFactory
  Bases: 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)

(continues on next page)
Growth Group \(e^x)^{\mathbb{Z}} \times x^{\mathbb{Z}}\)

```
sage: GrowthGroup(('e^x)^{\mathbb{Z}} \times x^{\mathbb{Z}')}
Growt Group \(e^x)^{\mathbb{Z}} \times x^{\mathbb{Z}}\)
```

```python
>>> from sage.all import *

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

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

Create an object from the given arguments.

```python
class sage.rings.asymptotic.growth_group.MonomialGrowthElement(parent, raw_element)
```

Bases: `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 `variable^{exponent}`. The multiplication corresponds to the addition of the exponents.

**EXAMPLES:**

```python
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
```

```bash
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import MonomialGrowthGroup

>>> P = MonomialGrowthGroup(ZZ, 'x')

>>> e1 = P(Integer(1)); e1
1

>>> e2 = P(raw_element=Integer(2)); e2
x^2

>>> e1 == e2
False

>>> P.le(e1, e2)
True

>>> P.le(e1, P.gen()) and P.le(P.gen(), e2)
True
```

**property exponent**

The exponent of this growth element.

**EXAMPLES:**

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

sage: P(x^42).exponent
42
```

4.3. (Asymptotic) Growth Groups
```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> P = GrowthGroup("x^ZZ")
>>> P(x**Integer(42)).exponent
42
```

```python
class sage.rings.asymptotic.growth_group.MonomialGrowthGroup(base, var, category)
Bases: 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:

```python
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

```
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
growth_group = GrowthGroup('x^ZZ').construction()
(MonomialGrowthGroup[x], Integer Ring)
```

classmethod factory(base, var, extend_by_non_growth_group=False, return_factors=False, **kwds)

Create a monomial growth group.

INPUT:

- base, var, keywords – use in the initialization of the exponential growth group; see MonomialGrowthGroup for details.
- extend_by_non_growth_group – a boolean (default False). If set, then the growth group consists of two parts, one part dealing with the absolute values of the bases and one for their arguments.
- return_factors – a boolean (default: False). If set, then a tuple of the (cartesian) factors of this growth group is returned.

OUTPUT:

A growth group or tuple of growth groups.

EXAMPLES:

```sage
from sage.rings.asymptotic.growth_group import MonomialGrowthGroup
from sage.groups.misc_gps.imaginary_groups import ImaginaryGroup
growth_group = MonomialGrowthGroup.factory(ZZ, 'n')
Growth Group n^ZZ

growth_group = MonomialGrowthGroup.factory(ImaginaryGroup(ZZ), 'n')
Growth Group n^(ZZ*I)
```

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, \(x^{\mathbb{Z}}\) has no logarithmic generator, while \(\log(x)^{\mathbb{Z}}\) has \(\log(x)\) as logarithmic generator.

**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, \(x^{\mathbb{Z}}\) has \(x\) as a monomial generator, while \(\log(x)^{\mathbb{Z}}\) or \(\text{icecream}(x)^{\mathbb{Z}}\) do not have monomial generators.

**non_growth_group()**
Return a non-growth group (with an imaginary group as base) compatible with this monomial growth group.

OUTPUT:
A group group.

EXAMPLES:

```sage
definitely
```

```sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('n^{\mathbb{Z}}').non_growth_group()
Growth Group n^{(\mathbb{Z}^*)}
```

```sage: from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> GrowthGroup('n^{\mathbb{Z}}').non_growth_group()
Growth Group n^{(\mathbb{Z}^*)}
```

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

**Bases:** `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^{\mathbb{Q}}').construction()[0]
    MonomialGrowthGroup[z]
```
See also:


```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup,
˓→MonomialGrowthGroupFunctor
>>> GrowthGroup('z^QQ').construction()[Integer(0)]
MonomialGrowthGroup[z]
```

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

A growth group whose base is an imaginary group.

EXAMPLES:

```python
sage: from sage.groups.misc_gps.imaginary_groups import ImaginaryGroup
sage: J = MonomialNonGrowthGroup(ImaginaryGroup(ZZ), 'n')
sage: J.an_element()
n^I
```

Element

alias of MonomialNonGrowthElement

collection()

Return the construction of this growth group.

OUTPUT:

A pair whose first entry is a MonomialNonGrowthGroupFunctor and its second entry the base.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: GrowthGroup('x^(QQ*I)').construction()
(MonomialNonGrowthGroup[x], Imaginary Group over Rational Field)
```

4.3. (Asymptotic) Growth Groups
class sage.rings.asymptotic.growth_group.MonomialNonGrowthGroupFunctor(var)

   Bases: MonomialGrowthGroupFunctor

   A construction functor for MonomialNonGrowthGroup.

class sage.rings.asymptotic.growth_group.PartialConversionElement(growth_group, raw_element)

   Bases: SageObject

   A not converted element of a growth group.

   INPUT:
   
   * growth_group -- a group
   * raw_element -- an object

   A PartialConversionElement is an element growth_group(raw_element) which usually appears in conjunction with PartialConversionValueError. In this case, it was not possible to create that element, although the conversion went partially well in the sense that a raw_element (e.g. an exponent for MonomialGrowthElement or a base for ExponentialGrowthElement) could be extracted.

   Its main purpose is to carry data used during the creation of elements of cartesian products of growth groups.

   is_compatible(other)

   Wrapper to GenericGrowthGroup.is_compatible().

   split()  

   Split the contained raw_element according to the growth group's GrowthGroup.

   _split_raw_element_()

   EXAMPLES:

   sage: from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup, PartialConversionValueError
   sage: E = ExponentialGrowthGroup(ZZ, 'x')
   sage: try:
   ....:     E((-2)^x)
   ....: except PartialConversionValueError as e:
   ....:     e.element.split()
   (2^x, element with parameter -1 (<class 'int'>) in Growth Group ZZ^x)

   >>> from sage.all import *
   >>> from sage.rings.asymptotic.growth_group import ExponentialGrowthGroup, PartialConversionValueError
   >>> E = ExponentialGrowthGroup(ZZ, 'x')
   >>> try:
   ...     E((-Integer(2))**x)
   ... except PartialConversionValueError as e:
   ...     e.element.split()
   (2^x, element with parameter -1 (<class 'int'>) in Growth Group ZZ^x)

   exception sage.rings.asymptotic.growth_group.PartialConversionValueError(element, *args, **kws)

   Bases: ValueError

   A special ValueError which is raised when (partial) conversion fails.
INPUT:

• element – a PartialConversionElement

The remaining argument passed on to ValueError.

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

Bases: CachedRepresentation, 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.

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',)

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import Variable
>>> Variable.extract_variable_names('')
()
>>> Variable.extract_variable_names('x')
('x',)
>>> Variable.extract_variable_names('exp(x)')
('x',)
>>> 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',)

>>> from sage.all import *

>>> from sage.all import *  
>>> Variable.extract_variable_names('log(77w)')  
('w',)  
>>> Variable.extract_variable_names('log(x)')  
Traceback (most recent call last):
...
TypeError: Bad function call: log(x) !!!!
>>> Variable.extract_variable_names('x')
Traceback (most recent call last):
...
TypeError: Malformed expression: x) !!!!
>>> Variable.extract_variable_names('log(x)')
Traceback (most recent call last):
...
TypeError: Malformed expression: log) !!!! x(  
>>> Variable.extract_variable_names('log(x)+y')  
('x', 'y')  
>>> Variable.extract_variable_names('icecream(summer)')

>>> 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')

(continues on next page)
from sage.all import *

Variable.extract_variable_names(a + b)
(a, b)

Variable.extract_variable_names(a+b)
(a, b)

Variable.extract_variable_names(a + b)
(a, b)

Variable.extract_variable_names('+a')
(a,

Variable.extract_variable_names(a+)
Traceback (most recent call last):
...
TypeError: Malformed expression: a+ !!!

Variable.extract_variable_names('b!')
(b,

Variable.extract_variable_names('-a')
(a,

Variable.extract_variable_names('a*b')
(a, b)

Variable.extract_variable_names('2^q')
(q,

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

sage: from sage.all import *

from sage.rings.asymptotic.growth_group import Variable

Variable('x').is_monomial()
True

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',)
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import Variable
>>> Variable('x').variable_names()
('x',)
>>> 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.

4.4.1 Classes and Methods

```
class sage.rings.asymptotic.growth_group_cartesian.CartesianProductFactory
Bases: 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
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> A = GrowthGroup('x^ZZ'); A
Growth Group x^ZZ
>>> B = GrowthGroup('log(x)^ZZ'); B
Growth Group log(x)^ZZ
>>> C = cartesian_product([A, B]); C  # indirect doctest
Growth Group x^ZZ * log(x)^ZZ
>>> C._le_ == C.le_lex
True
>>> D = GrowthGroup('y^ZZ'); D
Growth Group y^ZZ
>>> E = cartesian_product([A, D]); E  # indirect doctest
Growth Group x^ZZ * y^ZZ
>>> E._le_ == E.le_product
True
>>> F = cartesian_product([C, D]); F  # indirect doctest
Growth Group x^ZZ * log(x)^ZZ * y^ZZ
>>> F._le_ == F.le_product
```

(continues on next page)
True
>>> cartesian_product([A, E]); G # indirect doctest
Traceback (most recent call last):
...
ValueError: The growth groups (Growth Group \(x^\mathbb{Z}\), Growth Group \(x^\mathbb{Z} \ast y^\mathbb{Z}\)) need to have pairwise disjoint or equal variables.
>>> cartesian_product([A, B, D]) # indirect doctest
Growth Group \(x^\mathbb{Z} \ast \log(x)^\mathbb{Z} \ast y^\mathbb{Z}\)

create_key_and_extra_args (growth_groups, category, **kwds)

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

create_object (version, args, **kwds)

Create an object from the given arguments.

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

Bases: CartesianProductPoset, GenericGrowthGroup

A Cartesian product of growth groups.

EXAMPLES:

```python
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> P = GrowthGroup('x'^QQ)
>>> L = GrowthGroup('log(x)^ZZ')
>>> C = cartesian_product([P, L], order='lex'); C # indirect doctest
Growth Group \(x^\mathbb{Q} \ast \log(x)^\mathbb{Z}\)
>>> C.an_element()
\(x^{(1/2)} \ast \log(x)\)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> P = GrowthGroup('x'^QQ)
>>> L = GrowthGroup('log(x)^ZZ')
>>> C = cartesian_product([P, L], order='lex'); C # indirect doctest
Growth Group \(x^\mathbb{Q} \ast \log(x)^\mathbb{Z}\)
>>> C.an_element()
\(x^{(1/2)} \ast \log(x)\)
```

```python
>>> Px = GrowthGroup('x'^QQ)
>>> Lx = GrowthGroup('log(x)^ZZ')
>>> Cx = cartesian_product([Px, Lx], order='lex') # indirect doctest
>>> Py = GrowthGroup('y'^QQ)
>>> C = cartesian_product([Cx, Py], order='product'); C # indirect doctest
Growth Group \(x^\mathbb{Q} \ast \log(x)^\mathbb{Z} \ast y^\mathbb{Q}\)
>>> C.an_element()
\(x^{(1/2)} \ast \log(x) \ast y^{(1/2)}\)
```

```python
>>> from sage.all import *
>>> Px = GrowthGroup('x'^QQ)
>>> Lx = GrowthGroup('log(x)^ZZ')
>>> Cx = cartesian_product([Px, Lx], order='lex') # indirect doctest
>>> Py = GrowthGroup('y'^QQ)
>>> C = cartesian_product([Cx, Py], order='product'); C # indirect doctest
Growth Group \(x^\mathbb{Q} \ast \log(x)^\mathbb{Z} \ast y^\mathbb{Q}\)
```
See also:
CartesianProduct, CartesianProductPoset.

CartesianProduct =
<sage.rings.asymptotic.growth_group_cartesian.CartesianProductFactory
object>

class Element

Bases: Element

exp()
The exponential of this element.

INPUT:
Nothing.

OUTPUT:
A growth element.

EXAMPLES:

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

def exp(log(log(x)))
```

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

```sage
def 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'
```

```sage
def from sage.all import *
def 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'

**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:**

```python
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 * 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)
```

---

108 Chapter 4. Asymptotic Expansions — Table of Contents
>>> from sage.all import *
>>> G = GrowthGroup('x^ZZ * log(x)^ZZ * log(log(x))^ZZ * y^QQ')
>>> x, y = G.gens_monomial()
>>> f = (x * log(x) * y).factors(); f
(x, log(x), y)
>>> tuple(factor.parent() for factor in f)
(Growth Group x^ZZ, Growth Group log(x)^ZZ, Growth Group y^QQ)

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

>>> from sage.all import *
>>> 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

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 * log(x)^ZZ)
sage: x, = G.gens_monomial()
sage: log(x)  # indirect doctest

log
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

4.4. Cartesian Products of Growth Groups 109
Asymptotic Expansions, Release 10.4

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.

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup('x^ZZ * log(x)^ZZ')
>>> x, = G.gens_monomial()
>>> log(x)  # indirect doctest
log(x)
>>> log(x**Integer(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=Integer(2))  # indirect doctest
x

>>> from sage.all import *
>>> G = GrowthGroup('(QQ_+)^x * x^ZZ')
>>> x, = G.gens_monomial()
>>> el = x.rpow(Integer(2)); el
2^x
>>> 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.
>>> log(el, base=Integer(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.

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GenericGrowthGroup
>>> x = GenericGrowthGroup(ZZ).an_element()
log_factor \( \text{(base=\text{None}, \text{locals=\text{None}})} \)

Return the logarithm of the factorization of this element.

**INPUT:**

- base – the base of the logarithm. If \text{None} (default value) is used, the natural logarithm is taken.
- locals – a dictionary which may contain the following keys and values:
  - 'log' – value: a function. If not used, then the usual \text{log} 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:**

```python
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=Integer(2)) # indirect doctest
((x, 1), (\log(x), 2/\log(2)))
```
\texttt{\texttt{doctest}}
\begin{verbatim}
    ((x, 1), (log(x), 2/log(2)))
\end{verbatim}

\begin{verbatim}
sage: G(1).log_factor()
()
\end{verbatim}

\begin{verbatim}
\texttt{>> from sage.all import *}
\texttt{>>> G(Integer(1)).log_factor()}
()
\end{verbatim}

\begin{verbatim}
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 \(\mathbb{Q}^x x^{\mathbb{Z}} \log(x)^{\mathbb{Z}} y^{\mathbb{Z}} \log(y)^{\mathbb{Z}}\).
\end{verbatim}

\begin{verbatim}
\texttt{>> from sage.all import *}
\texttt{>>> 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 \(\mathbb{Q}^x x^{\mathbb{Z}} \log(x)^{\mathbb{Z}} y^{\mathbb{Z}} \log(y)^{\mathbb{Z}}\).
\end{verbatim}

See also:
\texttt{factors()}, \texttt{log()}.

\texttt{rpow(base)}
Calculate the power of \texttt{base} to this element.

\begin{verbatim}
 INPUT:
    • \texttt{base} – an element.
 OUTPUT:
    A growth element.
 EXAMPLES:
\end{verbatim}

\begin{verbatim}
sage: \texttt{from sage.rings.asymptotic.growth\_group import GrowthGroup}
sage: G = GrowthGroup('((\mathbb{Q}^+)^x x^{\mathbb{Z}})')
sage: x = G('x')
sage: x.rpow(2)  \# indirect doctest
2^x
sage: x.rpow(Integer(1)/Integer(2))  \# indirect doctest
(1/2)^x
\end{verbatim}

\begin{verbatim}
\texttt{>> from sage.all import *}
\texttt{>> from sage.rings.asymptotic.growth\_group import GrowthGroup}
\texttt{>>> G = GrowthGroup('((\mathbb{Q}^+)^x x^{\mathbb{Z}})')}
\texttt{>>> x = G('x')}
\texttt{>>> x.rpow(Integer(2))  \# indirect doctest}
2^x
\texttt{>>> x.rpow(Integer(1)/Integer(2))  \# indirect doctest}
(1/2)^x
\end{verbatim}
```python
sage: x.rpow(0)  # indirect doctest
ValueError: 0 is not an allowed base for calculating the power to x.
```
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')

>>> G(2^m * m^4 * log(n)).variable_names()
(m, n)
>>> G(2^m * m^4).variable_names()
(m,)
>>> G(log(n)).variable_names()
(n,)
>>> G(m^3).variable_names()
(m,)
>>> G(m^0).variable_names()
()
```

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.

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 \texttt{gens_monomial()} method on the individual factors of this Cartesian product and concatenates the respective outputs.

**EXAMPLES:**

```python
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)
```

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

\texttt{some_elements()}

Return some elements of this Cartesian product of growth groups.

See \texttt{TestSuite} for a typical use case.

**OUTPUT:**

An iterator.

**EXAMPLES:**

```python
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(), 10r))
(x^(1/2)*(1/2)^y,
x^(-1/2)*log(x)*2^y,
x^2*log(x)^(-1),
x^(-2)*log(x)^2*42^y,
log(x)^(-2)*(2/3)^y,
x*log(x)^3*(3/2)^y,
x^(-1)*log(x)^(-3)*(4/5)^y,
x^42*log(x)^4*(5/4)^y,
x^(2/3)*log(x)^(-4)*(6/7)^y,
x^(-2/3)*log(x)^5*(7/6)^y)
```

```python
>>> from sage.all import *
>>> from itertools import islice
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup((QQ_+)^y * x^QQ * log(x)^ZZ)
>>> tuple(islice(G.some_elements(), 10))
(x^(1/2)*(1/2)^y,
x^(-1/2)*log(x)*2^y,
x^2*log(x)^(-1),
x^(-2)*log(x)^2*42^y,
log(x)^(-2)*(2/3)^y,
x*log(x)^3*(3/2)^y,
x^(-1)*log(x)^(-3)*(4/5)^y,
x^42*log(x)^4*(5/4)^y,
x^(2/3)*log(x)^(-4)*(6/7)^y,
x^(-2/3)*log(x)^5*(7/6)^y)
```

4.4. Cartesian Products of Growth Groups 115
variable_names()

Return the names of the variables.

OUTPUT:
A tuple of strings.

EXAMPLES:

```python
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')
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> 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: 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. \(x^ZZ \times y^ZZ\)).

See also:

UnivariateProduct, GenericProduct.

CartesianProduct =
<sage.rings.asymptotic.growth_group_cartesian.CartesianProductFactory object>

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

Bases: 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. \(x^ZZ \times \log(x)^ZZ\): polynomial growth dominates logarithmic growth).

See also:

MultivariateProduct, GenericProduct.

CartesianProduct =
<sage.rings.asymptotic.growth_group_cartesian.CartesianProductFactory object>
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 $3x^5$ is able to absorb $-2x^5$ (with result $x^5$). Essentially, absorption can be interpreted as the addition of “compatible” terms (partial addition).

### 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:

```python
sage: from sage.rings.asymptotic.term_monoid import OTermMonoid, ExactTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
  TermMonoid
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: OT = OTermMonoid(TermMonoid, growth_group=G, coefficient_ring=QQ)
sage: ET = ExactTermMonoid(TermMonoid, growth_group=G, coefficient_ring=QQ)
sage: ot1 = OT(x); ot2 = OT(x**Integer(2))
sage: et1 = ET(x**2, coefficient=Integer(2))
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.term_monoid import OTermMonoid, ExactTermMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
  TermMonoid
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup('x^ZZ'); x = G.gen()
>>> OT = OTermMonoid(TermMonoid, growth_group=G, coefficient_ring=QQ)
>>> ET = ExactTermMonoid(TermMonoid, growth_group=G, coefficient_ring=QQ)
>>> ot1 = OT(x); ot2 = OT(x**Integer(2))
>>> et1 = ET(x**2, coefficient=Integer(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:

```python
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) $OTerm$:

```python
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:

```python
>>> from sage.all import *

>>> et1 = 2*x^2
>>> et1.can_absorb(ET(x^2, coefficient=5))
True
>>> any(et1.can_absorb(t) for t in [ot1, ot2])
False
```

(continues on next page)
As mentioned above, absorption directly corresponds to addition in this case:

```python
sage: et1.absorb(ET(x^2, coefficient=5))
7*x^2
```

When adding two exact terms, they might cancel out. For technical reasons, `None` is returned in this case:

```python
sage: ET(x^2, coefficient=5).can_absorb(ET(x^2, coefficient=-5))
True
sage: ET(x^2, coefficient=5).absorb(ET(x^2, coefficient=-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:

```python
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:

```python
sage: ot2.absorb(ot1)
O(x^2)
```

4.5. (Asymptotic) Term Monoids
4.5.2 Comparison

The comparison of asymptotic terms with \( \leq \) is implemented as follows:

- When comparing \( t_1 \leq t_2 \), the coercion framework first tries to find a common parent for both terms. If this fails, \texttt{False} is returned.
- In case the coerced terms do not have a coefficient in their common parent (e.g. \texttt{OTerm}), the growth of the two terms is compared.
- Otherwise, if the coerced terms have a coefficient (e.g. \texttt{ExactTerm}), we compare whether \( t_1 \) has a growth that is strictly weaker than the growth of \( t_2 \). If so, we return \texttt{True}. If the terms have equal growth, then we return \texttt{True} if and only if the coefficients coincide as well.

In all other cases, we return \texttt{False}.

Long story short: we consider terms with different coefficients that have equal growth to be incomparable.

4.5.3 Various

**Warning:** The code for \texttt{B-Terms} is experimental, so a warning is thrown when a \texttt{BTerm} is created for the first time in a session (see \texttt{sage.misc.superseded.experimental}).

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
growth_group='x^ZZ')
doctest:warning
... FutureWarning: This class/method/function is marked as experimental. It, its functionality or its interface might change without a formal deprecation. See https://github.com/sagemath/sage/issues/31922 for details.
```

Todo:

- Implementation of more term types (e.g. \( \Omega \) terms, \( o \) terms, \( \Theta \) terms).

AUTHORS:

- Benjamin Hackl (2015)
- Daniel Krenn (2015)
- Clemens Heuberger (2016)
4.5.4 Classes and Methods

class sage.rings.asymptotic.term_monoid.BTerm(parent, growth, valid_from, **kwds)

Bases: TermWithCoefficient

Class for asymptotic B-terms.

A B-term represents all functions which (in absolute value) are bounded by the given growth and coefficient for the parameters given by valid_from. For example, we have terms that represent functions

- bounded by $5|x|^2$ for $|x| \geq 3$,
- bounded by $42|x|^3$ for $|x| \geq 15$ and $|y| \geq 15$, or
- bounded by $42|x|^3|y|^2$ for $|x| \geq 10$ and $|y| \geq 20$ (see below for the actual examples).

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
- valid_from – dictionary mapping variable names to lower bounds for the corresponding variable. The bound implied by this term is valid when all variables are at least their corresponding lower bound. If a number is passed to valid_from, then the lower bounds for all variables of the asymptotic expansion are set to this number

EXAMPLES:

We revisit the examples from the introduction:

```
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
           TermMonoid
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: G = GrowthGroup('x^ZZ * y^ZZ')
sage: T = TermMonoid('B', growth_group=G, coefficient_ring=ZZ)
sage: x, y = G('x'), G('y')
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
       TermMonoid
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup('x^ZZ * y^ZZ')
>>> T = TermMonoid('B', growth_group=G, coefficient_ring=ZZ)
>>> x, y = G('x'), G('y')
```

This is a term bounded by $5|x|^2$ for $|x| \geq 3$:
Asymptotic Expansions, Release 10.4

\begin{verbatim}
sage: T(x^2, coefficient=5, valid_from={x: 3})
B(5*x^2, x >= 3)

>>> from sage.all import *
>>> T(x**Integer(2), coefficient=Integer(5), valid_from={x: Integer(3)})
B(5*x^2, x >= 3)

This is a term bounded by 42|x|^3 for |x| \geq 15 and |y| \geq 15:

sage: T(x^3, coefficient=42, valid_from={x: 15, y: 15})
B(42*x^3, x >= 15, y >= 15)

>>> from sage.all import *
>>> T(x**Integer(3), coefficient=Integer(42), valid_from={x: Integer(15), y: Integer(15)})
B(42*x^3, x >= 15, y >= 15)

This is a term bounded by 42|x|^3|y|^2 for |x| \geq 10 and |y| \geq 20:

sage: T(x^3*y^2, coefficient=42, valid_from={x: 10, y: 20})
B(42*x^3*y^2, x >= 10, y >= 20)

>>> from sage.all import *
>>> T(x**Integer(3)*y**Integer(2), coefficient=Integer(42), valid_from={x: Integer(10), y: Integer(20)})
B(42*x^3*y^2, x >= 10, y >= 20)

can_absorb\(\text{other}\)

Check whether this B-term can absorb other.

\textbf{INPUT:}

\begin{itemize}
  \item other – an asymptotic term
\end{itemize}

\textbf{OUTPUT:}

A boolean

\textbf{Note:} A B\text{Term} can absorb another B\text{Term} with weaker or equal growth.

See the module description for a detailed explanation of absorption.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
sage: BT = TermMonoid(B, GrowthGroup(x^ZZ), QQ)
sage: t1 = BT(x^3, coefficient=3, valid_from={x: 20})
sage: t2 = BT(x^2, coefficient=1, valid_from={x: 10})
sage: t3 = BT(x^3, coefficient=10, valid_from={x: 10})
sage: t1.can_absorb(t2)
True
sage: t2.can_absorb(t1)
False
sage: t1.can_absorb(t3)
(continues on next page)
\end{verbatim}
\end{verbatim}
continued from previous page

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as _TermMonoid
>>> BT = _TermMonoid('B', GrowthGroup('x^ZZ'), QQ)
>>> t1 = BT(x**Integer(3), coefficient=Integer(3), valid_from={x:... ->Integer(20)})
>>> t2 = BT(x**Integer(2), coefficient=Integer(1), valid_from={x:... ->Integer(10)})
>>> t3 = BT(x**Integer(3), coefficient=Integer(10), valid_from={x:... ->Integer(10)})
>>> t1.can_absorb(t2)
True
>>> t2.can_absorb(t1)
False
>>> t1.can_absorb(t3)
True
>>> t3.can_absorb(t1)
True
>>> ET = TermMonoid('exact', GrowthGroup('x^ZZ'), QQ)
>>> t4 = ET(x**3, coefficient=Integer(5))
>>> t1.can_absorb(t4)
True
```

construction()

Return a construction of this term.

INPUT:

Nothing.

OUTPUT:

A pair `(cls, kwds)` such that `cls(**kwds)` equals this term.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermMonoidFactory
sage: TermMonoid = TermMonoidFactory(__main__.TermMonoid)

sage: T = TermMonoid('B', GrowthGroup('x^ZZ'), QQ)
sage: a = T.an_element(); a
B(1/2*x, x >= 42)
sage: cls, kwds = a.construction(); cls, kwds
(<class 'sage.rings.asymptotic.term_monoid.BTermMonoid_with_category.element_class'>,
 {'coefficient': 1/2,
  'growth': 'x,'
  'parent': B-Term Monoid x^ZZ with coefficients in Rational Field,
```

(continues on next page)
Asymptotic Expansions, Release 10.4

(continued from previous page)

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import TermMonoidFactory

>>> TermMonoid = TermMonoidFactory(__main__.TermMonoid)

>>> T = TermMonoid('B', GrowthGroup('x^ZZ'), QQ)

>>> a = T.an_element(); a
B(1/2*x, x >= 42)

>>> cls, kwds = a.construction(); cls, kwds
(<class 'sage.rings.asymptotic.term_monoid.BTermMonoid_with_category.element
˓→class'>,
 {'coefficient': 1/2,
 'growth': x,
 'parent': B-Term Monoid x^ZZ with coefficients in Rational Field,
 'valid_from': {'x': 42}})

>>> cls(**kwds) == a
True

See also:

GenericTerm.construction(),
TermWithCoefficient.construction(),
GenericTermMonoid.from_construction()

class sage.rings.asymptotic.term_monoid.BTermMonoid(term_monoid_factory, growth_group, coefficient_ring, category)

Bases: TermWithCoefficientMonoid

Parent for asymptotic B-terms.

INPUT:

- growth_group – a growth group
- coefficient_ring – the ring which contains the 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

EXAMPLES:

sage: from sage.rings.asymptotic.growth_group import MonomialGrowthGroup
sage: from sage.rings.asymptotic.term_monoid import BTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as ...
˓→TermMonoid

sage: G = MonomialGrowthGroup(ZZ, 'x')
sage: BT = TermMonoid('B', G, QQ)
sage: BT
B-Term Monoid x^ZZ with coefficients in Rational Field
sage: BT is BTermMonoid(TermMonoid, G, QQ)
True
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import MonomialGrowthGroup
>>> from sage.rings.asymptotic.term_monoid import BTermMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid

>>> G = MonomialGrowthGroup(ZZ, 'x')
>>> BT = TermMonoid('B', G, QQ)
>>> BT
B-Term Monoid x^ZZ with coefficients in Rational Field
>>> BT is BTermMonoid(TermMonoid, G, QQ)
True

**Element**

alias of BTerm

**some_elements()**

Return some elements of this B-term monoid.

See TestSuite for a typical use case.

**INPUT:**

Nothing.

**OUTPUT:**

An iterator.

**EXAMPLES:**

```sage
sage: from itertools import islice
sage: from sage.rings.asymptotic.term_monoid import TermMonoidFactory
term_monoid_factory
sage: TermMonoid = TermMonoidFactory('main.TermMonoid')

sage: G = GrowthGroup('z^QQ')

sage: T = TermMonoid('B', G, ZZ)

sage: tuple(islice(T.some_elements(), int(10)))

(B(z^(-1/2), z >= 1),
 B(z^(1/2), z >= 0),
 B(z^2, z >= 42),
 B(z^(-1/2), z >= 0),
 B(2*z^(1/2), z >= 1),
 B(z^2, z >= 42),
 B(z^(-2), z >= 3),
 B(z^2, z >= 42),
 B(2*z^(-1/2), z >= 0),
 B(2*z^(1/2), z >= 1))
```

(continues on next page)
Asymptotic Expansions, Release 10.4

B(z^{1/2}, z >= 3),
B(z^2, z >= 42),
B(z^{-1/2}, z >= 0),
B(2*z^{1/2}, z >= 1),
B(z^{-2}, z >= 3),
B(z^2, z >= 42),
B(2*z^{-1/2}, z >= 0),
B(2*z^{1/2}, z >= 1))
They may also be multiplied with $O$-terms:

```
sage: OT = TermMonoid('O', G, QQ)
sage: ET(x^2, coefficient=42) * OT(x)
O(x^3)
```

Absorption for asymptotic exact terms relates to addition:

```
sage: ET(x^2, coefficient=5).can_absorb(ET(x^5, coefficient=12))
False
sage: ET(x^2, coefficient=5).can_absorb(ET(x^2, coefficient=1))
True
sage: ET(x^2, coefficient=5).absorb(ET(x^2, coefficient=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, coefficient=42).can_absorb(ET(x^2, coefficient=-42))
True
sage: ET(x^2, coefficient=42).absorb(ET(x^2, coefficient=-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
```

4.5. (Asymptotic) Term Monoids
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.

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:

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

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

is_exact()

Return whether this term is an exact term.

OUTPUT:
A boolean.

EXAMPLES:

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

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

(continues on next page)
is_little_o_of_one()

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

**INPUT:**

Nothing.

**OUTPUT:**

A boolean.

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
    as 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
```

```python
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
```

```python
sage: from sage.all import *
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ * y^ZZ'), QQ)
    as TermMonoid
sage: T(x * y^(-1)).is_little_o_of_one()
False
sage: T(x^(-1) * y).is_little_o_of_one()
False
```

Asymptotic Expansions, Release 10.4

(continued from previous page)

```python
>>> T('x^(-2) * y^(-3)').is_little_o_of_one()
True
```

```python
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
```

```python
from sage.all import *
```

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

### log_term (base=None, locals=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.
- `locals` – a dictionary which may contain the following keys and values:
  - `'log'` – value: a function. If not used, then the usual `log` 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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
growth_group = GrowthGroup('x^ZZ * log(x)^ZZ')
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
  as TermMonoid
sage: T = TermMonoid('exact', growth_group, SR)
sage: T(3*x^2).log_term()
(3, log(x))
sage: T(x^1234).log_term()
(1234, log(x))
sage: T(49*x^7).log_term(base=7)
(2, 7*log(x))
```

4.5. (Asymptotic) Term Monoids 131
```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> T = TermMonoid('exact', GrowthGroup('x^ZZ * log(x)^ZZ'), SR)
>>> T(Integer(3)*x**Integer(2)).log_term()
(log(3), 2*log(x))
>>> T(x**Integer(1234)).log_term()
(1234*log(x),)
>>> T(Integer(49)*x**Integer(7)).log_term(base=Integer(7))
(2, 7/log(7)*log(x))
```

```python
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)
(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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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
```
Asymptotic Expansions, Release 10.4

>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GrowthGroup

>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as _

TermMonoid

>>> T = TermMonoid('exact', GrowthGroup('QQ^x * x^ZZ * log(x)^ZZ'), QQ)

>>> T('x').rpow(Integer(2))
2^x

>>> T('log(x)').rpow('e')
x

>>> T('42*log(x)').rpow('e')
x^42

>>> T('3*x').rpow(Integer(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 * Signs^x
> *previous* TypeError: unsupported operand parent(s) for *:
'Growth Group QQ^x * x^ZZ * log(x)^ZZ * Signs^x' and
'Growth Group ZZ^(x^2)'

>>> from sage.all import *

>>> T = TermMonoid('exact', GrowthGroup('(QQ_+)^n * n^QQ'), SR)

>>> n = T('n')

>>> n.rpow(2)
2^n

sage: _.parent()
Exact Term Monoid QQ^n * n^QQ with coefficients in Symbolic Ring

>>> from sage.all import *

>>> T = TermMonoid('exact', GrowthGroup('(QQ_+)^n * n^QQ'), SR)

>>> n = T('n')

>>> n.rpow(Integer(2))
2^n

sage: _.parent()
Exact Term Monoid QQ^n * n^QQ with coefficients in Symbolic Ring

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

Bases: 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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import ExactTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as _

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(TermMonoid, G_ZZ, ZZ); ET_ZZ
Exact Term Monoid x^ZZ with coefficients in Integer Ring
sage: ET_QQ = ExactTermMonoid(TermMonoid, G_QQ, QQ); ET_QQ
Exact Term Monoid x^QQ with coefficients in Rational Field
sage: ET_QQ.coerce_map_from(ET_ZZ)
Coercion map:
  From: Exact Term Monoid x^ZZ with coefficients in Integer Ring
  To: Exact Term Monoid x^QQ with coefficients in Rational Field

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import ExactTermMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as _

>>> G_ZZ = GrowthGroup('x^ZZ'); x_ZZ = G_ZZ.gen()
>>> G_QQ = GrowthGroup('x^QQ'); x_QQ = G_QQ.gen()
>>> ET_ZZ = ExactTermMonoid(TermMonoid, G_ZZ, ZZ); ET_ZZ
Exact Term Monoid x^ZZ with coefficients in Integer Ring
>>> ET_QQ = ExactTermMonoid(TermMonoid, G_QQ, QQ); ET_QQ
Exact Term Monoid x^QQ with coefficients in Rational Field
>>> ET_QQ.coerce_map_from(ET_ZZ)
Coercion 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:

```python
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
```

```python
>>> from sage.all import *
>>> TermMonoid('exact', G_ZZ, ZZ) is ET_ZZ
True
>>> 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: 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: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: G = GrowthGroup('x^ZZ'); x = G.gen()

sage: t = GenericTermMonoid(TermMonoid, 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
from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
G_QQ = GrowthGroup('x^QQ'); x = G_QQ.gen()
OT = TermMonoid('O', G_QQ, coefficient_ring=QQ)
ET = TermMonoid('exact', G_QQ, coefficient_ring=QQ)

ot1 = OT(x); ot2 = OT(x^2)
et1 = ET(x, coefficient=100); et2 = ET(x^2, coefficient=2)
et3 = ET(x^2, coefficient=1); et4 = ET(x^2, coefficient=-2)
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> G_QQ = GrowthGroup('x^QQ'); x = G_QQ.gen()
>>> OT = TermMonoid('O', G_QQ, coefficient_ring=QQ)
>>> ET = TermMonoid('exact', G_QQ, coefficient_ring=QQ)
>>> ot1 = OT(x); ot2 = OT(x^2)
>>> et1 = ET(x, coefficient=Integer(100)); et2 = ET(x^2, coefficient=Integer(2))
>>> et3 = ET(x^2, coefficient=Integer(1))
```

\(O\)-Terms are able to absorb other \(O\)-terms and exact terms with weaker or equal growth.

```sage
sage: ot1.absorb(ot1)
O(x)
sage: ot1.absorb(et1)
O(x)
sage: ot1.absorb(et1) is ot1
True
```

```sage
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> G_QQ = GrowthGroup('x^QQ'); x = G_QQ.gen()
>>> OT = TermMonoid('O', G_QQ, coefficient_ring=QQ)
>>> ET = TermMonoid('exact', G_Q_Q, coefficient_ring=QQ)
>>> ot1 = OT(x); ot2 = OT(x^2)
>>> et1 = ET(x, coefficient=Integer(100)); et2 = ET(x^2, coefficient=Integer(2))
>>> et3 = ET(x^2, coefficient=Integer(1))
>>> et4 = ET(x^2, coefficient=-Integer(2))
```

```
>>> from sage.all import *
>>> OT = TermMonoid('O', G_Q_Q, coefficient_ring=QQ)
>>> ET = TermMonoid('exact', G_Q_Q, coefficient_ring=QQ)
>>> ot1 = OT(x); ot2 = OT(x^2)
>>> et1 = ET(x, coefficient=Integer(100)); et2 = ET(x^2, coefficient=Integer(2))
>>> et3 = ET(x^2, coefficient=Integer(1))
```
*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
eval = et2.absorb(et3)
3*x^2
sage: eval = et3.absorb(et2)
3*x^2
sage: eval = et3.absorb(et4)
-x^2
```

```python
>>> from sage.all import *
>>> eval = et2.absorb(et3)
3*x^2
>>> eval = et3.absorb(et2)
3*x^2
>>> eval = 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
eval = et2.absorb(et4)
sage: eval = et4.absorb(et2)
None
```

```python
>>> from sage.all import *
>>> eval = et2.absorb(et4)
>>> eval = et4.absorb(et2)
None
```

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
sage: from sage.rings.asymptotic.growth_group import GenericGrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
as TermMonoid
sage: G = GenericGrowthGroup(ZZ)
sage: T = GenericTermMonoid(TermMonoid, 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
```
construction()

Return a construction of this term.

INPUT:

Nothing.

OUTPUT:

A pair (cls, kwds) such that cls(**kwds) equals this term.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
growth_group = GrowthGroup('x^ZZ', QQ)
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
as_TermMonoid
T = DefaultTermMonoidFactory(growth_group, QQ)
sage: a = T.an_element(); a
O(x)
sage: cls, kwds = a.construction(); cls, kwds
(<class 'sage.rings.asymptotic.term_monoid.OTermMonoid_with_category.element_class'>,
{'growth': x, 'parent': O-Term Monoid x^ZZ with implicit coefficients in Rational Field})
sage: cls(**kwds) == a
True
```
See also:

`TermWithCoefficient.construction()`,

`GenericTermMonoid.from_construction()`

**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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: T = GenericTermMonoid(TermMonoid, 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
```

```python
>>> from sage.all import *

>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import GenericTermMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid

>>> T = GenericTermMonoid(TermMonoid, GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
>>> t = T.an_element(); t
Generic Term with growth x*log(x)
>>> t.is_constant()
False
```

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

```python
>>> from sage.all import *

>>> T = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
>>> T('x').is_constant()
```

(continues on next page)
is_exact()

Return whether this term is an exact term.

OUTPUT:
A boolean.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import GenericTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: T = GenericTermMonoid(TermMonoid, 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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: T = TermWithCoefficientMonoid(TermMonoid, GrowthGroup('x^ZZ'), QQ)
sage: T.an_element().is_little_o_of_one()
False
```

(continues on next page)
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
TermWithCoefficient Monoid x^ZZ with coefficients in Rational Field.

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import (GenericTermMonoid, ...
... TermWithCoefficientMonoid)
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as
... TermMonoid

>>> T = GenericTermMonoid(TermMonoid, GrowthGroup('x^ZZ'), QQ)
>>> 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
GenericTerm Monoid x^ZZ with (implicit) coefficients in Rational Field.
>>> T = TermWithCoefficientMonoid(TermMonoid, GrowthGroup('x^ZZ'), QQ)
>>> 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
TermWithCoefficient Monoid x^ZZ with coefficients in Rational Field.

log_term (base=None, locals=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.
- locals – a dictionary which may contain the following keys and values:
  - 'log' – value: a function. If not used, then the usual log 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: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...

4.5. (Asymptotic) Term Monoids

141
sage: T = GenericTermMonoid(TermMonoid, 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.

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import GenericTermMonoid
>>> T = GenericTermMonoid(TermMonoid, GrowthGroup('x^ZZ'), QQ)
>>> 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(TermMonoid, 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.

>>> from sage.all import *
>>> from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
>>> T = TermWithCoefficientMonoid(TermMonoid, GrowthGroup('x^ZZ'), QQ)
>>> 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: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid

(continues on next page)
sage: T = GenericTermMonoid(TermMonoid, 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
GenericTerm Monoid x^ZZ * log(x)^ZZ with (implicit) coefficients in RationalField.

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 DefaultTermMonoidFactory as 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(term_monoid_factory, growth_group, coefficient_ring, category)

Bases: UniqueRepresentation, Parent, WithLocals

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
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ

    sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as
    sage: G_x = GrowthGroup('x^ZZ'); x = G_x.gen()
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ
    sage: G_y = GrowthGroup('y^QQ'); y = G_y.gen()
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ
    sage: T_x_ZZ = GenericTermMonoid(TermMonoid, G_x, QQ)
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ
    sage: T_y_QQ = GenericTermMonoid(TermMonoid, G_y, QQ)
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ
    sage: T_x_ZZ
    GenericTerm Monoid x^ZZ with (implicit) coefficients in Rational Field
data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAkAAAAcCAIAAACQdOQ4AAAABGdBTUEAALGPC/xhBQAAAAB3RJ
    sage: T_y_QQ
    GenericTerm Monoid y^QQ with (implicit) coefficients in Rational Field
Element

alias of GenericTerm

change_parameter\( (growth\_group=\text{None}, \text{coefficient\_ring}=\text{None})\)

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

INPUT:

- \text{growth\_group} -- (default: \text{None}) the new growth group.
- \text{coefficient\_ring} -- (default: \text{None}) the new coefficient ring.

OUTPUT:

A term monoid.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: E = TermMonoid(exact, GrowthGroup(n^ZZ), ZZ)
```

```python
sage: E.change_parameter(coefficient_ring=QQ)
```

```
Exact Term Monoid n^ZZ with coefficients in Rational Field
```

```python
sage: E.change_parameter(growth_group=GrowthGroup(n^QQ))
```

```
Exact Term Monoid n^QQ with coefficients in Integer Ring
```

property coefficient_ring

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

EXAMPLES:

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

```
Integer Ring
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import GenericTermMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
```

```
GenericTermMonoid(TermMonoid, GrowthGroup('x^ZZ'), ZZ).coefficient_ring
```

```
Integer Ring
```
from_construction (construction, **kwds_overrides)

Create a term from the construction of another term.

INPUT:

• construction – a pair (cls, kwds_construction)

• kwds_overrides – a dictionary

OUTPUT:

A term.

EXAMPLES:

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
˓→as TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: x = G.gen()
sage: T = TermMonoid('O', G, QQ)
sage: o = T.an_element()
```

We use a construction directly as input:

```
sage: T.from_construction(o.construction())
O(x)
```

We can override the given data:

```
sage: T.from_construction(o.construction(), growth=x^2)
O(x^2)
```

A minimalistic example:

```
sage: T.from_construction((None, {'growth': x}))
O(x)
```
See also:

```
GenericTerm.construction(), TermWithCoefficient.construction()
```

**property 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 DefaultTermMonoidFactory
sage: TermMonoid('exact', GrowthGroup('x^2'), QQ).growth_group
Growth Group x^2
```

**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(TermMonoid, G, QQ)
sage: t1 = T(x); t2 = T(x^2)
sage: T.le(t1, t2)
True
```

```bash
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import GenericTermMonoid
>>> G = GrowthGroup('x^ZZ'); x = G.gen()
>>> T = GenericTermMonoid(TermMonoid, G, QQ)
```

4.5. (Asymptotic) Term Monoids
some_elements()

Return some elements of this term monoid.

See TestSuite for a typical use case.

INPUT:

Nothing.

OUTPUT:

An iterator.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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), ...)
```

148 Chapter 4. Asymptotic Expansions — Table of Contents

term_monoid(type)

Return the term monoid of specified type.

INPUT:

• type – ‘O’ or ‘exact’, or an instance of an existing term monoid. See TermMonoidFactory for more details.

OUTPUT:

A term monoid object derived from GenericTermMonoid.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: E = TermMonoid(exact, GrowthGroup('x^ZZ'), ZZ); E
Exact Term Monoid x^ZZ with coefficients in Integer Ring
sage: E.term_monoid(O)
O-Term Monoid x^ZZ with implicit coefficients in Integer Ring
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> E = TermMonoid(exact, GrowthGroup('x^ZZ'), ZZ); E
Exact Term Monoid x^ZZ with coefficients in Integer Ring
>>> E.term_monoid(O)
O-Term Monoid x^ZZ with implicit coefficients in Integer Ring
```
property term_monoid_factory

The term monoid factory capable of creating this term monoid.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
sage: DefaultTermMonoidFactory('exact', GrowthGroup('x^ZZ'), ZZ).term_monoid_factory
Term Monoid Factory 'sage.rings.asymptotic.term_monoid.DefaultTermMonoidFactory'

sage: TermMonoid('exact', GrowthGroup('x^ZZ'), ZZ).term_monoid_factory
Term Monoid Factory '__main__.TermMonoid'
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
>>> DefaultTermMonoidFactory('exact', GrowthGroup('x^ZZ'), ZZ).term_monoid_factory
Term Monoid Factory 'sage.rings.asymptotic.term_monoid.DefaultTermMonoidFactory'

>>> TermMonoid('exact', GrowthGroup('x^ZZ'), ZZ).term_monoid_factory
Term Monoid Factory '__main__.TermMonoid'
```

class sage.rings.asymptotic.term_monoid.OTerm

```python
class sage.rings.asymptotic.term_monoid.OTerm(parent, growth)

Bases: 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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
```

(continues on next page)
The conversion of growth elements also works for the creation of \(O\)-terms:

```python
sage: x = SR('x'); x.parent()
Symbolic Ring
sage: OT(x^17)
O(x^17)
```

### can_absorb \(\text{other}\)

Check whether this \(O\)-term can absorb \(\text{other}\).

**INPUT:**

- \(\text{other}\) – 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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> OT = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
>>> t1 = OT(x**Integer(21)); t2 = OT(x**Integer(42))
>>> t1.can_absorb(t2)
False
>>> 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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> T = TermMonoid('O', GrowthGroup('x^ZZ'), QQ)
>>> T(x).is_little_o_of_one()
```

(continues on next page)
False

```python
>>> T(Integer(1)).is_little_o_of_one()
False
```  

```python
>>> T(x**(-Integer(1))).is_little_o_of_one()
True
```  

```python
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
```  

```python
>>> from sage.all import *
```  

```python
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
```  

```python
>>> from sage.all import *
```  

```python
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, locals=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.
- **locals** – a dictionary which may contain the following keys and values:
  - `'log'` – value: a function. If not used, then the usual `log` 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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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)),)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid

>>> T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)

>>> T(x^Integer(2)).log_term()
(O(log(x)),)
>>> T(x^Integer(1234)).log_term()
(O(log(x)),)
```

```python
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)))
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid

>>> T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ * y^ZZ * log(y)^ZZ'), QQ)

>>> T(x * y).log_term()
(O(log(x)), O(log(y)))
```

See also:

* `ExactTerm.log_term()`

\texttt{rpow}(\texttt{base})

Return the power of \texttt{base} to this O-term.

INPUT:

\begin{itemize}
  \item \texttt{base} – an element or \texttt{e}.
\end{itemize}

OUTPUT:

A term.

Note: For OTerm, the powers can only be constructed for exponents \( O(1) \) or if \texttt{base} is 1.

EXAMPLES:
```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> T = TermMonoid('O', GrowthGroup('x^ZZ * log(x)^ZZ'), QQ)
>>> T(Integer(1)).rpow('e')
O(1)
>>> T(Integer(1)).rpow(Integer(2))
O(1)
```

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

```python
>>> from sage.all import *
>>> T.an_element().rpow(Integer(1))
1
>>> T('x^2').rpow(Integer(1))
1
```

```python
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
```

```python
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
```

```python
>>> from sage.all import *
>>> 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
```

```python
>>> from sage.all import *
>>> 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
```

```python
class sage.rings.asymptotic.term_monoid.OTermMonoid
Bases: GenericTermMonoid
```

Chapter 4. Asymptotic Expansions — Table of Contents

154
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:

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import OTermMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: G_x_ZZ = GrowthGroup('x^ZZ')
```

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

```python
sage: TermMonoid('O', G_x_ZZ, QQ) is OT_x_ZZ
True
```

Element

alias of `OTerm`

```python
class sage.rings.asymptotic.term_monoid.TermMonoidFactory (name, exact_term_monoid_class=None, O_term_monoid_class=None, B_term_monoid_class=None)
```

Bases: `UniqueRepresentation, UniqueFactory`

Factory for asymptotic term monoids. It can generate the following term monoids:
• $OTermMonoid$,
• $ExactTermMonoid$,
• $BTermMonoid$.

**Note:** An instance of this factory is available as $DefaultTermMonoidFactory$.

**INPUT:**

- $term_monoid$ – the kind of terms held in the new term monoid. Either a string 'exact', 'O' (capital letter O) or 'B' 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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
˓→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

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
˓→TermMonoid
>>> G = GrowthGroup('x^ZZ')
>>> TermMonoid('O', G, QQ)
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
>>> 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 * Signs^m * log(n)^ZZ
with coefficients in Integer Ring
```

(continues on next page)
Exact Term Monoid \( x^{\mathbb{Z}} \) with coefficients in Rational Field

```python
>>>
TermMonoid('0', asymptotic_ring=R)
O-Term Monoid \( x^{\mathbb{Z}} \) with implicit coefficients in Rational Field
```

```python
>>>
TermMonoid('exact', 'QQ^m * m^QQ * log(n)^ZZ', ZZ)
```

```
Exact Term Monoid \( QQ^m * m^QQ * \text{Signs}^m * \log(n)^ZZ \) with coefficients in Integer Ring
```

### create_key_and_extra_args

\( \text{term_monoid}, \text{growth_group}=\text{None}, \text{coefficient_ring}=\text{None}, \text{asymptotic_ring}=\text{None}, **\text{kwds} \)

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

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: G = GrowthGroup('x^ZZ')
sage: TermMonoid.create_key_and_extra_args('0', G, QQ)
((<class 'sage.rings.asymptotic.term_monoid.OTermMonoid'>, Growth Group \( x^{\mathbb{Z}} \), Rational Field), {})
sage: TermMonoid.create_key_and_extra_args('exact', G, ZZ)
((<class 'sage.rings.asymptotic.term_monoid.ExactTermMonoid'>, Growth Group \( x^{\mathbb{Z}} \), 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'
```

### create_object

\( \text{version}, \text{key}, **\text{kwds} \)

Create an object from the given arguments.

**EXAMPLES:**

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> G = GrowthGroup('x^ZZ')
>>> TermMonoid.create_key_and_extra_args('0', G, QQ)
((<class 'sage.rings.asymptotic.term_monoid.OTermMonoid'>, Growth Group \( x^{\mathbb{Z}} \), Rational Field), {})
>>> TermMonoid.create_key_and_extra_args('exact', G, ZZ)
((<class 'sage.rings.asymptotic.term_monoid.ExactTermMonoid'>, Growth Group \( x^{\mathbb{Z}} \), Integer Ring), {})
>>> 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'
```

4.5. (Asymptotic) Term Monoids

(continues on next page)
sage: TermMonoid('0', 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

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
   TermMonoid
>>> G = GrowthGroup('x^ZZ')
>>> TermMonoid('0', G, QQ) # indirect doctest
O-Term Monoid x^ZZ with implicit coefficients in Rational Field
>>> 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: 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: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
   TermMonoid
sage: G = GrowthGroup('x^ZZ'); x = G.gen()
sage: CT_ZZ = TermWithCoefficientMonoid(TermMonoid, G, ZZ)
sage: CT_QQ = TermWithCoefficientMonoid(TermMonoid, G, QQ)
sage: CT_ZZ(x**Integer(2), coefficient=Integer(5))
Term with coefficient 5 and growth x^2
sage: CT_QQ(x**Integer(3), coefficient=Integer(3)/Integer(8))
Term with coefficient 3/8 and growth x^3

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
   TermMonoid
>>> G = GrowthGroup('x^ZZ'); x = G.gen()
>>> CT_ZZ = TermWithCoefficientMonoid(TermMonoid, G, ZZ)
>>> CT_QQ = TermWithCoefficientMonoid(TermMonoid, G, QQ)
>>> CT_ZZ(x**Integer(2), coefficient=Integer(5))
Term with coefficient 5 and growth x^2
>>> CT_QQ(x**Integer(3), coefficient=Integer(3)/Integer(8))
Term with coefficient 3/8 and growth x^3
**construction()**

Return a construction of this term.

**INPUT:**

Nothing.

**OUTPUT:**

A pair \((\texttt{cls}, \texttt{kwds})\) such that \(\texttt{cls(**kwds)}\) equals this term.

**EXAMPLES:**

```
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory
sage: T = TermMonoid(exact, GrowthGroup('x^ZZ'), QQ)
sage: a = T.an_element(); a
1/2*x
sage: cls, kwds = a.construction(); cls, kwds
(<class 'sage.rings.asymptotic.term_monoid.ExactTermMonoid_with_category.element_class'>,
 {'coefficient': 1/2, 'growth': x, 'parent': Exact Term Monoid x^ZZ with coefficients in Rational Field})
sage: cls(**kwds) == a
True
```

See also:

- `GenericTerm.construction()`
- `GenericTermMonoid.from_construction()`

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

**Bases:** `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:**

4.5. (Asymptotic) Term Monoids
- **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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import TermWithCoefficientMonoid
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...

˓→ TermMonoid

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(TermMonoid, G_ZZ, QQ); TC_ZZ
TermWithCoefficientMonoid x^ZZ with coefficients in Rational Field
sage: TC_QQ = TermWithCoefficientMonoid(TermMonoid, G_QQ, QQ); TC_QQ
TermWithCoefficientMonoid x^QQ with coefficients in Rational Field
sage: TC_ZZ == TC_QQ or TC_ZZ is TC_QQ
False
sage: TC_QQ.coerce_map_from(TC_ZZ)
Coercion map:
  From: TermWithCoefficientMonoid x^ZZ with coefficients in Rational Field
  To:   TermWithCoefficientMonoid x^QQ with 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:

```python
sage: from itertools import islice
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as 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(), int(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))
```

```
>>> from sage.all import *
>>> from itertools import islice
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> G = GrowthGroup('z^QQ')
>>> T = TermMonoid('exact', G, ZZ)
>>> tuple(islice(T.some_elements(), int(Integer(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.ZZeroCoefficientError
```

```
Bases: ValueError

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

Let one of the two passed terms absorb the other.

**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:**

```python
sage: from sage.rings.asymptotic.growth_group import GrowthGroup
sage: from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as TermMonoid
sage: from sage.rings.asymptotic.term_monoid import 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)
```

4.5. (Asymptotic) Term Monoids
>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
    TermMonoid
>>> from sage.rings.asymptotic.term_monoid import absorption
>>> T = TermMonoid('O', GrowthGroup('x^ZZ'), ZZ)
>>> absorption(T(x**Integer(2)), T(x**Integer(3)))
O(x^3)
>>> absorption(T(x**Integer(3)), T(x**Integer(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.

>>> from sage.all import *
>>> from sage.rings.asymptotic.growth_group import GrowthGroup
>>> from sage.rings.asymptotic.term_monoid import DefaultTermMonoidFactory as...
    TermMonoid
>>> from sage.rings.asymptotic.term_monoid import can_absorb
sage: T = TermMonoid('exact', GrowthGroup('x^ZZ'), ZZ)
sage: can_absorb(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 DefaultTermMonoidFactory as...
    TermMonoid
sage: from sage.rings.asymptotic.term_monoid import 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

(continues on next page)
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

class sage.rings.asymptotic.misc.Locals
    Bases: dict
    
    A frozen dictionary-like class for storing locals of an AsymptoticRing.

    EXAMPLES:

        sage: from sage.rings.asymptotic.misc import Locals
        sage: locals = Locals({'a': 42})
        sage: locals['a']
        42

        >>> from sage.all import *
        >>> from sage.rings.asymptotic.misc import Locals
        >>> locals = Locals({'a': Integer(42)})
        >>> locals['a']
        42

    The object contains default values (see default_locals()) for some keys:

        sage: locals['log']
        <function log at 0x...>

        >>> from sage.all import *
        >>> locals['log']
        <function log at 0x...>

    default_locals()
        Return the default locals used in the AsymptoticRing.

    OUTPUT:
A dictionary.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.misc import Locals
sage: locals = Locals({'a': 2, 'b': 1})
sage: locals
{'a': 2, 'b': 1}
sage: locals.default_locals()
{'log': <function log at 0x...>}
sage: locals['log']
<function log at 0x...>
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.misc import Locals
>>> locals = Locals({'a': Integer(2), 'b': Integer(1)})
>>> locals
{'a': 2, 'b': 1}
>>> locals.default_locals()
{'log': <function log at 0x...>}
>>> locals['log']
<function log at 0x...>
```

**exception** `sage.rings.asymptotic.misc.NotImplementedBZero`(asymptotic_ring=None, var=None, exact_part=0)

Bases: `NotImplementedError`

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

**exception** `sage.rings.asymptotic.misc.NotImplementedOZero`(asymptotic_ring=None, var=None, exact_part=0)

Bases: `NotImplementedError`

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

**class** `sage.rings.asymptotic.misc.WithLocals`

Bases: `SageObject`

A class extensions for handling local values; see also `Locals`.

This is used in the `AsymptoticRing`.

EXAMPLES:

```python
sage: A.<n> = AsymptoticRing('n^ZZ', QQ, locals={'a': 42})
sage: A.locals()
{'a': 42}
```

```python
>>> from sage.all import *
>>> A = AsymptoticRing('n^ZZ', QQ, locals={'a': Integer(42)}, names=('n',)); (n,)
˓→= A._first_ngens(1)
>>> A.locals()
{'a': 42}
```

`locals (locals=None)`

Return the actual `Locals` object to be used.

**INPUT:**
• **locals** — an object

  If **locals** is not **None**, then a **Locals** object is created and returned. If **locals** is **None**, then a stored **Locals** object, if any, is returned. Otherwise, an empty (i.e. no values except the default values) **Locals** object is created and returned.

**OUTPUT:**
A **Locals** object.

**sage.rings.asymptotic.misc.bidirectional_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 **bidirectional_merge_overlapping()** returns the pair \((acb, acb)\).

Suppose a **key**-function is specified and \(A = ac_A\) and \(B = cb_B\), where the list of keys of the elements of \(c_A\) equals the list of keys of the elements of \(c_B\). Then **bidirectional_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.

**sage.rings.asymptotic.misc.bidirectional_merge_sorted(\(A, B, key=None\))**

Merge the two tuples/lists, keeping the orders provided by them.

**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 containing all elements totally ordered. (The first component uses **A** as a merge base, the second component **B**.)

If merging fails, then a **RuntimeError** is raised.

**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:

```python
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.
```

```python
sage: raise combine_exceptions(ValueError('Outer.'),
                   ....: TypeError('Inner1.'), TypeError('Inner2.'))
Traceback (most recent call last):
...  
ValueError: Outer.
> *previous* TypeError: Inner1.
> *and* TypeError: Inner2.
```

```python
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.
```

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

```python
>>> raise combine_exceptions(ValueError('Outer.'),
                   ....: TypeError('Inner1.'), TypeError('Inner2.'))
Traceback (most recent call last):
...  
ValueError: Outer.
> *previous* TypeError: Inner1.
> *and* TypeError: Inner2.
```

```python
>>> 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)'

>>> from sage.all import *
>>> from sage.rings.asymptotic.misc import log_string

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(RR)
'RR'
sage: parent_to_repr_short(CC)
'CC'
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'

>>> from sage.all import *
>>> from sage.rings.asymptotic.misc import parent_to_repr_short
```
sage.rings.asymptotic.misc.\texttt{repr\_op}(left, op, right=\texttt{None}, latex=\texttt{False})

Create a string \texttt{left op right} with taking care of parentheses in its operands.

**INPUT:**
- \texttt{left} – an element.
- \texttt{op} – a string.
- \texttt{right} – an element.
- \texttt{latex} – (default: \texttt{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'
```
The possible short representations are shown in the examples below.

EXAMPLES:

```python
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
sage: repr_short_to_parent('UU')
Group of Roots of Unity
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.misc import repr_short_to_parent
>>> repr_short_to_parent('ZZ')
Integer Ring
>>> repr_short_to_parent('QQ')
Rational Field
>>> repr_short_to_parent('SR')
Symbolic Ring
>>> repr_short_to_parent('NN')
Non negative integer semiring
>>> repr_short_to_parent('UU')
Group of Roots of Unity
```

```
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.

```
sage.rings.asymptotic.misc.strip_symbolic(expression)
```

Return, if possible, the underlying (numeric) object of the symbolic expression.

If `expression` is not symbolic, then `expression` is returned.

INPUT:

- `expression` – an object

OUTPUT:

An object.

EXAMPLES:
sage: from sage.rings.asymptotic.misc import strip_symbolic
sage: strip_symbolic(SR(2)); _.parent()
2
Integer Ring
sage: strip_symbolic(SR(2/3)); _.parent()
2/3
Rational Field
sage: strip_symbolic(SR('x')); _.parent()
x
Symbolic Ring
sage: strip_symbolic(pi); _.parent()
pi
Symbolic Ring

>>> from sage.all import *
>>> from sage.rings.asymptotic.misc import strip_symbolic
>>> strip_symbolic(SR(Integer(2))); _.parent()
2
Integer Ring
>>> strip_symbolic(SR(Integer(2)/Integer(3))); _.parent()
2/3
Rational Field
>>> strip_symbolic(SR('x')); _.parent()
x
Symbolic Ring
>>> strip_symbolic(pi); _.parent()
pi
Symbolic Ring

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.

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 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
```

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

>>> transform_category(Objects(), S, A)
Traceback (most recent call last):
...
ValueError: Category of objects is not a subcategory of Category of sets.

>>> transform_category(Objects(), S, A)
Category of sets

>>> transform_category(AdditiveAssociative, False)
Category of semigroups

>>> transform_category(AdditiveUnital, False)
Category of monoids

>>> transform_category(AdditiveInverse, False)
Category of groups

>>> transform_category(AdditiveCommutative, False)
Category of commutative groups

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

>>> from sage.all import *

>>> 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

>>> from sage.all import *

>>> transform_category(ZZ.category(), S, A)
Category of commutative groups

>>> transform_category(QQ.category(), S, A)
Category of commutative groups

>>> transform_category(SR.category(), S, A)
Category of commutative groups

>>> transform_category(Fields(), S, A)
Category of commutative groups

(continues on next page)
4.7 Asymptotics of Multivariate Generating Series

Let \( F(x) = \sum_{\nu \in \mathbb{N}^d} F_\nu x^\nu \) 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 \( r\alpha \in \mathbb{N}^d \) for \( \alpha \) 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 algebraic variety \( H = 0 \).

The algorithms and formulas implemented here come from [RW2008] and [RW2012]. For a general reference take a look in the book [PW2013].
sage: decomp = F.asymptotic_decomposition(alpha)
sage: decomp
(0, []) +
(-1/2*r^2*(x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 6*x/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 9/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)) - 1/2*r*(5*x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 24*x/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 27/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)) - 3*x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) - 9*r*(x^5 + 9*x^4 + 27*x^3 + 27*x^2) - 9/(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])]

from sage.all import *
R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
FFPD = FractionWithFactoredDenominatorRing(R, SR)
H = (x - Integer(1)/Integer(2))**Integer(3)
Hfac = H.factor()
G = -Integer(1)/(x + Integer(3))/Hfac.unit()
F = FFPD(G, Hfac)
F
(-1/(x + 3), [(x - 1/2, 3)])
alpha = [Integer(1)]
decomp = F.asymptotic_decomposition(alpha)
decomp
(0, []) +
(-1/2*r^2*(x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 6*x/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 9/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)) - 1/2*r*(5*x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 24*x/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) + 27/(x^5 + 9*x^4 + 27*x^3 + 27*x^2)) - 3*x^2/(x^5 + 9*x^4 + 27*x^3 + 27*x^2) - 9*r*(x^5 + 9*x^4 + 27*x^3 + 27*x^2) - 9/(x^5 + 9*x^4 + 27*x^3 + 27*x^2), [(x - 1/2, 1)])
F1 = decomp[Integer(1)]
p = {x: Integer(1)/Integer(2)}
asy = F1.asymptotics(p, alpha, Integer(3))
asy
(8/343*(49*r^2 + 161*r + 114)*2^r, 2, 8/7*r^2 + 184/49*r + 912/343)
F.relative_error(asy[Integer(0)], alpha, [Integer(1), Integer(2), Integer(4), Integer(8), Integer(16)], asy[Integer(1)])
[((1,), 7.555555556, [7.556851312], [-0.0001714971672]),
Another smooth point example (Example 5.4 of [RW2008]):

```python
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
```

```python
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
```

```python
sage: p = s[0]
sage: asy = F.asymptotics(p, alpha, 1, verbose=True)

Creating auxiliary functions...

Computing derivatives of auxiliary functions...

Computing derivatives of more auxiliary functions...

Computing second order differential operator actions...
```

```python
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)))
```

```python
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)))
```

```python
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8, 16], asy[1])
```

```
[((4,), 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...]])
```
G = (Integer(1) - q*x)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> alpha = list(qq*vector([Integer(2), Integer(1) - q]))
>>> alpha
[4, 1]
>>> I = F.smooth_critical_ideal(alpha)
>>> I
Ideal (y^2 - 2*y + 1, x + 1/4*y - 5/4) of
Multivariate Polynomial Ring in x, y over Rational Field
>>> s = solve([SR(z) for z in I.gens()],
...        [SR(z) for z in R.gens()], solution_dict=true)
>>> s == [{SR(x): Integer(1), SR(y): Integer(1)}
True
>>> p = s[Integer(0)]
>>> asy = F.asymptotics(p, alpha, Integer(1), verbose=True)
Creating auxiliary functions...
Computing derivatives of auxiliary functions...
Computing derivatives of more auxiliary functions...
Computing second order differential operator actions...
>>> 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)))
>>> r = SR('r')
>>> tuple((a*r**(Integer(1)/Integer(3))).full_simplify() // r**(Integer(1)/Integer(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)))
>>> F.relative_error(asy[Integer(0)], alpha, [Integer(1), Integer(2), Integer(4),
˓→Integer(8), Integer(16)], asy[Integer(1)])
[((4, 1), 0.1875000000, [0.1953794675..., [-0.0420238262...]),
(8, 2), 0.1523437500, [0.1550727862..., [-0.0179136733...]),
(16, 4), 0.1221771240, [0.1230813519..., [-0.0074009592...]),
(32, 8), 0.09739671811, [0.09768973377..., [-0.0030847572...]),
(64, 16), 0.07744253816, [0.07753639308..., [-0.0012119297...])]

A multiple point example (Example 6.5 of [RW2012]):

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: asy = F.asymptotic_decomposition(alpha); asy
(0, [])
(continues on next page)
\[-\frac{1}{9}r^2(2a^2/x^2 + 2b^2/y^2 - 5a*b/(x*y))
- \frac{1}{9}r(6a/x^2 + 6b/y^2 - 5a/(x*y) - 5b/(x*y))
- \frac{4}{9}x^2 - 4/9y^2 + 5/9/(x*y),
[(x + 2*y - 1, 1), (2*x + y - 1, 1)]\]
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/(1/3)^a*(1/3)^b)^r,
1/(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
\((3*(10*r^2 - 7*r - 3)*2187^r, 2187, 30*r^2 - 21*r - 9)\)

\[3*(10^r^2 - 7*r - 3)*2187^r, 2187, 30^r^2 - 21^r - 9\]
sage: F.relative_error(asy[0], alpha, [1, 2, 4, 8], asy[1])
\[[(4, 3), 30.72702332, [0.0000000000], [1.0000000000]],
((6, 6), 111.9315678, [69.00000000], [0.3835519207]),
((16, 12), 442.7813138, [387.0000000], [0.1259793763]),
((32, 24), 1799.879232, [1743.000000], [0.03160169385])\]

>>> from sage.all import *
>>> R = PolynomialRing(QQ, names=('x', 'y')); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> H = (Integer(1) - Integer(2)*x - y)**Integer(2) * (Integer(1) - x -
-Integer(2)*y)**Integer(2)
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F
\((1, \{(x + 2*y - 1, 2), (2*x + y - 1, 2)\}\)
>>> I = F.singular_ideal()
>>> I
Ideal \((x - 1/3, y - 1/3)\) of
Multivariate Polynomial Ring in \(x, y\) over Rational Field
>>> p = {x: Integer(1)/Integer(3), y: Integer(1)/Integer(3)}
>>> F.is_convenient_multiple_point(p)
(True, 'convenient in variables [x, y]')
>>> alpha = (var('a'), var('b'))
>>> decomp = F.asymptotic_decomposition(alpha); decomp
\((0, []) +
(-1/9*r^2*(2*a^2/x^2 + 2*b^2/y^2 - 5*a*b/(x*y))
- 1/9*r*(6a/x^2 + 6b/y^2 - 5a/(x*y) - 5b/(x*y))
- 4/9/x^2 - 4/9/y^2 + 5/9/(x*y),
[(x + 2*y - 1, 1), (2*x + y - 1, 1)]\)
>>> F1 = decomp[1]
>>> F1.asymptotics(p, alpha, Integer(2))
\(-3*((2*a^2 - 5*a*b + 2*b^2)*r^2 + (a + b)*r + 3)*(1/(1/3)^a*(1/3)^b)^r,
1/(1/3)^a*(1/3)^b), -3*(2*a^2 - 5*a*b + 2*b^2)*r^2 - 3*(a + b)*r - 9\]
>>> alpha = [Integer(4), Integer(3)]
>>> decomp = F.asymptotic_decomposition(alpha)
>>> F1 = decomp[1]
>>> asy = F1.asymptotics(p, alpha, Integer(2))
>>> asy
\((3*(10*r^2 - 7*r - 3)*2187^r, 2187, 30*r^2 - 21*r - 9)\)
>>> F.relative_error(asy[Integer(0)], alpha, [Integer(1), Integer(2), Integer(4),
-Integer(8)], asy[Integer(1)])

4.7. Asymptotics of Multivariate Generating Series
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.FractionWithFactoredDenominator

Bases: 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 FractionWithFactoredDenominatorRing
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)])
```

(continues on next page)
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)])

Singular throws a ‘not implemented’ error when trying to factor in a multivariate polynomial ring over an inexact field:
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 $[q^e \text{ for } (q, e) \in \text{self.denominator_factored()}]$, which could be the zero ideal. The ideal $J$ lies in a polynomial ring over the field `self.denominator_ring.base_ring()` that has $m = \text{len(self.denominator_factored())}$ indeterminates.

OUTPUT:
An ideal.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
sage: R.<x,y> = PolynomialRing(QQ)
```

```python
sage: f = 1/(x^2 * (x*y + 1) * y^3)
```

```python
sage: ff = FFPD(f)
```

```python
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
```

```python
g = J gens()[0]
```

```python
sage: g(*([q^e for q, e in df]) == 0
```

True
```
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> f = Integer(1)/(x**Integer(2) * (x*y + Integer(1)) * y**Integer(3))
>>> ff = FFPD(f)
>>> 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
>>> g = J.gens()[Integer(0)]
>>> df = ff.denominator_factored()
>>> g(*(q**e for q, e in df)) == Integer(0)
True

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: G = exp(x + y)
sage: H = x**Integer(2) * (x*y + 1) * y**Integer(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: J.gens()[0]
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: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == Integer(0)
True

>>> from sage.all import *
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> G = exp(x + y)
>>> H = x**Integer(2) * (x*y + 1) * y**Integer(3)
>>> ff = FFPD(G, H.factor())
>>> 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: J.gens()[0]
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: df = ff.denominator_factored()
sage: g(*(q**e for q, e in df)) == Integer(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

>>> from sage.all import *
>>> f = Integer(1)/(x**Integer(3) * y**Integer(2))
>>> J = FFPD(f).algebraic_dependence_certificate()
>>> 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

>>> from sage.all import *
>>> f = sin(Integer(1))/(x**Integer(3) * y**Integer(2))
>>> J = FFPD(f).algebraic_dependence_certificate()
>>> 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 $q_1^{e_1} \cdots q_m^{e_m}$ 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 [Rai2012], $f$ can be written as

\[
(*) \sum_A \frac{p_A}{\prod_{i \in A} q_i^{b_i}},
\]

where the $b_i$ are positive integers, each $p_A$ is a product of $p$ and an element 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 [Rai2012].

OUTPUT:

An instance of FractionWithFactoredDenominatorSum.

EXAMPLES:

sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y',))
>>> (x, y,) = R._first_ngens(2)


Asymptotic Expansions, Release 10.4

>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> f = Integer(1)/(x**Integer(2) * (x*y + Integer(1)) * y**Integer(3))
>>> ff = FFPD(f)
>>> decomp = ff.algebraic_dependence_decomposition()
>>> decomp
(0, []) + (-x, [(x*y + 1, 1)]) +
(x**2*y**2 - x*y + 1, [(y, 3), (x, 2)])
>>> decomp.sum().quotient() == f
True

>>> 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

>>> from sage.all import *

>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> G = sin(x)
>>> H = x**2 * (x*y + 1) * y**3
>>> f = FFPD(G, H.factor())
>>> decomp = f.algebraic_dependence_decomposition()
>>> 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)])
>>> bool(decomp.sum().quotient() == G/H)
True

>>> 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 \texttt{self}.

The asymptotic decomposition of $F$ is a sum that has the same asymptotic expansion as $f$ in the direction \texttt{alpha} but each summand has a denominator factorization of the form $[(q_1, 1), \ldots, (q_n, 1)]$, where $n$ is at most the \texttt{dimension()} of $F$.

\textbf{INPUT:}

- \texttt{alpha} – a \texttt{d}-tuple of positive integers or symbolic variables
- \texttt{asy_var} – (default: \texttt{None}) a symbolic variable with respect to which to compute asymptotics; if \texttt{None} is given, we set \texttt{asy_var = var('r')}

\textbf{OUTPUT:}

An instance of \texttt{FractionWithFactoredDenominatorSum}.

The output results from a Leinartas decomposition followed by a cohomology decomposition.

\textbf{EXAMPLES:}

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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 + 2*a^2/x + 11*a^2/x^2)*r^2
 - 1/54*(5*a - 2*a/x - 33*a/x^2)*r + 11/27/x^2,
[(x - 1, 1)]) + (-5/27, [(x + 2, 1)])
```

```python
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*r*(a/x - 2*b/y) - 1/3/x + 2/3/y,
[(x + 2*y - 1, 1), (2*x + y - 1, 1)])
```
from sage.all import *
R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
FFPD = FractionWithFactoredDenominatorRing(R, SR)
H = (Integer(1) - Integer(2)*x -y)*(Integer(1) - x -
   Integer(2)*y)**Integer(2)
Hfac = H.factor()
G = Integer(1)/Hfac.unit()
F = FFPD(G, Hfac)
alpha = var('a, b')
F.asymptotic_decomposition(alpha)
(0, []) +
(-1/3*r*(a/x - 2*b/y) - 1/3/x + 2/3/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 [RW2008] and [RW2012].

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:
A smooth point example:

```python
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, []) + (... - 1/2, [(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 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,
3/5*sqrt(5)*sqrt(3)*sqrt(2)*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])
# abs tol 1e-10 # long time
[((4, 3), 2.083333333, [2.092576110, -0.004436533009]),
((8, 6), 2.787374614, [2.790732875, -0.001204811281]),
((16, 12), 3.82659447, [3.827462310, -0.0003143703383]),
((32, 24), 5.328112821, [5.328540787, -0.0000803230388]),
((64, 48), 7.475927885, [7.476079664, -0.00002032302879])
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions...
˓→import FractionWithFactoredDenominatorRing
```

4.7. Asymptotics of Multivariate Generating Series 187
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, []) + (..., [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 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 second-order differential operator actions...
sage: asy
(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)))
```

```
>>> from sage.all import *
```
Asymptotics, Release 10.4

>>> F1 = decomp[Integer(1)]
>>> p = {x: Integer(1), y: Integer(1), z: Integer(1)}
>>> asy = F1.asymptotics(p, alpha, Integer(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...

>>> 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)))

>>> F.relative_error(asy[Integer(0)], alpha, [Integer(1), Integer(2), ...
˓→Integer(4), Integer(8)], asy[Integer(1)]) # long time
[((3, 3, 2), 0.9812164307, [1.515572606], [-0.54458543...]),
((6, 6, 4), 1.576181132, [1.992989399], [-0.2644185...]),
((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 [RW2012].

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, ..., 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 FractionWithFactoredDenominatorRing
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)))

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y', 'z')); (x, y, z,) = R._first_ngens(3)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = (Integer(4) - Integer(2)*x - y - z)*(Integer(4) - x -Integer(2)*y - z)
>>> Hfac = H.factor()
>>> G = Integer(16)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F
(16, [(x + 2*y + z - 4, 1), (2*x + y + z - 4, 1)])
>>> p = {x: Integer(1), y: Integer(1), z: Integer(1)}
>>> alpha = [Integer(3), Integer(3), Integer(2)]
>>> F.asymptotics_multiple(p, alpha, Integer(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))

>>> H = (Integer(1) - x*(Integer(1) + y))*(Integer(1) - Integer(2)*x + (Integer(1) + Integer(2)*y))
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F
(1, [(x*y + x - 1, 1), (2*x^2*y*z + x^2*z - 1, 1)])

>>> p = (x: Integer(1)/Integer(2), z: Integer(4)/Integer(3), y: Integer(1)}
>>> alpha = [Integer(8), Integer(3), Integer(3)]
>>> F.asymptotics_multiple(p, alpha, Integer(2), var(r),
˓→coordinate=Integer(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))))

4.7. Asymptotics of Multivariate Generating Series 191

sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
sage: H = (1 - 2*x - y) * (1 - x - 2*y)
>>> Hfac = H.factor()
>>> G = exp(x + y)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F
(e^(x + y), [(x + 2*y - 1, 1), (2*x + y - 1, 1)])

>>> p = (x: 1/3, y: 1/3)
>>> alpha = (var('a'), var('b'))
>>> F.asymptotics_multiple(p, alpha, 2, var(r))
# long time
(3*(1/((1/3)^a*(1/3)^b))^r*e^(2/3), 1/((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 [RW2008] with the exponent of $H$ equal to 1. Theorem 3.2 is a specialization of Theorem 3.4 of [RW2012] with $n = 1.$
INPUT:

- p – a dictionary with keys that can be coerced to equal `self.denominator_ring.gens()`
- alpha – a tuple of length \(d = \text{self.dimension}()\) 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 \(\{0, \ldots, d - 1\}\) 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 \(\text{self}\) with `numerical` digits of precision; otherwise return exact values
- `verbose` – (default: `False`) print the current state of the algorithm

OUTPUT:

The asymptotic expansion.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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)
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = Integer(2) - Integer(3)*x
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F
(-1/3, [(x - 2/3, 1)])
>>> alpha = [Integer(2)]
>>> p = {x: Integer(2)/Integer(3)}
>>> asy = F.asymptotics_smooth(p, alpha, Integer(3), asy_var=var('r'))
>>> 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 auxiliary 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
(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)))

>>> from sage.all import *
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = Integer(1)-x-y-x*y
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> alpha = [Integer(3), Integer(2)]
>>> p = {y: Integer(1)/Integer(2)*sqrt(Integer(13)) - Integer(3)/Integer(2), x: Integer(1)/Integer(3)*sqrt(Integer(13)) - Integer(2)/Integer(3)}
>>> F.asymptotics_smooth(p, alpha, Integer(2), var('r'), numerical=Integer(3), verbose=True)
Creating auxiliary functions...
Computing derivatives of auxiliary 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)))

(continues on next page)
```python
>>> q = Integer(1)/Integer(2)
>>> qq = q.denominator()
>>> H = Integer(1) - q*x + q*x*y - x**Integer(2)*y
>>> Hfac = H.factor()
>>> G = (Integer(1) - q*x)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> alpha = list(qq*vector([Integer(2), Integer(1) - q]))
>>> alpha
[4, 1]
>>> p = {x: Integer(1), y: Integer(1)}
>>> F.asymptotics_smooth(p, alpha, Integer(5), var('r'), verbose=True) # not tested (140 seconds)
Creating auxiliary functions...
Computing derivatives of auxiliary 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
- 1/96*sqrt(3)*2^(2/3)*gamma(1/3)/(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 \( V_1 \cap \cdots \cap V_n \) of the algebraic varieties \( V_i = \{ x \in L^d | q_i(x) = 0 \} \), 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 [AY1983].

OUTPUT:

An instance of FractionWithFactoredDenominatorSum.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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)])
```

```python
sage: from sage.all import *
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
sage: R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = Integer(1)/(x**Integer(2) + x + Integer(1))**Integer(3)
sage: decomp = FFPD(f).cohomology_decomposition()
sage: decomp
(0, []) + (2/3, [(x^2 + x + 1, 1)])
```
```python
sage: R.<x,y> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: FFPD(1, [(x, 1), (y, 2)]).cohomology_decomposition()
(0, [])
```

```python
>>> from sage.all import *
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> FFPD(Integer(1), [(x, Integer(1)), (y, Integer(2))]).cohomology_decomposition()
(0, [])
```

The following example was fixed in Issue #29465:

```python
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, [(x**2 + y**2 - 1, 1)]) +
(1/3, [(x*y - 1, 1), (x^2 + y^2 - 1, 1)])
```

```python
>>> from sage.all import *
>>> p = Integer(1)
>>> qs = [(x*y - Integer(1), Integer(1)), (x**2 + y**2 - Integer(1), Integer(2))]
>>> f = FFPD(p, qs)
>>> f.cohomology_decomposition()
(0, []) + (-4/3*x*y, [(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 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:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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()
```

(continues on next page)
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

\begin{verbatim}
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions_
˓→import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y', 'z')); (x, y, z,) = R._first_˓→ngens(3)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> G = Integer(1)
>>> H = (Integer(1) - x*(Integer(1) + y)) * (Integer(1) -˓→z**Integer(2)*(Integer(1) + Integer(2)*y))
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> p = {x: Integer(1)/Integer(2), y: Integer(1), z: Integer(4)/Integer(3)}
>>> F.critical_cone(p)

((2, 1, 0), (3, 1, 3/2)), 2-d cone in 3-d lattice N
\end{verbatim}

denominator()

Return the denominator of self.

OUTPUT:

The denominator (i.e., the product of the factored denominator).

EXAMPLES:

\begin{verbatim}
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions_
˓→import FractionWithFactoredDenominatorRing
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()
\end{verbatim}
\begin{verbatim}
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
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions_
˓→import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y', )); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> H = (Integer(1) - x - y - x*y)**2*(Integer(1)-x)
>>> Hfac = H.factor()
>>> G = exp(y)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> 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
\end{verbatim}

denominator_factored()
Return the factorization in \texttt{self.denominator\_ring} of the denominator of \texttt{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:**

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
sage: R.<x,y> = PolynomialRing(QQ)
```

```python
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
```

```python
sage: H = (1 - x - y - x*y)**2*(1-x)
```

```python
sage: Hfac = H.factor()
```

```python
sage: G = exp(y)/Hfac.unit()
```

```python
sage: F = FFPD(G, Hfac)
```

```python
sage: F.denominator_factored()
```

```
[(x - 1, 1), (x*y + x + y - 1, 2)]
```

**property denominator\_ring**

Return the ring of the denominator.

**OUTPUT:**

A ring.

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing

sage: R.<x,y> = PolynomialRing(QQ)
```

```python
sage: FFPD = FractionWithFactoredDenominatorRing(R, SR)
```

```python
sage: H = (1 - x - y - x*y)**2*(1-x)
```

```python
sage: Hfac = H.factor()
```

```python
sage: G = exp(y)/Hfac.unit()
```

```python
sage: F = FFPD(G, Hfac)
```

```python
sage: F.denominator_ring
```

```
Multivariate Polynomial Ring in x, y over Rational Field
```

```python
sage: F = FFPD(G/H)
```

```python
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
```
Asymptotic Expansions, Release 10.4

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> H = (Integer(1) - x - y - x*y)**Integer(2)*(Integer(1)-x)
>>> Hfac = H.factor()
>>> G = exp(y)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> F.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
>>> F = FFPD(G/H)
>>> F
e^y, [(x - 1, 1), (x*y + x + y - 1, 2)]
>>> 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 FractionWithFactoredDenominatorRing
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()] evaluated at p.

INPUT:

• p – (optional; default: None) a dictionary whose keys are the generators of self.denominator_ring

OUTPUT:
A list.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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)

(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])

sage: f grads(p)
(0, 1), (y, x), (3*x^2, 6*y)

sage: p = {x: sqrt(2), y: var('a')}

(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 [RW2012] 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:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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')
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=(x, y, z,)); (x, y, z,) = R._first_ngens(3)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = (Integer(1) - x*(Integer(1) + y)) * (Integer(1) - z*x**Integer(2)*(Integer(1) + Integer(2)*y))
>>> df = H.factor()
>>> G = Integer(1) / df.unit()
>>> F = FFPD(G, df)
>>> p1 = {x: Integer(1)/Integer(2), y: Integer(1), z: Integer(4)/Integer(3)}
>>> p2 = {x: Integer(1), y: Integer(2), z: Integer(1)/Integer(2)}
>>> F.is_convenient_multiple_point(p1)
(True, 'convenient in variables [x, y]')
>>> F.is_convenient_multiple_point(p2)
(False, 'not a singular point')
```

```python
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 $q_1^{e_1} \cdots q_m^{e_m}$ 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 [Rai2012], $f$ can be written as

$$\sum_A \frac{p_A}{\prod_{i \in A} q_i^{b_i}},$$

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 [Rai2012].
```
OUTPUT:

An instance of FractionWithFactoredDenominatorSum.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
sage: R.<x> = PolynomialRing(QQ)
sage: FFPD = FractionWithFactoredDenominatorRing(R)
sage: f = (x^2 + 1)/((x + 2)*(x - 1)*(x^2 + x + 1))
sage: decom = FFPD(f).leinartas_decomposition()
sage: decom
(0, []) + (2/9, [(x - 1, 1)]) + (-5/9, [(x + 2, 1)]) + (1/3*x, [(x^2 + x + 1, 1)])
sage: decom.sum().quotient() == f
True

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> f = (x^2 + Integer(2) + Integer(1))/((x + Integer(2))^2*(x - Integer(1))^2*(x^2 + x + Integer(1)))
>>> decom = FFPD(f).leinartas_decomposition()
>>> decom
(0, []) + (2/9, [(x - 1, 1)]) + (-5/9, [(x + 2, 1)]) + (1/3*x, [(x^2 + x + 1, 1)])
>>> decom.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: decom = FFPD(f).leinartas_decomposition()
sage: decom
(0, []) + (1, [(x*y + 1, 1)]) + (x + y, [(y, 1), (x, 1)])
sage: decom.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 decom)
True
```

(continues on next page)
...        J = r.algebraic_dependence_certificate()
...        return L is None and (J is None or J == J.ring().ideal())

>>> 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
# random - non canonical depends on singular version
(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

>>> from sage.all import *

>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> f = sin(x)/x + Integer(1)/y + Integer(1)/(x*y + Integer(1))
>>> G = f.numerator()
>>> H = R(f.denominator())
>>> ff = FFPD(G, H.factor())
>>> decomp = ff.leinartas_decomposition()
>>> decomp
# random - non canonical depends on singular version
(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)])
>>> bool(decomp.sum().quotient() == f)
True
>>> 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

>>> from sage.all import *

>>> R = PolynomialRing(GF(Integer(2), a), names=('x', 'y', 'z',)); (x, y, z, ...) = R._first_ngens(3)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> f = Integer(1)/(x * y * z * (x*y + z))

(continues on next page)
Asymptotics Expansions, Release 10.4

```python
>>> decomp = FFPD(f).leinartas_decomposition()
>>> decomp
(0, []) + (1, [(z, 2), (x*y + z, 1)]) +
(1, [(z, 2), (y, 1), (x, 1)])
>>> decomp.sum().quotient() == f
True
```

**log_grads(p)**

Return a list of the logarithmic gradients of the polynomials \([q \text{ for } (q, e) \text{ in } \text{self.denominator_factored()}]\) evaluated 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 \(\text{self.denominator_ring}\)

**OUTPUT:**

A list.

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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)]
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=(x, y,)); (x, y,)= R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)
>>> p = exp(x)
>>> df = [(x**Integer(3) + Integer(3)*y**Integer(2), Integer(5)), (x*y,**Integer(2)), (y, Integer(1))]
>>> f = FFPD(p, df)
>>> f
(e^x, [(y, 1), (x*y, 2), (x^3 + 3*y^2, 5)])
>>> R.gens()
(x, y)
>>> p = None
>>> f.log_grads(p)
```

(continues on next page)

---

4.7. Asymptotics of Multivariate Generating Series 203
maclaurin_coefficients \((\text{multi\_indices, numerical}=0)\)

Return the Maclaurin coefficients of \(\text{self}\) with given \(\text{multi\_indices}\).

**INPUT:**

- \(\text{multi\_indices}\) – a list of tuples of positive integers, where each tuple has length \(\text{self}.\)\(\text{dimension}()\)
- \(\text{numerical}\) – (optional; default: 0) a natural number; if positive, return numerical approximations of
  coefficients with \(\text{numerical}\) digits of accuracy

**OUTPUT:**

A dictionary whose value of the key \(\text{nu}\) are the Maclaurin coefficient of index \(\text{nu}\) of \(\text{self}\).

**Note:** Uses iterated univariate Maclaurin expansions. Slow.

**EXAMPLES:**

```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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}
```
(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)  # long time
{(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
\(h_1q_1 + \cdots + h_mq_n = 1\) 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 FractionWithFactoredDenominatorRing
sage: R.<x,y> = PolynomialRing(QQ)
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 $q_1^{e_1} \cdots q_m^{e_m}$ be the unique factorization of $q$ in $K[X]$ into irreducible factors and let $V_i$ be the algebraic variety $\{x \in L^d | q_i(x) = 0\}$ of $q_i$ over the algebraic closure $L$ of $K$. By [Rai2012], $f$ can be written as

\[
(*) \quad \sum_A \prod_{i \in A} \frac{p_A}{q_i^{e_i}},
\]

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 [Rai2012].

Note: Recursive. Only works for multivariate self.
OUTPUT:

An instance of \texttt{FractionWithFactoredDenominatorSum}.

EXAMPLES:

```python
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]
```

```python
>>> from sage.all import *

>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)

>>> FFPD = FractionWithFactoredDenominatorRing(R)

>>> f = Integer(1)/(x*(x*y + Integer(1)))

>>> decomp = FFPD(f).nullstellensatz_decomposition()

>>> decomp
(0, []) + (1, [(x, 1)]) + (-y, [(x*y + 1, 1)])

>>> decomp.sum().quotient() == f
True

>>> [r.nullstellensatz_certificate() is None for r in decomp]
[True, True, True]
```

```python
sage: R.<x,y> = PolynomialRing(QQ)

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]
```

```python
>>> from sage.all import *

>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)

>>> FFPD = FractionWithFactoredDenominatorRing(R, SR)

>>> G = sin(y)

>>> H = x*(x*y + 1)

>>> f = FFPD(G, H.factor())

>>> decomp = f.nullstellensatz_decomposition()

>>> decomp
(0, []) + (sin(y), [(x, 1)]) + (-y*sin(y), [(x*y + 1, 1)])

>>> bool(decomp.sum().quotient() == G/H)
True

>>> [r.nullstellensatz_certificate() is None for r in decomp]
[True, True, True]
```

(continues on next page)
numerator()

Return the numerator of self.

OUTPUT:

The numerator.

EXAMPLES:

sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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

property numerator_ring

Return the ring of the numerator.

OUTPUT:

A ring.

EXAMPLES:

sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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
from sage.all import *
from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
R = PolynomialRing(QQ, names=('x', 'y'))
(x, y) = R._first_ngens(2)
FFPD = FractionWithFactoredDenominatorRing(R, SR)
H = (Integer(1) - x - y - x*y)**Integer(2)*(Integer(1)-x)
Hfac = H.factor()
G = exp(y)/Hfac.unit()
F = FFPD(G, Hfac)
F.numerator_ring
Symbolic Ring
F = FFPD(G/H)
F
(-e^y, [(x - 1, 1), (x*y + x + y - 1, 2)])
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 FractionWithFactoredDenominatorRing
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.
Asymptotic Expansions, Release 10.4

INPUT:

- \texttt{approx} – an individual or list of symbolic expressions in one variable
- \texttt{alpha} – a list of positive integers of length \texttt{self.denominator\_ring.ngens()}
- \texttt{interval} – a list of positive integers
- \texttt{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 \texttt{interval}. Here \(r*alpha\) is a tuple; \(a_r\) is the \(r*alpha\) (multi-index) coefficient of the Maclaurin series for \texttt{self} divided by \(\texttt{exp\_scale}^r\); \(b_r\) is a list of the values of the functions in \texttt{approx} evaluated at \(r\) and divided by \(\texttt{exp\_scale}^m\); \(err_r\) 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 FractionWithFactoredDenominatorRing
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])]
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y')); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = Integer(1) - x - y - x*y
>>> Hfac = H.factor()
>>> G = Integer(1) / Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> alpha = [Integer(1), Integer(1)]
>>> r = var('r')
>>> a1 = (RealNumber('0.573')/sqrt(r))*RealNumber('5.83')**r
>>> a2 = (RealNumber('0.573')/sqrt(r) - RealNumber('0.0674')/r**(Integer(3)/
... Integer(2)))*RealNumber('5.83')**r
>>> es = RealNumber('5.83')

(continues on next page)
singual_ideal()

Return the singular ideal of self.

Let $R$ be the ring of self and $H$ its denominator. Let $H_{\text{red}}$ be the reduction (square-free part) of $H$. Return the ideal in $R$ generated by $H_{\text{red}}$ 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
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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 [RW2012] 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 FractionWithFactoredDenominatorRing
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
```

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> H = (1-x-y-x*y)**2
>>> Hfac = H.factor()
>>> G = Integer(1)/Hfac.unit()
>>> F = FFPD(G, Hfac)
>>> alpha = [Integer(7)/Integer(3), var('a')]
>>> F.smooth_critical_ideal(alpha)
Ideal (y^2 + 14/(3*a)*y - 1, x + (-3*a)/7*y + (3*a - 7)/7) 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 \( q_1 \cdot \cdots \cdot q_n \) 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 \texttt{partial_fraction()} or \texttt{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 \texttt{FractionWithFactoredDenominatorSum}.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing

R.<x> = PolynomialRing(QQ)
FFPD = FractionWithFactoredDenominatorRing(R)
f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
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^4 - x^3 + 1/3*x^2 - 1/3*x)
```

```python
decomp = FFPD(f).univariate_decomposition()
decomp
(5*x^3, []) + (1, [(x - 1, 1)]) + (1, [(x, 1)]) + (1/3, [(x^2 + 1/3, 1)])
```

```python
decomp.sum().quotient() == f
True
```

```python
R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
FFPD = FractionWithFactoredDenominatorRing(R)
f = Integer(5)*x**Integer(3) + Integer(1)/x + Integer(1)/(x-Integer(1)) +...
```
\( \frac{\text{Integer}(1)}{\text{Integer}(3) \cdot x^2 + \text{Integer}(1)} \)

\[
\begin{align*}
\text{f} & \quad (5x^7 - 5x^6 + 5/3x^5 - 5/3x^4 + 2x^3 - 2/3x^2 + 1/3x - 1/3)/(x^4 - x^3 + 1/3x^2 - 1/3x) \\
\text{decomp} & = \text{FFPD(f).univariate_decomposition()} \\
\text{decomp} & \quad (5x^3, []) + (1, [(x - 1, 1)]) + (1, [(x, 1)]) + (1/3, [(x^2 + 1/3, 1)]) \\
\text{decomp.sum().quotient()} & = f \\
\text{True}
\end{align*}
\]

One variable with numerator in symbolic ring:

\[
\begin{align*}
\text{sage: } & \quad R.<x> = \text{PolynomialRing}(\text{QQ}) \\
\text{sage: } & \quad \text{FFPD} = \text{FractionWithFactoredDenominatorRing}(R, \text{SR}) \\
\text{sage: } & \quad f = 5x^3 + 1/x + 1/(x-1) + \exp(x)/(3x^2 + 1) \\
\text{sage: } & \quad \text{f} \\
\text{sage: } & \quad (5x^5 - 5x^4 + 2x - 1)/(x^2 - x) + e^x/(3x^2 + 1) \\
\text{sage: } & \quad \text{decomp} = \text{FFPD(f).univariate_decomposition()} \\
\text{sage: } & \quad \text{decomp} \\
\text{sage: } & \quad (0, []) + (15/4x^7 - 15/4x^6 + 5/4x^5 - 5/4x^4 + 3/2x^3 + 1/4x^2e^x - 3/4x^2 - 1/4xe^x + 1/2x - 1/4, [(x - 1, 1)]) + (-15x^7 + 15x^6 - 5x^5 + 5x^4 - 6x^3 - x^2e^x + 3x^2 + xe^x - 2x + 1, [(x, 1)]) + (1/4x^2 - 5x^5 - 5x^4 + 6x^3 + x^2e^x - 3x^2 - xe^x + 2x - 1)*(3x^2 - 1), [(x^2 + 1/3, 1)])
\end{align*}
\]

One variable over a finite field:

\[
\begin{align*}
\text{sage: } & \quad R.<x> = \text{PolynomialRing}(\text{GF}(2)) \\
\text{sage: } & \quad \text{FFPD} = \text{FractionWithFactoredDenominatorRing}(R) \\
\text{sage: } & \quad f = 5x^3 + 1/x + 1/(x-1) + 1/(3x^2 + 1) \\
\text{sage: } & \quad \text{f} \\
\text{sage: } & \quad (x^6 + x^4 + 1)/(x^3 + x) \\
\text{sage: } & \quad \text{decomp} = \text{FFPD(f).univariate_decomposition()} \\
\text{sage: } & \quad \text{decomp}
\end{align*}
\]
\[(x^3, []) + (1, [(x, 1)]) + (x, [(x + 1, 2)])\]
\[
sage: \text{decomp}.\text{sum}().\text{quotient}() == f
\]
\[
True
\]

\[
\textbf{4.7. Asymptotics of Multivariate Generating Series}
\]

(continued from previous page)

\[
\textbf{One variable over an inexact field:}
\]
\[
sage: R.<x> = \text{PolynomialRing}(\text{CC})
\]
\[
sage: \text{FFPD} = \text{FractionWithFactoredDenominatorRing}(R)
\]
\[
sage: f = 5*x^3 + 1/x + 1/(x-1) + 1/(3*x^2 + 1)
\]
\[
sage: f
\]
\[
(5.00000000000000*x^7 - 5.00000000000000*x^6 + 1.66666666666667*x^5 - 1.66666666666667*x^4 + 2.00000000000000*x^3 - 0.66666666666667*x^2 + 0.333333333333333*x^2 - 0.333333333333333*x - 0.333333333333333)/(x^4 - x^3 + 0.333333333333333*x^2 - 0.333333333333333*x)
\]
\[
sage: \text{decomp} = \text{FFPD}(f).\text{univariate_decomposition}()
\]
\[
sage: \text{decomp}
\]
\[
(5.00000000000000*x^3, []) + (1.00000000000000, [(x - 1.00000000000000, 1)]) + (-0.288675134594813*I, [(x - 0.577350269189626*I, 1)]) + (1.00000000000000, [(x, 1)]) + (0.288675134594813*I, [(x + 0.577350269189626*I, 1)])
\]
\[
sage: \text{decomp}.\text{sum}().\text{quotient}() == f
\]
\[
# Rounding error coming
False
\]
AUTHORS:

- Daniel Krenn (2014-12-01)

```python
class sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominatorRing
Bases: UniqueRepresentation, Parent

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 FractionWithFactoredDenominatorRing
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)])
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions
    import FractionWithFactoredDenominatorRing
>>> R = PolynomialRing(QQ, names=('x', 'y')); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> df = [x, Integer(1)], [y, Integer(1)], [x*y+Integer(1), Integer(1)]
>>> f = FFPD(x, df)  # indirect doctest
>>> 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:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing
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
```

```python
>>> from sage.all import *

>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing

>>> P = ZZ['X', 'Y']; (X, Y,) = P._first_ngens(2)

>>> F = FractionWithFactoredDenominatorRing(P); F
Ring of fractions with factored denominator over Multivariate Polynomial Ring in X, Y over Integer Ring

>>> F.base_ring()
Integer Ring

>>> F.base()
Multivariate Polynomial Ring in X, Y over Integer Ring
```
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing, FractionWithFactoredDenominatorSum
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

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing, FractionWithFactoredDenominatorSum
>>> R = PolynomialRing(QQ, names=('x', 'y',)); (x, y,) = R._first_ngens(2)
>>> FFPD = FractionWithFactoredDenominatorRing(R)
>>> f = FFPD(x + y, [(y, Integer(1)), (x, Integer(1))])
>>> s = FractionWithFactoredDenominatorSum([f])
>>> s.denominator_ring
Multivariate Polynomial Ring in x, y over Rational Field
>>> g = FFPD(x + y, [])
>>> t = FractionWithFactoredDenominatorSum([g])
>>> 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 FractionWithFactoredDenominatorRing, FractionWithFactoredDenominatorSum
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]).sum()
(2, [(y, 1), (x, 1)]) + (2, [(x*y + 1, 1)])

sage: f = FFPD(cos(x), [(x, 2)])
sage: g = FFPD(cos(y), [(x, 1), (y, 2)])
sage: FractionWithFactoredDenominatorSum([f, g]).sum()
(cos(x), [(x, 2)]) + (cos(y), [(y, 2), (x, 1)])
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: 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)])
Coerce the keys of the dictionary \( p \) into the ring \( R \).

**Warning:** This method assumes that it is possible.

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import FractionWithFactoredDenominatorRing, coerce_point
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, key=str):
    ...: print("{} {} {}").format(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, key=str):
    ...: print("{} {} {}").format(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 $C^{n+1}$ 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 $\{s_1:deriv_1, \ldots, s_r:deriv_r\}$.

Here $s_1, \ldots, s_r$ 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 $\text{str}(w) > \text{str}(v)$, 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 $s_1, \ldots, s_r$ 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:

```python
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)]
diff(f(x), x, x, 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

>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions ...
˓→ import diff_all
>>> f = function('f')(x)
>>> dd = diff_all(f, [x], Integer(3))
>>> dd[(x, x, x)]
diff(f(x), x, x, x)

>>> d1 = {diff(f, x): Integer(4)*x**Integer(3)}
>>> dd = diff_all(f, [x], Integer(3), sub=d1)
>>> dd[(x, x, x)]
24*x

>>> dd = diff_all(f, [x], Integer(3), sub=d1, rekey=f)
>>> dd[diff(f, x, Integer(3))]
24*x

>>> a = {x:Integer(1)}
>>> dd = diff_all(f, [x], Integer(3), sub=d1, rekey=f, sub_final=a)
>>> dd[diff(f, x, Integer(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)]
diff(f(x, y, z), y, y, y, z)

>>> from sage.all import *
>>> X = var('x, y, z')
>>> f = function('f')(*X)
>>> dd = diff_all(f, X, Integer(2), ending=[y, y, y])
>>> dd[(z, y, y, y)]
diff(f(x, y, z), y, y, y, z)

sage: g = function('g')(*X)
sage: dd = diff_all([f, g], X, 2)
sage: dd[[0, y, z]]
diff(f(x, y, z), y, z)
sage: dd[[1, z, z]]
Asymptotic Expansions, Release 10.4

(continued from previous page)

\[
diff(g(x, y, z), z, z)
\]

```python
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)
```

```python
>>> from sage.all import *
>>> g = function('g')(*X)
>>> dd = diff_all([f, g], X, Integer(2))
>>> dd[(Integer(0), y, z)]
diff(f(x, y, z), y, z)
>>> dd[(Integer(1), z, z)]
diff(g(x, y, z), z, z)
>>> f = exp(x*y*z)
>>> ff = function('ff')(*X)
>>> dd = diff_all(f, X, Integer(2), rekey=ff)
>>> dd[diff(ff, x, z)]
x*y^2*z*e^(x*y*z) + y*e^(x*y*z)
```

```python
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 a 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 \( A[0], \ldots, A[r-1] \) up to order \( 2 \times N-2 \) and the (symbolic) derivatives of \( B \) up to order \( 2 \times N \); 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.

4.7. Asymptotics of Multivariate Generating Series 223
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:**

```python
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)  # long time (see :issue:`35207`)

sage: sorted(DD)
# long time
[(0, 0, 0), (0, 1, 0), (0, 1, 1), (0, 1, 2)]

sage: DD[(0, 1, 2)].number_of_operands()  # long time
246
```

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()`.

224 Chapter 4. Asymptotic Expansions — Table of Contents
INPUT:
- \(A, B\) – Symbolic functions in the variable \(x\)
- \(\text{AB}_\text{derivs}\) – a dictionary whose keys are the (symbolic) derivatives of \(A\) up to order \(2 \times N\) if \(v\) is even or \(N\) if \(v\) is odd and the (symbolic) derivatives of \(B\) up to order \(2 \times N + v\) 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 \(l \leq 2k\) and whose values are \(DD^k(ek + vl)(AB^l)\) 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 \(\left(a^{-1/\epsilon}d/dt\right)\) if \(v\) is even and \(\left(|a|^{-1/\epsilon}\text{sgn}(a)d/dt\right)\) if \(v\) is odd.

EXAMPLES:

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_op_simple
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)*diff(A(x), x)), ((1, 1), 1/4*2^(2/3)*(B(x)*diff(A(x), x, x, x, x) + 4*diff(A(x), x, x, x)*diff(B(x), x) + 6*diff(A(x), x)*diff(B(x), x, x, x) + A(x)*diff(B(x), x, x, x, x)))]
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import diff_op_simple
>>> A = function('A')(x)
>>> B = function('B')(x)
>>> AB_derivs = {}
>>> sorted(diff_op_simple(A, B, AB_derivs, x, Integer(3), Integer(2), Integer(2)).items())
[((0, 0), A(x)), ((1, 0), 1/2*I*2^(2/3)*diff(A(x), x)), ((1, 1), 1/4*2^(2/3)*(B(x)*diff(A(x), x, x, x, x) + 4*diff(A(x), x, x, x)*diff(B(x), x) + 6*diff(A(x), x)*diff(B(x), x, x, x) + A(x)*diff(B(x), x, x, x, x)))]
```
Take various derivatives of the equation $f = ug$, evaluate them at a point $c$, and solve for the derivatives of $u$.

**INPUT:**
- $f$ - 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:**
```python
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
```
from sage.all import *
from sage.rings.asymptotic.asymptotics_multivariate_generating_functions

u = function('u')(x)
g = function('g')(x)
fd = {(x,):Integer(1), (x, x):Integer(1)}
ud = {u(x=Integer(2)): Integer(1)}
atc = {x: Integer(2), g(x=Integer(2)): Integer(3), diff(g, x)(x=Integer(2)): Integer(5)}
atc[diff(g, x, x)(x=Integer(2))] = Integer(7)
dd = diff_prod(fd, u, g, [x], [Integer(1), Integer(2)], [], ud, atc)

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 $\text{range}(\text{len}(V))$

OUTPUT:

The tuple $\{V[tt] \text{ for } tt \text{ in } \text{sorted}(t)\}$, 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)

sage: from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions
    import diff_seq
>>> V = list(var('x, t, z'))
>>> diff_seq(V,[[Integer(0), Integer(1)],[Integer(0), Integer(2), Integer(1)],
             [Integer(0), Integer(0)]])
(x, x, x, x, t, t, z)

Note: This function is for internal use by $\text{diff_op}()$.

sage.rings.asymptotic.asymptotics_multivariate_generating_functions.direction(v, coordinate=None)

Return $[vv/v[\text{coordinate}] \text{ for } vv \text{ in } v]$ where coordinate is the last index of $v$ if not specified otherwise.

INPUT:
Asymptotic Expansions, Release 10.4

- \( v \) – 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)
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import direction
>>> direction([Integer(2), Integer(3), Integer(5)])
(2/5, 3/5, 1)
>>> direction([Integer(2), Integer(3), Integer(5)], Integer(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 \( 1, \ldots, \text{len}(u) \) that is induced by the sublist \( s \) of \( u \).

Note: This function was intended for internal use and is deprecated now (Issue #29465).

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 permutation_sign
sage: u = ['a', 'b', 'c', 'd', 'e']
sage: s = ['b', 'd']
sage: permutation_sign(s, u)
doctest:...: DeprecationWarning: the function permutation_sign is deprecated
See https://github.com/sagemath/sage/issues/29465 for details.
-1
sage: s = ['d', 'b']
sage: permutation_sign(s, u)
```

```
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions import permutation_sign
>>> u = ['a', 'b', 'c', 'd', 'e']
>>> s = ['b', 'd']
>>> permutation_sign(s, u)
```

(continues on next page)
doctest:....: DeprecationWarning: the function permutation_sign is deprecated
See https://github.com/sagemath/sage/issues/29465 for details.

```python
>>> s = ['d', 'b']
>>> permutation_sign(s, u)
1
```

```
payment = fin.read('Default/IncomeSeries/UnemploymentDurations.xls')
```

```
sage.rings.asymptotic.asymptotics_multivariate_generating_functions.subs_all(f, sub, simplify=False)
```

Return the items of \( f \) substituted by the dictionaries of \( \text{sub} \) in order of their appearance in \( \text{sub} \).

**INPUT:**
- \( f \) – an individual or list of symbolic expressions or dictionaries
- \( \text{sub} \) – an individual or list of dictionaries
- \( \text{ simplify} \) – (default: \( \text{False} \)) boolean; set to \( \text{True} \) to simplify the result

**OUTPUT:**
The items of \( f \) substituted by the dictionaries of \( \text{sub} \) in order of their appearance in \( \text{sub} \). The \( \text{subs()} \) command is used. If simplify is \( \text{True} \), then \( \text{simplify()} \) is used after substitution.

**EXAMPLES:**

```python
sage: from sage.rings.asymptotic.asymptotics_multivariate_generating_functions...
    import subs_all
sage: var('x, y, z')
(x, y, z)
```

```python
sage: a = {x:1}
sage: b = {y:2}
sage: c = {z:3}
sage: subs_all(x + y + z, a)
y + z + 1
```

```python
sage: subs_all(x + y + z, [c, a])
y + 4
```

```python
sage: subs_all((x + y + z, y^2), b)
x + z + 2, 4
```

```python
sage: subs_all((x + y + z, y^2), [b, c])
x + 5, 4
```

```python
>>> from sage.all import *
>>> from sage.rings.asymptotic.asymptotics_multivariate_generating_functions...
    import subs_all
```

```python
>>> var('x, y, z')
(x, y, z)
```

```python
>>> a = {x:Integer(1)}
>>> b = {y:Integer(2)}
>>> c = {z:Integer(3)}
>>> subs_all(x + y + z, a)
y + z + 1
```

```python
>>> subs_all(x + y + z, [c, a])
y + 4
```

```python
>>> subs_all((x + y + z, y^2), b)
x + z + 2, 4
```

```python
>>> subs_all((x + y + z, y^2), [b, c])
x + 5, 4
```

(continues on next page)
>>> subs_all([x + y + z, y**Integer(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}

>>> from sage.all import *

sage: var('x, y')
(x, y)

sage: a = {'foo': x**Integer(2) + y**Integer(2), 'bar': x - y}
sage: b = {x: Integer(1), y: Integer(2)}
sage: subs_all(a, b)
{'bar': -1, 'foo': 5}
CHAPTER
FIVE

INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.rings.asymptotic.asymptoticExpansionGenerators, 54
sage.rings.asymptotic.asymptoticRing, 7
sage.rings.asymptotic.asymptoticsMultivariateGeneratingFunctions, 173
sage.rings.asymptotic.growthGroup, 67
sage.rings.asymptotic.growthGroupCartesian, 104
sage.rings.asymptotic.misc, 163
sage.rings.asymptotic.termMonoid, 117
INDEX

A

absorb() (sage.rings.asymptotic.term_monoid.GenericTerm method), 135
absorption() (in module sage.rings.asymptotic.term_monoid), 161
AbstractGrowthGroupFunctor (class in sage.rings.asymptotic.growth_group), 72
AdditiveMagmas (sage.rings.asymptotic.growth_group.GenericGrowthGroup attribute), 86
AdditiveMagmas (sage.rings.asymptotic.growth_group.MonomialGrowthGroup attribute), 96
algebraic_dependence_certificate() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 181
algebraic_dependence_decomposition() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 183
asymptotic_decomposition() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 184
asymptotic_expansions (in module sage.rings.asymptotic.asymptotic_expansion_generators), 67
AsymptoticExpansion (class in sage.rings.asymptotic.asymptotic_ring), 15
AsymptoticExpansionGenerators (class in sage.rings.asymptotic.asymptotic_expansion_generators), 55
AsymptoticRing (class in sage.rings.asymptotic.asymptotic_ring), 41
AsymptoticRingFunctor (class in sage.rings.asymptotic.asymptotic_ring), 52
asymptotics() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 186
asymptotics_multiple() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 189
asymptotics_smooth() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 191

B

B() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 18
B() (sage.rings.asymptotic.asymptotic_ring.AsymptoticRing static method), 44;base (sage.rings.asymptotic.growth_group.ExponentialGrowthElement property), 74;base (sage.rings.asymptotic.growth_group.GrowthGroupFactor attribute), 92;base_ring() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominatorRing method), 217
bidirectional_merge_overlapping() (in module sage.rings.asymptotic.misc), 165
bidirectional_merge_sorted() (in module sage.rings.asymptotic.misc), 165
Binomial_kn_over_n() (sage.rings.asymptotic.asymptotic_expansion_generators.AsymptoticExpansionGenerators static method), 55
BTerm (class in sage.rings.asymptotic.term_monoid), 121
BTermMonoid (class in sage.rings.asymptotic.term_monoid), 124

C
can_absorb() (in module sage.rings.asymptotic.term_monoid), 162
can_absorb() (sage.rings.asymptotic.term_monoid.BTerm method), 122
can_absorb() (sage.rings.asymptotic.term_monoid.ExactTerm method), 128
can_absorb() (sage.rings.asymptotic.term_monoid.GenericTerm method), 137
can_absorb() (sage.rings.asymptotic.term_monoid.OTerm method), 150
Index 237

denominator_ring (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator property), 197
diff_all() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 221
diff_op() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 223
diff_op_simple() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 224
diff_prod() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 225
diff_seq() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 227
dimension() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominatorRing), 198
direction() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 227
DivisionRings (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup attribute), 75

e
Element (sage.rings.asymptotic.asymptotic_ring.AsymptoticRing attribute), 44
Element (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominatorRing attribute), 216
Element (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup attribute), 75
Element (sage.rings.asymptotic.growth_group.ExponentialNonGrowthGroup attribute), 78
Element (sage.rings.asymptotic.growth_group.GenericProduct.Element method), 107
Element (sage.rings.asymptotic.growth_group.ExponentialGrowthElement property), 95

E


e

exact_part() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 22
ExactTerm (class in sage.rings.asymptotic.term_monoid), 126
ExactTermMonoid (class in sage.rings.asymptotic.term_monoid), 133
exp() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 23
exp() (sage.rings.asymptotic.growth_group.cartesian.GenericProduct.Element method), 107
exponent (sage.rings.asymptotic.growth_group.MonomialGrowthElement property), 95
ExponentialGrowthElement (class in sage.rings.asymptotic.growth_group), 73
ExponentialGrowthGroup (class in sage.rings.asymptotic.growth_group), 74
ExponentialGrowthGroupFunctor (class in sage.rings.asymptotic.growth_group), 77
ExponentialNonGrowthElement (class in sage.rings.asymptotic.growth_group), 78
ExponentialNonGrowthGroup (class in sage.rings.asymptotic.growth_group), 78
ExponentialNonGrowthGroupFunctor (class in sage.rings.asymptotic.growth_group), 79
extend_by_non_growth_group (sage.rings.asymptotic.growth_group.GenericGrowthGroupFactor attribute), 93
extended_by_non_growth_group() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 86
extract_variable_names() (sage.rings.asymptotic.growth_group.Variable static method), 101

F

factorial() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 24
factors() (sage.rings.asymptotic.growth_group.cartesian.GenericProduct.Element method), 108
factors() (sage.rings.asymptotic.growth_group.GenericGrowthElement method), 79
factory() (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup class method), 75
factory() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 97
FractionWithFactoredDenominator (class in sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 178
FractionWithFactoredDenominatorRing (class in sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 216
FractionWithFactoredDenominatorSum (class in sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 217

Index 237
from_construction() (sage.rings.asymp-
totic.term_monoid.GenericTermMonoid method), 145

G

gen() (sage.rings.asymptotic.asymptotic_ring.Asymptoti-
cRing method), 49
gen() (sage.rings.asymptotic.growth_group.Generic-
GrowthGroup method), 87
GenericGrowthElement (class in sage.rings.asym-
totic.growth_group), 79
GenericGrowthGroup (class in sage.rings.asym-
totic.growth_group), 85
GenericNonGrowthElement (class in sage.rings.asym-
totic.growth_group), 92
GenericNonGrowthGroup (class in sage.rings.asymp-
totic.growth_group), 92
GenericProduct (class in sage.rings.asymptotic.
growth_group_cartesian), 106
GenericProduct.Element (class in sage.rings.asymptotic.
growth_group_cartesian), 107
GenericTerm (class in sage.rings.asymp-
totic.term_monoid), 134
GenericTermMonoid (class in sage.rings.asymp-
totic.term_monoid), 144
gens() (sage.rings.asymptotic.asymptotic_ring.Asymptoti-
cRing method), 49
gen() (sage.rings.asymptotic.growth_group.Exponential-
GrowthGroup method), 76
gen() (sage.rings.asymptotic.growth_group.Generic-
GrowthGroup method), 87
gens_logarithmic() (sage.rings.asymp-
totic.growth_group.MonomialGrowthGroup method), 97
gens_monomial() (sage.rings.asymp-
totic.growth_group_cartesian.GenericProduct method), 114
gens_monomial() (sage.rings.asymp-
totic.growth_group.GenericGrowthGroup method), 88
gens_monomial() (sage.rings.asymp-
totic.growth_group.MonomialGrowthGroup method), 98

H

HarmonicNumber() (sage.rings.asymptotic.asymp-
totic_expansion_generators.AsymptoticExpansionGenerators static method), 56
has_same_summands() (sage.rings.asymptotic.
asymptotic_ring.AsymptoticExpansion method), 25

I

ImplicitExpansion() (sage.rings.asymptotic.asymp-
totic_expansion_generators.AsymptoticExpansionGenerators static method), 57
ImplicitExpansionPeriodicPart() (sage.rings.asymp-
totic.asymptotic_expansion_generators.AsymptoticExpansionGenerators static method), 59
InverseFunctionAnalysis() (sage.rings.asymp-
totic.asymptotic_expansion_generators.AsymptoticExpansionGenerators static method), 60
invert() (sage.rings.asymptotic.asymp-
totic_ring.AsymptoticExpansion method), 26
is_compatible() (sage.rings.asymp-
totic.growth_group.GenericGrowthGroup method), 88
is_compatible() (sage.rings.asymp-
totic.growth_group.PartialConversionElement method), 100
is_constant() (sage.rings.asymp-
totic.term_monoid.ExactTerm method), 128
is_constant() (sage.rings.asymp-
totic.term_monoid.GenericTerm method), 139
is_convenient_multiple_point() (sage.rings.asymp-
totic.asymptotics_multivariate_generating_functions.FractionWithFactored-
Denominator method), 199
is_exact() (sage.rings.asymptotic.asymp-
totic_ring.AsymptoticExpansion method), 27
is_exact() (sage.rings.asymp-
totic.term_monoid.ExactTerm method), 129
is_exact() (sage.rings.asymp-
totic.term_monoid.GenericTerm method), 140
is_little_o_of_one() (sage.rings.asymp-
totic.asymptotic_ring.AsymptoticExpansion method), 27
is_little_o_of_one() (sage.rings.asymp-
totic.term_monoid.ExactTerm method), 130
is_little_o_of_one()  
(sage.rings.asymptotic.term_monoid.GenericTerm method), 140
is_little_o_of_one()  
(sage.rings.asymptotic.term_monoid.OTerm method), 151
is_lt_one()  
(sage.rings.asymptotic.growth_group_cartesian.GenericProduct.Element method), 109
is_lt_one()  
(sage.rings.asymptotic.growth_group.GenericGrowthElement method), 80
is_monomial()  
(sage.rings.asymptotic.growth_group.Variable method), 103
Le()  
(sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 89
Le()  
(sage.rings.asymptotic.term_monoid.GenericTermMonoid method), 147
leinartas_decomposition()  
(sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 200
limit()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 28
Locals  
(class in sage.rings.asymptotic.misc), 163
locals()  
(sage.rings.asymptotic.misc.WithLocals method), 164
log()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 29
log()  
(sage.rings.asymptotic.growth_group_cartesian.GenericProduct.Element method), 109
log()  
(sage.rings.asymptotic.growth_group.GenericGrowthElement method), 80
log_factor()  
(sage.rings.asymptotic.growth_group_cartesian.GenericProduct.Element method), 111
log_factor()  
(sage.rings.asymptotic.growth_group.GenericGrowthElement method), 82
log_grads()  
(sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 203
log_Stirling()  
(sage.rings.asymptotic.asymptotic_expansion_generators.AsymptoticExpansionGenerators static method), 67
log_string()  
(in module sage.rings.asymptotic.misc), 166
log_term()  
(sage.rings.asymptotic.term_monoid.ExactTerm method), 131
log_term()  
(sage.rings.asymptotic.term_monoid.GenericTerm method), 141
log_term()  
(sage.rings.asymptotic.term_monoid.OTerm method), 152
maclaurin_coefficients()  
(sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 204
Magmas  
(sage.rings.asymptotic.growth_group.ExponentialGrowthGroup attribute), 75
Magmas  
(sage.rings.asymptotic.growth_group.GenericGrowthGroup attribute), 86
Magmas  
(sage.rings.asymptotic.growth_group.MonomialGrowthGroup attribute), 96
map_coefficients()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 31
merge()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticRingFunctor method), 53
merge()  
(sage.rings.asymptotic.growth_group.AbstractGrowthGroupFunctor method), 72
module
sage.rings.asymptotic.asymptotic_expansion_generators, 54
sage.rings.asymptotic.asymptotic_ring, 7
sage.rings.asymptotic.asymptotics_multivariate_generating_functions, 173
sage.rings.asymptotic.growth_group, 67
sage.rings.asymptotic.growth_group_cartesian, 104
sage.rings.asymptotic.misc, 163
sage.rings.asymptotic.term_monoid, 117
monomial_coefficient()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 31
MonomialGrowthElement  
(class in sage.rings.asymptotic.growth_group), 95
MonomialGrowthGroup  
(class in sage.rings.asymptotic.growth_group), 96
MonomialGrowthGroupFunctor  
(class in sage.rings.asymptotic.growth_group), 98
MonomialNonGrowthElement  
(class in sage.rings.asymptotic.growth_group), 99
MonomialNonGrowthGroup  
(class in sage.rings.asymptotic.growth_group), 99
MonomialNonGrowthGroupFunctor  
(class in sage.rings.asymptotic.growth_group), 99
MultivariateProduct  
(class in sage.rings.asymptotic.growth_group_cartesian), 116
N
gen()  
(sage.rings.asymptotic.asymptotic_ring.AsymptoticRing method), 50
Asymptotic Expansions, Release 10.4

ngen() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 90
NoConvergenceError, 53
non_growth_group() (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup method), 76
non_growth_group() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 90
non_growth_group() (sage.rings.asymptotic.growth_group.MonomialGrowthGroup method), 98
NotImplementedBZero, 164
NotImplementedOZero, 164
nullstellensatz_certificate() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 205
nullstellensatz_decomposition() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 206
numerator() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 208
numerator_ring (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator property), 208
O
O() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 19
OTerm (class in sage.rings.asymptotic.term_monoid), 149
OTermMonoid (class in sage.rings.asymptotic.term_monoid), 154
P
parent_to_repr_short() (in module sage.rings.asymptotic.misc), 167
PartialConversionElement (class in sage.rings.asymptotic.growth_group), 100
PartialConversionValueError, 100
permutation_sign() (in module sage.rings.asymptotic.asymptotics_multivariate_generating_functions), 228
plot_comparison() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 32
Posets (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup attribute), 75
Posets (sage.rings.asymptotic.growth_group.GenericGrowthGroup attribute), 86
Posets (sage.rings.asymptotic.growth_group.MonomialGrowthGroup attribute), 96
pow() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 33
Q
quotient() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 209
R
rank (sage.rings.asymptotic.asymptotic_ring.AsymptoticRingFunctor attribute), 53
rank (sage.rings.asymptotic.growth_group.AbstractGrowthGroupFunctor attribute), 73
relative_error() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominator method), 209
repr_op() (in module sage.rings.asymptotic.misc), 168
repr_short_to_parent() (in module sage.rings.asymptotic.misc), 168
rpow() (sage.rings.asymptotic.asymptotic_ring.AsymptoticExpansion method), 35
rpow() (sage.rings.asymptotic.growth_group.cartesian.GenericProduct.Element method), 112
rpow() (sage.rings.asymptotic.growth_group.GenericGrowthElement method), 83
rpow() (sage.rings.asymptotic.term_monoid.ExactTerm method), 132
rpow() (sage.rings.asymptotic.term_monoid.GenericTerm method), 142
rpow() (sage.rings.asymptotic.term_monoid.OTerm method), 153
S
sage.rings.asymptotic.asymptotic_expansion_generators module, 54
sage.rings.asymptotic.asymptotic_ring module, 7
sage.rings.asymptotic.asymptotics_multivariate_generating_functions module, 173
sage.rings.asymptotic.growth_group module, 67
sage.rings.asymptotic.growth_group.cartesian module, 104
sage.rings.asymptotic.misc module, 163
sage.rings.asymptotic.term_monoid module, 117
Sets (sage.rings.asymptotic.growth_group.ExponentialGrowthGroup attribute), 75
Sets (sage.rings.asymptotic.growth_group.GenericGrowthGroup attribute), 86
variable_names() (sage.rings.asymptotic.growth_group.GenericGrowthElement method), 85
variable_names() (sage.rings.asymptotic.growth_group.GenericGrowthGroup method), 91
variable_names() (sage.rings.asymptotic.growth_group.Variable method), 103
variable_names() (sage.rings.asymptotic.term_monoid.GenericTerm method), 143

W

whole_and_parts() (sage.rings.asymptotic.asymptotics_multivariate_generating_functions.FractionWithFactoredDenominatorSum method), 219

WithLocals (class in sage.rings.asymptotic.misc), 164

Z

ZeroCoefficientError, 161