Modules over Ore rings

Release 10.7

The Sage Development Team

Aug 14, 2025

CONTENTS

Let R be a commutative ring, $\theta : K \to K$ by a ring endomorphism and $\partial : K \to K$ be a θ-derivation, that is an additive map satisfying the following axiom

$$\partial(xy) = \theta(x)\partial(y) + \partial(x)y$$

The Ore polynomial ring associated to these data is $\mathcal{S} = R[X; \theta, \partial]$; its elements are the usual polynomials over R but the multiplication is twisted according to the rule

$$\partial(xy) = \theta(x)\partial(y) + \partial(x)y$$

We refer to `sage.rings.polynomial.ore_polynomial_ring.OrePolynomial` for more details.

A Ore module over (R, θ, ∂) is by definition a module over $\mathcal{S}$; it is the same than a R-module M equipped with an additive $f : M \to M$ such that

$$f(ax) = \theta(a)f(x) + \partial(a)x$$

Such a map f is called a pseudomorphism (see also `sage.modules.free_module.FreeModule_generic.pseudohom()`).

SageMath provides support for creating and manipulating Ore modules that are finite free over the base ring R. This includes, in particular, Frobenius modules and modules with connexions.

CHAPTER

ONE

MODULES, SUBMODULES AND QUOTIENTS

1.1 Ore modules

Let R be a commutative ring, $\theta : K \to K$ by a ring endomorphism and $\partial : K \to K$ be a θ-derivation, that is an additive map satisfying the following axiom

$$\partial(xy) = \theta(x)\partial(y) + \partial(x)y$$

A Ore module over (R, θ, ∂) is a R-module M equipped with a additive $f : M \to M$ such that

$$f(ax) = \theta(a)f(x) + \partial(a)x$$

Such a map f is called a pseudomorphism.

Equivalently, a Ore module is a module over the (noncommutative) Ore polynomial ring $\mathcal{S} = R[X; \theta, \partial]$.

Defining Ore modules

SageMath provides support for creating and manipulating Ore modules that are finite free over the base ring R.

To start with, the method `sage.rings.polynomial.ore_polynomial_ring.OrePolynomialRing.quotient_module()` creates the quotient $\mathcal{S}/\mathcal{S}P$, endowed with its structure of $\mathcal{S}$-module, that is its structure of Ore module:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: M
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Classical methods are available and we can work with elements in M as we do usually for vectors in finite free modules:

```
sage: M.basis()
[(1, 0), (0, 1)]

sage: v = M((z, z^2)); v
(z, z^2)
sage: z*v
(z^2, 2*z + 2)
```

The Ore action (or equivalently the structure of $\mathcal{S}$-module) is also easily accessible:

```
sage: X*v
(3*z^2 + 2*z, 2*z^2 + 4*z + 4)
```

The method *sage.modules.ore_module.OreModule.pseudohom()* returns the map f defining the action of X:

```
sage: M.pseudohom()
Free module pseudomorphism (twisted by z |--> z^5) defined by the matrix
[  0   1]
[4*z   0]
Domain: Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
Codomain: Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^
↪5
```

A useful feature is the possibility to give chosen names to the vectors of the canonical basis. This is easily done as follows:

```
sage: N.<u,v,w> = S.quotient_module(X^3 + z*X + 1)
sage: N
Ore module <u, v, w> over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: N.basis()
[u, v, w]
```

Alternatively, one can pass in the argument `names`; this could be useful in particular when we want to name the vectors basis $e_0, e_1, \ldots$:

```
sage: A = S.quotient_module(X^11 + z, names='e')
sage: A
Ore module <e0, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10> over Finite Field in z of␣
↪size 5^3 twisted by z |--> z^5
sage: A.basis()
[e0, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10]
```

Do not forget to use the method `inject_variables()` to get the e_i in your namespace:

```
sage: e0
Traceback (most recent call last):
...
NameError: name 'e0' is not defined
sage: A.inject_variables()
Defining e0, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10
sage: e0
e0
```

Submodules and quotients

SageMath provides facilities for creating submodules and quotient modules of Ore modules. First of all, we define the Ore module $\mathcal{S}/\mathcal{S}P^2$ (for some Ore polynomials P), which is obviously not simple:

```
sage: P = X^2 + z*X + 1
sage: U = S.quotient_module(P^2, names='u')
sage: U.inject_variables()
Defining u0, u1, u2, u3
```

We now build the submodule $\mathcal{S}P/\mathcal{S}P^2$ using the method *sage.modules.ore_module.OreModule.span()*:

```
sage: V = U.span(P*u0)
sage: V
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: V.basis()
[u0 + (z^2+2*z+2)*u2 + 4*z*u3,
 u1 + (2*z^2+4*z+4)*u2 + u3]
```

We underline that the span is really the $\mathcal{S}$-span and not the R-span (as otherwise, it will not be a Ore module).

As before, one can use the attributes `names` to give explicit names to the basis vectors:

```
sage: V = U.span(P*u0, names='v')
sage: V
Ore module <v0, v1> over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: V.inject_variables()
Defining v0, v1
sage: v0
v0
sage: U(v0)
u0 + (z^2+2*z+2)*u2 + 4*z*u3
```

A coercion map from V to U is automatically created. Hence, we can safely combine vectors in V and vectors in U in a single expression:

```
sage: v0 - u0
(z^2+2*z+2)*u2 + 4*z*u3
```

We can create the quotient U/V using a similar syntax:

```
sage: W = U.quo(P*u0)
sage: W
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: W.basis()
[u2, u3]
```

We see that SageMath reuses by default the names of the representatives to denote the vectors in the quotient U/V. This behaviour can be overridden by providing explicit names using the attributes `names`.

Shortcuts for creating quotients are also available:

```
sage: U / (P*u0)
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: U/V
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Morphisms of Ore modules

For a tutorial on morphisms of Ore modules, we refer to *sage.modules.ore_module_morphism*.

AUTHOR:

- Xavier Caruso (2024-10)

class sage.modules.ore_module.**OreAction**

Bases: Action

Action by left multiplication of Ore polynomial rings over Ore modules.

class sage.modules.ore_module.**OreModule**(*mat*, *ore*, *names*, *category*)

Bases: UniqueRepresentation, FreeModule_ambient

Generic class for Ore modules.

Element

alias of *OreModuleElement*

basis()

Return the canonical basis of this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 - z)
sage: M.basis()
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

gen(*i*)

Return the i-th vector of the canonical basis of this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 - z)
sage: M.gen(0)
(1, 0, 0)
sage: M.gen(1)
(0, 1, 0)
sage: M.gen(2)
(0, 0, 1)
sage: M.gen(3)
Traceback (most recent call last):
...
IndexError: generator is not defined
```

gens()

Return the canonical basis of this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 - z)
sage: M.gens()
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

hom(*im_gens*, *codomain=None*)

Return the morphism from this Ore module to `codomain` defined by `im_gens`.

INPUT:

- `im_gens` – a datum defining the morphism to build; it could either a list, a tuple, a dictionary or a morphism of Ore modules

- `codomain` (default: `None`) – a Ore module, the codomain of the morphism; if `None`, it is inferred from `im_gens`

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X^3 + 2*t*X^2 + (t^2 + 2)*X + t
sage: Q = t*X^2 - X + 1

sage: U = S.quotient_module(P, names='u')
sage: U.inject_variables()
Defining u0, u1, u2
sage: V = S.quotient_module(P*Q, names='v')
sage: V.inject_variables()
Defining v0, v1, v2, v3, v4
```

The first method for creating a morphism from U to V is to explicitly write down its matrix in the canonical bases:

```
sage: mat = matrix(3, 5, [1, 4, t, 0, 0,
....:                     0, 1, 0, t, 0,
....:                     0, 0, 1, 1, t])
sage: f = U.hom(mat, codomain=V)
sage: f
Ore module morphism:
  From: Ore module <u0, u1, u2> over Fraction Field of Univariate Polynomial␣
↪Ring in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <v0, v1, v2, v3, v4> over Fraction Field of Univariate␣
↪Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
```

This method is however not really convenient because it requires to compute beforehand all the entries of the defining matrix. Instead, we can pass the list of images of the generators:

```
sage: g = U.hom([Q*v0, X*Q*v0, X^2*Q*v0])
sage: g
Ore module morphism:
  From: Ore module <u0, u1, u2> over Fraction Field of Univariate Polynomial␣
↪Ring in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <v0, v1, v2, v3, v4> over Fraction Field of Univariate␣
↪Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
sage: g.matrix()
[1 4 t 0 0]
[0 1 0 t 0]
[0 0 1 1 t]
```

One can even give the values of the morphism on a smaller set as soon as the latter generates the domain as Ore module. The syntax uses dictionaries as follows:

```
sage: h = U.hom({u0: Q*v0})
sage: h
Ore module morphism:
  From: Ore module <u0, u1, u2> over Fraction Field of Univariate Polynomial␣
↪Ring in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <v0, v1, v2, v3, v4> over Fraction Field of Univariate␣
```

(continues on next page)

(continued from previous page)

```
↪Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
sage: g == h
True
```

Finally `im_gens` can also be itself a Ore morphism, in which case SageMath tries to cast it into a morphism with the requested domains and codomains. As an example below, we restrict g to a subspace:

```
sage: C.<c0,c1> = U.span((X + t)*u0)
sage: gC = C.hom(g)
sage: gC
Ore module morphism:
  From: Ore module <c0, c1> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <v0, v1, v2, v3, v4> over Fraction Field of Univariate␣
↪Polynomial Ring in t over Finite Field of size 5 twisted by d/dt

sage: g(c0) == gC(c0)
True
sage: g(c1) == gC(c1)
True
```

identity_morphism()

Return the identity morphism of this Ore module.

EXAMPLES:

```
sage: K.<a> = GF(7^5)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<u,v> = S.quotient_module(X^2 + a*X + a^2)
sage: id = M.identity_morphism()
sage: id
Ore module endomorphism of Ore module <u, v> over Finite Field in a of size 7^
↪5 twisted by a |--> a^7

sage: id(u)
u
sage: id(v)
v
```

is_zero()

Return `True` if this Ore module is reduced to zero.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: M
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: M.is_zero()
False

sage: Q = M.quo(M)
```

(continues on next page)

(continued from previous page)

```
sage: Q.is_zero()
True
```

matrix()

Return the matrix giving the action of the Ore variable on this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 - z^2*X + (z+2)
sage: M = S.quotient_module(P)
sage: M.matrix()
[      0       1       0]
[      0       0       1]
[4*z + 3     z^2     4*z]
```

We recognize the companion matrix attached to the Ore polynomial P. This is of course not a coincidence given that the pseudomorphism corresponds to the left multiplication

See also

pseudohom()

module()

Return the underlying free module of this Ore module.

EXAMPLES:

```
sage: A.<t> = QQ['t']
sage: S.<X> = OrePolynomialRing(A, A.derivation())
sage: M = S.quotient_module(X^3 - t)
sage: M
Ore module of rank 3 over Univariate Polynomial Ring in t over Rational Field␣
↪twisted by d/dt

sage: M.module()
Ambient free module of rank 3 over the principal ideal domain Univariate␣
↪Polynomial Ring in t over Rational Field
```

multiplication_map(P)

Return the multiplication by P acting on this Ore module.

INPUT:

- `P` – a scalar in the base ring, or a Ore polynomial

EXAMPLES:

```
sage: K.<a> = GF(7^5)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + a*X^2 + X - a^2
sage: M = S.quotient_module(P)
```

We define the scalar multiplication by an element in the base ring:

```
sage: f = M.multiplication_map(3)
sage: f
Ore module endomorphism of Ore module of rank 3 over Finite Field in a of␣
↪size 7^5 twisted by a |--> a^7
sage: f.matrix()
[3 0 0]
[0 3 0]
[0 0 3]
```

Be careful that an element in the base ring defines a Ore morphism if and only if it is fixed by the twisting morphisms and killed by the derivation (otherwise the multiplication by this element does not commute with the Ore action). In SageMath, attempting to create the multiplication by an element which does not fulfill these requirements leads to an error:

```
sage: M.multiplication_map(a)
Traceback (most recent call last):
...
ValueError: does not define a morphism of Ore modules
```

As soon as it defines a Ore morphism, one can also build the left multiplication by an Ore polynomial:

```
sage: g = M.multiplication_map(X^5)
sage: g
Ore module endomorphism of Ore module of rank 3 over Finite Field in a of␣
↪size 7^5 twisted by a |--> a^7
sage: g.matrix()
[    3*a^4 + 3*a^3 + 6*a^2 + 5*a       4*a^4 + 5*a^3 + 2*a^2 + 6         6*a^
↪4 + 6*a^3 + a^2 + 4]
[                       a^2 + 3 5*a^4 + 5*a^3 + 6*a^2 + 4*a + 1             ␣
↪   a^3 + 5*a^2 + 4]
[6*a^4 + 6*a^3 + 3*a^2 + 3*a + 1         4*a^4 + 2*a^3 + 3*a + 5 6*a^4 + 6*a^
↪3 + 2*a^2 + 5*a + 2]
```

We check that the characteristic polynomial of g is the reduced norm of the Ore polynomial P we started with (this is a classical property):

```
sage: g.charpoly()
x^3 + 4*x^2 + 2*x + 5
sage: P.reduced_norm(var='x')
x^3 + 4*x^2 + 2*x + 5
```

ore_ring(*names='x'*, *action=True*)

Return the underlying Ore polynomial ring.

INPUT:

- `names` (default: `x`) – a string, the name of the variable
- `action` (default: `True`) – a boolean; if `True`, an action of the Ore polynomial ring on the Ore module is set

EXAMPLES:

```
sage: K.<a> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<e1,e2> = S.quotient_module(X^2 - a)
sage: M.ore_ring()
Ore Polynomial Ring in x over Finite Field in a of size 5^3 twisted by a |-->␣
↪a^5
```

We can use a different variable name:

```
sage: M.ore_ring('Y')
Ore Polynomial Ring in Y over Finite Field in a of size 5^3 twisted by a |-->␣
↪a^5
```

Alternatively, one can use the following shortcut:

```
sage: T.<Z> = M.ore_ring()
sage: T
Ore Polynomial Ring in Z over Finite Field in a of size 5^3 twisted by a |-->␣
↪a^5
```

In all the above cases, an action of the returned Ore polynomial ring on M is registered:

```
sage: Z*e1
e2
sage: Z*e2
a*e1
```

Specifying `action=False` prevents this to happen:

```
sage: T.<U> = M.ore_ring(action=False)
sage: U*e1
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for *:
    'Ore Polynomial Ring in U over Finite Field in a of size 5^3 twisted by a␣
↪|--> a^5' and
    'Ore module <e1, e2> over Finite Field in a of size 5^3 twisted by a |-->␣
↪a^5'
```

pseudohom()

Return the pseudomorphism giving the action of the Ore variable on this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 - z^2*X + (z+2)
sage: M = S.quotient_module(P)
sage: M.pseudohom()
Free module pseudomorphism (twisted by z |--> z^5) defined by the matrix
[      0       1       0]
[      0       0       1]
[4*z + 3     z^2     4*z]
Domain: Ore module of rank 3 over Finite Field in z of size 5^3 twisted by z␣
```

(continues on next page)

(continued from previous page)

```
↪|--> z^5
Codomain: Ore module of rank 3 over Finite Field in z of size 5^3 twisted by␣
↪z |--> z^5
```

See also

matrix()

quo(*sub*, *names=None*, *check=True*)

Return the quotient of this Ore module by the submodule generated (over the underlying Ore ring) by `gens`.

INPUT:

- `gens` – a list of vectors or submodules of this Ore module
- `names` (default: `None`) – the name of the vectors in a basis of the quotient
- `check` (default: `True`) – a boolean, ignored

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X^2 + t*X + 1
sage: M = S.quotient_module(P^3, names='e')
sage: M.inject_variables()
Defining e0, e1, e2, e3, e4, e5
```

We create the quotient M/MP:

```
sage: modP = M.quotient(P*e0)
sage: modP
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
```

As a shortcut, we can write `quo` instead of `quotient` or even use the `/` operator:

```
sage: modP = M / (P*e0)
sage: modP
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
```

By default, the vectors in the quotient have the same names as their representatives in M:

```
sage: modP.basis()
[e4, e5]
```

One can override this behavior by setting the attributes `names`:

```
sage: modP = M.quo(P*e0, names='u')
sage: modP.inject_variables()
Defining u0, u1
sage: modP.basis()
[u0, u1]
```

Note that a coercion map from the initial Ore module to its quotient is automatically set. As a consequence, combining elements of `M` and `modP` in the same formula works:

```
sage: t*u0 + e1
(t^3+4*t)*u0 + (t^2+2)*u1
```

One can combine the construction of quotients and submodules without trouble. For instance, here we build the space MP/MP^2:

```
sage: modP2 = M / (P^2*e0)
sage: N = modP2.span(P*e0)
sage: N
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
sage: N.basis()
[e2 + (2*t^2+2)*e4 + 2*t*e5,
 e3 + 4*t*e4 + 4*e5]
```

See also

quo(), *span()*

quotient (*sub*, *names=None*, *check=True*)

Return the quotient of this Ore module by the submodule generated (over the underlying Ore ring) by `gens`.

INPUT:

- `gens` – a list of vectors or submodules of this Ore module
- `names` (default: `None`) – the name of the vectors in a basis of the quotient
- `check` (default: `True`) – a boolean, ignored

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X^2 + t*X + 1
sage: M = S.quotient_module(P^3, names='e')
sage: M.inject_variables()
Defining e0, e1, e2, e3, e4, e5
```

We create the quotient M/MP:

```
sage: modP = M.quotient(P*e0)
sage: modP
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
```

As a shortcut, we can write `quo` instead of `quotient` or even use the `/` operator:

```
sage: modP = M / (P*e0)
sage: modP
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
```

By default, the vectors in the quotient have the same names as their representatives in M:

```
sage: modP.basis()
[e4, e5]
```

One can override this behavior by setting the attributes `names`:

```
sage: modP = M.quo(P*e0, names='u')
sage: modP.inject_variables()
Defining u0, u1
sage: modP.basis()
[u0, u1]
```

Note that a coercion map from the initial Ore module to its quotient is automatically set. As a consequence, combining elements of `M` and `modP` in the same formula works:

```
sage: t*u0 + e1
(t^3+4*t)*u0 + (t^2+2)*u1
```

One can combine the construction of quotients and submodules without trouble. For instance, here we build the space MP/MP^2:

```
sage: modP2 = M / (P^2*e0)
sage: N = modP2.span(P*e0)
sage: N
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
sage: N.basis()
[e2 + (2*t^2+2)*e4 + 2*t*e5,
 e3 + 4*t*e4 + 4*e5]
```

See also

quo(), *span()*

random_element(**args*, ***kwds*)

Return a random element in this Ore module.

Extra arguments are passed to the random generator of the base ring.

EXAMPLES:

```
sage: A.<t> = QQ['t']
sage: S.<X> = OrePolynomialRing(A, A.derivation())
sage: M = S.quotient_module(X^3 - t, names='e')
sage: M.random_element()    # random
(-1/2*t^2 - 3/4*t + 3/2)*e0 + (-3/2*t^2 - 3*t + 4)*e1 + (-6*t + 2)*e2

sage: M.random_element(degree=5)    # random
(4*t^5 - 1/2*t^4 + 3/2*t^3 + 6*t^2 - t - 1/10)*e0 + (19/3*t^5 - t^3 - t^2 +␣
↪1)*e1 + (t^5 + 4*t^4 + 4*t^2 + 1/3*t - 33)*e2
```

rename_basis(*names*, *coerce=False*)

Return the same Ore module with the given naming for the vectors in its distinguished basis.

INPUT:

- `names` – a string or a list of strings, the new names
- `coerce` (default: `False`) – a boolean; if `True`, a coercion map from this Ore module to renamed version is set

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: M
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5

sage: Me = M.rename_basis('e')
sage: Me
Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Now compare how elements are displayed:

```
sage: M.random_element()   # random
(3*z^2 + 4*z + 2, 3*z^2 + z)
sage: Me.random_element()  # random
(2*z+4)*e0 + (z^2+4*z+4)*e1
```

At this point, there is no coercion map between `M` and `Me`. Therefore, adding elements in both parents results in an error:

```
sage: M.random_element() + Me.random_element()
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for +:
'Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
↪' and
'Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5'
```

In order to set this coercion, one should define `Me` by passing the extra argument `coerce=True`:

```
sage: Me = M.rename_basis('e', coerce=True)
sage: M.random_element() + Me.random_element()  # random
2*z^2*e0 + (z^2+z+4)*e1
```

Warning

Use `coerce=True` with extreme caution. Indeed, setting inappropriate coercion maps may result in a circular path in the coercion graph which, in turn, could eventually break the coercion system.

Note that the bracket construction also works:

```
sage: M.<v,w> = M.rename_basis()
sage: M
Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

In this case, v and w are automatically defined:

```
sage: v + w
v + w
```

span(*gens*, *names=None*)

Return the submodule of this Ore module generated (over the underlying Ore ring) by `gens`.

INPUT:

- `gens` – a list of vectors or submodules of this Ore module
- `names` (default: `None`) – the name of the vectors in a basis of this submodule

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X^2 + t*X + 1
sage: M = S.quotient_module(P^3, names='e')
sage: M.inject_variables()
Defining e0, e1, e2, e3, e4, e5
```

We create the submodule MP:

```
sage: MP = M.span([P*e0])
sage: MP
Ore module of rank 4 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
sage: MP.basis()
[e0 + (t^4+t^2+3)*e4 + t^3*e5,
 e1 + (4*t^3+2*t)*e4 + (4*t^2+3)*e5,
 e2 + (2*t^2+2)*e4 + 2*t*e5,
 e3 + 4*t*e4 + 4*e5]
```

When there is only one generator, encapsulating it in a list is not necessary; one can equally write:

```
sage: MP = M.span(P*e0)
```

If one wants, one can give names to the basis of the submodule using the attribute `names`:

```
sage: MP2 = M.span(P^2*e0, names='u')
sage: MP2.inject_variables()
Defining u0, u1
sage: MP2.basis()
[u0, u1]

sage: M(u0)
e0 + (t^2+4)*e2 + 3*t^3*e3 + (t^2+1)*e4 + 3*t*e5
```

Note that a coercion map from the submodule to the ambient module is automatically set:

```
sage: M.has_coerce_map_from(MP2)
True
```

Therefore, combining elements of `M` and `MP2` in the same expression perfectly works:

```
sage: t*u0 + e1
t*e0 + e1 + (t^3+4*t)*e2 + 3*t^4*e3 + (t^3+t)*e4 + 3*t^2*e5
```

Here is an example with multiple generators:

```
sage: MM = M.span([MP2, P*e1])
sage: MM.basis()
[e0, e1, e2, e3, e4, e5]
```

In this case, we obtain the whole space.

Creating submodules of submodules is also allowed:

```
sage: N = MP.span(P^2*e0)
sage: N
Ore module of rank 2 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
sage: N.basis()
[e0 + (t^2+4)*e2 + 3*t^3*e3 + (t^2+1)*e4 + 3*t*e5,
 e1 + (4*t^2+4)*e3 + 3*t*e4 + 4*e5]
```

See also

quotient()

twisting_derivation()

Return the twisting derivation corresponding to this Ore module.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: T.<Y> = OrePolynomialRing(R, R.derivation())
sage: M = T.quotient_module(Y + t^2)
sage: M.twisting_derivation()
d/dt
```

When the twisting derivation in zero, nothing is returned:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X + z)
sage: M.twisting_derivation()
```

See also

twisting_morphism()

twisting_morphism()

Return the twisting morphism corresponding to this Ore module.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X + z)
sage: M.twisting_morphism()
Frobenius endomorphism z |--> z^5 on Finite Field in z of size 5^3
```

When the twisting morphism is trivial (that is, the identity), nothing is returned:

```
sage: R.<t> = QQ[]
sage: T.<Y> = OrePolynomialRing(R, R.derivation())
sage: M = T.quotient_module(Y + t^2)
sage: M.twisting_morphism()
```

See also

twisting_derivation()

class sage.modules.ore_module.**OreQuotientModule**(*cover*, *basis*, *names*)

Bases: *OreModule*

Class for quotients of Ore modules.

cover()

If this quotient in M/N, return M.

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M.<v,w> = S.quotient_module((X + t)^2)
sage: N = M.quo((X + t)*v)

sage: N.cover()
Ore module <v, w> over Fraction Field of Univariate Polynomial Ring in t over␣
↪Finite Field of size 5 twisted by d/dt
sage: N.cover() is M
True
```

See also

relations()

morphism_modulo(*f*)

If this quotient in M/N and $f : X \to M$ is a morphism, return the induced map $X \to M/N$.

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X + t
sage: M.<v,w> = S.quotient_module(P^2)
sage: Q.<wbar> = M.quo(P*v)
```

(continues on next page)

(continued from previous page)

```
sage: f = M.multiplication_map(X^5)
sage: f
Ore module endomorphism of Ore module <v, w> over Fraction Field of␣
↪Univariate Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
sage: g = Q.morphism_modulo(f)
sage: g
Ore module morphism:
  From: Ore module <v, w> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <wbar> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
```

morphism_quotient(f)

If this quotient in M/N and $f : M \to X$ is a morphism vanishing on N, return the induced map $M/N \to X$.

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: P = X + t
sage: M.<v,w> = S.quotient_module(P^2)
sage: Q.<wbar> = M.quo(P*v)

sage: f = M.hom({v: P*v})
sage: f
Ore module endomorphism of Ore module <v, w> over Fraction Field of␣
↪Univariate Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
sage: g = Q.morphism_quotient(f)
sage: g
Ore module morphism:
  From: Ore module <wbar> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module <v, w> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
```

When the given morphism does not vanish on N, an error is raised:

```
sage: h = M.multiplication_map(X^5)
sage: h
Ore module endomorphism of Ore module <v, w> over Fraction Field of␣
↪Univariate Polynomial Ring in t over Finite Field of size 5 twisted by d/dt
sage: Q.morphism_quotient(h)
Traceback (most recent call last):
...
ValueError: the morphism does not factor through this quotient
```

projection_morphism()

Return the projection from the cover module to this quotient.

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M.<v,w> = S.quotient_module((X + t)^2)
sage: Q = M.quo((X + t)*v)
sage: Q.projection_morphism()
Ore module morphism:
  From: Ore module <v, w> over Fraction Field of Univariate Polynomial Ring␣
↪in t over Finite Field of size 5 twisted by d/dt
  To:   Ore module of rank 1 over Fraction Field of Univariate Polynomial␣
↪Ring in t over Finite Field of size 5 twisted by d/dt
```

relations(*names=None*)

If this quotient in M/N, return N.

INPUT:

- `names` – the names of the vectors of the basis of N, or `None`

EXAMPLES:

```
sage: K.<t> = Frac(GF(5)['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M.<v,w> = S.quotient_module((X + t)^2)
sage: Q = M.quo((X + t)*v)

sage: N = Q.relations()
sage: N
Ore module of rank 1 over Fraction Field of Univariate Polynomial Ring in t␣
↪over Finite Field of size 5 twisted by d/dt
sage: (X + t)*v in N
True
sage: Q == M/N
True
```

It is also possible to define names for the basis elements of N:

```
sage: N.<u> = Q.relations()
sage: N
Ore module <u> over Fraction Field of Univariate Polynomial Ring in t over␣
↪Finite Field of size 5 twisted by d/dt
sage: M(u)
v + 1/t*w
```

See also

relations()

rename_basis(*names*, *coerce=False*)

Return the same Ore module with the given naming for the vectors in its distinguished basis.

INPUT:

- `names` – a string or a list of strings, the new names

- `coerce` (default: `False`) – a boolean; if `True`, a coercion map from this Ore module to the renamed version is set

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z*X + 1)
sage: M
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5

sage: Me = M.rename_basis('e')
sage: Me
Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Now compare how elements are displayed:

```
sage: M.random_element()   # random
(3*z^2 + 4*z + 2, 3*z^2 + z)
sage: Me.random_element()  # random
(2*z + 4)*e0 + (z^2 + 4*z + 4)*e1
```

At this point, there is no coercion map between `M` and `Me`. Therefore, adding elements in both parents results in an error:

```
sage: M.random_element() + Me.random_element()
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for +:
'Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
↪' and
'Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5'
```

In order to set this coercion, one should define `Me` by passing the extra argument `coerce=True`:

```
sage: Me = M.rename_basis('e', coerce=True)
sage: M.random_element() + Me.random_element()  # random
2*z^2*e0 + (z^2 + z + 4)*e1
```

Warning

Use `coerce=True` with extreme caution. Indeed, setting inappropriate coercion maps may result in a circular path in the coercion graph which, in turn, could eventually break the coercion system.

Note that the bracket construction also works:

```
sage: M.<v,w> = M.rename_basis()
sage: M
Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

In this case, v and w are automatically defined:

```
sage: v + w
v + w
```

class sage.modules.ore_module.**OreSubmodule**(*ambient*, *basis*, *names*)

Bases: *OreModule*

Class for submodules of Ore modules.

ambient()

Return the ambient Ore module in which this submodule lives.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<v,w> = S.quotient_module((X + z)^2)
sage: N = M.span((X + z)*v)
sage: N.ambient()
Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: N.ambient() is M
True
```

injection_morphism()

Return the inclusion of this submodule in the ambient space.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<v,w> = S.quotient_module((X + z)^2)
sage: N = M.span((X + z)*v)
sage: N.injection_morphism()
Ore module morphism:
  From: Ore module of rank 1 over Finite Field in z of size 5^3 twisted by z␣
↪|--> z^5
  To:   Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |-->
↪ z^5
```

morphism_corestriction(f)

If the image of f is contained in this submodule, return the corresponding corestriction of f.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X + z
sage: M.<v,w> = S.quotient_module(P^2)
sage: N = M.span(P*v)

sage: f = M.hom({v: P*v})
sage: f
Ore module endomorphism of Ore module <v, w> over Finite Field in z of size 5^
↪3 twisted by z |--> z^5

sage: g = N.morphism_corestriction(f)
sage: g
Ore module morphism:
  From: Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |-->
```

(continues on next page)

(continued from previous page)

```
↪ z^5
  To:   Ore module of rank 1 over Finite Field in z of size 5^3 twisted by z␣
↪|--> z^5
sage: g.matrix()
[    z]
[4*z^2]
```

When the image of the morphism is not contained in this submodule, an error is raised:

```
sage: h = M.multiplication_map(X^3)
sage: N.morphism_corestriction(h)
Traceback (most recent call last):
...
ValueError: the image of the morphism is not contained in this submodule
```

morphism_restriction(*f*)

Return the restriction of f to this submodule.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<v,w> = S.quotient_module((X + z)^2)
sage: N = M.span((X + z)*v)

sage: f = M.multiplication_map(X^3)
sage: f
Ore module endomorphism of Ore module <v, w> over Finite Field in z of size 5^
↪3 twisted by z |--> z^5

sage: g = N.morphism_restriction(f)
sage: g
Ore module morphism:
  From: Ore module of rank 1 over Finite Field in z of size 5^3 twisted by z␣
↪|--> z^5
  To:   Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |-->
↪ z^5
sage: g.matrix()
[        3 4*z^2 + 2]
```

rename_basis(*names*, *coerce=False*)

Return the same Ore module with the given naming for the vectors in its distinguished basis.

INPUT:

- `names` – a string or a list of strings, the new names
- `coerce` (default: `False`) – a boolean; if `True`, a coercion map from this Ore module to renamed version is set

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z^2)
```

(continues on next page)

(continued from previous page)

```
sage: M
Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5

sage: Me = M.rename_basis('e')
sage: Me
Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Now compare how elements are displayed:

```
sage: M.random_element()   # random
(3*z^2 + 4*z + 2, 3*z^2 + z)
sage: Me.random_element()  # random
(2*z + 4)*e0 + (z^2 + 4*z + 4)*e1
```

At this point, there is no coercion map between `M` and `Me`. Therefore, adding elements in both parents results in an error:

```
sage: M.random_element() + Me.random_element()
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for +:
'Ore module of rank 2 over Finite Field in z of size 5^3 twisted by z |--> z^5
↪' and
'Ore module <e0, e1> over Finite Field in z of size 5^3 twisted by z |--> z^5'
```

In order to set this coercion, one should define `Me` by passing the extra argument `coerce=True`:

```
sage: Me = M.rename_basis('e', coerce=True)
sage: M.random_element() + Me.random_element()  # random
2*z^2*e0 + (z^2 + z + 4)*e1
```

Warning

Use `coerce=True` with extreme caution. Indeed, setting inappropriate coercion maps may result in a circular path in the coercion graph which, in turn, could eventually break the coercion system.

Note that the bracket construction also works:

```
sage: M.<v,w> = M.rename_basis()
sage: M
Ore module <v, w> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

In this case, v and w are automatically defined:

```
sage: v + w
v + w
```

class sage.modules.ore_module.**ScalarAction**

Bases: `Action`

Action by scalar multiplication on Ore modules.

sage.modules.ore_module.**normalize_names**(*names*, *rank*)

Return a normalized form of `names`.

INPUT:

- `names` – a string, a list of strings or `None`
- `rank` – the number of names to normalize

EXAMPLES:

```
sage: from sage.modules.ore_module import normalize_names
```

When `names` is a string, indices are added:

```
sage: normalize_names('e', 3)
('e0', 'e1', 'e2')
```

When `names` is a list or a tuple, it remains untouched except that it is always casted to a tuple (in order to be hashable and serve as a key):

```
sage: normalize_names(['u', 'v', 'w'], 3)
('u', 'v', 'w')
```

Similarly, when `names` is `None`, nothing is returned:

```
sage: normalize_names(None, 3)
```

If the number of names is not equal to `rank`, an error is raised:

```
sage: normalize_names(['u', 'v', 'w'], 2)
Traceback (most recent call last):
...
ValueError: the number of given names does not match the rank of the Ore module
```

1.2 Elements in Ore modules

AUTHOR:

- Xavier Caruso (2024-10)

class sage.modules.ore_module_element.**OreModuleElement**

Bases: `FreeModuleElement_generic_dense`

A generic element of a Ore module.

image()

Return the image of this element by the pseudomorphism defining the action of the Ore variable on this Ore module.

EXAMPLES:

```
sage: K.<t> = Frac(QQ['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M.<v,w> = S.quotient_module(X^2 + t)
sage: v.image()
w
```

(continues on next page)

(continued from previous page)

```
sage: w.image()
-t*v
```

is_mutable()

Always return `False` since elements in Ore modules are all immutable.

EXAMPLES:

```
sage: K.<t> = Frac(QQ['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M = S.quotient_module(X^2 + t)

sage: v, w = M.basis()
sage: v
(1, 0)
sage: v.is_mutable()
False
sage: v[1] = 1
Traceback (most recent call last):
...
ValueError: vectors in Ore modules are immutable
```

vector()

Return the coordinates vector of this element.

EXAMPLES:

```
sage: K.<t> = Frac(QQ['t'])
sage: S.<X> = OrePolynomialRing(K, K.derivation())
sage: M.<v,w> = S.quotient_module(X^2 + t)
sage: v.vector()
(1, 0)
```

We underline that this vector is not an element of the Ore module; it lives in K^2. Compare:

```
sage: v.parent()
Ore module <v, w> over Fraction Field of Univariate Polynomial Ring in t over␣
↪Rational Field twisted by d/dt
sage: v.vector().parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in t over Rational Field
```

CHAPTER

TWO

MORPHISMS

2.1 Space of morphisms between Ore modules

AUTHOR:

- Xavier Caruso (2024-10)

class sage.modules.ore_module_homspace.**OreModule_homspace**(*domain*, *codomain*, *category=None*)

Bases: UniqueRepresentation, HomsetWithBase

Class for hom spaces between Ore modules.

Element

alias of *OreModuleMorphism*

identity()

Return the identity morphism in this homspace.

EXAMPLES:

```
sage: K.<z> = GF(7^2)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 + z*X + 1)
sage: End(M).identity()
Ore module endomorphism of Ore module of rank 3 over Finite Field in z of␣
↪size 7^2 twisted by z |--> z^7
```

matrix_space()

Return the matrix space used to represent the morphisms in this homspace.

EXAMPLES:

```
sage: K.<z> = GF(7^2)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 + z*X + 1)
sage: End(M).matrix_space()
Full MatrixSpace of 3 by 3 dense matrices over Finite Field in z of size 7^2
```

```
sage: N = S.quotient_module(X^2 + z)
sage: Hom(M, N).matrix_space()
Full MatrixSpace of 3 by 2 dense matrices over Finite Field in z of size 7^2
```

zero()

Return the zero morphism in this homspace.

EXAMPLES:

```
sage: K.<z> = GF(7^2)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^3 + z*X + 1)
sage: End(M).zero()
Ore module endomorphism of Ore module of rank 3 over Finite Field in z of␣
↪size 7^2 twisted by z |--> z^7
```

2.2 Morphisms between Ore modules

Let R be a commutative ring, $\theta : R \to R$ by a ring endomorphism and $\partial : R \to R$ be a θ-derivation. Let also $\mathcal{S} = R[X; \theta, \partial]$ denote the associated Ore polynomial ring.

By definition, a Ore module is a module over $\mathcal{S}$. In SageMath, there are rather represented as modules over R equipped with the map giving the action of the Ore variable X. We refer to *sage.modules.ore_module* for more details.

A morphism of Ore modules is a R-linear morphism commuting with the Ore action, or equivalenty a $\mathcal{S}$-linear map.

Construction of morphisms

There are several ways for creating Ore modules morphisms in SageMath. First of all, one can use the method *sage.modules.ore_module.OreModule.hom()*, passing to it the matrix (in the canonical bases) of the morphism we want to build:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<e0,e1> = S.quotient_module(X^2 + X + z)

sage: mat = matrix(2, 2, [z,      3*z^2 + z + 2,
....:                     z + 1,        4*z + 4])
sage: f = M.hom(mat)
sage: f
Ore module endomorphism of Ore module <e0, e1> over Finite Field in z of size 5^3␣
↪twisted by z |--> z^5
```

Clearly, this method is not optimal: typing all the entries of the defining matrix is long and a potential source of errors.

Instead, one can use a dictionary encoding the values taken by the morphism on a set of generators; the morphism is then automatically prolonged by $\mathcal{S}$-linearity. Actually here, f was just the multiplication by X^3 on M. We can then redefine it simply as follows:

```
sage: g = M.hom({e0: X^3*e0})
sage: g
Ore module endomorphism of Ore module <e0, e1> over Finite Field in z of size 5^3␣
↪twisted by z |--> z^5
```

One can then recover the matrix by using the method *sage.modules.ore_module_morphism.OreModuleMorphism.matrix()*:

```
sage: g.matrix()
[            z 3*z^2 + z + 2]
[        z + 1       4*z + 4]
```

Alternatively, one can use the method *sage.modules.ore_module.OreModule.multiplication_map()*:

```
sage: h = M.multiplication_map(X^3)
sage: h
Ore module endomorphism of Ore module <e0, e1> over Finite Field in z of size 5^3␣
↪twisted by z |--> z^5
sage: g == h
True
```

Of course, this method also accepts values in the base ring:

```
sage: h = M.multiplication_map(2)
sage: h
Ore module endomorphism of Ore module <e0, e1> over Finite Field in z of size 5^3␣
↪twisted by z |--> z^5
sage: h.matrix()
[2 0]
[0 2]
```

Be careful that scalar multiplications do not always properly define a morphism of Ore modules:

```
sage: M.multiplication_map(z)
Traceback (most recent call last):
...
ValueError: does not define a morphism of Ore modules
```

SageMath provides methods to compute kernels, cokernels, images and coimages. In order to illustrate this, we will build the sequence

$$0 \to \mathcal{S}/\mathcal{S}P \to \mathcal{S}/\mathcal{S}PQ \to \mathcal{S}/\mathcal{S}Q \to 0$$

and check that it is exact. We first build the Ore modules:

```
sage: P = X^2 + z*X + 1
sage: Q = X^3 + z^2*X^2 + X + z
sage: U = S.quotient_module(P, names='u')
sage: U.inject_variables()
Defining u0, u1
sage: V = S.quotient_module(P*Q, names='v')
sage: V.inject_variables()
Defining v0, v1, v2, v3, v4
sage: W = S.quotient_module(Q, names='w')
sage: W.inject_variables()
Defining w0, w1, w2
```

Next, we build the morphisms:

```
sage: f = U.hom({u0: Q*v0})
sage: g = V.hom({v0: w0})
```

We can now check that f is injective by computing its kernel:

```
sage: f.kernel()
Ore module of rank 0 over Finite Field in z of size 5^3 twisted by z |--> z^5
```

We see on the output that it has dimension 0; so it vanishes. Instead of reading the output, one can check programmatically the vanishing of a Ore module using the method *sage.modules.ore_module.OreModule.is_zero()*:

```
sage: f.kernel().is_zero()
True
```

Actually, in our use case, one can, more simply, use the method *sage.modules.ore_module_morphism.OreModuleMorphism.is_injective()*:

```
sage: f.is_injective()
True
```

Similarly, one checks that g is surjective:

```
sage: g.is_surjective()
True
```

or equivalently:

```
sage: g.cokernel().is_zero()
True
```

Now, we need to check that the kernel of g equals the image of f. For this, we compute both and compare the results:

```
sage: ker = g.kernel()
sage: im = f.image()
sage: ker == im
True
```

As a sanity check, one can also verity that the composite $g \circ f$ vanishes:

```
sage: h = g * f
sage: h
Ore module morphism:
  From: Ore module <u0, u1> over Finite Field in z of size 5^3 twisted by z |--> z^5
  To:   Ore module <w0, w1, w2> over Finite Field in z of size 5^3 twisted by z |-->␣
↪z^5
sage: h.is_zero()
True
```

Let us now consider another morphism f and build the canonical isomorphism coim $f \to$ im f that it induces. We start by defining f:

```
sage: A = X + z
sage: B = X + z + 1
sage: P = X^2 + X + z
sage: U = S.quotient_module(B*P, names='u')
sage: U.inject_variables()
Defining u0, u1, u2
sage: V = S.quotient_module(P*A, names='v')
sage: V.inject_variables()
```

(continues on next page)

(continued from previous page)

```
Defining v0, v1, v2

sage: f = U.hom({u0: A*v0})
sage: f
Ore module morphism:
  From: Ore module <u0, u1, u2> over Finite Field in z of size 5^3 twisted by z |-->␣
↪z^5
  To:   Ore module <v0, v1, v2> over Finite Field in z of size 5^3 twisted by z |-->␣
↪z^5
```

Now we compute the image and the coimage:

```
sage: I = f.image(names='im')
sage: I
Ore module <im0, im1> over Finite Field in z of size 5^3 twisted by z |--> z^5

sage: C = f.coimage(names='co')
sage: C
Ore module <co0, co1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

We can already check that the image and the coimage have the same rank. We now want to construct the isomorphism between them. For this, we first need to corestrict f to its image. This is achieved via the method *sage.modules.ore_module.OreSubmodule.morphism_corestriction()* of the Ore module:

```
sage: g = I.morphism_corestriction(f)
sage: g
Ore module morphism:
  From: Ore module <u0, u1, u2> over Finite Field in z of size 5^3 twisted by z |-->␣
↪z^5
  To:   Ore module <im0, im1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

Next, we want to factor g by the coimage. We proceed as follows:

```
sage: h = C.morphism_quotient(g)
sage: h
Ore module morphism:
  From: Ore module <co0, co1> over Finite Field in z of size 5^3 twisted by z |--> z^5
  To:   Ore module <im0, im1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

We have found the morphism we were looking for: it is h. We can now check that it is an isomorphism:

```
sage: h.is_isomorphism()
True
```

As a shortcut, we can use explicit conversions as follows:

```
sage: H = Hom(C, I)  # the hom space
sage: h2 = H(f)
sage: h2
Ore module morphism:
  From: Ore module <co0, co1> over Finite Field in z of size 5^3 twisted by z |--> z^5
  To:   Ore module <im0, im1> over Finite Field in z of size 5^3 twisted by z |--> z^5
```

(continues on next page)

(continued from previous page)

```
sage: h == h2
True
```

For endomorphisms, one can compute classical invariants as determinants and characteristic polynomials. To illustrate this, we check on an example that the characteristic polynomial of the multiplication by X^3 on the quotient $\mathcal{S}/\mathcal{S}P$ is the reduced norm of P:

```
sage: P = X^5 + z*X^4 + z^2*X^2 + z + 1
sage: M = S.quotient_module(P)
sage: f = M.multiplication_map(X^3)
sage: f.charpoly()
x^5 + x^4 + x^3 + x^2 + 1

sage: P.reduced_norm('x')
x^5 + x^4 + x^3 + x^2 + 1
```

AUTHOR:

- Xavier Caruso (2024-10)

class sage.modules.ore_module_morphism.**OreModuleMorphism**(*parent*, *im_gens*, *check=True*)

Bases: Morphism

Generic class for morphism between Ore modules.

characteristic_polynomial(*var='x'*)

Return the determinant of this endomorphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + (z^2 + 3)*X + z^5
sage: M = S.quotient_module(P)
sage: f = M.multiplication_map(X^3)
sage: f.characteristic_polynomial()
x^3 + x^2 + 2*x + 2
```

We check that the latter is equal to the reduced norm of P:

```
sage: P.reduced_norm('x')
x^3 + x^2 + 2*x + 2
```

charpoly(*var='x'*)

Return the determinant of this endomorphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + (z^2 + 3)*X + z^5
sage: M = S.quotient_module(P)
sage: f = M.multiplication_map(X^3)
sage: f.characteristic_polynomial()
x^3 + x^2 + 2*x + 2
```

We check that the latter is equal to the reduced norm of P:

```
sage: P.reduced_norm('x')
x^3 + x^2 + 2*x + 2
```

coimage(*names=None*)

Return `True` if this morphism is injective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = M.hom({m0: n0})
sage: coim = f.coimage()
sage: coim
Ore module of rank 3 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: coim.basis()
[m3, m4, m5]
```

cokernel(*names=None*)

Return `True` if this morphism is injective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = N.hom({n0: P*m0})
sage: coker = f.cokernel()
sage: coker
Ore module of rank 3 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: coker.basis()
[m3, m4, m5]
```

det()

Return the determinant of this endomorphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<m0,m1> = S.quotient_module(X^2 + z)
sage: f = M.multiplication_map(X^3)
sage: f.determinant()
2
```

If the domain differs from the codomain (even if they have the same rank), an error is raised:

```
sage: N.<n0,n1> = S.quotient_module(X^2 + z^25)
sage: g = M.hom({z*m0: n0})
sage: g.determinant()
Traceback (most recent call last):
...
ValueError: determinants are only defined for endomorphisms
```

determinant()

Return the determinant of this endomorphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<m0,m1> = S.quotient_module(X^2 + z)
sage: f = M.multiplication_map(X^3)
sage: f.determinant()
2
```

If the domain differs from the codomain (even if they have the same rank), an error is raised:

```
sage: N.<n0,n1> = S.quotient_module(X^2 + z^25)
sage: g = M.hom({z*m0: n0})
sage: g.determinant()
Traceback (most recent call last):
...
ValueError: determinants are only defined for endomorphisms
```

image(*names=None*)

Return `True` if this morphism is injective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = N.hom({n0: P*m0})
sage: im = f.image()
```

(continues on next page)

(continued from previous page)

```
sage: im
Ore module of rank 3 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: im.basis()
[m0 + (2*z^2+3*z+1)*m3 + (4*z^2+3*z+3)*m4 + (2*z^2+3*z)*m5,
 m1 + (z+3)*m3 + (z^2+z+4)*m4,
 m2 + (2*z^2+4*z+2)*m4 + (2*z^2+z+1)*m5]
```

inverse()

Return the inverse of this morphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^2 + z
sage: M.<e0,e1,e2,e3> = S.quotient_module(P^2)

sage: f = M.multiplication_map(X^3)
sage: g = f.inverse()
sage: (f*g).is_identity()
True
sage: (g*f).is_identity()
True
```

If the morphism is not invertible, an error is raised:

```
sage: h = M.hom({e0: P*e0})
sage: h.inverse()
Traceback (most recent call last):
...
ValueError: this morphism is not invertible
```

is_bijective()

Return `True` if this morphism is bijective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: f = M.multiplication_map(X^3)
sage: f.is_bijective()
True

sage: N = S.quotient_module(X^2)
sage: g = N.multiplication_map(X^3)
sage: g.is_bijective()
False
```

is_identity()

Return `True` if this morphism is the identity.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M.<v,w> = S.quotient_module(X^2 + z)
sage: f = M.hom({v: v})
sage: f.is_identity()
True

sage: f = M.hom({v: 2*v})
sage: f.is_identity()
False
```

is_injective()

Return `True` if this morphism is injective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = N.hom({n0: P*m0})
sage: f.is_injective()
True

sage: g = M.hom({m0: n0})
sage: g.is_injective()
False
```

is_isomorphism()

Return `True` if this morphism is an isomorphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: f = M.multiplication_map(X^3)
sage: f.is_isomorphism()
True

sage: N = S.quotient_module(X^2)
sage: g = N.multiplication_map(X^3)
sage: g.is_isomorphism()
False
```

is_surjective()

Return `True` if this morphism is surjective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = N.hom({n0: P*m0})
sage: f.is_surjective()
False

sage: g = M.hom({m0: n0})
sage: g.is_surjective()
True
```

is_zero()

Return `True` if this morphism is zero.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2 + 1
sage: M = S.quotient_module(P^2, names='e')
sage: M.inject_variables()
Defining e0, e1, e2, e3, e4, e5

sage: f = M.hom({e0: P*e0})
sage: f.is_zero()
False
sage: (f*f).is_zero()
True
```

kernel(*names=None*)

Return `True` if this morphism is injective.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: P = X^3 + z*X^2 + z^2
sage: M = S.quotient_module(P^2, names='m')
sage: M.inject_variables()
Defining m0, m1, m2, m3, m4, m5
sage: N = S.quotient_module(P, names='n')
sage: N.inject_variables()
Defining n0, n1, n2

sage: f = M.hom({m0: n0})
sage: ker = f.kernel()
sage: ker
```

(continues on next page)

(continued from previous page)

```
Ore module of rank 3 over Finite Field in z of size 5^3 twisted by z |--> z^5
sage: ker.basis()
[m0 + (2*z^2+3*z+1)*m3 + (4*z^2+3*z+3)*m4 + (2*z^2+3*z)*m5,
 m1 + (z+3)*m3 + (z^2+z+4)*m4,
 m2 + (2*z^2+4*z+2)*m4 + (2*z^2+z+1)*m5]
```

matrix()

Return the matrix defining this morphism.

EXAMPLES:

```
sage: K.<z> = GF(5^3)
sage: S.<X> = OrePolynomialRing(K, K.frobenius_endomorphism())
sage: M = S.quotient_module(X^2 + z)
sage: f = M.multiplication_map(2)
sage: f.matrix()
[2 0]
[0 2]

sage: g = M.multiplication_map(X^3)
sage: g.matrix()
[            0 3*z^2 + z + 1]
[      2*z + 1             0]
```

class sage.modules.ore_module_morphism.**OreModuleRetraction**

Bases: Map

Conversion (partially defined) map from an ambient module to one of its submodule.

class sage.modules.ore_module_morphism.**OreModuleSection**

Bases: Map

Section map of the projection onto a quotient. It is not necessarily compatible with the Ore action.

PYTHON MODULE INDEX

m

INDEX

N

O

P

Q

R

S

T

V

Z